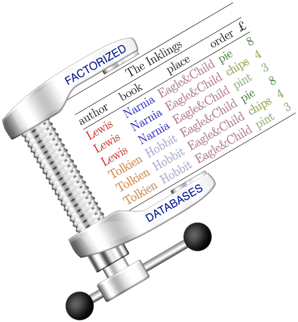
Joins $\rightarrow$ Aggregates $\rightarrow$ Optimization

https://fdbresearch.github.io



Dan Olteanu

PhD Open School
University of Warsaw
November 24, 2018

Acknowledgements

Some work reported in this course has been done in the context of the FDB project, LogicBlox, and RelationalAI by

- Zavodný, Schleich, Kara, Nikolic, Zhang, Ciucanu, and Olteanu (Oxford)
- Abo Khamis and Ngo (RelationalAI), Nguyen (U. Michigan)

Some of the following slides are derived from presentations by

- Aref (motivation)
- Abo Khamis (optimization diagrams)
- Kara (covers, IVM$^{\epsilon}$, and many graphics)
- Ngo (functional aggregate queries)
- Schleich (performance and quizzes)

Lastly, Kara and Schleich proofread the slides.

I would like to thank them for their support!

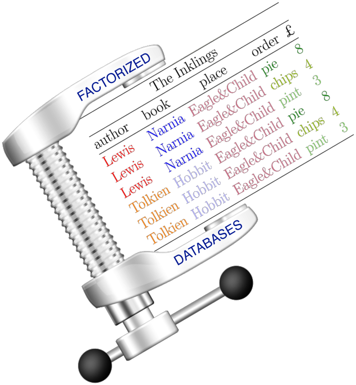
Goal of This Course

$\boxed{\text{Introduction to a principled approach to in-database computation}}$

This course starts where mainstream database courses finish.

- Part 1: Joins
- Part 2: Aggregates
- **Part 3: Optimization**
  - Learning models inside vs outside the database
  - From learning to factorized aggregate computation
  - Learning under functional dependencies
  - In-database linear algebra: Decompositions of matrices defined by joins

Outline of Part 3: Optimization



In-Database Learning

Model Reformulation

Learning under Functional Dependencies

In-Database Linear Algebra

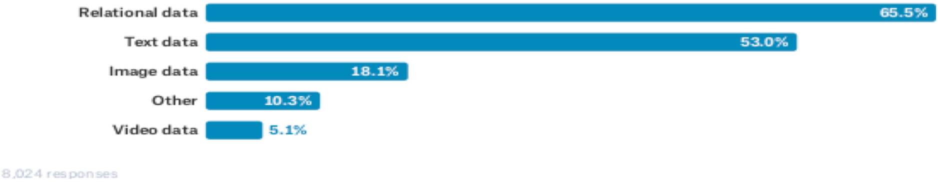
References

Quiz

AI/ML: The Next Big Opportunity

- AI is emerging as general purpose technology
  - Just as computing became general purpose 70 years ago
- A core ability of intelligence is the ability to predict
  - Convert information you have into information you need
- The quality of the prediction is increasing as the cost per prediction is decreasing
  - We use more of it to solve existing problems
    - Consumer demand forecasting
  - We use it for new problems where it was not used before
    - From broadcast to personalized advertising
    - From shop-then-ship to ship-then-shop

Most Enterprises Rely on Relational Data for AI Models



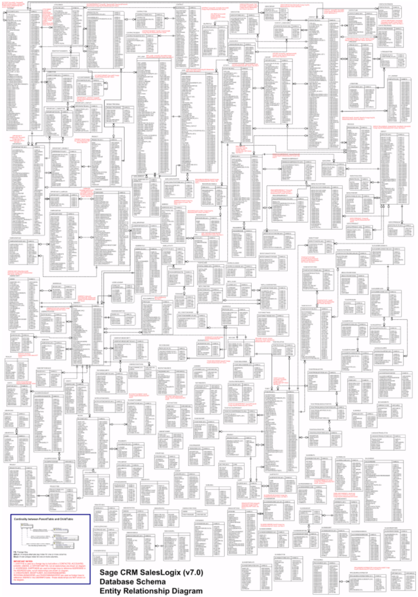
- Retail: 86% relational
- Insurance: 83% relational
- Marketing: 82% relational
- Financial: 77% relational

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

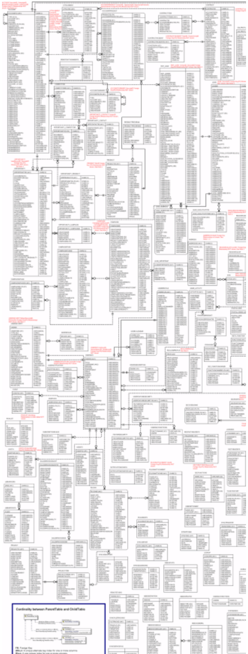
Relational Model: The Jewel in the Database Crown

- Last 40 years have witnessed massive adoption of the Relational Model
- Many human hours invested in building relational models
- Relational databases are rich with knowledge of the underlying domains
- Availability of curated data made it possible to learn from the past and to predict the future for both

  humans (BI) and machines (AI)



Current State of Affairs in Building Predictive Models



→

Current ML technology

THROWS AWAY

the relational structure

and domain knowledge

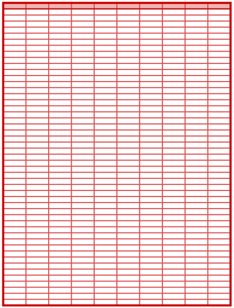
that can help build

BETTER MODELS

Design matrix

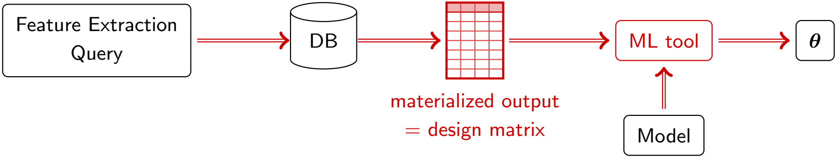
Features

Samples



Learning over Relational Databases: Revisit from First Principles

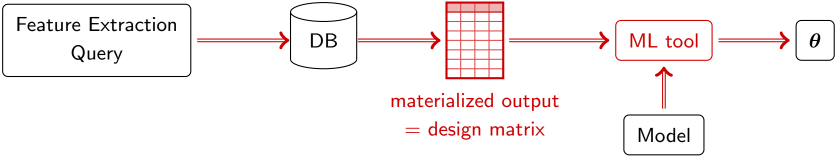
In-database vs. Out-of-database Learning



Out-of-database learning requires: [KBY17,PRWZ17]

1. Materializing the query result
2. DBMS data export and ML tool import
3. One/multi-hot encoding of categorical variables

In-database vs. Out-of-database Learning

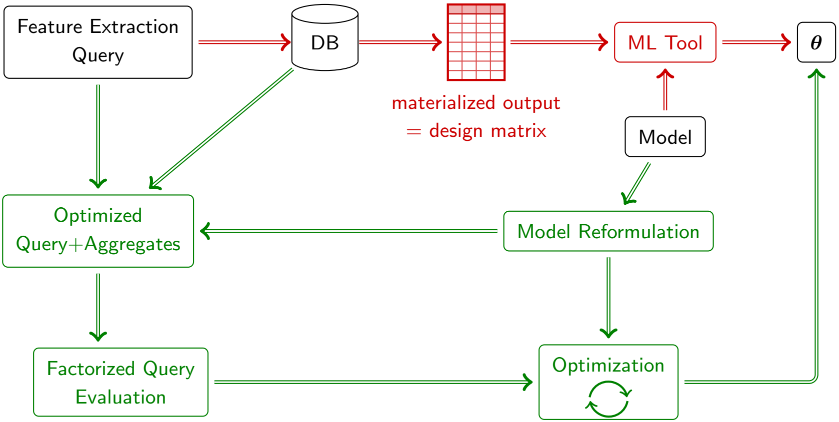


Out-of-database learning requires: [KBY17,PRWZ17]

1. Materializing the query result
2. DBMS data export and ML tool import
3. One/multi-hot encoding of categorical variables

All these steps are very expensive and unnecessary!

