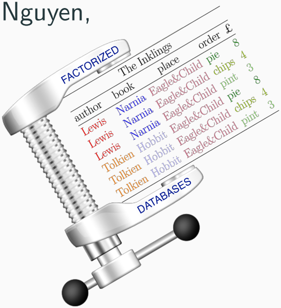
In-Database Factorised Learning

fdbresearch.github.io

Mahmoud Abo Khamis, Hung Ngo, XuanLong Nguyen,
Dan Olteanu, and Maximilian Schleich

December 2017

Logic for Data Science Seminar
Alan Turing Institute



Talk Outline

Current Landscape for ML over DB

Factorised Learning over Normalised Data

Learning under Functional Dependencies

General Problem Formulation

Brief Outlook at Current Landscape for ML over DB (1/2)

No integration

- ML & DB distinct tools on the technology stack
- DB exports data as one table, ML imports it in own format
- Spark/PostgreSQL + R supports virtually any ML task
- Most ML over DB solutions operate in this space

Brief Outlook at Current Landscape for ML over DB (1/2)

No integration

- ML & DB distinct tools on the technology stack
- DB exports data as one table, ML imports it in own format
- Spark/PostgreSQL + R supports virtually any ML task
- Most ML over DB solutions operate in this space

Loose integration

- Each ML task implemented by a distinct UDF inside DB
- Same running process for DB and ML
- DB computes one table, ML works directly on it
- MadLib supports comprehensive library of ML UDFs

Brief Outlook at Current Landscape for ML over DB (2/2)

Unified programming architecture

- One framework for many ML tasks instead of one UDF per task, with possible code reuse across UDFs
- DB computes one table, ML works directly on it
- Bismark supports incremental gradient descent for convex programming; up to 100% overhead over specialized UDFs

Brief Outlook at Current Landscape for ML over DB (2/2)

Unified programming architecture

- One framework for many ML tasks instead of one UDF per task, with possible code reuse across UDFs
- DB computes one table, ML works directly on it
- Bismark supports incremental gradient descent for convex programming; up to 100% overhead over specialized UDFs

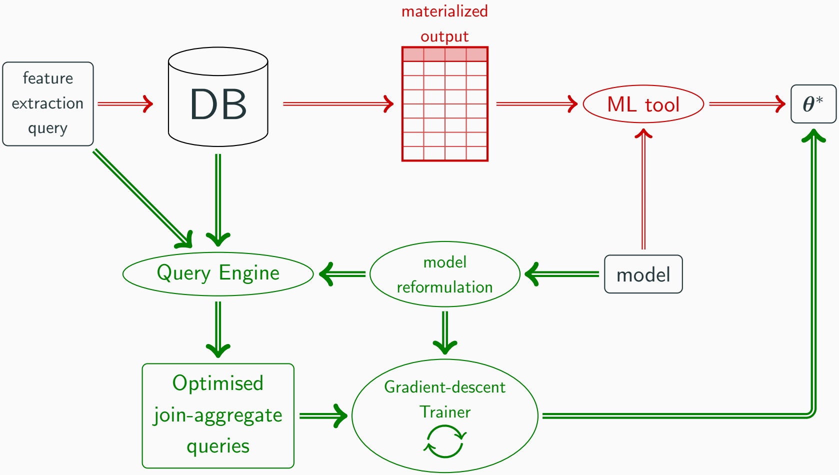
Tight integration ⇒ In-Database Analytics

- One evaluation plan for both DB and ML workload; opportunity to push parts of ML tasks past DB joins
- Morpheus + Hamlet supports GLM and naïve Bayes
- Our approach supports PR/FM, decision trees, ...

In-Database Analytics

- Move the analytics, not the data
  - Avoid expensive data export/import
  - Exploit database technologies
  - Exploit the relational structure (schema, query, dependencies)
  - Build better models using larger datasets and faster
- Cast analytics code as join-aggregate queries
  - Many similar queries that massively share computation
  - Fixpoint computation needed for model convergence

In-database vs. Out-of-database Analytics



Does It Pay Off in Practice?

Retailer dataset (records)		excerpt (17M)	full (86M)
Linear regression			
Features	(cont+categ)	33 + 55	33+3,653
Aggregates	(cont+categ)	595+2,418	595+145k
MadLib	Learn	1,898.35 sec	$> 24h$
R	Join (PSQL)	50.63 sec	–
	Export/Import	308.83 sec	–
	Learn	490.13 sec	–
Our approach	Join-Aggregate	25.51 sec	380.31 sec
(1core, commodity machine)	Converge (runs)	0.02 (343) sec	8.82 (366) sec
Polynomial regression degree 2			
Features	(cont+categ)	562+2,363	562+141k
Aggregates	(cont+categ)	158k+742k	158k+37M
MadLib	Learn	$> 24h$	–
Our approach	Join-Aggregate	132.43 sec	1,819.80 sec
(1core, commodity machine)	Converge (runs)	3.27 (321) sec	219.51 (180) sec

Talk Outline

Current Landscape for ML over DB

Factorised Learning over Normalised Data

Learning under Functional Dependencies

General Problem Formulation

Unified In-Database Analytics for Optimisation Problems

Our target: retail-planning and forecasting applications

- Typical databases: weekly sales, promotions, and products
- Training dataset: Result of a feature extraction query
- Task: Train model to predict additional demand generated for a product due to promotion
- Training algorithm: batch gradient descent
- ML tasks: ridge linear regression, polynomial regression, factorisation machines; logistic regression, SVM; PCA.

