Modelling Machine Learning Algorithms on Relational Data with Datalog

Nantia Makrynioti
Athens University of Economics and Business
Infor Retail
makriniotik@aueb.gr

Nikolaos Vasiloglou*
RelationalAI, Inc
nvasil@gmail.com

Emir Pasalic
Infor Retail
emir.pasalic@infor.com

Vasilis Vassalos
Athens University of Economics and Business
vassalos@aueb.gr

ABSTRACT

The standard process of data science tasks is to prepare features inside a database, export them as a denormalized data frame and then apply machine learning algorithms. This process is not optimal for two reasons. First, it requires denormalization of the database that can convert a small data problem into a big data problem. The second shortcoming is that it assumes that the machine learning algorithm is disentangled from the relational model of the problem. That seems to be a serious limitation since the relational model contains very valuable domain expertise. In this paper we explore the use of convex optimization and specifically linear programming, for modelling machine learning algorithms on relational data in an integrated way with data processing operators. We are using SolverBlox, a framework that accepts as an input Datalog code and feeds it into a linear programming solver. We demonstrate the expression of common machine learning algorithms and present use case scenarios where combining data processing with modelling of optimization problems inside a database offers significant advantages.

ACM Reference Format:
Nantia Makrynioti, Nikolaos Vasiloglou, Emir Pasalic, and Vasilis Vassalos. 2018. Modelling Machine Learning Algorithms on Relational Data with Datalog. In *DEEM'18: International Workshop on Data Management for End-to-End Machine Learning, June 15, 2018, Houston, TX, USA.* ACM, New York, NY, USA, 4 pages. https://doi.org/10.1145/3209889.3209893

1 INTRODUCTION

As machine learning (ML) becomes more and more prevalent in a large range of applications and domains, a lot of effort has been invested on building specialized languages for developing machine learning tasks. Programs written in these languages usually run on top of distributed processing engines, in a way systems such as SystemML [4], Mahout Samsara [10] and TensorFlow [1] propose. Despite the declarative nature of these languages, the provided set of operators focuses on linear algebra and assumes that data are stored in matrices and vectors. Given that most of the data nowadays are still stored in relational databases, working with ML targeted systems requires tedious processes of exporting/importing data between a database and a ML system. Denormalization also results in losing important domain information embedded in the relational representation.

We argue that developing a unified environment which would combine both data processing and machine learning on relational data is of great use, as it would solve/aleviate many of shortcomings described above. For that purpose, we propose the definition of machine learning models inside a relational database using a variant of Datalog named LogiQL (in the rest of the paper we will use these terms interchangeably), and constrained convex optimization. The user defines the cost function of the model and annotates which of the involved predicates (think those as tables or views) are unknown model parameters. The computation of optimal parameters is performed by low-level constrained optimization (e.g. linear/quadratic programming) solvers and the values are stored back in the selected predicates, which can be queried like any other predicate in the database. Using constrained optimization solvers, it's also possible to add domain knowledge to a model via constraints that include database queries. Currently we focus on linear cost functions and constraints, but our approach can be extended to support more classes of optimization problems, such as quadratic programming.

To integrate linear with logical programming, we use the SolverBlox framework [3] on top of the LogicBlox database [2] and demonstrate that we can express machine learning problems, which can then be exported to the exact same format (.lp Gurobi[1]) and handled by external solvers. The contributions of the paper are summarized below:

- A suitable interface for expressing deterministic machine learning algorithms on relational data, where the user is able to define her model in a declarative manner and get the solution by the system.
- A unified framework that combines data processing with expressing machine learning tasks, which allows users to

*Work done while the author was a member of Infor Retail.

DEEM'18, June 15, 2018, Houston, TX, USA

ACM ISBN 978-1-4503-5828-6/18/06…$15.00
https://doi.org/10.1145/3209889.3209893

[1] http://www.gurobi.com/products/gurobi-optimizer

build different models easily, resulting in a more interactive way for doing data science.
- Implementation of linear regression as a linear program in LogiQL and use case scenarios, which demonstrate the advantages of blending machine learning with databases.

2 RELATED WORK

Due to the focus on integrating machine learning inside a database, our work is close with MLog [8] and efforts to expand SQL with linear algebra data structures and operations [9]. The framework we propose differs from MLog in that we use Datalog instead of a new language, so that the user can still exploit database operators and at the same time simulate a number of linear algebra operations. Regarding the latter direction of work, the difference lies in that we aim for the user to express only the logic of the optimization problem and leave lower-level implementation details of the solution to the system.

