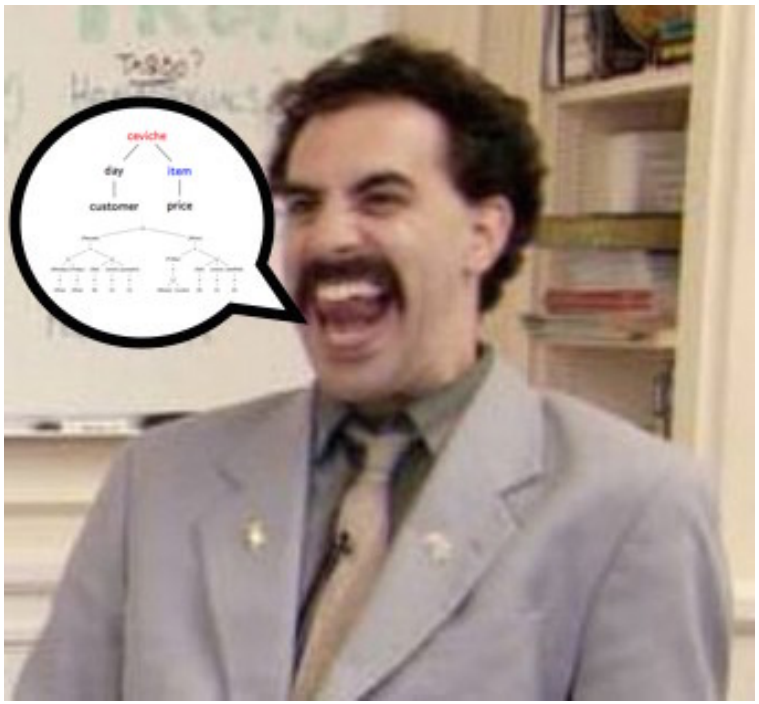
Learning Linear Regression Models over Factorised Joins

(or how to make glorious benefit to regression tasks)



Maximilian Schleich

MSc Thesis, University of Oxford

April 16, 2016

Vienna Center for Logic and Algorithms

Problem Statement and Motivation

- Use very recent results in database research to provide **runtime guarantees** for machine learning algorithms.
- Particular focus on **learning over joins**, a very common scenario in industry.
- Main observation is that the **join contains a lot of redundancy**, which can be avoided by **factorising** the join representation.
- Learn machine learning models in **linear time over much more succinct representation** of the join result, which leads to **orders of magnitude performance improvements** on real datasets.

Outline

Factorised Databases by Example

Orders (O for short)

customer	day	dish
Elise	Monday	dish1
Elise	Friday	dish1
Lukas	Friday	dish2
Reiner	Friday	dish2

Dish (D for short)

dish	item
dish1	schnitzel
dish1	potatoes
dish1	salad
dish2	sausage
dish2	potatoes
dish2	salad

Items (I for short)

item	price
schnitzel	6
potatoes	1
salad	1
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	dish1	schnitzel	6
Elise	Monday	dish1	potatoes	1
Elise	Monday	dish1	salad	1
Elise	Friday	dish1	schnitzel	6
Elise	Friday	dish1	potatoes	1
Elise	Friday	dish1	salad	1
...	...	...	...	...

Factorised Databases by Example

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

customer	day	dish	item	price
Elise	Monday	dish1	schnitzel	6
Elise	Monday	dish1	potatoes	1
Elise	Monday	dish1	salad	1
Elise	Friday	dish1	schnitzel	6
Elise	Friday	dish1	potatoes	1
Elise	Friday	dish1	salad	1
...	...	...	...	...

