In-Database Factorized Learning

Hung Q. Ngo[1], XuanLong Nguyen[2], Dan Olteanu[3], and Maximilian Schleich[3]

[1]LogicBlox, Inc. hung.ngo@logicblox.com
[2]University of Michigan xuanlong@umich.edu
[3]University of Oxford {dan.olteanu,max.schleich}@cs.ox.ac.uk

1 Introduction

In this paper, we overview recent contributions on in-database analytics for a class of optimization problems that are important for LogicBlox retail-planning and forecasting applications [4, 5, 7]. The class includes ridge linear regression, polynomial regression, factorization machines, principal component analysis and classification models. Such problems are typically computed over input data defined by a *feature extraction join query* on data sources residing inside a database. The query result can have a large number of attributes and records, which leads to large compute times or failure to process the entire dataset for conventional analytics engines.

Pushing analytical computation inside the database engine saves non-trivial time usually spent on data import/export at the interface between database systems and statistical packages. In addition, a large part of the computational challenge for optimization problems can be addressed with conventional database techniques. To show this, we decouple the data-dependent computation from the computation of the optimal solution. The data-dependent step can be phrased as factorized computation of many inter-related aggregates over database joins [3,7]. We further exploit functional dependencies to reduce the dimensionality of the optimization problem [4]. Motivated by the industrial applications, this line of work attracted increasing interest in academia recently [1].

2 Problem formulation

We use a unified framework to express and solve optimization problems [4].

In the following, bold face letters (e.g., $\mathbf{x}$, $\mathbf{x}_i$, $\boldsymbol{\theta}$) denote vectors or matrices, normal face letters (e.g., x_i, θ_j) denote scalars, and $\langle \cdot, \cdot \rangle$ denotes the Frobenius inner product of two matrices. Let Q be a feature extraction join query and D a database that defines the training dataset $Q(D)$ for an optimization problem. Suppose the problem has p parameters $\boldsymbol{\theta} = (\theta_1, \ldots, \theta_p) \in \mathbb{R}^p$, as well as response y and n numeric features $\boldsymbol{x} = (x_1, \ldots, x_n)$, provided by the data points $(\mathbf{x}, y) \in Q(D)$. For a positive integer m, there exist two vector-valued functions $g : \mathbb{R}^p \to \mathbb{R}^m$ and $h : \mathbb{R}^n \to \mathbb{R}^m$. Each component function g_j of $g = (g_j)_{j\in[m]}$ is a multivariate *polynomial* of model parameters. Each component function h_j of $h = (h_j)_{j\in[m]}$ is a multivariate *monomial* of input features. Using the least-squares

loss function with ℓ_2-regularization, all problems in our class of optimization problems are captured in the following objective function:

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

We exemplify the case of ridge linear and polynomial regression [4].

The *ridge linear regression* model with response y and features $\mathbf{x} = (x_0 = 1, x_1, \ldots, x_n)$ has $p = n + 1$ parameters $\boldsymbol{\theta} = (\theta_0, \ldots, \theta_n)$ and the objective

$$J(\boldsymbol{\theta}) = \frac{1}{2|Q(D)|} \sum_{(\mathbf{x},y)\in Q(D)} \left(\sum_{i=0}^{n} \theta_i x_i - y\right)^2 + \frac{\lambda}{2}\left\|\boldsymbol{\theta}\right\|_2^2.$$

This has the form (1) with $m = n+1$, where g and h are the identity functions: $g(\boldsymbol{\theta}) = \boldsymbol{\theta}$ and $h(\boldsymbol{x}) = \boldsymbol{x}$.

The *degree-d polynomial regression* model with response y and features $\mathbf{x} = (x_0 = 1, x_1, \ldots, x_n)$ extends the ridge linear regression model with all possible feature interactions up to degree d. Thus, each component function h_j defines one interaction (e.g. $x_2x_3x_5$) and g_j defines the corresponding parameter (θ_{235}). Formally, the model has $p = m = \sum_{i=0}^{d} n^i$ parameters $\boldsymbol{\theta} = (\theta_{\mathbf{a}})$, where $\mathbf{a} = (a_1, \ldots, a_n)$ is a tuple of non-negative integers such that $\sum_{i=1}^{n} a_i \leq d$. In this case, $g(\boldsymbol{\theta}) = \boldsymbol{\theta}$ and h is defined by the component functions $h_{\mathbf{a}}(\mathbf{x}) = \prod_{i=1}^{n} x_i^{a_i}$.

3 Efficiently Solving the Optimization Problems

We outline how to compute efficiently solutions to problems of the form (1). First, we consider problems with continuous features [5,7], and then generalize to continuous *and* categorical features [4]. The difference is that continuous features are quantitative, whereas categorical features encode qualitative properties (e.g. color or store id). We then show how functional dependencies can be exploited to reduce the dimensionality of the problem.

