Trinity College

Honors Thesis

Fast Monte Carlo Algorithms for Computing a Low-Rank Approximation to a Matrix

Author:
Vlad Ştefan Burcă

Advisor:
Professor Mary Sandoval

Second Reader:
Professor Melanie Stein

April 2014

TRINITY COLLEGE

Abstract

Mary Sandoval
Department of Mathematics

Bachelor's of Science in Mathematics with Honors

Fast Monte Carlo Algorithms for Computing a Low-Rank Approximation to a Matrix

by Vlad Ştefan Burcă

Many of today's applications deal with big quantities of data; from DNA analysis algorithms, to image processing and movie recommendation algorithms. Most of these systems store the data in very large matrices. In order to perform analysis on the collected data, these big matrices have to be stored in the *RAM* (random-access memory) of the computing system. But this is a very expensive process since RAM is a scarce computational resource. Ideally, one would like to be able to store most of the data matrices on the memory disk (hard disk drive) while loading only the necessary parts of the data in the RAM. In order to do so, the data matrix has to be decomposed into smaller matrices. *Singular value decomposition* (SVD) is an algorithm that can be used to find a low-rank approximation of the input matrix, creating thus an approximation of smaller sizes. Methods like SVD require memory and time that are super-linear (increase at a rate higher than linear) in the sizes of the input matrix. This constraint is a burden for many of the applications that analyze large quantities of data. In this thesis we are presenting a more efficient algorithm based on Monte Carlo methods, `LinearTimeSVD`, that achieves a low-rank approximation of the input matrix while maintaining memory and time requirements that are only linear in regards to the sizes of the original matrix. Moreover, we will prove that the errors associated to this new construction method are bounded in terms of properties of the input matrix. The main idea behind the algorithm is a sampling step that will construct a lower size matrix from a subset of the columns of the input matrix. Using SVD on this new matrix (that has a constant number of columns with respect to the sizes of the input matrix), the method presented will generate approximations of the top k singular values and corresponding singular vectors of A, where k will denote the rank of the approximated matrix. By sampling enough columns, it can be shown that the approximation error can be decreased.

Acknowledgements

I would like to thank Professor Sandoval for having the patience of advising me through the struggle of trying to find a thesis subject in the first semester, and then for giving me invaluable support and guidance in the process of understanding my subject of choice during the second semester. Moreover, I would like to thank the Department of Mathematics for giving me the opportunity to dive deeply into a subject that sparked my interest while exploring the field of recommendation systems, but also for the amazing experience I had in this major over my past four years at Trinity College.

Contents

For my family, who made it possible for me to study in the USA and who always supported me in pursuing my goals.

Chapter 1

Introduction

Most of today's computer applications are required to gather a large amount of data in order to satisfy the users' needs. Examples of such applications include social networks, that need to gather information about social graphs, DNA analysis, where tests on different mutations of the DNA are performed, or recommendation systems, in applications such as online radio, that try to predict the musical tastes of the listener. Moreover, some of these applications might not rely fully on analysis of the gathered data, but they could use the data to improve the user experience. One such example is in e-commerce, where an online store could provide you with suggestions based on what you previously considered buying or actually purchased[1].

The field that studies the most efficient methods of storing and analyzing big quantities of data is called *Big Data Analysis*. Most of the times, this Big Data is represented in the data systems and algorithms as matrices of very large sizes.

In order to better understand what kind of information, and how this information, is stored in these matrices, we will present a couple of applications, of various backgrounds.

1.1 Applications

1.1.1 Blind Signal Separation

The Blind Signal Separation is an application that is most easily described by the *Cocktail Party Problem*. This problem assumes that we have **two sources** of some signal (in this case audio signal - a dialog) and **two receivers** (two microphones) situated at different distances from the sources. This setup is illustrated in Figure 1.1.

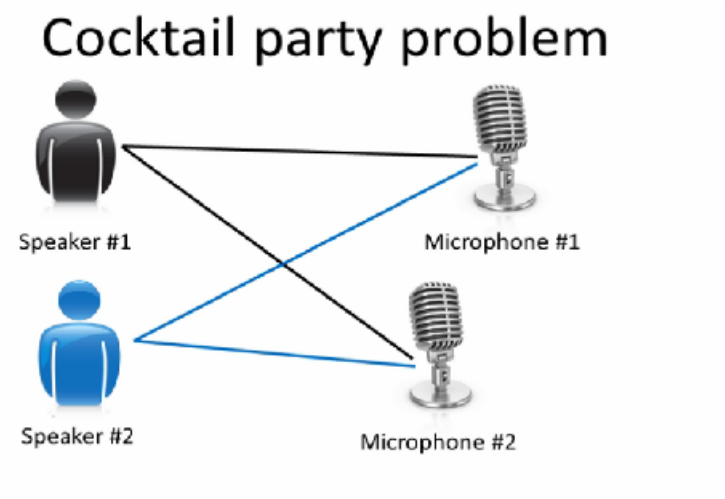

Figure 1.1: Ilustration of the Cocktail Party Problem

The goal of the problem is to separate the individual audio signals from the mixed signals that the microphones will record. For example, we would want the output of **Microphone 1** to be a signal where the audio signal of **Speaker 2** would be minimized (considered to be the *noise* in this case), thus emphasizing that of **Speaker 1** (the signal that we want to extract). The opposite output is expected from **Microphone 2**. This method of separating the two audio signals is called *Blind Signal Separation* (*Blind* because the method has no prior knowledge of the data setup and has to separate the signals only based on the recordings from the microphones).

To generalize the problem, assume we have **J sources** (speakers) and **N receivers** (microphones) [2]. Moreover, consider that each of the signals coming from a source is formed out of **M signal channels**. Therefore, we can now denote $\mathbf{Z} \in \mathbb{R}^{J \times M}$ (equivalent matrix notation to $\mathbf{Z} \in \mathcal{M}_{jm}(\mathbb{R})$, that we will be using throughout the rest of the thesis) as the matrix of the sources and, similarly, $\mathbf{X} \in \mathbb{R}^{N \times M}$ as

the matrix of the recordings, or set of observations. Moreover, since we want to mix the signals from the sources, we will introduce a mixing matrix $\mathbf{A} \in \mathbb{R}^{J \times N}$. Therefore, modeling the Cocktail Party Problem, we get the following equation:

$$\mathbf{X}^\intercal = \mathbf{A} \times \mathbf{Z}^\intercal$$

As described in more details in [2], one way of solving our problem is through the method of **Singular Value Decomposition** (discussed later, in Section 2.1.3). This method will approximate matrix $\mathbf{X}$ as follows:

$$\mathbf{X} = \mathbf{U} \times \mathbf{\Sigma} \times \mathbf{V}^\intercal$$

where $\mathbf{V}$ is the matrix that filters the mixed matrix $\mathbf{X}$ and $\mathbf{V}$ is the approximation matrix of the source [3]. We will go into more detail with this method in Section 2.1.3.

1.1.2 DNA Microarray Data

Another interesting application of Big Data Analysis is in bioinformatics, more specifically, in DNA microarray data experiments. Here, the goal is to pass a number of **M genes** through a series of biological processes (i.e. mutations) and observe how the genes react to these experiments. These experiments probe the genome-wide expression levels in **N different samples**. Thus, we can construct a similar matrix $\mathbf{Z} \in \mathbb{R}^{M \times N}$ (as the one in the Cocktail Party Problem) which is going to be the result of the biological process, ran over *N samples*. The main problem with the gene expression microarray experiments is that, most of the times, it generates data sets with missing values. Later on, this will be hard to process by other algorithms. Therefore, there needs to be a way to approximate the missing values, after these processes are applied to the initial set of genes.

A solution to this problem is Singular Value Decomposition [4]. With this method, knowing most of the data points from the matrix that resulted from the biological processes, we can approximate a matrix that does not have missing values and that can be used further on for other types of bioinformatic analysis.

1.1.3 Recommendation Systems

The last application that we are going to present in this chapter is also the one that represented the motivation for the topic of the thesis. Recommendation systems are widely used today, especially in online applications, such as e-commerce websites, online radio or music streaming services, or even online dating and matchmaking [5, Chapter 9]. The popularity of such systems grew along with the user data that was gathered throughout the internet: the more data available to analyze, the more inferences and predictions can be made. This will not only be the backbone of some of the previously mentioned applications, but it will also improve the user experience, overall.

Collecting large quantities of data, these applications end up in the same field of Big Data Analysis. In the case of recommendation systems, most of the data will be represented as matrices, where the rows represent **N elements** and the columns represent **M features** that we want to describe each element by. The elements could be things like products, songs, users, while the features are characteristics that can be quantified, usually numerically. Therefore, having this kind of data, we can organize it as a matrix $\mathbf{M} \in \mathbb{R}^{N \times M}$.

