MLearn: A Declarative Machine Learning Language for Database Systems

Maximilian E. Schüle
m.schuele@tum.de

Matthias Bungeroth
matthias.bungeroth@tum.de

Alfons Kemper
alfons.kemper@tum.de

Stephan Günnemann
stephan.guennemann@tum.de

Thomas Neumann
thomas.neumann@tum.de

Technical University of Munich

ABSTRACT

This paper outlines the requirements of our ML2SQL compiler that allows a dedicated machine learning language (MLearn) to be run on different target architectures. The language was designed to cover an end-to-end machine learning process, including initial data curation, with the focus on moving computations inside the core of database systems. To move computations to the data, we explain the architecture of a compiler that translates into target specific user-defined-functions for the PostgreSQL and HyPer database systems. For computations inside user-defined-functions, we explain the necessary tensor datatypes and the corresponding functions. We base the explanations on an accompanying example of linear regression. To face the challenges to database systems arising from array-like data, we propose such solutions as integrating ArrayQL as stored procedures to unify the relational and array perspectives.

CCS CONCEPTS

• **Information systems → Main memory engines**;

KEYWORDS

SQL, Declarative Language, Database Scripting Languages

ACM Reference Format:
Maximilian E. Schüle, Matthias Bungeroth, Alfons Kemper, Stephan Günnemann, and Thomas Neumann. 2019. MLearn: A Declarative Machine Learning Language for Database Systems. In *International Workshop on Data Management for End-to-End Machine Learning (DEEM'30), June 30, 2019, Amsterdam, Netherlands.* ACM, New York, NY, USA, 4 pages. https://doi.org/10.1145/3329486.3329494

DEEM'30, June 30, 2019, Amsterdam, Netherlands

ACM ISBN 978-1-4503-6797-4/19/06…$15.00
https://doi.org/10.1145/3329486.3329494

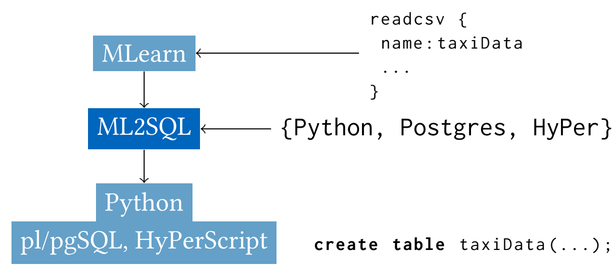

Figure 1: The Conception of *MLearn*: as a metalanguage, ML2SQL compiles code for a target (Python, PostgreSQL, HyPer), also including CSV file preprocessing.

1 INTRODUCTION

Data for machine learning has to be preprocessed until it is possible to apply linear algebra to features aggregated to matrices. For data curation, text-like data such as CSV files have to be processed and cut until a suitable format is found. Afterwards, such optimisation methods as gradient descent find the optimal weights for a parameterised loss function in order to allow predictions on unlabelled data.

From a systems developer point of view, database systems form the native way of efficiently storing data in index structures. Inside of database systems, SQL, as the declarative language, simplifies data curation because it allows feature extraction as projections and selections of the only relevant tuples by design. In addition, hybrid main-memory database systems such as HyPer [4] combine transactional and analytical workload to avoid separate systems for each domain. However, for machine learning purposes, data still has to be extracted before training a model.

In the last decade, many studies have presented architectures for building end-to-end machine learning systems [1, 3, 9]. To assist computations in database systems, the support of matrix or tensor algebra is essential. Therefore, different systems tackle the challenges of representing arrays natively in database systems. Array database systems such as RasDaMan [2] or SciDB [8] replace tables by arrays as the native way of storing data.

As we want to move computations to the data, we have analysed the power of scripting languages within database systems. For database systems that support tensor operations, we have demonstrated the ML2SQL compiler [7] that translates a dedicated machine learning language (MLearn) into domain-specific database language extensions. This paper focuses on the architectural stack for compiling the language (s. Figure 1) and executing the result as stored procedures inside of database systems.

