Bayesian Variable Selection in Linear Regression in One Pass for Large Data Sets

Carlos Ordonez, University of Houston
Carlos Garcia-Alvarado, University of Houston
Veerabhadran Baladandayuthapani, UT MD Anderson Cancer Center [*]

Abstract

Bayesian models are generally computed with Markov Chain Monte Carlo (MCMC) methods. The main disadvantage about MCMC methods is the large number of iterations they need to sample the posterior distributions of model parameters, especially for large data sets. On the other hand, variable selection remains a challenging problem due to its combinatorial search space, where Bayesian models are a promising solution. In this work, we study how to accelerate Bayesian model computation for variable selection in linear regression. We propose a fast Gibbs sampler algorithm, a widely used MCMC method, that incorporates several optimizations. We use a Zellner prior for the regression coefficients, an improper prior on variance and a conjugate prior Gaussian distribution, which enable data set summarization in one pass exploiting an augmented set of sufficient statistics. Thereafter the algorithm iterates in main memory. Sufficient statistics are indexed with a sparse binary vector to efficiently compute matrix projections based on selected variables. Discovered variable subsets probabilities, selecting and discarding each variable, are stored on a hash table for fast retrieval in future iterations. We study how to integrate our algorithm into a database management system (DBMS), exploiting aggregate User-Defined Functions for parallel data summarization and stored procedures to manipulate matrices with arrays. An experimental evaluation with real data sets evaluates accuracy and time performance, comparing our DBMS-based algorithm, with the R package. Our algorithm is shown to produce accurate results, scale linearly on data set size and run orders of magnitude faster than the R package.

1 Introduction

Bayesian statistics has been successfully applied in fundamental problems such as regression, variable selection, classification and clustering, among others. The distinctive characteristic about Bayesian models is their unique ability to incorporate prior knowledge into the statistical analysis [2], honest estimation of parameter uncertainty and flexibility in accommodating complex data interrelationships via hierarchical modeling [5, 10]. Bayesian inference typically involves estimation via stochastic search methods, such as Markov Chain Monte Carlo (MCMC) algorithms, to generate a long sequence of samples from the posterior distribution function [2, 5, 14]. In this work, we study how to optimize MCMC methods to estimate a Bayesian model on large data sets. The computational challenge about MCMC methods for Bayesian models is that they generally require a large number of iterations. Therefore, for a large data set residing on secondary storage, MCMC processing can be extremely slow. We study how to accelerate the Gibbs sampler, a common MCMC method used to perform variable selection in linear regression [6, 8]. With that motivation in mind, we introduce a fast Gibbs sampler for variable selection, based on a combination of Gaussian and non-informative priors, exploiting augmented sufficient statistics, hashing of highly probable variable combinations and block-based matrices. The optimized algorithm iterates in main memory on a compressed representation of the data set. We study how to integrate our fast Gibbs sampler with a DBMS.

[*]© ACM, 2014. This is the author's **unofficial** version of the work. The official version of this article was published in ACM Transactions on Knowledge Discovery from Data (TKDD Journal), 2014. DOI: 10.1145/

2 Definitions and Background

2.1 Data Set

Let $X = \{x_1, \ldots, x_n\}$ be the input data with n points, where each point has d numeric explanatory variables and a dependent variable Y. To make mathematical notation intuitive, we use matrices with n columns. Thus the data set X is a $(d+1) \times n$ matrix, where x_i represents a column vector (i.e., a $d \times 1$ matrix). In a similar manner, Y is $1 \times n$ matrix and y_i represents the ith output value. To avoid confusion, X_j (upper case) refers to the jth variable and x_i (lower case) is the ith point. We will refer to a statistical model as Θ, composed of several vectors, matrices and error statistics.

2.2 Linear Regression and Variable Selection

Linear regression is a predictive model capturing a linear relationship between a response variable Y, and a set of explanatory variables $X = X_1, \ldots, X_d$ [5]. The regression model is given by

$$Y = \beta^T \mathbf{X} + \epsilon, \tag{1}$$

where β is a $(d+1)$-vector (column vector) of regression coefficients $\beta = [\beta_0, \beta_1, \ldots, \beta_d]$, β_0 is the Y axis intercept and ϵ represents error with a Gaussian distribution. The predicted y_i value is defined as $\hat{y_i} = \hat{\beta}^T x_i$, where $\hat{\beta}$ is estimated, in general, with the least squares minimization method [10]. To make notation more concise, X is augmented with n 1s stored in an extra variable X_0 and β_0 represents the Y axis intercept. This augmented matrix is denoted by $\mathbf{X}$.

In general, not all d variables help predict Y. Therefore, it is generally necessary to select subsets of variables such that they better explain Y. The main computational issue is that there are 2^d potential variable combinations. A model of selected variables is represented by a d-vector $\gamma \in \{0,1\}^d$, where variable X_j is selected if $\gamma_j = 1$. Let k be the number of selected variables for a given γ: $k = \gamma^T \gamma$. Vector γ is used as an index on selected variables throughout the paper.

2.3 Gibbs Sampler for SSVS

The two most common MCMC methods in Bayesian statistics are the Gibbs sampler and the Metropolis-Hasting algorithm [5]. We focus on the Gibbs sampler, given its generality and popularity.

Gibbs Sampler

Given K model parameters $\Theta = \{\theta_1, \theta_2, \ldots, \theta_K\}$ the Gibbs sampler builds a Markov chain by sampling from the posterior distribution of θ_k fixing the remaining $K-1$ model parameters. That is, it builds a sequence of N iterations where at each iteration I it draws K samples from the conditional distributions

$$\pi(\theta_k^{[I]} | \theta_1^{[I]}, \ldots, \theta_{k-1}^{[I]}, \theta_{k+1}^{[I-1]}, \ldots, \theta_K^{[I-1]}),$$

for $k = 1 \ldots K$. In order to explore the joint posterior distribution by sampling it is necessary to specify a prior distribution $\pi(\cdot)$ on each model parameter.