We will consider the example of recommendation systems in online radio or music streaming services. In this situation, our **N users** will be able to rate on songs that they listen to in order to improve their experience. Assuming that the service has a total of **M songs**, a user $\mathbf{u}$ can only rate a fraction of them, based on his/her listening habits. The users' musical preference matrix, $\mathbf{USERS} \in \mathbb{Z}^{N \times M}$, will be a matrix of ratings (alternatively, it could be a boolean matrix of *favorited* songs - 0 for not favorited, 1 for favorited). Therefore, the purpose of the recommendation system is to look at users with similar musical taste as the user $\mathbf{u}$ and recommend songs from their favorites to user $\mathbf{u}$. Note that we could have another matrix, $\mathbf{SONGS} \in \{0, 1\}^{M \times F}$, where $\mathbf{SONGS}$ represents a matrix of songs and the boolean features (of total size $\mathbf{F}$) that they could have. In this case, the recommendation system will not look for similarities between users, but for similarities between songs that user $\mathbf{u}$ liked and songs that are similar to his taste but that he did not discover yet.

Thus, the goal of recommending songs to a user $\mathbf{u}$ is actually represented by finding a way of approximating the ratings that this user might give to the unrated songs (which are empty entries in $\mathbf{USERS}$, the musical preference matrix). In order to predict those entries, we need to produce an approximation of $\mathbf{USERS}$. This approximation can be done with Singular Value Decomposition [6].

1.2 Motivation

As we presented in the previous subsection, the **Singular Value Decomposition** method has important applications in Big Data Analysis. Even though it seemed to be the solution to the three example applications we mentioned, it has an important computational limitation. This problem starts to affect the performance of the method when we are dealing with large sizes of the matrices that store the Big Data.

The **Pass Efficient Model** [7, Section 3.2] is a computational model that describes the performance inefficiencies that appear when an algorithm has to move large amounts of data to the RAM (Random Accessed Memory) of a computer, which, compared to the regular disk storage, is a scarce resource. Moreover, it suggests a different approach, using the actual disk space as storage for most parts of the matrices, and bringing only the computationally necessary parts in the RAM.

Since we are dealing with Big Data systems, most of the applications previously mentioned suffer from this performance issue. Therefore, in this thesis, we will describe a faster way of performing Singular Value Decomposition, by approximating the matrix of interest with a low-rank one, which is computationally easier to deal with. This will be done by using Fast Monte Carlo algorithms to sample a set of relevant columns from the total number of columns of the matrix of interest, and then constructing the approximation from these sampled columns.

Chapter 2

Background Review

In order to understand the details of the method that will be described in this thesis, we first have to present a review of some mathematical background. This will mostly consist of linear algebra background that will be useful for describing the steps of the algorithm (Section 2.1). Moreover, we will address methods of analyzing the performance of an algorithm (Big-O analysis) in order to understand the improvement that the Fast Monte Carlo method brings to the classical ways of computing a low-rank approximation to a matrix (Section 2.2). We will conclude this section by briefly presenting a matrix multiplication algorithm that also takes advantage of Fast Monte Carlo methods [8](Section 2.3). This will be used in one of the steps of the algorithm presented in a later chapter of this thesis.

2.1 Linear Algebra

For the upcoming sections, unless specified otherwise, we will use the following definitions and notations.

Let $m, n \in \mathbb{N}$. Let $x \in \mathbb{R}^n$ be a vector and denote x_i the ith element of x, with $i = 1, 2, \ldots, n$. Moreover, let $A \in \mathbb{R}^{m \times n}$ denote a matrix with $A^{(j)}$, $j = 1, 2, \ldots, n$, the jth column of A as a column vector and $A_{(i)}$, $i = 1, 2, \ldots, m$, the ith row of A as a row vector. Therefore, in order to represent the (i, j)th element of A, we can use $A_{ij} = (A^{(j)})_i = (A_{(i)})_j$.

2.1.1 Matrix and Vector Norms

The L^2 vector norm. For $x \in \mathbb{R}^n$, we define the following to be the L^2 norm of vector x:

$$|x| = \sqrt{\sum_{i=1}^{n} |x_i|^2}$$

The Spectral norm of a matrix. Let the matrix $A \in \mathbb{R}^{m \times n}$. Then the spectral norm of A is defined by:

$$\|A\|_2 = \max_{x \in \mathbb{R}^n, |x| \neq 0} \frac{|Ax|}{|x|}$$

The meaning of this norm is represented by how much the size of x is being changed by A.

The Frobenius norm of a matrix. For a matrix $A \in \mathbb{R}^{m \times n}$, we define the following as the Frobenius norm of A:

$$\|A\|_F = \sqrt{\sum_{i=1}^{m} \sum_{j=1}^{n} A_{ij}^2}$$

Moreover, if we denote $\mathbf{Tr}(M)$ as the matrix trace of a square matrix M (the sum of the diagonal elements of M) and $A^\intercal$ as the transpose matrix of A, then we observe that $\mathbf{Tr}(AA^\intercal) = \mathbf{Tr}(A^\intercal A) = \sum_{i=1}^{m} \sum_{j=1}^{n} A_{ij}^2$ and, therefore, we get the following identity for the *Frobenius norm* of A: $\|A\|_F^2 = \mathbf{Tr}(AA^\intercal) = \mathbf{Tr}(A^\intercal A)$.

The *Spectral norm* of A and the *Frobenius norm* of A are related through the following inequality:

$$\|A\|_2 \leq \|A\|_F \leq \sqrt{n}\|A\|_2$$

In addition, note that if $A \in \mathbb{R}^{m \times n}$ and $\{x^1, x^2, \ldots, x^n\}$ is any basis of $\mathbb{R}^n$, with $|x^i| = 1$ (orthonormal basis), then $\|A\|_F^2 = \sum_{i=1}^{n} |Ax^i|^2$. In order to show why this is true, consider the matrix A to be of the following form:

$$\begin{pmatrix} A_{11} & A_{12} & \cdots & A_{1n} \\ A_{21} & A_{22} & \cdots & A_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ A_{m1} & A_{m2} & \cdots & A_{mn} \end{pmatrix}$$

Letting $S = \sum_{i=1}^{n} |Ax^i|^2$, since x^i is an element of an orthonormal basis of $\mathbb{R}^n$, it implies that $S_i = \sum_{j=1}^{m} A_{ji}^2$, by the definition of the L^2 *vector norm*. Therefore, $S = \sum_{i=1}^{n} S_i = \sum_{i=1}^{n} \sum_{j=1}^{m} A_{ji}^2 = \sum_{j=1}^{m} \sum_{i=1}^{n} A_{ji}^2 = \|A\|_F^2$, by the definition of the *Frobenius norm*.

These identities and definitions will be useful when we will present the general ideas of the *Singular Value Decomposition*, in Section 2.1.3.

2.1.2 Eigenvalues, Eigenvectors, Singular Values and Singular Vectors

Eigenvalues and Eigenvectors. We will present a short background on what eigenvalues and eigenvectors are, how to compute them and why do we need them for the purpose of our presented method.

Definition 2.1. Let $A \in \mathbb{R}^{n \times n}$ be a square matrix. Suppose $\mathbf{x} \in \mathbb{R}^n$ is a *nonzero* vector and λ is a number (possibly 0) such that

$$A\mathbf{x} = \lambda\mathbf{x}$$

that is, $A\mathbf{x}$ is a scalar multiple of $\mathbf{x}$. Then $\mathbf{x}$ is called an **eigenvector** of A, and λ is called an **eigenvalue** of A. We say that λ is the eigenvalue associated with $\mathbf{x}$, and $\mathbf{x}$ is an eigenvector associated with λ. [9, Chapter 5.2]

In order to find eigenvalues, we start from the definition equation $A\mathbf{x} = \lambda\mathbf{x}$. Thus

$$\begin{aligned} A\mathbf{x} &= \lambda\mathbf{x} \\ A\mathbf{x} &= \lambda I\mathbf{x} \\ \lambda I\mathbf{x} - A\mathbf{x} &= 0 \\ (\lambda I - A)\mathbf{x} &= 0 \end{aligned}$$

And, since $\mathbf{x}$ is a *nonzero* vector, we can now see that

$$\begin{aligned} & \lambda \text{ is an eigenvalue of } A \\ \Leftrightarrow\ & (\lambda I - A)\mathbf{x} = 0 \text{ has a nontrivial solution} \\ \Leftrightarrow\ & (\lambda I - A) \text{ is singular (does not have a matrix inverse)} \\ \Leftrightarrow\ & det(\lambda I - A) = 0 \end{aligned}$$

Therefore, we can conclude this result with the following theorem.

Theorem 2.2. *Let $A \in \mathbb{R}^{n\times n}$ be a square matrix. The number λ is an eigenvalue of A if and only if $det(\lambda I - A) = 0$.*

We observe that the eigenvalues can only be applied to square matrices. In order to deal with a similar concept, but for matrices of type $A \in \mathbb{R}^{m\times n}$, we will now present some background on **singular values**. Their underlying theory still relies on eigenvalues. Moreover, as we will see in Section 2.1.3, this theoretical background represents the basis for the *Singular Value Decomposition* method.

Singular Value and Singular Vectors. We will present the main definitions and properties of *singular values* in order to establish the theoretical foundations for Section 2.1.3.

Definition 2.3. A *singular value* and a pair of *singular vectors* of a real square or rectangle matrix A are a nonnegative scalar σ and two nonzero vectors u and v such that ([10])

$$\begin{aligned} Av &= \sigma u \\ A^\intercal u &= \sigma v \end{aligned}$$

Moreover, vector v is called *right singular vector* of A and vector u is the *left singular vector* of A.

