# Toward a Declarative Query Language for Machine Learning

Hasan M Jamil
Department of Computer Science
University of Idaho, USA
jamil@uidaho.edu

## ABSTRACT

The popularity of data science as a discipline and its importance in the emerging economy dictate that machine learning be practiced as a basic tool. This also means that the current practice of workforce training and using machine learning tools with low level statistical and algorithmic details is a barrier that needs to be addressed. Similar to data management languages such as SQL, machine learning needs to be taught, practiced and used at a conceptual level to help make it a staple tool for learners and practitioners alike. In this paper, we introduce a new declarative machine learning query language, called *MQL*, for naive users. We discuss its merits and possible ways of implementing it in a traditional relational database system.

**VLDB Workshop Reference Format:**
Hasan M Jamil. Toward a Declarative Query Language for Machine Learning. VLDB 2024 Workshop: Tabular Data Analysis Workshop (TaDA).

**VLDB Workshop Artifact Availability:**
The source code, data, and/or other artifacts have been made available at https://github.com/hmjamil/mql.

## 1 INTRODUCTION

In this paper, we ask the question, how difficult is it to design a declarative query language for machine learning (ML) analysis by pointing out how difficult and arcane it is to write code segments on popular ML platforms such as SciKit-Learn, Pytorch, R, or TensorFlow by non ML experts. By declarative, we mean that if a language for machine learning that is as simple and as powerful as SQL can be designed, can it perform the most complex analysis a modern machine learning algorithm can?

The current state of ML is not accessible to most potential users of data science [8], and we concur with many researchers who believe that a significant barrier exists toward exploiting ML without a declarative platform [18]. In the absence of a language similar to SQL, it is extremely difficult and unlikely for naive users to comprehend, let alone devise, a simple regression analysis code fragment easily executable on a machine. For example, the process to performing a clustering analysis [21] (or classification [17]) on the Boston housing dataset on Kaggle [19] is by no means an easy task, even for a good computational scientist without adequate proficiency in regression analysis. It requires exploratory data analysis, principal component analysis, and more to get a sense of the data and to make a decision about the number of clusters that are appropriate, most of which can also be performed by a smart algorithm. Then there is the issue of accuracy, and the selection of the best model for the analysis [1, 9, 28].

The natural question then is: are all these details necessary, at least most of the time? Could these analysis algorithms be chosen by a query processor from an abstract request for prediction, clustering, or classification based on the properties of the data sets the same way relational database engines select join algorithms, aggregate function algorithms, or procedures for OLAP functions? Could optimization be possible and decided by query processors in ways analogous to SQL engines?

While we do not currently have all the answers, we believe that the starting point should be the development of a suitable declarative query language for ML that will be simple in spirit and expressive enough to be able to support most, if not all, ML analysis needs on tabular data. To that end, our goal in this paper is to introduce an ML query language, called *MQL*, capable of supporting three basic classes of ML tasks - prediction, classification, and clustering. We organize the language constructs into three tiers – data preparation (or wrangling), model construction, and ML analysis. These language constructs have distinct semantics and no inherent inter-dependencies. In the sections to follow, we first present MQL's syntax and semantics and then discuss its merit over contemporary declarative languages before we conclude.

## 2 RELATED WORK

The main purpose of a declarative language is to reduce the human-machine interfacing barriers by making machine instructions simple and easy to conceptualize. An all time great example of declarative languages is SQL. While this definition of declarativity is subject to interpretation, the essence should remain. From this standpoint, a simple language for ML has to be highly abstract and should support the so called naive users' use of ML tasks, having only conceptual and rudimentary knowledge of this technology while the machines assume the bulk of the technical underpinnings and efficiency concerns [30]. Given that ML tasks are complex, involved, and require subject expertise, meeting such levels of abstraction requirements in a human-machine interfacing language, or query language, is undoubtedly a tall order.

