F-IVM: Learning over Fast-Evolving Relational Data

F-IVM: Learning over Fast-Evolving Relational Data

Milos Nikolic
University of Edinburgh

Haozhe Zhang
University of Oxford

Ahmet Kara
University of Oxford

Dan Olteanu
University of Oxford

ABSTRACT

F-IVM is a system for real-time analytics such as machine learning applications over training datasets defined by queries over fast-evolving relational databases. We will demonstrate F-IVM for three such applications: model selection, Chow-Liu trees, and ridge linear regression.

CCS CONCEPTS

• **Information systems** → **Online analytical processing engines**; *Stream management*; Query optimization.

KEYWORDS

incremental maintenance, feature selection, regression

ACM Reference Format:
Milos Nikolic, Haozhe Zhang, Ahmet Kara, and Dan Olteanu. 2020. F-IVM: Learning over Fast-Evolving Relational Data. In *Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data (SIGMOD'20), June 14–19, 2020, Portland, OR, USA.* ACM, New York, NY, USA, 4 pages. https://doi.org/10.1145/3318464.3384702

1 LEARNING UNDER UPDATES

F-IVM (https://github.com/fdbresearch/FIVM) is a system for real-time analytics over fast-evolving relational databases [5].

F-IVM innovates on two fronts. First, F-IVM puts forward a novel incremental maintenance mechanism for batches of aggregates over arbitrary project-join queries. It constructs a tree of views, with the input relations as leaves, the query as root, and each view defined by the join of its children possibly followed by projecting away attributes. For updates to a relation, it maintains the views along the path to the tree root using delta processing and view materialization.

Second, F-IVM captures the data-intensive computation of many applications using application-specific rings, which define sum and product operations over data values. The ring of integers suffices to treat updates uniformly for sum-product aggregates over joins: negative/positive tuple multiplicities denote deletes/inserts [3]. More complex applications call for richer rings or even composition of rings. F-IVM introduces the *degree-m matrix* ring to maintain the gradients for linear regression models. Moreover, it uses the same view tree to maintain factorized conjunctive query evaluation, matrix chain multiplication, and linear regression, with the only computational change captured by the ring.

F-IVM differs from existing online learning algorithms in at least two ways. (1) Whereas the latter only consider inserts [4], F-IVM also considers deletes. (2) F-IVM avoids the materialization of the training dataset defined by a feature extraction query over multi-relational data. It casts the data-intensive computation of the learning task as ring computation inside the views and pushes it past the joins and down the view tree of the query. This comes with great performance benefits: F-IVM can maintain model gradients over a join faster than maintaining the join, since the latter may be much larger and have many repeating values. It is also competitive against state-of-the-art incremental view maintenance systems: Experiments showed several orders of magnitude performance speedup over DBToaster [3], with an average throughput of 10K updates per second for batches of up to thousands of aggregates over joins of five relations on one thread of a standard Azure machine [5].

We will demonstrate F-IVM's unique ability to maintain the pairwise mutual information (MI) and the covariance matrices (COVAR) over categorical and continuous attributes. These matrices represent the data-intensive computation of common machine learning applications. MI is used for model selection, Chow-Liu trees (optimal tree-shaped Bayesian networks), as cost function in learning decision trees, and in determining the similarity of clusterings of a dataset. COVAR is used for ridge linear regression [6], forward/backward model selection, polynomial regression, and factorization machines. For this demonstration, we will use *model selection, ridge linear regression, and Chow-Liu trees.* Our web-based user interface uses F-IVM to maintain these applications under updates to the Retailer [6] and Favorita [2] databases.

