A Different VIM: Visualizing Incremental Machine Learning

Sikder Tahsin Al-Amin
University of Houston
Houston, USA

Mohammad Imtiaz Nur
University of Houston
Houston, USA

Aisha M. Farooque
University of Houston
Houston, USA

Guoning Chen
University of Houston
Houston, USA

Robin Varghese
University of Houston
Houston, USA

Carlos Ordonez
University of Houston
Houston, USA

ABSTRACT

Incremental learning is used to continuously update and tune an existing model, as more data points come in. However, there is a tradeoff between speed and accuracy as the model becomes stable. From a big data angle, computing machine learning models is challenging when data sets cannot fit in main memory or when they exceed CPU capacity. On the other hand, data summarization is a fundamental technique that has promise to accelerate data science computations and compress a data set. Keeping these motivations in mind, we present an innovative system, VIM, that computes machine learning models in an incremental manner, visualizing continuous learning of model parameters as the data set is scanned. Our system is fast, it works for a wide spectrum of machine learning models and it can handle data sets larger than main memory. We developed an intuitive GUI which: (1) guides the user to upload data sets and choose a machine learning model, (2) offers interactive visualization of model parameters, and (3) helps getting an approximate model, stopping early, without reading the whole data set.

CCS CONCEPTS

• **Information systems** → **Data mining**.

KEYWORDS

Machine Learning Model; Matrix; Incremental algorithm

1 INTRODUCTION

With the advancement of big data, machine learning model nowadays are usually trained with a large amount of data. In many practical scenarios, the samples in the data set can change over time mostly due to the addition of new data samples and removal of existing data samples. It is still computationally intensive to recompute the model parameters on the entire data set when only a few samples in the data set change. Rather, it is smarter and more efficient to update the model by including or excluding the influence of specific data samples, which is known as incremental and decremental learning [4], [10]. In our work, we focus on the computation of incremental machine models in a single machine when the data set size is large. On the other hand, accelerating machine learning algorithms does not always mean adding hardware and more memory. Therefore, processing and analyzing a large volume of data, specially when it is larger than the main memory, becomes infeasible with a traditional serial approach. Data summarization has been a standard mechanism to accelerate the computation of machine learning models [1], [2]. Our work is inspired by previous research extending DBMSs with machine learning computation with SQL queries and UDFs [6–8].

Our contributions are the following: (1) We present a system that can compute several incremental machine models using data summarization for large data sets in a single machine, (2) We visualize the learning of the model parameters as the data is being summarized and model is being computed. (3) Our solution can get an approximate model faster than the built-in Python incremental library. (4) We provide a GUI to help the user navigate our system and display our results. In this work, we focus on the incremental computation of three popular and fundamental machine learning models: Linear Regression (LR) [5], Principal Component Analysis (PCA) [5], and Naïve Bayes (NB). The models were solved by computing the summarization matrix first while we read the data set in blocks and compute the models from the summarization matrix for each block. From a system angle, our system can show the learning of the model parameters as we read through the data set. Also, the system can show the evaluation metrics on the test data to help determine along with the model parameters to see if the model has been saturated.

2 SYSTEM OVERVIEW

2.1 Definitions

The input matrix is defined as X which is a set of n column-vectors. Θ is used to represent a machine learning model. All the models take a $d \times n$ matrix X as input. Let the input data set be defined as $X = \{x_1, ..., x_n\}$ with n points, where each point x_i is a column-vector in $\mathbf{R}^d$. Intuitively, X is a wide rectangular matrix with d columns. We augment X with an output variable Y, making X a $(d+1) \times n$ matrix, called $\mathbf{X}$. Thereafter, we augment $\mathbf{X}$ with an extra row of n 1s, resulting in a new matrix called Z, with $(d+2) \times n$ size.

2.2 Theory and Algorithm

Here, we first review the powerful summarization matrix (Γ) [3, 9] and the computation of several ML models (Θ), based on Γ. The previous algorithm had two independent phases [9]. In contrast, our incremental algorithm efficiently computes these two phases together, interleaving them, at each iteration, for each block of data.

(1) Phase 1: Compute summarization matrix: one matrix Γ or k matrices Γ^k.
(2) Phase 2: Compute ML model Θ based on Gamma matrix (matrices).

