In-Database Learning with Sparse Tensors

Mahmoud Abo Khamis, Hung Ngo, XuanLong Nguyen, Dan Olteanu, and Maximilian Schleich
Toronto, October 2017

RelationalAI
Current Landscape for DB+ML

What We Did So Far

Factorized Learning over Normalized Data
Learning under Functional Dependencies

Our Current Focus
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
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
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**
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

- Feature extraction query
- DB
- Materialized output
- ML tool
- $\theta^*$

FAQ/FDB

Optimized join-aggregate queries

Model reformulation

Gradient-descent Trainer
## Does It Pay Off?

<table>
<thead>
<tr>
<th>Retailer dataset (records)</th>
<th>excerpt (17M)</th>
<th>full (86M)</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Linear regression</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Features (cont+categ)</td>
<td>33 + 55</td>
<td>33+3,653</td>
</tr>
<tr>
<td>Aggregates (cont+categ)</td>
<td>595+2,418</td>
<td>595+145k</td>
</tr>
<tr>
<td>MadLib Learn</td>
<td>1,898.35</td>
<td>&gt; 24h</td>
</tr>
<tr>
<td>R Join (PSQL)</td>
<td>50.63</td>
<td>–</td>
</tr>
<tr>
<td>Export/Import</td>
<td>308.83</td>
<td>–</td>
</tr>
<tr>
<td>Learn</td>
<td>490.13</td>
<td>–</td>
</tr>
<tr>
<td><strong>Our approach</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Aggregate+Join</td>
<td>25.51</td>
<td>380.31</td>
</tr>
<tr>
<td>Converge (runs)</td>
<td>0.02 (343)</td>
<td>8.82 (366)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th><strong>Polynomial regression degree 2</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>Features (cont+categ)</td>
</tr>
<tr>
<td>Aggregates (cont+categ)</td>
</tr>
<tr>
<td>MadLib Learn</td>
</tr>
<tr>
<td><strong>Our approach</strong></td>
</tr>
<tr>
<td>Aggregate+Join</td>
</tr>
<tr>
<td>Converge (runs)</td>
</tr>
</tbody>
</table>
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** \( Q \):

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

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

- We learn the ridge linear regression model

\[
\langle \theta, x \rangle = \sum_{f \in F} \langle \theta_f, x_f \rangle
\]

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

- The parameters \( \theta \) are obtained by minimizing the objective function:

\[
J(\theta) = \frac{1}{2|D|} \sum_{(x,y) \in D} (\langle \theta, x \rangle - y)^2 + \|	heta\|_2^2
\]
Side Note: One-hot Encoding of Categorical Variables

- **Continuous** variables are mapped to scalars
  - $y_{\text{unitsSold}} \in \mathbb{R}$.

- **Categorical** variables are mapped to indicator vectors
  - country has categories vietnam and england
  - country is then mapped to an indicator vector
    - $x_{\text{country}} = [x_{\text{vietnam}}, x_{\text{england}}]^T \in (\{0, 1\}^2)^T$.
  - $x_{\text{country}} = [0, 1]^T$ for a tuple with country = ‘england’.

This encoding leads to wide training datasets and many 0s.
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
We can solve $\theta^* := \arg\min_{\theta} J(\theta)$ by repeatedly updating $\theta$ in the direction of the gradient until convergence:

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

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

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

$$\sigma_{ij} = \frac{1}{|D|} \sum_{(x,y) \in D} x_i x_j^\top, \quad c_i = \frac{1}{|D|} \sum_{(x,y) \in D} y \cdot x_i, \quad s_Y = \frac{1}{|D|} \sum_{(x,y) \in D} y^2.$$
We can solve $\theta^* := \arg\min_{\theta} J(\theta)$ by repeatedly updating $\theta$ in the direction of the gradient until convergence:

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

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

$$\sigma_{ij} = \frac{1}{|D|} \sum_{(x,y) \in D} x_i x_j^\top, \quad c_i = \frac{1}{|D|} \sum_{(x,y) \in D} y \cdot x_i, \quad s_Y = \frac{1}{|D|} \sum_{(x,y) \in D} y^2.$$

Then,

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

$$= \frac{1}{2} \theta^\top \Sigma \theta - \langle \theta, c \rangle + \frac{s_Y}{2} + \frac{\lambda}{2} \|\theta\|_2^2.$$
Expressing $\Sigma$, $c$, $s_Y$ as SumProduct FAQ Queries

FAQ queries for $\sigma_{ij} = \frac{1}{|D|} \sum_{(x,y) \in D} x_i x_j^\top$ (w/o factor $\frac{1}{|D|}$):

- $x_i$, $x_j$ continuous $\Rightarrow$ no free variable
  \[ \psi_{ij} = \sum_{f \in F} \sum_{a_f \in \text{Dom}(x_f)} \sum_{b \in B} a_i \cdot a_j \cdot \prod_{k \in [5]} 1_{R_k(a_S(R_k))} \]