Solutions for (1) can be computed with a variant of *batch gradient descent* (BGD), which repeatedly updates the parameters in the direction of the gradient until convergence $\boldsymbol{\theta} := \boldsymbol{\theta} - \alpha\boldsymbol{\nabla} J(\boldsymbol{\theta})$. To compute each BGD *iteration* efficiently, we rewrite (1) to decouple the data-dependent computation from the parameters.

Theorem 1 ([4]) *Let* $w = \frac{1}{|Q(D)|}$. *Define the matrix* $\boldsymbol{\Sigma} = (\sigma_{ij})_{i,j\in[m]}$, *the vector* $\mathbf{c} = (c_i)_{i\in[m]}$, *and the scalar* s_Y *by*

$$\boldsymbol{\Sigma} = w \cdot \sum_{(\mathbf{x},y)\in Q(D)} h(\mathbf{x})h(\mathbf{x})^\top; \qquad \mathbf{c} = w \cdot \sum_{(\mathbf{x},y)\in Q(D)} y \cdot h(\mathbf{x}); \qquad s_Y = w \cdot \sum_{(\mathbf{x},y)\in Q(D)} y^2$$

Then, (1) *becomes*

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

$$\boldsymbol{\nabla} J(\boldsymbol{\theta}) = \frac{\partial g(\boldsymbol{\theta})^\top}{\partial \boldsymbol{\theta}} \boldsymbol{\Sigma} g(\boldsymbol{\theta}) - \frac{\partial g(\boldsymbol{\theta})^\top}{\partial \boldsymbol{\theta}} \mathbf{c} + \lambda\boldsymbol{\theta} \tag{2}$$

$\boldsymbol{\Sigma}$, $\mathbf{c}$, and s_Y can be expressed as sum-product functional aggregate queries (FAQ) and computed inside the database. For example, in the case of linear regression, the aggregates in $\boldsymbol{\Sigma}$ are the sums of all possible pairwise products of features (e.g. the SQL-aggregate $\mathsf{SUM}(X_i \cdot X_j)$, $\forall i \leq j \in [n]$).

Our algorithms draw on earlier work on factorized databases [6] and the FAQ framework for computing aggregates over joins [3]. All aggregates can be computed in one pass over the (non-materialized) factorized join [7]. Once these aggregates are computed, each BGD iteration computes the gradient in Equation (2) *without* scanning the data $Q(D)$. The following runtime bounds do not include log factors in the database size.

Proposition 2 ([5, 7]) Let Q be a feature extraction join query, $\mathsf{fhtw}(Q)$ denote its fractional hypertree width, and D a database. If all variables in Q are continuous, then the aggregates $(\boldsymbol{\Sigma}, \mathbf{c}, s_Y)$ for an optimization problem (1) can be computed in $O\left(m^2 \cdot |D|^{\mathsf{fhtw}(Q)}\right)$.

For continuous variables, the size of $\boldsymbol{\Sigma}$ and $\mathbf{c}$ is $O(m^2)$ and respectively $O(m)$. Thus, using the gradient (2) and the precomputed quantities $(\boldsymbol{\Sigma}, \mathbf{c}, s_Y)$, the running-time for BGD is bounded by a polynomial in terms of m, p, and the number of iterations, but *not* database size. To put this result into context, if we would first compute the join using a worst-case optimal join algorithm and then the aggregates, this would take time $O(m^2 \cdot |D|^{\rho^*(Q)})$. For acyclic joins, $\mathsf{fhtw}(Q) = 1$, whereas $\rho^*(Q)$ can be as large as the number of relations in Q [2].

Our problem formulation (1) naturally captures the case of *categorical variables* [4]. Such variables are a common source of sparsity in optimization problems, because they are one-hot encoded. Assume city is a categorical variable in query Q and London, Oxford, Bristol are the only three cities occurring in the query result. One-hot encoding transforms each component x_{city} of $(\mathbf{x}, y) \in Q(D)$ into a 3-dimensional vector $\mathbf{x}_{\mathsf{city}} = [x_{\mathsf{London}}, x_{\mathsf{Oxford}}, x_{\mathsf{Bristol}}]$, where one of the three values is 1 and the others 0, indicating which city occurs in this data point. If a particular data point has $\mathsf{city} = \text{“Oxford”}$, then $\mathbf{x}_{\mathsf{city}} = [0, 1, 0]$.

One-hot encoding can potentially blow up the size of the input data, which makes subsequent processing inefficient. In order to avoid the blow up, modern analytics engines use a sparse representation of the input data. Transforming the data into a sparse format, however, requires non-trivial time.