In-database vs. Out-of-database Learning [ANNOS18a+b]

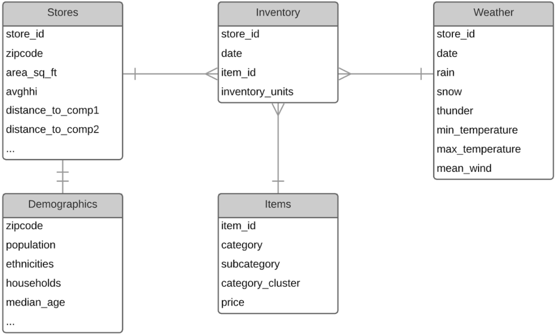


In-database learning exploits the query structure, the database schema, and the constraints.

Aggregation is the Aspiring to All Problems [SOANN19]

Model	# Features	# Aggregates
Supervised: Regression		
Linear regression	n	$O(n^2)$
Polynomial regression degree d	$O(n^d)$	$O(n^{2d})$
Factorization machines degree d	$O(n^d)$	$O(n^{2d})$
Supervised: Classification		
Decision tree (k nodes)	n	$O(k \cdot n \cdot p \cdot c)$
(c conditions/feature, p categories/label)		
Unsupervised		
k-means (const approx)	n	$O(k \cdot n)$
PCA (rank k)	n	$O(k \cdot n^2)$
Chow-Liu tree	n	$O(n^2)$

Does This Matter in Practice? 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
			2.1 GB

Out-of-Database Solution: PostgreSQL+TensorFlow

Train a linear regression model to predict inventory units

Design matrix defined by

- the natural join of all relations, where
- the join keys are removed

Join of Inventory, Items, Stores, Demographics, Weather	
Cardinality (# rows)	84,055,817
Arity (# columns)	44 (3 + 41)
Size on disk	23GB
Time to compute in PostgreSQL	217 secs
Time to Export from PostgreSQL	373 secs
Time to learn parameters with TensorFlow*	> 12,000 secs

TensorFlow: 1 epoch; no shuffling; 100K tuple batch; FTRL gradient descent

In-Database versus Out-of-Database Learning

	PostgreSQL+TensorFlow		In-Database (Sept'18)	
	Time	Size (CSV)	Time	Size (CSV)
Input data	–	2.1 GB	–	2.1 GB
Join	217 secs	23 GB	–	–
Export	373 secs	23 GB	–	–
Aggregates	–	–	18 secs	37 KB
GD	$>$ 12K secs	–	0.5 secs	–
Total time	$>$ 12.5K secs		18.5 secs	

In-Database versus Out-of-Database Learning

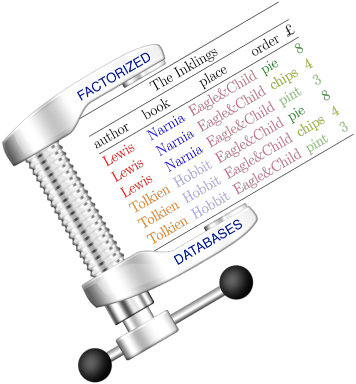
	PostgreSQL+TensorFlow		In-Database (Sept'18)	
	Time	Size (CSV)	Time	Size (CSV)
Input data	–	2.1 GB	–	2.1 GB
Join	217 secs	23 GB	–	–
Export	373 secs	23 GB	–	–
Aggregates	–	–	18 secs	37 KB
GD	$>$ 12K secs	–	0.5 secs	–
Total time	$>$ 12.5K secs		18.5 secs	

$>$ 676$\times$ faster while 600$\times$ more accurate (RMSE on 2% test data) [SOANN19]

TensorFlow trains one model.

In-Database Learning takes 0.5 sec for any extra model over a subset of the given feature set.

Outline of Part 3: Optimization



In-Database Learning

Model Reformulation

Learning under Functional Dependencies

In-Database Linear Algebra

References

Quiz

Learning Regression Models with Least Square Loss

We consider here ridge linear regression

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

- Training dataset $D = Q(I)$, where
  - $Q(\mathbf{X}_F)$ is a feature extraction query, I is the input database
  - D consists of tuples $(\mathbf{x}, y)$ of feature vector $\mathbf{x}$ and response y
- Parameters $\boldsymbol{\theta}$ obtained by minimizing the objective function:

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

Side Note: One-hot Encoding of Categorical Variables

- Continuous variables are mapped to scalars
  - $x_{\texttt{unitsSold}}, x_{\texttt{sales}} \in \mathbb{R}$.

- 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 Optimization to SumProduct 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 (in more detail, Algorithm 1 in [ANNOS18a]):

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

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 $\boldsymbol{\nabla} J(\boldsymbol{\theta})$.

From Optimization to SumProduct FAQs

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

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

$$\boldsymbol{\nabla} J(\boldsymbol{\theta}) = \boldsymbol{\Sigma}\boldsymbol{\theta} - \mathbf{c} + \lambda\boldsymbol{\theta},$$

From Optimization to SumProduct FAQs

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

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

$$\boldsymbol{\nabla} J(\boldsymbol{\theta}) = \boldsymbol{\Sigma}\boldsymbol{\theta} - \mathbf{c} + \lambda\boldsymbol{\theta},$$

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

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

Expressing Σ, $\mathbf{c}$, s_Y using SumProduct FAQs

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

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

$\mathcal{S}(R_k)$ is the set of variables of R_k; $\mathbf{a}_{\mathcal{S}(R_k))}$ is a tuple in relation R_k; $\mathbf{1}_E$ is the Kronecker delta that evaluates to 1 (0) whenever the event E (not) holds.

Expressing Σ, $\mathbf{c}$, s_Y using SQL Queries

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

Expressing Σ, $\mathbf{c}$, s_Y using SQL Queries

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 result of the feature extraction query.

Expressing Σ, $\mathbf{c}$, s_Y using SQL Queries

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 result of the feature extraction query.

Expressing Σ, $\mathbf{c}$, s_Y using SQL Queries

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 result of the feature extraction query.

Expressing Σ, $\mathbf{c}$, s_Y using SQL Queries

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

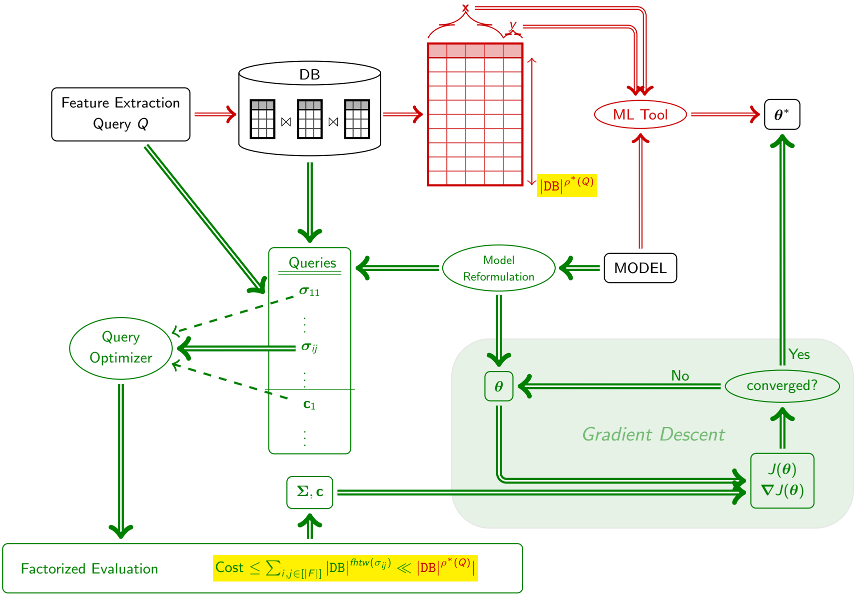
  SELECT x_i, x_j, SUM(1) FROM D GROUP BY x_i, x_j;

where D is the result of the feature extraction query.