We study how to accelerate the Gibbs sampler to perform stochastic search for variable selection (SSVS) introduced in the seminal paper [6]. We follow the Gibbs sampler specified in [10], which provides an algorithm more suitable for optimization than [6] and [7]. Let $\beta_\gamma = \{\beta_0 \cup \beta_j | \gamma_1 = 1\}$ be the βs of selected variables and $\mathbf{X}_\gamma$ be a projection of $\mathbf{X}$ on those variables where $\gamma_j = 1$. Vector β_γ is $k+1 \times 1$ and matrix $\mathbf{X}_\gamma$ is $k+1 \times n$. Therefore, for the SSVS problem the desired model is $\Theta = \{\beta, \sigma, \gamma\}$. Then the Gibbs sampler iterates N times to build the sequence

$$\beta^{[0]}, \sigma^{[0]}, \gamma^{[0]}, \beta^{[1]}, \sigma^{[1]}, \gamma^{[1]}, \ldots, \beta^{[N]}, \sigma^{[N]}, \gamma^{[N]},$$

where the selected variable subsets will embedded in the sequence $\gamma^{[B]}, \gamma^{[B+1]}, \gamma^{[B+2]}, \ldots, \gamma^{[B+N]}$, where the most promising variable subsets γ are expected to appear with higher frequency after the Markov chain stabilizes after a "burn in" period of B iterations.

Prior Distributions Specification for Gibbs Sampler

We now specify the prior probability distribution of each model Θ parameter (recall $\Theta = \{\beta, \sigma, \gamma\}$) to sample from their corresponding posterior distribution, following the SSVS model presented in [10].

We start by explaining priors for β. Since it is infeasible to define a prior for every potential γ (2^d possibilities) we use a single global prior distribution corresponding to the full regression model where $k = d$ (i.e. all variables are selected). We use the well-known Zellner G-prior [3, 10, 9], in which β is sampled assuming the prior distribution $\pi(\beta|\sigma^2, \gamma)$ follows a normal distribution

$$\beta|\sigma^2, \gamma \sim N(\tilde{\beta}, c\sigma^2(\mathbf{X}\mathbf{X}^T)^{-1}), \tag{2}$$

where $\tilde{\beta}$ is a prior hyper-parameter (a parameter not in Θ) and $c > 0$ is an input parameter. This prior distribution corresponds to a two-component mixture of normal distributions on the βs, generally referred to as the spike-slab distribution [6], where the "spike" corresponds to point-mass at zero and the "slab" distribution corresponds to a Gaussian distribution with large variance. In an analogous manner, for a set γ of selected variables y is conditioned as follows:

$$y|\beta_\gamma, \sigma^2, X \sim N_n(\beta_\gamma^T \mathbf{X}_\gamma, \sigma^2 I_n). \tag{3}$$

In a similar manner to β, the coefficients vector of selected variables β_γ is sampled assuming a prior Gaussian distribution

$$\beta_\gamma|\sigma^2, \gamma \sim N(\tilde{\beta}_\gamma, c\sigma^2(\mathbf{X}_\gamma {\mathbf{X}_\gamma}^T)^{-1}), \tag{4}$$

where β_γ and σ represent model parameters to estimate.

We now specify the prior for σ. To avoid computing the prior correlation structure we opted for a non-informative prior for σ, following the priors specification given in [10]. Model parameter σ has an improper (Jeffrey's) non-informative prior specified as

$$\pi(\sigma^2|X) \propto \sigma^{-2}. \tag{5}$$

Although the prior on σ^2 is improper, it results in a proper posterior distribution (integrating to 1) not only for the conditional posterior distribution, but also for the marginal distributions [10]. This prior choice obviates the user having to introduce and specify hyper-parameters. Based on the β prior defined above and this non-informative prior for σ, the Gibbs sampler is simplified to only require the specification of $\tilde{\beta}$ and c.

Finally, we explain the prior specification for γ based on a Zellner G-prior [3], which has two versions: an informative prior for c and a non-informative prior for c [10] [1]. Vector γ has a uniform prior

$$\pi(\gamma|X) = 2^{-d}. \tag{6}$$

Vector γ is iteratively sampled for each dimension j with a conditional distribution on the remaining $d-1$ vector dimensions using

$$\pi(\gamma_j|X, Y, \gamma_1, \gamma_2, \ldots, \gamma_{j-1}, \gamma_{j+1}, \ldots, \gamma_d,) \propto \pi(\gamma|X, Y), \tag{7}$$

because the full conditional distribution is proportional to the distribution conditioning on X, Y only. Then based on this prior specification the posterior distribution of $\gamma|X, Y$ satisfies the following respective equations depending on the chosen prior:

[1] the Zellner G constant is c in our paper.

$$\begin{aligned}\pi(\gamma|X,Y) \propto\ & c^{-1}(c+1)^{-(k+1)/2}[YY^T \\ & - c(c+1)^{-1}\cdot Y\mathbf{X}_\gamma^T(\mathbf{X}_\gamma\mathbf{X}_\gamma^T)\mathbf{X}_\gamma Y^T \\ & - (c+1)^{-1}\tilde{\beta}_\gamma\mathbf{X}_\gamma\mathbf{X}_\gamma^T\tilde{\beta}_\gamma^T]^{-n/2}\end{aligned} \tag{8}$$

for the informative Zellner prior. Otherwise, it satisfies

$$\pi(\gamma|X,Y) \propto \sum_{c=1}^{\infty} c^{-1}(c+1)^{-(k+1)/2}[YY^T - c(c+1)^{-1}\cdot Y\mathbf{X}_\gamma^T(\mathbf{X}_\gamma\mathbf{X}_\gamma^T)\mathbf{X}_\gamma Y^T]^{-n/2} \tag{9}$$

