Integrating Machine Learning Model Ensembles to the SAVIME Database System

Anderson Silva[1], **Patrick Valduriez**[2], **Fabio Porto**[1]

[1]Laboratório Nacional de Computação Científica (LNCC)
Petrópolis, Rio de Janeiro, Brazil

{achaves,fporto}@lncc.br

[2]Inria, University of Montpellier, CNRS, LIRMM
Montpellier, France

patrick.valduriez@inria.fr

Abstract. *The integration of machine learning algorithms into database systems has brought new opportunities in different areas from indexing to query optimization. In this paper, we describe the integration of an approach for the automatic computation of model ensembles to answer a predictive query. We have extended the SAVIME multi-dimensional array DBMS by adding a new function to its query language and implementing the selection and allocation ensemble model dataflow into the query processing component of SAVIME. We show some initial experimental results depicting its performance against a pure Python implementation of the ensemble approach. Interestingly enough the C++ implementation within SAVIME is up to 4 times faster than its competitor.*

1. Introduction

The adoption of Machine Learning (ML) models in replace of different database management techniques has been a hot research topic since the appearance of the *Learned Index* paper by Kraska [Kraska et al. 2018]. Since then, a number of different applications of machine learning in databases has been proposed, such as learning cardinality estimation [Kim et al. 2022], a long time difficult problem to be solved algorithmically, as well as join ordering selection and query optimization [Marcus et al. 2021], just to name a few. Another area of intense activity, strongly pushed by the database systems industry, is the integration of machine learning models as predictive functions receiving input from query expressions [Fard et al. 2020][Jasny et al. 2020]. There has also been strong interest in the investigation of the performance of training machine learning models in the database, taking advantage of years of efficient data processing, specially for large input training sets [Sandha et al. 2019].

In this paper, we present initial results on the integration of machine learning models ensembles into the SAVIME database system [L. S. Lustosa et al. 2021]. We consider the DJEnsemble approach [Pereira et al. 2021] that combines a set of spatio-temporal deep learning models automatically selected by a cost-based model. As in other works, we argue that database systems already offer a declarative query language to which model invocation can be easily integrated, as in SQL user defined functions [Duta and Grust 2020]. Once added to a query plan, all data transformation needed when preparing the model input can be specified as a query expression placing ML model inference as part of an efficient query processing framework. Moreover, we observe that the multidimensional array model of SAVIME is particularly suited to model 3D input tensors, as required by deep learning input data. We describe the decisions taken during the design of the integration of DJEnsemble into the SAVIME system and some challenges still remaining to be solved. We highlight that by adding DJEnsemble to SAVIME, or any other DBMS, its model selection and allocation problem can be solved as part of the query optimization process.

We have run some initial experiments highlighting the overhead of invoking the model integrated into SAVIME against its execution on pure ML engine/programming language combination. The rest of this paper is organized as follows. Section 2 presents the original DJEnsemble approach and the SAVIME DBMS system. Next, in Section 3, we describe the implementation of the ensemble approach into SAVIME as a new query operator. Section 4 presents some initial experimental results, and Section 5 discusses previous works that intended to integrate ML into database systems. Finally, Section 6 presents some final remarks and points to future work.

2. Preliminaries

In this section we provide background on the DJEnsemble ensemble approach and on the SAVIME multidimensional array database system.

2.1. DJEnsemble Approach

The machine learning ensemble approach is a technique to improve the accuracy of predictions by using multiple models for the same prediction task and applying a linear function to combine their results. The assumption is that by combining different models, the weaknesses of each one are compensated by the strengths of the others. However, DJEnsemble takes a slightly different approach [Pereira et al. 2021]. As the traditional ensemble approach, it considers a set of available trained models $M = \{M_1, M_2, \ldots, M_n\}$ for the prediction of a spatio-temporal variable, i.e. a spatial time series. Examples of spatio-temporal variables are *temperature* and *precipitation*. However, instead of invoking all available models to compute a prediction, DJEnsemble selects the best models to be applied to each data subdomain. We consider a domain the dataset composed of time series of a variable. For instance, the domain of the precipitation variable comprises the dataset of time series of a region. In this context subdomains characterize regions sharing similar precipitation behavior.