This query encoding

- is more compact than one-hot encoding
- can sometimes be computed with lower complexity than D

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



Complexity Analysis: The General Case

Complexity of learning models falls back to factorized computation of aggregates over joins

[BKOZ13,OZ15,SOC16,ANR16]

Let

- $(\mathcal{V}, \mathcal{E})$ = hypergraph of the feature extraction query Q
- $fhtw_{ij}$ = fractional hypertree width of the query that expresses $\boldsymbol{\sigma}_{ij}$ over Q
- $\texttt{DB}$ = input database

The tensors $\boldsymbol{\sigma}_{ij}$ and $\mathbf{c}_j$ can be computed in time [ANNOS18a]

$$O\left(|\mathcal{V}|^2 \cdot |\mathcal{E}| \cdot \sum_{i,j \in [|F|]} \left(|\texttt{DB}|^{fhtw_{ij}} + |\boldsymbol{\sigma}_{ij}|\right) \cdot \log |\texttt{DB}|\right).$$

Complexity Analysis: Continuous Features Only

Recall the complexity in the general case:

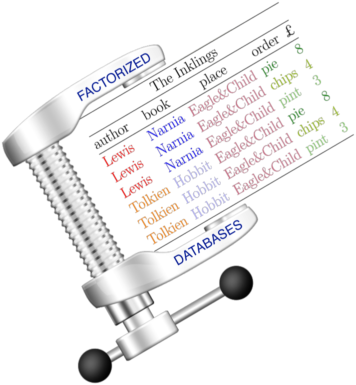
$$O\left(|\mathcal{V}|^2 \cdot |\mathcal{E}| \cdot \sum_{i,j \in [|F|]} \left(|\mathtt{DB}|^{fhtw_{ij}} + |\boldsymbol{\sigma}_{ij}|\right) \cdot \log |\mathtt{DB}|\right).$$

Complexity in case all features are continuous: [SOC16]

$$O(|\mathcal{V}|^2 \cdot |\mathcal{E}| \cdot |F|^2 \cdot |\mathtt{DB}|^{fhtw(Q)} \cdot \log |\mathtt{DB}|).$$

$fhtw_{ij}$ becomes the fractional hypertree width $fhtw$ of Q.

Outline of Part 3: Optimization



In-Database Learning

Model Reformulation

Learning under Functional Dependencies

In-Database Linear Algebra

References

Quiz

Indicator Vectors under 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 $\equiv$ city is either saigon or hanoi

  $x_{\texttt{vietnam}} = 1 \equiv$ 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 $\equiv$ city is either oxford, leeds, or bristol

  $x_{\texttt{england}} = 1 \equiv$ either $x_{\texttt{oxford}} = 1$ or $x_{\texttt{leeds}} = 1$ or $x_{\texttt{bristol}} = 1$

Indicator Vector Mappings

- 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

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

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., $\mathbf{x}_{\texttt{country}} = \mathbf{R}\mathbf{x}_{\texttt{city}} = [1, 0]^\top$.

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

Rewriting the Loss Function

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

$$\sum_{f \in F - \{\texttt{city},\texttt{country}\}} \langle \boldsymbol{\theta}_f, \mathbf{x}_f \rangle + \langle \boldsymbol{\theta}_{\texttt{country}}, \mathbf{x}_{\texttt{country}} \rangle + \langle \boldsymbol{\theta}_{\texttt{city}}, \mathbf{x}_{\texttt{city}} \rangle$$

$$= \sum_{f \in F - \{\texttt{city},\texttt{country}\}} \langle \boldsymbol{\theta}_f, \mathbf{x}_f \rangle + \langle \boldsymbol{\theta}_{\texttt{country}}, \mathbf{R}\mathbf{x}_{\texttt{city}} \rangle + \langle \boldsymbol{\theta}_{\texttt{city}}, \mathbf{x}_{\texttt{city}} \rangle$$

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

Rewriting the Loss Function

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

$$\sum_{f \in F - \{\texttt{city},\texttt{country}\}} \langle \boldsymbol{\theta}_f, \mathbf{x}_f \rangle + \langle \boldsymbol{\theta}_{\texttt{country}}, \mathbf{x}_{\texttt{country}} \rangle + \langle \boldsymbol{\theta}_{\texttt{city}}, \mathbf{x}_{\texttt{city}} \rangle$$

$$= \sum_{f \in F - \{\texttt{city},\texttt{country}\}} \langle \boldsymbol{\theta}_f, \mathbf{x}_f \rangle + \langle \boldsymbol{\theta}_{\texttt{country}}, \mathbf{R}\mathbf{x}_{\texttt{city}} \rangle + \langle \boldsymbol{\theta}_{\texttt{city}}, \mathbf{x}_{\texttt{city}} \rangle$$

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

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

Rewriting the Regularizer (1/2)

Functional dependency: `city` $\rightarrow$ `country`

$\mathbf{x}_{\texttt{country}} = \mathbf{R}\mathbf{x}_{\texttt{city}}$ $\qquad$ $\boldsymbol{\gamma}_{\texttt{city}} = \mathbf{R}^\top \boldsymbol{\theta}_{\texttt{country}} + \boldsymbol{\theta}_{\texttt{city}}$

The penalty term is:

$$\frac{\lambda}{2}\left\|\boldsymbol{\theta}\right\|_2^2 = \frac{\lambda}{2}\Big(\sum_{j \neq \texttt{city}} \|\boldsymbol{\theta}_j\|_2^2 + \left\| \boldsymbol{\gamma}_{\texttt{city}} - \mathbf{R}^\top \boldsymbol{\theta}_{\texttt{country}} \right\|_2^2 + \|\boldsymbol{\theta}_{\texttt{country}}\|_2^2 \Big)$$

We can optimize out $\boldsymbol{\theta}_{\texttt{country}}$ by expressing it in terms of $\gamma_{\texttt{city}}$:

$$\frac{1}{\lambda}\frac{\partial\left(\frac{\lambda}{2}\left\|\boldsymbol{\theta}\right\|_2^2\right)}{\partial \boldsymbol{\theta}_{\texttt{country}}} = \mathbf{R}(\mathbf{R}^\top \boldsymbol{\theta}_{\texttt{country}} - \boldsymbol{\gamma}_{\texttt{city}}) + \boldsymbol{\theta}_{\texttt{country}}$$

By setting this to 0 we obtain $\boldsymbol{\theta}_{\texttt{country}}$ in terms of $\boldsymbol{\gamma}_{\texttt{city}}$ ($\mathbf{I}_{\mathrm{v}}$ is the order-N_{v} identity matrix):

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

Rewriting the Regularizer (2/2)

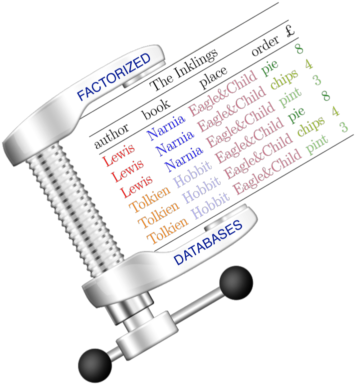
We obtained ($\mathbf{I}_{\mathtt{v}}$ is the order-$N_{\mathtt{v}}$ identity matrix):

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

The penalty term becomes (after several derivation steps)

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

Outline of Part 3: Optimization



- In-Database Learning
- Model Reformulation
- Learning under Functional Dependencies
- **In-Database Linear Algebra**
- References
- Quiz

Linear Algebra is a Key Building Block for ML

Setting: Input matrices defined by queries over relational databases

Matrix $\mathbf{A} = Q(\mathbf{D})$

- Q is a feature extraction query and $\mathbf{D}$ a database
- $\mathbf{A}$ has $m = |Q(\mathbf{D})|$ rows $=$ number of tuples in $Q(\mathbf{D})$
- $\mathbf{A}$ has n columns ($=$ variables in Q) that define features and label
- In our setting: $m \gg n$, i.e., we train in the column space

We should avoid materializing $\mathbf{A}$ whenever possible.

Why?