for the non-informative Zellner prior, where c is specified to have a non-informative diffuse prior $\pi(c) = 1/c$ and it can be truncated at $c = 10^5$ [10]. This non-informative prior for c simplifies usage, but incurs on higher computational cost. A key advantage provided by γ is that the only potential values for γ_j are $\gamma_j = 0$ and $\gamma_j = 1$. Therefore, in order to decide if a variable is selected or not it suffices to compute $p_0 = \pi(\gamma_j = 0|X,Y)$ and $p_1 = \pi(\gamma_j = 1|X,Y)$, adding both probablities $p_0 + p_1$ to get a proper posterior on γ_j. Then $p_0/(p_0+p_1)$ above a randomized threshold in [0,1] decides if variable j is discarded or selected. Another advantage of the hierarchical version of the Zellner G-prior is that hyper-parameter $\tilde{\beta}$ can be set to $\tilde{\beta} = 0$, simplifying Gibbs initialization. Therefore, the only parameters that are necessary to specify for each Gibbs run are c, which can be interpreted as the prior odds that a given set of variables are excluded (i.e. if $\gamma_j = 0$ then $\beta_j \approx 0$) and N, a sufficiently large number of iterations. Large c values correspond to a very vague (flat) prior on β and they allow data to inform the variable selection. On the other hand, the Gibbs sampler iterates for a large number of iterations N until convergence or until the posterior distributions have been sufficiently explored. Posterior probabilities of each variable X_j can be computed by averaging γ over $N - B$ iterations. Variable subsets, represented by different explored γs, are ranked by their probability of appearance over the window of $N - B$ iterations.

3 Bayesian Variable Selection in One Pass

3.1 Generalizing Sufficient Statistics

The sufficient statistics for X to compute a family of linear models are n, L, Q [12], where n is the number of points, and L and Q are defined as follows:

$$L = \sum_{i=1}^{n} x_i \tag{10}$$

and

$$Q = XX^T = \sum_{i=1}^{n} x_i \cdot x_i^T. \tag{11}$$

We now extend n, L, Q so that the Gibbs sampler can work on summaries of X, Y. Let $\mathbf{Z} = [1, X, Y]$ be a $(d+2)\times n$ matrix where we store the augmented X matrix with 1s and Y together. Basically, $\mathbf{Z} = \{z_1, \ldots, z_n\}$ is mathematically the augmented data set of $(d+2)$-vectors. Recall $\mathbf{X} = [1, X]$ is the augmented X matrix with 1s. Let $\mathbf{Q} = \mathbf{X}\mathbf{X}^T$ be a $(d+1)\times(d+1)$ symmetric matrix. We compute generalized sufficient statistics, up to the second moment, with matrix

$$\Gamma = \mathbf{Z}\mathbf{Z}^T, \tag{12}$$

a $(d+2)\times(d+2)$ matrix. Then Γ in more detailed form is as follows:

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

Matrix Γ can be computed in a single pass, reading $\mathbf{X}$ once, updating the Q product in main memory. Notice $(XY^T)^T = YX^T$ and $YY^T \neq Y^TY$ and $\Gamma = \Gamma^T$. The linear regression model can be computed from $\mathbf{ZZ}^T$ as follows. Estimating β with the traditional least squares method [5] leads to $\hat{\beta} = (\mathbf{XX}^T)^{-1}\mathbf{X}Y^T$. Rewriting this equation based on Γ gives

$$\hat{\beta} = \mathbf{Q}^{-1}(\mathbf{X}Y^T). \tag{13}$$

Notice the parentheses indicate how the matrix product is derived only from Γ. Therefore, Γ is a sufficient summary of X, Y to solve linear regression in one pass. The properties below summarize the importance of Γ.

Property 1: Γ can be computed in a single pass, reading $\mathbf{X}$ once.

Property 2: The $\hat{\beta}$ vector of estimated regression coefficients with least squares minimization can be computed reading $\mathbf{X}$ once.

3.2 Optimized Sampling from Posterior Probability Distributions

Sampling the posterior distribution of β_γ can exploit $\mathbf{Q} = \mathbf{XX}^T$. Moreover, Y is not needed directly, but as the product with $\mathbf{X}^TY$, which is available in Γ. Therefore, β_γ can be computed from Γ, eliminating the need to read the data set X at every iteration.

Under a general Bayesian framework, sampling the posterior distribution of σ at iteration I requires reading Y to compute $\sum_i (y_i - \hat{y})^2$, which cannot be obtained from Γ. The main reason is that Γ summarizes Y with YY^T and as a product $\mathbf{X}^TY$. Therefore, we change the prior specification to eliminate this posterior probability computation. As discussed above, we simplify this posterior probability function using a non-informative prior. Then

$$\sigma^2 \sim \pi(\sigma^2|X) \propto \sigma^{-2}. \tag{14}$$

As discussed above, to simplify prior specification we use a Zellner G-prior, which introduces a hyper-parameter $\tilde{\beta}$. Based on Γ we optimize sampling from the posterior distribution of β as follows:

$$\beta|\sigma^2, \gamma \sim N(\tilde{\beta}, c\sigma^2\mathbf{Q}^{-1}). \tag{15}$$

In a similar manner, let $\mathbf{Q}_\gamma$ be $\mathbf{Q}$ projected on the variables selected by γ. Then

$$\beta_\gamma|\sigma^2, \gamma \sim N(\tilde{\beta_\gamma}, c\sigma^2{\mathbf{Q}_\gamma}^{-1}). \tag{16}$$

