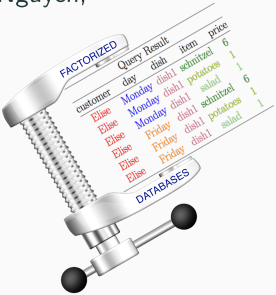
In-Database Learning with Sparse Tensors

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

Toronto, October 2017

RelationalAI



Talk Outline

Brief Outlook at Current Landscape for DB+ML (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 DB+ML solutions seem to operate in this space

Brief Outlook at Current Landscape for DB+ML (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 DB+ML solutions seem to 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 DB+ML (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 DB+ML (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

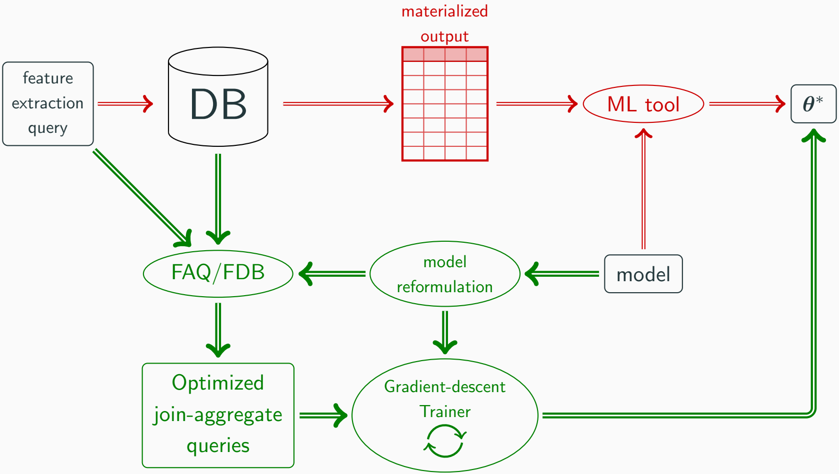
Tight integration ⇒ In-Database Analytics

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

In-Database Analytics

- Move the analytics, not the data
  - Avoid expensive data export/import
  - Exploit database technologies
  - Build better models using larger datasets
- 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?

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	$> 24h$
R	Join (PSQL)	50.63	–
	Export/Import	308.83	–
	Learn	490.13	–
Our approach	Aggregate+Join	25.51	380.31
	Converge (runs)	0.02 (343)	8.82 (366)
Polynomial regression degree 2			
Features	(cont+categ)	562+2,363	562+141k
Aggregates	(cont+categ)	158k+742k	158k+37M
MadLib	Learn	$> 24h$	–
Our approach	Aggregate+Join	132.43	1,819.80
	Converge (runs)	3.27 (321)	219.51 (180)

Talk Outline

Current Landscape for DB+ML

What We Did So Far

- Factorized Learning over Normalized Data
- Learning under Functional Dependencies

Our Current Focus

Unified In-Database Analytics for Optimization 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 for regression: batch gradient descent
  - Convergence rates are dimension-free
- ML tasks: ridge linear regression, degree-d polynomial regression, degree-d factorization machines; logistic regression, SVM; PCA.

Typical Retail Example

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

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

- 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

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

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

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

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

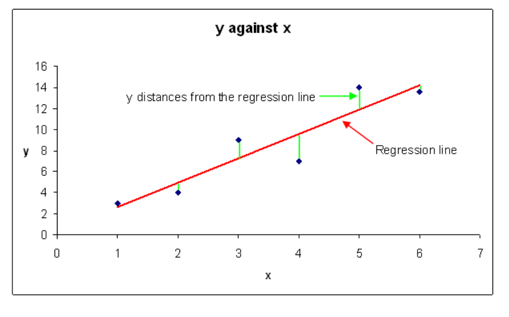
Side Note: One-hot Encoding of Categorical Variables

- 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

Side Note: Least Square Loss Function

Goal: Describe a linear relationship $fun(x) = \theta_1 x + \theta_0$ so we can estimate new y values given new x values.