Examples of linear algebra computation needed for $\frac{\text{ML}}{\text{DB}}$ (assuming $\mathbf{A} \in \mathbb{R}^{m \times n}$):

- Matrix multiplication for learning linear regression models:

$$\mathbf{\Sigma} = \mathbf{A}^{\mathsf{T}}\mathbf{A} \in \mathbb{R}^{n \times n}$$

- Matrix inversion for learning under functional dependencies:

$$(\mathbf{I}_{\texttt{city}} + \mathbf{R}^{\mathsf{T}}\mathbf{R})^{-1}$$

- Matrix factorization
  - QR decomposition

$$\mathbf{A} = \mathbf{Q}\,\mathbf{R}, \text{ where } Q \in \mathbb{R}^{m \times n} \text{ is orthogonal and } \mathbf{R} \in \mathbb{R}^{n \times n} \text{ is upper triangular}$$

  - Rank-k approximation of $\mathbf{A}$

$$\mathbf{A} \approx \mathbf{X}\,\mathbf{Y}, \text{ where } \mathbf{X} \in \mathbb{R}^{m \times k} \text{ and } \mathbf{Y} \in \mathbb{R}^{k \times n}$$

From $\mathbf{A}$ to $\mathbf{\Sigma} = \mathbf{A}^\mathsf{T}\mathbf{A}$

The matrix $\mathbf{\Sigma} = \mathbf{A}^\mathsf{T}\mathbf{A}$ pops up in several ML-relevant computations, eg:

- Least squares problem

  $\boxed{\text{Given } \mathbf{A} \in \mathbb{R}^{m\times n},\ \mathbf{b} \in \mathbb{R}^{m\times 1},\ \text{find } \mathbf{x} \in \mathbb{R}^{n\times 1} \text{ that minimizes } \|\mathbf{A}\mathbf{x} - \mathbf{b}\|^2.}$

  If $\mathbf{A}$ has linearly independent columns, then the unique solution of the least square problem is

  $$\mathbf{x} = (\mathbf{A}^\mathsf{T}\mathbf{A})^{-1}\mathbf{A}^\mathsf{T}\mathbf{b}$$

  $\mathbf{A}^\dagger = (\mathbf{A}^\mathsf{T}\mathbf{A})^{-1}\mathbf{A}^\mathsf{T}$ is called the Moore-Penrose pseudoinverse.

  <u>In-DB setting</u>: The query defines the extended input matrix $[\mathbf{A}\ \mathbf{b}]$.

- Gram-Schmidt process for QR decomposition

Classical QR Factorization

$$\begin{bmatrix} \mathbf{a}_1 & \dots & \mathbf{a}_n \end{bmatrix} = \begin{bmatrix} \mathbf{e}_1 & \dots & \mathbf{e}_n \end{bmatrix} \begin{bmatrix} \langle \mathbf{e}_1, \mathbf{a}_1 \rangle & \langle \mathbf{e}_1, \mathbf{a}_2 \rangle & \dots & \langle \mathbf{e}_1, \mathbf{a}_n \rangle \\ 0 & \langle \mathbf{e}_2, \mathbf{a}_2 \rangle & \dots & \langle \mathbf{e}_2, \mathbf{a}_n \rangle \\ \vdots & \ddots & \dots & \vdots \\ 0 & 0 & \dots & \langle \mathbf{e}_n, \mathbf{a}_n \rangle \end{bmatrix}$$

- $\mathbf{A} \in \mathbb{R}^{m \times n}$. We do not discuss the categorical case here.
- $\mathbf{Q} = [\mathbf{e}_1, \dots, \mathbf{e}_n] \in \mathbb{R}^{m \times n}$ is orthogonal: $\forall i, j \in [n], i \neq j : \langle \mathbf{e}_i, \mathbf{e}_j \rangle = 0$
- $\mathbf{R} \in \mathbb{R}^{n \times n}$ is upper triangular: $\forall i, j \in [n], i > j : \mathbf{R}_{i,j} = 0$
- This is the *thin* QR decomposition.

Applications of QR Factorization

- Solve linear equations $\mathbf{Ax} = \mathbf{b}$ for nonsingular $\mathbf{A} \in \mathbb{R}^{n \times n}$

  1. Decompose $\mathbf{A}$ as $\mathbf{A} = \mathbf{QR}$.

     Then, $\mathbf{QRx} = \mathbf{b} \Rightarrow \mathbf{Q^T QRx} = \mathbf{Q^T b} \Rightarrow \mathbf{Rx} = \mathbf{Q^T b}$

  2. Compute $\mathbf{y} = \mathbf{Q^T b}$

  3. Solve $\mathbf{Rx} = \mathbf{y}$ by back substitution

- Variant: Solve k sets of linear equations with the same $\mathbf{A}$
  Use QR decomposition of $\mathbf{A}$ only once for all k sets!

Applications of QR Factorization

- Pseudo-inverse of a matrix with linearly independent columns

$$
\begin{aligned}
\mathbf{A}^{\dagger} &= (\mathbf{A}^{\mathsf{T}}\mathbf{A})^{-1}\mathbf{A}^{\mathsf{T}} = ((\mathbf{QR})^{\mathsf{T}}(\mathbf{QR}))^{-1}(\mathbf{QR})^{\mathsf{T}} \\
&= (\mathbf{R}^{\mathsf{T}}\mathbf{Q}^{\mathsf{T}}\mathbf{Q}\mathbf{R})^{-1}\mathbf{R}^{\mathsf{T}}\mathbf{Q}^{\mathsf{T}} \\
&= (\mathbf{R}^{\mathsf{T}}\mathbf{R})^{-1}\mathbf{R}^{\mathsf{T}}\mathbf{Q}^{\mathsf{T}} \qquad (\text{since } \mathbf{Q}^{\mathsf{T}}\mathbf{Q} = \mathbf{I}) \\
&= \mathbf{R}^{-1}\mathbf{R}^{-\mathsf{T}}\mathbf{R}^{\mathsf{T}}\mathbf{Q}^{\mathsf{T}} \qquad (\text{since } \mathbf{R} \text{ is nonsingular}) \\
&= \mathbf{R}^{-1}\mathbf{Q}^{\mathsf{T}}
\end{aligned}
$$

- Inverse of a nonsingular square matrix

$$
\mathbf{A}^{-1} = (\mathbf{QR})^{-1} = \mathbf{R}^{-1}\mathbf{Q}^{\mathsf{T}}
$$

- Singular Value Decomposition (SVD) of **A** via Golub-Kahan bidiagonalization of **R**

Applications of QR Factorization

- Least square problem

  Given $\mathbf{A} \in \mathbb{R}^{m \times n}$, $\mathbf{b} \in \mathbb{R}^{m \times 1}$, find $\mathbf{x} \in \mathbb{R}^{n \times 1}$ that minimizes $\|\mathbf{Ax} - \mathbf{b}\|^2$.

  If $\mathbf{A}$ has linearly independent columns, then the unique solution of the least square problem is

  $$\hat{\mathbf{x}} = (\mathbf{A}^\mathsf{T}\mathbf{A})^{-1}\mathbf{A}^\mathsf{T}\mathbf{b} = \mathbf{A}^\dagger\mathbf{b} = \mathbf{R}^{-1}\mathbf{Q}^\mathsf{T}\mathbf{b}$$

  For $m > n$ this is an overdetermined system of linear equations.

  <u>In-DB setting</u>: The query defines the extended input matrix $[\mathbf{A}\ \mathbf{b}]$.

QR Factorization using the Gram-Schmidt Process

Project the vector $\mathbf{a}_k$ orthogonally onto the line spanned by vector $\mathbf{u}_j$:

$$\mathrm{proj}_{\mathbf{u}_j}\mathbf{a}_k = \frac{\langle \mathbf{u}_j, \mathbf{a}_k \rangle}{\langle \mathbf{u}_j, \mathbf{u}_j \rangle}\mathbf{u}_j.$$

Gram-Schmidt orthogonalization:

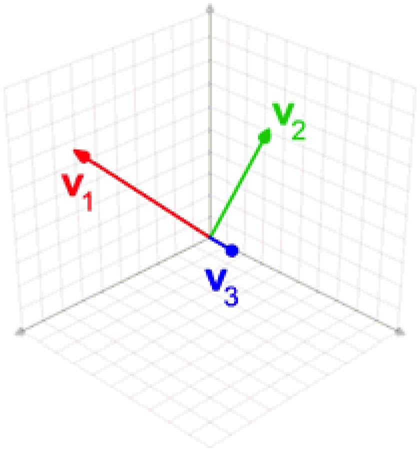
$$\forall k \in [n] : \mathbf{u}_k = \mathbf{a}_k - \sum_{j \in [k-1]} \mathrm{proj}_{\mathbf{u}_j}\mathbf{a}_k = \mathbf{a}_k - \sum_{j \in [k-1]} \frac{\langle \mathbf{u}_j, \mathbf{a}_k \rangle}{\langle \mathbf{u}_j, \mathbf{u}_j \rangle}\mathbf{u}_j.$$

