Scalable Machine Learning in the R Language Using a Summarization Matrix

Siva Uday Sampreeth Chebolu, Carlos Ordonez, Sikder Tahsin Al-Amin

Department of Computer Science
University of Houston, Houston TX 77204, USA

Abstract. Big data analytics generally rely on parallel processing in large computer clusters. However, this approach is not always the best. CPUs speed and RAM capacity keep growing, making small computers faster and more attractive to the analyst. Machine Learning (ML) models are generally computed on a data set, aggregating, transforming and filtering big data, which is orders of magnitude smaller than raw data. Users prefer "easy" high-level languages like R and Python, which accomplish complex analytic tasks with a few lines of code, but they present memory and speed limitations. Finally, data summarization has been a fundamental technique in data mining that has great promise with big data. With that motivation in mind, we adapt the Γ (Gamma) summarization matrix, previously used in parallel DBMSs, to work in the R language. Γ is significantly smaller than the data set, but captures fundamental statistical properties. Γ works well for a remarkably wide spectrum of ML models, including supervised and unsupervised models, assuming dimensions (variables) are either dependent or independent. An extensive experimental evaluation proves models on summarized data sets are accurate and their computation is significantly faster than R built-in functions. Moreover, experiments illustrate our R solution is faster and less resource hungry than competing parallel systems including a parallel DBMS and Spark.

1 INTRODUCTION

Machine Learning has become popular and gained a lot of demand in the present world with the availability of abundant data and abundant processing power. There are a lot of tools and technologies like Python, R, Scala, Java, C# and many more which compute these machine learning models. However, data sets can be so large that they do not fit in the main memory. For these types of data, Hadoop stack or distributed systems or DBMSs like Vertica, SciDB is a popular choice to compute the Machine Learning models [15], [10]. Contrary to the popular belief, we propose that the size of the cleaned data set, rather than its raw counterpart should dictate the data processing platform to be used. Data cleaning strips off a lot of unwanted and inaccurate data. As a result, the size of the data set is significantly reduced and with it, the need to use a heavyweight data processing platform like Hadoop. Therefore, with a refined data set, data processing can be limited to a single system environment like, in our case, R.

With a vast package ecosystem coupled with extensive developer support, R ticks all the right boxes when it comes to being a data analytics platform [7]. However, R has few shortcomings of which memory management, speed and efficiency are the most noticeable. While parallelism in R can be achieved by using packages like *parallel*, the shortcoming becomes evident with an increasing number of cores. The language design sometimes poses a great problem when working with large data sets since the data has to be stored in physical memory. With the dedicated physical memory, R cannot scale to deal with data sets larger than the proportion of memory allocated to it and is forced to crash in such cases. So, the physical memory limitation clearly outweighs the need to address the issue of parallelization in R. In an attempt to address the above limitation in R, we used the summarization technique in the first Phase of our approach. But again, summarization technique can be used only for those models which accept Gramian Matrix product like Linear Regression (LR), Principal Component Analysis (PCA), Naïve Bayes (NB), K-means (KM) and few others. Furthermore, we built upon the parallel database systems algorithm in [10] to make it work in a serial scalable manner in R. Here, we implemented the models from [10] and also explored new models like Naïve Bayes and K-means which require a new gamma matrix, Diagonal Gamma, instead of the old ones stated in [10]. The environment does not crash even for large data sets, works independent of the physical memory allocated to the R environment and gives as accurate results as the existing packages that compute the above models in R.

2 DEFINITIONS

This is a reference section which introduces definitions of input data sets and models from mathematical perspective, R runtime and RCpp package. Each subsection can be skipped by a reader familiar with the material.

2.1 Mathematical Definitions

First, we define the inputs given to the models. The most obvious one is the input data set, interpreted as a matrix, which is defined to be a set of n column-vectors. All the models take a $d \times n$ matrix as input. Let the input data set be defined as X, which is considered to have n points, where each point is a vector in R. Therefore, we can see X as a wide rectangular matrix. In the case of Linear Regression (LR) and Principal Component Analysis (PCA), we take an extra dimension (output variable Y) resulting a change in the dimensions of X to $(d+1) \times n$, which we call $\mathbf{X}$. We use *i=1...n* and *j=1...d* as matrix subscripts. We augment X with an extra row of n 1s and call that as matrix $\mathbf{Z}$ $((d+2) \times n)$ for mathematical convenience. Column-vectors and column-oriented matrices are used for mathematical convenience because they allow simpler equations.