Nonetheless, several attempts were made to simplify the use of ML technologies for the masses. Among them, AutoML [23] may be the most prominent effort of all. While challenges remain [5], the emergence of large language models appears to address many of these challenges to some extent [29] toward democratizing ML. AutoML, or Automated Machine Learning, is a set of techniques aimed at automating the process of building ML models. The basic idea behind AutoML is to make ML more accessible to users with


Proceedings of the VLDB Endowment. ISSN 2150-8097.

limited ML expertise by automating some of the complex and time-consuming tasks, such as data preprocessing, feature engineering, model selection, and hyperparameter optimization, involved in model development. By automating these tasks, AutoML aims to reduce the amount of manual effort required to build and deploy ML models, making it easier for non-experts to leverage the power of ML in their applications.

Variants of the ideas behind AutoML are also being investigated. Among them, MLBase [11] attempted to help automate the pipeline by proposing a declarative language and an optimizer to lend a hand in balancing the efficiency aspects of declarativity that usually delegate this responsibility to the system. Despite the design goal, the language they support appears to have procedural features and is not abstract enough compared to languages such as SQL to have a wider appeal.

WOLFE [27] was an early natural language interpreter ML front-end development effort. In approaches such as WOLFE, query understanding and mapping the intent into some form of executable code is employed, which is written in TensorFlow. A similar translational approach is used in languages such as sql4ml [16], ML2SQL/MLearn [24, 25], P6 [14], MLog [15], Datalog [32], and Dyna [31]. The popularity of translational implementation of declarative languages into ML frameworks such as TensorFlow, PyTorch, or SciKit Learn is not by accident. Rather, it is convenience and a desire to leverage community investments in powerful and large bodies of algorithms for ML developed over a few decades. Until more powerful end-to-end ML systems are developed and mature, such as SystemML [4], SystemDS [3], EndToEndML [20], Merlion [2], VeML [13], and Relax [12], we believe these translation-based systems will continue to play a major role in democratizing ML.

## 3 MACHINE LEARNING QUERY LANGUAGE

With the intent to stay close to an SQL-like language, we propose a syntax similar to SQL for MQL and design lower level operational procedures to assign semantics to the declarative statements of MQL. MQL retains part of the SQL flavor to leverage the community knowledge of SQL and reduce cognitive overload.

### 3.1 Syntax of MQL

MQL supports two basic statements – GENERATE for querying tables and CONSTRUCT for creating an ML model. While the GENERATE statement is able to exploit an existing model, it can also operate without one by generating its own model.

*3.1.1 The GENERATE Statement.* The GENERATE statement stands at the level of SQL's SELECT statement and is the main workhorse of MQL. It operates on tabular data to make predictions, categorize objects, and group sets of objects into bins. It has five basic clauses – an ML class selection (one of PREDICTION, CLASSIFICATION, and CLUSTER), optional object labeling, feature selection, a data set, a filter condition over the data set, and an input table of unknown cases (the test set).

```
GENERATE [DISPLAY OF]
PREDICTION v [OVER s] |
CLASSIFICATION INTO L1, L2, ..., Lp [OVER s] |
CLUSTER OF k
[USING MODEL ModelName | ALGORITHM AlgorithmName]
[WITH MODEL ACCURACY P]
[LABEL B1, B2, ..., Bm]
[FEATURES A1, A2, ..., An
FROM r1, r2, ..., rq
WHERE c]
```

In the above statement, $r_l$ is a table over the scheme $R_l$, $c$ is a Boolean condition, $A_i \in \cup_l R_l$, $s$ is a table over the scheme $\cup_j B_j \bigcup \cup_i A_i$, k is an integer, and $v \in \cup_l R_l$, $L_k \in dom(X)$[1]. In this statement and in all the MQL statements, the vertical bar (|) means exclusive OR, and the square bracket ([]) means optional.

