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

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

Sales

P	S
p_1	s_1
p_1	s_2
p_2	s_3
p_2	s_4
p_3	s_5

Branch

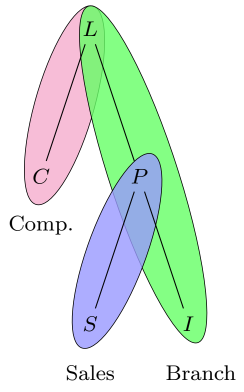
L	P	I
l_1	p_1	i_1
l_1	p_1	i_2
l_1	p_2	i_3
l_2	p_2	i_4
l_2	p_3	i_5

Competition

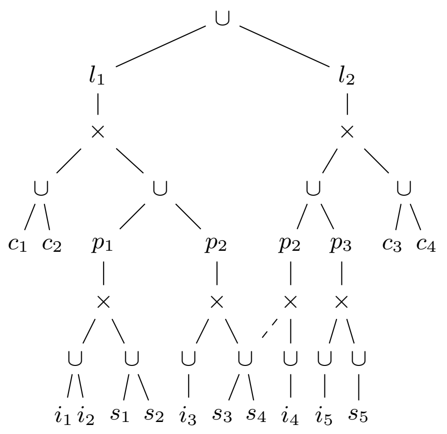
L	C
l_1	c_1
l_1	c_2
l_2	c_3
l_2	c_4

L	C	P	I	S
l_1	c_1	p_1	i_1	s_1
l_1	c_1	p_1	i_1	s_2
l_1	c_1	p_1	i_2	s_1
l_1	c_1	p_1	i_2	s_2
l_1	c_1	p_2	i_3	s_3
l_1	c_1	p_2	i_3	s_4
....				
above block for c_2				
....				
l_2	c_3	p_2	i_4	s_3
l_2	c_3	p_2	i_4	s_4
l_2	c_3	p_3	i_5	s_5
....				
above block for c_4				

(a) Three tables and their natural join.



(b) Factorized schema.



(c) Factorized join.

Figure 1: (a) Database with tables Branch(Location, Product, Inventory), Competition(Location, Competitor), Sales(Product, Sale), where the attribute names are abbreviated and values are not necessarily distinct; (b) Factorized schema for the natural join of the tables; (c) Factorized join over the factorized schema.

values. It is standard for learning that all input values are converted to reals and normalized prior to ingestion by **F**.

We would like to learn a regression model over the natural join of these tables; while SPJA queries are used in practical settings, we focus in this example on joins since they relate features from different tables and are very costly. The widespread approach is to materialize the join result J and use it as the training dataset. However, J exhibits a high degree of redundancy: Since l_1 is paired in Branch with i_1 to i_3 and in Competition with c_1 and c_2, all combinations of the former and the latter values occur in the join result. We can represent this local product symbolically instead of eagerly materializing it. If we systematically apply this observation, we obtain an equivalent factorized representation of the join result that is much more compact than the tabular representation of the join result. Figure 1(c) shows a factorization of the join result, called the factorized join. Each tuple in the join result is represented once in the factorization and can be constructed by following one branch of every union and all branches of a product. Its nesting structure is depicted in Figure 1(b): It is a union of L-values occurring in the join of Branch and Competition on L. For each L-value l, we represent separately the union of C-values paired with l in Competition and the union of P-values paired with l in Branch. That is, given l, the C-values are independent of the P-values and can be stored separately. This is where the factorization saves computation and space as it avoids an explicit enumeration of all combinations of C and P-values for a given l. Also, there are unions of S-values and of I-values under each P-value. The factorization can be further compacted by caching common expressions. For instance, p_2 occurs with the union $s_3 \cup s_4$ from Sales regardless of which L-values p_2 is paired with in Branch. We can store this union once and reuse it for every occurrence of p_2.