- $x_i$ categorical, $x_j$ continuous $\Rightarrow$ one free variable
  \[ \psi_{ij}[a_i] = \sum_{f \in F \setminus \{i\}} \sum_{a_f \in \text{Dom}(x_f)} \sum_{b \in B} a_j \cdot \prod_{k \in [5]} 1_{R_k(a_S(R_k))} \]

- $x_i$, $x_j$ categorical $\Rightarrow$ two free variables
  \[ \psi_{ij}[a_i, a_j] = \sum_{f \in F \setminus \{i,j\}} \sum_{a_f \in \text{Dom}(x_f)} \sum_{b \in B} a_i \cdot \prod_{k \in [5]} 1_{R_k(a_S(R_k))} \]
Expressing $\Sigma$, $c$, $s_Y$ as SQL Queries

SQL queries for $\sigma_{ij} = \frac{1}{|D|} \sum_{(x,y) \in D} x_i x_j^\top$ (w/o factor $\frac{1}{|D|}$):

- $x_i$, $x_j$ continuous $\Rightarrow$ no group-by attribute
  
  $$\text{SELECT SUM}(x_i \cdot x_j) \text{ FROM } D;$$

- $x_i$ categorical, $x_j$ continuous $\Rightarrow$ one group-by attribute
  
  $$\text{SELECT } x_i, \text{ SUM}(x_j) \text{ FROM } D \text{ GROUP BY } x_i;$$

- $x_i$, $x_j$ categorical $\Rightarrow$ two group-by variables
  
  $$\text{SELECT } x_i, x_j, \text{ SUM}(1) \text{ FROM } D \text{ GROUP BY } x_i, x_j;$$

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

• 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
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!
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
Consider the functional dependency *city* → *country* and

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

The one-hot encoding enforces the following identities:

- $x_{\text{vietnam}} = x_{\text{saigon}} + x_{\text{hanoi}}$
  - *country is vietnam* ⇒ *city is either saigon or hanoi*
  - $x_{\text{vietnam}} = 1$ ⇒ either $x_{\text{saigon}} = 1$ or $x_{\text{hanoi}} = 1$

- $x_{\text{england}} = x_{\text{oxford}} + x_{\text{leeds}} + x_{\text{bristol}}$
  - *country is england* ⇒ *city is either oxford, leeds, or bristol*
  - $x_{\text{england}} = 1$ ⇒ either $x_{\text{oxford}} = 1$ or $x_{\text{leeds}} = 1$ or $x_{\text{bristol}} = 1$
Model Reparameterization using Functional Dependencies

- Identities due to one-hot encoding
  \[ x_{vietnam} = x_{saigon} + x_{hanoi} \]
  \[ x_{england} = x_{oxford} + x_{leeds} + x_{bristol} \]
- Encode \( x_{\text{country}} \) as \( x_{\text{country}} = R x_{\text{city}} \), where

\[
R = \begin{bmatrix}
1 & 1 & 0 & 0 & 0 & 0 & 0 & \text{vietnam} \\
0 & 0 & 1 & 1 & 1 & 1 & 0 & \text{england}
\end{bmatrix}
\]

For instance, if city is saigon, i.e., \( x_{\text{city}} = [1, 0, 0, 0, 0]^{\top} \), then country is vietnam, i.e., \( x_{\text{country}} = R x_{\text{city}} = [1, 0]^{\top} \).
Model Reparameterization using Functional Dependencies

- Functional dependency: \( \text{city} \rightarrow \text{country} \)
- \( x_{\text{country}} = Rx_{\text{city}} \)
- Replace all occurrences of \( x_{\text{country}} \) by \( Rx_{\text{city}} \):

\[
\sum_{f \in F - \{\text{city}, \text{country}\}} \langle \theta_f, x_f \rangle + \langle \theta_{\text{country}}, x_{\text{country}} \rangle + \langle \theta_{\text{city}}, x_{\text{city}} \rangle
\]

\[
= \sum_{f \in F - \{\text{city}, \text{country}\}} \langle \theta_f, x_f \rangle + \langle \theta_{\text{country}}, Rx_{\text{city}} \rangle + \langle \theta_{\text{city}}, x_{\text{city}} \rangle
\]

\[
= \sum_{f \in F - \{\text{city}, \text{country}\}} \langle \theta_f, x_f \rangle + \left( R^\top \theta_{\text{country}} + \theta_{\text{city}}, x_{\text{city}} \right)_{\gamma_{\text{city}}}
\]

We avoid computing aggregates over \( x_{\text{country}} \).
We reparameterize and ignore parameters \( \theta_{\text{country}} \).

