2010 International Conference on Computational and Information Sciences

Dynamic Methods for Missing Value Estimation for DNA Sequences

Fen Qin
Shippensburg University
1871 Old Main Drive
Shippensburg, PA 17257
Email: fq0205@ship.edu

Jeonghwa Lee
Shippensburg University
1871 Old Main Drive
Shippensburg, PA 17257
Email: jlee@ship.edu

***Abstract*—Many gene expressions in the DNA microarray and gene sequences have missing values in the datasets. It is critical to estimate these missing values accurately, because most of the existing algorithms for gene expression analysis require the complete DNA dataset as an input, which affects the performance of gene classifications. This paper introduces dynamic local least squares imputation (DLLSimpute), which selects local matrices dynamically and consequently uses more gene information each time to recover the missing values. Numerical results show that the DLLSimpute recovers missing values more accurately than the k-nearest neighboring imputation (KNNimpute) and the local least squares imputation (LLSimpute) regardless of the completeness of the datasets.**

I. Introduction

DNA data analysis algorithms have been developed to deal with missing values. These algorithms include the singular value decomposition imputation (SVDimpute) [1], weighted K-nearest neighbors imputation (KNNimpute) [2], least square imputation (LSimpute) [3] and local least squares imputation (LLSimpute) [4]. Microarray data were used for testing these methods.

These existing methods work well for datasets that include at least one complete gene in them, but they suffer from two main drawbacks. The first drawback is that they cannot use the information from genes which contain missing values, because the existence of these missing values impedes the usage of other observed values in said gene. In other words, if the datasets have a missing value at each row, these methods will fail to recover. This is a limitation of the performance of imputation. The other drawback is that these methods must use a fixed number of neighboring genes. Recently, the sequential local least squares imputation (SLLSimpute) [5] has been presented; it solved the previous two problems, estimated missing values sequentially and partially utilized these estimated values compared with other imputation methods. In the SLLSimpute, once the local matrix is selected, it is used to recover all the missing values. An improved KNNimpute method was discussed in [8].

This paper introduces dynamic local least squares imputation (DLLSimpute). Instead of using a fixed local matrix, the DLLSimpute uses the largest available local matrix, which solves the aforementioned issues. The DLLSimpute method was tested with the H1N1 Segment 4 Hemagglutinin (HA) gene sequences from the National Institutes of Health (NIH) pubMed [6]. Numerical results are compared with those of the KNNimpute and the LLSimpute.

The DLLSimpute method works by selecting the gene with the largest missing values in it as the target gene. Then one of the missing values in this gene sequence is recovered. After recovering a missing value, all the genes are sorted to select the next target gene. Note that the DLLSimpute dynamically selects the target gene and the size of the local matrices. This procedure is repeated until there are no missing values left in the DNA dataset. In the following section, the imputation process is described in detail.

II. Dynamic Local Least Squares Imputes

A. Dynamic search for the largest local matrix

$G \in R^{m\times n}$ denotes a gene expression data matrix with m genes and n experiments. Assume $m >> n$. In the matrix G, a row $g_i^T \in R^{l\times n}$ represents expressions of the i-th gene in n experiments:

$$g_1^T = (g_{i_0,1}, \cdots, g_{i_0,,j}, \cdots, g_{i_0,n}) \quad i_0 \in \{1, 2, \cdots, m\} \quad (1)$$

Next, the gene expression matrix is sorted with respect to the number of missing values in each gene. The gene in the first row of the expression matrix is then selected as the target gene, and the first missing value of this gene is chosen to determine the largest local matrix.

The details are in the following algorithm.

B. Algorithm of the DLLSimpute

Input: A matrix which contains the DNA sequence.
Output: A matrix which contains the DNA sequence without any missing values.
Step 1: Sort each row by the number of missing values.
Step 2: Find the first missing position in the first row.
Step 3: Starting from the missing value position (i,j), the column j is scanned. Increasing i, if the position (i,j) is a missing value, remove the whole row.
Step 4: Separate the rest of the matrix into left and right matrices, selecting the largest matrix between these two.
Step 5: Repeat Step 1 to Step 4 until there are no missing values in the matrix.