Learning Regression Models with F. We next show how **F** learns least-squares regression models with gradient descent over the factorized join; **F** works similarly for the other models and optimization techniques. Least-squares models are defined by linear functions $h_\theta(x) = \theta_0 + \theta_1 x_1 + \ldots + \theta_n x_n$ with parameters $\theta_0, \ldots, \theta_n$ and features $x_1, \ldots, x_n$. The parameters are computed iteratively using the update program

$$\forall 0 \le j \le n : \theta_j := \theta_j - \alpha \sum_{i=1}^{m} (\sum_{k=0}^{n} \theta_k x_k^{(i)}) x_j^{(i)} := \theta_j - \alpha S_j,$$

where i and k denote the indices of the record and respectively of the feature in the training dataset, and α is the learning rate. For simplicity, the label to predict is one of the features $x_0, \ldots, x_n$ and its parameter is -1. The goal of **F** is to compute the family of aggregates Cofactor$_{k,j}$:

$$S_j = \sum_{k=0}^{n} \theta_k \times \text{Cofactor}_{k,j}, \text{ where } \text{Cofactor}_{k,j} = \sum_{i=1}^{m} x_k^{(i)} x_j^{(i)}.$$

F first computes the cofactors over the non-materialized factorized join of the input tables [5]. It then performs parameter convergence directly on the cofactor matrix.

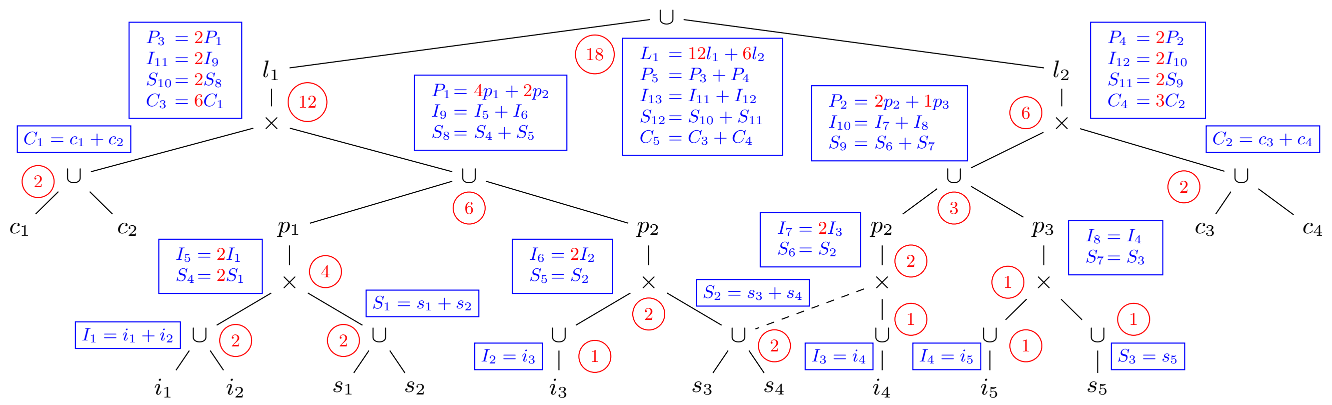
The factorization of the training dataset can be mirrored in the computation of the cofactors. For the training dataset in Figure 1(a), Cofactor$_{P,I}$ exploits the algebraic factorizations $\sum_{i=1}^{n} x \rightarrow x \cdot n$ and $\sum_{i=1}^{n} x \cdot a_i \rightarrow x \cdot \sum_{i=1}^{n} a_i$:

$$\text{Cofactor}_{P,I} = 2p_1 \cdot 2(i_1 + i_2) + 2p_2 \cdot 2i_3 + 2p_2 \cdot 2i_4 + 2p_3 \cdot i_5$$

where the coefficients are the number of C-values that are paired with P-values and I-values. Similarly, we can factorize the pairs of values of independent features as follows: $\sum_{i=1}^{r} \sum_{j=1}^{s} (x_i \cdot y_j) \rightarrow (\sum_{i=1}^{r} x_i) \cdot (\sum_{j=1}^{s} y_j)$. For features P and C, the cofactor would then be:

$$\text{Cofactor}_{P,C} = (4p_1 + 2p_2)(c_1 + c_2) + (2p_2 + p_3)(c_3 + c_4)$$

While exploring the factorized join, **F** computes aggregates at each node that is the root of a factorization E: constant (degree 0) aggregates corresponding to the number of tuples in the table represented by E; linear (degree 1) aggregates for each feature A of E, which are sums of all A-values, weighted by the number of times they occur in the table; and quadratic (degree 2) aggregates, which are