Typical Retail Example

- Database $I = (R_1, R_2, R_3, R_4, R_5)$
- Feature selection query Q:

$$Q(\texttt{sku}, \texttt{store}, \texttt{color}, \texttt{city}, \texttt{country}, \mathit{unitsSold}) \leftarrow$$
$$R_1(\texttt{sku}, \texttt{store}, \texttt{day}, \mathit{unitsSold}), R_2(\texttt{sku}, \texttt{color}),$$
$$R_3(\texttt{day}, \texttt{quarter}), R_4(\text{store}, \text{city}), R_5(\texttt{city}, \texttt{country}).$$

- Free variables
  - Categorical (qualitative): $F = \{\texttt{sku}, \texttt{store}, \texttt{color}, \texttt{city}, \texttt{country}\}$.
  - Continuous (quantitative): *unitsSold*.
- Bound variables
  - Categorical (qualitative): $B = \{\texttt{day}, \texttt{quarter}\}$

Typical Retail Example

- We learn the ridge linear regression model

$$\langle \boldsymbol{\theta}, \mathbf{x} \rangle = \sum_{f \in F} \langle \boldsymbol{\theta}_f, \mathbf{x}_f \rangle$$

  - Training dataset: $D = Q(I)$
  - Feature vector $\mathbf{x}$ and response $y = \mathit{unitsSold}$

- The parameters $\boldsymbol{\theta}$ obtained by minimising the objective function:

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

Side Note: One-hot Encoding of Categorical Variables

- Continuous variables are mapped to scalars
  - $y_{unitsSold} \in \mathbb{R}$.
- Categorical variables are mapped to indicator vectors
  - `country` has categories `vietnam` and `england`
  - `country` is then mapped to an indicator vector $\mathbf{x}_{\texttt{country}} = [x_{\texttt{vietnam}}, x_{\texttt{england}}]^\top \in (\{0,1\}^2)^\top$.
  - $\mathbf{x}_{\texttt{country}} = [0,1]^\top$ for a tuple with `country = ‘‘england’’`

This encoding leads to wide training datasets and many 0s

From Optimisation to Sum-Product Queries

We can solve $\boldsymbol{\theta}^* := \arg\min_{\boldsymbol{\theta}} J(\boldsymbol{\theta})$ by repeatedly updating $\boldsymbol{\theta}$ in the direction of the gradient until convergence:

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

From Optimisation to Sum-Product Queries

We can solve $\boldsymbol{\theta}^* := \arg\min_{\boldsymbol{\theta}} J(\boldsymbol{\theta})$ by repeatedly updating $\boldsymbol{\theta}$ in the direction of the gradient until convergence:

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

Define the matrix $\boldsymbol{\Sigma} = (\boldsymbol{\sigma}_{ij})_{i,j \in [|F|]}$, the vector $\mathbf{c} = (c_i)_{i \in [|F|]}$, and the scalar s_Y:

$$\boldsymbol{\sigma}_{ij} = \frac{1}{|D|} \sum_{(\mathbf{x},y) \in D} \mathbf{x}_i \mathbf{x}_j^\top \qquad \mathbf{c}_i = \frac{1}{|D|} \sum_{(\mathbf{x},y) \in D} y \cdot \mathbf{x}_i \qquad s_Y = \frac{1}{|D|} \sum_{(\mathbf{x},y) \in D} y^2.$$

From Optimisation to Sum-Product Queries

We can solve $\boldsymbol{\theta}^* := \arg\min_{\boldsymbol{\theta}} J(\boldsymbol{\theta})$ by repeatedly updating $\boldsymbol{\theta}$ in the direction of the gradient until convergence:

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

Define the matrix $\boldsymbol{\Sigma} = (\boldsymbol{\sigma}_{ij})_{i,j\in[|F|]}$, the vector $\mathbf{c} = (c_i)_{i\in[|F|]}$, and the scalar s_Y:

$$\boldsymbol{\sigma}_{ij} = \frac{1}{|D|}\sum_{(\mathbf{x},y)\in D} \mathbf{x}_i\mathbf{x}_j^\top \qquad \mathbf{c}_i = \frac{1}{|D|}\sum_{(\mathbf{x},y)\in D} y\cdot\mathbf{x}_i \qquad s_Y = \frac{1}{|D|}\sum_{(\mathbf{x},y)\in D} y^2.$$

Then,

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

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

Σ, c, s_Y can be Expressed as SQL Queries

SQL queries for $\boldsymbol{\sigma}_{ij} = \frac{1}{|D|}\sum_{(\mathbf{x},y)\in D} \mathbf{x}_i\mathbf{x}_j^\top$ (w/o factor $\frac{1}{|D|}$):

Σ, **c**, s_Y can be Expressed as SQL Queries

SQL queries for $\boldsymbol{\sigma}_{ij} = \frac{1}{|D|} \sum_{(\mathbf{x},y) \in D} \mathbf{x}_i \mathbf{x}_j^\top$ (w/o factor $\frac{1}{|D|}$):

- x_i, x_j continuous $\Rightarrow$ no group-by variable

  SELECT SUM (x_i * x_j) FROM D;

where D is the natural join of tables R_1 to R_5 in our example.