- We are given n (black) data points $(x_i, y_i)_{i \in [n]}$
- We would like to find a (red) regression line $fun(x)$ such that the (green) error $\sum_{i \in [n]} (fun(x_i) - y_i)^2$ is minimized

From Optimization to SumProduct FAQ 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 Optimization to SumProduct FAQ 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 Optimization to SumProduct FAQ 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:

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

Then,

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

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

Expressing Σ, c, s_Y as SumProduct FAQ Queries

FAQ 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 free variable

$$\psi_{ij} = \sum_{f \in F : a_f \in \mathrm{Dom}(x_f)} \sum_{b \in B : a_b \in \mathrm{Dom}(x_b)} a_i \cdot a_j \cdot \prod_{k \in [5]} \mathbf{1}_{R_k(\mathbf{a}_{\mathcal{S}(R_k)})}$$

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

$$\psi_{ij}[a_i] = \sum_{f \in F - \{i\} : a_f \in \mathrm{Dom}(x_f)} \sum_{b \in B : a_b \in \mathrm{Dom}(x_b)} a_j \cdot \prod_{k \in [5]} \mathbf{1}_{R_k(\mathbf{a}_{\mathcal{S}(R_k)})}$$

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

$$\psi_{ij}[a_i, a_j] = \sum_{f \in F - \{i,j\} : a_f \in \mathrm{Dom}(x_f)} \sum_{b \in B : a_b \in \mathrm{Dom}(x_b)} \prod_{k \in [5]} \mathbf{1}_{R_k(\mathbf{a}_{\mathcal{S}(R_k)})}$$

Expressing Σ, $\mathbf{c}$, s_Y 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 attribute

$$\textbf{SELECT SUM}(x_i * x_j)\ \textbf{FROM}\ D;$$

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

$$\textbf{SELECT}\ x_i,\ \textbf{SUM}(x_j)\ \textbf{FROM}\ D\ \textbf{GROUP BY}\ x_i;$$

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

$$\textbf{SELECT}\ x_i, x_j,\ \textbf{SUM}(1)\ \textbf{FROM}\ D\ \textbf{GROUP BY}\ x_i, x_j;$$

This query encoding avoids drawbacks of one-hot encoding

Side Note: Factorized Learning over Normalized Data

Idea: Avoid Redundant Computation for DB Join and ML

Realized to varying degrees in the literature

Side Note: Factorized Learning over Normalized Data

Idea: Avoid Redundant Computation for DB Join and ML

Realized to varying degrees in the literature

- Rendle (libFM): Discover repeating blocks in the materialized join and then compute ML once for all
  - Same complexity as join materialization!
  - NP-hard to (re)discover join dependencies!

Side Note: Factorized Learning over Normalized Data

Idea: Avoid Redundant Computation for DB Join and ML

Realized to varying degrees in the literature

- Rendle (libFM): Discover repeating blocks in the materialized join and then compute ML once for all
  - Same complexity as join materialization!
  - NP-hard to (re)discover join dependencies!
- Kumar (Morpheus): Push down ML aggregates to each input tuple, then join tables and combine aggregates
  - Same complexity as listing materialization of join results!

Side Note: Factorized Learning over Normalized Data

Idea: Avoid Redundant Computation for DB Join and ML

Realized to varying degrees in the literature

- Rendle (libFM): Discover repeating blocks in the materialized join and then compute ML once for all
  - Same complexity as join materialization!
  - NP-hard to (re)discover join dependencies!
- Kumar (Morpheus): Push down ML aggregates to each input tuple, then join tables and combine aggregates
  - Same complexity as listing materialization of join results!
- Our approach: Morpheus + Factorize the join to avoid expensive Cartesian products in join computation
  - Arbitrarily lower complexity than join materialization

Model Reparameterization 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 Reparameterization 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

$\mathbf{R} =$	saigon	hanoi	oxford	leeds	bristol	
	1	1	0	0	0	vietnam
	0	0	1	1	1	england

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 Reparameterization 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 Reparameterization using Functional Dependencies

- Functional dependency: `city` → `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 computing aggregates over $\mathbf{x}_{\texttt{country}}$.
- We reparameterize and ignore parameters $\boldsymbol{\theta}_{\texttt{country}}$.
- What about the penalty term in the loss function?