CIKM '22, October 17–21, 2022, Atlanta, GA, USA
2022. ACM ISBN 978-x-xxxx-xxxx-x/YY/MM…$15.00
https://doi.org/10.1145/nnnnnnn.nnnnnnn

2.2.1 Phase 1: We consider X as the input data set, n counts the total number of points in the dataset, L is the linear sum of x_i, and Q is the sum of vector outer products of x_i, then from [3], the Gamma (Γ) is defined below in Eq. 2. The Γ matrix can be computed in the two ways: (1) matrix-matrix multiplication i.e., $Z \cdot Z^T$ (2) sum of vector outer products i.e., $\sum_i z_i \cdot z_i^T$. Here, we evaluate the later one.

$$n = |X|;\ L = \sum_{i=1}^{n} x_i;\ Q = XX^T = \sum_{i=1}^{n} x_i \cdot x_i^T. \tag{1}$$

$$\Gamma = \begin{bmatrix} n & L^T & \mathbf{1}^T \cdot Y^T \\ L & Q & XY^T \\ Y \cdot \mathbf{1} & YX^T & YY^T \end{bmatrix} \tag{2}$$

Now, as explained in [3], we need k-Gamma (Γ^k) for classification/clustering models which is given in Eq. 3 (k=number of classes/clusters). Here, we need only a few parameters, $n, L, L^T, diag(Q)$. Both L and $diag(Q)$ can be represented as a single vector and we do not need to store Q as a matrix. Hence, the k-Gamma matrix can be represented as a single matrix of size $d \times 2k$, where each Gamma is represented in two columns (L and Q). We still need to store the value of n in a row, which makes the k-Gamma $(d+1) \times 2k$.

$$\Gamma^k = \begin{bmatrix} n_1 & 0 & \dots & n_k & 0 \\ L_{1_1} & Q_{11_1} & \dots & L_{1_k} & Q_{11_k} \\ .. & .. & \dots & .. & .. \\ L_{d_1} & Q_{dd_1} & \dots & L_{d_k} & Q_{dd_k} \end{bmatrix} \tag{3}$$

2.2.2 Phase 2: Both Γ and Γ^k provide summarization for a different set of machine learning models (Θ). We briefly discuss how to compute each model (Θ) below. The details of the model computation are discussed in [2].

LR:. We can get the column vector of regression coefficients ($\hat{\beta}$), from the above mentioned Γ with:

$$\hat{\beta} = Q^{-1}(XY^T) \tag{4}$$

Eq. 4 will yield a $d \times 1$ matrix, the regression coefficients. To get the intercept along with it, we need to modify Eq. 4 as: $\hat{\beta} = \tilde{Q}^{-1}(\mathbf{X}Y^T)$, where $\tilde{Q}$ is a $(d+1) \times (d+1)$ matrix.

PCA:. PCA can be computed on the covariance matrix (V), or the correlation matrix (ρ). We compute ρ, the correlation matrix as $\rho = UD^2U^T = (UD^2U^T)^T$. We can also compute the covariance matrix as $V = Q/n - LL^T/n^2$. Then we compute PCA from the ρ by solving the Singular Value Decomposition on it ($svd(\rho)$). We express ρ in terms of sufficient statistics in Eq. 5, where a, b are indexes/subscripts (e.g. the term Q_{ab} represents a particular entry like $Q[a, b]$).

$$\rho_{ab} = \frac{(nQ_{ab} - L_aL_b)}{\sqrt{nQ_{aa} - L_a^2}\sqrt{nQ_{bb} - L_b^2}} \tag{5}$$

.

NB:. Here, we need the k-Gamma matrix. We assume, there are k number of classes (where $g = 1, .., k$), and for each class (g), we compute n_g, L_g, Q_g as discussed in Phase 1. The output of NB model is three parameters: mean (C), variance (R), and the prior probabilities (W). We can compute these parameters from the Γ^k matrix for each class label ($g = 1, .., k$) with the following statistical equations.