$P_3 = 2P_1$, $I_{11} = 2I_9$, $S_{10} = 2S_8$, $C_3 = 6C_1$
$C_1 = c_1 + c_2$
l_1
12
18
$L_1 = 12l_1 + 6l_2$, $P_5 = P_3 + P_4$, $I_{13} = I_{11} + I_{12}$, $S_{12} = S_{10} + S_{11}$, $C_5 = C_3 + C_4$
$P_1 = 4p_1 + 2p_2$, $I_9 = I_5 + I_6$, $S_8 = S_4 + S_5$
$P_2 = 2p_2 + 1p_3$, $I_{10} = I_7 + I_8$, $S_9 = S_6 + S_7$
l_2
6
$P_4 = 2P_2$, $I_{12} = 2I_{10}$, $S_{11} = 2S_9$, $C_4 = 3C_2$
$C_2 = c_3 + c_4$
2
c_1 c_2
6
p_1
$I_5 = 2I_1$, $S_4 = 2S_1$
4
$S_1 = s_1 + s_2$
$I_6 = 2I_2$, $S_5 = S_2$
p_2
$I_7 = 2I_3$, $S_6 = S_2$
p_2
3
p_3
$I_8 = I_4$, $S_7 = S_3$
c_3 c_4
$S_2 = s_3 + s_4$
$I_1 = i_1 + i_2$
$I_2 = i_3$
$I_3 = i_4$
$I_4 = i_5$
$S_3 = s_5$
i_1 i_2 s_1 s_2 i_3 s_3 s_4 i_4 i_5 s_5

	θ_L	θ_P	θ_I	θ_S	θ_C
Σ_0	$12l_1 + 6l_2$	$2(4p_1 + 2p_2) + 2(2p_2 + p_3)$	$4(i_1 + i_2 + i_3 + i_4) + 2i_5$	$4(s_1 + s_2 + s_3 + s_4) + 2s_5$	$6(c_1 + c_2) + 3(c_3 + c_4)$
Σ_L	$12l_1^2 + 6l_2^2$	$l_1P_3 + l_2P_4$	$l_1I_{11} + z_2I_{12}$	$l_1S_{10} + l_2S_{11}$	$l_1C_3 + l_2C_4$
Σ_P	Σ_Z/θ_P	$2(4p_1^2 + 2p_2^2) + 2(2p_2^2 + p_3^2)$	$2(p_1I_5 + p_2(I_6 + I_7) + p_3I_8)$	$2(p_1S_4 + p_2(S_5 + S_6) + p_3S_7)$	$P_1C_1 + P_2C_2$
Σ_I	Σ_L/θ_I	Σ_P/θ_I	$4(i_1^2 + i_2^2 + i_3^2 + i_4^2) + 2i_5^2$	$2(I_1S_1 + I_2S_2 + I_3S_2 + I_4S_3)$	$I_9C_1 + I_{10}C_2$
Σ_S	Σ_L/θ_S	Σ_P/θ_S	Σ_I/θ_S	$4(s_1^2 + s_2^2 + s_3^2 + s_4^2) + 2s_5^2$	$S_8C_1 + S_9C_2$
Σ_C	Σ_L/θ_C	Σ_P/θ_C	Σ_I/θ_C	Σ_S/θ_C	$6(c_1^2 + c_2^2) + 3(c_3^2 + c_4^2)$

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 θ_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 Σ with one leaf node per input table under its lowest attribute in Σ. 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 $\boxed{L,}$ $\boxed{I_n, I_d, Branch_n, Branch_d,}$

$\boxed{S_n, S_d, Sales_n, Sales_d,}$ $\boxed{P_n, P_d,}$

$\boxed{(I_{deg} + S_{deg} + P_d)}$ **AS** P_{deg},

$\boxed{\text{sum}(\text{power}(P, P_d) * I_{agg} * S_{agg})}$ **AS** P_{agg}

FROM Q_I **NATURAL JOIN** Q_S, P_{type}

WHERE $\boxed{(I_{deg} + S_{deg} + P_d)} <= 2$

GROUP BY $\boxed{L,}$ $\boxed{I_n, I_d, Branch_n, Branch_d,}$

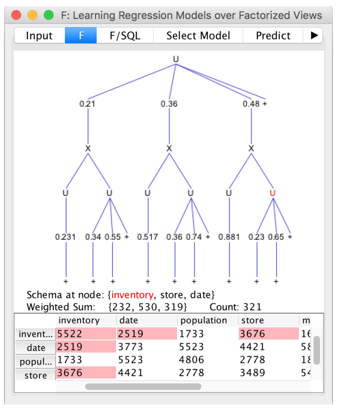
$\boxed{S_n, S_d, Sales_n, Sales_d,}$ $\boxed{P_n, P_d, P_{deg}}$