978-0-7695-4270-6/10 $26.00 © 2010 IEEE
DOI 10.1109/ICCIS.2010.115

C. Recover the first missing value by using the singular value decomposition (SVD)

Local least squares methods solve the following equation

$$\min_x \left\|A^T x - w\right\|_2. \tag{2}$$

Solving Eq. (2) is equivalent to solving the following equation

$$\min_x \left\|A^T x - w\right\|_2^2. \tag{3}$$

By the definition of the inner product, we have

$$\min_x \left\|A^T x - w\right\|_2^2 = \min_x (A^T x - w)^T (A^T x - w). \tag{4}$$

The Eq. (4) is the same as

$$\frac{\partial}{\partial x_j} \sum_{i=1}^{n-1} (A^T x - w)_i^2 = 2 \sum_{j=1}^{n-1} (A^T x - w)_j^T A_j^T = 0,$$
$$j = 1, 2, \cdots, k, \tag{5}$$

where $(A^T x - w)_i$ is the i-th component of the column vector. The Eq. (5) comes down to the critical point of $\left\|A^T x - w\right\|_2^2$, where

$$\sum_{j=1}^{n-1} A_j (A^T x - w)_j = 0, j = 1, 2, \cdots, k \tag{6}$$

and a vector form

$$\begin{pmatrix} A_1(A^T x - w)_1 \\ A_2(A^T x - w)_2 \\ \cdots \\ A_n(A^T x - w)_n \end{pmatrix} = A(A^T x - w) = 0. \tag{7}$$

The above are as follows

$$\begin{pmatrix} g_1^T \\ g_{s_1}^T \\ \vdots \\ g_{s_k}^T \end{pmatrix} \begin{pmatrix} \alpha & w_1 & w_2 & \cdots & w_{n-1} \\ b_1 & A_{1,1} & A_{1,2} & \cdots & A_{1,n-1} \\ \vdots & \vdots & \vdots & \cdots & \vdots \\ b_k & A_{k,1} & A_{k,2} & \cdots & A_{k,n-1} \end{pmatrix}$$

$$, g_{s_i}^T = \begin{pmatrix} b_i & A_{i,1} & A_{i,2} & \cdots & A_{i,n-1} \end{pmatrix},$$

where

$$g_1^T = \begin{pmatrix} \alpha & w_1 & w_2 & \cdots & w_{n-1} \end{pmatrix},$$

$$\begin{pmatrix} b_1 \\ \vdots \\ b_k \end{pmatrix} = \begin{pmatrix} g_{s_1}(1) \\ \vdots \\ g_{s_k}(1) \end{pmatrix},$$

$$A = \begin{pmatrix} A_{1,1} & A_{1,2} & \cdots & A_{1,n-1} \\ \vdots & \vdots & \cdots & \vdots \\ A_{k,1} & A_{k,2} & \cdots & A_{k,n-1} \end{pmatrix},$$

where g_1^T is a gene with a missing value (depicted as α in the first location of g_1^T) and $g_{s_i}^T, i = 1, 2, \cdots, k$ are the k-nearest neighbor gene vectors for g_1^T. Solutions of Eq. (7) involve the generalized inverse. Specifically, if the matrix A is reversible, we have

$$AA^T x = Aw \tag{8}$$

and the solution

$$x = (AA^T)^{-1} Aw = (A^T)^\dagger w, \tag{9}$$

where $(A^T)^\dagger = (AA^T)^{-1} A$, $(A^T)^\dagger$ is the pseudoinverse of A^T. Then, the missing value α can be solved as follows,

$$\alpha = \sum_{i=1}^{n-1} x_i b_i = b^T (A^T)^\dagger w. \tag{10}$$

D. Improvement of the SVD methods

The result obtained by the previous methods to calculate the pseudoinverse of the matrix $A \in R^{m \times n}$,

$$A^\dagger = V \begin{bmatrix} \Sigma_{r_A}^{-1} & 0 \\ 0 & 0 \end{bmatrix} U^T = V_{r_A} \Sigma_{r_A}^{-1} U_{r_A}^T \tag{11}$$

does not satisfy the 4 Moore Penrose equations [7] :

$$AA^\dagger A = A,$$
$$A^\dagger AA^\dagger = A^\dagger,$$
$$(AA^\dagger)^T = (AA^\dagger),$$
$$(A^\dagger A)^T = (A^\dagger A).$$

