GSMMC 2025 Project Report

Randomized Matrix Sketching and Least Squares Methods for Classification Problems

Lander Besabe[1], Harshit Bhatt[2], Kendra Calman[3], Jessie Chen[2], Nipuni de Silva[4], Sucharitha Dodamgodage[4], Mahrokh Najaf[5], Rebecca Rodrigues[6], Thabo Samakhoana[7], Mentor: Henry Boateng[8]

Abstract

This report explores the use of randomized matrix sketching techniques, specifically Sparse Johnson-Lindenstrauss Transform (SJLT) and Subsampled Randomized Hadamard Transform (SRHT), as efficient alternatives to Gaussian-based projections in linear least-squares regression and low-rank matrix approximation for classification problems. Motivated by the scalability challenges of classical methods like singular value decomposition (SVD) in large-scale settings, the work evaluates the performance, storage efficiency, and computational feasibility of these sketching methods. To approximate the numerical rank of a data matrix, we apply the Johnson-Lindenstrauss theorem. The proposed sketching methods are applied to two real-world datasets, Wisconsin Diagnostic Breast Cancer (WDBC) and LC25000 (histopathology images), demonstrating that SJLT and SRHT can provide comparable accuracy to deterministic least-squares while significantly reducing memory and computation requirements. Additionally, SRHT is extended for non-power-of-two dimensions, making it suitable for scalability. Our findings suggest that randomized sketching is a viable tool for scalable algorithms for high-dimensional biomedical data.

Keywords— Randomized matrix sketching, Randomized singular value decomposition, Sparse Johnson-Lindenstrauss Transform, Subsampled Randomized Hadamard Transform, Least-squares, Subspace projection, Johnson-Lindenstrauss theorem, Classification

1 Introduction

Over the past decades, numerical linear algebra has played a fundamental role in the advancement of science and engineering [9]. The field is ever evolving, and much of the effort has been devoted to the development of deterministic methods for matrix computations. Matrix decomposition remains one of the most efficient tools in computational mathematics [10]. However, for modern large-scale applications, classical approaches, such as singular value decomposition, become prohibitively expensive.

To address this issue, in recent years, researchers have investigated the role of randomization in designing new methods for different problems, such as random sketching [15, 8] and least-squares regression [18, 21, 17].

In this work, we aim to explore the performance of randomized singular value decomposition (SVD) [10] for different choices of the sketching matrix. The classical implementation of randomized SVD [14] uses a Gaussian sketching matrix $\mathbf{\Omega} \sim \mathcal{N}(0, 1)$ which we compare with two other possible approaches to sketch a matrix: the sparse Johnson-Lindenstrauss transform (SJLT) [11, 19] and subsampled randomized Hadamard transform (SRHT) [3]. This work examines whether the randomized least-squares method works well when we replace the Gaussian matrix with the SJLT or SRHT sketch operators, and if the randomized subspace projection can be implemented with SRHT for a matrix $\mathbf{A}_{m\times n}$ when m is not a power of 2.

⋆ *All authors contributed equally to this work. The order of authorship is alphabetical by last name.*

[1]University of Houston [2]North Carolina State University [3]Cal Poly Pomona [4]Clarkson University [5]Marquette University [6]Rochester Institute of Technology [7]Johns Hopkins University [8]San Francisco State University

The motivation behind exploring these questions is that the SJLT sketching matrix requires extremely low storage space, whereas the Gaussian sketching matrix is more costly for large-scale problems because of its dense nature, requiring large storage to handle the problem. We also extend the idea of the original SRHT (where the matrix size is restricted) to handle large matrices without any restrictions on their dimensions.

For a thorough discussion on randomized numerical linear algebra, including theory and applications, the readers are referred to [10, 7, 13].

The rest of the report is organized as follows: Sec. 2 introduces the two methods that we examined in this report. Sec. 3 discusses the nature of the data that we utilize for the numerical tests and how we handle them without forming the entire matrix. Sec. 4 presents the numerical results for the two datasets considered using the proposed methods. We draw some conclusions and provide future perspectives in Sec. 5.

2 Theory and Methods

In this section, we discuss the theoretical background of the proposed approaches for the sketching of a matrix. We begin by introducing the least-squares problems and randomized SVD, and then present the algorithms inspired by SJLT and SRHT to sketch a matrix.

2.1 Least Squares (LS) Method

Consider a system of linear equations

$$\mathbf{A}\mathbf{x} = \mathbf{b}, \tag{1}$$

where $\mathbf{A} \in \mathbb{R}^{m\times n}$, $\mathbf{x} \in \mathbb{R}^n$, and $\mathbf{b} \in \mathbb{R}^m$.

We use the least squares technique when the exact solution to the problem in (1) does not exist or is not unique, e.g., overdetermined systems (more equations than unknowns) or noisy data. A least squares solution $\hat{\mathbf{x}}$ gives the orthogonal projection of vector $\mathbf{b}$ onto the column space of the matrix $\mathbf{A}$, minimizing the Euclidean norm of the residual

$$\|\mathbf{A}\mathbf{x} - \mathbf{b}\|_{\mathbf{2}}.$$

The residual vector $\mathbf{r} = \mathbf{b} - \mathbf{A}\hat{\mathbf{x}}$ satisfies the orthogonality condition

$$\mathbf{A}^{\mathbf{T}}\mathbf{r} = \mathbf{0},$$

which can be solved by using the normal equation

$$\mathbf{A}^{\mathbf{T}}\mathbf{A}\hat{\mathbf{x}} = \mathbf{A}^{\mathbf{T}}\mathbf{b}.$$

However, the result can be numerically unstable if the matrix product $\mathbf{A}^{\mathbf{T}}\mathbf{A}$ is ill-conditioned. Therefore, in practice, one can use methods such as QR decomposition or singular value decomposition (SVD) to solve the normal equation.

2.2 Randomized Singular Value Decomposition

Consider the problem of finding a low-rank approximation of a matrix $\mathbf{A} \in \mathbb{R}^{m\times n}$. In [6], it is shown that the truncated SVD is the best low-rank approximation of $\mathbf{A}$. The goal of randomized SVD is to efficiently decompose $\mathbf{A} \approx \mathbf{U}\mathbf{\Sigma}\mathbf{V}^{\mathbf{T}}$ where $\mathbf{U} \in \mathbb{R}^{m\times k}$ and $\mathbf{V} \in \mathbb{R}^{k\times n}$ are orthonormal matrices containing the approximate left and right singular vectors of $\mathbf{A}$ respectively, $\mathbf{\Sigma} \in \mathbb{R}^{k\times k}$ is the diagonal matrix containing the approximate largest k singular values of $\mathbf{A}$, and k is the approximate numerical (or target) rank of $\mathbf{A}$.