DJEnsemble is applicable to answer spatio-temporal queries (SPTQ), which are expressed as $Q =< R, I, V, t >$, where R is a spatio-temporal region where predictions are desired, for instance the region of the Rio de Janeiro city in January 2022, I is a tensor of input to the model, V is the predicted variable, for instance, precipitation, and t is the number of time instances ahead for the prediction. The approach is integrated into the SAVIME system (see section 2.2) in two parts: offline and online. The *offline stage* structures the domain dataset into clusters and tiles. The former finds the time series sharing similar behaviors in time, whereas the latter structures the dataset domain into regular tiles i.e. rectangles with a high percentage of time-series of a single cluster. Lastly, to each tile, we associate its representative, which is a time series elected as the medoid of the cluster bearing the majority of time series in the corresponding tile.

Figure 1 illustrates the process described. The image at the left shows the clusters of a spatial domain containing precipitation readings, discretized in 7 x 7 regions. An analysis using silhouette clustering index [Rousseeuw 1987] indicates 2 clusters. At right, the results of the tiling process applied over the clustered spatio-temporal domain are illustrated. At each iteration of the tiling algorithm, a position is chosen as the starting point of a new tile. The tile area is then extended to the right and bottom, as long as the number of regions from different clusters in the tile is not exceeded.

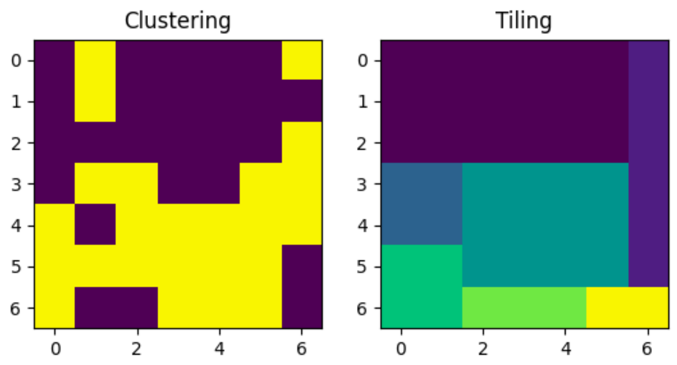


Figure 1. DJEnsemble clustering and tiling identified by the offline stage

The online stage is when the query processing happens. It considers a query Q with a region $Q.R$, with Q.R representing a 3D region of a spatio-temporal tensor of *precipitation* data. The input $Q.I$ is the result of a spatio-temporal query expression. The dataflow in Figure 2 depicts the various stages of the DJEnsemble online execution. We identify tasks into colours blue(B), orange (O) and blue-orange (BO).

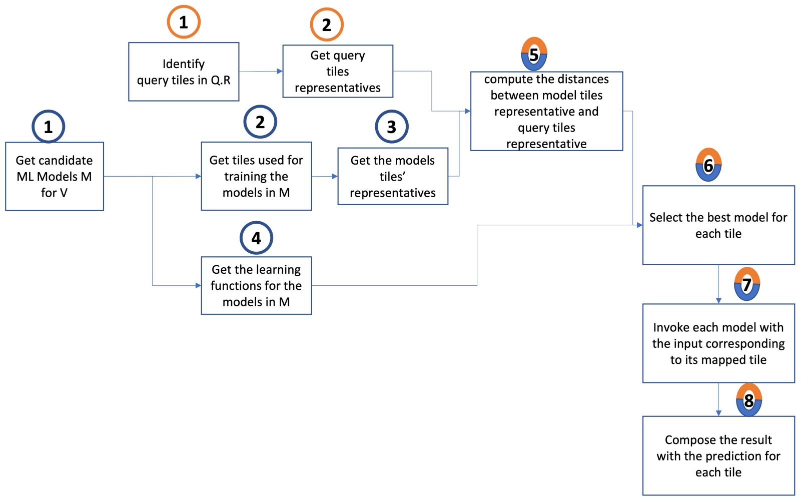

1
Identify query tiles in Q.R
2
Get query tiles representatives
5
compute the distances between model tiles representative and query tiles representative
1
Get candidate ML Models M for V
2
Get tiles used for training the models in M
3
Get the models tiles' representatives
6
Select the best model for each tile
4
Get the learning functions for the models in M
7
Invoke each model with the input corresponding to its mapped tile
8
Compose the result with the prediction for each tile

Figure 2. Dataflow for the execution of DJEnsemble

- B1: obtain from the catalog a set of trained models for forecasting a variable V;
- B2: extract the model metadata informing the data used in training, identified by the corresponding tiles;
- B3: obtain the representative for each tile identified in B2
- B4: obtain the learning function (LF) for the tile. A LF is a model that given a distance predicts a model error;

- O1: obtain the set of the domain tiles whose spatial volume intersects with that of the the query Q region R;
- O2: obtain the tiles representatives;
- BO5: for each query tile, compute its representative distance to the tiles representative of the candidate models;
- BO6: use the models learning functions to compute their estimated error for each query tile and associate to each tile the model with the least estimated error
- BO7: invoke each of the selected models having as input the part of I corresponding to its tile
- BO8: compose the predictions from each model into a single prediction volume.