As mentioned above, based on the Zellner G-prior, sampling from the posterior distribution of γ does not depend on Y. Therefore, such computation can exploit Γ for both Zellner priors:

$$\pi(\gamma|X,Y) \propto (YY^T) - c(c+1)^{-1} \cdot (Y\mathbf{X}_\gamma^T)(\mathbf{Q}_\gamma)(\mathbf{X}_\gamma Y^T) - (c+1)^{-1}\tilde{\beta}_\gamma \mathbf{Q}_\gamma \tilde{\beta}_\gamma^T]^{-n/2} \tag{17}$$

for the Zellner informative prior. and

$$\pi(\gamma|X,Y) \propto \sum_{c=1}^{\infty} c^{-1}(c+1)^{-(k+1)/2}[(YY^T) - c(c+1)^{-1} \cdot (Y\mathbf{X}_\gamma^T)(\mathbf{Q}_\gamma)(\mathbf{X}_\gamma Y^T)]^{-n/2} \tag{18}$$

for the Zellner non-informative prior. Notice matrix expressions in parentheses are efficiently obtained from Γ. In summary, SSVS can iterate in main memory after computing Γ reading the data set once.

3.3 Projections of Generalized Sufficient Statistics based on Selected Variables

It is necessary to have an efficient mechanism to get projections of Γ as indicated by γ. In SSVS each Gibbs iteration will work on a subset of $X_1, \ldots, X_d$ as indicated by the binary vector γ. Thus we introduce further notation to compute a projection of Γ on specific dimensions indicated by γ. Let P be a $k \times d$ projection matrix, where $k = \gamma^T \gamma$, such that $P_{k'j} = 1$ if $\gamma_j = 1$ and $k' \leq j$ and $P_{k'l} = 0$ if $l \neq j$ and $\gamma_j = 0$ for $j = 1 \ldots d, k' = 1 \ldots k$. Then the matrix product $P\mathbf{X}$ is a projection of those dimensions X_j such that $\gamma_j = 1$. Therefore, $(P\mathbf{X})(P\mathbf{X})^T = P\mathbf{X}\mathbf{X}^T P^T = P\mathbf{Q}P^T$ are the $\mathbf{Q}$ projection indicated by γ. It is expected that $k < d$ to guarantee efficient processing in time $O(k^3)$. For instance, if $n = 100$, $d = 10$ and $k = 3$ then P is 3×10, projecting three dimensions such that $\gamma_j = 1$, resulting in a 3×100 matrix.

We will not explicitly compute matrix products involving P. Instead, we use γ to build an index on dimensions j such that $\gamma_j = 1$. Based on Γ and P we compute β_γ as follows: $\beta_\gamma = (P\mathbf{Q}P^T)^{-1} P\mathbf{Q}\tilde{\beta}$. In a similar manner, we compute the posterior probability $\pi(\gamma_j|X,Y)$ as follows:

$$\pi(\gamma_j|X,Y) = (c+1)^{-(p+1)/2}[YY^T - c(c+1)^{-1}Y\mathbf{X}^T P^T (P\mathbf{Q}P^T)^{-1} P\mathbf{X}Y^T]. \quad (19)$$

Notice several terms in this long equation are repeated in transposed form: $Y\mathbf{X}^T P^T = (P\mathbf{X}Y^T)^T$ and $P\beta_\gamma^T = (\beta_\gamma P^T)^T$.

3.4 Optimizing SSVS Algorithm in a DBMS

We assume main memory is limited and secondary storage is unlimited. Therefore, we assume it is not possible to store X in main memory, but it is feasible for Γ and Θ. We assume $d < n$ and n is large; this is the most common case to analyze large data sets. Integrating SSVS with a DBMS is difficult. From an algorithmic time complexity perspective, the main challenges are: iterating a large number of times to sample from the posterior distributions, reading X from secondary storage many times and re-computing $\pi(\gamma_j|X,Y)$. Consequently, we need to recompute $\mathbf{Q}_\gamma^{-1}$ for each dimension at each iteration. From a DBMS perspective, the main challenges are using a small amount of RAM (i.e. not loading data set in RAM), fast data set summarization to accelerate the computation of Γ and computing matrix operations accurately and efficiently. We now study how to integrate SSVS with a relational DBMS using SQL programming mechanisms, especially aggregate User-Defined Functions (UDFs).

Storage

In a DBMS the data set X is stored on a table with an additional column i identifying the point (i.e. a record identifier like customer id), which is not used for statistical analysis. The table for X in a horizontal layout is defined as $X(\underline{i}, X_1, X_2, \ldots, X_d, Y)$. Notice matrices X is mathematically manipulated with column vectors and each column will be stored as a table row in a DBMS.

Optimizations in a DBMS

Our SSVS algorithm for $d < n$ has two major phases: (1) computing Γ; (2) iterating SSVS in RAM. The first optimization is to compute Γ in an aggregate UDF, which a standard programming mechanism on any DBMS. The main advantages of this aggregate UDF is that Γ can be computed in parallel: parallel I/O, multithreaded processing interleaving I/O with CPU operations and array-based computation in RAM. A User-Defined Type (UDT) is the interface to pass the vector z_i to the UDF: it transforms table rows into vectors. The aggregate UDF is a significant improvement over previous solutions based on SQL queries and Table-Valued Functions [11]. SQL queries are evaluated in parallel, but do not allow arrays and matrix entries are converted to multiple columns, resulting in slow computation. On the other hand, TVFs allow arrays but I/O is strictly sequential, but with low RAM footprint. Aggregate UDFs remove these limitations but RAM usage grows proportional to the number of working threads: this results in each thread having its own local version of Γ. Let N be the number of processing units in a parallel system, each with its own RAM and disk. Then there are N local summarization matrices in the UDF processing threads $\Gamma_1, \Gamma_2, \ldots, \Gamma_N$, where $\Gamma_I = \{n_I, L_I, \mathbf{Q}_I\}$. Then $n = \sum_{I=1}^{N} n_I, L = \sum_{I=1}^{N} L_I, \mathbf{Q} = \sum_{I=1}^{N} \mathbf{Q}_I$. Therefore, $\Gamma = \sum_{I=1}^{N} \Gamma_I$.