As a result, if the size of the local matrix is increased, the number of computational errors is also increased−that is, the SVD results become less accurate. There are five different ways to test the accuracy of the pseudoinverse:

$$||AA^\dagger A - A||_\infty, \quad ||A^\dagger AA^\dagger - A^\dagger||_\infty,$$
$$||(AA^\dagger)^T - AA^\dagger||_\infty, \quad ||(A^\dagger A)^T - A^\dagger A||_\infty,$$
$$||Ax - b||_\infty.$$

In this paper, the Hilbert matrix

$$A = H_{200*200} = \left(\frac{1}{i+j+1}\right)_{200*200}$$

is used to do the experiments. Table 1 shows the errors of five different test cases.

MATLAB TEST

Test cases	$pinv(A)$	$inv(A)$	$Vinv(D)U^T$
$\|\|AA^\dagger A - A\|\|_\infty$	0.7171e-5	5.8272	0.7171e-5
$\|\|A^\dagger AA^\dagger - A^\dagger\|\|_\infty$	0.7594e9	4.1592e20	0.7594e9
$\|\|(AA^\dagger)^T - AA^\dagger\|\|_\infty$	0.4702e-3	122.75	0.4702e-3
$\|\|(A^\dagger A)^T - A^\dagger A\|\|_\infty$	0.9385e-3	3.1764e13	0.9385e-3
$\|\|Ax - b\|\|_\infty$	0.1675e-3	59128	0.1675e-3

Table 1

As shown in Table 1, the $pinv(A)$ does not satisfy $A^\dagger AA^\dagger = A^\dagger$ and $inv(A)$ does not satisfy the 4 Moore Penrose equations. Algorithms from articles [9], [10] are used to reduce computational errors. Since U and V are orthogonal matrices, they do not lead to an error, which means that the computational errors are coming from $\Sigma_{r_A}^{-1}$. Usually, the matrix Σ_{r_A} is dependent on the accuracy of the system

executing its implementation. If the singular values $\sigma_i < eps$ (eps is the error bound), the system will set it to zero, then Eq. (11) will contain computation errors.

Suppose the singular values of the matrix A are $\sigma_1 \geq \sigma_2 \geq \cdots \geq \sigma_{r_A}$, then

$$\Sigma_{r_A}^{-1} = dial(\frac{1}{\sigma_1}, \frac{1}{\sigma_2}, \cdots, \frac{1}{\sigma_{r_A}}, 0, \cdots, 0). \quad (13)$$

Since the cumulative error is $\varepsilon_i, i = 1, 2, \cdots, r_A$, the singular values of the matrix A will become to $\sigma_i + \varepsilon_i, i = 1, 2, \cdots, r_A$, then

$$\Sigma_{r_A}^{-1} = dial(\frac{1}{\sigma_1 + \varepsilon_1}, \frac{1}{\sigma_2 + \varepsilon_2}, \cdots, \frac{1}{\sigma_{r_A} + \varepsilon_{r_A}}, 0, \cdots, 0), \quad (14)$$

where the maximum computing error is

$$\varepsilon\left(\Sigma_{r_A}^{-1}\right) = dial\left(\begin{array}{c} \frac{|\varepsilon_1|}{\sigma_1|\sigma_1+\varepsilon_1|}, \frac{|\varepsilon_1|}{\sigma_2|\sigma_2+\varepsilon_2|}, \\ \cdots, \frac{|\varepsilon_1|}{\sigma_{r_A}|\sigma_{r_A}+\varepsilon_{r_A}|}, 0, \cdots, 0 \end{array}\right). \quad (15)$$