Let $A \in \mathbb{R}^{m\times n}$ be a matrix. In order to compute the singular values of A we will proceed as follows. Consider the matrix $A^\intercal A \in \mathbb{R}^{n\times n}$. This is a square matrix and thus, we can compute its eigenvalues. Let $\lambda_1 \geq \lambda_2 \geq \cdots \geq \lambda_n > 0$ be the real positive eigenvalues of $A^\intercal A$, in decreasing order. Then, the singular values of A are $\sigma_1 \geq \sigma_2 \geq \cdots \geq \sigma_p \geq 0$, where $\sigma_i = \sqrt{\lambda_i}$. Note that if $A \in \mathbb{R}^{n\times n}$ is a symmetric matrix with non-negative eigenvalues, then the eigenvalues and singular values coincide.

2.1.3 Singular Value Decomposition (SVD)

We will describe the general ideas behind *Singular Value Decomposition*. These will represent the motivation for the next chapter, where we will introduce the fast Monte Carlo method of computing a low-rank approximation to a matrix. Both SVD and the Monte Carlo method are ways of approximating a matrix A, but the Monte Carlo one is faster, while maintaining a bounded error for the approximated matrix.

Thus, we will start from the definition of singular values, Definition 2.3. Consider $A \in \mathbb{R}^{m\times n}$ with $rank(A) = r \leq p = min\{m, n\}$. Let $\sigma_1 \geq \sigma_2 \geq \cdots \geq \sigma_r > 0$ denote the positive singular values of

A, in decreasing order, and $\sigma_{r+1} = \cdots = \sigma_n = 0$. Moreover, let $\{u_j\}_{j=1}^m \in \mathbb{R}^{m\times 1}$ be the *left singular vectors* of A, and $\{v_i\}_{i=1}^n \in \mathbb{R}^{n\times 1}$ be the *right singular vectors* of A. We will consider the normalized singular vectors, such that $\|u\|_2 = \|v\|_2 = 1$. Each such pair (u_i, v_i) corresponds to a singular value σ_i: $Av_i = \sigma_i u_i$.

Now, consider the following matrices. Let $U \in \mathbb{R}^{m\times m}$ and $V \in \mathbb{R}^{n\times n}$ be the orthogonal matrices whose columns are the v_i's and, respectively, the u_i's, as follows

$$\begin{aligned} U &= \begin{pmatrix} u_1 & \vdots & u_2 & \vdots & \cdots & \vdots & u_m \end{pmatrix} \\ V &= \begin{pmatrix} v_1 & \vdots & v_2 & \vdots & \cdots & \vdots & v_n \end{pmatrix} \end{aligned}$$

Since U and V are orthogonal, by definition, we know that $U^\intercal U = I_m$ and $V^\intercal V = I_n$. Moreover, let $\Sigma \in \mathbb{R}^{m\times n}$ denote the the matrix of the form

$$\Sigma = \begin{pmatrix} D & \vdots & 0 \\ \cdots\cdots & & \\ 0 & \vdots & 0 \end{pmatrix}, \text{where } D \in \mathbb{R}^{r\times r} \text{ is of the form}$$

$$D = \begin{pmatrix} \sigma_1 & & \\ & \ddots & \\ & & \sigma_r \end{pmatrix}$$

We can summarize these identities in the following theorem

Theorem 2.4 (Singular Value Decomposition Theorem)**.** *Any $m \times n$ matrix can be factored into a singular value decomposition (SVD),*

$$A = U\Sigma V^\intercal \tag{2.1}$$

where $U \in \mathbb{R}^{m\times m}$ and $V \in \mathbb{R}^{n\times n}$ are orthogonal matrices and $\Sigma \in \mathbb{R}^{m\times n}$ is diagonal with $r = rank(A)$ of the form defined earlier. The p diagonal entries of Σ are usually denoted by σ_i for $i = 1, 2, \ldots, p$, where $p = min\{m, n\}$, and σ_i are called the singular values of A. The singular values are the square roots of the nonzero eigenvalues of both $AA^\intercal$ and $A^\intercal A$, and they satisfy the property $\sigma_1 \geq \sigma_2 \geq \cdots \geq \sigma_p$.[11, Section 1] [1]

A visual representation of how SVD is defined can be depicted in Figure 2.1.

[1] The proof of this theorem can be found at [9, Chapter 6.2]

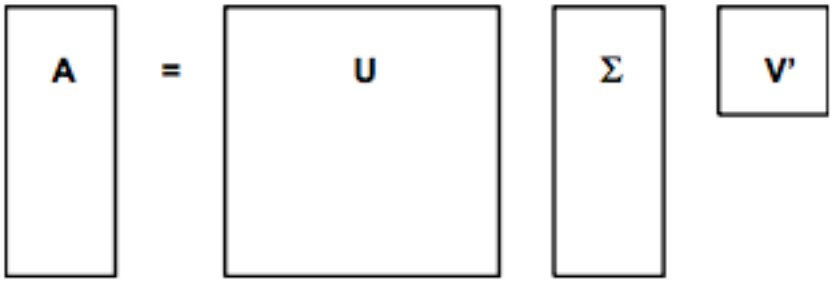

Figure 2.1: Illustration of the matrices involved in SVD

Moreover, if we define $r = rank(A)$ with $\sigma_1 \geq \sigma_2 \geq \cdots \sigma_r > \sigma_{r+1} = \cdots = 0$, then $null(A) = NS(A) = span(v_{r+1}, \ldots, v_p)$ and $range(A) = span(u_1, \ldots, u_r)$. If we let $U_r \in \mathbb{R}^{m \times r}$ denote the matrix consisting of the first r columns of U, $V_r \in \mathbb{R}^{r \times n}$ denote the matrix consisting of the first r columns of V, and $\Sigma_r \in \mathbb{R}^{r \times r}$ denote the principal $r \times r$ sub-matrix of Σ, then

$$\begin{aligned} A &= U_r \Sigma_r V_r^\intercal \\ &= \sum_{i=1}^{r} u_i \sigma_i v_i^\intercal \end{aligned}$$

Applications of SVD. As mentioned earlier, one of the applications of Singular Value Decomposition is computing a low-rank approximation of a matrix.

Theorem 2.5 (Eckart-Young Theorem [11]). *Let the SVD of A be given by 2.1. If $k < r = rank(A)$ and $A_k = \sum_{i=1}^{k} u_i \sigma_i v_i^\intercal$, then*

$$\min_{rank(B)=k} \|A - B\|_2 = \|A - A_k\|_2 = \sigma_{k+1}$$

$$\min_{rank(B)=k} \|A - B\|_F^2 = \|A - A_k\|_F^2 = \sum_{i=k+1}^{r} \sigma_i^2$$

Thus, A_k constructed from the k largest singular triplets of A (u, σ and v) is the optimal $rank - k$ approximation to A with respect to both $\|\cdot\|_F$ and $\|\cdot\|_2$. More generally, $\|A\|_2 = \sigma_1$ and $\|A\|_F^2 = \sum_{i=1}^{r} \sigma_i^2$. [7]

2.2 Big-O analysis of algorithms

In order to be able to mathematically compare two algorithms, and to prove later in this thesis that the presented algorithm is an improvement to the classical SVD method, we first have to define what *Big-O analysis* is.

Big-O notation is a symbolism used in complexity theory, computer science, and mathematics to describe the asymptotic behavior of functions [12]. Thus, the Big-O notation characterizes a function in terms of its growth rate. Here is an example

$$\begin{aligned} f(x) &= 8x^3 + 2x + 1 && \text{is } O(x^3) \\ f(x) &= 17 && \text{is } O(1) \\ f(x) &= 3log(x) + 2 && \text{is } O(log(x)) \end{aligned}$$

These are the basic Big-O classes of functions, ordered from slower growing functions to faster ones ($c \in \mathbb{R}$, constant)

$O(1)$	constant time
$O(log(x))$	logarithmic time
$O(x)$	linear time
$O(xlog(x))$	
$O(x^c)$	polynomial
$O(c^x)$	exponential
$O(x!)$	factorial

Note: An algorithm whose runtime is characterized by a function that has a growth rate greater than $O(x)$ is said to have a **super-linear complexity**.

Consider two functions, $f(x)$ and $g(x)$. Then $f(x)$ is in the class of functions of the order $g(x)$, or $f(x) = O(g(x))$ as $x \to \infty$ if and only if there exist constants N and C such that

$$|f(x)| \leq C|g(x)| \text{ for all } x > N$$

This means that $f(x)$ does not grow faster than $g(x)$.

2.3 Matrix multiplication approximation algorithm

The method presented in this thesis involves a series of matrix multiplications. Similar to computing a low-rank approximation to a matrix, there is also a randomized algorithm for calculating the product of two matrices based on Fast Monte Carlo methods [8]. In this subsection, we will give a short presentation of the ideas behind it. These will be useful in the next chapter, when we will go into the details of the algorithm for computing a low-rank approximation to a matrix.