However, we assume that we cannot form the entire matrix $\mathbf{A}$, but we can access each column $\mathbf{a}_i \in \mathbb{R}^m$ of $\mathbf{A}$, i.e.,

$$\mathbf{A} = [\mathbf{a}_1 \quad \cdots \quad \mathbf{a}_i \quad \cdots \quad \mathbf{a}_n],$$

one at a time. First, we construct a sketching matrix $\mathbf{\Omega} \in \mathbb{R}^{n \times k}$ which allows us to compress the data for efficient subsequent computations. We assume that $\mathbf{\Omega}$ has the form:

$$\mathbf{\Omega} = \begin{bmatrix} \omega_1^T \\ \vdots \\ \omega_n^T \end{bmatrix},$$

with $\omega \in \mathbb{R}^k$. We exploit the fact that the matrix multiplication $\mathbf{A\Omega}$ is a sum of rank-1 matrices from outer products, i.e.,

$$\mathbf{Z} \coloneqq \mathbf{A\Omega} = \sum_{i=1}^{n} \mathbf{a}_i \omega_i^T \in \mathbb{R}^{m \times k}. \tag{2}$$

In [10], it is shown that an oversampling by p columns improves the approximation of the singular vectors, and the authors suggest that $5 \leq p \leq 20$ is sufficient. Hence, we consider $\mathbf{\Omega} \in \mathbb{R}^{n \times (k+p)}$ instead for p belonging to the aforementioned region.

To further improve accuracy, we also implement q subspace iterations [7]. This approach is known to help amplify the influence of the most dominant singular values. In our case, performing subspace iteration means

$$\mathbf{Z} \leftarrow \sum_{i=1}^{n} (\mathbf{a}_i \mathbf{a}_i^T)^q \mathbf{Z}.$$

Next, we compute an approximate orthonormal basis of the column space of $\mathbf{A}$ through the columns of the matrix $\mathbf{Q}$ in QR decomposition of $\mathbf{Z}$: $\mathbf{Z} = \mathbf{QR}$ where $\mathbf{Q} \in \mathbb{R}^{m \times (k+p)}$ and $\mathbf{R} \in \mathbb{R}^{(k+p) \times (k+p)}$ is an upper triangular matrix. We then project $\mathbf{A}$ onto the column space of $\mathbf{Q}$ and in this case, perform

$$\mathbf{B} = \begin{bmatrix} \mathbf{Q}^T\mathbf{a}_1 & \cdots & \mathbf{Q}^T\mathbf{a}_i & \cdots & \mathbf{Q}^T\mathbf{a}_n \end{bmatrix} \in \mathbb{R}^{(k+p) \times n}.$$

Since $\mathbf{B}$ is substantially smaller than $\mathbf{A}$, we can perform SVD efficiently on $\mathbf{B}$ to obtain

$$\mathbf{B} = \tilde{\mathbf{U}} \mathbf{\Sigma} \mathbf{V}^T.$$

For the approximation of the left singular vectors of matrix $\mathbf{A}$, we set

$$\mathbf{U} \approx \mathbf{Q}\tilde{\mathbf{U}}.$$

Lastly, we obtain the approximate low-rank approximation of $\mathbf{A}$ to be

$$\mathbf{A} \approx \mathbf{U}_k \Sigma_k \mathbf{V}_k^T,$$

where $\mathbf{U}_k$ is the matrix containing first k columns of $\mathbf{U}$, $\mathbf{\Sigma}_k$ is the diagonal matrix containing the largest k singular values in $\mathbf{\Sigma}$, and $\mathbf{V}_k^T$ is the matrix containing the first k columns of $\mathbf{V}$.

Note that if one only has access to the rows of matrix $\mathbf{A}$ and wish to pre-multiply $\mathbf{\Omega}$ to $\mathbf{A}$, a similar process may be performed. In fact, this is what we do for the least-squares approach concerning the SJLT sketching matrix.

The next subsection discusses an efficient way to select the target rank k through the Johnson-Lindenstrauss Theorem.

2.3 Johnson-Lindenstrauss (JL) Theorem

Theorem 1 (Johnson-Lindenstrauss (JL) Theorem) *For any $0 < \epsilon < 1$ and any integer n, let k be a positive integer such that*

$$k \geq \frac{24 \ln\ n}{3\epsilon^2 - 2\epsilon^3}. \tag{3}$$

Then for any set A of n points in $\mathbb{R}^m$, there is a map such that for all $\mathbf{u}, \mathbf{v} \in A$,

$$(1-\epsilon)||\mathbf{u}-\mathbf{v}||^2 \leq ||f(\mathbf{u})-f(\mathbf{v})||^2 \leq (1+\epsilon)||\mathbf{u}-\mathbf{v}||^2, \textit{i.e.,}$$
$$(1-\epsilon) \leq \frac{||f(\mathbf{u})-f(\mathbf{v})||^2}{||\mathbf{u}-\mathbf{v}||^2} \leq (1+\epsilon).$$

The proof of the JL theorem is available in [5].

JL Theorem for our purposes: Let $\mathbf{A} \in \mathbb{R}^{m\times n}$, i.e., $\mathbf{A}$ has n (columns) vectors $\mathbf{a}_i$, $i = 1, \ldots, n$ in $\mathbb{R}^m$. Let $\mathbf{R}$ be a Gaussian matrix with $\mathbf{R} \in \mathbb{R}^{k\times m}$ and $\mathbf{R}_{ij} \sim \mathcal{N}(0,1)$ *i.i.d.* Then, with high probability, $f(\mathbf{a}_i) = \frac{1}{\sqrt{k}}\mathbf{R}\mathbf{a}_i$ is a map that satisfies the JL Theorem.

$$(1-\epsilon) \leq \frac{||R(\mathbf{a}_i-\mathbf{a}_j)||^2}{||\mathbf{a}_i-\mathbf{a}_j||^2} \leq (1+\epsilon)$$

In this work, we study the performance of randomized least-squares when the standard Gaussian projection matrix is replaced by a Sparse Johnson–Lindenstrauss Transform (SJLT) sketch operator (see Section 2.3). We also explore the feasibility and effectiveness of implementing randomized least-squares using a Subsampled Randomized Hadamard Transform (SRHT) described in Section 2.4 in the case where the number of rows m of $\mathbf{A}$ is not a power of 2.

2.4 Sparser Johnson-Lindenstrauss Transforms (SJLT)

An SJLT [12] matrix is a structured, sparse matrix whose entries have two possible non-zero values. $\mathbf{\Omega} \in \mathbb{R}^{k\times n}$, where n is the number of columns in $\mathbf{A}$ and k is as the smallest integer which satisfies (3). The matrix $\mathbf{\Omega}$ is an SJLT matrix with α non-zeros in each row, each nonzero drawn from $\{1/\sqrt{\alpha}, -1/\sqrt{\alpha}\}$ with equal probability [20]. Since the matrices are binary, the nonzero values are always known, so we do not need to store the values at these nonzero positions. We can avoid performing any matrix multiplications in our algorithm, instead generating the $\mathbf{\Omega A}$ matrix and $\mathbf{\Omega b}$ directly. For each row, we only need to get the random α positions and randomly assign values drawn from $\{1/\sqrt{\alpha}, -1/\sqrt{\alpha}\}$ at those positions. In this study, we assigned the random α positions using two different methods. In the first method, the positions were selected uniformly at random from all n columns. In the second method, the interval from 1 to n was divided into α sub-intervals, and one non-zero position was randomly selected from each sub-interval.