$v$ in the PREDICTION clause is the target variable, and $A_i$s are the features. The optional LABEL clause identifies attributes $B_j$ as the object identifiers for all three ML tasks. The CLASSIFICATION clause classifies each object $\cup_j B_j$ into one of $L_k$ categories. The $k$ in CLUSTER clause is an integer expression that can include SQL aggregate functions over the tables $r_l$. Finally, the optional USING clause is meant to either use an existing model (MODEL option) generated using the CONSTRUCT clause (discussed next), or a specific ML algorithm (ALGORITHM option) for the generation of the model. As in SQL, WHERE is an optional clause, but unlike SQL, FROM is required. The OVER clause supplies the unknown test dataset over the scheme $A_i \cup B_j$. The ACCURACY option accepts a threshold within the interval (0,1).

*3.1.2 CONSTRUCT Statement.* To create an explicit model, MQL uses the CONSTRUCT statement below. It stands at a level similar to SQL's CREATE TABLE statement, but is at the data level and more functional. It is able to generate a default model for prediction, classification, or clustering, optionally using a specific algorithm for supervised or unsupervised learning. The TRAIN ON parameter N (similarly TEST ON) is an integer value less than the cardinality of the table $r$, and can be expressed as an expression, possibly using SQL aggregate functions. While the expression for $M$ should be such that $N + M \leq |r|$ (where $|r| = |r_1| \times |r_2| \times \ldots \times |r_n|$), MQL will not object if the condition $N + M \leq |r|$ is not met and will assign the eventual semantics entailed by these two expressions. In this statement, $A_i$ is the feature vector over which the model is created.

```
CONSTRUCT ModelName [AS SUPERVISED | UNSUPERVISED]
FOR PREDICTION v |
CLASSIFICATION INTO L1, L2, ..., Lp |
CLUSTER OF k
[USING AlgorithmName]
[WITH MODEL ACCURACY P]
TRAIN ON N TEST ON M
FEATURES A1, A2, ..., An
FROM r1, r2, ..., rn
WHERE c
```

*3.1.3 The INSPECT Statement.* The INSPECT statement is similar to the UPDATE statement of SQL and helps editing or wrangling the tables. For attributes $A_i$, it allows the values in these columns to be categorized, missing values predicted, convert categories to continuous values and eliminate duplicate rows. This statement affords MQL the power to manipulate a table to make it suitable for a potential learning task fully autonomously by a smart pre-processing engine. INSPECT returns a table with a scheme of a relation reflective of the resulting table in the FROM clause.

[1] $dom(X)$ is the set of elements in the domain of the column $X$, and $X \in \cup_i A_i$.

```
INSPECT A1 [CATEGORIZE INTO L1, L2, ..., Lx |
IMPUTE | NUMERIZE AS E | DEDUPLICATE],
A2 [CATEGORIZE INTO L1, L2, ..., Lx |
IMPUTE | NUMERIZE AS E | DEDUPLICATE], ...,
An [CATEGORIZE INTO L1, L2, ..., Lx |
IMPUTE | NUMERIZE AS E | DEDUPLICATE]
FROM r1, r2, ..., rn
WHERE c
```

## 3.2 Semantics of MQL

The semantics we assign to each of these statements are system and implementation specific. By that, we mean that each system implementing these statements will play a major role in their meaning, efficiency, accuracy, and performance. For example, if they are implemented in TensorFlow as opposed to PyTorch or R, they will demonstrate different characteristics, i.e., the predictions made by the underlying TensorFlow algorithms could be different from the Pytorch or SciKit-Learn based algorithms, and the prediction accuracies may vary. We consider this aspect of MQL somewhat similar to SQL's optimization strategies. The only difference is that in the case of MQL, it is more about the quality of the predictions or the semantic interpretations of the data. In this paper, we do not address these issues and only focus on the generic semantics we expect from each of these statements.