The `BasicMatrixMultiplication` algorithm, presented in [8], is approximating the product of two matrices, $A \in \mathbb{R}^{m\times n}$ and $B \in \mathbb{R}^{n\times p}$. As input, the algorithm is given the two initial matrices A and B, a probability distribution $\{p_i\}_{i=1}^n$ and a number $c \leq n$. The output consists of two other matrices, $C \in \mathbb{R}^{m\times c}$ and $R \in \mathbb{R}^{c\times p}$, such that $CR \approx AB$ and the columns of C are c randomly chosen columns of A, while the rows of R are c randomly chosen rows of B.

The probability distribution $\{p_i\}_{i=1}^n$ is used for the randomized step of selecting a column/row for approximating AB. Moreover, in order for the algorithm to perform optimally, the probability distribution has to obey the following theorem.

Theorem 2.6. *Suppose* $A \in \mathbb{R}^{m\times n}, B \in \mathbb{R}^{n\times p}, c \in \mathbb{Z}^+$ *such that* $1 \leq c \leq n$, *and* $\{p_i\}_{i=1}^n$ *are such that* $p_i \geq 0$, $\sum_{i=1}^n p_i = 1$ *and such that for some positive* $\beta \leq 1$

$$p_k \geq \frac{\beta |A^{(k)}||B_{(k)}|}{\sum_{k'=1}^n |A^{(k')}||B_{(k')}|} \tag{2.2}$$

Note: A set of sampling probabilities $\{p_i\}_{i=1}^n$ are *nearly optimal probabilities* if they are of the form 2.2; moreover, they are the *optimal probabilities* (with respect to approximating the product AB) if they are of the form 2.2 with $\beta = 1$ [8].

By using the `BasicMatrixMultiplication` algorithm to approximate AB with matrices C and R, we get the following relation ([8])

$$\mathbf{E}[\|AB - CR\|_F^2] \leq \frac{1}{\beta c}\|A\|_F^2\|B\|_F^2 \qquad [2]$$

[2] **E[]** is the expected value of the error $AB - CR$ given the randomized construction of C and R based on the probability distribution $\{p_i\}_{i=1}^n$

Furthermore, for $\delta \in (0,1)$ and $\eta = 1 + \sqrt{(8/\beta)log(1/\delta)}$, with probability at least $1-\delta$ we have

$$\|AB - CR\|_F^2 \leq \frac{\eta^2}{\beta c}\|A\|_F^2\|B\|_F^2$$

This algorithm is of interest for this thesis because of the case when $B = A^\intercal$. Thus, we would like to approximate the product $AA^\intercal$ in an efficient way in order to then compute its eigenvalues and therefore find the singular values of A. Using the sampling method of the `BasicMatrixMultiplication` algorithm, which we will describe in the next chapter, random samples can be drawn according to nearly optimal probabilities (described above) using $O(1)$ additional space[3] and time (per sampling). That means that we would be able to select columns/rows in constant space and time (since the randomly sampled column of A would be the randomly sampled row of $A^\intercal$). Thus, since we would want to extract c columns/rows, the entire sampling process would be $O(c)$. A more detailed analysis will be presented in the following chapter, along with the analysis for the entire method of computing a low-rank approximation to a matrix.

[3] Using a constant amount of computer memory, independent of the sizes of A and B.

Chapter 3

Linear time SVD approximation algorithm

A complexity analysis on the SVD algorithm presented in Section 2.1.3 can prove that the algorithm is super-linear in m and n. That is, the running time of the algorithm increases faster than the size of the problem - in our case, the size of the problem is directly defined by the size of matrix A: m and n.

The goal of this chapter is to present a linear time SVD approximation algorithm. We will analyze an algorithm whose complexity will not be super-linear in either of the sizes of A. We will prove that it is actually linear in terms of both space and time complexity.

3.1 Algorithm description

First, we will give an overview description of the algorithm and of the methods used to approximate the top k singular values and their corresponding singular vectors of matrix $A \in \mathbb{R}^{m \times n}$ in linear time. We will then dive into the implementation details of this method by presenting the outline of the algorithm along with a line by line explanation of what it actually does.

The algorithm receives as input the matrix $A \in \mathbb{R}^{m \times n}$, integers c, k and a probability distribution, and its goal is to produce as output an approximation of the top k singular values and the corresponding left singular vectors of A [7, Chapter 4]. It does that by first sampling c columns of matrix A in order to construct a new matrix, $C \in \mathbb{R}^{m \times c}$. These columns are picked according to a probability

distribution defined through the input of the algorithm. Moreover, before matrix C is constructed from these sampled columns, each column c_i out of the c columns picked from matrix A is being scaled by an appropriate factor, corresponding to the probability of selecting c_i.

Having constructed C in this way, the algorithm computes its right singular vectors, along with the associated singular values, by using the regular *SVD* method for the matrix $C^\intercal C$. It does so because the size of $C^\intercal C \in \mathbb{R}^{c \times c}$ is smaller than the initial matrix sizes, and thus it can run faster - we will provide a more detailed explanation in the next subsection. Then, by the definition of singular vectors, it computes the left singular vectors of C and uses them to construct matrix $H_k \in \mathbb{R}^{m \times k}$ which represents the approximation of the left singular vectors associated to the top k singular values of A.

Algorithm implementation

Here is the implementation outline for the algorithm along with a line-by-line explanation of how it works.

Algorithm 1 `LinearTimeSVD` Algorithm [7]

Input: $A \in \mathbb{R}^{m \times n}$, $c, k \in \mathbb{Z}^+$ such that $1 \leq k \leq c \leq n$, $\{p_i\}_{i=1}^n$ such that $p_i \geq 0$ and $\sum_{i=1}^n p_i = 1$
Output: $H_k \in \mathbb{R}^{m \times k}$ and $\sigma_t(C)$, $t = 1, \ldots, k$.

1: **procedure** LINEARTIMESVD($A, k, c, \{p_i\}_{i=1}^n$)
2: **for** $t = 1$ to c **do**
3: Pick $i_t \in 1, \ldots, n$ with $\mathbf{Pr}[i_t = \alpha] = p_\alpha$, $\alpha = 1, \ldots, n$
4: Set $C^{(t)} = A^{(i_t)} / \sqrt{c p_{i_t}}$
5: **end for**

6: Compute $C^\intercal C$ and its SVD. Thus $C^\intercal C = \sum_{t=1}^c \sigma_t^2(C) y^t y^{t^T}$
7: Compute $h^t = C y^t / \sigma_t(C)$ for $t = 1, \ldots, k$
8: Construct H_k, where $H_k^{(t)} = h^{(t)}$

9: **return** H_k and $\sigma_t(C)$, $t = 1, \ldots, k$
10: **end procedure**

Line 1: The method is receiving the input defined above.

Line 2: The sampling process begins, for c columns.

Line 3: Given the probability distribution $\{p_i\}_{i=1}^n$ from the input, the probability of picking column α on sampling round t (where i_t represents the column picked at sampling round t) is $\mathbf{Pr}[i_t = \alpha] = p_\alpha$. Therefore, at sampling round t, the column i_t will be picked picked by the algorithm with probability

p_α.

Line 4: Normalize the sampled column, $A^{(i_t)}$ and assign it to $C^{(t)}$. Doing this c times will construct a matrix $C \in \mathbb{R}^{m \times c}$. The normalization is being done based on the number of columns we need to sample and the probability of picking the current column.

Line 5: End of sampling process.

Line 6: Here, the algorithm computes the SVD of $C^\intercal C$ in order to calculate the singular values of C and their corresponding right singular vectors. We have to note that, here, even though we apply the SVD (which has a super-linear complexity in the dimensions of the matrix), since our sample size c is constant, this computation *will not be super-linear* (even though we increase the sizes of the input matrix A, the sample size c will not necessarily change, and thus the computation time will stay constant in regards to the varying sizes of the input). Moreover, we will now show we get that form of SVD for $C^\intercal C$.

By the SVD Theorem 2.4, let $C = \sum_{i=1}^{c} \sigma_i(C) u^{(i)} {v^{(i)}}^T$ where $u^{(i)}$ are the left orthonormal singular vectors and $v^{(i)}$ are the right orthonormal singular vectors (note that both singular vectors are column vectors). Moreover, $C^\intercal = \sum_{j=1}^{c} \sigma_j(C) v^{(j)} {y^{(j)}}^T$. Now let's evaluate the SVD for $C^\intercal C$:

$$\begin{aligned} C^\intercal C &= \left(\sum_{j=1}^{c} \sigma_j(C) v^{(j)} {y^{(j)}}^T \right) \left(\sum_{i=1}^{c} \sigma_i(C) u^{(i)} {v^{(i)}}^T \right) \\ &= \sum_{j=1}^{c} \sum_{i=1}^{c} \sigma_j(C) \sigma_i(C) v^{(j)} \left({u^{(j)}}^T u^{(i)} \right) {v^{(i)}}^T \end{aligned}$$

Notice that ${u^{(j)}}^T u^{(i)}$ is a vector dot product that is equal to 0 when $i \neq j$ due to orthogonality. Therefore, when $i \neq j$, all of the summation terms are cancelled, leaving us only with the case when $i = j$, when ${u^{(j)}}^T u^{(i)} = 1$.

$$C^\intercal C = \sum_{i=1}^{c} \sigma_i^2(C) v^{(i)} {v^{(i)}}^T$$

where $v^{(i)}$ are the right singular vectors of C.