Systems that provide declarative domain specific languages for machine learning or embed such languages in imperative ones, such as SystemML [4], Mahout Samsara [16], MiningZinc [5], TensorFlow [1], WOLFE [17] and LBJava [15], require denormalized data as they operate on matrices or tensors. Our approach focuses on bringing machine learning computations where data usually live, i.e. relational databases.

Finally, RELOOP [13], Saul [7] and logical programming variants of AMLP [6], facilitate the modelling of optimization problems on relational data either by querying databases via common APIs or by using a set of abstractions to represent the relational data model. However, the main difference between this third line of work and our approach is that the systems above are still separate from the relational database engine, which means that the latter is not aware of the optimization process. Using SolverBlox, we demonstrate that the user can express optimization problems in an integrated way with the database and take advantage of LogiQL's incremental evaluation.

3 SOLVERBLOX AND LOGIQL

The ideas presented in this paper are implemented on a database management system called LogicBlox (LB). At the core of the LogicBlox database is LogiQL, a declarative language derived from Datalog [11]. In LogiQL, rules based on first order logic are used to specify declaratively *what* the computation should produce (i.e., the logical structure of a program's result), rather than step-by step algorithmic details of how to compute it. A further level of abstraction, SolverBlox, is a framework for expressing linear and mixed integer programs with LogiQL.

With SolverBlox, the user can define predicates in LogiQL to represent the objective function and the constraints of a linear program, as well as other business logic. The SolverBlox framework goes beyond the usual semantics of Datalog by allowing the programmer access to a *second order existential quantifier* [12] (the quantifier is second order because it quantifies not over primitive values, but over predicates): for a predicate `x[i,j]` which corresponds to the (indexed) variable x_{ij} in a mathematical programming problem, the programmer need not specify a set of rules that describe its computation in LogiQL. Rather, she asserts that a predicate `x[i,j]=v` exists such that it (a) maximizes the objective function and (b) satisfies all the model constraints, expressed as a set of *integrity constraints* [14] over `x`. It is at that point up to the SolverBlox implementation to find a predicate that satisfies those conditions.

A linear program is usually represented as a triple consisting of a vector c, a matrix A and a vector b such that

$$\begin{aligned} &maximize \quad c^T x \\ &subject \quad to \quad Ax \leq b \end{aligned} \tag{1}$$

Under the hood, the SolverBlox implementation creates a collection of LogiQL predicates that represent the components of the linear program, as well as rules for populating these predicates.

```
matrix[row,col]=v -> int(row), int(col), float(v).
bound[row]=v -> int(row), float(v).
objective[col]=v -> int(col), float(v).
solution[col]=v -> int(col), float(v).
```

This is a compile-rewrite step that with the exception of a call to a foreign function interface results in a pure LogiQL program P', which has the intended semantics of finding suitable values for predicates marked with `lang:solver:variable` such that the objective function is maximized. At runtime, the call to the foreign function interface marshals the contents of the predicates to an external solver, invokes the solver, and populates the predicate that stores the solution.

The process of going from a constraint based specification in LogiQL (as in our examples below) to the concrete problem instance (c,A,b) is called grounding. The automatic synthesis of a LogiQL program that translates constraints over variable predicates into a representation that can be consumed by the solver is benefited by the optimized and efficient LogiQL evaluation engine. When the data in the LB database change, the program P' is automatically re-executed by the system (part of standard LogiQL semantics) to update the instance (predicates matrix, bound, etc.), and re-invoke the external solver, updating the solution predicate. These updates are also evaluated incrementally, which means that the grounding logic modifies only the parts of the instance matrix that are affected by the changes of the input. The present system has the capacity to execute this process for any linear and mixed-integer programming problem expressed with the syntax described in section 4. More details of the process of grounding in SolverBlox can be found in [2].