The paper's main contributions are the description of the architecture behind MLearn with an accompanying example, an extension of PostgreSQL by linear algebra and gradient descent on array datatypes, as well as a look on integrating array query languages.

This work is structured as follows: firstly, the MLearn language is introduced together with the functionality of the ML2SQL compiler for translation into Python, pl/pgSQL or HyPerScript. Therefore, we will provide an example of linear regression using data read from an CSV file to show the modularity of the language. Then we will show how the ML2SQL compiler translates these statements into the target language, here as user-defined functions for PostgreSQL using pl/pgSQL. We therefore explain the technical background of features we have added, in order to run different workloads.

2 THE MLEARN LANGUAGE

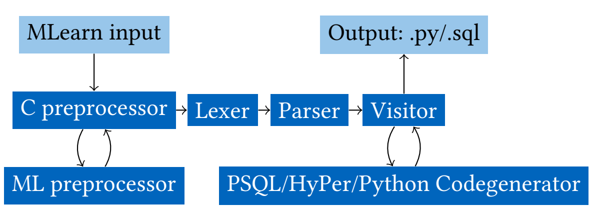

Figure 2: Compiling the MLearn language with the ML2SQL compiler (dark blue): it first preprocesses import and include statements, then it compiles to SQL or Python code.

The dedicated machine learning language MLearn is aimed at preprocessing data and training models intuitively. It provides building blocks for data loading as well as for k-fold cross validation. The underlying models can be optimised by gradient descent or numerically using matrix operations. In addition, the language allows the definition of customised building blocks as functions based on procedural statements and tensor algebra. The classical approach of working on data is to use Python with libraries such as Pandas (for data analysis), NumPy (for matrix operations) and TensorFlow (for optimisations). We simplify the use of these by providing building blocks that make use of the libraries stated when using Python. For example, NumPy computations and TensorFlow models for gradient descent can be specified in linear algebra instead of creating an expression tree manually.

```
readcsv{
 name:taxiData
 file:'/tmp/data/small.csv'
 columns: trip_seconds,trip_miles,pickup_community_area,
      dropoff_community_area,fare,tips
 delimiter: ','
 delete empty entries
}
```

Listing 1: CSV file loading in *MLearn*: the building block requires the target name, the source file, the necessary columns and the delimiter; it allows preprocessing options.

Listing 1 shows the building block for loading CSV files. It requires the target name (to create tensors with NumPy or relations in database systems), the name of the source file, the necessary columns and the delimiter, as well as the preprocessing options such as the deletion of certain entries or string replacements. When compiling for SQL, the resulting code creates a new table, copies CSV files with domain-specific commands while the preprocessing options are translated in update or delete queries (s. Listing 2).

```
create table taxiData(trip_seconds float,
  trip_miles float, pickup_community_area float,
  dropoff_community_area float, fare float, tips float);
\COPY taxiData(trip_seconds,trip_miles,
     pickup_community_area,dropoff_community_area,fare,
     tips) FROM '/tmp/data/small.csv'WITH DELIMITER ','
     CSV HEADER;
delete from taxiData where ((trip_seconds is null) or (
     trip_miles is null) or (pickup_community_area is
     null) or (dropoff_community_area is null) or (fare
     is null) or (tips is null) );
create table taxiDataTest(trip_seconds float,
  trip_miles float, pickup_community_area float,
  dropoff_community_area float, fare float, tips float);
\COPY taxiDataTest(trip_seconds,trip_miles,
     pickup_community_area,dropoff_community_area,fare,
     tips) FROM '/tmp/data/small.csv'WITH DELIMITER ','
     CSV HEADER;
delete from taxiDataTest where ((trip_seconds is null) or
      (trip_miles is null) or (pickup_community_area is
     null) or (dropoff_community_area is null) or (fare
     is null) or (tips is null) );
```

Listing 2: The compiled code for loading CSV files in PostgreSQL: `create`, `update` and `delete` SQL statements are used, as well as the copy command.