For illustrative purposes, we use Kaggle's Boston housing market dataset [19] as our example. This data has 506 observations with 13 continuous and 1 binary attribute stored as the file *bostonHomes* (the full list of features and their interpretations can be found in [19]). Many interesting analyses of this dataset by a large number of researchers point to how a smart query processor and optimizer could exploit them to develop processing strategies to meet user needs. Our goal, however, is not to delve into processing strategies or optimization opportunities but to offer these passing comments for interested readers. Instead, we refer readers to [10] for the Python code segment that implements a linear regression model assuming a Pandas DataFrame "**df**" with columns *MEDV, CRIM, ZN, NOX, DIS, TAX,* and *PTRATIO*. It splits the data into training and testing sets, standardizes the features, builds a linear regression model using SciKit-Learn, trains the model on the training set, evaluates it on the test set, and makes predictions. The number of epochs and other hyperparameters can be adjusted by a query processor for the dataset, as needed, to meet any user specified performance threshold. Similar code segments can be generated to implement the CONSTRUCT and INSPECT statements.

*3.2.1 Query Processing.* Compared to SQL databases, ML databases and query processing are likely more nuanced, complex, and require more user involvement in library and algorithm selection, or code customization. Query processing for MQL currently needs additional instructions beyond the Python scripts. A file handler has been implemented to bring data to the MQL store and link with the query processor. The directory path for the Boston housing data in CSV format can be included in the Python code segment or copied into the directory where the code is running.

The MQL query below for the prediction of home values using the Boston housing data can be submitted in command line mode in the MQL engine for execution. In this query, the median home value is being predicted for homes in the file *homesNew* given a subset of features in the set {*CRIM, ZN, NOX, DIS, TAX, PTRATIO*}. In the plot, the *HomeNo* in the file *homesNew* is used as label.

```
GENERATE DISPLAY OF
PREDICTION MEDV
OVER homesNew
LABEL HomeNo
FEATURES CRIM, ZN, NOX, DIS, TAX, PTRATIO
FROM bostonHomes
```

*3.2.2 Results.* We assign translational semantics to the query in Fig **??** by mapping it to the SciKit-Learn program for execution. In this program, we first extract the features ('CRIM', 'ZN', 'NOX', 'DIS', 'TAX', 'PTRATIO') and the target ('MEDV') from the DataFrame. We then split the data into training and testing sets using train_test_split. Next, we create a Linear Regression model and train it on the training data using fit. We then make predictions on the test set using predict, and evaluate the model using mean squared error (mean_squared_error). On execution over the slightly sparse dataset in Fig 1, it produced the plot in Fig 2(b) in which we assumed zero for missing values as shown in Fig 2(a). If imputed values are used for missing values as shown in the code, predictions will be slightly different. The predicted versus actual plot is shown in Fig 2(c).

| HomeNo | CRIM | ZN | NOX | DIS | TAX | PTRATIO |
|---|---|---|---|---|---|---|
| 1 | 0.00632 | 18 | 0.538 | 4.09 | 296 | 15.3 |
| 2 | 0.50031 | 7 | - | 3.20 | 107 | 3.5 |
| 3 | - | 12 | - | 2.78 | 148 | 11.6 |
| 4 | 0.02731 | 0 | 0.469 | - | 242 | - |

**Figure 1: Test data input table *homesNew*.**

# 4 IMPLEMENTATION STRATEGY

We have implemented the MQL statements using SciKit-Learn using Python over CSV datasets. Currently, we only support one table in CSV format in the FROM clause and no WHERE clause condition is allowed[2]. Note that these restrictions are not a limitation of the language and does not affect its expressive power.

A more serious implementation in PostgreSQL using User Defined Functions (UDFs) written in SQL and PL/Python [22, 26] is underway. Once completed, we should be able to compare performance of the current file based and the PostgreSQL based approaches to implementation and comment more on how these choices influence various ML query processing parameters in ways similar to P2D [7] that also takes a similar translational approach. Opportunities also exist to decide system defaults for DISPLAY OF, data wrangling for test data (e.g., missing value imputation), etc.