We use Θ to represent a statistical model in general. That is, Θ can be a LR or PCA model as well as any of the clustering and classification models such as Naïve Bayes(NB) and K-means(KM). PCA is an unsupervised model to

reduce dimensionality. LR is a fundamental supervised model, whose solution helps in understanding and building other linear models. Naïve Bayes is another classic supervised model, whose solution assigns a numerical value between 0 and 1 to each class label denoting the probability of data belonging to a specific class. K-means is a clustering algorithm whose goal is to find k similar groups in the data. The algorithm works iteratively to assign each data point to one of k groups based on the features that are provided. Data points are clustered based on feature similarity. Therefore, for each model, $\Theta = \{list\ of\ matrices/vectors\}$, as follows. For LR: $\Theta = \beta$ where β is the vector or regression coefficients; for PCA: $\Theta = U, D$ where U are the eigen vectors and D contains the squared eigenvalues obtained from SVD; for NB: $\Theta = \{\pi, \mu, \sigma\}$, where π is the vector of k class priors, μ is a set of k mean vectors and σ are k diagonal matrices with standard deviations; and for KM: $\Theta = \{W, C, R\}$, where W is a vector of k (number of clusters) weights, C is a set of k centroid vectors and R is a set of k variance matrices.

2.2 R Runtime and RCpp Package

R is a dynamic language for statistical computing that combines lazy functional features and object-oriented programming [6], [12]. In R, vectors are stored as one contiguous block, matrices are 2-dimensional arrays of real numbers, which are stored as one block in column major order dynamically allocated, Lists are the most general ones and can have elements of diverse data types, including atomic data types and nested data structures. R uses a dynamic interpreter and also it utilizes C language for matrix and data frame operations and LAPACK library for linear algebra and numerical methods. When R functions are called, the R run-time creates nested variable environments, which are dynamically scoped.

The advantage of the RCpp package is its memory management. We can pass values to and from R and RCpp. When we pass the values, only the reference gets passed to the other side but not the actual value. So, memory consumption is very efficient and the runtime is the same. We can even pass matrices, lists, vectors and similar data to RCpp and return any of those from RCpp.

3 Theory and Algorithm

We present our main technical contribution in this section. First, we propose our main algorithm and then we discuss it in details. Then we discuss the implementation of our algorithm in R and RCpp. Finally, we give the run time complexity of our algorithm.

3.1 Algorithm

Our main algorithm consists of two steps:

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

In phase 1, first, we review the Gamma matrix (Non-Diagonal Gamma) and the statistics in it which was proposed in [10]. Matrix Γ (Gamma), defined below, is a fundamental matrix which contains a complete, accurate and sufficient summary. Then we describe the design and implementation of our main technical contribution, the Diagonal Gamma matrix. Both Non-Diagonal Gamma and Diagonal Gamma provides summarization for a different set of models which are presented in phase 2. For PCA and LR, we need one full Γ matrix assuming element off-diagonal is not zero. And for NB and KM, we need k matrices Γ_j (k classes, or k clusters respectively), where each Γ_j is "diagonal" meaning we assume Q is diagonal where off-diagonal elements are assumed to be zero. We discuss both phases in details in the following sections.

3.2 Phase 1: Computing Summarization Matrices

First we review the sufficient statistics for X which are integrated to form the Non-Diagonal Gamma Matrix, which are:

$$n = |X|, \tag{1}$$

$$L = \sum_{i=1}^{n} x_i, \tag{2}$$

$$Q = XX^T = \sum_{i=1}^{n} x_i \cdot x_i^T \tag{3}$$

Here, X is the data set, n counts total number of points in the data set, L is a linear sum of x_i and Q is a sum of vector outer product where x_i is multiplied by itself, i.e., Q is simply the "quadratic" sum of x_i. As defined earlier in 2.1, X is $d \times n$, $\mathbf{Z}$ has $(d+2)$ rows and n columns, where row [0] are 1s and row $[d+1]$ is Y. Hence, z_i can be defined as $z_i = [1, x_i, y_i]$. Then the $\mathbf{Z}$ matrix becomes:

$$\mathbf{Z} = \begin{bmatrix} 1 & 1 & ... & 1 \\ x_1 & x_2 & ... & x_n \\ y_1 & y_2 & ... & y_n \end{bmatrix} \tag{4}$$

Matrix Γ (Gamma), which is defined below, is a fundamental Gamma matrix which contains a complete, definite, and sufficient summary of X to efficiently compute models like LR and PCA that have been previously defined. We define a complementary Gamma matrix, Diagonal Gamma, in Section 3.2 for models assuming variable independence, like Naïve Bayes and K-means.

$$\Gamma = \begin{bmatrix} n & L^T & \mathbf{1}^T \cdot Y^T \\ L & Q & XY^T \\ Y \cdot \mathbf{1} & YX^T & YY^T \end{bmatrix} = \begin{bmatrix} n & \sum x_i^T & \sum y_i \\ \sum x_i & \sum x_i x_i^T & \sum x_i y_i \\ \sum y_i & \sum y_i x_i^T & \sum y_i^2 \end{bmatrix} \tag{5}$$

Here, Γ which can be computed in two ways from [10]. Alternative (1) is matrix-matrix multiplications i.e. $\mathbf{Z}\mathbf{Z}^T$; Alternative (2) is sum of vector outer products i.e. $\sum_i z_i \cdot z_i^T$. So, $\Gamma = \mathbf{Z}\mathbf{Z}^T = \sum_{i=1}^{n} z_i \cdot z_i^T$. That is, the square of matrix $\mathbf{Z}$ gives us Γ, which is significantly smaller than X. In general, if $d << n$, Γ comfortably fits in main memory.