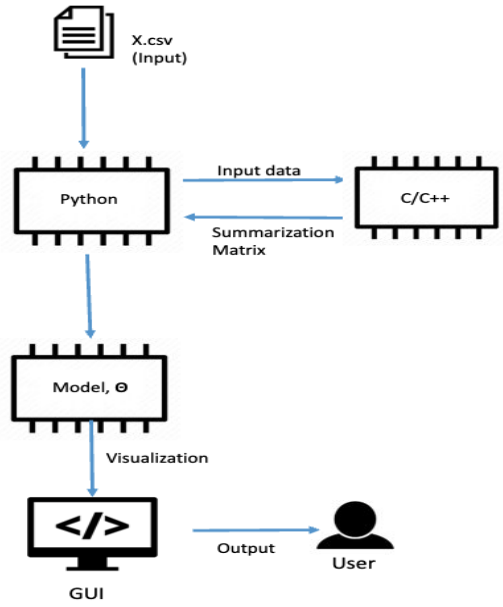

Figure 1: System architecture and data flow.

$$W_g = \frac{n_g}{n};\ C_g = \frac{L_g}{n_g};\ R_g = \frac{Q_g}{n_g} - diag\frac{[L_gL_g^T]}{n_g^2} \tag{6}$$

2.2.3 Integration with Python: The input data set X is read into m blocks ($m \ll n$) of the same size ($|m| > 1$), $X = \{X_1, X_2, .., X_m\}$. We read each block ($X_b$) into the main memory and compute the summarization matrix (Γ_Δ or Γ_Δ^k) for that block as mentioned in Phase 1. This partial Gamma (Γ_Δ or Γ_Δ^k) is added to the Gamma computed up to the previous block (X_{b-1}) and we get the Gamma (Γ or Γ^k) for up to b^{th} block. This phase 1 is comptued in C/C++ as the sum of vector outer products ($z_i * z_i^T$) can be computed block by block efficiently in C++. Computing this operation using traditional loops in Python is slow, usually one row at a time. Based on the summarization matrix computed here, we compute the machine learning models in Phase 2 as discussed earlier. We compute this Phase 2 in Python. Reprogramming all the models in C++ is unnecessary and error-prone. Since our model computation requires just changing certain steps in the numerical method, we rewrite the equations based on the data summaries and program them in the data science language efficiently. To call C++ from Python we use the Python SWIG library. However, our solution can be integrated with any language that supports API calls to C/C++.

2.3 System Architecture

We assume the user needs to compute the machine learning models quickly and accurately. We assume visualizing the learning of model parameters can help them understand how the model is behaving, when to stop the model (if needed), and satisfy their needs. Our system offers incremental machine learning algorithms for large

datasets as a service and provides a graphical visualization that will be easily understandable to the user. Our system splits the work among Python and C++.

Fig 1 shows the data flow diagram of our proposed system. Our system has three main parts: (1) Host Module, (2) Processing, and (3) Visualization. Here, Python will act as the host program as it is interactive and popular in analytics. Moreover, it can run on any machine and it has many extensive library support. The computation is divided between Python and C++. The main processing will be done (computing summarization matrix, Phase 1) in C++ while the other parts will be done in Python. C++ works at the back end and it is hidden from the user. As mentioned above, a Python library ('SWIG') will handle the data passing through Python and C++. We emphasize that we only need to compute the summarization in C/C++ as Python is slower for this kind of processing. After processing is done in C++, it returns the output as a matrix back to the Python module. As the size of our summarization matrix is concise, transferring data to Python will be fast and it will easily fit in the main memory. Python processes the matrix and compute machine learning models exploiting it as mentioned earlier. While computing the machine learning models, we visualize the learning of the model parameters to the user using a GUI. We use plots, and graphs in the GUI to visualize the model learning to the user.

From a system perspective, our proposed system can show the learning of the machine learning models quickly and accurately. Users can also interact with the system to explore the models. For example, users can stop the computation at any time and get an approximate model there. Also, users have the option to visualize the evaluation metrics. Our system will divide the data set into train and test set, and visualize the respective evaluation metrices for each model based on the test data set. Our system has the potential to be released as a standard Python package that can be easily installed in any machine, solving all the dependencies (SWIG, C++ in this case).

2.4 Benchmarking

Table 1: Comparing accuracy to get approximate machine learning models.

					Relative error	
Θ	Data set (X)	n	d	% of X	Our Sol.	Python
LR	YearPrediction	500k	91	38	3.1E-2	14.9
PCA	YearPrediction	500k	91	8	1.8E-3	6.6E-2
NB	SkinNonskin	200k	4	2	1.1E-3	4.5E-3

To the best of our knowledge, there is little work putting together visualization of complex math models as they are gradually computed. Mostly, users call Python or R libraries in their code, and they have to wait until the computation ends. So, we cannot compare our system end-to-end with another system. Here, we compare our algorithm utilizing summarization matrix with the Python incremental library. Table 1 shows the result of our experimental outcome. We used data sets obtained from UCI machine learning repository, where n and d represent the original size of the data sets. We first compute the model parameters on the whole data set to get reference measurements. For each model, we show the amount of the data needed to get an approximate model (% of X column), the relative error of our solution for the approximate model, and the relative error from the Python incremental library at the same point. However, the final error after reading the whole data set between our solution and Python is less than 0.01%. Our incremental algorithm can get a solution in about 50% of time taken by the Python library [3]. Therefore, our solution can get an approximate model faster than Python with much less relative error.