Line 7: Having the right singular vectors of C, we can now use the definition of singular values and singular vectors from 2.3 to derive the left singular vectors of C. Moreover, the algorithm is only selecting the left singular vectors corresponding to the top k singular values of C.

Line 8: Now we construct the matrix H_k of the k left singular vectors associated to the top k singular

values of C.

Line 9: The algorithm returns H_k along with the top k singular values of C.

Here is a visualization of the algorithm, presenting the transformations represented by the matrices A and C along with their SVD's.

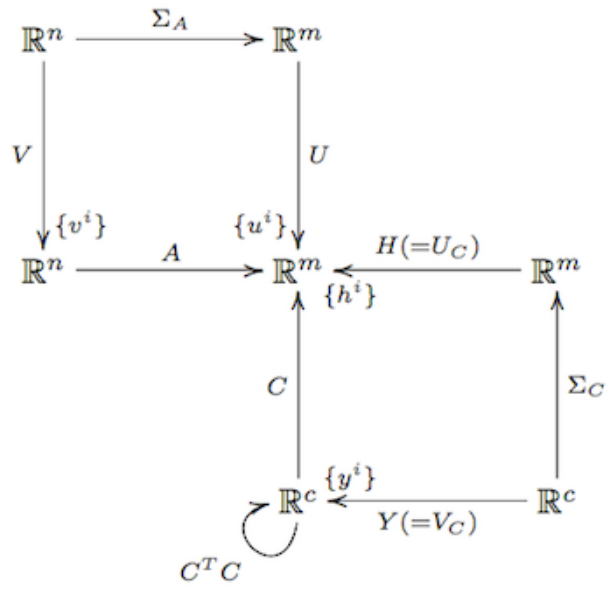

Figure 3.1: Diagram of the `LinearTimeSVD` algorithm [7]

3.2 Complexity analysis

As we mentioned at the beginning of this chapter, the goal of the algorithm is to avoid the super-linear complexity in terms of the sizes of matrix A and to produce a low-rank approximation of the input matrix in linear time instead. Here we will analyze the complexity of the method described in the previous section in order to see how it achieves the goal.

Sampling Process. The sampling process takes place on *line 3* of algorithm 1. By using nearly optimal sampling probabilities (2.3) along with the `Select` algorithm presented in [8, Chapter 3.2], we can perform each sampling step in $O(1)$ space complexity (constant with respect to the size of the sampling space). Moreover, since we have to do this over c steps, the total space and time complexities will be: **time complexity** $O(c)$, **space complexity** $O(c)$.

Constructing C. Since we have that $C \in \mathbb{R}^{m \times c}$, and we construct C by adding together all c sampled columns, where each of the columns is made out of m elements, we get: **time complexity** $O(mc)$, **space complexity** $O(mc)$.

Constructing $C^\intercal C$. In order to create the $C^\intercal C$ matrix, we need to create $C^\intercal$. This will take $O(mc)$ additional space, since $C^\intercal \in \mathbb{R}^{c \times m}$. Moreover, the matrix multiplication algorithm is going to be of the following form (the naive implementation):

Input: $A \in \mathbb{R}^{c \times m}$, $B \in \mathbb{R}^{m \times c}$ and the initially empty result matrix, $R \in \mathbb{R}^{c \times c}$.
Output: R, where $R = A \times B$.

```
procedure NaiveMatrixMultiplication(A, B, R)
    for i = 1 to c do                          ▷ For each row of A
        for j = 1 to c do                      ▷ For each column of B
            for k = 1 to m do                  ▷ For each element
                R_ij = R_ij + A_ik * B_kj
            end for
        end for
    end for

    return R                                   ▷ The result of the matrix multiplication
end procedure
```

In our particular case, the algorithm takes each row of $C^\intercal$ and performs a vector dot product of it with each column of C. Therefore, it will perform in $O(mc^2)$ since there are c rows of $C^\intercal$, c columns of C and each of these contains m elements. Thus: **time complexity** $O(mc^2)$, **space complexity** $O(mc)$.

Computing the SVD of $C^\intercal C$. In order to compute the SVD of the newly constructed $C^\intercal C \in \mathbb{R}^{c \times c}$, we will need to store another set of matrices U and V and also perform matrix multiplications. Therefore, according to [13, Chapter 9.4.4], **time complexity** $O(c^3)$, **space complexity** $O(c^2)$ (since for us, the dimensions m and n are both equal to c).

Constructing H_k. We can see that, in order to construct H_k, we first need to calculate the k left singular vectors, on line 7. Since $C \in \mathbb{R}^{m \times c}$ and $y^{(t)} \in \mathbb{R}^{c \times 1}$ (column vector), we need to perform the matrix-vector multiplication k times. One such operation will take $O(mc)$ time, and thus, constructing H_k, we will have: **time complexity** $O(mck)$, **space complexity** $O(mck)$.

Putting everything together, we get the overall analysis: **overall time complexity** $O(mc^2+c^3)$, **overall space complexity** $O(cm+c^2)$. We can now see that the presented algorithm does not perform in super-linear complexity in regards to either m or n. Even though we get polynomial times in terms of c, we have to note that c does not increase as the sizes of the input matrix A increase. The term c represents the sampling count and is defined by user at the beginning of the algorithm (i.e. it can stay constant for matrices A of different sizes, and thus the overall complexity of the presented algorithm will also be bounded just by c and thus not super-linear in regards to the sizes of A).

3.3 Algorithm proof and analysis

Now that we proved the complexity improvement of the algorithm, we will proceed by proving its error bound. First, we will prove that the low-rank approximation of A produced by the presented algorithm has an error that depends on $\|AA^\intercal - CC^\intercal\|_F$ with respect to both $\|\cdot\|_F$ and $\|\cdot\|_2$ [13, Chapter 4.3]. In the end of this section we will show that the error of the low-rank approximation of A can be made small by taking advantage of the nearly optimal properties of the probability distribution provided as input to the algorithm - that is, by sampling enough columns.

As previously discussed, the regular SVD method provides the optimal low-rank (rank-k) approximation with respect to both $\|\cdot\|_F$ and $\|\cdot\|_2$. This approximation is $A_k = U_k U_k^\intercal A$ (2.5). The sampling-based method presented in this thesis creates an approximating matrix, $D_k = H_k H_k^\intercal A$, from the approximations of the left singular vectors of A. The error of D_k is dependent on the error of the best approximation, A_k, along with an additional error term of the form $\|AA^\intercal - CC^\intercal\|_F$.

Theorem 3.1 (Frobenius error of D_k [7])**.** *Suppose $A \in \mathbb{R}^{m\times n}$ and let H_k be constructed from the `LinearTimeSVD` algorithm. Then*

$$\|A - H_k H_k^\intercal A\|_F^2 \leq \|A - A_k\|_F^2 + 2\sqrt{k}\|AA^\intercal - CC^\intercal\|_F \tag{3.1}$$

Proof. Recall from Chapter 2.1.1 that for two square matrices X and Y we have the following: $\|X\|_F^2 = \mathbf{Tr}(XX^\intercal) = \mathbf{Tr}(X^\intercal X)$, $\mathbf{Tr}(X+Y) = \mathbf{Tr}(X) + \mathbf{Tr}(Y)$ and $H_k^\intercal H_k = I_k$ since H_k is an orthogonal matrix (its columns form an orthonormal basis). Then, we expand the Frobenius error of D_k by these

identities and we get:

$$\begin{aligned}\|A - D_k\|_F^2 &= \|A - H_k H_k^\intercal A\|_F^2 \\ &= \mathbf{Tr}\big((A - H_k H_k^\intercal A)^\intercal (A - H_k H_k^\intercal A)\big) \\ &= \mathbf{Tr}\big(A^\intercal A - A^\intercal H_k H_k^\intercal A - A^\intercal H_k H_k^\intercal A + A^\intercal H_k H_k^\intercal H_k H_k^\intercal A\big)\end{aligned}$$

by expanding the product and using transpose distributivity

$$\|A - D_k\|_F^2 = \mathbf{Tr}\big(A^\intercal A - A^\intercal H_k H_k^\intercal A\big)$$

by orthogonality of H_k

$$\begin{aligned}\|A - D_k\|_F^2 &= \mathbf{Tr}\big(A^\intercal A\big) - \mathbf{Tr}\big((A^\intercal H_k)(H_k^\intercal A)\big) \\ &= \|A\|_F^2 - \|A^\intercal H_k\|_F^2\end{aligned}$$

Now, we will use the relationship between the Frobenius norm and the singular values of a matrix, mentioned at the end of Theorem 2.5

$$\begin{aligned}\|A - D_k\|_F^2 &= \sum_{t=1}^{r=rank(A)} \sigma_t^2(A) - \|A^\intercal H_k\|_F^2 \\ &= \sum_{t=1}^{k} \sigma_t^2(A) + \sum_{t=k+1}^{r} \sigma_t^2(A) - \|A^\intercal H_k\|_F^2\end{aligned}$$

which, by applying the Eckart-Young Theorem 2.5

$$\|A - D_k\|_F^2 = \|A - A_k\|_F^2 + \Big(\sum_{t=1}^{k} \sigma_t^2(A) - \|A^\intercal H_k\|_F^2\Big)$$

Therefore, we now have:

$$\|A - D_k\|_F^2 = \|A - A_k\|_F^2 + \Big(\sum_{t=1}^{k} \sigma_t^2(A) - \|A^\intercal H_k\|_F^2\Big) \tag{3.2}$$

We are now looking for a relation between $\|A^\intercal H_k\|_F^2$ and $\sum_{t=1}^{k} \sigma_t^2(A)$. In order to find it, we will first find the relation between $\|A^\intercal H_k\|_F^2$ and $\sum_{t=1}^{k} \sigma_t^2(C)$, and then express the relation between $\sum_{t=1}^{k} \sigma_t^2(C)$ and $\sum_{t=1}^{k} \sigma_t^2(A)$. Putting everything together, we will get the desired inequality. Let's proceed.