Algorithm *Sparser Johnson-Lindenstrauss Transforms (SJLT)*

Require: Data matrix $\mathbf{A} \in \mathbb{R}^{m\times n}$, target vector $\mathbf{b} \in \mathbb{R}^m$, projection size α, convergence threshold ε, method $\in \{1, 2\}$
Goal: Approximate solution $\mathbf{x}$

1. Initialize $\mathbf{\Omega A} \in \mathbb{R}^{k\times n}$, $\mathbf{\Omega b} \in \mathbb{R}^{k\times 1}$ with zeros
2. **for** $row = 1$ **to** k **do**
3. **if** method $= 1$ **then**
4. Pick α random indices $i_1, \ldots, i_\alpha \in \{1, \ldots, m\}$
5. Assign random signs $s_j \in \{\pm 1\}$ with equal probability
6. Scale rows $\mathbf{A}_{i_j}$ by $s_j/\sqrt{\alpha}$
7. Set $\mathbf{\Omega A}[row, :] = \sum_{j=1}^{\alpha} \frac{s_j}{\sqrt{\alpha}} \mathbf{A}_{i_j,:}$
8. **else if** method $= 2$ **then**
9. Partition $\{1, \ldots, m\}$ into α intervals
10. Select one index from each interval uniformly at random

11. Proceed as in steps 5–7
12. **end if**
13. Set $\mathbf{\Omega b}[row] = \sum_{j=1}^{\alpha} \frac{s_j}{\sqrt{\alpha}} \mathbf{b}_{i_j}$
14. **end for**
15. Solve $\mathbf{\Omega A} \cdot \mathbf{x} = \mathbf{\Omega b}$ using SVD
16. **Return** solution vector $\mathbf{x}$

2.5 Subsampled randomized Hadamard Transform (SRHT)

Another sketching approach [1] is to use a matrix $\mathbf{\Omega}$ containing subsets of the rows of a random Hadamard matrix. The interest in this method lies in the highly structured nature of $\mathbf{\Omega}$ which may be leveraged to reduce the computational cost of the sketching process.

To understand this structure, we introduce the definition of a normalized Hadamard matrix $\mathbf{H}_m \in \mathbb{R}^m$ and the SRHT sketching matrix:

Definition 1 *Let $m = 2^p$ for some $p \in \mathbb{Z}^+$. A normalized Hadamard matrix $\mathbf{H}_m$ of order m is an $m \times m$ matrix whose entries are either $+1$ and -1 such that $\mathbf{H}_m \mathbf{H}_m^T = m\mathbf{I}_m$.*

Definition 2 *Let k and $m = 2^p$ be fixed positive integers with $k \ll m$. An SRHT matrix $\mathbf{\Omega}$ is a $k \times m$ matrix of the form*

$$\mathbf{\Omega} = \sqrt{\frac{m}{k}} \mathbf{P} \mathbf{H}_m \mathbf{D}, \tag{4}$$

where $\mathbf{D} \in \mathbb{R}^{m \times m}$ is a diagonal matrix with entries independently drawn from $\{+1, -1\}$ with equal probability, $\mathbf{H}_m \in \mathbb{R}^{m \times m}$ denote a normalized Hadamard matrix, and $\mathbf{P} \in \mathbb{R}^{k \times m}$ consists of k rows selected uniformly at random from the identity matrix $\mathbf{I}_m \in \mathbb{R}^{m \times m}$.

One can follow the Sylvester's construction to build the normalized Hadamard matrix:

$$\mathbf{H}_1 = \frac{1}{\sqrt{2}}[1]$$
$$\mathbf{H}_{2^m} = \frac{1}{2^{m/2}} \begin{bmatrix} \mathbf{H}_{2^{m-1}} & \mathbf{H}_{2^{m-1}} \\ \mathbf{H}_{2^{m-1}} & -\mathbf{H}_{2^{m-1}} \end{bmatrix}.$$

Note that the Hadamard matrix $\mathbf{H}_m$ only works for m which are some power of 2. This is a bottleneck of this method as this means that we cannot use the typical SRHT matrix for a matrix with arbitrary size. To address this, we develop a different construction of the SRHT matrix. First, consider the element-wise definition of $\mathbf{H}_m$:

$$(\mathbf{H}_m)_{i,j} = \frac{1}{2^{m/2}} (-1)^{\sum_l i_l j_l}$$

where i_l and j_l are the bit digits of i and j, respectively, and $i = j = 0$ for the $(1, 1)$ entry.

This means that instead of constructing the entire Hadamard matrix, we compute only the rows corresponding to those selected by $\mathbf{P}$, which we do not need to build explicitly. For example, if row 5 has been randomly chosen, we construct the 5th row of the Hadamard matrix by fixing $i = 4$ and iterating over $0 \leq j \leq m - 1$. This gives us the first m columns $\mathbf{PH}_d$, where d is the closest power of 2 that is larger than m. Hence, this approach does not require m to be a power of 2 and requires less storage. Moreover, this is equivalent to padding zeros to the data matrix $\mathbf{A}$ and multiplying $\mathbf{\Omega}$ to $\mathbf{A}$ so we can extend $\mathbf{A}$ to a matrix of size $d \times d$ without requiring extra storage.

2.6 Subspace Projection

Consider a vector $\mathbf{y}$ that is not in the space $W = \text{span}\{\mathbf{u}_1, \ldots, \mathbf{u}_m\}$, with $\{\mathbf{u}_i\}_{i=1}^m$ mutually orthonormal. Its projection is given by $\text{proj}_W \mathbf{y} \coloneqq \mathbf{U}\mathbf{U}^T\mathbf{y}$ where

$$\mathbf{U} = [\mathbf{u}_1 \quad \cdots \quad \mathbf{u}_m] .$$

For an example of a geometric rendering of orthogonal projection on a subspace, see Fig. 1.

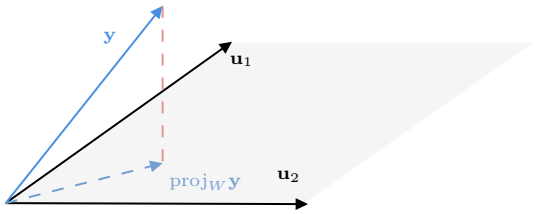

Figure 1: Projection of a vector $\mathbf{y}$ on the space spanned by the orthogonal vectors $\mathbf{u}_1$ and $\mathbf{u}_2$.

The quality of the projection depends on how low the value of the norm of the difference between $\mathbf{y}$ and $\text{proj}_W \mathbf{y}$ is, i.e., how far the vector $\mathbf{y}$ is from the subspace W defined by the columns of U.

For our numerical tests, we use the orthonormal matrix $\mathbf{U}$ given by the (randomized) SVD of the matrix $\mathbf{A}$ and project the data points $\mathbf{y}$ onto the subspace defined by $\mathbf{U}$.