The vectors in the orthogonal matrix $\mathbf{Q}$ are normalized:

$$Q = \left[\mathbf{e}_1 = \tfrac{\mathbf{u}_1}{\|\mathbf{u}_1\|}, \quad \ldots, \quad \mathbf{e}_n = \tfrac{\mathbf{u}_n}{\|\mathbf{u}_n\|}\right]$$

Example: QR Factorization using Gram-Schmidt

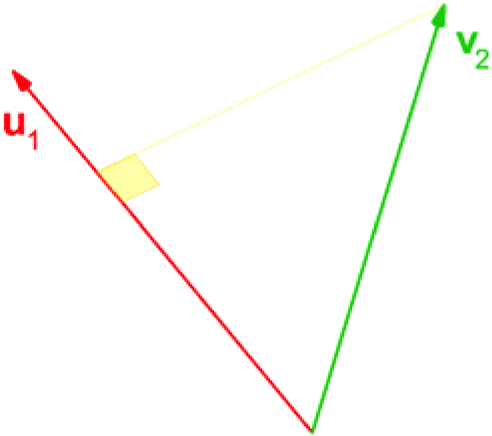
Given $\mathbf{A} = [\mathbf{v}_1, \mathbf{v}_2, \mathbf{v}_3]$. Task: Compute $\mathbf{Q} = [\mathbf{e}_1 = \frac{\mathbf{u}_1}{\|\mathbf{u}_1\|}, \mathbf{e}_2 = \frac{\mathbf{u}_2}{\|\mathbf{u}_2\|}, \mathbf{e}_3 = \frac{\mathbf{u}_3}{\|\mathbf{u}_3\|}]$.



Source: Wikipedia

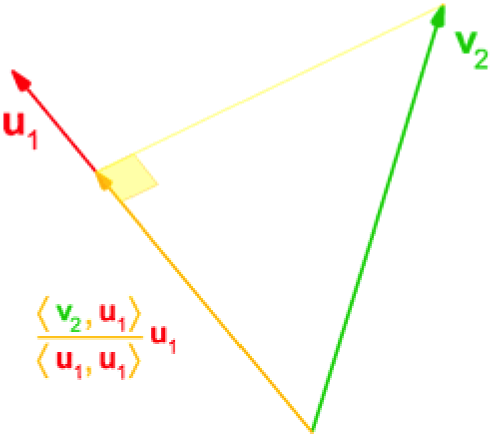
Example: QR Factorization using Gram-Schmidt

Given $\mathbf{A} = [\mathbf{v}_1, \mathbf{v}_2, \mathbf{v}_3]$. Task: Compute $\mathbf{Q} = [\mathbf{e}_1 = \frac{\mathbf{u}_1}{\|\mathbf{u}_1\|}, \mathbf{e}_2 = \frac{\mathbf{u}_2}{\|\mathbf{u}_2\|}, \mathbf{e}_3 = \frac{\mathbf{u}_3}{\|\mathbf{u}_3\|}]$.



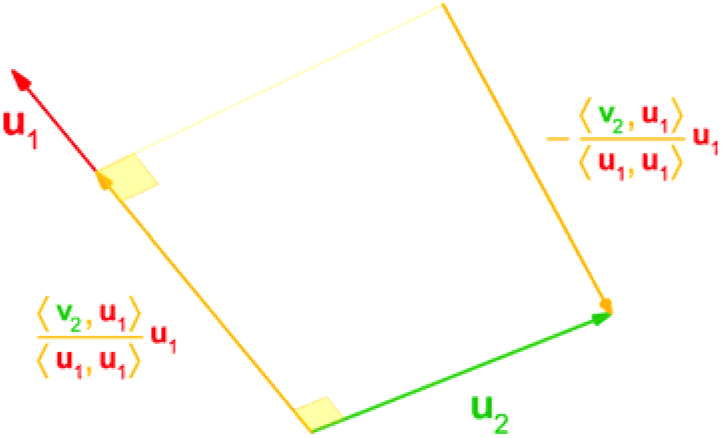
Example: QR Factorization using Gram-Schmidt

Given $\mathbf{A} = [\mathbf{v}_1, \mathbf{v}_2, \mathbf{v}_3]$. Task: Compute $\mathbf{Q} = [\mathbf{e}_1 = \frac{\mathbf{u}_1}{\|\mathbf{u}_1\|}, \mathbf{e}_2 = \frac{\mathbf{u}_2}{\|\mathbf{u}_2\|}, \mathbf{e}_3 = \frac{\mathbf{u}_3}{\|\mathbf{u}_3\|}]$.



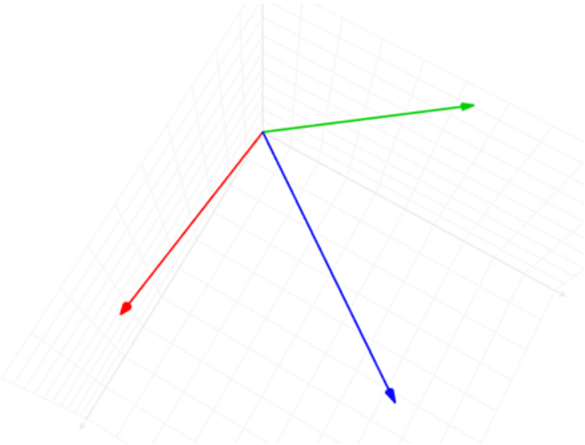
Example: QR Factorization using Gram-Schmidt

Given $\mathbf{A} = [\mathbf{v}_1, \mathbf{v}_2, \mathbf{v}_3]$. Task: Compute $\mathbf{Q} = [\mathbf{e}_1 = \frac{\mathbf{u}_1}{\|\mathbf{u}_1\|}, \mathbf{e}_2 = \frac{\mathbf{u}_2}{\|\mathbf{u}_2\|}, \mathbf{e}_3 = \frac{\mathbf{u}_3}{\|\mathbf{u}_3\|}]$.



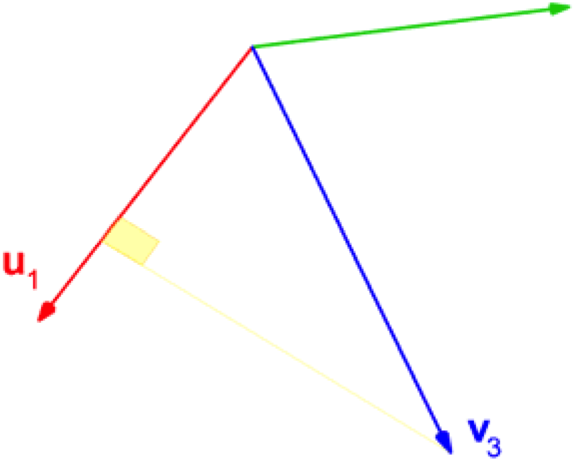
Example: QR Factorization using Gram-Schmidt

Given $\mathbf{A} = [\mathbf{v}_1, \mathbf{v}_2, \mathbf{v}_3]$. Task: Compute $\mathbf{Q} = [\mathbf{e}_1 = \frac{\mathbf{u}_1}{\|\mathbf{u}_1\|}, \mathbf{e}_2 = \frac{\mathbf{u}_2}{\|\mathbf{u}_2\|}, \mathbf{e}_3 = \frac{\mathbf{u}_3}{\|\mathbf{u}_3\|}]$.



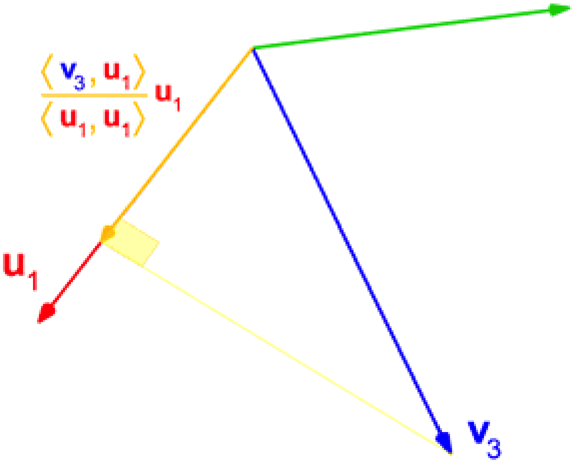
Example: QR Factorization using Gram-Schmidt

Given $\mathbf{A} = [\mathbf{v}_1, \mathbf{v}_2, \mathbf{v}_3]$. Task: Compute $\mathbf{Q} = [\mathbf{e}_1 = \frac{\mathbf{u}_1}{\|\mathbf{u}_1\|}, \mathbf{e}_2 = \frac{\mathbf{u}_2}{\|\mathbf{u}_2\|}, \mathbf{e}_3 = \frac{\mathbf{u}_3}{\|\mathbf{u}_3\|}]$.