For the following step, we will need the *Cauchy-Schwartz Inequality*, which we define by:

$$\Big|\sum_{i=1}^{n} a_i b_i\Big| \leq \Big(\sum_{i=1}^{n} (a_i)^2 \sum_{i=1}^{n} (b_i)^2\Big)^{1/2} \tag{3.3}$$

for $a_i, b_i \in \mathbb{R}^n$.

Moreover, by the definition of the Frobenius norm 2.1, it can be seen that we have the following identity:

$$\|A^\intercal H_k\|_F^2 = \sum_{t=1}^{k} \left|A^\intercal h^t\right|^2$$

where h^k are the left singular vectors of C, as computed on line 7 of algorithm 1.

Thus, using the above identity, we can now consider the following relation:

$$\begin{aligned}
\left|\|A^\intercal H_k\|_F^2 - \sum_{t=1}^{k} \sigma_t^2(C)\right| &= \left|\sum_{t=1}^{k} \left|A^\intercal h^t\right|^2 - \sum_{t=1}^{k} \sigma_t^2(C)\right| \\
&= \left|\sum_{t=1}^{k} \left(\left|A^\intercal h^t\right|^2 - \sigma_t^2(C)\right)\right| \\
&= \left|\sum_{t=1}^{k} \left[(1) \cdot \left(\left|A^\intercal h^t\right|^2 - \sigma_t^2(C)\right)\right]\right|
\end{aligned}$$

By applying the Cauchy-Schwartz inequality 3.3 with $a_i = 1$ and $b_i = \left|A^\intercal h^i\right|^2 - \sigma_i^2(C)$, we get

$$\leq \sqrt{k}\left(\sum_{t=1}^{k} \left(\left|A^\intercal h^t\right|^2 - \sigma_t^2(C)\right)^2\right)^{1/2} \tag{3.4}$$

Now we would like to express $\sigma_t^2(C)$ in terms of $C^\intercal$ and h^t. We start by noting that, by definition, the right singular vectors of C are eigenvectors of $C^\intercal C$. Thus:

$$(C^\intercal C)y^t = \lambda_t(C^\intercal C)y^t$$

where $\lambda_t(X)$ denotes the t^{th} eigenvalue of a matrix X and y^t is both the right singular vector of C, as computed on line 6 of algorithm 1, and the eigenvector of $C^\intercal C$. Therefore:

$$\begin{aligned}
|C^\intercal C y^t|^2 &= |\lambda_t(C^\intercal C)y^t|^2 \\
&= \lambda_t^2(C^\intercal C)
\end{aligned}$$

by the definition of the L^2 norm 2.1, with y^t being a unit vector. Moreover, by properties of singular values, recall that $\sigma_t^2(C) = \lambda_t(C^\intercal C)$. Thus, we get:

$$\begin{aligned}
|C^\intercal C y^t|^2 &= \lambda_t^2(C^\intercal C) \\
&= \sigma_t^4(C)
\end{aligned} \tag{3.5}$$

Moreover, $Cy^t = \lambda_t(C)h^t$ since y^t and h^t are the right and, respectively, left singular vectors of C. Therefore:

$$\begin{aligned} |C^\intercal C y^t|^2 &= |C^\intercal \lambda_t(C) h^t|^2 \\ &= \sigma_t^2(C)|C^\intercal h^t|^2 \end{aligned} \tag{3.6}$$

Now, putting together the results in 3.5 and 3.6, we get the following identity:

$$|C^\intercal h^t|^2 = \sigma_t^2(C) \tag{3.7}$$

With this, we can now continue our results from 3.4:

$$\begin{aligned} \Big| \|A^\intercal H_k\|_F^2 - \sum_{t=1}^{k} \sigma_t^2(C) \Big| &\leq \sqrt{k} \Big(\sum_{t=1}^{k} \big(\big|A^\intercal h^t\big|^2 - \sigma_t^2(C) \big)^2 \Big)^{1/2} \\ &= \sqrt{k} \Big(\sum_{t=1}^{k} \big(\big|A^\intercal h^t\big|^2 - \big|C^\intercal h^t\big|^2 \big)^2 \Big)^{1/2} \end{aligned} \tag{3.8}$$

For a column vector $b \in \mathbb{R}^{n \times 1}$ it is easy to check that $b^\intercal b = |b|^2$. Let $b_1 = A^\intercal h^t$ and $b_2 = C^\intercal h^t$. With this observation, expand the L^2 norms from 3.8:

$$\begin{aligned} \Big| \|A^\intercal H_k\|_F^2 - \sum_{t=1}^{k} \sigma_t^2(C) \Big| &\leq \sqrt{k} \Big(\sum_{t=1}^{k} \big(\big|A^\intercal h^t\big|^2 - \sigma_t^2(C) \big)^2 \Big)^{1/2} \\ &= \sqrt{k} \Big(\sum_{t=1}^{k} \big(\big|A^\intercal h^t\big|^2 - \big|C^\intercal h^t\big|^2 \big)^2 \Big)^{1/2} \\ &= \sqrt{k} \Big(\sum_{t=1}^{k} \big(h^{tT} A A^\intercal h^t - h^{tT} C C^\intercal h^t \big)^2 \Big)^{1/2} \\ &= \sqrt{k} \Big(\sum_{t=1}^{k} \big(h^{tT} (A A^\intercal - C C^\intercal) h^t \big)^2 \Big)^{1/2} \\ &\leq \sqrt{k} \|A A^\intercal - C C^\intercal\|_F \end{aligned}$$

since $\{h^t\}_{t=1}^k$ is part of a unitary basis.

Therefore, we got the relation:

$$\Big|\|A^\intercal H_k\|_F^2 - \sum_{t=1}^{k}\sigma_t^2(C)\Big| \leq \sqrt{k}\|AA^\intercal - CC^\intercal\|_F \tag{3.9}$$

Now we want to obtain the relation between $\sum_{t=1}^{k}\sigma_t^2(C)$ and $\sum_{t=1}^{k}\sigma_t^2(A)$. First, note that, for a matrix X, $\sigma_t^2(X) = \sigma_t(XX^\intercal)$ as previously shown in section 3.1. Therefore, we proceed similarly, as in the previously established relation:

$$\Big|\sum_{t=1}^{k}\sigma_t^2(C) - \sum_{t=1}^{k}\sigma_t^2(A)\Big| \leq \sqrt{k}\Big(\sum_{t=1}^{k}\big(\sigma_t^2(C) - \sigma_t^2(A)\big)^2\Big)^{1/2}$$

by the Cauchy-Schwartz inequality 3.3 with $a_i = 1$ and $b_i = \sigma_i^2(C) - \sigma_i^2(A)$. Moreover, applying the singular value property that we mentioned, we get:

$$\begin{aligned}\Big|\sum_{t=1}^{k}\sigma_t^2(C) - \sum_{t=1}^{k}\sigma_t^2(A)\Big| &= \sqrt{k}\Big(\sum_{t=1}^{k}\big(\sigma_t(CC^\intercal) - \sigma_t(AA^\intercal)\big)^2\Big)^{1/2} \\ &\leq \sqrt{k}\Big(\sum_{t=1}^{m}\big(\sigma_t(CC^\intercal) - \sigma_t(AA^\intercal)\big)^2\Big)^{1/2}\end{aligned} \tag{3.10}$$

To advance to the next step, we need to introduce the *Hoffman-Wielandt Inequality*, which is defined in the following way: if $X, Y \in \mathbb{R}^{m\times n}$, $m \geq n$, then

$$\sum_{k=1}^{n}\big(\sigma_k(X+Y) - \sigma_k(X)\big)^2 \leq \|Y\|_F^2 \tag{3.11}$$

Let's prepare our previous inequality 3.10 in order to be able to clearly use the Hoffman-Wielandt Inequality 3.11.

$$\begin{aligned}\Big|\sum_{t=1}^{k}\sigma_t^2(C) - \sum_{t=1}^{k}\sigma_t^2(A)\Big| &\leq \sqrt{k}\Big(\sum_{t=1}^{m}\big(\sigma_t(CC^\intercal) - \sigma_t(AA^\intercal)\big)^2\Big)^{1/2} \\ &= \sqrt{k}\Big(\sum_{t=1}^{m}\Big(\sigma_t\big(AA^\intercal + (CC^\intercal - AA^\intercal)\big) - \sigma_t(AA^\intercal)\Big)^2\Big)^{1/2}\end{aligned}$$