In the WDBC data presented in Sec. 3, we construct two subspaces, one for malignant and one for benign. Meanwhile, in the LC2500 data also described in Sec. 3, we construct five subspaces: colon adenocarcinomas, benign colonic tissues, lung adenocarcinomas, lung squamous cell carcinomas, and benign lung tissues using a set of training images. In the validation stage, we project the set of validation images to the five subspaces and predict that they belong to the subspace in which they are closest to using the 2-norm:

$$\|\mathbf{y}_{\text{val}} - \mathbf{U}\mathbf{U}^T\mathbf{y}_{\text{val}}\|_2.$$

For more details on other methods regarding randomized projection on subspaces for classification problems, the readers are referred to [4].

3 Datasets Discussion

3.1 Data Introduction

In this project, we use two datasets:

1. The Wisconsin Diagnostic Breast Cancer (WDBC) dataset is a collection of data used for research and machine learning applications related to breast cancer diagnosis. It contains 560 samples (patients), each representing a fine needle aspirate (FNA) of a breast mass. The data includes 30 features derived from the images of cell nuclei, along with a diagnosis label indicating whether the sample is benign or malignant. Also, the samples are split into a test set (300 patients) and a validation (260 patients) set [16].
2. LC25000 is an image dataset with 25,000 color images in 5 classes. Each class contains 5,000 images of the following histologic entities: colon adenocarcinoma, benign colonic tissue, lung adenocarcinoma, lung squamous cell carcinoma, and benign lung tissue. Additionally, all images are 768 x 768 pixels in size and are in jpeg file format [2].

3.2 Data Preparation for Computation

The WDBC dataset is clean and ready to use for this project's purposes. However, the LC25000 dataset contains RGB images of size 768×768 pixels, so we needed to convert it to numerical data which we can make computations with. To obtain this, each image is converted to grayscale, shrunk by half, and then converted to a vector in $\mathbb{R}^{1\times(384\times384)}$.

4 Results

In this section, we present several numerical results for the implementation of the two randomized sketching methods, i.e., SJLT, SRHT, using the classification problems associated to the WDBC and LC25000 datasets 3.1.

4.1 Wisconsin Diagnostic Breast Cancer (WDBC)

For SJLT, the hyperparameters α and ϵ must be selected to optimize the prediction accuracy. Since there is no closed-form method to determine the optimal α, we use a Monte Carlo simulation to identify the value of α that produces the highest prediction accuracy, see Fig. 2. We set $\epsilon = 0.5$ as mentioned in the literature.

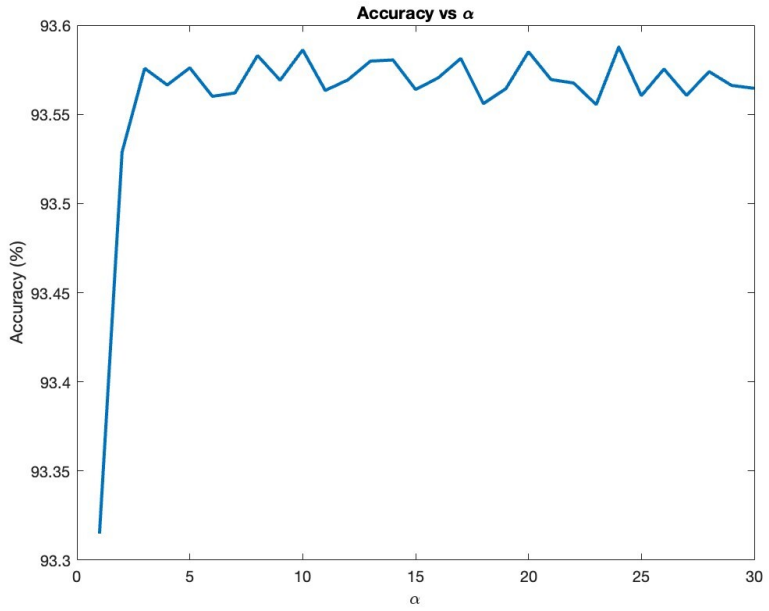

Figure 2: Among the tested values, $\alpha = 4$ resulted in the highest accuracy on the WDBC dataset.

The following table shows the accuracy of using deterministic least-squares (LS), SJLT, and SRHT for classifying malignant or benign samples from the WDBC for both the training data (second row) and validation data (third row). For all the methods considered, we use 300 training data points and 260 validation data points.

Accuracy (%)	LS	SJLT	SRHT
Training	95.3333	94.6667	95.3333
Vaidation	96.9231	95.7692	96.9231

We note that their accuracies are comparable but SJLT and SRHT performs more efficiently and thus are less computationally costly.

4.2 Histopathology images data (LC25000)

The LC25000 dataset contains data for two types of cancer: colon cancer and lung cancer. See Section 3.1 for the full description.

To test the randomized least-squares with SJLT and SHRT sketching method, we use the colon cancer subset, which includes images classified as either benign or malignant (adenocarcinomas), as our next dataset to evaluate the proposed method.

For the randomized subspace projection, we use all five classes in the LC25000 dataset. This means that we attempt to classify all images as to which classification in the dataset they belong to: colon adenocarcinoma, benign colonic tissue, lung adenocarcinoma, lung squamous cell carcinoma, and benign lung tissue.

In Table 1, we present the accuracy of each method in training and validation sets where we use an 80-20 split of the dataset. The second column presents the deterministic least-squares for classifying benign or malignant.

The SJLT method did not perform well in the LC25000 dataset. We ran the algorithm multiple times and selected the best-performing result, and the highest accuracy obtained is highlighted in blue. On the other hand, we ran SRHT in the validation set and it performed slightly better.

Moreover, the randomized subspace projection with SRHT sketching operator did better than the least-squares method in the training set and had an approximately 35% accuracy in the validation which is slightly better than guessing, i.e., 20% of guessing that classification of an images among five classes.

Accuracy (%)	LS	SJLT (10 trials)					SRHT	SP (SRHT)
Training	88.9875	57.8000	57.3000	59.0500	57.8000	59.3250	-	98.9100
		57.6125	58.1000	57.7500	58.6125	58.1375		
Validation	54.9000	53.55000	53.7500	55.2000	52.9000	56.3500	64.0000	35.42000
		54.3000	52.3500	52.9000	56.2500	54.6000		

Table 1: Accuracy of the deterministic least-squares (LS), LS with SJLT and SRHT, and subspace projection (SP) with SRHT.

We note that the goal of randomized numerical linear algebra is not to perform better than their deterministic counterparts but to perform as well as the deterministic methods but with increased efficiency.

5 Conclusions and future direction

In this work, we explored switching the Gaussian sketching matrix with two other possible choices: the sparse Johnson-Lindenstrauss Transform (SJLT) and subsampled randomized Hadamard Transform (SRHT), in the randomized singular value decomposition (SVD), least-squares, and subspace projection methods.