Σ, c, s_Y can be Expressed as SQL Queries

SQL queries for $\boldsymbol{\sigma}_{ij} = \frac{1}{|D|} \sum_{(\mathbf{x},y) \in D} \mathbf{x}_i \mathbf{x}_j^\top$ (w/o factor $\frac{1}{|D|}$):

- x_i, x_j continuous $\Rightarrow$ no group-by variable

  ```
  SELECT SUM (x_i * x_j) FROM D;
  ```

- $\boldsymbol{x}_i$ categorical, x_j continuous $\Rightarrow$ one group-by variable

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

where D is the natural join of tables R_1 to R_5 in our example.

Σ, $\mathbf{c}$, s_Y can be Expressed as SQL Queries

SQL queries for $\boldsymbol{\sigma}_{ij} = \frac{1}{|D|}\sum_{(\mathbf{x},y)\in D} \mathbf{x}_i \mathbf{x}_j^\top$ (w/o factor $\frac{1}{|D|}$):

- x_i, x_j continuous $\Rightarrow$ no group-by variable

  `SELECT SUM (`x_i `*` x_j`) FROM` D`;`

- $\boldsymbol{x}_i$ categorical, x_j continuous $\Rightarrow$ one group-by variable

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

- $\boldsymbol{x}_i$, $\boldsymbol{x}_j$ categorical $\Rightarrow$ two group-by variables

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

where D is the natural join of tables R_1 to R_5 in our example.

Σ, c, s_Y can be Expressed as SQL Queries

SQL queries for $\boldsymbol{\sigma}_{ij} = \frac{1}{|D|} \sum_{(\mathbf{x},y) \in D} \mathbf{x}_i \mathbf{x}_j^\top$ (w/o factor $\frac{1}{|D|}$):

- x_i, x_j continuous $\Rightarrow$ no group-by variable

  `SELECT SUM (`x_i` * `x_j`) FROM `D`;`

- $\boldsymbol{x}_i$ categorical, x_j continuous $\Rightarrow$ one group-by variable

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

- $\boldsymbol{x}_i$, $\boldsymbol{x}_j$ categorical $\Rightarrow$ two group-by variables

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

where D is the natural join of tables R_1 to R_5 in our example.

This query encoding avoids drawbacks of one-hot encoding

How To Compute Efficiently These Join-Aggregate Queries?

Factorised Query Computation by Example

Orders (O for short)

customer	day	dish
Elise	Monday	burger
Elise	Friday	burger
Steve	Friday	hotdog
Joe	Friday	hotdog

Dish (D for short)

dish	item
burger	patty
burger	onion
burger	bun
hotdog	bun
hotdog	onion
hotdog	sausage

Items (I for short)

item	price
patty	6
onion	2
bun	2
sausage	4

Factorised Query Computation by Example

Orders (O for short)

customer	day	dish
Elise	Monday	burger
Elise	Friday	burger
Steve	Friday	hotdog
Joe	Friday	hotdog

Dish (D for short)

dish	item
burger	patty
burger	onion
burger	bun
hotdog	bun
hotdog	onion
hotdog	sausage

Items (I for short)

item	price
patty	6
onion	2
bun	2
sausage	4

Consider the natural join of the above relations:

O(customer, day, dish), D(dish, item), I(item, price)

customer	day	dish	item	price
Elise	Monday	burger	patty	6
Elise	Monday	burger	onion	2
Elise	Monday	burger	bun	2
Elise	Friday	burger	patty	6
Elise	Friday	burger	onion	2
Elise	Friday	burger	bun	2
...	...	...	...	...

Factorised Query Computation by Example

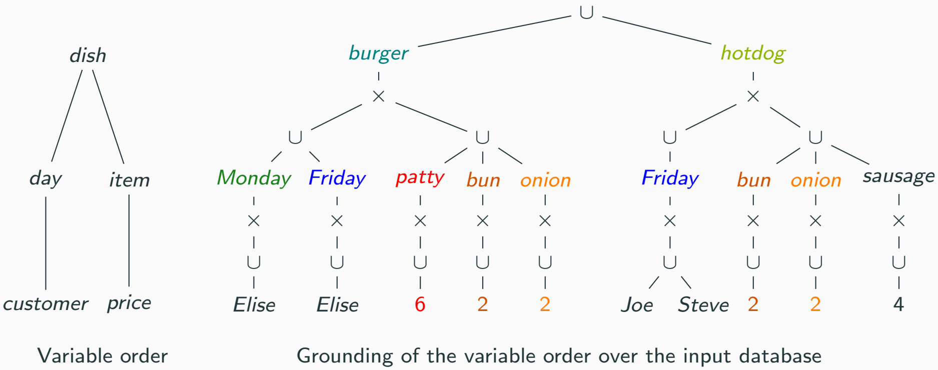
O(customer, day, dish), D(dish, item), I(item, price)

customer	day	dish	item	price
Elise	Monday	burger	patty	6
Elise	Monday	burger	onion	2
Elise	Monday	burger	bun	2
Elise	Friday	burger	patty	6
Elise	Friday	burger	onion	2
Elise	Friday	burger	bun	2
...	...	...	...	...