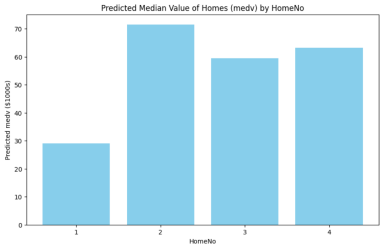
## 4.1 Discussion

In our view, there are not too many declarative ML languages that stand at the same level as MQL. By that we mean, a language that does not require users to express analysis needs using a language more akin to procedural codes. It should be readily noticed that the languages such as Dyna, ML2SQL, sql4ml, and P6 [14] though

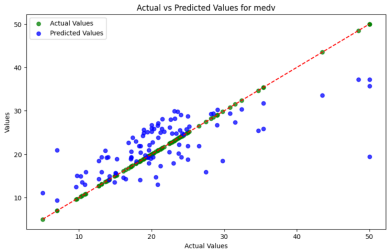
[2]This means that if data has to come from multiple tables, users will need to pre-process and create a single table.

| HomeNo | crim | zn | nox | dis | tax | ptratio | predicted_medv |
|---|---|---|---|---|---|---|---|
| 1 | 0.00632 | 18 | 0.538 | 4.09 | 296 | 15.3 | 29.081795 |
| 2 | 0.50031 | 7 | 0.000 | 3.20 | 107 | 3.5 | 71.541988 |
| 3 | 0.00000 | 12 | 0.000 | 2.78 | 148 | 11.6 | 59.524050 |
| 4 | 0.02731 | 0 | 0.469 | 0.00 | 242 | 0.0 | 63.187736 |

(a) homesNew assumed table.




(b) Bar plot.




(c) Predicted v actual plot.

**Figure 2: Predicted home median values.**

possibly are more powerful and customizable, they are closer to procedural languages such as Python or C++, and thus give an appearance and flavor of imperative programming. The CREATE FUNCTION or the CREATE OPERATOR statements and the elaborate codes in Python or C++ is a significant barrier. In contrast, we hide all procedurality and offer a flavor of SQL like semantics.

We agree with Gleeson [6] and believe that declarativity should be SQL like, even for ML. Gleeson, however, encoded several ML tasks directly in PostgreSQL using ML features supported in it. For example, regression has been coded as follows where the objective is to "learn" the parameters $m$ and $c$ of a linear equation of the form $y = mx + c$ from the training data.

```
WITH regression AS
  (SELECT regr_slope(y, x) AS gradient,
      regr_intercept(y, x) AS intercept
   FROM linear_regression
   WHERE y IS NOT NULL)
SELECT x, (x * gradient) + intercept
  AS prediction
FROM linear_regression CROSS JOIN
  regression
WHERE y IS NULL;
```

In this code segment, the *regr_slope()* and *regr_intercept()* functions are used to estimate the gradient and intercept terms, respectivel corresponding to the equation $y = gradient \times x + intercept$. In MQL, we will express the same functionality as follows, which we are able to execute on any database engine using a front-end.

```
GENERATE DISPLAY OF
PREDICTION y
OVER unknown_xs
FEATURES x, gradient, intercept
FROM linear_regression
```

In the above query, the table *unknown_xs* contains the values $x$ for which $y$ needs to be predicted. The DISPLAY OF option plots a graph to show the $y$ values against each $x$ in *unknown_xs*. Without the DISPLAY OF option, MQL will just compute a table with the columns $x$ and $y$. The difference obviously is, in MQL, users think in a more abstract manner and at a very conceptual level.

## 4.2 Future Improvements

Our current implementation has a few drawbacks that we plan to address in MQL's future editions. The first drawback is a design choice for the first edition of MQL. In this edition, we did not include an option to generate visualization primitives for the CONSTRUCT and INSPECT statements, only GENERATE supports data visualization. But, it is necessary to allow visualization of various relationships during model building, feature selection and data wrangling. We are in the process of designing an extended set of suitable features to support data visualization.

The second limitation of MQL is related to the quality of analysis and query processing performance. There are numerous ML frameworks and a large number of ML algorithms that are suitable for applications on a case by case basis. Therefore, it is imperative that an MQL query optimizer be developed to identify candidate algorithms most relevant to a specific ML task, data set and analytic options. In absence of such an optimizer, MQL is in risk to compromise quality of analysis or performance, or both. We hope to address this limitation soon.