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

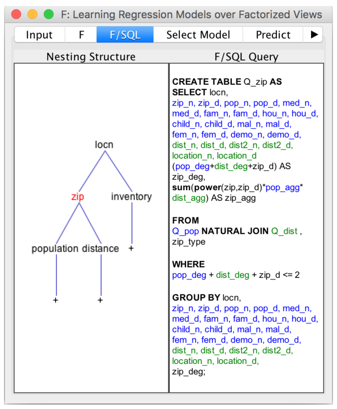
The users load tables and inspect their schemas, which are interpreted as feature sets. **F** builds regression models over an SPJA query on the selected tables; by default, this query is their natural join. Three alternative next steps can be triggered at this stage: compute the cofactor matrix; show **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 **+**, 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



Schema at node: {inventory, store, date}
Weighted Sum: {232, 530, 319} Count: 321

	inventory	date	population	store	m
invent...	5522	2519	1733	3676	16
date	2519	3773	5523	4421	5
popul...	1733	5523	4806	2778	1
store	3676	4421	2778	3489	5

(a) **F** on a factorized view.



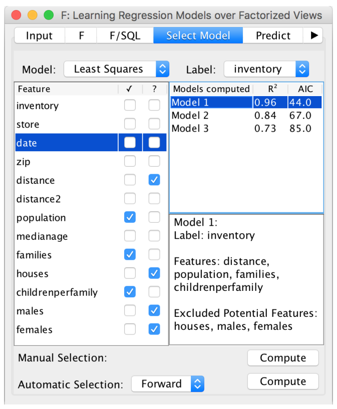
```
CREATE TABLE Q_zip AS
SELECT locn,
zip_n, zip_d, pop_n, pop_d, med_n,
med_d, fam_n, fam_d, hou_n, hou_d,
child_n, child_d, mal_n, mal_d,
fem_n, fem_d, demo_n, demo_d,
dist_n, dist_d, dist2_n, dist2_d,
location_n, location_d
(pop_deg+dist_deg+zip_d) AS
zip_deg,
sum(power(zip,zip_d)*pop_agg*
dist_agg) AS zip_agg

FROM
Q_pop NATURAL JOIN Q_dist ,
zip_type

WHERE
pop_deg + dist_deg + zip_d <= 2

GROUP BY locn,
zip_n, zip_d, pop_n, pop_d, med_n,
med_d, fam_n, fam_d, hou_n, hou_d,
child_n, child_d, mal_n, mal_d,
fem_n, fem_d, demo_n, demo_d,
dist_n, dist_d, dist2_n, dist2_d,
location_n, location_d,
zip_deg;
```

(b) **F/SQL**: **F** in SQL.



Model: Least Squares
Label: inventory

Feature	✓	?
inventory		
store		
date		
zip		
distance		✓
distance2		
population	✓	
medianage		
families	✓	
houses		✓
childrenperfamily	✓	
males		✓
females		✓

Models computed	R^2	AIC
Model 1	0.96	44.0
Model 2	0.84	67.0
Model 3	0.73	85.0

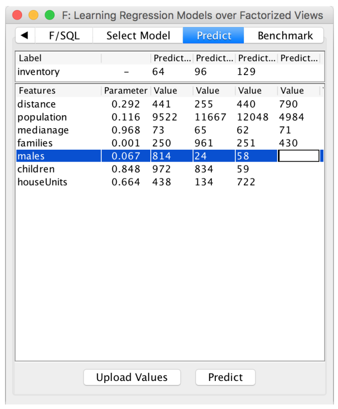
Model 1:
Label: inventory

Features: distance, population, families, childrenperfamily

Excluded Potential Features: houses, males, females

Manual Selection:
Automatic Selection: Forward

(c) Model Selection.



Label		Predict...	Predict...	Predict...	Predict...
inventory	–	64	96	129	

Features	Parameter	Value	Value	Value	Value
distance	0.292	441	255	440	790
population	0.116	9522	11667	12048	4984
medianage	0.968	73	65	62	71
families	0.001	250	961	251	430
males	0.067	814	24	58	
children	0.848	972	834	59	
houseUnits	0.664	438	134	722	

(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.