Now, with $X = AA^\intercal$ and $Y = CC^\intercal - AA^\intercal$, we can apply the previously defined inequality 3.11, and get our desired relation:

$$\Big|\sum_{t=1}^{k}\sigma_t^2(C) - \sum_{t=1}^{k}\sigma_t^2(A)\Big| \leq \sqrt{k}\|CC^\intercal - AA^\intercal\|_F \tag{3.12}$$

Having the two inequalities 3.9 and 3.12, we want to obtain the relation between $\|A^\intercal H_k\|_F^2$ and $\sum_{t=1}^k \sigma_t^2(A)$.

$$\left|\|A^\intercal H_k\|_F^2 - \sum_{t=1}^k \sigma_t^2(C)\right| \leq \sqrt{k}\|AA^\intercal - CC^\intercal\|_F$$
$$\left|\sum_{t=1}^k \sigma_t^2(C) - \sum_{t=1}^k \sigma_t^2(A)\right| \leq \sqrt{k}\|CC^\intercal - AA^\intercal\|_F$$

Thus, in a similar fashion as we obtained the previous two relations,

$$\begin{aligned}\left|\|A^\intercal H_k\|_F^2 - \sum\nolimits_{t=1}^k \sigma_t^2(A)\right| &= \left|[\|A^\intercal H_k\|_F^2 - \sum\nolimits_{t=1}^k \sigma_t^2(C)] + [\sum\nolimits_{t=1}^k \sigma_t^2(C) - \sum\nolimits_{t=1}^k \sigma_t^2(A)]\right| \\ &\leq \left|\|A^\intercal H_k\|_F^2 - \sum\nolimits_{t=1}^k \sigma_t^2(C)\right| + \left|\sum\nolimits_{t=1}^k \sigma_t^2(C) - \sum\nolimits_{t=1}^k \sigma_t^2(A)\right|\end{aligned}$$

by the triangle inequality. Moreover, using 3.9 and 3.12, we now get the desired relation:

$$\left|\|A^\intercal H_k\|_F^2 - \sum\nolimits_{t=1}^k \sigma_t^2(A)\right| \leq 2\sqrt{k}\|CC^\intercal - AA^\intercal\|_F \tag{3.13}$$

Finally, by putting together 3.2 and 3.13,

$$\|A - D_k\|_F^2 = \|A - A_k\|_F^2 + \left(\sum_{t=1}^k \sigma_t^2(A) - \|A^\intercal H_k\|_F^2\right)$$
$$\left|\|A^\intercal H_k\|_F^2 - \sum\nolimits_{t=1}^k \sigma_t^2(A)\right| \leq 2\sqrt{k}\|CC^\intercal - AA^\intercal\|_F$$

where $D_k = H_k H_k^\intercal A$, we get the result of the theorem:

$$\|A - H_k H_k^\intercal A\|_F^2 \leq \|A - A_k\|_F^2 + 2\sqrt{k}\|AA^\intercal - CC^\intercal\|_F$$

□

We proved the error bound of the approximating low-rank (k-rank) matrix $D_k = H_k H_k^\intercal A$ on $\|\cdot\|_F$. Now we need to give a similar result for $\|\cdot\|_2$.

Theorem 3.2 (Spectral error of D_k [7]). *Suppose $A \in \mathbb{R}^{m\times n}$ and let H_k be constructed from the `LinearTimeSVD` algorithm. Then*

$$\|A - H_k H_k^\intercal A\|_2^2 \leq \|A - A_k\|_2^2 + 2\|AA^\intercal - CC^\intercal\|_2 \tag{3.14}$$

Proof. Let $\mathcal{H}_k = \mathbf{range}(H_k) = \mathbf{span}(h^1, \ldots, h^k)$ and consider $\mathcal{H}_{m-k}$ to be the orthogonal complement of $\mathcal{H}_k$. That is, all the vectors of $\mathcal{H}_{m-k}$ are orthogonal to every vector in $\mathcal{H}_k$.

Moreover, let $x \in \mathbb{R}^{m\times 1}$ be a column vector such that $x = \alpha y + \beta z$, where $y \in \mathcal{H}_k$, $z \in \mathcal{H}_{m-k}$ and $\alpha^2 + \beta^2 = 1$.

We want to find the bound of the error $\|A - H_k H_k^\intercal A\|_2$. Therefore, using the definition of the Spectral norm 2.1 and substituting for x, we get:

$$\begin{aligned}
\|A - H_k H_k^\intercal A\|_2 &= \max_{x\in\mathbb{R}^{m\times 1}, |x|=1} \left|x^\intercal (A - H_k H_k^\intercal A)\right| \\
&= \max_{y\in\mathcal{H}_k, |y|=1, z\in\mathcal{H}_{m-k}, |z|=1, \alpha^2+\beta^2=1} \left|(\alpha y^\intercal + \beta z^\intercal)(A - H_k H_k^\intercal A)\right| \\
&= \max_{y\in\mathcal{H}_k, |y|=1, z\in\mathcal{H}_{m-k}, |z|=1, \alpha^2+\beta^2=1} \left|\alpha y^\intercal (A - H_k H_k^\intercal A) + \beta z^\intercal (A - H_k H_k^\intercal A)\right| \\
&\leq \max_{y\in\mathcal{H}_k, |y|=1} \left|y^\intercal (A - H_k H_k^\intercal A)\right| + \max_{z\in\mathcal{H}_{m-k}, |z|=1} \left|z^\intercal (A - H_k H_k^\intercal A)\right| \qquad (3.15) \\
&\quad \text{by the triangle inequality and by the fact that } \alpha, \beta \leq 1.
\end{aligned}$$

By direct computation it can be shown that $y^\intercal H_k H_k^\intercal A = y^\intercal A$ since the right singular vectors (y) are orthonormal vectors. Therefore, we have:

$$\max_{y\in\mathcal{H}_k, |y|=1} \left|y^\intercal (A - H_k H_k^\intercal A)\right| = 0 \tag{3.16}$$

Moreover, since $z \in \mathcal{H}_{m-k}$, where $\mathcal{H}_{m-k}$ is the orthogonal complement of $\mathcal{H}_k$, we get that $z^\intercal H_k H_k^\intercal A = 0 \in \mathbb{R}^{1\times n}$. Using this fact and 3.16 in 3.15, we get:

$$\begin{aligned}
\|A - H_k H_k^\intercal A\|_2 &\leq \max_{y\in\mathcal{H}_k, |y|=1} \left|y^\intercal (A - H_k H_k^\intercal A)\right| + \max_{z\in\mathcal{H}_{m-k}, |z|=1} \left|z^\intercal (A - H_k H_k^\intercal A)\right| \\
&\leq \max_{z\in\mathcal{H}_{m-k}, |z|=1} \left|z^\intercal A\right|
\end{aligned}$$

Thus, by squaring this result, we got:

$$\|A - H_k H_k^\intercal A\|_2^2 \leq \max_{z \in \mathcal{H}_{m-k}, |z|=1} \left|z^\intercal A\right|^2 \tag{3.17}$$

Now let's look at the bound of $\max_{z \in \mathcal{H}_{m-k}, |z|=1} \left|z^\intercal A\right|$. Again, we will use the fact that for a column vector $b \in \mathbb{R}^{n \times 1}$, $b^\intercal b = |b|^2 = |b^\intercal|^2$. Thus, let $b = A^\intercal z \in \mathbb{R}^{n \times 1}$. We get:

$$\begin{aligned}
\left|z^\intercal A\right|^2 &= \left(z^\intercal A\right)\left(A^\intercal z\right) \\
&= z^\intercal C C^\intercal z + \left(z^\intercal A A^\intercal z - z^\intercal C C^\intercal z\right) \\
&= z^\intercal C C^\intercal z + z^\intercal \left(A A^\intercal - C C^\intercal\right) z \\
&= \left|z^\intercal C\right|^2 + z^\intercal \left(A A^\intercal - C C^\intercal\right) z
\end{aligned} \tag{3.18}$$

Notice that the maximum possible value in $\mathcal{H}_{m-k}$ is the $(k+1)$st left singular vector, $z = h^{k+1}$. Therefore, since $C \in \mathbb{R}^{m \times c}$, this will also be the value that will maximize $\left|z^\intercal C\right|$: $\max_{z \in \mathcal{H}_{m-k}, |z|=1} \left|z^\intercal C\right| = \sigma_{k+1}^2(C)$. Therefore:

$$\begin{aligned}
\left|z^\intercal A\right|^2 &= \left|z^\intercal C\right|^2 + z^\intercal \left(A A^\intercal - C C^\intercal\right) z \\
&\leq \sigma_{k+1}^2(C) + \|A A^\intercal - C C^\intercal\|_2
\end{aligned} \tag{3.19}$$