Diagonal Q matrix assuming dimensions are independent: From [10], it is clear that Non-Diagonal Gamma matrix, despite being iterative algorithms, avoids reading the entire data sets at every iteration. But that approach cannot be applied on models like Naïve Bayes(NB) or K-means(KM) which require more than one summarization matrix and may also require to read the entire data set more than once. For example, Naïve Bayes requires k summarization matrices for a given data set, where k being the number of unique class labels in the data set and K-means requires k matrices for summarization of a data set with k as the number of clusters given by the user, i.e., one for each cluster. Furthermore, these models do not require the complete computation of the Non-Diagonal Gamma as described in 3.2. The reason behind that is, the LR and PCA are computed in rotated space whereas in NB and KM we assume that the dimensions are independent, making Gamma diagonal. Due to this reason, we introduce another matrix, Diagonal Gamma, which helps to compute these models. Here, we do not require the Y parameter for Naive Bayes and K-means as used in LR and PCA. The major difference between the two forms of Gamma is we do not require parameters off the diagonal in Diagonal Gamma matrix as in Non-Diagonal Gamma matrix. So, we need only a few parameters out of the whole Non-Diagonal Gamma, namely, n, L, L^T, Q. That is, we require only a few sub-matrices from Non-Diagonal Gamma, which can be visualized as:

$$\Gamma_{diag} = \begin{bmatrix} n & L^T & 0 \\ L & Q & 0 \\ 0 & 0 & 0 \end{bmatrix}, where\ Q = \begin{bmatrix} Q_{11} & 0 & 0..... & 0 \\ 0 & Q_{22} & 0..... & 0 \\ 0 & 0 & Q_{33}..... & 0 \\ 0 & 0 & 0..... & Q_{dd} \end{bmatrix} \tag{6}$$

Furthermore, if we see the above sub-matrix, we observe that if we compute the terms in the lower triangle, we can get the whole sub-matrix just by copying the L to L^T, i.e., we need to compute the terms in the lower triangle and copy it to the upper triangle. This is the major change in definition of the Non-Diagonal Gamma to that of the Diagonal Gamma. Also, in Non-Diagonal Gamma, the Q is computed completely. On the other hand, in Diagonal Gamma, the Q is diagonal. From which we came up with the name of the matrices as Diagonal and Non-Diagonal Gamma. So, Q is diagonal or non-diagonal but not Γ.

3.3 Phase 2: Computing Models

Models are computed using the two versions of Gamma. One is with one Non-Diagonal Gamma Matrix and another one is k-Diagonal Gamma Matrices. Both of them were introduced previously.

Models based on one Non-Diagonal Gamma:

Linear Regression (LR): From [10], the standard definition of LR is given as $Y = \beta^T \mathbf{X} + \epsilon$, where β is the column vector of regression coefficients and ϵ represents the Gaussian error. $\mathbf{X}$ is a $(d+1) \times n$ augmented matrix where we

have X with a row of n 1s. β can be defined as $\hat{\beta} = (\mathbf{X}\mathbf{X}^T)^{-1}\mathbf{X}Y^T$. From the discussed Non-Diagonal Gamma, we can rewrite this equation as

$$\hat{\beta} = Q^{-1}(\mathbf{X}Y^T) \tag{7}$$

Principal Component Analysis (PCA): PCA is mainly implemented on a data set to reduce noise and redundancy of dimensions. PCA can be computed on the covariance matrix (V), or the correlation matrix (ρ), of the data set from [4]. This model require two parameters. First is U, which is a set of d orthogonal vectors, principal components of the data set, ordered in decreasing order by their variance. Second is the diagonal matrix D^2 which contains the squared eigen values. From [10], we can compute ρ, the correlation matrix, from the two parameters, D and U 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 by using Eigen decomposition of the ρ, which is a symmetric matrix factorization. That is, we compute PCA from the correlation matrix by solving Singular Value Decomposition (SVD) on it. Also, we express ρ in terms of the sufficient statistics to compute SVD as follows:

$$\rho_{ab} = \frac{(nQ_{ab} - L_aL_b)}{(\sqrt{nQ_{aa} - L_a^2}\sqrt{nQ_{bb} - L_b^2})} \tag{8}$$

.