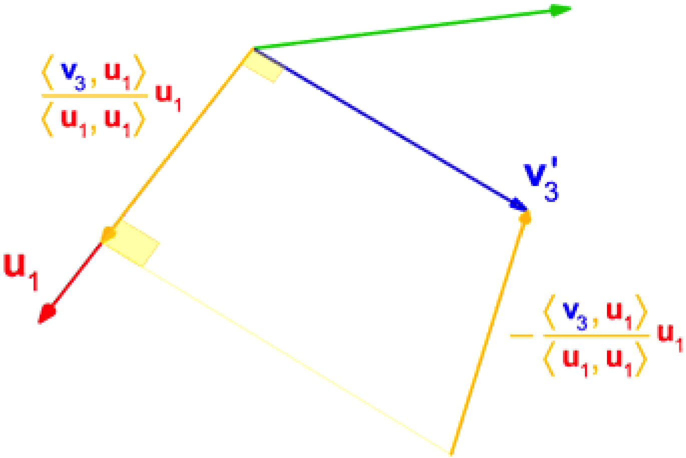
Example: QR Factorization using Gram-Schmidt

Given $\mathbf{A} = [\mathbf{v}_1, \mathbf{v}_2, \mathbf{v}_3]$. Task: Compute $\mathbf{Q} = [\mathbf{e}_1 = \frac{\mathbf{u}_1}{\|\mathbf{u}_1\|}, \mathbf{e}_2 = \frac{\mathbf{u}_2}{\|\mathbf{u}_2\|}, \mathbf{e}_3 = \frac{\mathbf{u}_3}{\|\mathbf{u}_3\|}]$.



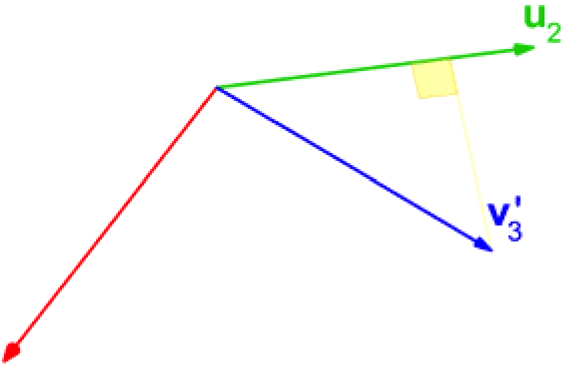
Example: QR Factorization using Gram-Schmidt

Given $\mathbf{A} = [\mathbf{v}_1, \mathbf{v}_2, \mathbf{v}_3]$. Task: Compute $\mathbf{Q} = [\mathbf{e}_1 = \frac{\mathbf{u}_1}{\|\mathbf{u}_1\|}, \mathbf{e}_2 = \frac{\mathbf{u}_2}{\|\mathbf{u}_2\|}, \mathbf{e}_3 = \frac{\mathbf{u}_3}{\|\mathbf{u}_3\|}]$.



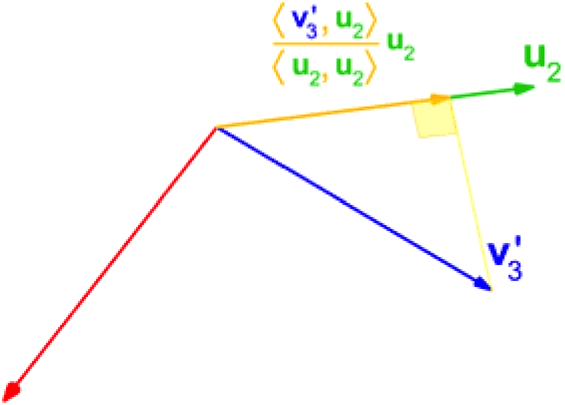
Example: QR Factorization using Gram-Schmidt

Given $\mathbf{A} = [\mathbf{v}_1, \mathbf{v}_2, \mathbf{v}_3]$. Task: Compute $\mathbf{Q} = [\mathbf{e}_1 = \frac{\mathbf{u}_1}{\|\mathbf{u}_1\|}, \mathbf{e}_2 = \frac{\mathbf{u}_2}{\|\mathbf{u}_2\|}, \mathbf{e}_3 = \frac{\mathbf{u}_3}{\|\mathbf{u}_3\|}]$.



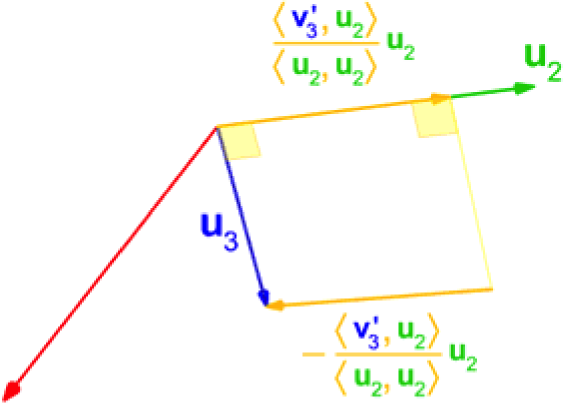
Example: QR Factorization using Gram-Schmidt

Given $\mathbf{A} = [\mathbf{v}_1, \mathbf{v}_2, \mathbf{v}_3]$. Task: Compute $\mathbf{Q} = [\mathbf{e}_1 = \frac{\mathbf{u}_1}{\|\mathbf{u}_1\|}, \mathbf{e}_2 = \frac{\mathbf{u}_2}{\|\mathbf{u}_2\|}, \mathbf{e}_3 = \frac{\mathbf{u}_3}{\|\mathbf{u}_3\|}]$.



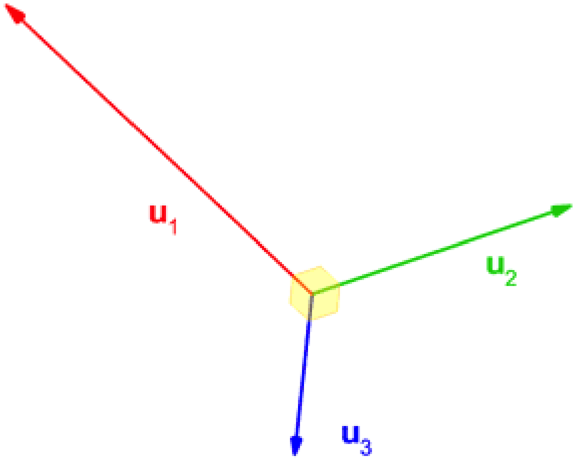
Example: QR Factorization using Gram-Schmidt

Given $\mathbf{A} = [\mathbf{v}_1, \mathbf{v}_2, \mathbf{v}_3]$. Task: Compute $\mathbf{Q} = [\mathbf{e}_1 = \frac{\mathbf{u}_1}{\|\mathbf{u}_1\|}, \mathbf{e}_2 = \frac{\mathbf{u}_2}{\|\mathbf{u}_2\|}, \mathbf{e}_3 = \frac{\mathbf{u}_3}{\|\mathbf{u}_3\|}]$.



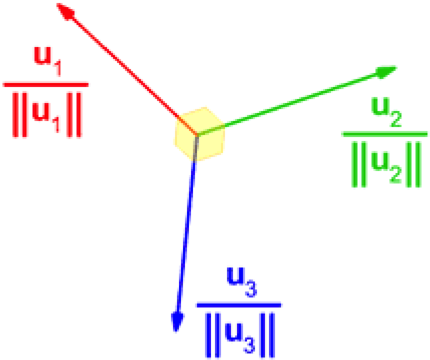
Example: QR Factorization using Gram-Schmidt

Given $\mathbf{A} = [\mathbf{v}_1, \mathbf{v}_2, \mathbf{v}_3]$. Task: Compute $\mathbf{Q} = [\mathbf{e}_1 = \frac{\mathbf{u}_1}{\|\mathbf{u}_1\|}, \mathbf{e}_2 = \frac{\mathbf{u}_2}{\|\mathbf{u}_2\|}, \mathbf{e}_3 = \frac{\mathbf{u}_3}{\|\mathbf{u}_3\|}]$.