The sketching matrix provided by SJLT requires significantly lower storage space compared to the dense Gaussian matrix and does not require any matrix-matrix multiplication to perform the sketching. In each row, it only contains α non-zero entries which are either $1/\sqrt{\alpha}$ or $-1/\sqrt{\alpha}$. This means that this approach has low time and space complexity requirements.

On the other hand, the SRHT approach exploits the highly structured nature of the Hadamard matrix which may result in computational cost reduction in the sketching of the matrix. In this work, we extended this approach to matrices of arbitrary sizes, i.e., omitting the requirement that the matrix must have $m = 2^p$ rows.

For both choices, we assumed that we do not have full access to the entire data set, e.g., we can access the data of

one patient at a time instead of the data of all patients at the same time. This allows for scalable implementation of the algorithms for large-scale problems such as the LC25000 classification problem.

We tested these approaches on two classification problems concerned with determining whether a tumor is malignant or benign. In the Wisconsin Diagnostic Breast Cancer (WDBC) dataset, we are given 30 features from the images of the breast tissues from 560 patients. Both approaches performed well in both training and validation and are comparable to the deterministic linear least-squares method. In the LC25000 dataset, we are given microscopic images of lung and colonic tissues which are classified into five categories. Neither approach gave accurate classification for this dataset, especially in the validation stage. However, deterministic linear least-squares method also performed badly, with accuracy of 89% in the training data and 55% in the validation stage, in which our methods performed comparably.

The approaches in the classification problem presented in this paper may be improved in several ways. One can use kernels to transform the LC25000 dataset to be more well-suited to a linear least-squares problem. Secondly, one may use autoencoder networks to project the data into a low-dimensional latent space and perform least-squares in the latent space before projecting back to the high-dimensional space. Lastly, one may consider extending these approaches for randomized tensor decomposition methods in order to keep the structure of the images.

6 Acknowledgments

We would like to thank the organizers of the *2025 Graduate Student Mathematical Modeling Camp* and show our gratitude for the received support from the *Society for Industrial and Applied Mathematics (SIAM)* to attend the workshop. We are particularly grateful to Dr. Manuchehr Aminian and Dr. Richard O. Moore for their constant support at *California State Polytechnic University, Pomona*, during the entire program.

References

[1] Nir Ailon and Bernard Chazelle. Approximate nearest neighbors and the fast Johnson-Lindenstrauss transform. In *Proceedings of the thirty-eighth annual ACM symposium on Theory of Computing*, New York, NY, USA, May 2006. ACM.

[2] Andrew A Borkowski, Marilyn M Bui, L Brannon Thomas, Catherine P Wilson, Lauren A DeLand, and Stephen M Mastorides. Lung and colon cancer histopathological image dataset (lc25000). *arXiv preprint arXiv:1912.12142*, 2019.

[3] Christos Boutsidis and Alex Gittens. Improved matrix algorithms via the subsampled randomized hadamard transform. *SIAM Journal on Matrix Analysis and Applications*, 34(3):1301–1340, January 2013.

[4] Timothy I Cannings and Richard J Samworth. Random-projection ensemble classification. *Journal of the Royal Statistical Society Series B: Statistical Methodology*, 79(4):959–1035, September 2017.

[5] Sanjoy Dasgupta and Anupam Gupta. An elementary proof of a theorem of johnson and lindenstrauss. *Random Structures & Algorithms*, 22(1):60–65, 2003.

[6] Carl Eckart and Gale Young. The approximation of one matrix by another of lower rank. *Psychometrika*, 1(3):211–218, September 1936.

[7] M. Gu. Subspace Iteration Randomization and Singular Value Problems. *SIAM Journal on Scientific Computing*, 37:A1139–A1173, 2015.

[8] Stefan Güttel, Daniel Kressner, and Bart Vandereycken. Randomized sketching of nonlinear eigenvalue problems. *SIAM Journal of Scientific Computing*, 46(5):A3022–A3043, October 2024.

[9] Masoud Hajarian, Jinyun Yuan, and Ivan Kyrchei. Applications of methods of numerical linear algebra in engineering 2016. *Mathematical Problems in Engineering*, 2016:1–2, 2016.

[10] N. Halko, P.G. Martinsson, and J.A. Tropp. Finding Structure with Randomness: Probabilistic Algorithms forConstructing ApproximateMatrix Decompositions. *SIAM Review*, 53:217–288, 2011.

[11] Daniel M Kane and Jelani Nelson. Sparser Johnson-Lindenstrauss transforms. *J. ACM*, 61(1):1–23, 2014.

[12] Daniel M Kane and Jelani Nelson. Sparser johnson-lindenstrauss transforms. *Journal of the ACM (JACM)*, 61(1):1–23, 2014.

[13] Per-Gunnar Martinsson and Joel A Tropp. Randomized numerical linear algebra: Foundations and algorithms. *Acta Numerica*, 29:403–572, May 2020.

[14] Riley Murray, James Demmel, Michael W Mahoney, N Benjamin Erichson, Maksim Melnichenko, Osman Asif Malik, Laura Grigori, Piotr Luszczek, Michał Dereziński, Miles E Lopes, Tianyu Liang, Hengrui Luo, and Jack Dongarra. Randomized numerical linear algebra : A perspective on the field with an eye to software. 2023.

[15] Garvesh Raskutti and Michael W Mahoney. A statistical perspective on randomized sketching for ordinary least-squares. *J. Mach. Learn. Res.*, 17(213):1–31, 2016.

[16] Mangasarian Olvi Street Nick Wolberg, William and W. Street. Breast Cancer Wisconsin (Diagnostic). UCI Machine Learning Repository, 1993. DOI: https://doi.org/10.24432/C5DW2B.

[17] Franco Woolfe, Edo Liberty, Vladimir Rokhlin, and Mark Tygert. A fast randomized algorithm for the approximation of matrices. *Applied and Computational Harmonic Analysis*, 25(3):335–366, November 2008.

[18] Pengpeng Xie, Hua Xiang, and Yimin Wei. Randomized algorithms for total least squares problems. *Numerical Linear Algebra with Applications*, 26(1):e2219, January 2019.

[19] Yotam Yaniv, Pieter Ghysels, Osman Asif Malik, Henry A Boateng, and Xiaoye S Li. Construction of hierarchically Semi-Separable matrix representation using adaptive Johnson-Lindenstrauss sketching. 2023.

[20] Yotam Yaniv, Pieter Ghysels, Osman Asif Malik, Henry A Boateng, and Xiaoye S Li. Construction of hierarchically semiseparable matrix representation using adaptive johnson–lindenstrauss sketching. *Communications in Applied Mathematics and Computational Science*, 20(1):67–117, 2025.

[21] Anastasios Zouzias and Nikolaos M Freris. Randomized extended kaczmarz for solving least squares. *SIAM Journal on Matrix Analysis and Applications*, 34(2):773–793, January 2013.