An algebraic encoding uses product ($\times$), union ($\cup$), and values:

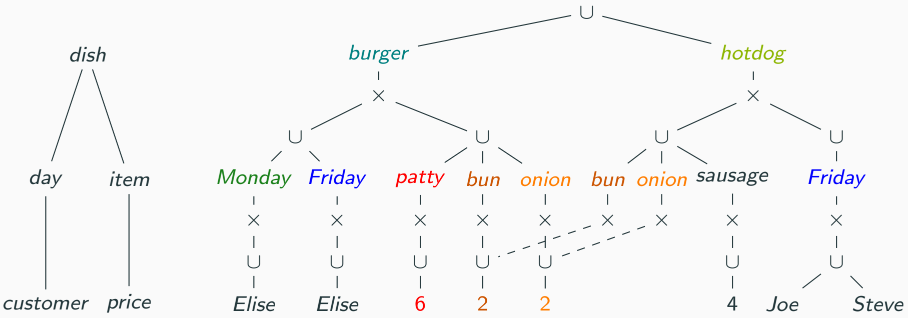
$$
\begin{array}{lllllllllll}
\textit{Elise} & \times & \textit{Monday} & \times & \textit{burger} & \times & \text{patty} & \times & 6 & \cup \\
\textit{Elise} & \times & \textit{Monday} & \times & \textit{burger} & \times & \text{onion} & \times & 2 & \cup \\
\textit{Elise} & \times & \textit{Monday} & \times & \textit{burger} & \times & \text{bun} & \times & 2 & \cup \\
\textit{Elise} & \times & \textit{Friday} & \times & \textit{burger} & \times & \text{patty} & \times & 6 & \cup \\
\textit{Elise} & \times & \textit{Friday} & \times & \textit{burger} & \times & \text{onion} & \times & 2 & \cup \\
\textit{Elise} & \times & \textit{Friday} & \times & \textit{burger} & \times & \text{bun} & \times & 2 & \cup \ldots
\end{array}
$$

Factorised Join



There are several algebraically equivalent factorised joins defined by distributivity of product over union and their commutativity.

.. Now with Further Compression

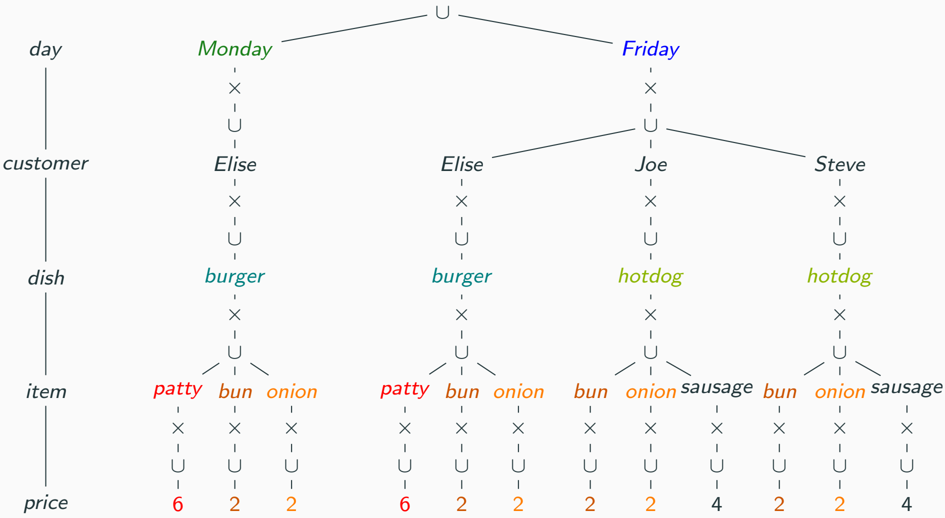


Observation:

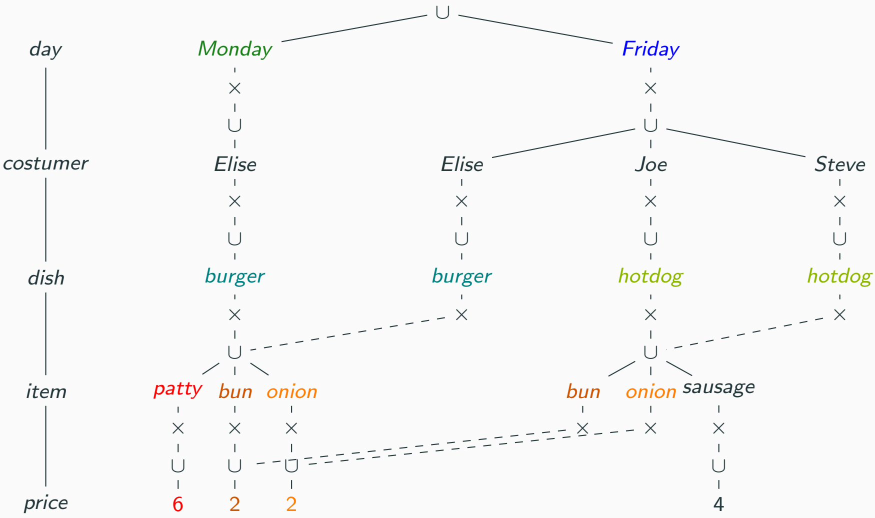
- `price` is under `item`, which is under `dish`, but only *depends* on `item`,
- .. so the same `price` appears under an `item` *regardless* of the `dish`.