The step BO6 runs an optimization weighted cost function that takes into account the error estimate computed by the learning function and the estimated execution cost. Thus, at the end of this process an ensemble of models has been selected and run for computing the predictions as requested by the query Q.

2.2. SAVIME Array DBMS

The SAVIME multidimensional array database system is a NOSQL system developed at the DEXL Laboratory [Lustosa 2020]. SAVIME logical data representation is modeled by a Typed Array (TAR) data model. A TAR schema (TARS) specifies the arrays in a database. A TAR definition holds a name that uniquely identifies the array, and counts with two main data elements for an array definition: dimensions and attributes. There is no restriction on the number of dimensions an array may specify. A typical dimension is defined as $d = ([1 : n], step)$, where the interval indicates integer indexing from 1 to n, with integer *steps* between index values. For a k dimensional array, a cell is specified as the array position indicated by a vector of dimensions $v =< v_1, v_2, \ldots, v_k >$, where v_i is an index of dimension i, for all $1 \leq i \leq k$. A cell may hold a tuple of attributes of simple types, such as: integer, float, double etc. SAVIME implements a functional query language, inspired by AFL, one of the query languages adopted by the SciDB system [Brown 2010]. A function in SAVIME is implemented by an operator that consumes TAR objects as input, and produce a single TAR array as output. The list of current available operators in SAVIME is depicted in Table 1.

Query optimization in SAVIME adopts a generalization of push down selection approach, pushing down all operations that reduce the size of arrays: Where, Subset, Select and Aggregate. The *Predict* operation specifies ML models to be executed with inputs produced by inner operations of a query expression. During optimization, the *predict* operation is pulled up in the physical query plan. A TAR is redefined by one or multiple SubTARs. The latter is an irregular partitioning of a TAR, such that the union of all SubTARs $S = \{S_1, S_2, ..., S_n\}$ reproduces in definition and in content the corresponding TAR T. The SubTARs are the unit of processing of an operator. SAVIME uses the SubTAR for parallelizing the computation of operators. Considering system characteristics, SAVIME is an in-memory system that considers arrays to be allocated in main memory for query processing. Moreover, the system reads data in raw file format, not requiring an ingestion phase for processing it. Another interesting feature is its integration with the Python language, through the Pysavime library. TAR arrays returned by queries in SAVIME are transformed into Numpy arrays for processing using the scripting programming language.

Operator	Description
SELECT	Projects dimensions and attributes
WHERE	Filters data according to a predicate on dimensions and attributes
SUBSET	Creates a n-dimensional data slice according to specified bounds
DERIVE	Adds an attribute with derived values
CROSS_JOIN	Implements a cartesian product of cells in TARs
DIM_JOIN	Implements an equi-join by corresponding dimensions of two TARs
AGGREGATE	Summarizes data according to a subset of dimensions and applying a common aggregate function
PREDICT	Invokes a single ML model with input from a TAR

Table 1. SAVIME query language operators

3. Integrating DJEnsemble in SAVIME

We present the integration of DJEnsemble into SAVIME in order to support spatio-temporal queries execution. To execute an inductive query, SAVIME makes use of a predictive engine module external to the system based on tensorflow server [Baylor et al. 2017]. This module allows SAVIME to answer predictive queries in general.

Currently, SAVIME is capable of executing the entire online stage of DJEnsemble within the system, through the ENSEMBLE operator built into the query language. The syntax of ENSEMBLE operator is:

```
ENSEMBLE(<tar>, <query-region>, predictive-variable)
```

Figure 3 illustrates how this process is achieved. The results of the offline stage must be previously registered in the system through metadata files, namely: the tiling resulting of the time series categorization process and the error estimation function for each predictive model, along with the time series that best represents the models training data. Once a query is performed, SAVIME executes the error estimation functions in order to choose the best candidate models to answer the query. The error estimated to each model for each tile intersecting with the query region is based on the distance between its representative time series and each model's representative sequences. We utilize Dynamic Time Warping to calculate the distances between sequences. The composition of each tile paired with the model with the least estimated cost for it constitutes the query model ensemble, which can be finally executed to return the result to the user.

It is worth mentioning that the Ensemble operator is part of SAVIME's DML, and thus the operator's input TAR can be resulting from a previous executed nested query. Similarly, to make use of the full potential of SAVIME's query language, the data returned by Ensemble can also be chained together as input to new queries.