Actually, MLearn is designed as a meta-language to facilitate the use of database scripting languages or Python frameworks. The ML2SQL compiler[1] (s. Figure 2) produces Python or SQL code runnable in PostgreSQL or HyPer. The

[1] https://gitlab.db.in.tum.de/ml2sql/ml2sql

compiler is written in ANTLR [5] and allows `import` and `include` statements. As the other computations rely on the initial data loading, the modular language design will allow the inclusion of `C` preprocessor statements (internally, it uses the `gcc -E` command). We allow two kinds of import statements, `include` and `import`. The first command allows basic text insertions known from the gcc preprocessor to load self-defined modules. The second command imports predefined libraries for time measurements, distributions and other convenient functions. In Listing 3, we make use of the import statements to perform linear regression numerically: we first load the libraries for time measurements, then we include the predefined building blocks for loading a training and a test dataset. Afterwards, we define tensors for the attributes (X) and the labels (y); then we add one dimension to X for the bias and finally we compute the optimal weights, all using tensor algebra.

```
import time
import regression
import functions
#include "include/loadTaxiData.ml"
#include "include/loadTestTaxiData.ml"
create tensor DATA from taxiData(trip_seconds, trip_miles
     , pickup_community_area, dropoff_community_area,
     fare, tips)
create tensor TEST_DATA from taxiDataTest(trip_seconds,
     trip_miles, pickup_community_area,
     dropoff_community_area, fare, tips)
X = DATA[: , 0:4]
y = DATA[: , 5]
X_test = TEST_DATA[: , 0:4]
y_test = TEST_DATA[: , 5:5]
start_bias = time()
X = addBiasTerm(X)
X_test = addBiasTerm(X_test)
end_bias = time()
start_reg = time()
w = regression(X , y)
end_reg = time()
start_pred = time()
err = predict(X_test , w) - y_test
avgLoss = sum(err.T * err)/len(X_test)
end_pred = time()
print 'AvgLoss:␣%' , avgLoss
print 'Time␣bias:␣%' , (end_bias - start_bias)
print 'Time␣reg:␣%' , (end_reg - start_reg)
print 'Time␣pred:␣%' , (end_pred - start_pred)
print 'Time␣total:␣%' , ((end_bias-start_bias) + (end_reg
      - start_reg) + (end_pred - start_pred))
```

Listing 3: Numerical linear regression in *MLearn*.

3 TECHNICAL BACKGROUND

In order to allow machine-learning-related computations within database systems, they have to provide tensors and functionalities for training a model. HyPer has already extended its array datatype to serve as tensors by allowing algebra on those types. Inside select clauses of query statements, linear algebra on array datatypes allows various computations. To reach a broader audience for our declarative machine learning language, we also provide some matrix

Operation	Symbol	Arguments	Result
scalar mul.	$a * b$	$a \in \mathbb{R}, b \in \mathbb{R}^{n\times m}$	$\mathbb{R}^{n\times m}$
tensor mul.	$a * b$	$a \in \mathbb{R}^{n\times o}, b \in \mathbb{R}^{o\times m}$	$\mathbb{R}^{n\times m}$
tensor add.	$a + b$	$a, b \in \mathbb{R}^{n\times m}$	$\mathbb{R}^{n\times m}$
tensor sub.	$a - b$	$a, b \in \mathbb{R}^{n\times m}$	$\mathbb{R}^{n\times m}$
tensor power	a^b	$a \in \mathbb{R}^{n\times n}, b \in \mathbb{Z}$	$\mathbb{R}^{n\times n}$
transpose	a^t	$a \in \mathbb{R}^{n\times m}$	$\mathbb{R}^{m\times n}$
identity	$id(n)$	$n \in \mathbb{N}$	$\mathbb{B}^{n\times n}$
array fill	$fill(r, n, m)$	$r \in \mathbb{R}, n, m \in \mathbb{N}$	$\mathbb{R}^{n\times m}$

Table 1: Implemented matrix operations for PostgreSQL needed by MLearn.