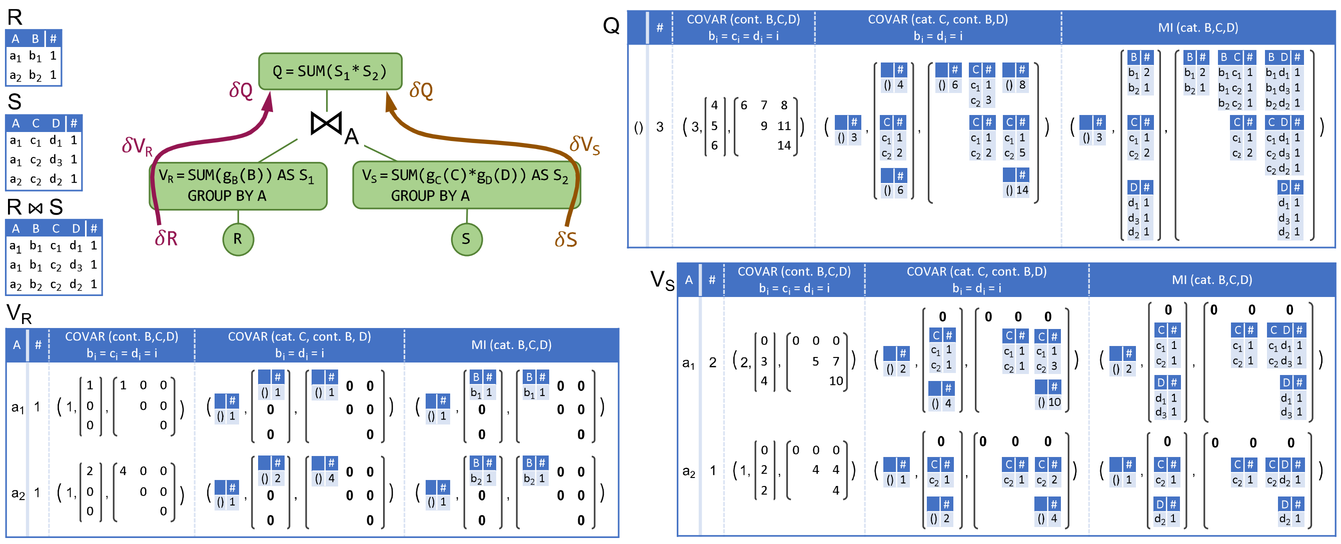

R
S
R ⋈ S
V_R
Q = SUM(S_1 * S_2)
δQ
δQ
⋈_A
δV_R
δV_S
V_R = SUM(g_B(B)) AS S_1 GROUP BY A
V_S = SUM(g_C(C)*g_D(D)) AS S_2 GROUP BY A
δR
R
S
δS
Q
V_S
COVAR (cont. B,C,D) b_i = c_i = d_i = i
COVAR (cat. C, cont. B,D) b_i = d_i = i
MI (cat. B,C,D)

Figure 1: Maintaining $\texttt{SUM}(g_B(B) * g_C(C) * g_D(D))$ **over the join** $R(A,B) \bowtie S(A,C,D)$ **in four scenarios: tuple multiplicities using the** $\mathbb{Z}$ **ring (#); COVAR matrix using the degree-3 ring (continuous B, C, D; also categorical C and continuous B, D); MI matrix using the degree-3 ring (categorical B, C, D). The symmetric matrix values are omitted.**

2 F-IVM BY EXAMPLE

Consider the next query over relations $R(A,B)$ and $S(A,C,D)$:

```
Q = SELECT SUM(g_B(B) * g_C(C) * g_D(D)) FROM R NATURAL JOIN S
```

The aggregates are from a ring $(\mathcal{R}, +, *, \mathbf{0}, \mathbf{1})$. The SUM operator uses the addition + from $\mathcal{R}$. The attribute functions g_B, g_C, and g_D map attribute values to elements in $\mathcal{R}$.

F-IVM exploits the distributivity of multiplication over the SUM operator to push the aggregate past the join and later combine the partial aggregates to produce the query result. For instance, the view V_S computes such partial sums over S:

```
V_S = SELECT A, SUM(g_C(C) * g_D(D)) AS S_2 FROM S GROUP BY A
```

In the view V_S, we treat the A-values as keys and the aggregate S_2-values as payloads. Similarly, we can compute partial sums over R as view V_R. These views are joined as depicted by the *view tree* in Figure 1, which is akin to a query plan.

Figure 1 gives a toy database consisting of relations $R(A,B)$ and $S(A,C,D)$, each mapping tuples to their multiplicity. We next demonstrate how computing Q over this database with different rings can support our application scenarios.

Count Aggregate. We start with the simple count aggregate SUM(1). The aggregate values are from $\mathbb{Z}$, and + and $*$ are the arithmetic operations over $\mathbb{Z}$. The attribute functions g_B, g_C, and g_D map all values to 1. Figure 1 shows the contents of V_R and V_S on the toy database under the ring $\mathbb{Z}$ (the payload column #). To compute Q, we multiply the payloads of matching A-values from V_R and V_S and then sum up these products. The result of Q is a relation mapping the empty tuple () to the total number of tuples in the join of R and S.