For SSVS iterations, recall the augmented data set is $\mathbf{Z} = \{z_1, z_2, \ldots, z_n\}$. Sampling from the posterior probability distribution of $\pi(\gamma|X, Y)$ requires computing $(P\mathbf{Q}P^T)^{-1}$, $P\mathbf{Q}$. The most intensive computation is $(P\mathbf{Q}P^T)^{-1}$, which takes time $O(k^3)$. All those matrix operations are computed in RAM. Our algorithm is highly efficient, but it does not compute an approximation of Θ. In summary, X is read once and SSVS iterations are performed in main memory.

The second optimization is hashing on frequent variable subsets γ. We added an important optimization to accelerate the posterior probability computation for γ. Since the Markov chain will likely visit some highly frequent dimension combinations, indicated by γ, it does not make sense to recompute such probabilities over and over. Therefore, if a specific γ value was computed before, a hash key based on $\{j_1, j_2, \ldots\}$ s.t. $\gamma_{j_k} = 1$ will give us the pre-computed probability stored in a hash table. Then time goes down from $O(p^3)$ to $O(1)$ for frequent γ values. We will explain the dimension combination hashing when we present optimizations in a DBMS. In this case γ is converted to a string data type, which is the input key for a hashing function. Since we need to compute p_0 and p_1 to sample from the posterior, it is necessary to insert two keys into the hash table, one with $\gamma_j = 0$ and another one with $\gamma_j = 1$. Then if p_0 or p_1 are not found then they are inserted into the hash table. Otherwise, such probabilities are retrieved from the hash table. Collisions are handled with linear probing. Recall that SSVS is after γ dimension combinations such that their R^2 is slightly smaller than R^2 for the full d model. It is tempting to apply a downward closure optimization to γ (similar to frequent itemset computations) since γ can be seen as an itemset. Unfortunately, γ probabilities do not obey downward closure: adding one dimension $\gamma_j = 1$ to a previously rejected dimension combination may increase R^2. The size of the hash table is fixed before the stored procedure starts and it is inversely proportional to c: smaller c values require a larger hash table.

The third optimization is exploiting matrices stored as a single block for $\mathbf{Q}$. Matrices are allocated as one-dimensional arrays on contiguous memory space, like traditional numerical linear algebra algorithms [4]. Such arrays provide faster performance than two-dimensional arrays because memory is less fragmented. Also, address computation for an entry $[i, j]$ is faster. We apply a fast matrix inversion with an LU decomposition. This optimization is key to compute $(P\mathbf{Q}P^T)^{-1}$ (the projected matrix of $\mathbf{Q}$ based on γ), which is the bottleneck for SSVS. The remaining matrix products can be efficiently computed inside the stored procedure. Notice that the hash table can significantly decrease the number of times sub-matrices (projections) from $\mathbf{Q}$ need to be inverted.

3.5 Time Complexity and I/O Cost

We summarize time efficiency based on $O()$ and I/O cost for both SSVS versions. We assume row-based storage with either d values or n values per row. Also, here we assume k represents average γ length. Vector γ is used as a hash key to get probabilities in time $O(k)$. Since dimension combinations vary and we allocate ample space for the hash table there are generally no collisions. Therefore, the time to insert or retrieve from the hash table is $O(k)$.

Summarization with Γ is computed in time $O(d^2n)$. On the other hand, the I/O cost is n I/Os to read X. There is a second pass to compute regression error σ in time $O(dn)$ with an additional n I/Os. Once Γ is computed, each SSVS iteration per variable takes time $O(k^3)$, where $k < d$, which rapidly decreases to $O(k)$ in some iterations as the Markov chain stabilizes. During iterative behavior SSVS takes time $O(dk^3N)$, where N depends on the probabilistic distribution of data. Overall, time is $O(d^2n + dk^3N)$.

Table 1: Data sets sizes and time comparison: DBMS versus R package (defaults for input parameters c=1000, $N = 10000$; * means more than 3 hours).

Data set	d	n	DBMS	R
activity	41	2871079	46.0	*
airplane	40	7154	14.6	*
car	7	398	0.2	409
caterpillar	10	33	0.2	59
deathrate	15	60	0.7	112
isolet	617	7797	*	*
parkinsons	20	5875	2.1	*
slump	7	103	0.1	59
violence	101	1995	119.4	*
yearpredict	89	463715	18.0	*

4 Experimental Evaluation

We present an experimental evaluation analyzing accuracy and time performance with several real data sets. We use the R statistical package as the golden standard to compare accuracy. When comparing our optimized Gibbs algorithm running on the DBMS with the Gibbs sampler running on the R package, we analyze real data sets that can fit in main memory. Time complexity and scalability are analyzed with a real data set having high d, replicating rows several times to get larger n.

4.1 Setup and Data Sets

We present our experimental evaluation on the Microsoft SQL Server 2008 DBMS, which supports TVFs and aggregate UDFs. The physical row size limit was 8,000 bytes which translates into $d \approx 1000$. The hardware configuration for the DBMS server was one Quadcore CPU running at 2.13 GHz per core, 4 GB of RAM and 1 TB on disk. The aggregate UDF, the TVF and SP were programmed in the C# language.

The data sets (described below) were available as relational tables in the DBMS and as plain text files when analyzed by the R package. In general, the DBMS cache memory was cleared before each run to make sure the data set was read from secondary storage. Experiments were repeated seven times, discarding the minimum and maximum times, computing the average from the remaining five runs. All times are reported in seconds by default. In general, we stopped program execution when time went beyond one hour.