Models based on *k* Diagonal Gammas:

Naïve Bayes (NB): The input for this model is a data set X and the output is a Naïve Bayes classification model which contains C(mean per dimension), R (variance per dimension), and W (prior per class). First, we take the data set X as input in chunks of fixed size. In each chunk, we split the data based on number of classes in the data set. We compute one gamma for each part of the chunk and at last add up these Γ matrices with respect to the classes and arrive at a final list of Γ matrices one for each class. We focus on $k = 2$ classes for NB. Then finally we have Γ_0 for class 0 and Γ_1 for class 1. We extract N_g, L_g, Q_g as defined in 3.2, from this final list of Γs. So, we arrive at lists of N_g, L_g, Q_g from where we compute π, μ and σ per dimension per item in the list separately like:

$$\pi_g = \frac{N_g}{n}, \tag{9}$$

$$\mu_g = \frac{L_g}{N_g}, \tag{10}$$

$$\sigma_g = \frac{Q_g}{N_g} - diag[\frac{L_gL_g^T}{N_g^2}] \tag{11}$$

Here, $N_g = |Xg|$ and we take the diagonal of $L \cdot L^T$ and Q, which can be manipulated as a 1-dimensional array instead of a 2D array. These are the 3 parameters included in the Naïve Bayes model. Now, we can predict class labels for new data using this model. For the prediction, for each point in the input

data, we compute a probability value per class using the model parameters and assign the class with maximum probability. We compute the probability using, $P_{xi_{class}} = (1/\sqrt{2\pi\sigma_{g_j}^2})e^{(-0.5(x_i-\mu_{xi})^2/\sigma_{g_j}^2)}$

K-means (KM): The input for this model is a data set X and the number of clusters (k) and the output is three matrices C, R, W, containing the means, the variances and the weights respectively for each cluster of X. For K-means with k clusters, we have list of matrices as $\Gamma_1, \Gamma_2, .., \Gamma_k$, where $k \geq 2$. Following definitions from 3.2, we introduce similar model parameters X_j, N_j, L_j, Q_j as the subset of X which belong to cluster j, the total number of points per cluster ($|X_j|$), the sum of points in a cluster ($\sum_{\forall x_i \in X_j} x_i$) and the sum of squared points in each cluster ($\sum_{\forall x_i \in X_j} x_i x_i^t$) respectively. From these statistics, we compute C_j, R_j, W_j as:

$$C_j = \frac{L_j}{N_j}, \tag{12}$$

$$R_j = \frac{Q_j}{N_j} - diag[\frac{L_j L_j^t}{N_j^2}], \tag{13}$$

$$W_j = \frac{N_j}{n} \tag{14}$$

Here $N_j = |Xj|$ and we take diagonal of $L \cdot L^T$ and Q, which can be treated as vectors instead of a matrix. The algorithm iterates executing two steps starting from random initialization until cluster centroids become stable.

Step 1 determines the closest cluster for each point and adds the point to it. K-means uses Euclidean distance to determine the closest centroid to each point x_i which is defined as $d(x_i, C_j) = (x_i - C_j)^t(x_i - C_j)$

Step 2 updates all centroids C_j by computing the mean vector of points belonging to cluster j. The cluster weights W_j and diagonal covariance matrices R_j are also updated based on the new centroids. The quality of a clustering solution is measured by the average quantization error $q(C)$, defined in [8] (also known as distortion and squared reconstruction error). Lower is the value of $q(C)$, better is the quality of clustering. $q(C) = \frac{1}{n}\sum_{i=1}^{n} d(x_i, C_j)$, where $x_i \in X_j$.

The K-means algorithm stops when centroids change by a marginal fraction in consecutive iterations which is measured by the quantization error. With decreasing $q(C)$ at each iteration, K-means is theoretically guaranteed to converge, yet a threshold is set on the number of iterations to avoid excessively long runs.

3.4 Computing Gamma Matrix and Machine Learning Models in R

We discuss how Γ is computed exploiting RCpp and how the models are computed in R itself. Depending on the models, we choose between the Non-Diagonal or the Diagonal Gamma matrix to compute at first.

Phase 1: This part is computed exploiting RCpp package. From section 3.1, phase 1 takes are of computing the sum, $\sum_i z_i z_i^T$. The main idea is to evaluate this equation in C++ code instead of R code, following the same UDF idea presented in [10].

First, we take the input data set (X) and split that into chunks of equal size. Chunks are a subset of X so that chunk fits in RAM and it has many points. If there are M chunks, then X is partitioned into $X_1, X_2, .., X_M$ chunks, where each chunk X_I (uppercase i) fits in RAM. Regarding chunks most libraries in R use data frames and therefore it is sort of a table, not a matrix. It seems the conversion from data frame to matrix is done somewhere. We read text files because they are the most common. However, our program would be more efficient with binary files.