7 Appendix

7.1 JL

```
%%
%%
%% Randomized least-squares with Johnson-Lindenstrauss (JL) Sketch
%% An Example:
% <https://archive.ics.uci.edu/dataset/17/breast+cancer+wisconsin+diagnostic
% Wisconsin Diagnostic Breast Cancer (WDBC)> dataset. 560 patients - split into
% test set (300 patients) and validation  (260 patients) set
%
% *Files*:
%
% *train.txt* : data for 300 patients (_Take a look_)
%
% *train_values.txt* : Indicator for malignant specimen (+1) or benign specimen
% (-1) (_Take a look_)
%
% *validate.txt* : data for 260 patients
%
% *validate_values.txt* : Indicator for malignant specimen (+1) or benign specimen
% (-1)
% Goal: Use least-squares to learn the training data and to predict the malignancy
    of the cells in the validation data.
%%
% * $A \in \mathbb{R}^{300 \times 30}$ : Each row corresponds to the data for
% each patient in the training set. Elements in a row correspond to the 30 features
% measured for a patient
% * ${\bf b} \in \mathbb{R}^{300}$ : vector whose domain is the set of patients
% in the training set. $b_i = 1$ if specimen $i$ is malignant, otherwise $b_i
% = -1$
%%
% *Solving inconsistent system* $Ax = {\bf b}$*using least-squares*:

% Load in matrix A and vector b
A = load('train.txt');
b = load('train_values.txt');

% Also load in the validation data
B = load('validate.txt');
z = load('validate_values.txt');
%%
% *Approach 1*: Solve the normal equations $A^TA \hat{x} = A^T{\bf b}$ directly

x1 = A'*A\A'*b; % x1 is the least-squares solution
%%
% _*How well does x1 model the training data?*_

```

```
45 b1 = sign(A*x1); % Specimen is malignant (-1) if prediction < 0 otherwise its benign
       (1)
46 disp("Accuracy on training data from Approach 1 is")
47 disp(100*(1-length(find(b-b1))/length(b)))
48 %%
49 % _*How well does x1 perform on the validation data?*_
50
51 y1 = sign(B*x1);
52 disp("Accuracy on validation data from Approach 1 is")
53 disp(100*(1-length(find(z-y1))/length(z)))
54 %%
55 % *Approach 2*: $\hat{x} = A^+{\bf b}$ where $A^+$ is the pseudo-inverse of
56 % $A$.
57
58 [U,S,V] = svd(A,"econ"); % Singular value decomposition of A
59 x2 = V*inv(S)*U'*b; % x2 is the least-squares solution and it's the same as the x1
       from Approach 1
60 %%
61 % _*How well does x2 perform on the training data?*_
62
63 b2 = sign(A*x2);
64 disp("Accuracy on training data from Approach 2 is")
65 disp(100*(1-length(find(b-b2))/length(b)))
66 %%
67 % _*How well does x2 perform on the validation data?*_
68
69 y2 = sign(B*x2);
70 disp("Accuracy on validation data from Approach 2 is")
71 disp(100*(1-length(find(z-y2))/length(z)))
72 %%
73 % *Approach 3*: LSQR - Solve $A^TA\hat{x} = A^T{\bf b}$ iteratively
74
75 x3 = lsqr(A,b,1e-8,150); % Iterative solve with tolerance 1e-8 and maximum of 150
       iterations
76 %%
77 % _*How well does x3 perform on the training data?*_
78
79 b3 = sign(A*x3);
80 disp("Accuracy on training data from Approach 3 is")
81 disp(100*(1-length(find(b-b3))/length(b)))
82 %%
83 % _*How well does x3 perform on the validation data?*_
84
85 y3 = sign(B*x3);
86 disp("Accuracy on validation data from Approach 3 is")
87 disp(100*(1-length(find(z-y3))/length(z)))
88 %% Randomized least-squares: Solve the least-squares problem in a lower-dimensional
       space
89 % A few references:
```

```
%%
% # Avron, H., Maymounkov, P., Toledo S., Blendenpik: Supercharging LAPACK's
% least-squares solver. _SIAM J. Sci. Comput._, Vol. 32, No. 3, pp. 1217-1236,
% *2010*
% # Drineas, P., Mahoney, M. W., Muthukrishnan, S., Sarlos, T., Faster least
% squares approximation. _Numer. Math._, 117, pp. 219-249, *2011*
% # Epperly, E. N., Fast and forward stable randomized algorithms for linear
% least-squares problems, _SIAM J. Matrix Anal. Appl._, Vol. 45, No. 4, pp.
    1782-1804,
% *2024*
%%
% *Johnson-Lindenstrauss (JL) Theorem*: For any $0 < \epsilon < 1$ and any integer
% $n$, let $k$ be the positive integer such that
%
% $k \ge \frac{24 \ln ~n}{3\epsilon^2 - 2\epsilon^3}$.
%
% Then for any set $A$ of $n$ points in $\mathbb{R}^m$, there is a map $f: \mathbb{R
    }^m
% \rightarrow \mathbb{R}^k$such that for all ${\bf u}, {\bf v} \in A$,
%
% $(1-\epsilon)||{\bf u} - {\bf v}||^2 \le || f({\bf u}) - f({\bf v})||^2 \le
% (1+\epsilon)||{\bf u} - {\bf v}||^2$, i.e.
%
% $$(1-\epsilon) \le \frac{|| f({\bf u}) - f({\bf v})||^2}{||{\bf u} - {\bf
% v}||^2} \le (1+\epsilon)$$
%
%
%
% _There's a proof in_: Dasgupta, S., Gupta, A., An elementary proof of a theorem
% of Johnson and Lindenstraus. _Random Structures and Algorithms_, 22(1), pp.
% 60-65, *2003*
%
% *JL Theorem for our purposes*: Let $A \in \mathbb{R}^{m\times n}$, i.e. $A$has
% $n$ vectors (columns) ${\bf a}_i$, $i=1,\ldots,n$ in $\mathbb{R}^m$. Let $R$
% be a Gaussian matrix with $R \in \mathbb{R}^{k \times m}$ and $R_{ij} \sim
% \mathcal{N}(0,1)$, i. i. d. Then $f({\bf a}_i) = \frac{1}{\sqrt{k}}R{\bf a}_i$
% is a map that satisfies the JL Theorem.
%
% $$(1-\epsilon) \le \frac{|| R({\bf a}_i - {\bf a}_j)||^2}{||{\bf a}_i - {\bf
% a}_j||^2} \le (1+\epsilon)$$
%%
% *An example of JL Theorem in action*: $\epsilon = 0.5$, $n=30$

epsl=0.5; %
k=ceil(24*log(30)/(3*epsl^2-2*epsl^3));
disp("Lower-dimensional space k = ")
disp(k)

[m,n]=size(A);
```

```
R=randn(k,m); % Gaussian matrix
rk = 1/k;
%The matrix of the linear transformation is $R/sqrt(k)$

nn1 = (n-1)*n/2;
daij=zeros(nn1,1); draij=zeros(nn1,1);
l=0;
for i=1:n-1
    ai=A(:,i);
    for j=i+1:n
        l=l+1;
        aij=ai-A(:,j);            % a_i - a_j
        daij(l)=norm(aij)^2;      % ||a_i - a_j||^2
        draij(l)=rk*norm(R*aij)^2; % ||R(a_i - a_j)/sqrt(k)||^2
    end
end
scale = draij./daij;
ulim = (1+epsl)*ones(nn1,1); llim=(1-epsl)*ones(nn1,1);

plot(1:nn1,llim,'b-',1:nn1,ulim,'k-',1:nn1,scale,'r.--',MarkerSize=4,LineWidth=2)
legend('$1-\epsilon$','$1+\epsilon$','$\frac{|| R({\bf a}_i - {\bf a}_j)||^2}{||{\bf
    a}_i - {\bf a}_j||^2}$',...
    'interpreter','latex')
%% I. Randomized least-squares: Normal Equations in k-Dimensional Space

xr1=ones(n,1); rerr=norm(A*xr1-b)/norm(b);
nruns=10;
for i=1:nruns % Run nruns times and choose the solution that gives the best relative
    error
    % JL sketch of the row-space of A
    R = randn(k,m)/sqrt(k);
    RA = R*A;
    % Sketch the right-hand side
    Rb = R*b;
    % Now solve the lower-dimensional normal equations
    xh = RA'*RA\RA'*Rb;
    cerr = norm(A*xh-b)/norm(b);
    if cerr < rerr
        xr1=xh; rerr=cerr;
    end
end
%%
% If $k << m$, we save time.
%
% |Note that instead of solving the normal equations directly, we can use LSQR
% (Blendenpik)|
%
% _*How well does xr1 perform on the training data?*_