# 5 CONCLUSION

The MQL language we have introduced has several basic strengths and advantages over other similar languages. First, its basic structure is simple and easy to understand. The algorithmic and procedural separation of MQL and its declarative semantics also offers the opportunity for selecting implementation strategies, optimization and system level customization not offered by most contemporary declarative ML languages (the few that we are aware of). Better opportunities for using large language models now emerge to map natural language queries into MQL in ways similar to SQL for a more streamlined execution, instead of mapping to archaic Python codes . While we are contemplating a PostgreSQL implementation of MQL and explore optimization strategies, its current implementation on a file based store serves as a proof of concept and demonstrates its merits. While an extended version of this article is already available [10], detailed description of MQL's implementation will be published elsewhere. A public and open source GitHub repository for this project is also being maintained at https://github.com/hmjamil/mql with the goal of MQL's shared community development.


# ACKNOWLEDGEMENT

This publication was made possible in part by an Institutional Development Award (IDeA) from the National Institute of General Medical Sciences of the National Institutes of Health under Grant #P20GM103408.

# REFERENCES


[1] Houria Abouloifa and Mohamed Bahaj. 2022. Predicting late delivery in Supply chain 4.0 using feature selection: a machine learning model. In *5th International Conference on Advanced Communication Technologies and Networking, CommNet 2022, Marrakech, Morocco, December 12-14, 2022*. IEEE, 1–5.

[2] Aadyot Bhatnagar, Paul Kassianik, Chenghao Liu, Tian Lan, Wenzhuo Yang, Rowan Cassius, Doyen Sahoo, Devansh Arpit, Sri Subramanian, Gerald Woo, Amrita Saha, Arun Kumar Jagota, Gokulakrishnan Gopalakrishnan, Manpreet Singh, K. C. Krithika, Sukumar Maddineni, Dae-ki Cho, Bo Zong, Yingbo Zhou, Caiming Xiong, Silvio Savarese, Steven C. H. Hoi, and Huan Wang. 2023. Merlion: End-to-End Machine Learning for Time Series. *J. Mach. Learn. Res.* 24 (2023), 226:1–226:6.

[3] Matthias Boehm, Iulian Antonov, Sebastian Baunsgaard, Mark Dokter, Robert Ginthör, Kevin Innerebner, Florijan Klezin, Stefanie N. Lindstaedt, Arnab Phani, Benjamin Rath, Berthold Reinwald, Shafaq Siddiqui, and Sebastian Benjamin Wrede. 2020. SystemDS: A Declarative Machine Learning System for the End-to-End Data Science Lifecycle. In *CIDR 2020, Amsterdam, The Netherlands, January 12-15, 2020*.

[4] 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. *Proc. VLDB Endow.* 9, 13 (2016), 1425–1436.

[5] Swarnava Dey, Avik Ghose, and Soumik Das. 2023. Challenges of Accurate and Efficient AutoML. In *IEEE/ACM ASE 2023, Luxembourg, September 11-15, 2023*. IEEE, 1834–1839.

[6] Peter Gleeson. 2018. Machine Learning Directly in SQL – How to Use ML in Databases. https://www.freecodecamp.org/news/machine-learning-directly-in-sql/ Accessed: 3/17/2024.

[7] Yordan Grigorov, Haralampos Gavriilidis, Sergey Redyuk, Kaustubh Beedkar, and Volker Markl. 2023. P2D: A Transpiler Framework for Optimizing Data Science Pipelines. In *Proceedings of the Seventh Workshop on Data Management for End-to-End Machine Learning, DEEM 2023, Seattle, WA, USA, 18 June 2023*. ACM, 3:1–3:4.

[8] William R. Hersh, Jessica Ancker, Robert Hoyt, and Aditi Gupta. 2022. Beyond Wrangling and Modeling: Data Science and Machine Learning Competencies and Curricula for The Rest of Us. In *AMIA 2022, American Medical Informatics Association Annual Symposium, Washington, DC, USA, November 5-9, 2022*. AMIA.