4 MACHINE LEARNING ALGORITHMS IN SOLVERBLOX

In this section we describe how we implement linear regression as a convex optimization problem using SolverBlox. With objective function being the least absolute error, L_1 Linear Regression can be naturally expressed as a linear program, as displayed below.

$$\begin{aligned} &\min \sum_{i=1}^{n} \varepsilon_i \\ &subject\ to\ -\varepsilon_i \leq (\hat{y_i} - y_i) \leq \varepsilon_i \end{aligned} \tag{2}$$

In order to present how we express and compute the optimal parameters of a Linear Regression model with LogiQL and SolverBlox, we showcase our implementation on a data science problem from the retail domain. The goal is to predict the future demand of a

SKU (Stock-Keeping Unit) based on historical sales. Predictions are generated for each SKU, at each store and on each day of a forecast horizon.

The main components of LogiQL programs are extensional predicates (EDB), intensional predicates (IDB) and language pragmas. An EDB predicate stores values of the extensional database, that is, values that the user explicitly imported into the database. On the other hand, the values of an IDB predicate are computed via IDB rules, i.e. logical implications that specify what values the predicate should contain, based on the values of other EDB or IDB predicates.

In the code snippet below we define IDB predicates for generating dot products between features and coefficients, as well as sums of these products and predictions for every sku-store-day combination in the input data, e.g. `brand_coeffF`, `sum_of_sku_features` and `prediction` respectively . In this case, features are binary so dot products are basically equal to the value of the corresponding coefficient. Values of dot products and sums are indexed based on whether they concern SKU features, store features, day features or a combination of these. The `error` predicate stores the difference between the target value and the prediction, whereas `totalError` is the sum of individual absolute errors. Due to space limitations, we assume that predicates which are not explicitly defined in the following sample, such as `subfamily_coeffF`, are defined by the user in other parts of the code.

```
sku_coeffF[sku]=v <- unique_skus(sku), sku_coeff[sku]=v.

brand_coeffF[sku]=v <- brand_coeff[br]=v, unique_skus(sku
    ), brand[sku]=br.

sum_of_sku_features[sku]=v <- unique_skus(sku),
sku_coeffF[sku]=v1, brand_coeffF[sku]=v2,
brandType_coeffF[sku]=v3, codeUB_coeffF[sku]=v4,
subfamily_coeffF[sku]=v5, v=v1+v2+v3+v4+v5.

prediction[sku, str, day] = v <- observations(sku,str,day
    ),
sum_of_sku_features[sku]=z1,
sum_of_store_features[str]=z2, sum_of_day_features[day]=
    z3,
sum_of_sku_store_day_features[sku,str,day]=z4,
v=z1+z2+z3+z4.

//IDB predicate of error between prediction and actual
    value
error[sku, str, day] += prediction[sku, str, day] -
    total_sales[sku, str, day].

//IDB predicate of objective function
totalError[] += abserror[sku, str, day] <- observations(
    sku, str, day).
lang:solver:minimal( totalError).

//Constraints for absolute error
observations(sku, str, day),
abserror[sku, str, day]=v1,
error[sku, str, day]=v2
-> v1>=v2.

observations(sku, str, day),
abserror[sku, str, day]=v1,
error[sku, str, day]=v2,
w=0.0f-v2 -> v1>=w.
```

The language annotation `lang:solver:minimal(totalError)` defines that `totalError` is the objective function which needs to be minimized. Variables of the linear program are defined by using the pragma `lang:solver:variable([name_of_predicate])`, which informs the compiler that these predicates are existentially quantified. Next, constraints of the linear program, such as the ones regarding absolute error in our example, are represented by integrity constraints.

SolverBlox will use the described annotations as a starting point to build the main components of the linear program and translate the code to a format supported by the solver.. The solver will compute the optimal parameters for the model and send it back to SolverBlox, which will populate the corresponding predicates. As a result, the user will be able to print the values of the model parameters or access parts of the solution via queries in the same way as with any predicate in the LogicBlox database.

5 USE CASES

Expressing machine learning algorithms as constrained optimization problems in Datalog comes with important benefits. We showcase some of these benefits using specific scenarios.

5.1 Preprocessing of training data

Bringing mathematical optimization modelling to a database gives the user the capability to integrate various data processing operations in the machine learning pipeline. For example, given that a dataset of training observations is loaded to the database, observations can be filtered in many ways using queries and create different training datasets without modifying or parsing the original data file.