If $\sigma_i + \varepsilon_i$ is close to zero, the error is significantly magnified, and, therefore, it should be set to zero.. The Eq. (15) becomes as follows,

$$\Sigma_{r_A}^{-1}(\alpha) = dial(\frac{1}{\sigma_1 + \varepsilon_1}, \frac{1}{\sigma_2 + \varepsilon_2}, \cdots, \frac{1}{\sigma_k + \varepsilon_k}, 0, \cdots, 0),$$
$$\sigma_i + \varepsilon_i \leq eps^{\alpha}, i = k+1, \cdots, r_A. \quad (16)$$

Experiment results using Eq. (16) are shown in Table 2. When α=0.70, the errors of the new algorithm are much smaller than those presented in Table 1. In particular, the error of $||A^{\dagger}AA^{\dagger} - A^{\dagger}||_{\infty}$ is nearly one million times smaller. When α=0.60, the pseudoinverse of Hilbert matrix $A = H_{200*200}$ does satisfy the 4 Moore Penrose equations, yet the solution of linear equations of α=0.60 is worse than α=0.70. This is due to the removal of more singular values which are close to zero. Therefore, α cannot be too small. thus α=0.70 is used in this research.

Test cases	$\alpha = 0.60$	$\alpha = 0.65$	$\alpha = 0.70$
$\|\|AA^{\dagger}A - A\|\|_{\infty}$	0.1373e-8	0.9632e-8	0.2709e-7
$\|\|A^{\dagger}AA^{\dagger} - A^{\dagger}\|\|_{\infty}$	0.3996	16.243	693.64
$\|\|(AA^{\dagger})^T - AA^{\dagger}\|\|_{\infty}$	0.4908e-8	0.2071e-6	0.1047e-5
$\|\|(A^{\dagger}A)^T - A^{\dagger}A\|\|_{\infty}$	0.3974e-7	0.6524e-7	0.1032e-5
$\|\|Ax - b\|\|_{\infty}$	0.4041e-4	0.1477e-4	0.5965e-5

Table 2

III. Numerical Results

The normalized root mean squared error (NRMSE) is used to measure accuracy.

$$\text{NRMSE} = \sqrt{\frac{mean[(\gamma_{estimated} - \gamma_{known})]^2}{std[\gamma_{known}]}},$$

where $\gamma_{estimated}$ are the estimated vectors for missing values, and γ_{known} are the known values. The mean and the standard deviation are calculated over missing values in the whole dataset. If $[\gamma_{estimated} - \gamma_{known}]^2$ is not equal to zero, it is set to one. This technique makes the NRMSE a little small, but it does not affect the result. We present the test results of the DLLSimpute, KNNimpute and LLSimpute in Table 3 and Figure 1.

	%	KNNimpute	LLSimpute	DLLSimpute
NRSME	1	0.1721	0.0820	0.1058
	2	0.1346	0.0750	0.1019
	5	0.0942	0.0509	0.0758
	10	0.0841	0.0413	0.0773
	20	0.0756	0.0393	0.0674

Table 3

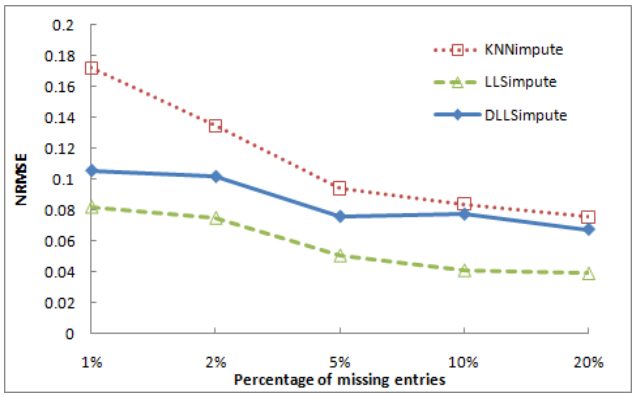

Figure 1

Table 3 and Figure 1 show the results for various percentages (1%, 2%, 5%, 10% and 20%) of missing entries. Although the LLSimpute shows better performance, the KNNimpute and LLSimpute methods failed to recover the missing values when at least one missing value was present in each gene. That is, at least one row should be complete. The DLLSimpute recovers the missing values regardless of completeness. Experiment results are shown in Table 4 and Figure 2.