algebra functionalities for PostgreSQL online[2]. The operations are first precompiled as a shared library (s. Listing 4), then loaded by SQL statements (s. Listing 5). Table 1 shows the list of matrix operations needed that we implemented in PostgreSQL to allow pl/pgSQL procedures with matrix operation produced by the ML2SQL compiler.

```
Datum tensor_add(PG_FUNCTION_ARGS)
{
...
 ArrayType *a1 = PG_GETARG_ARRAYTYPE_P(0),
   *a2 = PG_GETARG_ARRAYTYPE_P(1);
 int *dim1 = ARR_DIMS(a1);
 int *dim2 = ARR_DIMS(a2);
 length1 = ArrayGetNItems(ARR_NDIM(a1), ARR_DIMS(a1));
 Datum * ps1 = (Datum*) ARR_DATA_PTR(a1);
 Datum * ps2 = (Datum*) ARR_DATA_PTR(a2);
 #pragma omp parallel for
 for (int i = 0; i < length1; i++){
    float8 res = DatumGetFloat8(ps1[i]);
    res += DatumGetFloat8(ps2[i]);
    ps1[i] = Float8GetDatum(res);
 }
 PG_RETURN_ARRAYTYPE_P(a1);
}
```

Listing 4: Exemplary tensor operation (add) programmed in `C` for PostgreSQL.

```
CREATE or REPLACE FUNCTION tensor_add(FLOAT [], FLOAT [])
RETURNS FLOAT [] AS
  '/tmp/psql-matrix-extension/bin/MatrixOperations.so',
  'tensor_add' LANGUAGE C STRICT;
CREATE OPERATOR + (PROCEDURE=tensor_add,
  LEFTARG=FLOAT[],RIGHTARG=FLOAT[]);
```

Listing 5: Operator creation for addition on arrays in PostgreSQL out of the shared library.

In addition to matrix operations, a gradient descent optimiser is essential for training models inside database systems. When compiling MLearn to SQL, we make use of our gradient descent operator supporting automatic differentiation with lambda expressions already integrated in HyPer [6]. Lambda expressions inject user-defined code into hard-coded database operators, for example distance metrics in clustering algorithms. Here, they allow arbitrary loss functions to

[2] https://gitlab.db.in.tum.de/ml2sql/psql-matrix-extension

be specified as shown in Listing 6. Inside, we can combine attributes from a training dataset with weights, either individually by hand or aggregated to matrices.

```
select * from gradientdescent(
  λ(d,w) (w.a+d.x1*w.b+dx2*c-d.y)^2,
  (select x1,x2,y from trainingdata d),
  (select a,b,c from weights w),0.05, 100);
select * from gradientdescent(
  λ(d,w) (w.w*d.x-d.y)^2,
  (select ARRAY[1,x1,x2] x, y from trainingdata d),
  (select ARRAY[a,b,c] w from weights),0.05, 100);
```

Listing 6: Gradient descent table function using lambda expressions: inside the expression, each parameter can be combined separately or using matrix algebra.

In addition to matrix operations, we adapted our gradient descent framework to run with PostgreSQL. However, as the dimensions of PostgreSQL arrays are assigned dynamically, the PostgreSQL gradient descent operator expects the dimensions as futher arguments inside the lambda expression for the loss function. Together with these extensions, we are now able to translate the code in Listing 3 to pl/pgSQL to run in PostgreSQL. Listing 7 shows an extract of the resulting code.

```
CREATE OR REPLACE FUNCTION regression(X float[][], y
     float[][]) returns float[][] AS $$
declare
    Xt float[][];
begin
    Xt := matrix_transpose(X);
    return matrix_power((Xt*X),(-1)::integer)*Xt*y;
END; $$ LANGUAGE plpgsql;

CREATE OR REPLACE FUNCTION addBiasTerm(X float[][] )
returns float[][] AS $$
declare
    bias float[][];
begin
 bias:=array_fill(1::float,ARRAY[1,array_length(X,0+1)]);
 return matrix_transpose((bias||matrix_transpose(X)));
END; $$ LANGUAGE plpgsql;
```

Listing 7: Resulting pl/pgSQL procedures for linear regression and increasing the matrix by the bias.