3 SYSTEM DEMONSTRATION

We will run our system on a portable computer with Python and C++ installed (Python 3.6 and C++ 11). Our program provides a GUI to load or select data sets (path of the data set location), to choose a machine learning algorithm and to visualize model results and accuracy. Nevertheless, our system has the potential to work both from the OS command prompt and a GUI. Users can choose the machine learning model. Then it will be processed in Python and C++ in an incremental manner. Users can also select the block size (chunk size) to read the data. However, a default block size of 1000 will be set as the recommended block size which is justified in [3]. the visualization will be shown mostly in the GUI itself.

Python will be connected to C++ with a standard Python package (SWIG). A pre-loaded data set of small size will be available for faster demonstration purposes. If the data set size is large ($> 8Gb$), our system displays a warning. The output shown by our system is concise and it is updated as the data set is scanned.

3.1 Points to Emphasize

In our system demonstration we will emphasize the following points: (1) Our system is simple. It can be easily installed solving library dependencies, and the GUI is easy to use. (2) Our system is portable and it can work on any modern OS. That is, we do not need any system-specific libraries since computation is done by C++ and Python. (3) Our prototype is ideal to compute popular machine learning models. (4) Users can choose the machine learning model and main input parameters from Python GUI. Users can also tune the block size to read the data set. (5) Processing will be performed in C++ and Python. (6) Visualization is available in the GUI. (7) Interface between C++ and Python is performed with a Python library (SWIG). The connection and computation are hidden from the user. (8) We use a single machine for processing. However, it is possible to do the processing in parallel. The system is ran as a stand-alone application in the machine. (9) Our system is easy to modify for the developer. To extend the system, only the Python module needs to be changed without worrying about C++ code.

3.2 Demonstration Scenarios

Here, we walk through the demonstration scenarios showing our system can provide increasingly accurate output with excellent time performance.

Once the user runs our system as an application in any OS (e.g., Linux, Windows or Mac), the Python GUI will provide options to paste the file-path of a new data set or choose from existing

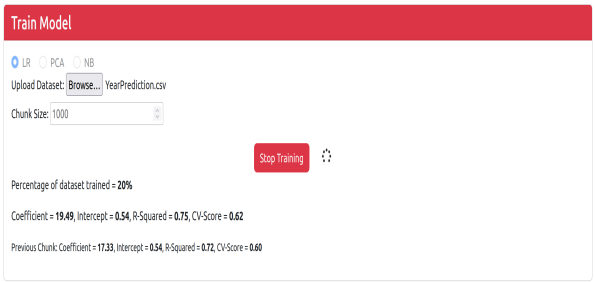

Figure 2: Input parameters.

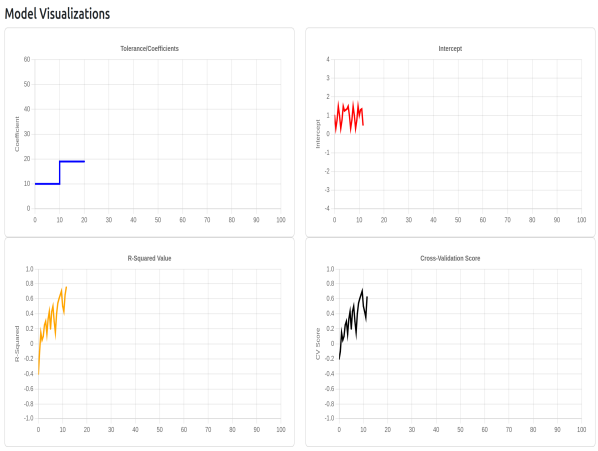

Figure 3: Visualizing incremental learning of model.