If the model to be computed is LR or PCA, we compute the Non-Diagonal Gamma based on the type of data set (whether it is dense or sparse) for each chunk. So, we have a list of Γs. If the model is Naïve Bayes, we compute the Diagonal Gamma, one for each class label for every chunk. If the model K-means, for the first iteration and first chunk, we initialize the k cluster centroids randomly and for successive iterations, we initialize the k cluster centroids with that of the first chunk. Then, we assign a cluster number to each data point and compute the Diagonal Gamma, one for each cluster in every chunk. Hnece, we have a list of list of k Γs. Since Γ is additive, we can add all the intermediate Γs to obtain a final Γ. This is straightforward for LR and PCA. But for Naïve Bayes and K-means, since we have list of list of Γs, we need to add the Γs corresponding to a given class/ cluster respectively such that we arrive at a final Γ which is a list of matrices representing each class/cluster.

Phase 2: In this part we compute each model (θ). While Phase 1 is basically exploiting RCpp, Phase 2 uses R itself "as is" (we use R existing functions and operators). After obtaining the final Γ, we use Non-Diagonal Gamma to compute LR and PCA and Diagonal Gamma to compute Naïve Bayes and K-means using the mathematical equations discussed previously. Since the models LR, PCA, and NB do not need to converge to a best solution like K-means, that will be the end of Phase 2 for them. On the other hand, K-means is not trivial to compute as it needs to converge to a best solution by the reduction of the quantization error to a minimum value. So, we need to repeat the Phase 1 and Phase 2 iteratively in order to achieve this. Every time we read the data set, we take the cluster centroids from the previous pass, which improves the accuracy of the model. This process terminates when there is no change in the clusters formed from previous iteration. In summary, for LR, PCA, and Naïve Bayes, we read each and every point in the data set only once but for K-means, we read the data set multiple times until a best solution is achieved. It is beyond the scope of this paper to justify why Γ eliminates the need to read X multiple times in LR and PCA, but not in KM.

Here, the input data set X, intermediate computations and output model, everything is a matrix. In summary, the Γs are computed in Cpp exploiting

RCpp package and the models are computed in R itself. To compute LR and PCA, we are forced to call R routines. But for NB and KM, we can compute it ourselves, helped by the fact that diagonal Q simplifies computations in addition to efficiency.

3.5 Time and Space Complexity Analysis

From [10], it is clear that the time complexity for the Phase 1 of the Non-Diagonal Gamma with dense data is $O(d^2n)$ and sparse data is $O(k^2n)$, assuming k entries in x_i are non-zero on an average. In Phase 2, we compute the machine learning models based on the Gamma from Phase 1. So, time for Phase 2 does not depend on n and is $\Omega(d^3)$, which for a dense matrix may approach $O(d^4)$, when the number of iterations in the factorization numerical method is proportional to d. This Non-Diagonal Gamma is used by models like LR and PCA.

A separate Gamma matrix, Diagonal Gamma, is used owing to the fact that a major set of the traditional Non-Diagonal Gamma has little-to-no utility for models like Naïve Bayes and K-means. Time complexity of Diagonal Gamma computation is $O(dn)$ as we compute only L and diagonal of Q of the whole Non-Diagonal matrix. This time complexity applies for all the models utilizing the Diagonal Gamma except K-means. The time complexity of K-means would be $O(kdn)$, where k is the number of clusters.

When we come to the space complexity, space required by Non-Diagonal Gamma matrix in main memory with dense representation is $O(d^2)$. However, it is $O(kd)$ for K-means and $O(d)$ for Naïve Bayes. In short, we can state that Diagonal Γ consumes much less memory than full Γ. However, Diagonal Gamma does not mean faster algorithms since KM requires multiple iterations.

4 Experimental Evaluation

We present an experimental evaluation of our R package and the machine learning models based on the Γ matrix. First, we show the models computed by our R package are accurate, down to almost zero error. Second, we compare the times from our package in R with those times obtained in three alternatives: a columnar DBMS (Vertica [5]), well-known R functions computing each model and the popular Hadoop stack system, Spark.

4.1 Experimental Setup

Hardware and software: The system and software configuration used for the experiments is a four core 2.83GHz system with Linux Ubuntu as operating system with 4GB physical memory and 294GiB storage space.

Data sets: The data sets which are used for the experiments are described in Table 1. All the data sets are taken from the UCI Machine Learning repository. We also include the information about the models which utilize these data sets. We replicated each of the data sets in order to get various combinations of n and d without altering statistical properties of the data. The

first one was sampled and replicated to get combinations of $d = (9, 91)$ and $n = (0.5M, 1M, 10M)$, second was replicated to get the combinations of $d = 30$ and $n = (0.2M, 1M, 10M, 100M)$ and the third one is replicated to get $d = 4$ and $n = (0.1M, 1M, 10M, 100M)$.