A *flat* relational algebra expression encoding the above query result is:

$$
\begin{array}{lllllllllll}
\langle \mathit{Elise} \rangle & \times & \langle \mathit{Monday} \rangle & \times & \langle \mathit{dish}1 \rangle & \times & \langle \text{schnitzel} \rangle & \times & \langle 6 \rangle & \cup \\
\langle \mathit{Elise} \rangle & \times & \langle \mathit{Monday} \rangle & \times & \langle \mathit{dish}1 \rangle & \times & \langle \text{potatoes} \rangle & \times & \langle 1 \rangle & \cup \\
\langle \mathit{Elise} \rangle & \times & \langle \mathit{Monday} \rangle & \times & \langle \mathit{dish}1 \rangle & \times & \langle \text{salad} \rangle & \times & \langle 1 \rangle & \cup \\
\langle \mathit{Elise} \rangle & \times & \langle \mathit{Friday} \rangle & \times & \langle \mathit{dish}1 \rangle & \times & \langle \text{schnitzel} \rangle & \times & \langle 6 \rangle & \cup \\
\langle \mathit{Elise} \rangle & \times & \langle \mathit{Friday} \rangle & \times & \langle \mathit{dish}1 \rangle & \times & \langle \text{potatoes} \rangle & \times & \langle 1 \rangle & \cup \\
\langle \mathit{Elise} \rangle & \times & \langle \mathit{Friday} \rangle & \times & \langle \mathit{dish}1 \rangle & \times & \langle \text{salad} \rangle & \times & \langle 1 \rangle & \cup \ldots
\end{array}
$$

It uses relational product ($\times$), union ($\cup$), and singleton relations (e.g., $\langle 1 \rangle$).

- The attribute names are not shown to avoid clutter.

Factorised Databases by Example

The previous relational expression entails lots of redundancy due to the joins:

$$
\begin{array}{lllllllllll}
\langle \mathit{Elise} \rangle & \times & \langle \mathit{Monday} \rangle & \times & \langle \mathit{dish1} \rangle & \times & \langle \text{schnitzel} \rangle & \times & \langle 6 \rangle & \cup \\
\langle \mathit{Elise} \rangle & \times & \langle \mathit{Monday} \rangle & \times & \langle \mathit{dish1} \rangle & \times & \langle \text{potatoes} \rangle & \times & \langle 1 \rangle & \cup \\
\langle \mathit{Elise} \rangle & \times & \langle \mathit{Monday} \rangle & \times & \langle \mathit{dish1} \rangle & \times & \langle \text{salad} \rangle & \times & \langle 1 \rangle & \cup \\
\langle \mathit{Elise} \rangle & \times & \langle \mathit{Friday} \rangle & \times & \langle \mathit{dish1} \rangle & \times & \langle \text{schnitzel} \rangle & \times & \langle 6 \rangle & \cup \\
\langle \mathit{Elise} \rangle & \times & \langle \mathit{Friday} \rangle & \times & \langle \mathit{dish1} \rangle & \times & \langle \text{potatoes} \rangle & \times & \langle 1 \rangle & \cup \\
\langle \mathit{Elise} \rangle & \times & \langle \mathit{Friday} \rangle & \times & \langle \mathit{dish1} \rangle & \times & \langle \text{salad} \rangle & \times & \langle 1 \rangle & \cup \ldots
\end{array}
$$

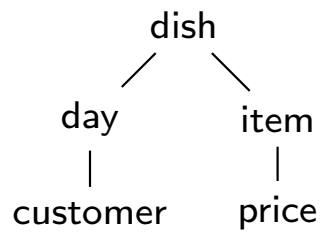
Factorised Databases by Example

The previous relational expression entails lots of redundancy due to the joins:

$$
\begin{array}{lllllllllll}
\langle \mathit{Elise} \rangle & \times & \langle \mathit{Monday} \rangle & \times & \langle \mathit{dish1} \rangle & \times & \langle \mathrm{schnitzel} \rangle & \times & \langle 6 \rangle & \cup \\
\langle \mathit{Elise} \rangle & \times & \langle \mathit{Monday} \rangle & \times & \langle \mathit{dish1} \rangle & \times & \langle \mathrm{potatoes} \rangle & \times & \langle 1 \rangle & \cup \\
\langle \mathit{Elise} \rangle & \times & \langle \mathit{Monday} \rangle & \times & \langle \mathit{dish1} \rangle & \times & \langle \mathrm{salad} \rangle & \times & \langle 1 \rangle & \cup \\
\langle \mathit{Elise} \rangle & \times & \langle \mathit{Friday} \rangle & \times & \langle \mathit{dish1} \rangle & \times & \langle \mathrm{schnitzel} \rangle & \times & \langle 6 \rangle & \cup \\
\langle \mathit{Elise} \rangle & \times & \langle \mathit{Friday} \rangle & \times & \langle \mathit{dish1} \rangle & \times & \langle \mathrm{potatoes} \rangle & \times & \langle 1 \rangle & \cup \\
\langle \mathit{Elise} \rangle & \times & \langle \mathit{Friday} \rangle & \times & \langle \mathit{dish1} \rangle & \times & \langle \mathrm{salad} \rangle & \times & \langle 1 \rangle & \cup \ldots
\end{array}
$$