[9] Sohail Hosseini. 2023. Selecting the Best Model for Boston Housing Dataset using Cross-Validation in Python. https://tinyurl.com/29t99tuu Accessed: 3/17/2024.

[10] Hasan M. Jamil. 2024. A Declarative Query Language for Scientific Machine Learning. *CoRR* abs/2405.16159 (2024).

[11] Tim Kraska, Ameet Talwalkar, John C. Duchi, Rean Griffith, Michael J. Franklin, and Michael I. Jordan. 2013. MLbase: A Distributed Machine-learning System. In *CIDR 2013, Asilomar, CA, USA, January 6-9, 2013*. www.cidrdb.org.

[12] Ruihang Lai, Junru Shao, Siyuan Feng, Steven S. Lyubomirsky, Bohan Hou, Wuwei Lin, Zihao Ye, Hongyi Jin, Yuchen Jin, Jiawei Liu, Lesheng Jin, Yaxing Cai, Ziheng Jiang, Yong Wu, Sunghyun Park, Prakalp Srivastava, Jared G. Roesch, Todd C. Mowry, and Tianqi Chen. 2023. Relax: Composable Abstractions for End-to-End Dynamic Machine Learning. *CoRR* abs/2311.02103 (2023).

[13] Van-Duc Le. 2023. VeML: An End-to-End Machine Learning Lifecycle for Large-scale and High-dimensional Data. *CoRR* abs/2304.13037 (2023).

[14] Jianping Kelvin Li and Kwan-Liu Ma. 2021. P6: A Declarative Language for Integrating Machine Learning in Visual Analytics. *IEEE Trans. Vis. Comput. Graph.* 27, 2 (2021), 380–389. https://doi.org/10.1109/TVCG.2020.3030453

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

[16] Nantia Makrynioti, Ruy Ley-Wild, and Vasilis Vassalos. 2019. sql4ml A declarative end-to-end workflow for machine learning. *CoRR* abs/1907.12415 (2019).

[17] James D. McCaffrey. 2021. Ordinal Classification for the Boston Housing Dataset Using PyTorch. https://tinyurl.com/5xxy525h Accessed: 3/17/2024.

[18] Piero Molino and Christopher Ré. 2021. Declarative Machine Learning Systems: The future of machine learning will depend on it being in the hands of the rest of us. *ACM Queue* 19, 3 (2021), 46–76.

[19] Prasad Perera. 2018. The Boston Housing Dataset. https://www.kaggle.com/code/prasadperera/the-boston-housing-dataset Accessed: 3/17/2024.

[20] Nisha Pillai, Athish Ram Das, Moses Ayoola, Ganga Gireesan, Bindu Nanduri, and Mahalingam Ramkumar. 2024. EndToEndML: An Open-Source End-to-End Pipeline for Machine Learning Applications. *CoRR* abs/2403.18203 (2024).

[21] Tommi Ranta. 2018. Clustering of the Boston Housing Dataset. https://github.com/tommiranta/data-science-blog/blob/master/python_in_powerbi/boston_housing.ipynb Accessed: 3/17/2024.

[22] Timofey Rechkalov and Mikhail L. Zymbler. 2017. An Approach to Data Mining inside PostgreSQL Based on Parallel Implementation of UDFs. In *Selected Papers of the XIX International Conference on Data Analytics and Management in Data Intensive Domains (DAMDID/RCDL 2017), Moscow, Russia, October 9-13, 2017 (CEUR Workshop Proceedings)*, Leonid A. Kalinichenko, Yannis Manolopoulos, Nikolay A. Skvortsov, and Vladimir Sukhomlin (Eds.), Vol. 2022. CEUR-WS.org, 114–121.

[23] Shubhra Kanti Karmaker Santu, Md. Mahadi Hassan, Micah J. Smith, Lei Xu, Chengxiang Zhai, and Kalyan Veeramachaneni. 2022. AutoML to Date and Beyond: Challenges and Opportunities. *ACM Comput. Surv.* 54, 8 (2022), 175:1–175:36.