Table 1 shows d and n for the real data sets, used to evaluate time performance and accuracy. All data sets were obtained from the UCI Machine Learning repository.

4.2 Parameters Specified by the User

We used the Zellner informative prior by default since its computation is more efficient. Therefore, the two main algorithm parameters controlled by the user are c, the mixing constant in the Zellner prior and N, the number of iterations.

Recall there is a burn-in period given by B to get a stable Markov chain. Table 2 shows how c influences time. A small c value is used to trigger significant mixing between $\gamma_j = 0$ and $\gamma_j = 1$. At the other extreme, high c values favor $\gamma_j = 0$ (i.e., not selecting variable j). We can observe time grows as c decreases, but it does not vary significantly. Therefore, our optimized SSVS algorithm allows the user to explore the posterior distribution well with no concern about time performance. In general, getting an acceptable model Θ under MCMC methods requires many trial and error runs, where each run requires a large number of iterations (thousands for some problems).

Table 2: Parameters: varying c (mixing) and N (MCMC iterations); times in seconds (defaults for input parameters $c = 1000, N = 5000$).

	c					N		
Data set	10	100	1000	10000	100000	5000	10000	20000
activity	8.2	9.0	14.5	14.6	14.2	14.5	27.2	53.2
airplane	14.6	14.6	14.6	14.7	14.6	14.6	29.1	58.5
car	0.2	0.2	0.2	0.2	0.2	0.2	0.4	0.7
caterpillar	0.5	0.4	0.3	0.2	0.2	0.3	0.7	1.5
deathrate	1.6	0.9	0.9	0.7	0.6	0.9	1.9	4.1
parkinsons	3.8	2.9	2.4	2.2	2.1	2.4	4.4	8.7
slump	0.2	0.2	0.2	0.1	0.1	0.2	0.5	1.1
yearpredict	369.6	321.6	322.2	275.7	195.4	322.2	505.3	986.2
violence	113.9	110.4	113.4	119.4	117.2	113.4	220.4	445.4

Table 2 shows time growth for iterations in the thousands as the number of iterations doubles. Clearly, time grows in the same proportion exhibiting linear behavior. Based on previous research [11], in general $N = 5000$ can produce good results. Therefore, $N = 20000$ is a pessimistic guarantee about performance. Based on these experimental results, we set $c = 1000$, $N = 10000$ and $B = 1000$ as default values. When time performance is an issue in the R package, especially for large n, we set $N = 5000$ and $B = 500$ in order to finish in no more than three hours.

4.3 Performance and Optimizations

Table 1 compares our optimized Gibbs sampler running on the DBMS with the reference Gibbs sampler from [10] running on the R package. All data sets can be loaded in main memory by R. However, recall that the data set X is summarized with Γ. Therefore, the DBMS does not load X in main memory. In both cases the data set is initially read from secondary storage. We can see that in every case the DBMS is at least two orders of magnitude faster than the R package. For the wine data set, where n is largest the DBMS is four orders of magnitude faster than R. Time in R heavily depends on n and to a lesser extent on d. For the DBMS, since n is small to influence heavy I/O time does not show a dependence on n. On the other hand, time is impacted by d, but it is not proportional to 2^d. These results prove that our optimized Gibbs sampler is much faster even if R can load the data set in main memory. The main reason is that the γ projection of $\mathbf{XX^T}$ and the corresponding matrix inverse are recomputed by R at every iteration. The second reason is of course hashing the most frequent variables combinations, avoiding the entire re-computation altogether. Needless to say, R would be slower if the data set was read from secondary storage at every iteration (not much slower since MCMC computations are CPU bound).

We turn our attention to analyzing the impact of each optimization individually, turning it on and off. All optimizations are turned on by default in order to get shorter running times. Some optimizations may interact together; we explain when that happens in order to correctly interpret results. We focus on analyzing time growth varying d since d is the most challenging problem size. We analyze the Isolet data set, which has high $d = 617$. We randomly replicate points and randomly sample dimensions to obtain large data sets. As justified above, the default values for method parameters are $c = 1000$ and $N = 10000$.

Table 3 compares three programming solutions to compute Γ: with an SQL query using standard aggregations (sum() and count()), a table-valued function exploiting a cursor interface to read X and an aggregate UDF. The three summarization programming solutions read the data set once. Clearly, the aggregate UDF is two orders of magnitude faster than the SQL query and two to three times faster than the TVF. The trend indicates the gap between the TVF and the aggregate UDF increases as d grows. The most important reasons are parallel processing with multiple threads and the ability to exploit arrays for blocked matrices. The SQL

Table 3: Optimizations: Summarization with Γ, hashing γ and blocked matrices (Isolet data set, n =1M).

d	SQL query	TVF	Aggr. UDF	hash=Y	hash=N	Blocked=Y	Blocked=N
20	44	14	11	17	76	17	20
40	155	32	19	35	428	35	50
60	352	59	25	52	1374	52	105
80	762	92	33	96	*	96	317
100	1154	202	44	134	*	134	629

Table 4: Accuracy: Comparing DBMS and R package on variable probability estimation (absolute and relative error).

Dataset	max abs	avg abs	max rel	avg rel
car	0.003	0.001	0.188	0.054
caterpillar	0.008	0.004	0.190	0.069
deathrate	0.039	0.013	0.300	0.127
slump	0.008	0.003	0.263	0.111

query creates a wide table with $d + d^2/2$ columns, resulting in an inefficient aggregation for Γ. On the other hand, the TVF can exploit arrays, but works in a sequential fashion.

Table 3 shows the importance of the hashing optimization. The impact at low d is modest, but at high d it becomes so significant that the program cannot finish in one hour. This optimization simply avoids re-computing the probability of exactly the same variable combination turning one variable on and off when sampling. Notice that this optimization is based on the assumption that β does not change across iterations.