Linear Regression. Consider the problem of learning a linear function f with parameters θ_0, θ_B and θ_C that predicts the label D given the features B and C, where the training dataset is the natural join of our relations:

$$f(B,C) = \theta_0 + \theta_B \cdot B + \theta_C \cdot C$$

We assume that B, C, and D have continuous domains; we consider the case when C is a categorical attribute later on.

We can learn f using batch gradient descent. This method iteratively updates the model parameters in the direction of the gradient to decrease the squared error loss and eventually converge to the optimal value. The gradient of the square loss objective function requires the computation of three types of aggregates: the count aggregate SUM(1), the linear aggregates SUM(B), SUM(C), and SUM(D), and the quadratic aggregates SUM(B$*$B), SUM(B$*$C), SUM(C$*$C), SUM(B$*$D), and SUM(C$*$D). These aggregates suffice to capture the correlation between the features B and C and the label D [6].

F-IVM can compute all these aggregates using the query Q and the same evaluation strategy from Figure 1! The only needed adjustment is the replacement of the SQL SUM and $*$ operators with appropriate new sum and product operators.

We treat this batch of aggregates as one compound aggregate $(c, \boldsymbol{s}, \boldsymbol{Q})$, where c is a scalar, $\boldsymbol{s}$ is a 3×1 vector with one sum of values per attribute, and $\boldsymbol{Q}$ is a 3×3 matrix of sums of products of values for any two attributes. This is the COVAR matrix. The compound aggregate can be pushed past the join similarly to the count aggregate discussed before. The payloads of keys carry these aggregates as values, see the payload column COVAR in Figure 1. The compound aggregates are from the *degree-3 matrix* ring

$(\mathcal{R}, +^{\mathcal{R}}, *^{\mathcal{R}}, \mathbf{0}, \mathbf{1})$, where $\mathbf{0} = (0, \mathbf{0}_{3\times1}, \mathbf{0}_{3\times3})$, $\mathbf{1} = (1, \mathbf{0}_{3\times1}, \mathbf{0}_{3\times3})$, and for $a = (c_a, \boldsymbol{s}_a, \boldsymbol{Q}_a) \in \mathcal{R}$ and $b = (c_b, \boldsymbol{s}_b, \boldsymbol{Q}_b) \in \mathcal{R}$:

$$a +^{\mathcal{R}} b = (c_a + c_b, \boldsymbol{s}_a + \boldsymbol{s}_b, \boldsymbol{Q}_a + \boldsymbol{Q}_b)$$
$$a *^{\mathcal{R}} b = (c_a c_b, c_b \boldsymbol{s}_a + c_a \boldsymbol{s}_b, c_b \boldsymbol{Q}_a + c_a \boldsymbol{Q}_b + \boldsymbol{s}_a \boldsymbol{s}_b^{\top} + \boldsymbol{s}_b \boldsymbol{s}_a^{\top})$$

We use attribute names to index elements in $\boldsymbol{s}$ and $\boldsymbol{Q}$; for instance, $\boldsymbol{s} = [\boldsymbol{s}_B\ \boldsymbol{s}_C\ \boldsymbol{s}_D]^{\top}$. For each X-value x, where $X \in \{B, C, D\}$, the attribute function is $\mathrm{g}_{\mathrm{X}}(x) = (1, \boldsymbol{s}, \boldsymbol{Q})$, where $\boldsymbol{s}$ is a 3×1 vector with all zeros except $\boldsymbol{s}_X = x$, and $\boldsymbol{Q}$ is a 3×3 matrix with all zeros except $\boldsymbol{Q}_{XX} = x^2$.

In Figure 1, the payload $V_R(a_1) = \mathrm{g_B}(b_1)$ represents the mapped B-value b_1; the payload $V_S(a_1) = \mathrm{g_C}(c_1) *^{\mathcal{R}} \mathrm{g_D}(d_1) +^{\mathcal{R}} \mathrm{g_C}(c_2) *^{\mathcal{R}} \mathrm{g_D}(d_2)$ represents the sum of products of the mapped (C, D)-pairs with the same A-value a_1; $V_R(a_2)$ and $V_S(a_2)$ are computed similarly. The result of Q maps the empty tuple to the payload $V_R(a_1) *^{\mathcal{R}} V_S(a_1) +^{\mathcal{R}} V_R(a_2) *^{\mathcal{R}} V_S(a_2)$, yielding the count, the vector of aggregates `SUM(X)`, and the matrix of aggregates `SUM(X*Y)`, for $X, Y \in \{B, C, D\}$, over the join of R and S. Our approach significantly shares the computation across the aggregates: The scalar aggregates are used to scale up the linear and quadratic ones, while the linear aggregates are used to compute the quadratic ones.

