

# F: Regression Models over Factorized Views

Dan Olteanu      Maximilian Schleich

Department of Computer Science, University of Oxford  
{dan.olteanu,max.schleich}@cs.ox.ac.uk

## ABSTRACT

We demonstrate **F**, a system for building regression models over database views. At its core lies the observation that the computation and representation of materialized views, and in particular of joins, entail non-trivial redundancy that is not necessary for the efficient computation of aggregates used for building regression models. **F** avoids this redundancy by factorizing data and computation and can outperform the state-of-the-art systems MADlib, R, and Python StatsModels by orders of magnitude on real-world datasets.

We illustrate how to incrementally build regression models over factorized views using both an in-memory implementation of **F** and its SQL encoding. We also showcase the effective use of **F** for model selection: **F** decouples the data-dependent computation step from the data-independent convergence of model parameters and only performs once the former to explore the entire model space.

## 1. WHAT IS F?

**F** is a fast learner of regression models over training datasets defined by select-project-join-aggregate (SPJA) views. It is part of an ongoing effort to integrate databases and machine learning including MADlib [2] and Santoku [3] and goes beyond the state of the art in two fundamental ways [5].

(1) The database joins are an unnecessarily expensive bottleneck for learning due to redundancy in their tabular representation. To alleviate this limitation, **F** *learns models in one pass over factorized joins*, where repeating data patterns are only computed and represented once. This has both theoretical and practical benefits. The computational complexity of **F** follows that of factorized materialized SPJA views [4, 1], which is the lowest worst-case complexity for SPJA views known to date. For learning over acyclic joins, this is linear time and thus worst-case optimal; in contrast, the existing approaches rely on tabular view materialization, which may take exponential time and space [4]. This complexity gap translates in practice to orders of magnitude

performance improvement of **F** over state-of-the-art public systems such as MADlib, Python StatsModels, and R [5].

(2) **F** *decomposes the learning task into a data-dependent step and a subsequent data-independent step*.

The first step computes the aggregates necessary for regression and the factorized view on the input database. The output of this step is a matrix of reals whose dimensions only depend on the arity of the view and is independent of the database size. This matrix contains the necessary information to compute the parameters of any model defined by a subset of the features in the view. This step comes in three flavors [5]. The first flavor computes the regression aggregates on top of the factorized materialized view. The second and fastest flavor is a memory-conscious algorithm that pushes the regression aggregates in the factorized view and avoids the materialization of this view. The third flavor *encodes F in SPJA SQL queries and is readily deployable on any relational database management system*.

The second step performs convergence of model parameters on the computed matrix for a specific model or (forward, backward, both-way) stepwise *automatic model selection* to find the best prediction model for a given label.

**F**'s factorization and task decomposition rely on a representation of data and computation as expressions in the sum-product commutative semiring, which is subject to the law of distributivity of product over sum. Results of SPJA queries are naturally represented in the semiring with Cartesian product as product and union as sum. The derivatives of the objective functions for Least-Squares, Ridge, Lasso, and Elastic-Net regression models are expressible in the sum-product semiring. Optimization methods such as gradient descent and (quasi) Newton, which rely on first and respectively second-order derivatives of such objective functions, can thus be used to train any such model using **F**.