We now evaluate the importance of using matrices stored as a single block like numerical linear algebra algorithms [4], compared to bi-dimensional arrays, as shown on Table 3. We can see the difference becomes significant as d grows. The explanation is simple. Memory management is more efficient for matrices stored as a single block and address computation of one entry is faster as well.

4.4 Accuracy

We now analyze accuracy. Since MCMC methods work with random numbers it is expected to have variation in results. Table 4 compares probability estimation per variable as follows. We use the Gibbs sampler R program for variable selection from [10] as the reference. We ran the R program and our UDF with the same parameters: the mixing value c, burn-in period B and number of iterations N. We only show results for data sets when R could finish within 2 hours. Otherwise, R was stopped. The output were the probabilities per variable across all γ_I models. That is, we derived their prior probability adding γ_I and dividing by $N - B$, which are numbers in [0,1]. We then computed two error measures: absolute error and relative error per variable. Since probabilities are real numbers in [0,1] absolute error is a reasonable accuracy measure. Nevertheless, we also computed relative error taking the R estimation as the reference value (i.e., dividing absolute error by the value computed by R). Table 4 reports the average and maximum for both error measures. Notice the maximum relative error across all d variables is a very sensitive error measure, especially when probabilities are close to zero. We can see our Gibbs sampler has two digits of precision for average absolute error (i.e. 99% accurate) and about 96% accuracy considering maximum absolute error. For relative error, given the random nature of MCMC computations error is higher, but still low: around 90% for average relative error and 50% for maximum relative error, which again, it is a pessimistic error estimation.

5 Related Work

We start our discussion with work on the Gibbs sampler used on variable selection. Formulating variable selection as a Bayesian statistics problem was proposed in the seminal paper [6], where the Gibbs sampler builds a Markov chain with frequent variable subsets selected. This work presents the hierarchical formulation of the SSVS problem, defines informative priors for all parameters and discusses guidelines to assess convergence. Linear regression dates back a long time. A good introduction to linear regression and the least squares method to fit a line to a set of points is presented in [5]; our basic definitions closely follow this work. Accelerating Bayesian model computation on large data sets has been studied before [14], where the fundamental idea is to use importance sampling, giving higher weight to high probability regions of the posterior distribution, making an additional pass on the data set to adjust weights. The basic issue is that model accuracy is sacrificed since MCMC methods rely on sampling from the entire data set. This work focuses on Hidden Markov models (HMMs) and logistic regression and studies both the Gibbs sampler and the more general Metropolis-Hastings method. In contrast, our method gets an exact estimation of the posterior probabilities from the entire data set in one pass. For linear regression, importance sampling would be less efficient since it requires reading the data set several times and it rests on the assumption there is an efficient and accurate sampling mechanism in the DBMS. Later, [10] presents several mathematical simplifications to model equations to make Bayesian variable selection simpler. The most important contribution of this work is to explain the importance of a non-informative prior on the regression coefficients to simplify sampling from the posterior distribution. Then regression coefficients of selected variables at each iteration are estimated from common regression coefficients across all models. Adapting and extending this optimization, our generalized sufficient statistics become the only ingredient needed to sample variables at each iteration, eliminating the need to read the data set at each iteration. We want to point out that the reference Gibbs sampler programmed in R used in our experiments was the publically available version from [10]. Our optimized SSVS method can be extended beyond normal linear models to generalized linear models such as exponential family models (binary, Poisson, ordinal data) through latent Gaussian variable formulations, such as those proposed in [1].

Despite the prominence of linear regression in statistics there is little work studying how to accelerate it to analyze large data sets that cannot fit in RAM. Integrating linear regression with a DBMS has received limited attention. Generalized sufficient statistics for several statistical models are proposed in [12], where linear regression is one of such models. This works shows sufficient statistics enable faster computation reducing the number of passes the data set needs to be read. It is also shown the most efficient SQL mechanism to compute sufficient statistics in a DBMS are aggregate UDFs. This paper briefly studies only the full dimensional regression problem (i.e. it does not study variable selection). In contrast, we study how to accelerate the Gibbs sampler for variable selection and how to efficiently compute projections of sufficient statistics to select many subsets of variables.

The most closely related works are [11] and [13], where sufficient statistics for the full dimensional regression problem are adapted for Bayesian variable selection and Singular Value Decomposition (SVD), respectively. In [11] there is a less general and significantly less efficient derivation of the accelerated Gibbs sampler, based on an informative Zellner prior, which requires careful tuning of model parameters. In contrast, our optimized SSVS algorithm is based on a Zellner non-informative prior, resulting in an easier to use algorithm and faster sampling from the posterior distributions. This work exploits SQL queries and TVFs as the main mechanism to summarize the data set, but it does not explore aggregate UDFs which are much faster and enable parallel processing. From a numerical computation perspective, blocked matrices are not considered either. We should emphasize the aggregate UDF is a significant optimization over SQL queries and Table-Valued Functions (TVFs) [11]: SQL queries are evaluated in parallel, but do not allow arrays leading to matrix entries being stored as multiple columns. On the other hand, TVFs allow arrays, but I/O and processing is sequential. In other words, data set summarization with aggregate UDFs is significantly faster due to parallelism and arrays. Data summarization with sufficient statistics in our SSVS algorithm generalizes previous work on solving PCA with SVD [13]. There are several important differences though. We have been able to summarize the data set with a single matrix (Γ), augmenting the data set with 1s and the dependent variable Y, computing it with a single outer product. It was not evident that all equations involving X or Y could exploit Γ. The previous SVD algorithm exploited SQL queries, which could not fully exploit parallel processing and arrays for blocked matrix operations.