Linear Regression with Categorical Attributes. Real-world datasets contain a mix of continuous and categorical attributes. The latter take on values from predefined sets of possible values (categories). It is common practice to one-hot encode categorical attributes as indicator vectors.

The COVAR matrix $\boldsymbol{Q}$ from above accounts for the interactions $\boldsymbol{Q}_{XY}$ = `SUM(X*Y)` of attributes $X, Y \in \{B, C, D\}$ with a continuous domain. Assume now that attribute C is categorical and attributes B and D remain continuous. The interaction $\boldsymbol{Q}_{BC}$ captures the aggregates `SUM(B)` per C-value:

$\boldsymbol{Q}_{BC}$ = `SELECT C, SUM(B) FROM R NATURAL JOIN S GROUP BY C`

Using the group-by clause ensures a compact representation of one-hot encoded C-values and that $\boldsymbol{Q}_{BC}$ considers only the C-values that exist in the join result.

We unify the representation of aggregates for continuous and categorical attributes by composing the degree-3 matrix ring with the ring over relations [5] as follows: we use relations as values in c, $\boldsymbol{s}$, and $\boldsymbol{Q}$ instead of scalars; we use union and join instead of scalar addition and multiplication; we use the empty relation $\mathbf{0}$ as zero. The operations $+^{\mathcal{R}}$ and $*^{\mathcal{R}}$ over triples $(c, \boldsymbol{s}, \boldsymbol{Q})$ remain unchanged.

The attribute function $\mathrm{g_X}$ now depends on whether X is continuous or categorical. For any X-value x, $\mathrm{g_X}(x) = (\mathbf{1}, \boldsymbol{s}, \boldsymbol{Q})$, where $\mathbf{1} = \{() \mapsto 1\}$, $\boldsymbol{s}$ is a 3×1 vector with all $\mathbf{0}$s except $\boldsymbol{s}_X = \{() \mapsto x\}$ if X is continuous and $\boldsymbol{s}_X = \{x \mapsto 1\}$ otherwise, and $\boldsymbol{Q}$ is a 3×3 matrix with all $\mathbf{0}$s except $\boldsymbol{Q}_{XX} = \{() \mapsto x^2\}$ if X is continuous and $\boldsymbol{Q}_{XX} = \{x \mapsto 1\}$ otherwise.

Figure 1 shows the contents of V_R, V_S, and Q when using the generalized degree-3 matrix ring with relational values (see the payload column COVAR with C as categorical). The payload in Q captures the count aggregate c, the vector $\boldsymbol{s}$ of aggregates $\boldsymbol{s}_B$ = `SUM(B)`, $\boldsymbol{s}_C$ = `SUM(1)` grouped by C, and $\boldsymbol{s}_D$ = `SUM(D)`, and the matrix $\boldsymbol{Q}$ of aggregates including $\boldsymbol{Q}_{BC}$ = `SUM(B)` grouped by C and $\boldsymbol{Q}_{BD}$ = `SUM(B*D)`. The computation follows the same pattern as with the count aggregate and linear regression with continuous attributes. The only difference is due to the ring used for payloads.

Mutual Information (MI). The MI of two discrete random variable X and Y is defined as:

$$I(X, Y) = \sum_{x \in Dom(X)} \sum_{y \in Dom(Y)} p_{XY}(x, y) \log \left(\frac{p_{XY}(x, y)}{p_X(x)\, p_Y(y)} \right)$$

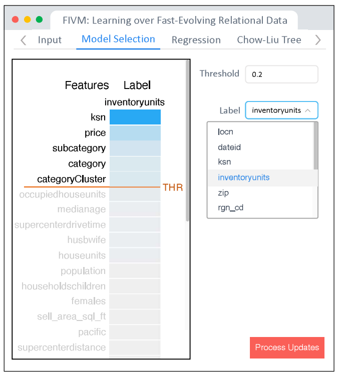
where p_{XY} is the joint probability mass function of X and Y, and p_X and p_Y are the probability mass functions of X and respectively Y. In our database setting, we can capture the joint distribution of two categorical attributes X and Y and the two marginal distributions using four count aggregates: $C_{\emptyset}$ = `SUM(1)`, C_X = `SUM(1)` grouped by X, C_Y = `SUM(1)` grouped by Y, and C_{XY} = `SUM(1)` grouped by (X, Y). The MI of X and Y is then computed as:

$$I(X, Y) = \sum_{x \in Dom(X)} \sum_{y \in Dom(Y)} \frac{C_{XY}(x, y)}{C_{\emptyset}} \log \frac{C_{\emptyset}\, C_{XY}(x, y)}{C_X(x)\, C_Y(y)}$$