## 2. HOW DOES F WORK?

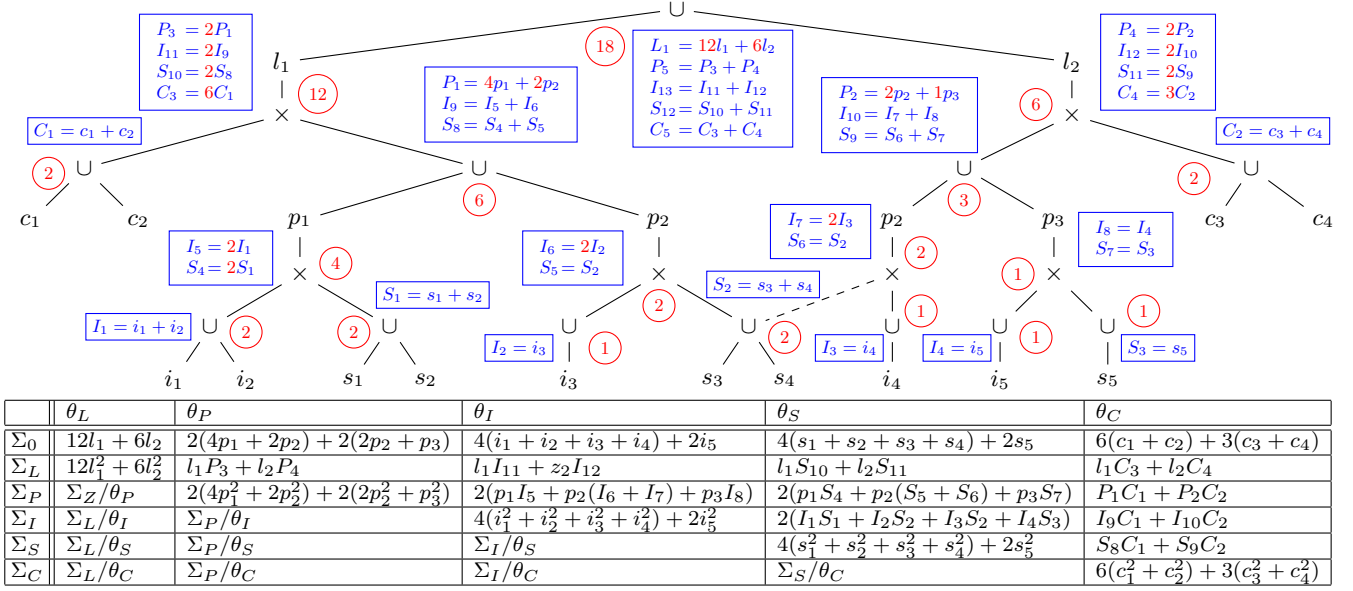
We next explain **F** by means of an example for learning a least-squares regression model over a factorized join.

**Factorized Joins.** Figure 1(a) shows three tables and their natural join. Branch records the location, products and daily inventory of each branch store in the chain. There are many products per location and many inventories per product. Competition records the competitors (e.g., the distance to competitor stores) of a store branch at a given location, with several competitors per location. Sales records all daily sales offered by the store chain for each product. For clarity, we use value placeholders, e.g.,  $p_1$  to  $p_3$ , instead of actual

This work is licensed under the Creative Commons Attribution-NonCommercial-NoDerivatives 4.0 International License. To view a copy of this license, visit <http://creativecommons.org/licenses/by-nc-nd/4.0/>. For any use beyond those covered by this license, obtain permission by emailing [info@vldb.org](mailto:info@vldb.org).

*Proceedings of the VLDB Endowment*, Vol. 9, No. 13  
Copyright 2016 VLDB Endowment 2150-8097/16/09.





**Figure 2: (Top) The factorized join annotated with constant (counts) and linear (weighted sums) aggregates used for cofactor computation. (Bottom) Cofactor matrix based on the annotated factorized join (Column for intercept  $\theta_0$  not shown). For any model, the convergence of model parameters is run on top of this matrix.**

products of values and/or linear aggregates, or of quadratic and constant aggregates. These regression aggregates are passed on to the parent of the current node and used to compute the aggregates at that parent node. The quadratic aggregates of the root node are the cofactors. Figure 2 displays the materialized factorized join, annotated with the constant (circles) and linear aggregates (rectangles), and an excerpt of the cofactor matrix for our training dataset.

**Learning Regression Models with F/SQL.** F’s computation of regression aggregates can be encoded in one SQL query. This is constructed in one bottom-up pass over an extension of the factorized schema  $\Sigma$  with one leaf node per input table under its lowest attribute in  $\Sigma$ . For our schema in Figure 1(b), we generate a query at each inner node following a template. Its instantiation for node  $P$  is below:

```

CREATE TABLE  $Q_P$  AS
SELECT  $L$ ,  $I_n, I_d, Branch_n, Branch_d$ ,
 $S_n, S_d, Sales_n, Sales_d$ ,  $P_n, P_d$ ,
 $(I_{deg} + S_{deg} + P_d)$  AS  $P_{deg}$ ,
sum(power( $P, P_d$ ) *  $I_{agg}$  *  $S_{agg}$ ) AS  $P_{agg}$ 
FROM  $Q_I$  NATURAL JOIN  $Q_S, P_{type}$ 
WHERE  $(I_{deg} + S_{deg} + P_d) \leq 2$ 
GROUP BY  $L$ ,  $I_n, I_d, Branch_n, Branch_d$ ,
 $S_n, S_d, Sales_n, Sales_d$ ,  $P_n, P_d, P_{deg}$ 