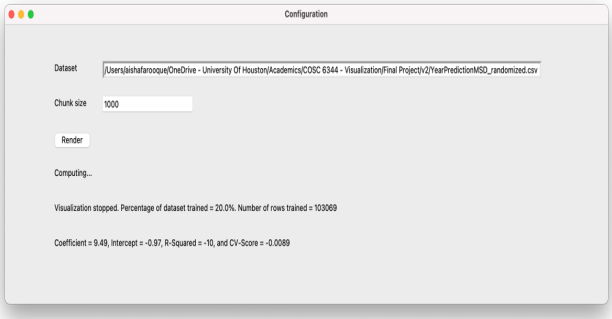

Figure 4: Output summary after running/stopping model computation.

data sets. For example, the user can choose YearPrediction or SkinNonSkin dataset available from UCI machine learning respiratory or paste the file-path of any dataset of their own in CSV format. However, small data sets will be loaded for the demonstration, but the user can load any data set of any size. The user can also choose the block size (or chunk size) of their own or choose the default one. Providing small block size will show a warning as the models will take more time to saturate. The user will also have the option to show the evaluation metrics or not. After that, the system will start the computation and Python will start reading the data set in blocks. As the summarization matrix and model parameters are computed, the results will be shown in the GUI. We show a demo scenario in Figure 2 and Figure 3 to compute Linear Regression in an incremental manner. We can see that our system is showing the learning of the model parameters (regression coefficients and intercepts). The X-axis shows the % of data that is being read and the Y-axis shows the model parameter values. Similarly, the bottom images show the accuracy evaluation metrics for the model on the test data set. Here, we show the R-Squared value as well as the cross-validation score.

As the model is being computed, the user has the option to wait until the model is computed on the full data set or the execution can be stopped at any time if the model is accurate enough. In any case, the user will be shown as summary report in a new window. For example, Figure 4 shows a report of the above mentioned model performance (LR). We can see that it notifies the user that the computation was stopped by the user, and the % of data read so far. Next, it shows the values of the model parameters (regression coefficients and Y axis intercept), as well as the values of the evaluation metrics (R-Squared and CV-Socre). Although here we present the demo scenario for the Linear Regression model, PCA and Naïve Bayes models behave the same way. Only output model parameters and ther corresponding evaluation metrics (if applicable) will be changed.

4 CONCLUSIONS

Our system provides many benefits. Our system helps users who need to analyze large data sets. Users can explore multiple models by visualizing their parameters and see how the model is behaving on the data set. Users do not have to wait until the completion: they can stop the computation at any time if they think the model is sufficiently accurate. For future work, we plan to integrate more ML models and compare our approach with models computed by gradient descent, the workhorse behind most libraries today.

REFERENCES

[1] M. Ahmed. Data summarization: a survey. *Knowl. Inf. Syst.*, 58(2):249–273, 2019.
[2] S. T. Al-Amin and C. Ordonez. Efficient machine learning on data science languages with parallel data summarization. *Data & Knowledge Engineering*, 136:101930, 2021.
[3] S. T. Al-Amin and C. Ordonez. Incremental and accurate computation of machine learning models with smart data summarization. *J. Intell. Inf. Syst. (JIIS)*, 59:149–172, 2022.
[4] A. Gepperth and B. Hammer. Incremental learning algorithms and applications. In *24th European Symposium on Artificial Neural Networks, ESANN*, pages 1–1, -, 2016. ESANN.
[5] T. Hastie, R. Tibshirani, and J. Friedman. *The Elements of Statistical Learning*. Springer, New York, 1st edition, 2001.
[6] C. Ordonez. Building statistical models and scoring with UDFs. In *Proc. ACM SIGMOD Conference*, pages 1005–1016, NY, USA, 2007. ACM Press.
[7] C. Ordonez and J. García-García. Vector and matrix operations programmed with UDFs in a relational DBMS. In *Proc. ACM CIKM Conference*, pages 503–512, 2006.
[8] C. Ordonez and S. Pitchaimalai. One-pass data mining algorithms in a DBMS with UDFs. In *Proc. ACM SIGMOD Conference*, pages 1217–1220, 2011.
[9] C. Ordonez, Y. Zhang, and W. Cabrera. The Gamma matrix to summarize dense and sparse data sets for big data analytics. *IEEE Transactions on Knowledge and Data Engineering (TKDE)*, 28(7):1906–1918, 2016.
[10] A. Spokoiny and Y. Shahar. Incremental application of knowledge to continuously arriving time-oriented data. *J. Intell. Inf. Syst.*, 1:1–33, 2008.