Table 1. Base data sets description

Data set	d	n	**Description**	**Used for Model**
CreditCard	30	285K	predict if there is raise in credit line	Naïve Bayes
YearPredictionMSD	90	515K	predict if there is rain or not	LR and PCA
Iris	3	150	to distinguish the flower species	K-means

4.2 Accuracy Evaluation

Table 2 below shows the results of the experiments that were performed using the two forms of Gamma. We compared the accuracy of model computations of our package with similar packages in R, which is a popular language and environment for statistical computing. We implemented four models in our package, namely, LR, PCA, Naïve Bayes and K-means. For each model, we have a different way of measuring the accuracy with the common underlying metric being Relative Error. From Table 2, we understand that the results from the functions of our package are almost an exact match with the output given by the currently existing best packages in R.

For LR, we get an intercept and a β per attribute as an output for the model computed by Gamma matrix. This is similar to the output given by *lm()*, the preferred default routine in R for LR, for the same input data set. We then compute the absolute differences among all the respective values of intercept and βs, from which we compute the Relative differences. Finally, we report the maximum of the relative differences among the intercept and the βs in Table 2.

For PCA, we get a diagonal matrix, D, of Eigen values and two ortho-normal matrices, S and V, which are Eigen vectors of the given input matrix. Unlike other models, we do not compute PCA completely in Cpp as it gives inaccurate results. Rather we use pure R routines to compute SVD of the correlation matrix generated from the Gamma matrix. The values in D depict the relative importance of each column in S and V matrices. So, we imply on the point that, for the computation of relative error, we take the values from D whose value is greater than 1. We first find the absolute differences among the pairs of corresponding values from the output of the Gamma matrices and that of the default R routines, from which we compute the relative differences. We report the maximum of these relative differences in Table 2.

In Naïve Bayes, we build a model to predict the class labels for the test data set. For that, we compute two separate Naïve Bayes models on the given input training data set using the default R routine and the aforementioned Gamma

functions. Consequently, we compute the prediction accuracy by finding the degree to which the predictions made by the functions of our package conforms to that from standard routine in R.

For K-means, we group the input data into k clusters, where k is pre-defined by the user. We compute K-means with both the default R routine and the previously discussed Gamma functions. The output from both the techniques have three vectors, namely, Centers, Radii and Weights. We take the weight vectors, sorted in decreasing order, from both the models and obtain the respective absolute errors. We use this absolute error to compute the relative errors with respect to the weight vector of the model computed from the default R routine. We report the maximum value of relative error in Table 2.

Table 2. Accuracy of models on respective data sets.

Model	**Maximum Relative Error**	**Data set used**
LR	5.89E-10	YearPredictionMSD
PCA	4.75E-13	YearPredictionMSD
Naïve Bayes	0	CreditCard
K-means	4.7E-2	Iris

4.3 Time Performance Evaluation and Benchmarking

We compare the performance of the models in our proposed package with the currently available best packages in R to compute the respective models, a similar implementation done in Vertica, which is a very fast columnar database [5] and also popular for big data analytics nowadays [1] and Spark which is the best representative from the Hadoop world. Since Naïve Bayes and K-means are new models that we explored in our research, there are no prior implementations of these in Vertica. So, we made the comparisons with Vertica for LR and PCA only.

Table 3 and 4 compares the time to compute PCA and LR on YearPrediction data set with Vertuca, R and Spark. We can see that as the as the size of the data set increases, the inbuilt R packages crash. One of the main reasons can be attributed to the fact that it tries to load the whole data set into main memory, eventually resulting in untimely aborts of the program. Howver, our package overcome this problem by not loading the entire data set into the memory, instead breaking the data set into chunks according to allocated memory. Also, though Vertica and Spark are able to compute the models even for large data sets, they perform slower than our package in R. As n grows, the time complexity of our method for LR and PCA is shown in Fig 1.

Table 5 compares the time to compute Naïve Bayes model in our package with the one given by R. We see that R crashes for large values of n which is not the case with our package. From Table 6, although the current packages in R scale well for small data sets, they result in untimely aborts for large data

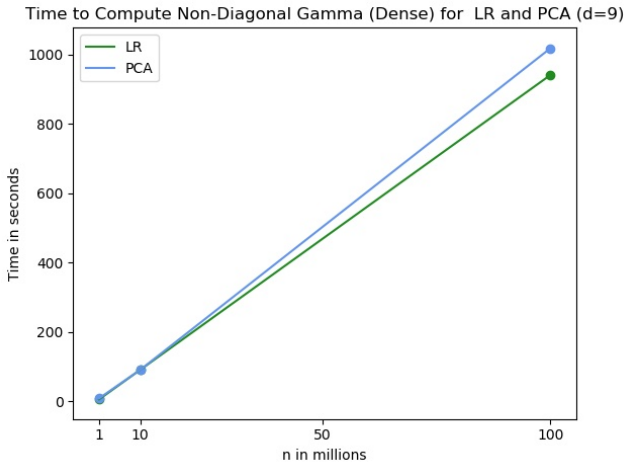

Fig. 1. Time Complexity to compute LR and PCA as n grows.

sets. As the size of data set increases, the performance of our package improves greatly. Spark, on the other hand, is able to compute the models though it is much slower than our package.