What about the penalty term in the loss function?
Model Reparameterization using Functional Dependencies

- Functional dependency: city \(\rightarrow\) country
- \(x_{\text{country}} = Rx_{\text{city}}\)
- Replace all occurrences of \(x_{\text{country}}\) by \(Rx_{\text{city}}\):

\[
\sum_{f \in F - \{\text{city, country}\}} \langle \theta_f, x_f \rangle + \langle \theta_{\text{country}}, x_{\text{country}} \rangle + \langle \theta_{\text{city}}, x_{\text{city}} \rangle
\]

\[
= \sum_{f \in F - \{\text{city, country}\}} \langle \theta_f, x_f \rangle + \langle \theta_{\text{country}}, Rx_{\text{city}} \rangle + \langle \theta_{\text{city}}, x_{\text{city}} \rangle
\]

\[
= \sum_{f \in F - \{\text{city, country}\}} \langle \theta_f, x_f \rangle + \left( R^T \theta_{\text{country}} + \theta_{\text{city}}, x_{\text{city}} \right)_{\gamma_{\text{city}}}
\]

- We avoid computing aggregates over \(x_{\text{country}}\).
- We reparameterize and ignore parameters \(\theta_{\text{country}}\).
- What about the penalty term in the loss function?
Model Reparameterization using Functional Dependencies

- Functional dependency: city → country
- $x_{\text{country}} = Rx_{\text{city}}$ $\gamma_{\text{city}} = R^T \theta_{\text{country}} + \theta_{\text{city}}$

- Rewrite the penalty term
  \[
  \|\theta\|_2^2 = \sum_{j \neq \text{city}} \|\theta_j\|_2^2 + \left\|\gamma_{\text{city}} - R^T \theta_{\text{country}}\right\|_2^2 + \|\theta_{\text{country}}\|_2^2
  \]

- Optimize out $\theta_{\text{country}}$ by expressing it in terms of $\gamma_{\text{city}}$:
  \[
  \theta_{\text{country}} = (I_{\text{country}} + RR^T)^{-1}R\gamma_{\text{city}} = R(I_{\text{city}} + R^T R)^{-1}\gamma_{\text{city}}
  \]

- The penalty term becomes
  \[
  \|\theta\|_2^2 = \sum_{j \neq \text{city}} \|\theta_j\|_2^2 + \left\langle (I_{\text{city}} + R^T R)^{-1}\gamma_{\text{city}}, \gamma_{\text{city}} \right\rangle
  \]
Hamlet & Hamlet++

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

- Linear classifiers (Naïve Bayes): model accuracy unlikely to be affected if we drop a few functionally determined features.
- Use simple decision rule: \( f_{\text{keys/key}} > 20? \)
- 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 $\theta^* := \arg \min_{\theta} J(\theta)$, where

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

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

Problems: ridge linear regression, polynomial regression, factorization machines; logistic regression, SVM; PCA.
Special Case: Ridge Linear Regression

Under

- square loss $\mathcal{L}$, $\ell_2$-regularization,
- data points $x = (x_0, x_1, \ldots, x_n, y)$,
- $p = n + 1$ parameters $\theta = (\theta_0, \ldots, \theta_n)$,
- $x_0 = 1$ corresponds to the bias parameter $\theta_0$, 
- identity functions $g$ and $h$,

we obtain the following formulation for ridge linear regression:

$$J(\theta) := \frac{1}{2|D|} \sum_{(x,y) \in D} (\langle \theta, x \rangle - y)^2 + \frac{\lambda}{2} \| \theta \|_2^2.$$
Special Case: Degree-\(d\) Polynomial Regression

Under

- square loss \(L\), \(\ell_2\)-regularization,
- data points \(x = (x_0, x_1, \ldots, x_n, y)\),
- \(p = m = 1 + n + n^2 + \cdots + n^d\) parameters \(\theta = (\theta_a)\), where \(a = (a_1, \ldots, a_n)\) with non-negative integers s.t. \(\|a\|_1 \leq d\).
- the components of \(h\) are given by \(h_a(x) = \prod_{i=1}^n x_i^{a_i}\),
- \(g(\theta) = \theta\),

we obtain the following formulation for polynomial regression:

\[
J(\theta) := \frac{1}{2|D|} \sum_{(x,y) \in D} (\langle g(\theta), h(x) \rangle - y)^2 + \frac{\lambda}{2} \|\theta\|^2.
\]
Special Case: Factorization Machines

Under

- square loss $\mathcal{L}$, $\ell_2$-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:

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

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

Feature extraction query $R_1 \times \ldots \times R_k$

Queries:

$\sigma_{11}$

$\ldots$

$\sigma_{ij}$

$\ldots$

$c_1$

$\ldots$

DB

$\theta^*$

ML tool

$|D|$}

$\Sigma, c$

Factorized query evaluation $Cost \leq N^{faqw} \ll |D|$
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!