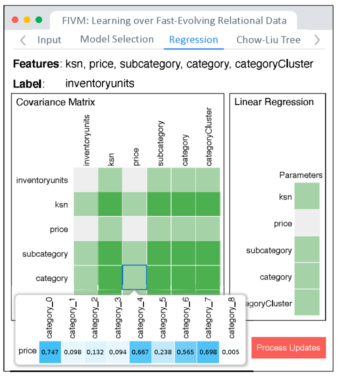
We compute the MI for all pairs (X, Y) of categorical attributes. The aggregates $C_{\emptyset}$, C_X, and C_{XY} are exactly those computed for the COVAR matrix over categorical attributes. We can thus assemble the C_X aggregates into a vector and the C_{XY} aggregates into a matrix, and share their computation as in the linear regression case. When computing the MI for continuous attributes, we first discretize their values into bins of finite size and then follow the same steps as with computing the MI for categorical attributes.

The views V_R, V_S, and Q from Figure 1 capture the aggregates $C_{\emptyset}$, C_X, and C_{XY} of categorical attributes $X, Y \in \{B, C, D\}$ using the degree-3 matrix ring with relational values (the last payload column MI). The payload in Q consists of the count aggregate $C_{\emptyset}$, the vector $\boldsymbol{s}$ of `SUM(1)` aggregates grouped by X, and the matrix $\boldsymbol{Q}$ of `SUM(1)` aggregates grouped by (X, Y), for $X, Y \in \{B, C, D\}$. As in the previous examples, the computation over keys remains the same.

The MI of two attributes quantifies their degree of correlation [4]: A value close to 0 means they are almost independent, while a large value means they are highly correlated. It can identify attributes that predict (are highly correlated with) a given label attribute and can thus be used for model selection [4]. It can also be used for learning the structure of Bayesian networks. The Chow-Liu algorithm [1] constructs an optimal tree-shaped Bayesian network with one node for each attribute. It proceeds in rounds and in each round it adds an edge between two nodes such that their pairwise MI is maximal among all pairs of attributes not chosen yet.



(a) Model Selection

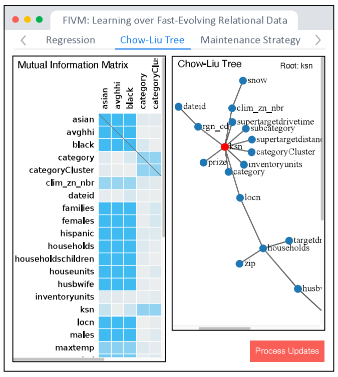


Features: ksn, price, subcategory, category, categoryCluster

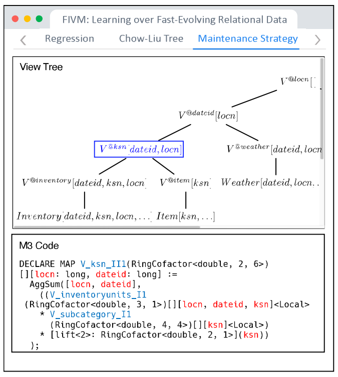
Label: inventoryunits

	category_0	category_1	category_2	category_3	category_4	category_5	category_6	category_7	category_8
price	0.747	0.098	0.132	0.094	0.667	0.238	0.565	0.698	0.005

(b) Regression



(c) Chow-Liu tree


FIVM: Learning over Fast-Evolving Relational Data
Regression
Chow-Liu Tree
Maintenance Strategy
View Tree
$V^{@locn}[$
$V^{@dateid}[locn]$
$V^{@ksn}[dateid, locn]$
$V^{@weather}[dateid, locn$
$V^{@inventory}[dateid, ksn, locn$
$V^{@item}[ksn$
$Weather[dateid, locn, ..$
$Inventory[dateid, ksn, locn, ...$
$Item[ksn, ...]$
M3 Code

```
DECLARE MAP V_ksn_II1(RingCofactor<double, 2, 6>)
[][locn: long, dateid: long] :=
  AggSum([locn, dateid],
    ((V_inventoryunits_I1
 (RingCofactor<double, 3, 1>)[][locn, dateid, ksn]<Local>
    * V_subcategory_I1
      (RingCofactor<double, 4, 4>)[][ksn]<Local>)
    * [lift<2>: RingCofactor<double, 2, 1>](ksn))
   );
```