```

```
br1 = sign(A*xr1);
disp("Accuracy on training data from randomized least-squares Approach I is")
disp(100*(1-length(find(b-br1))/length(b)))
%%
% _*How well does xr1 perform on the validation data?*_

yr1 = sign(B*xr1);
disp("Accuracy on validation data from randomized least-squares Approach I is")
disp(100*(1-length(find(z-yr1))/length(z)))
%% II. Randomized least-squares: Randomized-SVD to compute the pseudo-inverse $A^+$
% SVD on a sketch of the row-space of $A$

% Find the pseudo-inverse using randomized SVD
[Ur,Sr,Vr] = rsvd(A',k,1,5); % We can go over this later
xr2 = Ur*inv(Sr)*Vr'*b;
%%
% _*How well does xr2 perform on the training data?*_

br2 = sign(A*xr2);
disp("Accuracy on training data from randomized least-squares Approach II is")
disp(100*(1-length(find(b-br2))/length(b)))
%%
% _*How well does xr2 perform on the validation data?*_

yr2 = sign(B*xr2);
disp("Accuracy on validation data from randomized least-squares Approach II is")
disp(100*(1-length(find(z-yr2))/length(z)))
```

7.2 SJLT

```
alpha = 8;
epsl = 0.5;
N_train = 4000;
N_test = 1000;
m = 2*N_train;
n = 384^2;
method = 2;

num_trials = 10;
accu_train = zeros(num_trials, 1);
accu_test = zeros(num_trials, 1);
use_svd = 1;
for i=1:num_trials
    [RA, Rb] = R_matrix(alpha, epsl, m, n, method);
    x_sjlt = solveSystem(RA, Rb, use_svd);
    accu_train(i) = getAccuracy(N_train,N_train, 1, x_sjlt);
    accu_test(i) = getAccuracy(N_test,N_train, 0, x_sjlt);
end
%% Least Squares Solution
```

```
20
21 % get A_train
22 A_train = zeros(m, n);
23 b_train = zeros(m,1);
24 for i=1:N_train
25     b_train(i) = 1;
26     b_train(N_train + 1) = -1;
27     fname_aca = strcat("../Data/lung_colon_image_set/colon_image_sets/colon_aca/
           colonca", num2str(i),".jpeg");
28     fname_n = strcat("../Data/lung_colon_image_set/colon_image_sets/colon_n/colonn",
           num2str(i),".jpeg");
29     A_train(i, :) = load_data(fname_aca);
30     A_train(N_train+i, :) = load_data(fname_n);
31 end
32
33 % solve the system
34 use_svd = 1;
35 r = ceil(24*log(n)/(3*epsl^2-2*epsl^3));
36 x_ls = solveSystem(A_train, b_train, use_svd);
37
38
39 %% Test/Validation Data
40
41 % get accuracy
42 train = 1;
43 ls_accu_train = getAccuracy(N_train, N_train, train, x_ls);
44
45 % get validation accuracy
46 N_test = 1000;
47 train = 0;
48 ls_accu_test = getAccuracy(N_test, N_train, train, x_ls);
49 %%
50
51
52 %%
53 function x = solveSystem(A, b, use_svd)
54     if use_svd == 0
55         sol = A \ b;
56     else
57         [U, S, V] = svd(A, "econ");%rsvd(lhs',r, 1, 5);
58         rank_S = rank(S);
59         sol = V(:, 1:rank_S)* (S(1:rank_S, 1:rank_S) \ U(:, 1:rank_S)'*b);
60     end
61     x = sol;
62 end
63 %%
64 %
65
66 function accu = getAccuracy(M_test, M_train, train, x)
```

```
67      labels = zeros(2*M_test, 1);  % preallocate for labels
68      preds = zeros(2*M_test, 1);  % preallocate for predictions
69      for i=1:M_test
70          if train == 0
71              fname_aca = strcat('../Data/lung_colon_image_set/colon_image_sets/
                    colon_aca/colonca', num2str(M_train+i),'.jpeg');
72              fname_n = strcat('../Data/lung_colon_image_set/colon_image_sets/colon_n/
                    colonn', num2str(M_train+i),'.jpeg');
73          else
74              fname_aca = strcat('../Data/lung_colon_image_set/colon_image_sets/
                    colon_aca/colonca', num2str(i),'.jpeg');
75              fname_n = strcat('../Data/lung_colon_image_set/colon_image_sets/colon_n/
                    colonn', num2str(i),'.jpeg');
76          end
77          data_aca = load_data(fname_aca);
78          data_n = load_data(fname_n);
79          preds(i) = pos_neg(data_aca*x);
80          preds(M_test+i) = pos_neg(data_n*x);
81          labels(i) = 1;
82          labels(M_test+i) = -1;
83      end
84      accu = 100*(1-length(find(labels-preds))/length(labels));
85  end
86  %%
87  %
88
89  function v = pos_neg(r)
90      v = 1;
91      if r <= 0
92          v = -1;
93      end
94  end
95  function dta = load_data(fname)
96      img = imread(fname);
97      img = double(imresize(rgb2gray(img),0.5));
98      sz = size(img);
99      dta = reshape(img,[1, sz(1)*sz(2)]);
100 end
101
102 function [RA, Rb] = R_matrix(alpha, epsl, m, n, method)
103 % Inputs: alpha (scalar) - number of nonzero entries
104 %         epsl (scalar) - tolerance set to 0.5
105 %         m (scalar) - number of rows
106 %         n (scalar) - number of columns
107 %         method (scalar) - type of constructing operator
108 % Outputs: RA (kxn) - LHS of linear system
109 %          Rb (kx1) - RHS of linear system
110
111     k = ceil(24*log(n)/(3*epsl^2-2*epsl^3));
```

```
112
113     RA = zeros(k,n);  % preallocating matrix for RA
114     Rb = zeros(k,1);  % preallocating matrix for Rb
115     num_data = round(m/2);
116     for row = 1:k
117         % load the right rows
118         run_sum_RA = zeros(1, n); % initialize the row-th row of RA
119         run_sum_Rb = 0; % initializing running sum for RB
120         sgn = (2*randi([0,1], 1,alpha) - 1)/sqrt(alpha);
121         j = randperm(m, alpha);
122         if method == 2
123             interval_indexes = fix(linspace(1, m, alpha+1));
124             j = zeros(1, alpha);
125             for t = 1:alpha
126                 a = interval_indexes(t);
127                 b = interval_indexes(t+1); % bound
128
129                 j(1,t)=randi([a, b]);
130             end
131         end
132         for l=1:alpha
133             rl = j(l);%j(row, l);
134             if rl <= num_data
135                 malig = 1;
136                 fname = strcat("../Data/lung_colon_image_set/colon_image_sets/
                        colon_aca/colonca", num2str(rl),".jpeg");
137             else
138                 malig = -1;
139                 fname = strcat("../Data/lung_colon_image_set/colon_image_sets/
                        colon_n/colonn", num2str(rl-num_data),".jpeg");
140             end
141
142             row_l = sgn(l)*load_data(fname);%reshape(img,[1, sz(1)*sz(2)*sz(3)]);
143             run_sum_RA = run_sum_RA + row_l;
144             run_sum_Rb = run_sum_Rb + sgn(l)*malig;
145         end
146         RA(row, :) = run_sum_RA;
147         Rb(row,:) = run_sum_Rb;
148     end
149 end
```