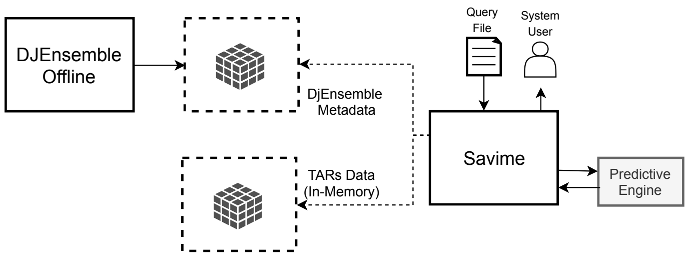


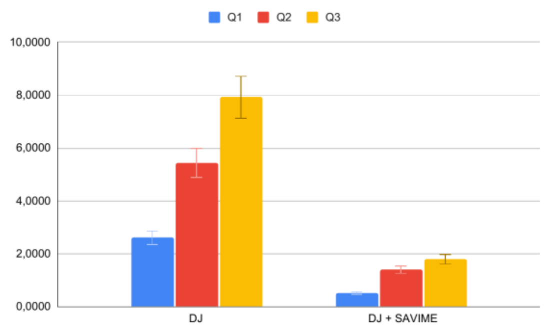
Figure 3. Overview of DJEnsemble integration into SAVIME

4. Experimental Evaluation

To analyze the algorithm performance when integrated to SAVIME, we performed a series of experiments evaluating the execution time of the different steps. For reference, we also measured the offline step execution time (already presented in [Pereira et al. 2021]). For our experiments, we built a dataset from rain data from the city of Rio de Janeiro, provided by 33 pluviometrical stations. The constructed dataset represents a 7 x 7 mesh over the city, containing the precipitation values along 20 years, recorded at 15-minute intervals. The experiments were run using the hardware configurations: Intel Core i5-5200U CPU @ 2.20GHz and 8 GB RAM.

We synthetically partitioned the obtained data into 3 tiles, representing 3 different regions of the city. We also built 3 Convolutional Long Short-Term Memory (ConvLSTM) neural network model models, with input dimensions of 3x3, 4x4 and 7x7. The training data for each model and corresponding representing sequences were defined in such a way that the estimated error for each tile would be optimal by choosing a different model by the algorithm.

Based on the described experimental scenario, we submitted 3 different predictive queries to SAVIME, Q1, Q2 and Q3, using the Ensemble operator. In order to evaluate the runtime variation as the number of tiles increases, each query is designed so that it would intersect with one, two or three tiles of the domain. For comparison purposes, we performed a test with the same data under the DJEnsemble version purely written in Python, not integrated to SAVIME.



	DJEnsemble	SAVIME
Candidate Models	3	3
Intersecting Query Tiles	3	3
Tile Execution Time	2,6240	0,6033
Model Execution Time	0,0612	0,0150
Total Ex. Time	7,9160	1, 8001

Figure 4. Comparative specification and results between DJEnsemble and SAVIME. All times refer to average in seconds

Total results of the experiments can be seen in Figure 4. Note that the relation is not linear, because two model executions are necessary for the tile 2 model, differently than others.

5. Related Work

The integration of machine learning algorithms into database systems initiated with the now famous *Learning Index* [Kraska et al. 2018] paper by Tim Kraska. In that paper Kraska and colleagues showed that a B+tree could be substituted by a learned model. Since then, other areas of the database systems have been explored as opportunities for ML algorithms. Computing cardinalities of operations has always been a difficult task that is at the heart of query optimization. Many authors have been working in learned techniques modeling it as a regression problem [Dutt et al. 2019][Woltmann et al. 2019]. A second approach involves applying ML models to data in databases. In this context, ML algorithm are integrated to the system that may support ML models training and inferencing [Fard et al. 2020]. Model inferencing is integrated to the system inheriting the existing code to support user-defined function execution. Other systems such as PostgreSQL [Pos 2022] and Greenplum [gre 2022] have added the support to the ML library *MADLib*. The work we describe in this paper considers the integration of ML models for inferencing and uses an external ML engine for model execution.

6. Conclusion

Machine learning algorithms have shown concrete results in a myriad of applications. In the database system this has not been different. In this paper, we describe a new opportunity for applying machine learning predictions by adding model ensembles as part of a query specification. We integrate the DJEnsemble approach to the SAVIME query engine. The approach automatically selects and allocates deep learning models to solve a spatio-temporal predictive query. Our initial results show that the C++ implementation of DJEnsemble in SAVIME is up to 4 times faster when compared against a typical data science Python implementation. There are a number of opportunities for future work, including integrating the cost function of DJEnsemble with the SAVIME query optimizer and improving models' results composition algorithm.