Idea: *Cache* `price` for a specific `item` and avoid repetition!

Same Data, Different Factorisation



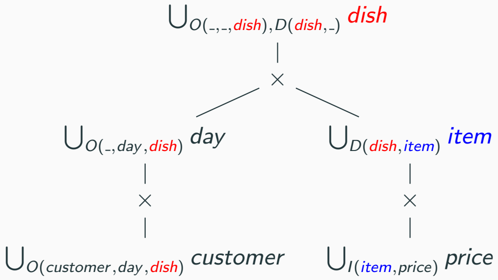
.. and Further Compressed



Grounding Variable Orders to Factorised Joins

Our join: O(customer, day, dish), D(dish, item), I(item, price)

can be grounded to a factorised join as follows:



This grounding follows the previous variable order.

Grounding Variable Orders to Factorised Joins

$$\bigcup_{O(\_,\_,\mathit{dish}),D(\mathit{dish},\_)} \mathit{dish}$$

$$\times$$

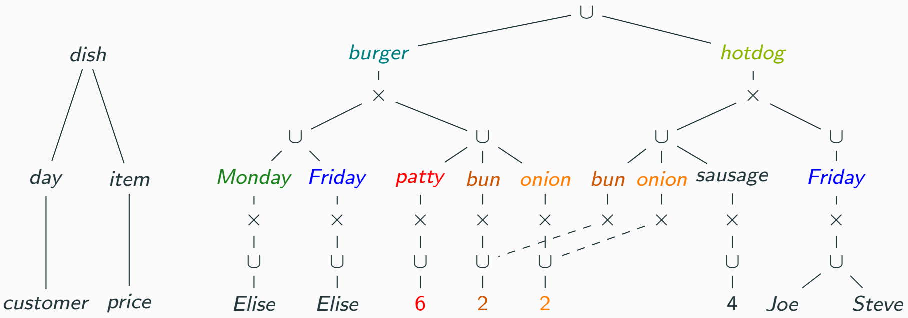
$$\bigcup_{O(\_,\mathit{day},\mathit{dish})} \mathit{day} \qquad\qquad \bigcup_{D(\mathit{dish},\mathit{item})} \mathit{item}$$

$$\times \qquad\qquad\qquad\qquad \times$$

$$\bigcup_{O(\mathit{customer},\mathit{day},\mathit{dish})} \mathit{customer} \qquad \bigcup_{I(\mathit{item},\mathit{price})} \mathit{price}$$

- Relations sorted following topological order of the variable order
- Intersection of O and D on *dish* in time $\widetilde{\mathcal{O}}(\min(|\pi_{\mathit{dish}} O|, |\pi_{\mathit{dish}} D|))$
- The remaining operations are lookups in the relations, where we first fix the *dish* value and then the *day* and *item* values

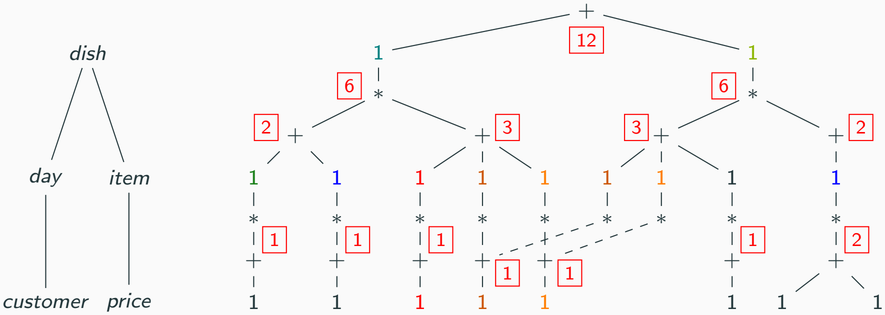
Factorising the Computation of Aggregates (1/2)



COUNT(*) computed in one pass over the factorisation:

- values $\mapsto 1$,
- $\cup \mapsto +$,
- $\times \mapsto *$.

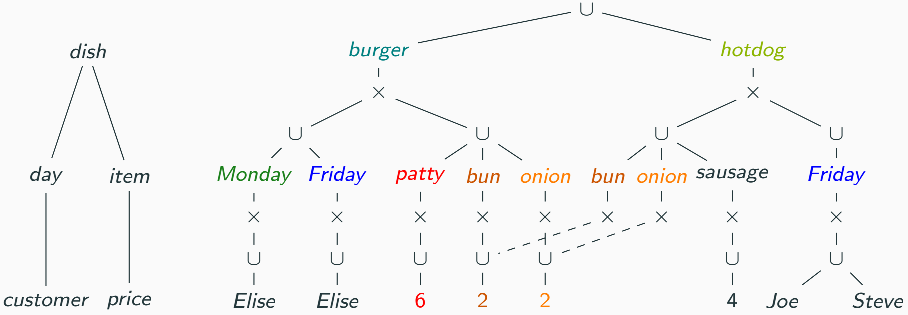
Factorising the Computation of Aggregates (1/2)



COUNT(*) computed in one pass over the factorisation:

- values $\mapsto 1$,
- $\cup \mapsto +$,
- $\times \mapsto *$.