```

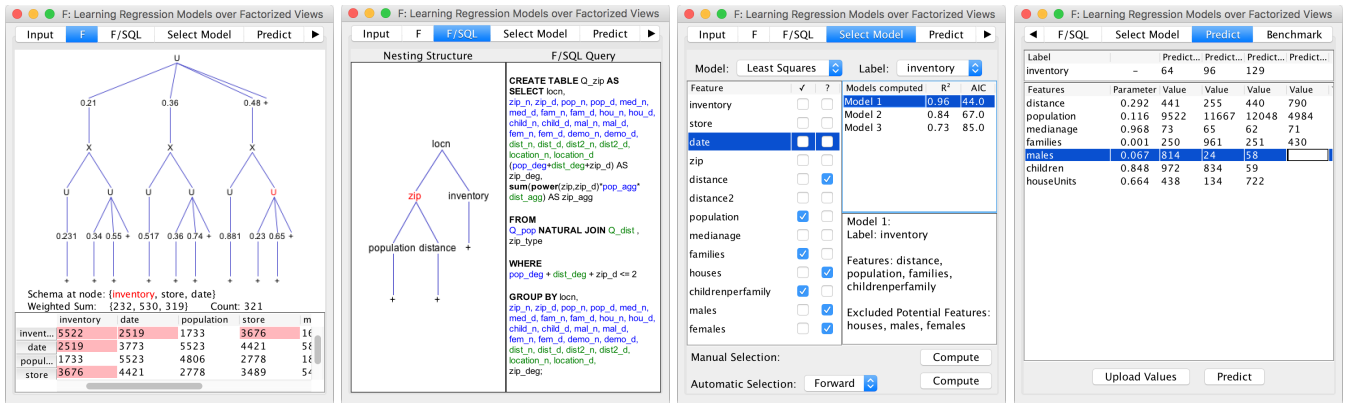
The factorization over any subtree  $\Sigma$  of the schema tree defines a table  $\tau$  with attributes in  $\Sigma$ . The query constructed at the root of  $\Sigma$  computes all (constant, linear, quadratic) regression aggregates over  $\tau$ . The query  $Q_P$  at

node  $P$  is a natural join of the queries  $Q_I$  and  $Q_S$  constructed at its children and has an inequality join with a new table  $P_{type}$  over schema  $(P_n, P_d)$ . This table has three tuples  $(P, 0)$ ,  $(P, 1)$ ,  $(P, 2)$  encoding constant (0), linear (1), quadratic (2) regression aggregates over the attribute  $P$ . There is one such table for each node in  $\Sigma$ .  $Q_P$  computes all aggregates  $(P_{agg})$  of degree  $(P_{deg})$  up to 2 by combining aggregates from children and for  $P$ . For each new aggregate and its degree,  $Q_P$  also maintains the *lineage* of their computation: This is recorded in the attributes with indices  $n$  and  $d$ . Finally,  $Q_P$  also keeps the ancestor attribute  $L$  of  $P$  in  $\Sigma$  to later join with  $Q_C$  in  $Q_L$ . For a leaf node representing a table  $X$ ,  $X_{type} = \{(X, 0)\}$  and the query  $Q_X$  is a product of  $X$  and  $X_{type}$ , where we add an extra attribute  $X_{deg}$  that is a copy of  $X_d$  and we set  $X_{agg} = 1$ .

### 3. HOW CAN USERS INTERACT WITH F?

Figure 3 depicts snapshots of F’s graphical user interface showing step by step the interaction of users with F from its inner workings to model selection and then prediction.

The users load tables and inspect their schemas, which are interpreted as feature sets. F builds regression models over an SPJA query on the selected tables; by default, this query is their natural join. Three alternative next steps can be triggered at this stage: compute the cofactor matrix; show F; and show F/SQL. In the first case, we skip the illustration of F’s inner workings and move to model selection. In the second case (Figure 3(a)), F computes the factorized view and depicts fragments of it along with the computed cofactor matrix. Users can explore the factorized view by clicking on any node  $n$ . If  $n$  is  $\pm$ , then the factorization is expanded by displaying one additional level below  $n$ . If  $n$  is union or product, then it is highlighted along with the cofactors to which the values under  $n$  contributed, and the users are presented with the features of the relation represented by the factorization rooted at  $n$  along with counts



(a) **F** on a factorized view. (b) **F/SQL**: **F** in SQL. (c) Model Selection. (d) Prediction.