We can *factorise* the expression following possible join structures, e.g.,:

$$
\begin{aligned}
&\langle \mathit{dish1} \rangle \times (\langle \mathit{Monday} \rangle \times \langle \mathit{Elise} \rangle \cup \langle \mathit{Friday} \rangle \times \langle \mathit{Elise} \rangle) \\
&\qquad \times (\langle \mathit{schnitzel} \rangle \times \langle 6 \rangle \cup \langle \mathit{potatoes} \rangle \times \langle 1 \rangle \cup \langle \mathit{salad} \rangle \times \langle 1 \rangle) \\
&\cup \langle \mathit{dish2} \rangle \times \langle \mathit{Friday} \rangle \times (\langle \mathit{Reiner} \rangle \cup \langle \mathit{Lukas} \rangle) \\
&\qquad \times (\langle \mathit{sausage} \rangle \times \langle 4 \rangle \cup \langle \mathit{potatoes} \rangle \times \langle 1 \rangle \cup \langle \mathit{salad} \rangle \times \langle 1 \rangle)
\end{aligned}
$$



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

Outline

Size Bounds for Flat and Factorised Join Results

Given a join query Q, for any database $\mathbf{D}$, the join result $Q(\mathbf{D})$ admits

- a flat representation of size $O(|\mathbf{D}|^{\rho^*(Q)})$. [AGM'08]
- a factorisation *with caching* of size $O(|\mathbf{D}|^{fhtw(Q)})$. [OZ'15]

- $fhtw(Q)$ is the fractional hypertree width of Q
- $\rho^*(Q)$ is the fractional edge cover number of Q

Size Bounds for Flat and Factorised Join Results

Given a join query Q, for any database $\mathbf{D}$, the join result $Q(\mathbf{D})$ admits

- a flat representation of size $O(|\mathbf{D}|^{\rho^*(Q)})$. [AGM'08]
- a factorisation *with caching* of size $O(|\mathbf{D}|^{fhtw(Q)})$. [OZ'15]

- $fhtw(Q)$ is the fractional hypertree width of Q
- $\rho^*(Q)$ is the fractional edge cover number of Q

$$\boxed{1 \leq fhtw(Q) \leq \rho^*(Q) \leq |Q|}$$

- the gap between $fhtw(Q)$ and $\rho^*(Q)$ is up to $|Q|$

Size Bounds for Flat and Factorised Join Results

Given a join query Q, for any database $\mathbf{D}$, the join result $Q(\mathbf{D})$ admits

- a flat representation of size $O(|\mathbf{D}|^{\rho^*(Q)})$. [AGM'08]
- a factorisation *with caching* of size $O(|\mathbf{D}|^{fhtw(Q)})$. [OZ'15]

These size bounds are asymptotically tight!

- There exist databases $\mathbf{D}$ such that all factorisations over nesting structure of Q have sizes $\Omega(|\mathbf{D}|^{fhtw(Q)})$.
- We can compute these factorisations optimally (up to $\log |\mathbf{D}|$ factor).
- We can compute aggregates in linear time over these factorisations.

Outline

Machine Learning and Factorisation

We consider the common setting where the training data is defined by a join of input tables. Our approach is based on **two conceptual contributions**:

1. Decoupling of **data-dependent computation** and **convergence.**
   - Data-dependent computation can be defined as set of aggregates; the number of aggregates is independent of the size of the data.

2. Data-dependent computation can be done **directly on factorised data.**
   - Provides the worst-case optimal bound to machine learning algorithms.
   - The redundancy in the flat data is not necessary for learning!

My Master thesis: Present a system **F** that shows these insights for Linear Regression Models.