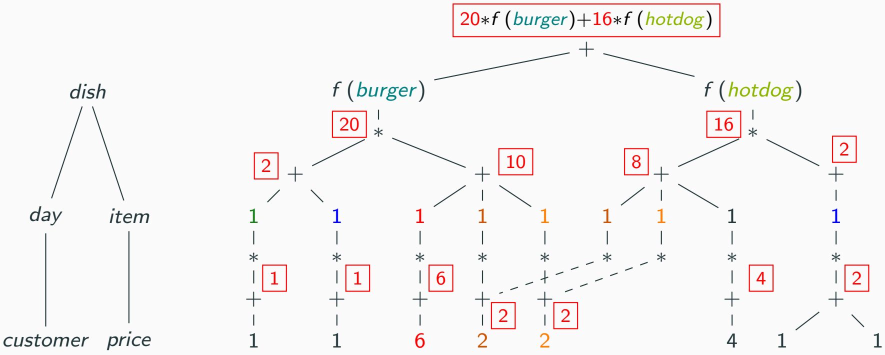
Factorising the Computation of Aggregates (2/2)



SUM(dish * price) computed in one pass over the factorisation:

- Assume there is a function f that turns dish into reals.
- All values except for dish & price $\mapsto 1$,
- $\cup \mapsto +$,
- $\times \mapsto *$.

Factorising the Computation of Aggregates (2/2)



SUM(dish * price) computed in one pass over the factorisation:

- Assume there is a function f that turns dish into reals.
- All values except for dish & price $\mapsto 1$,
- $\cup \mapsto +$,
- $\times \mapsto *$.

Talk Outline

Current Landscape for ML over DB

Factorised Learning over Normalised Data

Learning under Functional Dependencies

General Problem Formulation

Model Reparameterisation using Functional Dependencies

Consider the functional dependency `city` $\rightarrow$ `country` and

- `country` categories: `vietnam`, `england`
- `city` categories: `saigon`, `hanoi`, `oxford`, `leeds`,`bristol`

The one-hot encoding enforces the following identities:

- $x_{\texttt{vietnam}} = x_{\texttt{saigon}} + x_{\texttt{hanoi}}$
  country is vietnam $\Rightarrow$ city is either saigon or hanoi
  $x_{\texttt{vietnam}} = 1 \Rightarrow$ either $x_{\texttt{saigon}} = 1$ or $x_{\texttt{hanoi}} = 1$

- $x_{\texttt{england}} = x_{\text{oxford}} + x_{\text{leeds}} + x_{\text{bristol}}$

  country is england $\Rightarrow$ city is either oxford, leeds, or bristol
  $x_{\texttt{england}} = 1 \Rightarrow$ either $x_{\texttt{oxford}} = 1$ or $x_{\texttt{leeds}} = 1$ or $x_{\texttt{bristol}} = 1$

Model Reparameterisation using Functional Dependencies

- Identities due to one-hot encoding

$$x_{\texttt{vietnam}} = x_{\texttt{saigon}} + x_{\texttt{hanoi}}$$

$$x_{\texttt{england}} = x_{\text{oxford}} + x_{\text{leeds}} + x_{\text{bristol}}$$

- Encode $\mathbf{x}_{\texttt{country}}$ as $\mathbf{x}_{\texttt{country}} = \mathbf{R}\mathbf{x}_{\texttt{city}}$, where

$$\mathbf{R} = \begin{array}{ccccc c} \text{saigon} & \text{hanoi} & \text{oxford} & \text{leeds} & \text{bristol} & \\ 1 & 1 & 0 & 0 & 0 & \text{vietnam} \\ 0 & 0 & 1 & 1 & 1 & \text{england} \end{array}$$

For instance, if $\texttt{city}$ is $\texttt{saigon}$, i.e., $\mathbf{x}_{\texttt{city}} = [1, 0, 0, 0, 0]^\top$, then $\texttt{country}$ is $\texttt{vietnam}$, i.e., $\mathbf{x}_{\texttt{country}} = \mathbf{R}\mathbf{x}_{\texttt{city}} = [1, 0]^\top$.

$$\begin{bmatrix} 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 \end{bmatrix} \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix} = \begin{bmatrix} 1 \\ 0 \end{bmatrix}$$

Model Reparameterisation using Functional Dependencies

- Functional dependency: `city` $\rightarrow$ `country`
- $\mathbf{x}_{\texttt{country}} = \mathbf{R}\mathbf{x}_{\texttt{city}}$
- Replace all occurrences of $\mathbf{x}_{\texttt{country}}$ by $\mathbf{R}\mathbf{x}_{\texttt{city}}$:

$$
\begin{aligned}
& \sum_{f \in F - \{\texttt{city},\texttt{country}\}} \langle \boldsymbol{\theta}_f, \mathbf{x}_f \rangle + \langle \boldsymbol{\theta}_{\texttt{country}}, \mathbf{x}_{\texttt{country}} \rangle + \langle \boldsymbol{\theta}_{\texttt{city}}, \mathbf{x}_{\texttt{city}} \rangle \\
= & \sum_{f \in F - \{\texttt{city},\texttt{country}\}} \langle \boldsymbol{\theta}_f, \mathbf{x}_f \rangle + \langle \boldsymbol{\theta}_{\texttt{country}}, \mathbf{R}\mathbf{x}_{\texttt{city}} \rangle + \langle \boldsymbol{\theta}_{\texttt{city}}, \mathbf{x}_{\texttt{city}} \rangle \\
= & \sum_{f \in F - \{\texttt{city},\texttt{country}\}} \langle \boldsymbol{\theta}_f, \mathbf{x}_f \rangle + \left\langle \underbrace{\mathbf{R}^\top \boldsymbol{\theta}_{\texttt{country}} + \boldsymbol{\theta}_{\texttt{city}}}_{\gamma_{\texttt{city}}}, \mathbf{x}_{\texttt{city}} \right\rangle
\end{aligned}
$$

Model Reparameterisation using Functional Dependencies

- Functional dependency: `city` $\rightarrow$ `country`
- $\mathbf{x}_{\texttt{country}} = \mathbf{R}\mathbf{x}_{\texttt{city}}$
- Replace all occurrences of $\mathbf{x}_{\texttt{country}}$ by $\mathbf{R}\mathbf{x}_{\texttt{city}}$:

$$\begin{aligned}
& \sum_{f \in F - \{\texttt{city},\texttt{country}\}} \langle \boldsymbol{\theta}_f, \mathbf{x}_f \rangle + \langle \boldsymbol{\theta}_{\texttt{country}}, \mathbf{x}_{\texttt{country}} \rangle + \langle \boldsymbol{\theta}_{\texttt{city}}, \mathbf{x}_{\texttt{city}} \rangle \\
= & \sum_{f \in F - \{\texttt{city},\texttt{country}\}} \langle \boldsymbol{\theta}_f, \mathbf{x}_f \rangle + \langle \boldsymbol{\theta}_{\texttt{country}}, \mathbf{R}\mathbf{x}_{\texttt{city}} \rangle + \langle \boldsymbol{\theta}_{\texttt{city}}, \mathbf{x}_{\texttt{city}} \rangle \\
= & \sum_{f \in F - \{\texttt{city},\texttt{country}\}} \langle \boldsymbol{\theta}_f, \mathbf{x}_f \rangle + \left\langle \underbrace{\mathbf{R}^{\top} \boldsymbol{\theta}_{\texttt{country}} + \boldsymbol{\theta}_{\texttt{city}}}_{\gamma_{\texttt{city}}}, \mathbf{x}_{\texttt{city}} \right\rangle
\end{aligned}$$

- We avoid the computation of the aggregates over $\mathbf{x}_{\texttt{country}}$.
- We reparameterise and ignore parameters $\boldsymbol{\theta}_{\texttt{country}}$.
- What about the penalty term in the loss function?

Model Reparameterisation using Functional Dependencies

- Functional dependency: `city` $\rightarrow$ `country`
- $\mathbf{x}_{\texttt{country}} = \mathbf{R}\mathbf{x}_{\texttt{city}}$ $\qquad \gamma_{\texttt{city}} = \mathbf{R}^\top \boldsymbol{\theta}_{\texttt{country}} + \boldsymbol{\theta}_{\texttt{city}}$
- Rewrite the penalty term

$$\|\boldsymbol{\theta}\|_2^2 = \sum_{j \neq \texttt{city}} \|\boldsymbol{\theta}_j\|_2^2 + \left\|\gamma_{\texttt{city}} - \mathbf{R}^\top \boldsymbol{\theta}_{\texttt{country}}\right\|_2^2 + \|\boldsymbol{\theta}_{\texttt{country}}\|_2^2$$

- Optimise out $\boldsymbol{\theta}_{\texttt{country}}$ by expressing it in terms of $\gamma_{\texttt{city}}$:

$$\boldsymbol{\theta}_{\texttt{country}} = (\mathbf{I}_{\texttt{country}} + \mathbf{R}\mathbf{R}^\top)^{-1}\mathbf{R}\gamma_{\texttt{city}} = \mathbf{R}(\mathbf{I}_{\texttt{city}} + \mathbf{R}^\top\mathbf{R})^{-1}\gamma_{\texttt{city}}$$

- The penalty term becomes

$$\|\boldsymbol{\theta}\|_2^2 = \sum_{j \neq \texttt{city}} \|\boldsymbol{\theta}_j\|_2^2 + \left\langle (\mathbf{I}_{\texttt{city}} + \mathbf{R}^\top\mathbf{R})^{-1}\gamma_{\texttt{city}}, \gamma_{\texttt{city}} \right\rangle$$

Talk Outline

Current Landscape for ML over DB

Factorised Learning over Normalised Data

Learning under Functional Dependencies

General Problem Formulation

General Problem Formulation

We want to solve $\boldsymbol{\theta}^* := \arg\min_{\boldsymbol{\theta}} J(\boldsymbol{\theta})$, where

$$J(\boldsymbol{\theta}) := \sum_{(\mathbf{x},y)\in D} \mathcal{L}\left(\langle g(\boldsymbol{\theta}), h(\mathbf{x})\rangle, y\right) + \Omega(\boldsymbol{\theta}).$$

- $\boldsymbol{\theta} = (\theta_1, \ldots, \theta_p) \in \mathbf{R}^p$ are parameters
- functions $g : \mathbf{R}^p \to \mathbf{R}^m$ and $h : \mathbf{R}^n \to \mathbf{R}^m$
  - $g = (g_j)_{j\in[m]}$ is a vector of multivariate polynomials
  - $h = (h_j)_{j\in[m]}$ is a vector of multivariate monomials
- $\mathcal{L}$ is a loss function, Ω is the regulariser
- D is the training dataset with features $\mathbf{x}$ and response y.