(d) Maintenance Strategy

Figure 2: F-IVM's web user interface: (a) model selection using pairwise mutual information of attributes with a given label; inspect (b) learning the ridge linear regression model with the selected features and label; (c) the mutual information matrix and the Chow-Liu tree; (d) the F-IVM view tree and M3 code for views.

Incremental Maintenance. Figure 1 shows the leaf-to-root path taken to maintain the query result under updates δR to R. The delta δV_R captures the change in V_R:

$$\delta V_R \;=\; \texttt{SELECT A, SUM}(g_B(\texttt{B}))\ \texttt{AS}\ S_1\ \ \texttt{FROM}\ \delta R\ \ \texttt{GROUP BY A}$$

The delta δV_R further joins with V_S to compute δQ.

The update δR may consist of both inserts and deletes, which are encoded as keys with positive and respectively negative payloads. In our examples, a negative payload is -1 for the count aggregate and $(\{() \mapsto -1\}, \mathbf{0}_{3\times1}, \mathbf{0}_{3\times3})$ for the compound aggregate with relational values, where $\mathbf{0}_{m\times n}$ is the $m \times n$ matrix whose entries are the empty relation.

3 USER INTERACTION

Figure 2 depicts snapshots of F-IVM's user interface.

In the *Input* tab (not shown), the user chooses the database and gives the query defining the initial training dataset. A sequence of updates is prepared for each database. The MI and COVAR matrices and the applications built on top of them are first computed over this initial training dataset.

The *Model Selection* tab allows the user to specify a label attribute for a predictive model and an MI threshold. It then depicts the list of all attributes ranked based on their pairwise MI with the label. Only the attributes above the threshold are selected as features of the model. F-IVM processes one bulk of 10K updates before pausing for one second. The users can then observe how relevant attributes become irrelevant to predicting the label or vice-versa.

The *Regression* tab allows users to inspect the ridge linear regression model with the features and label chosen in the previous tab. F-IVM updates the COVAR matrix after each bulk of updates. Then, a batch gradient descent solver resumes the convergence of the model parameters using gradients that are made of the previous parameter values and the new COVAR matrix [5]. A dark green or white cell in this matrix is a 2D or respectively 0D tensor representing the interaction between two categorical or respectively two continuous attributes. A white cell is a 1D tensor that represents the interaction between a categorical and a continuous attribute. The same color coding is used for the model vector. A tensor is shown when clicking its cell.

The *Chow-Liu Tree* tab depicts the MI matrix for all pairs of attributes in the training dataset and the Chow-Liu tree constructed using this matrix. After each bulk of 10k updates, F-IVM updates the matrix and the tree and then pauses.

The *Maintenance Strategy* tab depicts the F-IVM view tree for the input query. For each view it shows the code for this view in the M3 interpreted representation language [3]. We use the DBToaster backend [3] to compile M3 code into efficient C++ code before running it on the updates.

ACKNOWLEDGMENTS

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 682588.

REFERENCES

[1] C. Chow and C. Liu. 2006. Approximating Discrete Probability Distributions with Dependence Trees. *Trans. Inf. Theor.* 14, 3 (2006), 462–467.

[2] Corporacion Favorita. 2017. Corp. Favorita Grocery Sales Forecasting: Can you accurately predict sales for a large grocery chain? https://www.kaggle.com/c/favorita-grocery-sales-forecasting/.

[3] Christoph Koch et al. 2014. DBToaster: Higher-order Delta Processing for Dynamic, Frequently Fresh Views. *VLDB J.* 23, 2 (2014), 253–278.

[4] Kevin P. Murphy. 2013. *Machine Learning : A Probabilistic Perspective.* MIT Press, Cambridge, Mass.

[5] Milos Nikolic and Dan Olteanu. 2018. Incremental View Maintenance with Triple Lock Factorization Benefits. In *SIGMOD*. 365–380.

[6] Maximilian Schleich, Dan Olteanu, and Radu Ciucanu. 2016. Learning Linear Regression Models over Factorized Joins. In *SIGMOD*. 3–18.