Table 3. Time to compute PCA on YearPrediction data set (Dense) (in secs)

n	d	**R+** $\Gamma_{non-diag}$ **(dense)**	**R+** $\Gamma_{non-diag}$ **(sparse)**	**Vertica**	**R**	**Spark**
0.5M	91	22	33	46	336	67
1M	91	66	80	115	575	130
10M	91	726	800	1290	crashed	1074
1M	9	9	9	10	21	31
10M	9	91	75	110	205	286
100M	9	1018	1020	1560	crashed	1780

4.4 Strengths and Weaknesses

Even though this model works efficiently for data sets with rows in the order of millions, it does not work as intended with the billion or higher rowed counterparts. This issue is magnified with the K-means algorithm as it requires multiple reads of the data set before returning the final clusters. Notwithstanding the long execution times, it still gives accurate results in contrast to the existing packages that result in untimely session aborts. As we see in the experimental results of K-means, the existing most efficient package for K-means model in R is aborted for a data set with five million rows or higher. In a similar manner, even for Naïve Bayes, the most efficient package in R is aborted when a data set with ten million rows is given as input while our solution returned accurate results within a reasonable amount of time.

Table 4. Time to compute LR on YearPrediction Data set (Dense) (in secs)

n	d	**R+** $\Gamma_{non-diag}$ **(dense)**	**R+** $\Gamma_{non-diag}$ **(Sparse)**	**Vertica**		**R Spark**
0.5M	91	22	36	46	276	67
1M	91	74	74	115	630	130
10M	91	720	828	1290	crashed	1074
1M	9	6	6	10	24	31
10M	9	91	69	110	285	286
100M	9	941	928	1560	crashed	1780

Table 5. Time to compute Naïve Bayes on Credit card data set (Dense) (in secs)

n	d	**R+**Γ_{diag}	**R**
0.2M	30	7	51
1M	30	40	158
10M	30	399	crashed
100M	30	1132	crashed

Table 6. Time to compute K-means on Iris data set (Dense) (in secs)

n	d	**R +** Γ_{diag}	**R Spark**	
150	4	0	0	3.2
0.1M	4	6	0	7.5
1M	4	65	6	43.3
5M	4	380	crashed	1370
10M	4	756	crashed	3012

Our solution adapts to the local machine and customizes the chunk size with respect to the available physical memory. The main drawback is that R cannot be easily parallelized unlike the Hadoop stack or other parallel systems to completely utilize the cores available in a system thus resulting in a decreased performance.

5 Related Work

There are many techniques to improve the performance of the models PCA, Naïve Bayes and K-means few of which are [14], which used decomposition of Classes via Clustering to improve Naïve Bayes, [3, 13], which used the triangle inequality and collaboration of compressed sensing theory and K-SVD approach to accelerate K-means, [15, 8], which did Fast PCA computation in a DBMS with Aggregate UDFs and LAPACK and improved performance on MapReduce environment. If we observe carefully, LR, Naïve Bayes or PCA does not require any initialization unlike the K-means model which require the number of clusters and their respective centroids to be initialized. If the initialization is bad, we never converge at a solution.

Summarization of scalable Machine Learning algorithms was done in a parallel manner in [10]. The authors of the [10] exploited HP Vertica's parallelization feature, similar to [11], to perform summarization on multiple systems simultaneously. We adapted the algorithms in [10] and implemented them such that they are serial, scalable and are 99 percent accurate in R. We made use of the chunking ability in R to read the infinite amount of input data which also makes the process faster. We removed the use of database system completely which is the main component in [10]. In [9], Naïve Bayes is computed inside the database with pure SQL queries. We adapted model computation from [9] and implemented it in R. We compared our work with the most efficient ones in R and

have shown that our package is faster and reliable than the former. Alternatively, there is another technology, Microsoft R Open, which is also designed to include an updated R engine (R 3.2.2), new fuzzy matching algorithms, the ability to write to databases via ODBC, and a streamlined install experience. This can also be used to obtain some optimization in building the models. Computers, nowadays have more physical memory, more computing power. So, using a single system, our solution is better for millions of records with all the 4 models. The algorithms programmed in R and C++ are presented in [2].