4 CONCLUSION AND ONGOING WORK

This paper has introduced the architectural details behind a declarative machine learning language (MLearn) with the aim of shifting computations inside of the core of database systems. It has shown how the ML2SQL compiler treats preprocessor statements to allow the inclusion of code snippets and libraries. It has provided a basic example of how to compute linear regression and referred to the necessary technical implementations such as gradient descent in PostgreSQL and HyPer. For this purpose, we have specified the required matrix operations in PostgreSQL with one coding example.

To sum up, we have discovered out that array processing represents the major building block for tasks related to machine learning. These tasks would strongly benefit from SQL especially for data preprocessing. In addition, when integrating the advantages of array database into hybrid OLTP and OLAP database systems, no domain specific systems would be required. We shall therefore work on applying matrix algebra to tables using stored procedures that are written in ArrayQL.

ACKNOWLEDGEMENT

This research has received funding from the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme (grant agreement No 725286).

REFERENCES

[1] Christopher R. Aberger, Andrew Lamb, Kunle Olukotun, and Christopher Ré. 2018. LevelHeaded: A Unified Engine for Business Intelligence and Linear Algebra Querying. In *ICDE 2018, Paris, France, April 16-19, 2018*. 449–460. https://doi.org/10.1109/ICDE.2018.00048

[2] Peter Baumann, Andreas Dehmel, Paula Furtado, Roland Ritsch, and Norbert Widmann. 1998. The Multidimensional Database System RasDaMan. In *SIGMOD 1998, Proceedings ACM SIGMOD International Conference on Management of Data, June 2-4, 1998, Seattle, Washington, USA.* 575–577. https://doi.org/10.1145/276304.276386

[3] Matthias Boehm, Michael Dusenberry, Deron Eriksson, Alexandre V. Evfimievski, Faraz Makari Manshadi, Niketan Pansare, Berthold Reinwald, Frederick Reiss, Prithviraj Sen, Arvind Surve, and Shirish Tatikonda. 2016. SystemML: Declarative Machine Learning on Spark. *PVLDB* 9, 13 (2016), 1425–1436. https://doi.org/10.14778/3007263.3007279

[4] Alfons Kemper and Thomas Neumann. 2011. HyPer: A hybrid OLTP&OLAP main memory database system based on virtual memory snapshots. In *ICDE 2011, April 11-16, 2011, Hannover, Germany*. 195–206. https://doi.org/10.1109/ICDE.2011.5767867

[5] Terence John Parr and Russell W. Quong. 1995. ANTLR: A Predicated-*LL(k)* Parser Generator. *Softw., Pract. Exper.* 25, 7 (1995), 789–810. https://doi.org/10.1002/spe.4380250705

[6] Maximilian Schüle, Frédéric Simonis, Thomas Heyenbrock, Alfons Kemper, Stephan Günneman, and Thomas Neumann. 2019. In-Database Machine Learning: Gradient Descent and Tensor Algebra for Main Memory Database Systems. In *18th symposium of "Database systems for Business, Technology and Web" (BTW), in Rostock, Germany. Proceedings.*

[7] Maximilian Schüle, Matthias Bungeroth, Dimitri Vorona, Alfons Kemper, Stephan Günnemann, and Thomas Neumann. 2019. ML2SQL - Compiling a Declarative Machine Learning Language to SQL and Python. In *EDBT 2019, Lisboa, Portugal, March 26-29, 2019.*

[8] Michael Stonebraker, Paul Brown, Alex Poliakov, and Suchi Raman. 2011. The Architecture of SciDB. In *SSDBM 2011, Portland, OR, USA, July 20-22, 2011. Proceedings*. 1–16. https://doi.org/10.1007/978-3-642-22351-8_1

[9] Manasi Vartak, Harihar Subramanyam, Wei-En Lee, Srinidhi Viswanathan, Saadiyah Husnoo, Samuel Madden, and Matei Zaharia. 2016. ModelDB: a system for machine learning model management. In *HILDA@SIGMOD 2016, San Francisco, CA, USA, June 26 - July 01, 2016*. 14. https://doi.org/10.1145/2939502.2939516