The following code shows how the user can build a model that does not include any observations affected by promotion campaigns or another one that involves only sales from Mondays. Such filters can be naturally expressed in LogiQL on the predicate that defines training observations, similarly to a filtering query in SQL.

```
/* Assuming predicate promo_observables is defined and
stores all the sku-loc-day triplets that involve
    promotions */

training_observables_no_promotion(sku,loc,day) <-
!promo_observables(sku, loc, day),
observables(sku, loc, day).

training_observables_only_monday(sku,loc,day) <-
    observables(sku, loc, day), dayOfWeek[day]=v, v="Mon
    ".
```

In a setting where data reside in a database and machine learning tasks are written in mathematical languages, such as R or Matlab, one would need to filter the data inside the database, then denormalize them and export then in a file, in order to load data in matrices.

5.2 User-defined constraints

It is well known that in machine learning there are several models that can minimize the objective but not all of them generalize well, or are interpretable. Using linear programming, the user can integrate domain knowledge to a machine learning model via ad-hoc constraints and examine their effect on the quality of the model. We describe two constraints that were applied on a linear regression model for the demand forecasting problem described in section 4 and report their effect on accuracy results. The point of this experiment is to showcase that the user can build a better model just

Table 1: Results on 3 SKUs (ids: 66, 88, 94) from LR1 and LR2

Model	SKU id	66	88	94
LR1	WAPE training	62.77	95.7	36.26
	BIAS training	0	0	0
	WAPE test	106.9	67.07	49.78
	BIAS test	68.68	-36.54	-1.6
LR2	WAPE training	59.99	98.07	34.44
	BIAS training	0	-1.96	0
	WAPE test	73.29	71.56	50.89
	BIAS test	0.97	-36.92	-0.5

by adding constraints independently of whether Linear Regression is the best choice for a demand forecasting problem.

Table 1 shows results on two metrics, Weighted Absolute Percent Error (WAPE) and bias [2], which are defined in equations 3 and 4 respectively. Results regard two models involving constraints.

$$WAPE = \frac{\sum |actual - forecast|}{\sum actual} \cdot 100 \quad (3)$$

$$Bias = \frac{\sum (actual - forecast)}{\sum actual} \cdot 100 \quad (4)$$

In the first model (LR1) we added a constraint which forces bias for every SKU to be zero during training. This is important in forecasting problems, because apart from low error, we do not want a model that has a general tendency to over-forecast or under-forecast. By setting these bias constraints, we guide the solver to take this requirement into account when optimizing for absolute error.

In the second model (LR2) we relaxed the previous constraints to bound bias by a constant and added another constraint saying that the generated predictions must be greater or equal to zero, as sales cannot have a negative value. This will guide the solver away from solutions that involve negative sales and don't make sense. We can also observe that WAPE and bias results for some of the SKUs are improved by the prediction constraint.

The user can easily experiment with constraints by creating branches of the database and store different versions of the models in a consistent way.

5.3 Aggregated forecasting

In many problems users are interested in forecasts at an aggregated level. For example, retailers many times prefer to monitor the total sales of a family of products at every store per day, including sales from every SKU of that family. Instead of generating sales at SKU-Store-Day level and then aggregating over all SKUs of a particular family, it's very easy to generate predictions at a coarser granularity level by exploiting the normalized structure of the data and indexing relevant predicates appropriately. For example, given a dataset of 2033354 observations, we trained a forecasting model that generated 60346 predictions at the Subfamily-Store-Day level (instead of 2033354 predictions at SKU-Store-Day level followed by aggregation over SKUs) regarding a single family of SKUs directly, which decreased computation time considerably.

[2]Bias denotes the tendency of the model to generally overforecast or underforecast the target variable.

6 DISCUSSION

In this paper we proposed the use of convex optimization to express machine learning algorithms on relational data and demonstrated the value of blending both worlds in order to accelerate and improve data science tasks. As future work, we aim to explore techniques for optimizing the grounding process and reduce the dimensionality of the optimization problems. Another important aspect of integrating optimization in a declarative database is extending to higher order convex programming, like Quadradic and Semidefinite programming. Integration of higher order solvers can extend the expressivity of machine learning algorithms but is not trivial.

REFERENCES

[1] Martín Abadi, Paul Barham, Jianmin Chen, Zhifeng Chen, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Geoffrey Irving, Michael Isard, Manjunath Kudlur, Josh Levenberg, Rajat Monga, Sherry Moore, Derek G. Murray, Benoit Steiner, Paul Tucker, Vijay Vasudevan, Pete Warden, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. 2016. TensorFlow: A System for Large-scale Machine Learning. In *Proceedings of the 12th USENIX Conference on Operating Systems Design and Implementation (OSDI'16)*. USENIX Association, Berkeley, CA, USA, 265–283.