**Figure 3:** Users can: load tables and inspect their schemas; inspect the computation of factorized views and cofactor matrix (a) using an in-memory implementation of **F** or (b) a SQL encoding; (c) explore the space of possible models; (d) predict label values for a given set of features; and benchmark **F** against competitors.

and weighted sums for these features. In the third case (Figure 3(b)), we show the factorized schema and **F/SQL**. The users can inspect the subquery at each node in the schema.

Once the cofactors are computed, we can explore the space of models (Figure 3(c)). The users specify the label and the model (least-squares, ridge, lasso, or elastic-net) to be learned. Automatic model selection considers all available features and performs either forward, backward or both-ways stepwise regression to find the best prediction model for the given label with respect to the Akaike information criterion (AIC). The choice for the direction is specified in the drop-down list at the bottom of the panel. Manual selection enables the users to explore the dataset in more detail. From the list of features, the users may select *sure*-features (✓) that are considered highly relevant and *maybe*-features (?) that might be relevant. **F** computes all possible models consisting of the sure-features, and any subset of the maybe-features. **F** displays the model with the best AIC score. Figure 3(c) shows an example of manual model selection. Model 1 only kept the `distance` maybe-feature since all models with a different subset of the maybe-features have a higher AIC value. The list of computed models records the adjusted  $R^2$  and AIC values for each model. The users can compute models with different features sets or labels.

Once a model is selected, the users can predict new label values (Figure 3(d)). The users can upload sets of values for the features of the model and may also manually add new values or change existing ones. Furthermore, the users can benchmark the performance of **F** against that of MADlib, Python StatsModels, or R for a chosen set of models.

#### 4. DEMONSTRATION SCENARIOS

We will demonstrate **F** using its graphical user interface on a range of public datasets (LastFM, MovieLens, Financial, Yelp, Housing, Public Policy) and retailer datasets; we used some of them for experimental benchmarks [5].

The retail sector relies upon predictions to maintain the competitive edge by foreseeing trends. A scenario is to predict how many inventory units a store should keep in order to satisfy the predicted demand. We will use a dataset of

100M records in six tables (inventory, sales, clearance, promotions, census, location) provided by our industrial collaborator LogicBlox. The compression factor of factorized over standard view sizes ranges from 26x to 160x for different sets of tables. To build the model with all features, **F** took 16 seconds while its best competitor MADlib (ols) took 680 seconds [5]. It is common for retail companies to collect a vast amount of data, but it is unclear what model can best predict inventory units. The norm is to try many models from scratch. This is a time and resource-intensive process. Instead, **F** provides a fast way to explore potential models by decoupling data-dependent computation from model space exploration and resolving the former once for all models. This significantly decreases the workload of the data scientist and the computation time and resources.

We will also use the textbook regression scenario that predicts house prices given several features including square meters, location, nearby shops, restaurants, and institutions. This scenario uses tables with 10Ks records and 10s of features extracted using Wrapidity from the property and restaurant aggregators Rightmove and TripAdvisor.

Further scenarios include: (1) Prediction of user ratings for businesses based on their features and previous ratings in the Yelp Dataset Challenge. (2) How demographical, geographical, micro- and macroeconomic features relate to family and individual income in US census data.

**Acknowledgement.** The authors acknowledge the support of the ERC grant FADAMS agreement 682588.

#### 5. REFERENCES

- [1] N. Bakibayev and et al. Aggregation and ordering in factorised databases. *PVLDB*, 6(14):1990–2001, 2013.
- [2] J. M. Hellerstein and et al. The MADlib analytics library or MAD skills, the SQL. *PVLDB*, 5(12):1700–1711, 2012.
- [3] A. Kumar, M. Jalal, B. Yan, J. Naughton, and J. Patel. Demonstration of Santoku: Optimizing machine learning over normalized data. *PVLDB*, 8(12):1864–1875, 2015.
- [4] D. Olteanu and J. Závodný. Size bounds for factorised representations of query results. *TODS*, 40(1):2, 2015.
- [5] M. Schleich, D. Olteanu, and R. Ciucanu. Learning linear regression models over factorized joins. In *SIGMOD*, 2016.