Outline

Glorious Family of Regression Tasks

- Training dataset computed as join of database tables

$$\begin{pmatrix} y^{(1)} & x_1^{(1)} & \dots & x_n^{(1)} \\ y^{(2)} & x_1^{(2)} & \dots & x_n^{(2)} \\ \vdots & \vdots & \ddots & \vdots \\ y^{(m)} & x_1^{(m)} & \dots & x_n^{(m)} \end{pmatrix}$$

  $y^{(i)}$ are labels, $x_1^{(i)}, \dots, x_n^{(i)}$ are features, all mapped to reals.

- We'd like to learn the parameters $\theta = (\theta_0, \dots, \theta_n)$ of the *linear* function

$$h_\theta(x) = \theta_0 + \theta_1 x_1 + \dots + \theta_n x_n.$$

  For uniformity, we add $x_0 = 1$ so that $h_\theta(x) = \sum_{k=0}^{n} \theta_k x_k$.

- Function h_θ approximates the label y of unseen tuples $(x_1, \dots, x_n)$.

Least-Squares Linear Regression

- We consider the least squares regression model with the cost function:

$$J(\theta) = \frac{1}{2} \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)})^2$$

Least-Squares Linear Regression

- We consider the least squares regression model with the cost function:

$$J(\theta) = \frac{1}{2}\sum_{i=1}^{m}(h_\theta(x^{(i)}) - y^{(i)})^2$$

Batch gradient descent (mini-batch/stochastic gradient descent similar):

- Repeatedly change θ to make $J(\theta)$ smaller until convergence:

$$\begin{aligned}\forall 0 \leq j \leq n : \theta_j &:= \theta_j - \alpha\frac{\delta}{\delta\theta_j}J(\theta) \\ &:= \theta_j - \alpha\sum_{i=1}^{m}(\sum_{k=0}^{n}\theta_k x_k^{(i)} - y^{(i)})x_j^{(i)}.\end{aligned}$$

- α is the learning rate.

Least-Squares Linear Regression

- We consider the least squares regression model with the cost function:

$$J(\theta) = \frac{1}{2}\sum_{i=1}^{m}(h_\theta(x^{(i)}) - y^{(i)})^2$$

Batch gradient descent (mini-batch/stochastic gradient descent similar):

- Repeatedly change θ to make $J(\theta)$ smaller until convergence:

$$\begin{aligned}\forall 0 \leq j \leq n : \theta_j &:= \theta_j - \alpha\frac{\delta}{\delta\theta_j}J(\theta) \\ &:= \theta_j - \alpha\sum_{i=1}^{m}(\sum_{k=0}^{n}\theta_k x_k^{(i)} - y^{(i)})x_j^{(i)}.\end{aligned}$$

- α is the learning rate.
- We consider wlog that y is also part of x's and has $\theta = -1$.
- We thus need to compute the following complex aggregates:

$$\forall 0 \leq j \leq n : S_j = \sum_{i=1}^{m}(\sum_{k=0}^{n}\theta_k x_k^{(i)})x_j^{(i)}.$$

F: Regression on Factorised Data

- The sums

$$\forall 0 \leq j \leq n : S_j = \sum_{i=1}^{m} (\sum_{k=0}^{n} \theta_k x_k^{(i)}) x_j^{(i)}.$$

  can be rewritten so that we can express the cofactor of each θ_k in S_j:

$$\forall 0 \leq j \leq n : S_j = \sum_{k=0}^{n} \theta_k \times \text{Cofactor}_{kj}$$

  where $\text{Cofactor}_{kj} = \sum_{i=1}^{m} x_k^{(i)} x_j^{(i)}$

- We decouple the computation of cofactors from convergence of θ's.
  - The cofactor computation only depends on the input data.
  - Convergence can be done once the cofactors are computed.

- **F** computes the cofactors **in one pass** over the factorisation.
  - The redundancy in the flat data is not necessary for learning!

Complexity of $\mathbf{F}$

Theorem

The parameters of any linear function over features from a training dataset defined by a database $\mathbf{D}$ and a join query Q can be learned in time $O(|\mathbf{D}|^{fhtw(Q)})$.

Extensions to F

- Push computation of cofactor matrix of **F** inside the factorized join computation!
  - Removing the lion share of the computation, and computing cofactor matrix in one pass over the input data.