[2] Molham Aref, Balder ten Cate, Todd J. Green, Benny Kimelfeld, Dan Olteanu, Emir Pasalic, Todd L. Veldhuizen, and Geoffrey Washburn. 2015. Design and Implementation of the LogicBlox System. In *SIGMOD 2015*. ACM, New York, NY, USA, 1371–1382. https://doi.org/10.1145/2723372.2742796

[3] C. Borraz-Sanchez, J. Ma, D. Klabjan, E. Pasalic, and M. Aref. [n. d.]. SolverBlox: Algebraic Modeling in Datalog. In *Declarative Logic Programming: Theory, Systems, and Applications*, Michael Kifer and Annie Liu (Eds.).

[4] Amol Ghoting, Rajasekar Krishnamurthy, Edwin Pednault, Berthold Reinwald, Vikas Sindhwani, Shirish Tatikonda, Yuanyuan Tian, and Shivakumar Vaithyanathan. 2011. SystemML: Declarative Machine Learning on MapReduce. In *ICDE 2011*. IEEE Computer Society, Washington, DC, USA, 231–242.

[5] Tias Guns, Anton Dries, Siegfried Nijssen, Guido Tack, and Luc De Raedt. 2017. MiningZinc: A declarative framework for constraint-based mining. *Artif. Intell.* 244 (2017), 6–29. https://doi.org/10.1016/j.artint.2015.09.007

[6] Kristian Kersting, Martin Mladenov, and Pavel Tokmakov. 2017. Relational linear programming. *Artif. Intell.* 244 (2017), 188–216.

[7] Parisa Kordjamshidi, Dan Roth, and Hao Wu. 2015. Saul: Towards Declarative Learning Based Programming. In *Proceedings of the Twenty-Fourth International Joint Conference on Artificial Intelligence, IJCAI 2015, Buenos Aires, Argentina, July 25-31, 2015*. 1844–1851. http://ijcai.org/Abstract/15/262

[8] Xupeng Li, Bin Cui, Yiru Chen, Wentao Wu, and Ce Zhang. 2017. MLog: Towards Declarative In-database Machine Learning. *Proc. VLDB Endow.* 10, 12 (Aug. 2017), 1933–1936.

[9] Shangyu Luo, Zekai J. Gao, Michael N. Gubanov, Luis Leopoldo Perez, and Christopher M. Jermaine. 2017. Scalable Linear Algebra on a Relational Database System. In *ICDE 2017*. 523–534. https://doi.org/10.1109/ICDE.2017.108

[10] Dmitriy Lyubimov and Andrew Palumbo. 2016. *Apache Mahout: Beyond MapReduce* (1st ed.). CreateSpace Independent Publishing Platform, USA.

[11] David Maier and David S. Warren. 1988. *Computing with Logic: Logic Programming with Prolog*. Benjamin-Cummings Publishing Co., Inc., Redwood City, CA, USA.

[12] Toni Mancini. 2004. *Declarative constraint modelling and specification-level reasoning*. Ph.D. Dissertation. Universita' degli Studi di Roma "La Sapienza".

[13] Martin Mladenov, Danny Heinrich, Leonard Kleinhans, Felix Gonsior, and Kristian Kersting. 2016. RELOOP: A Python-Embedded Declarative Language for Relational Optimization. In *Declarative Learning Based Programming, Papers from the 2016 AAAI Workshop, Phoenix, Arizona, USA, February 13, 2016.*

[14] Raghu Ramakrishnan and Jeffrey D. Ullman. 1995. A survey of deductive database systems. *The Journal of Logic Programming* 23, 2 (1995), 125 – 149.

[15] Nick Rizzolo and Dan Roth. 2010. Learning Based Java for Rapid Development of NLP Systems. In *LREC 2010*. European Language Resources Association.

[16] Sebastian Schelter, Andrew Palumbo, Shannon Quinn, Suneel Marthi, and Andrew Musselman. 2016. Samsara: Declarative Machine Learning on Distributed Dataflow Systems. (2016).

[17] Sameer Singh, Tim Rocktäschel, Luke Hewitt, Jason Naradowsky, and Sebastian Riedel. 2015. WOLFE: An NLP-friendly Declarative Machine Learning Stack. In *NAACL HLT 2015*. 61–65.