In our framework, we can capture categorical features by turning the aggregates required to compute $\boldsymbol{\Sigma}$, $\mathbf{c}$, and s_Y into *group-by* aggregates with categorical feature as free variables, as opposed to the sum aggregates without free variables in Theorem 1. In the problem formulation (1), the feature vector $\mathbf{x}$ becomes a vector of vectors, each component g_j of $g = (g_j)_{j \in [m]}$ and h_j of $h = (h_j)_{j \in [m]}$ is a matrix, and each $(\boldsymbol{\sigma}_{ij})_{i,j \in [m]}$ of $\boldsymbol{\Sigma}$ as well as $\mathbf{c}$ are tensors.

The dimensionality of $(\boldsymbol{\sigma}_{ij})_{i,j \in [m]}$ and $\mathbf{c}$ can be very large, but fortunately these tensors are very sparse and have many repeating values. Thus, we compute a sparse tensor representation of $\boldsymbol{\Sigma}$ and $\mathbf{c}$ with an algorithm that factorizes and massively shares aggregate computation. We refer the reader to the full report for details on the algorithm [4].

Proposition 3 ([4]) Let $\mathsf{faqw}(i,j)$ denote the FAQ-*width* of the query corresponding to $\boldsymbol{\sigma}_{ij}$, and $S(i,j)$ denote the size of the sparse representation, i.e., the number of aggregates used to represent the tensor $\boldsymbol{\sigma}_{ij}$. Then, the aggregates $(\boldsymbol{\Sigma}, \mathbf{c}, s_Y)$ can be computed in time $O\left(\sum_{i,j\in[m]}(|D|^{\mathsf{faqw}(i,j)} + S(i,j))\right)$.

For acyclic queries, the FAQ-width [3] is $\mathsf{faqw}(i,j) = 1$. The gradient (2) can now be computed over each tensor $\boldsymbol{\sigma}_{ij}$, whose size $S(i,j)$ is bounded by the database size due to the one-hot encoding. Thus, BGD is not completely independent of the data size if the feature extraction query has categorical variables.

A side effect of one-hot encoding is that the categorical variables become "linearized". We can exploit *functional dependencies* (FDs) on categorical variables to reduce the dimensionality of the optimization problem [4].

Suppose we have the FD $\mathsf{city} \to \mathsf{country}$, where city and $\mathsf{country}$ are two categorical variables in Q and $\mathbf{x}_{\mathsf{city}}$ and $\mathbf{x}_{\mathsf{country}}$ are one-hot encoded vectors. Then, using the one-hot encoding example from above, the following identity must hold: $x_{\mathsf{England}} = x_{\mathsf{London}} + x_{\mathsf{Oxford}} + x_{\mathsf{Bristol}}$. We use this linear relationship to rewrite the monomials in each component h_j of $h = (h_j)_{j\in[m]}$ such that all occurrences of $\mathbf{x}_{\mathsf{country}}$ are replaced by the functionally determining quantity $\mathbf{x}_{\mathsf{city}}$. This leads to a reparameterization of the loss term [4].

The direct implication of the reparameterization is that the number of required aggregates to compute $\boldsymbol{\Sigma}$ and $\boldsymbol{c}$ can be reduced drastically. The effect of the reparameterization on the parameter space is less obvious, because the ℓ_2-regularization term is non-linear. Depending on the structure of the FD, however, many of parameters corresponding to functionally determined statistics can be optimized out [4]. Therefore, the transformed parameter space is also reduced in dimension, which can help speed up the convergence in the optimization phase.

In practice, the interplay of efficient algorithms that compute $(\boldsymbol{\Sigma}, \mathbf{c}, s_Y)$ and the exploitation of FDs to reduce the dimensionality of the problem leads to orders-of-magnitude performance improvements over state-of-the-art analytics engines for optimization problems that are common for LogicBlox analytics [4,7].

References

1. S. Abiteboul and et al. Research Directions for Principles of Data Management (Dagstuhl Perspectives Workshop 16151). *CoRR*, abs/1701.09007, 2017.
2. A. Atserias, M. Grohe, and D. Marx. Size bounds and query plans for relational joins. *SIAM Journal on Computing*, 42(4):1737–1767, 2013.
3. M. A. Khamis, H. Q. Ngo, and A. Rudra. FAQ: Questions asked frequently. In *PODS*, pages 13–28, 2016.
4. H. Ngo, X. Nguyen, D. Olteanu, and M. Schleich. In-database learning with sparse tensors, Technical Report (submitted to VLDB). *CoRR*, abs/1703.04780, 2017.
5. D. Olteanu and M. Schleich. Factorized databases. *SIGMOD Rec.*, 45(2):5–16, 2016.
6. D. Olteanu and J. Závodný. Size bounds for factorised representations of query results. *TODS*, 40(1):2, 2015.
7. M. Schleich, D. Olteanu, and R. Ciucanu. Learning linear regression models over factorized joins. In *SIGMOD*, pages 3–18, 2016.