Model Reparameterization 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

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

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

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

- The penalty term becomes

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

Side Note: Learning over Normalized Data with FDs

Hamlet & Hamlet^{++}

- Linear classifiers (Naïve Bayes): model accuracy unlikely to be affected if we drop *a few* functionally determined features
- Use simple decision rule: fkeys/key $>$ 20?
- Hamlet^{++} shows experimentally that this idea does not work for more interesting classifiers, e.g., decision trees

Side Note: Learning over Normalized Data with FDs

Hamlet & Hamlet^{++}

- Linear classifiers (Naïve Bayes): model accuracy unlikely to be affected if we drop *a few* functionally determined features
- Use simple decision rule: fkeys/key $>$ 20?
- Hamlet^{++} shows experimentally that this idea does not work for more interesting classifiers, e.g., decision trees

Our approach

- Given the model A to learn, we map it to a much smaller model B without the functionally determined features in A
- Learning B can be OOM faster than learning A
- Once B is learned, we map it back to A

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 regularizer
- D is the training dataset with features $\mathbf{x}$ and response y.

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

Special Case: Ridge Linear Regression

Under

- square loss $\mathcal{L}$, ℓ_2-regularization,
- 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:

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

Special Case: Degree-d Polynomial Regression

Under

- square loss $\mathcal{L}$, ℓ_2-regularization,
- 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: Factorization Machines

Under

- square loss $\mathcal{L}$, ℓ_2-regularization,
- 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,

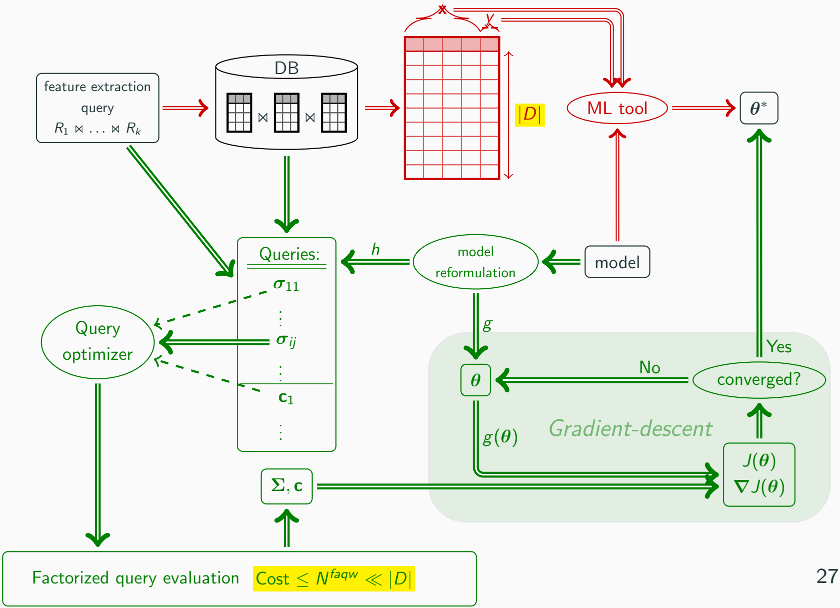
we obtain the following formulation for degree-2 rank-r factorization machines:

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

Special Case: Classifiers

- Typically, the regularizer 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



Talk Outline

Current Landscape for DB+ML

What We Did So Far

- Factorized Learning over Normalized Data
- Learning under Functional Dependencies

Our Current Focus

Our Current Focus

- MultiFAQ: Principled approach to computing many FAQs over the same hypertree decomposition
  - Asymptotically lower complexity than computing each FAQ independently
  - Applications: regression, decision trees, frequent itemset
- SGD using sampling from factorized joins
  - Applications: regression, decision trees, frequent itemset
- in-DB linear algebra
  - Generalization of current effort, add support for efficient matrix operations, e.g., inversion

Thank you!