7.3 SRHT

```
1 function R = srht(m, k)
2
3 p = randsample(m, k)-1;
4
5 %% PHD
6
```

```
p = (dec2bin(p)-'0')'; % get binary number of selected rows (permutation matrix)
d = sign(randn(m,1))'; % matrix D as a vector

cols = dec2bin(0:m-1)-'0'; % indices of columns of hadamard matrix
powers = (cols*p)'; % powers of elementwise P*H step without normalization
rows = (-1).^powers; % raise power of P*H step without normalization

n = ceil(log2(m)); % to scale to the next smallest power of 2
% this essentially truncated the concatenated hadamard matrix with zeros
% without storing zeros
R = sqrt(m/k)*(2^(-n/2)*rows).*d; % (PH)*D step
end
```

7.4 Subspace Projection

The code below is for the implementation of randomized SVD as described in Sec. 2.2:

```
function U = rsvdLargeScale(file_path, scale, epsl, p, q, varargin)

temp = cellstr(file_path);
index = find(temp{1} == '/', 1, 'last');
file_path_temp = extractBefore(temp{1}, index);
num_images = numel(dir(fullfile(file_path_temp, '*.jpeg')))*0.8;


train_image = imread(strcat(file_path, num2str(1), '.jpeg'));
train_image = imresize(train_image,scale);

train_image = rgb2gray(train_image);

[x,y] = size(train_image);

%% Computation of k

m = num_images; % number of patients i study
n = x*y;
epsl = 0.5;
k=ceil(24*log(n)/(3*epsl^2-2*epsl^3));

%% apply rSVD to each image
Z = zeros(n,k+p);

if isempty(varargin)
    disp('Using Gaussian Sketching')
    R = randn(m,k+p);
    for i = 1:num_images
        train_image = imread(strcat(file_path, num2str(i), '.jpeg'));

        train_image = imresize(train_image,scale);
        train_image = rgb2gray(train_image);
```

```
34          train_image = reshape(double(train_image),[], n)';
35
36          tempAR = train_image*R(i,:);
37          Z = Z + tempAR;
38      end
39
40      Z_temp = zeros(size(Z));
41      for j = 1:q
42          for i = 1:num_images
43              train_image = imread(strcat(file_path, num2str(i), '.jpeg'));
44
45              train_image = imresize(train_image,scale);
46              train_image = rgb2gray(train_image);
47              train_image = reshape(double(train_image),[], n)';
48
49              Z_i = train_image'*Z; % (X'*Z) step for each image
50              Z_temp = train_image*Z_i + Z_temp; % X*(X'*Z) for each image
51
52          end
53      end
54      Z = Z_temp;
55
56      [Q,~] = qr(Z,0); % QR factorization to get orthogonal vector Q
57
58      % Step 2: Compute SVD on projected Y=Q'*X;
59      temp_Y = [];
60      for i = 1:num_images
61          train_image = imread(strcat(file_path, num2str(i), '.jpeg'));
62
63          train_image = imresize(train_image,scale);
64          train_image = rgb2gray(train_image);
65          train_image = reshape(double(train_image),[], n)';
66
67          temp_Y = [temp_Y Q'*train_image ];
68
69
70      end
71      Y = temp_Y;
72
73      [UY,S,VT] = svd(Y,'econ');
74      U = Q*UY;
75
76  else
77      disp('Using SRHT Sketching')
78      R = srht(m,k+p)';
79      for i = 1:num_images
80          train_image = imread(strcat(file_path, num2str(i), '.jpeg'));
81
82          train_image = imresize(train_image,scale);
```

```
83          train_image = rgb2gray(train_image);
84          train_image = reshape(double(train_image),[], n)';
85
86          tempAR = train_image*R(i,:);
87          Z = Z + tempAR;
88      end
89
90      Z_temp = zeros(size(Z));
91      for j = 1:q
92          for i = 1:num_images
93              train_image = imread(strcat(file_path, num2str(i), '.jpeg'));
94
95              train_image = imresize(train_image,scale);
96              train_image = rgb2gray(train_image);
97              train_image = reshape(double(train_image),[], n)';
98
99              Z_i = train_image'*Z; % (X'*Z) step for each image
100             Z_temp = train_image*Z_i + Z_temp; % X*(X'*Z) for each image
101
102         end
103     end
104     Z = Z_temp;
105
106     [Q,~] = qr(Z,0); % QR factorization to get orthogonal vector Q
107
108     % Step 2: Compute SVD on projected Y=Q'*X;
109     temp_Y = [];
110     for i = 1:num_images
111         train_image = imread(strcat(file_path, num2str(i), '.jpeg'));
112
113         train_image = imresize(train_image,scale);
114         train_image = rgb2gray(train_image);
115         train_image = reshape(double(train_image),[], n)';
116
117         temp_Y = [temp_Y Q'*train_image ];
118
119
120     end
121     Y = temp_Y;
122
123     [UY,S,VT] = svd(Y,'econ');
124     U = Q*UY;
125
126 end
127
128
129 end
```