6 Conclusions

We introduced a powerful summarization matrix to compute fundamental ML models in two phases: Phase 1 to compute one summarization matrix or multiple summarization matrices and Phase 2 to update model parameters based on summarization, where Phase 1 is I/O intensive and Phase 2 is CPU bound. Based on our summarization matrix we developed an R package capable of computing ML models with high accuracy, high speed, and no main memory limitations. Specifically, our R package computes LR, PCA, NB and KM models in one pass over the input data set, except for KM which requires iterative processing. The main memory limitation is solved by reading the data set in small blocks (relative to available RAM) and incrementally updating summarization (with either one summarization matrix or multiple summarization matrices). High speed is achieved by computing the summarization matrix in high-performance C++ code, compiled and linked to run inside the R runtime. We introduced several variants on the Gamma matrix to work with sparse data sets, diagonal and non-diagonal variance matrices, as well as supervised and unsupervised models. That is, we cover a wide spectrum of data sets and ML models, thereby offering wide applicability. We presented interesting experiments to evaluate accuracy and time performance. We show our summarization matrix produces practically the same model, with negligible error, compared to standard R functions. On the other hand, we show our R algorithms are much faster than R built-in functions, removing main memory limitations, but preserving the ease of use. Extensive benchmarks show our package is faster than competing parallel systems: a parallel DBMS and the popular Spark system. In short, our R package opens the possibility of analyzing large data sets on an average personal computer.

Even though our research proves we can get better performance and scalable computing in the R language beyond RAM limits with single-threaded processing, there are many opportunities for future research. We need to explore non-linear ML models, like logistic regression and Support Vector Machines. We need to explore mechanisms to parallelize summarization inside the R runtime, via parallel C or C++ code running on multicore CPUs. Our approach has the promise to be applied in other high-level languages including Python, Matlab, and Javascript, being Python our first target. Given extensive past research work on parallel processing on big data it is worth investigating a data set size threshold to move processing from a single machine to a parallel cluster.

Acknowledgements

The second author would like to thank the guidance of Simon Urbanek, from ATT Labs, to understand the R language runtime source code.

References

1. Al-Amin, S.T., Ordonez, C., Bellatreche, L.: Big data analytics: Exploring graphs with optimized SQL queries. In: Database and Expert Systems Applications - DEXA 2018. pp. 88–100 (2018)
2. Chebolu, S.U.S.: A General Summarization Matrix for Scalable Machine Learning Model Computation in the R Language. Master's thesis, University of Houston (2019)
3. Elkan, C.: Using the triangle inequality to accelerate k-means. In: Machine Learning International Conference. vol. 20, p. 147 (2003)
4. Hastie, T., Tibshirani, R., Friedman, J.: The Elements of Statistical Learning. Springer, New York, 1st edn. (2001)
5. Lamb, A., Fuller, M., Varadarajan, R., Tran, N., Vandiver, B., Doshi, L., Bear, C.: The vertica analytic database: C-store 7 years later. Proc. VLDB Endow. **5**(12) (2012)
6. Morandat, F., Hill, B., Osvald, L., Vitek, J.: Evaluating the design of the R language - objects and functions for data analysis. In: ECOOP 2012 - Object-Oriented Programming - 26th European Conference. pp. 104–131 (2012)
7. Ordonez, C., Johnson, T., Urbanek, S., Shkapenyuk, V., Srivastava, D.: Integrating the R language runtime system with a data stream warehouse. In: Proc. DEXA Conference. pp. 217–231 (2017)
8. Ordonez, C., Omiecinski, E.: Efficient disk-based K-means clustering for relational databases. IEEE Transactions on Knowledge and Data Engineering (TKDE) **16**(8), 909–921 (2004)
9. Ordonez, C., Pitchaimalai, S.: Bayesian classifiers programmed in SQL. IEEE Transactions on Knowledge and Data Engineering (TKDE) **22**(1), 139–144 (2010)
10. Ordonez, C., Zhang, Y., Cabrera, W.: 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)
11. Raychev, V., Musuvathi, M., Mytkowicz, T.: Parallelizing user-defined aggregations using symbolic execution. In: Proceedings of the 25th Symposium on Operating Systems Principles. pp. 153–167. ACM (2015)
12. Stadler, L., Welc, A., Humer, C., Jordan, M.: Optimizing R language execution via aggressive speculation. In: Proceedings of the 12th Symposium on Dynamic Languages, DLS 2016. pp. 84–95 (2016)
13. Ueda, N., Nakano, R., Ghahramani, Z., Hinton, G.: SMEM algorithm for mixture models. Neural Computation **12**(9), 2109–2128 (2000)
14. Vilalta, R., Rish, I.: A decomposition of classes via clustering to explain and improve Naive Bayes. In: Proc. ECML Conference. pp. 444–455 (2003)
15. Zhang, Y., Ordonez, C., Cabrera, W.: Big data analytics integrating a parallel columnar DBMS and the R language. In: Proc. of IEEE CCGrid Conference (2016)