Example: QR Factorization using Gram-Schmidt

Given $\mathbf{A} = [\mathbf{v}_1, \mathbf{v}_2, \mathbf{v}_3]$. Task: Compute $\mathbf{Q} = [\mathbf{e}_1 = \frac{\mathbf{u}_1}{\|\mathbf{u}_1\|}, \mathbf{e}_2 = \frac{\mathbf{u}_2}{\|\mathbf{u}_2\|}, \mathbf{e}_3 = \frac{\mathbf{u}_3}{\|\mathbf{u}_3\|}]$.



How to Lower the Complexity of Gram-Schmidt?

Challenges:

- How does not materializing $\mathbf{A}$ help? $\mathbf{Q}$ has the same dimension as $\mathbf{A}$!

- The Gram-Schmidt process is inherently sequential and not parallelizable

  Computing $\mathbf{u}_k$ requires the computation of $\mathbf{u}_1, \ldots, \mathbf{u}_{k-1}$ in $\mathbf{Q}$.

How to Lower the Complexity of Gram-Schmidt?

Challenges:

- How does not materializing $\mathbf{A}$ help? $\mathbf{Q}$ has the same dimension as $\mathbf{A}$!

  Trick 1: Only use $\mathbf{R}$ and do not require full $\mathbf{Q}$ in subsequent computation

- The Gram-Schmidt process is inherently sequential and not parallelizable

  Computing $\mathbf{u}_k$ requires the computation of $\mathbf{u}_1, \ldots, \mathbf{u}_{k-1}$ in $\mathbf{Q}$.

  Trick 2: Rewrite $\mathbf{u}_k$ to refer to columns in $\mathbf{A}$ instead of $\mathbf{Q}$

Factorizing the QR Factorization

Express each vector $\mathbf{u}_j$ as a linear combination of vectors $\mathbf{a}_1, \ldots, \mathbf{a}_j$ in $\mathbf{A}$:

$$\begin{bmatrix} \mathbf{u}_1 & \ldots & \mathbf{u}_n \end{bmatrix} = - \begin{bmatrix} \mathbf{a}_1 & \ldots & \mathbf{a}_n \end{bmatrix} \begin{bmatrix} c_{1,1} & c_{1,2} & \ldots & c_{1,n} \\ 0 & c_{2,2} & \ldots & c_{2,n} \\ \vdots & \ddots & \ddots & \vdots \\ 0 & 0 & 0 & c_{n,n} \end{bmatrix}$$

That is, $\mathbf{u}_k = -\sum_{j \in [k]} c_{j,k} \mathbf{a}_j$. The coefficients $c_{j,k}$ are:

$$\forall j \in [k-1] : c_{j,k} = \sum_{i \in [j,k-1]} \frac{u_{i,k}}{d_i} \cdot c_{j,i} \qquad c_{k,k} = -1$$

$$\forall j \in [k-1] : u_{j,k} = \sum_{l \in [j]} c_{l,j} \cdot \langle \mathbf{a}_l, \mathbf{a}_k \rangle \qquad \forall i \in [n] : d_i = \sum_{l \in [i]} \sum_{p \in [i]} c_{l,i} \cdot c_{p,i} \cdot \langle \mathbf{a}_l, \mathbf{a}_p \rangle$$

Factorizing the QR Factorization

Express each vector $\mathbf{u}_j$ as a linear combination of vectors $\mathbf{a}_1, \ldots, \mathbf{a}_j$ in $\mathbf{A}$:

$$\begin{bmatrix} \mathbf{u}_1 & \ldots & \mathbf{u}_n \end{bmatrix} = - \begin{bmatrix} \mathbf{a}_1 & \ldots & \mathbf{a}_n \end{bmatrix} \begin{bmatrix} c_{1,1} & c_{1,2} & \ldots & c_{1,n} \\ 0 & c_{2,2} & \ldots & c_{2,n} \\ \vdots & \ddots & \ddots & \vdots \\ 0 & 0 & 0 & c_{n,n} \end{bmatrix}$$

That is, $\mathbf{u}_k = -\sum_{j \in [k]} c_{j,k} \mathbf{a}_j$. The coefficients $c_{j,k}$ are:

$$\forall j \in [k-1] : c_{j,k} = \sum_{i \in [j,k-1]} \frac{u_{i,k}}{d_i} \cdot c_{j,i} \qquad c_{k,k} = -1$$

$$\forall j \in [k-1] : u_{j,k} = \sum_{l \in [j]} c_{l,j} \cdot \langle \mathbf{a}_l, \mathbf{a}_k \rangle \qquad \forall i \in [n] : d_i = \sum_{l \in [i]} \sum_{p \in [i]} c_{l,i} \cdot c_{p,i} \cdot \langle \mathbf{a}_l, \mathbf{a}_p \rangle$$

$\boxed{\text{The coefficients are defined by FAQs over the entries in } \mathbf{\Sigma} = \mathbf{A}^{\mathsf{T}}\mathbf{A}}$

Expressing $\mathbf{Q}$

$$\mathbf{Q} = \mathbf{AC}, \text{ where}$$

$$\|\mathbf{u}_k\| = \sqrt{\langle \mathbf{u}_k, \mathbf{u}_k \rangle} = \sqrt{\sum_{l \in [k]} \sum_{p \in [k]} c_{l,k} \cdot c_{p,k} \cdot \langle \mathbf{a}_l, \mathbf{a}_p \rangle} = \sqrt{d_k}$$

$$\mathbf{C} = \begin{bmatrix} \mathbf{c}_1 & \dots & \mathbf{c}_n \end{bmatrix} = \begin{bmatrix} \frac{c_{1,1}}{\sqrt{d_1}} & \frac{c_{1,2}}{\sqrt{d_1}} & \dots & \frac{c_{1,n}}{\sqrt{d_1}} \\ 0 & \frac{c_{2,2}}{\sqrt{d_2}} & \dots & \frac{c_{2,n}}{\sqrt{d_2}} \\ \vdots & \ddots & \ddots & \vdots \\ 0 & 0 & 0 & \frac{c_{n,n}}{\sqrt{d_n}} \end{bmatrix}$$

Expressing $\mathbf{R}$

Entries in the upper triangular $\mathbf{R}$ are $\langle \mathbf{e}_i, \mathbf{a}_j \rangle = \frac{\langle \mathbf{u}_i, \mathbf{a}_j \rangle}{\sqrt{d_i}} = \langle \mathbf{A}\mathbf{c}_i, \mathbf{a}_j \rangle, \forall i \leq j$. Then,

$$\mathbf{R} = \begin{bmatrix} \left\langle \mathbf{c}_1, \mathbf{A}^\mathsf{T}\mathbf{a}_1 \right\rangle & \left\langle \mathbf{c}_1, \mathbf{A}^\mathsf{T}\mathbf{a}_2 \right\rangle & \dots & \left\langle \mathbf{c}_1, \mathbf{A}^\mathsf{T}\mathbf{a}_n \right\rangle \\ 0 & \left\langle \mathbf{c}_2, \mathbf{A}^\mathsf{T}\mathbf{a}_2 \right\rangle & \dots & \left\langle \mathbf{c}_2, \mathbf{A}^\mathsf{T}\mathbf{a}_n \right\rangle \\ \vdots & \ddots & \dots & \vdots \\ 0 & 0 & \dots & \left\langle \mathbf{c}_n, \mathbf{A}^\mathsf{T}\mathbf{a}_n \right\rangle \end{bmatrix}$$

$\boxed{\text{The entries in } \mathbf{R} \text{ are defined by FAQs over } \mathbf{\Sigma} = \mathbf{A}^\mathsf{T}\mathbf{A} = [A^\mathsf{T}\mathbf{a}_1, \dots, A^\mathsf{T}\mathbf{a}_n]}$

Revisiting The Least Squares Problem

Given $\mathbf{A} \in \mathbb{R}^{m \times n}$, $\mathbf{b} \in \mathbb{R}^{m \times 1}$, find $\mathbf{x} \in \mathbb{R}^{n \times 1}$ that minimizes $\|\mathbf{Ax} - \mathbf{b}\|^2$.