Problems: ridge linear regression, polynomial regression, Factorisation machines; logistic regression, SVM; PCA.

Special Case: Ridge Linear Regression

Under

- square loss $\mathcal{L}$, ℓ_2-regularisation,
- data points $\mathbf{x} = (x_0, x_1, \ldots, x_n, y)$,
- $p = n + 1$ parameters $\boldsymbol{\theta} = (\theta_0, \ldots, \theta_n)$,
- $x_0 = 1$ corresponds to the bias parameter θ_0,
- identity functions g and h,

we obtain the following formulation for ridge linear regression:

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

Special Case: Degree-d Polynomial Regression

Under

- square loss $\mathcal{L}$, ℓ_2-regularisation,
- data points $\mathbf{x} = (x_0, x_1, \ldots, x_n, y)$,
- $p = m = 1 + n + n^2 + \cdots + n^d$ parameters $\boldsymbol{\theta} = (\theta_{\mathbf{a}})$, where $\mathbf{a} = (a_1, \ldots, a_n)$ with non-negative integers s.t. $\|\mathbf{a}\|_1 \leq d$.
- the components of h are given by $h_{\mathbf{a}}(\mathbf{x}) = \prod_{i=1}^{n} x_i^{a_i}$,
- $g(\boldsymbol{\theta}) = \boldsymbol{\theta}$,

we obtain the following formulation for polynomial regression:

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

Special Case: Factorisation Machines

Under

- square loss $\mathcal{L}$, ℓ_2-regularisation,
- data points $\mathbf{x} = (x_0, x_1, \ldots, x_n, y)$,
- $p = 1 + n + r \cdot n$ parameters,
- $m = 1 + n + \binom{n}{2}$ features,

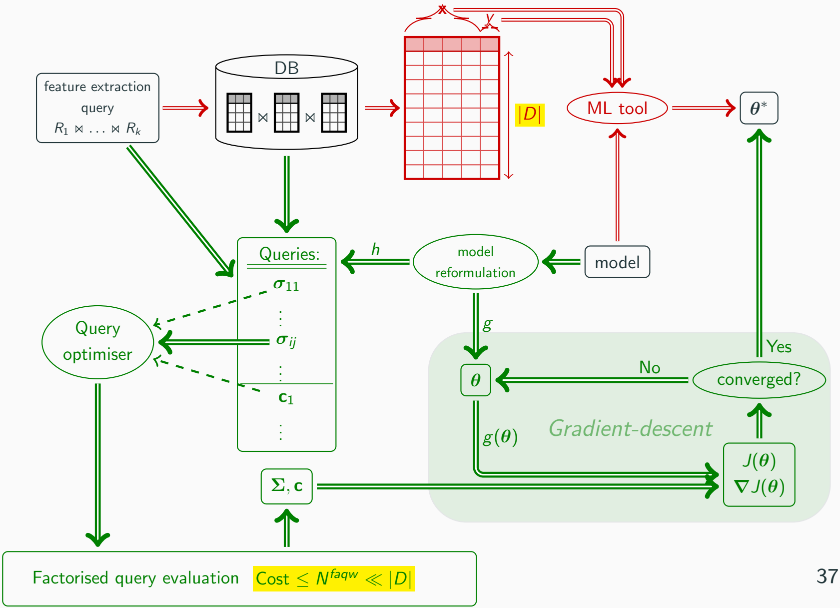
we obtain the following formulation for degree-2 rank-r Factorisation machines:

$$J(\boldsymbol{\theta}) := \frac{1}{2|D|} \sum_{(\mathbf{x},y)\in D} \left(\sum_{i=0}^{n} \theta_i x_i + \sum_{\substack{\{i,j\}\in\binom{[n]}{2} \\ \ell\in[r]}} \theta_i^{(\ell)} \theta_j^{(\ell)} x_i x_j - y \right)^2 + \frac{\lambda}{2} \|\boldsymbol{\theta}\|_2^2 .$$

Special Case: Classifiers

- Typically, the regulariser is $\frac{\lambda}{2}\|\boldsymbol{\theta}\|_2^2$
- The response is binary: $y \in \{\pm 1\}$
- The loss function $\mathcal{L}(\gamma, y)$, where $\gamma := \langle g(\boldsymbol{\theta}), h(\mathbf{x})\rangle$ is
  - $\mathcal{L}(\gamma, y) = \max\{1 - y\gamma, 0\}$ for support vector machines,
  - $\mathcal{L}(\gamma, y) = \log(1 + e^{-y\gamma})$ for logistic regression,
  - $\mathcal{L}(\gamma, y) = e^{-y\gamma}$ for Adaboost.

Zoom-in: In-database vs. Out-of-database Learning



Thank you!