References

(2022). Greenplum. `https://greenplum.org/`. [Online; accessed 20-July-2022].

(2022). PostgreSQL. `https://www.postgresql.org/`. [Online; accessed 20-July-2022].

Baylor, D., Breck, E., Cheng, H.-T., Fiedel, N., Foo, C. Y., Haque, Z., Haykal, S., Ispir, M., Jain, V., Koc, L., et al. (2017). Tfx: A tensorflow-based production-scale machine learning platform. In *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 1387–1395, Nova Scotia, Canada. Association for Computing Machinery.

Brown, P. G. (2010). Overview of scidb: Large scale array storage, processing and analysis. In *Proceedings of the 2010 ACM SIGMOD International Conference on Management of Data*, SIGMOD '10, page 963–968, New York, NY, USA. Association for Computing Machinery.

Duta, C. and Grust, T. (2020). Functional-style SQL UDFs with a capital 'F'. In *Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data*, SIGMOD '20, page 1273–1287, New York, NY, USA. Association for Computing Machinery.

Dutt, A., Wang, C., Nazi, A., Kandula, S., Narasayya, V., and Chaudhuri, S. (2019). Selectivity estimation for range predicates using lightweight models. *Proc. VLDB Endow.*, 12(9):1044–1057.

Fard, A., Le, A., Larionov, G., Dhillon, W., and Bear, C. (2020). Vertica-ML: Distributed machine learning in Vertica database. In *Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data*, SIGMOD '20, page 755–768, New York, NY, USA. Association for Computing Machinery.

Jasny, M., Ziegler, T., Kraska, T., Roehm, U., and Binnig, C. (2020). DB4ML - An in-memory database kernel with machine learning support. In *Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data*, SIGMOD '20, page 159–173, New York, NY, USA. Association for Computing Machinery.

Kim, K., Jung, J., Seo, I., Han, W.-S., Choi, K., and Chong, J. (2022). Learned cardinality estimation: An in-depth study. In *Proceedings of the 2022 International Conference on Management of Data*, SIGMOD '22, page 1214–1227, New York, NY, USA. Association for Computing Machinery.

Kraska, T., Beutel, A., Chi, E. H., Dean, J., and Polyzotis, N. (2018). The case for learned index structures. In *Proceedings of the 2018 International Conference on Management of Data*, SIGMOD '18, page 489–504, New York, NY, USA. Association for Computing Machinery.

L. S. Lustosa, H., C. Silva, A., N. R. da Silva, D., Valduriez, P., and Porto, F. (2021). SAVIME: An array DBMS for simulation analysis and ML models prediction. *Journal of Information and Data Management*, 11(3).

Lustosa, H. (2020). *SAVIME:Enabling Declarative Array Processing in Memory*. PhD thesis, National Laboratory of Scientific Computing.

Marcus, R., Negi, P., Mao, H., Tatbul, N., Alizadeh, M., and Kraska, T. (2021). Bao: Making learned query optimization practical. In *Proceedings of the 2021 International Conference on Management of Data*, SIGMOD '21, page 1275–1288, New York, NY, USA. Association for Computing Machinery.

Pereira, R., Souto, Y., Chaves, A., Zorilla, R., Tsan, B., Rusu, F., Ogasawara, E., Ziviani, A., and Porto, F. (2021). Djensemble: A cost-based selection and allocation of a disjoint ensemble of spatio-temporal models. In *33rd International Conference on Scientific and Statistical Database Management*, SSDBM 2021, page 226–231, New York, NY, USA. Association for Computing Machinery.

Rousseeuw, P. J. (1987). Silhouettes: a graphical aid to the interpretation and validation of cluster analysis. *Journal of computational and applied mathematics*, 20:53–65.

Sandha, S. S., Cabrera, W., Al-Kateb, M., Nair, S., and Srivastava, M. (2019). In-database distributed machine learning: Demonstration using teradata SQL engine. *Proc. VLDB Endow.*, 12(12):1854–1857.

Woltmann, L., Hartmann, C., Thiele, M., Habich, D., and Lehner, W. (2019). Cardinality estimation with local deep learning models. In *Proceedings of the Second International Workshop on Exploiting Artificial Intelligence Techniques for Data Management*, aiDM '19, New York, NY, USA. Association for Computing Machinery.