In-DB setting: The query defines the extended input matrix $[\mathbf{A}\ \mathbf{b}]$.

Solution $\hat{\mathbf{x}} = \mathbf{R}^{-1}\mathbf{Q}^{\mathsf{T}}\mathbf{b}$ requires:

- The inverse $\mathbf{R}^{-1}$ of the upper triangular matrix $\mathbf{R}$; or back substitution
- The vector $\mathbf{Q}^{\mathsf{T}}\mathbf{b}$ computable directly over the input data

$$\mathbf{Q}^{\mathsf{T}}\mathbf{b} = (\mathbf{AC})^{\mathsf{T}}\mathbf{b} = \mathbf{C}^{\mathsf{T}}\mathbf{A}^{\mathsf{T}}\mathbf{b} = \mathbf{C}^{\mathsf{T}} \begin{bmatrix} \langle \mathbf{a}_1, \mathbf{b} \rangle \\ \vdots \\ \langle \mathbf{a}_n, \mathbf{b} \rangle \end{bmatrix}$$

$\boxed{\text{The dot products } \langle \mathbf{a}_j, \mathbf{b} \rangle \text{ are FAQs computable without } \mathbf{A}!}$

Computing Coefficient Matrix **C** without **A**

$\boxed{\text{Data complexity of } \mathbf{C} \text{ is the same as of } \mathbf{\Sigma}}$

Given $\mathbf{\Sigma}$, $O(n^3)$ time to compute matrix **C** and vector **d**

- There are $n(n-2)/2$ entries in coefficient matrix **C** that are not 0 and -1
  - Each of them takes $3n$ arithmetic operations
- There are n entries in the vector **d**
  - Entry d_i takes i^2 arithmetic operations

Computing Coefficient Matrix C without A

Data complexity of $\mathbf{C}$ is the same as of $\mathbf{\Sigma}$

Given $\mathbf{\Sigma}$, $O(n^3)$ time to compute matrix $\mathbf{C}$ and vector $\mathbf{d}$

- There are $n(n-2)/2$ entries in coefficient matrix $\mathbf{C}$ that are not 0 and -1
  - Each of them takes $3n$ arithmetic operations
- There are n entries in the vector $\mathbf{d}$
  - Entry d_i takes i^2 arithmetic operations

Computing sparse-encoded $\mathbf{C}$ from sparse-encoded $\mathbf{\Sigma}$ a bit tricky

- Same complexity overhead as for $\mathbf{\Sigma}$
- Nicely parallelizable, accounting for the dependencies between entries in $\mathbf{C}$

Our Journey So Far with QR Factorization

F-GS system on top of LMFAO for QR factorization of matrices defined over database joins

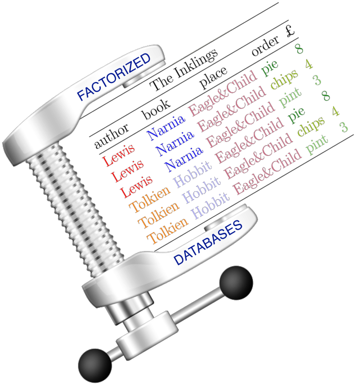
- 33 numerical + 3,702 categorical features
  - $\mathbf{\Sigma}$ computed on one core by LMFAO in 18 sec
  - **C**, **d**, and **R** (and Linear Regression on top) computed on one core by F-GS in 18 sec
  - F-GS on 8 cores is $3\times$ faster than on one core
  - Any of **C**, **d**, and **R** cannot be computed by LAPACK
- 33 numerical + 55 categorical features
  - $\mathbf{\Sigma}$ computed on one core by LMFAO in 6.4 sec
  - **C**, **d**, and **R** (and Linear Regression on top) computed one one core by F-GS in 1 sec
  - **R** can be computed by LAPACK on one core in 313 sec

    It also needs to read in the data: $+ \approx 70$ sec
  - LAPACK on 8 cores is $7\times$ faster than on one core

Retailer dataset (86M), acyclic natural join of 5 relations, 26x compression by factorization;

Intel i7-4770 3.40GHz/64bit/32GB, Linux 3.13.0, g++4.8.4, libblas3 1.2 (one core), OpenBLAS

Outline of Part 3: Optimization



In-Database Learning

Model Reformulation

Learning under Functional Dependencies

In-Database Linear Algebra

References

Quiz

Beyond Linear Regression

This approach has been or is applied to a host of ML models:

- Polynomial regression (done)
- Factorization machines (done)
- Decision trees (done)
- Principal component analysis (done)
- Generalised low-rank models (on-going)
- Sum-product networks (on-going)
- K-means & k-median clustering (on-going)
- Gradient boosting decision trees (on-going)
- Random forests (on-going)

Some models seem inherently hard for in-db learning

- Logistic regression (unclear)

Beyond Polynomial Loss

There are common loss functions that are:

- Convex,
- Non-differentiable, but
- Admit subgradients with respect to model parameters.

Examples:

- Hinge (used for linear SVM, ReLU)

$$J(\theta) = \max(0, 1 - y \cdot \langle \theta, \mathbf{x} \rangle)$$

- Huber, ℓ_1, scalene, fractional, ordinal, interval

Their subgradients may not be as factorisable as the gradient of the square loss.

References

SOC16 **Learning Linear Regression Models over Factorized Joins.**
Schleich, Olteanu, Ciucanu. In SIGMOD 2016.
http://dl.acm.org/citation.cfm?doid=2882903.2882939

A17 **Research Directions for Principles of Data Management (Dagstuhl Perspectives Workshop 16151).**
Abiteboul et al. In SIGMOD Rec. 2017.
https://arxiv.org/pdf/1701.09007.pdf

KBY17 **Data Management in Machine Learning: Challenges, Techniques, and Systems.**
Kumar, Boehm, Yang. In SIGMOD 2017, Tutorial.
https://www.youtube.com/watch?v=U8J0Dd_Z5wo

PRWZ17 **Data Management Challenges in Production Machine Learning.**
Polyzotis, Roy, Whang, Zinkevich. In SIGMOD 2017, Tutorial.
http://dl.acm.org/citation.cfm?doid=3035918.3054782

ANNOS18a **In-Database Learning with Sparse Tensors.**
Abo Khamis, Ngo, Nguyen, Olteanu, Schleich. In PODS 2018.
https://arxiv.org/abs/1703.04780 (extended version)

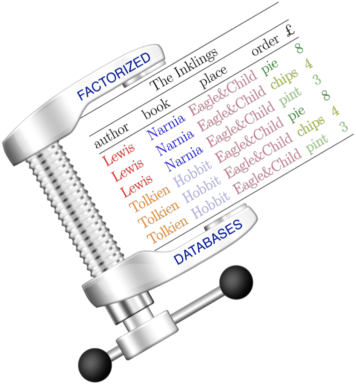
ANNOS18b **AC/DC: In-Database Learning Thunderstruck.**
Abo Khamis, Ngo, Nguyen, Olteanu, Schleich. In DEEM@SIGMOD 2018.
https://arxiv.org/abs/1803.07480

References

NO18 **Incremental View Maintenance with Triple Lock Factorisation Benefits.**
Nikolic, Olteanu. In SIGMOD 2018.
https://arxiv.org/abs/1703.07484

SOANN19 Under submission.

Outline of Part 3: Optimization



In-Database Learning

Model Reformulation

Learning under Functional Dependencies

In-Database Linear Algebra

References

Quiz

QUIZ on Optimization

Assume that the natural join of the following relations provides the features we use to predict revenue:

```
Sales(store_id, product_id, quantity, revenue),
Product(product_id, color),
Store(store_id, distance_city_center).
```

Variables `revenue`, `quantity`, and `distance_city_center` stand for continuous features, while `product_id` and `color` for categorical features.

1. Give the FAQs required to compute the gradient of the squares loss function for learning a ridge linear regression models with the above features. Give the same for a polynomial regression model of degree two.
2. We know that `product_id` functionally determines `color`. Give a rewriting of the objective function that exploits the functional dependency.
3. The FAQs require the computation of a lot of common sub-problems. Can you think of ways to share as much computation as possible?