6 Conclusions

Variable selection is a fundamental problem in statistics whose solution requires exploring a large combinatorial search space. Bayesian MCMC methods solve this problem with a stochastic search. In this paper, we focused on optimizing the Gibbs sampler, a popular MCMC method, to select variables in linear regression. We introduced an optimized MCMC algorithm that can analyze a large data set resident on secondary storage in one pass, making stochastic search iterations in main memory on a summarized representation of the data set. Mathematically, the most difficult optimization aspect was to specify Zellner informative and non-informative priors to exploit sufficient statistics on every matrix equation. As a result, all posterior probability functions are based on rewritten equations involving augmented sufficient statistics, combining the explanatory variables and the dependent variable. From a database system perspective, the most important optimizations, in order of importance, were parallel data summarization with aggregate UDFs, dimension indexing for efficient projection, hashing variable subsets and block-based matrix operations. Hashing on frequent variable subsets avoids repeatedly computing the posterior probabilities turning each variable on and off. The resulting algorithm iterates in main memory by sampling from the posterior distribution of a summarized representation of the data set. Secondary benefits include avoiding the need to sample the data set or to reduce the number of iterations, which would hinder proper convergence of the MCMC method. Experiments on real data sets evaluate accuracy and performance comparing our DBMS-based algorithm and the R package. Our algorithm is shown to be as accurate as R, but orders of magnitude faster. By comparing the time to export a large data set, our approach is shown to be preferable even if the data set originally resides outside the DBMS. We emphasize our algorithm has a minimal set of parameters: only a mixing constant and a maximum number of iterations. We show the mixing constant, controlling the breadth of stochastic search, does not affect performance significantly. On the other hand, since each iteration is fast it becomes feasible to efficiently run the Gibbs sampler for a large number of iterations. Hashing frequent variable subsets accelerates iterations as the Markov chain stabilizes. The algorithm is theoretically and experimentally proven to have linear scalability on data set size, which tends to be most important challenge for large data sets. On the other hand, our algorithm has quasi-linear time complexity on dimensionality, which is significantly faster than an exhaustive search with exponential time complexity.

Given the difficulty to optimize Bayesian MCMC methods and the inherent combinatorial explosion of variable subsets there are many research issues. We intend to extend stepwise variable selection to be guided by stochastic search. Since Bayesian statistics allow an experienced analyst to incorporate existing model knowledge, we want to give the user the ability to incorporate priors on model parameters favoring parsimonious models, maintaining fast computation. It is necessary to develop further database optimizations for high dimensional data sets that cannot fit in main memory. In particular, we want to study external hash tables for a large number of variable subsets. Given alternative acceleration approaches, it is necessary to understand the accuracy and efficiency tradeoffs between sampling and summarization with sufficient statistics. The Gibbs sampler can only be applied when it is possible to sample the posterior of one model parameter conditioning on the rest, which is not always feasible. Therefore, it is necessary to study which mathematical and database optimizations can work on more general MCMC methods, such as the Metropolis-Hastings algorithm. Going beyond, linear regression is not the only problem which can benefit from Bayesian methods. We want to optimize Bayesian methods for other prominent statistical models such as logistic regression, Gaussian mixtures and Hidden Markov models, covering the whole unsupervised and supervised spectrum.

Acknowledgments

This work was supported by NSF grants grants IIS 0914861 and IIS 0915196.

References

[1] A. Albert and S. Chib. Bayesian analysis of binary and polychotomous response data. *Journal of the American Statistical Association*, 88(422):669–679, 1993.

[2] V. Baladandayuthapani, R.J. Carroll, and B.K. Mallick. Spatially adaptive bayesian penalized regression splines (p-splines). *Journal of Computational & Graphical Statistics*, Volume 14:378–394, 2005.

[3] G. Celeux, M. El Anbari, J.M. Marin, and C. P. Robert. Regularization in regression: Comparing Bayesian and frequentist methods in a poorly informative situation. *Bayesian Analysis*, 7(2):477–502, 2012.

[4] J.W. Demmel. *Applied Numerical Linear Algebra*. SIAM, 1st edition, 1997.

[5] A. Gelman, J.B. Carlin, H.S. Stern, and D.B. Rubin. *Bayesian Data Analysis*. Chapman and Hall/CRC, 2003.

[6] Edward I. George and Robert E. McCulloch. Variable selection via Gibbs sampling. *Journal of the American Statistical Association*, 88(423):881–889, 1993.

[7] E.I. George and R.E. McCulloch. Approaches for bayesian variable selection. *Statistica Sinica*, 7:339–374, 1997.

[8] F. Li and N.R. Zhang. Bayesian variable selection in structured high-dimensional covariate spaces with applications in genomics. *Journal of the American Statistical Association*, 105(491), 2010.

[9] F. Liang, R. Paulo, G. Molina, M.A. Clyde, and J.O. Berger. Mixtures of g priors for Bayesian variable selection. *Journal of the American Statistical Association*, 103(481), 2008.

[10] J.M. Marin and C.P. Robert. *Bayesian Core: A Practical Approach to Computational Bayesian Statistics*. Springer, 2007.

[11] M. Navas, C. Ordonez, and V. Baladandayuthapani. On the computation of stochastic search variable selection in linear regression with UDFs. In *Proc. IEEE ICDM Conference*, pages 941 – 946, 2010.

[12] C. Ordonez. Statistical model computation with UDFs. *IEEE Transactions on Knowledge and Data Engineering (TKDE)*, 22(12):1752–1765, 2010.

[13] C. Ordonez, N. Mohanam, C. Garcia-Alvarado, P.T. Tosic, and E. Martinez. Fast PCA computation in a DBMS with aggregate UDFs and LAPACK. In *Proc. ACM CIKM Conference*, 2012.

[14] G. Ridgeway and D. Madigan. Bayesian analysis of massive datasets via particle filters. In *Proc. ACM SIGKDD*, pages 5–13. ACM New York, NY, USA, 2002.