- **F/SQL**: Compute cofactor matrix in SQL.
  - Allowing for direct implementation of **F** in any standard Relational DBMS.

- **F** supports any arbitrary nonlinear basis functions,
  - Including interaction terms between any number of features.
  - **F** can also learn polynomial regression models and factorisation machines with the same data complexity as for linear regression models.

Outline

Experimental Setup

We benchmark **F** against **R** (QR-decomp.), **P**ython StatsModels (ols) and **M**ADlib (glm, ols).
We use FDB and Postgres for computation of factorised and flat join.

US Retailer (real):

- Three tables: Inventory, Census, and Location
- Regression model predicts the amount of inventory units

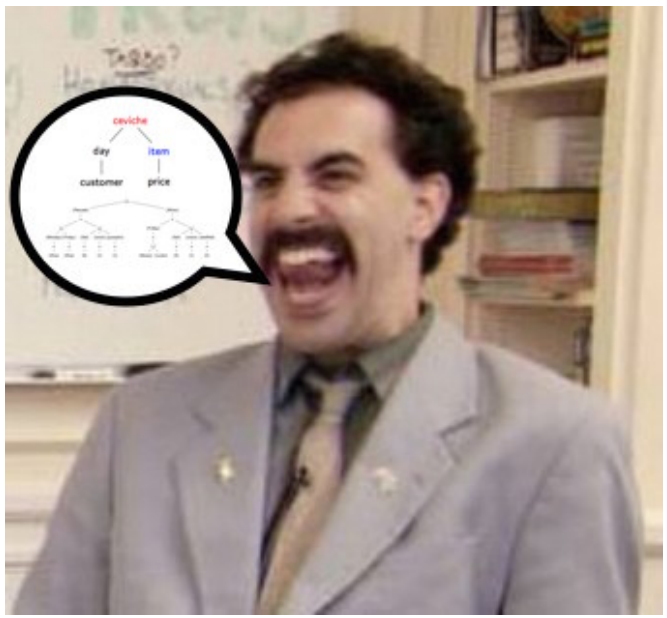
LastFM (real and public):

- Three tables
- Regression model predicts how often a user would listen to an artist based on similar information for its friends.

F vs. **R**, **P**ython StatsModels and **M**ADlib on Real Data

		US retailer L	US retailer N_1	LastFM L_1	LastFM L_2
# parameters		31	33	6	10
Size	Factorised	97,134,675	97,134,675	376,402	315,818
	Flat	2,585,046,352	2,585,046,352	369,986,292	590,793,800
	Compression	26.61×	26.61×	26.61×	982.86×
Join Time	Fact. (FDB)	36.03	36.03	4.79	9.94
	Flat (PSQL)	249.41	249.41	54.25	61.33
Import Time	R	1189.12*	1189.12*	155.91	276.77
	P	1164.40*	1164.40*	179.16	328.97
Learn Time	**F/FDB**	9.69	9.82	0.53	0.89
	M (glm)	2671.88	2937.49	572.88	746.50
	R	810.66*	873.14*	268.04	466.52
	P	1199.50*	1277.10*	35.74	148.84
Total Time	**F**	16.29	16.56	0.11	0.25
	F/FDB	45.72	45.85	5.32	10.83
	F/SQL	108.81	109.02	0.58	2.00
	M (ols)	680.60	737.02	152.37	196.60
	M (glm)	2921.29	3186.90	627.13	807.83
	R	2249.19*	2311.67*	478.20	804.62
	P	2613.31*	2690.91*	269.15	539.14
Speedup	**F** vs. M (ols)	41.78×	44.51×	1385.18×	786.40×
	F vs. M (glm)	179.33×	192.45×	5701.18×	3231.32×
	F vs. R	138.07×	139.59×	4347.27×	3218.48×
	F vs. P	160.42×	162.49×	2446.82×	2156.56×

Outline



Takeaway Messages

- **F** decouples convergence of parameters from data-dependent computation.
- **F** computes the cofactor matrix in one pass over the factorised join.
- Time and space complexity: $O(|\mathbf{D}|^{fhtw(Q)})$, where $fhtw(Q)$ is the fractional hypertree width of the query hypergraph.
- Orders of magnitude faster than learning over the flat join.

Thank you!