	1%	2%	5%	10%	20%
DLLSimpute	0.1124	0.1020	0.0719	0.0768	0.0686

Table 4

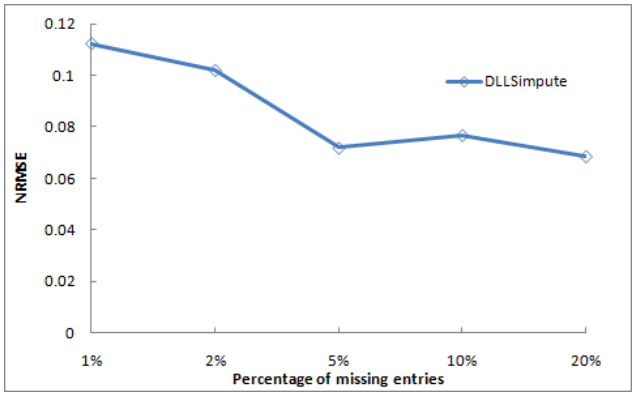

Figure 2

Table 4 and Figure 2 show that the DLLSimpute is not restricted by the position of the missing values. The DLLsimpute recovers all the missing values for the DNA sequences, which have a missing value in each row.

IV. Conclusion

Based on the experimental results, the DLLSimpute is a robust and accurate algorithm for recovering missing values. It is also a valuable imputation method when DNA gene sequences possess high missing rates. Although the LLSimpute grants better performance than the DLLSimpute when DNA sequences have complete rows, the DLLSimpute outperforms the LLSimpute approach when all the genes contain missing values. Depending on the structure of the incomplete datasets, the analysis presented here strongly suggests that the DLLSimpute is the ideal solution for data recovery.

Acknowledgment

This research is supported in part by the Miklausen-Likar grant from the Shippensburg University Foundation. We appreciate Dr. Dohoon Kim at Harvard Medical School for providing the insight of the dataset.

References

[1] T. Hastie, R. Tibshirani, G. Sherlock, M. Eisen, P. Brown and D. Botstein, *Imputing missing data for gene expression arrays*, Technical Report, Division of Biostatistics, Stanford University, 1999.

[2] O. Troyanskaya, M. Cantor, G. Sherlock, P. Brown, T. Hastie, R. Tibshirani, D. Botstein and R.B. Altman, *Missing value estimation methods for DNA microarrays*, Bioinformatics 17 (6) (2001) 520–525.

[3] T. H. BZ, B. Dysvik and I. Jonassen, *LSimpute: accurate estimation of missing values in microarray data with least squares methods*, Nucleic Acids Res. 32 (3) (2004) e34.

[4] H. Kim, G. H. Golub and H. Park, *Missing value estimation for DNA microarray gene expression data: local least squares imputation*, Bioinformatics 21 (2) (2005) 187–198.

[5] X. Zhang, X. Song, H. Wang and H. Zhang, *Sequential local least squares imputation estimating missing value of microarray data*, Computers in Biology and Medicine 38 (2008) 1112–1120.

[6] PubMed, U.S. National Library of Medicine, National Institutes of Health, http://www.ncbi.nlm.nih.gov/pubmed

[7] A. Albert, *Regression and the Moore-Penrose pseudoinverse*, Acdemic Press, INC, New York, 1972.

[8] Q. Cai, Q. Wu, H. Dong and H. Liu, *The research of missing value estimation of gene sequence based on improved KNN*, Computer Science & Education, ICCSE '09, 4th International Conference, 2009.

[9] Z. Drma and K. Veseli, *New Fast and Accurate Jacobi SVD Algorithm I*, SIAM Journal on Matrix Analysis and Applications, 29 (4) (2008) 1322-1342.

[10] Z. Drma and K. Veseli, *New Fast and Accurate Jacobi SVD Algorithm II*, SIAM Journal on Matrix Analysis and Applications, 29 (4) (2008) 1343-1362.

[11] William H. Press , Saul A. Teukolsky, William T. Vetterling and Brian P. Flannery, *Numerical Recipes: The Art of Scientific Computing*, Cambirdge University Press, 2007.