[24] Maximilian E. Schüle, Matthias Bungeroth, Alfons Kemper, Stephan Günnemann, and Thomas Neumann. 2019. MLearn: A Declarative Machine Learning Language for Database Systems. In *Proceedings of the 3rd International Workshop on Data Management for End-to-End Machine Learning, DEEM@SIGMOD 2019, Amsterdam, The Netherlands, June 30, 2019*, Sebastian Schelter, Neoklis Polyzotis, Stephan Seufert, and Manasi Vartak (Eds.). ACM, 7:1–7:4.

[25] Maximilian E. 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 *Advances in Database Technology - 22nd International Conference on Extending Database Technology, EDBT 2019, Lisbon, Portugal, March 26-29, 2019*, Melanie Herschel, Helena Galhardas, Berthold Reinwald, Irini Fundulaki, Carsten Binnig, and Zoi Kaoudi (Eds.). OpenProceedings.org, 562–565.

[26] Maximilian E. Schüle, Jakob Huber, Alfons Kemper, and Thomas Neumann. 2020. Freedom for the SQL-Lambda: Just-in-Time-Compiling User-Injected Functions in PostgreSQL. In *SSDBM 2020: 32nd International Conference on Scientific and Statistical Database Management, Vienna, Austria, July 7-9, 2020*, Elaheh Pourabbas, Dimitris Sacharidis, Kurt Stockinger, and Thanasis Vergoulis (Eds.). ACM, 6:1–6:12.

[27] Sameer Singh, Tim Rocktäschel, Luke Hewitt, Jason Naradowsky, and Sebastian Riedel. 2015. WOLFE: An NLP-friendly Declarative Machine Learning Stack. In *NAACL HLT 2015, The 2015 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Denver, Colorado, USA, May 31 - June 5, 2015*, Rada Mihalcea, Joyce Yue Chai, and Anoop Sarkar (Eds.). The Association for Computational Linguistics, 61–65.

[28] Daniel S. Soper. 2022. Rapid Selection of Machine Learning Models Using Greedy Cross Validation. In *55th Hawaii International Conference on System Sciences, HICSS 2022, Virtual Event / Maui, Hawaii, USA, January 4-7, 2022*. ScholarSpace, 1–10.

[29] Alexander Tornede, Difan Deng, Theresa Eimer, Joseph Giovanelli, Aditya Mohan, Tim Ruhkopf, Sarah Segel, Daphne Theodorakopoulos, Tanja Tornede, Henning Wachsmuth, and Marius Lindauer. 2023. AutoML in the Age of Large Language Models: Current Challenges, Future Opportunities and Risks. *CoRR* abs/2306.08107 (2023).

[30] Shivakumar Vaithyanathan. 2011. The Power of Declarative Languages: From Information Extraction to Machine Learning. In *Datenbanksysteme für Business, Technologie und Web (BTW), 14. Fachtagung des GI-Fachbereichs "Datenbanken und Informationssysteme" (DBIS), 2.-4.3.2011 in Kaiserslautern, Germany (LNI)*, Theo Härder, Wolfgang Lehner, Bernhard Mitschang, Harald Schöning, and Holger Schwarz (Eds.), Vol. P-180. GI, 23.

[31] Tim Vieira, Matthew Francis-Landau, Nathaniel Wesley Filardo, Farzad Khorasani, and Jason Eisner. 2017. Dyna: toward a self-optimizing declarative language for machine learning applications. In *Proceedings of the 1st ACM SIGPLAN International Workshop on Machine Learning and Programming Languages, MAPL@PLDI 2017, Barcelona, Spain, June 18, 2017*, Tatiana Shpeisman and Justin Gottschlich (Eds.). ACM, 8–17.

[32] Jin Wang, Jiacheng Wu, Mingda Li, Jiaqi Gu, Ariyam Das, and Carlo Zaniolo. 2021. Formal semantics and high performance in declarative machine learning using Datalog. *VLDB J.* 30, 5 (2021), 859–881.