For the following step, we need again the inequality: $\sigma_{k+1}^2(X) = \sigma_{k+1}(X X^\intercal)$ for some matrix X. Moreover, we need to introduce another inequality. If $X, Y \in \mathbb{R}^{m \times n}$, $m \geq n$, then:

$$\max_{t: 1 \leq t \leq n} \left|\sigma_t(X + Y) - \sigma_t(X)\right| \leq \|Y\|_2 \tag{3.20}$$

In our case, let $X = A A^\intercal$ and $Y = C C^\intercal - A A^\intercal$. Thus, using 3.20, we get:

$$\max_{t: 1 \leq t \leq n} \left|\sigma_t\left(A A^\intercal + (C C^\intercal - A A^\intercal)\right) - \sigma_t(A A^\intercal)\right| \leq \left\|C C^\intercal - A A^\intercal\right\|_2 \tag{3.21}$$

This implies that:

$$\begin{aligned}
\sigma_{k+1}(C C^\intercal) &\leq \|C C^\intercal - A A^\intercal\|_2 + \sigma_{k+1}(A A^\intercal) \\
\sigma_{k+1}^2(C) &\leq \sigma_{k+1}^2(A) + \|C C^\intercal - A A^\intercal\|_2
\end{aligned} \tag{3.22}$$

Therefore, using 3.22 in 3.19 we get:

$$\begin{aligned}\left|z^{\intercal}A\right|^2 &\leq \sigma_{k+1}^2(C) + \|AA^{\intercal} - CC^{\intercal}\|_2 \\ &\leq \sigma_{k+1}^2(A) + 2\|AA^{\intercal} - CC^{\intercal}\|_2 \\ &= \|A - A_k\|_2^2 + 2\|AA^{\intercal} - CC^{\intercal}\|_2 \\ &\text{by the Eckart-Young Theorem 2.5}\end{aligned} \tag{3.23}$$

Finally, from 3.17 and 3.23,

$$\begin{aligned}\|A - H_k H_k^{\intercal} A\|_2^2 &\leq \max_{z \in \mathcal{H}_{m-k}, |z|=1} \left|z^{\intercal}A\right|^2 \\ \left|z^{\intercal}A\right|^2 &\leq \|A - A_k\|_2^2 + 2\|AA^{\intercal} - CC^{\intercal}\|_2\end{aligned} \tag{3.24}$$

we get the desired theorem result:

$$\|A - H_k H_k^{\intercal} A\|_2^2 \leq \|A - A_k\|_2^2 + 2\|AA^{\intercal} - CC^{\intercal}\|_2$$

□

Having the error bounds for both $\|\cdot\|_F$ and $\|\cdot\|_2$ from 3.1 and 3.2, we can go ahead and look at what information they tell us. First, note that the probability distribution $\{p_i\}_{i=1}^n$ is not taken into account for either of those errors. More than that, the term $\|A - A_k\|_{F,2}$ is a property of the matrix A and thus it is not dependent on how we sample the c columns. If we would take into account the probabilities, the only term that would be affected in computing the error $\|A - H_k H_k^{\intercal} A\|_{F,2}$ would be $\|AA^{\intercal} - CC^{\intercal}\|_{F,2}$.

Therefore, in the following theorem we will describe what happens when the probability distribution is taken into account. We will use nearly optimal probabilities, as defined in 2.3. By choosing enough columns, the error in the approximation of the SVD can be made arbitrarily small.

Theorem 3.3 (Sampling advantage of the `LinearTimeSVD` [7]). *Suppose $A \in \mathbb{R}^{m \times n}$; let H_k be constructed from the* `LinearTimeSVD` algorithm by sampling c columns of A with probabilities $\{p_i\}_{i=1}^n$ such that $p_i \geq \beta\left|A^{(i)}\right|^2 / \|A\|_F^2$ for some $0 < \beta \leq 1$, and let $\eta = 1 + \sqrt{(8/\beta) log(1/\delta)}$. Let $\epsilon > 0$.

If $c \geq 4k/\beta\epsilon^2$, then

$$\mathbf{E}\left[\|A - H_k H_k^{\intercal} A\|_F^2\right] \leq \|A - A_k\|_F^2 + \epsilon\|A\|_F^2, \tag{3.25}$$

and if $c \geq 4k\eta^2/\beta\epsilon^2$, then with probability at least $1-\delta$,

$$\|A - H_k H_k^\intercal A\|_F^2 \leq \|A - A_k\|_F^2 + \epsilon\|A\|_F^2, \tag{3.26}$$

In addition, if $c \geq 4/\beta\epsilon^2$, then

$$\mathbf{E}\big[\|A - H_k H_k^\intercal A\|_2^2\big] \leq \|A - A_k\|_2^2 + \epsilon\|A\|_F^2, \tag{3.27}$$

and if $c \geq 4\eta^2/\beta\epsilon^2$, then with probability at least $1-\delta$,

$$\|A - H_k H_k^\intercal A\|_2^2 \leq \|A - A_k\|_2^2 + \epsilon\|A\|_F^2, \tag{3.28}$$

Proof. In this proof we will make use of the previously two theorems we proved already, 3.1 and 3.2, along with theorem 2.6, presented in the previous chapter. We will prove the inequalities associated with the Frobenius norm, and then the Spectral norm ones will come naturally.

First, let's look at 3.25. From 3.1, we have that:

$$\begin{aligned} \|A - H_k H_k^\intercal A\|_F^2 &\leq \|A - A_k\|_F^2 + 2\sqrt{k}\|AA^\intercal - CC^\intercal\|_F \\ \mathbf{E}\big[\|A - H_k H_k^\intercal A\|_F^2\big] &\leq \mathbf{E}\big[\|A - A_k\|_F^2 + 2\sqrt{k}\|AA^\intercal - CC^\intercal\|_F\big] \\ &= \|A - A_k\|_F^2 + 2\sqrt{k}\mathbf{E}\big[\|AA^\intercal - CC^\intercal\|_F\big] \end{aligned} \tag{3.29}$$

by linearity of the expected value $\mathbf{E}\big[\big]$ and since, as previously mentioned, $\|A - A_k\|_F^2$ does not depend on $\{p_i\}_{i=1}^n$.

Now we need to express $\mathbf{E}\big[\|AA^\intercal - CC^\intercal\|_F\big]$. By theorem 2.6,

$$\mathbf{E}\big[\|AA^\intercal - CC^\intercal\|_F^2\big] \leq 1/\beta c \; \|A\|_F^2$$

Now we will use the following identity regarding the expected value and variance. For a given random variable X we have: $\mathbf{Var}[X] = \mathbf{E}[X^2] - \big(\mathbf{E}[X]\big)^2$. In our case, $X = \|AA^\intercal - CC^\intercal\|_F^2$ and $\mathbf{Var}[\|AA^\intercal - CC^\intercal\|_F^2] = 0$. Therefore, solving for $\mathbf{E}\big[\|AA^\intercal - CC^\intercal\|_F\big]$,

$$\mathbf{E}\big[\|AA^\intercal - CC^\intercal\|_F\big] \leq \big(1/\beta c\big)^{1/2} \; \|A\|_F \tag{3.30}$$

From 3.29 and 3.30, we now get:

$$\mathbf{E}\left[\|A - H_k H_k^\intercal A\|_F^2\right] \leq \|A - A_k\|_F^2 + \left(4k/\beta c\right)^{1/2} \|A\|_F$$

which, when using the appropriate value of c, $c \geq 4k/\beta\epsilon^2$, becomes our desired result:

$$\mathbf{E}\left[\|A - H_k H_k^\intercal A\|_F^2\right] \leq \|A - A_k\|_F^2 + \epsilon\|A\|_F^2$$

Now let's focus on 3.26. Again, from 3.1, we have:

$$\|A - H_k H_k^\intercal A\|_F^2 \leq \|A - A_k\|_F^2 + 2\sqrt{k}\|AA^\intercal - CC^\intercal\|_F \tag{3.31}$$

By 2.6, with probability at least $1 - \delta$,

$$\|AA^\intercal - CC^\intercal\|_F^2 \leq \eta^2/\beta c \ \|A\|_F^2$$

which, by using the appropriate value of c, $c \geq 4k\eta^2/\beta\epsilon^2$, becomes:

$$\|AA^\intercal - CC^\intercal\|_F^2 \leq \epsilon^2/4k \ \|A\|_F^2$$

and thus,

$$\|AA^\intercal - CC^\intercal\|_F \leq \epsilon/2\sqrt{k} \ \|A\|_F^2 \tag{3.32}$$

From 3.31 and 3.32, we get our desired result:

$$\|A - H_k H_k^\intercal A\|_F^2 \leq \|A - A_k\|_F^2 + \epsilon\|A\|_F^2$$

In order to prove 3.27 and 3.28, we will make a short note on the relationship between $\|\cdot\|_2$ and $\|\cdot\|_F$ that will make the proofs follow, naturally, from the two previous ones regarding the Frobenius norm.

First, by theorem 3.2:

$$\|A - H_k H_k^\intercal A\|_2^2 \leq \|A - A_k\|_2^2 + 2\|AA^\intercal - CC^\intercal\|_2$$

By identity 2.1 presented in Section 2.1.1, we know that:

$$\|AA^\intercal - CC^\intercal\|_2 \leq \|AA^\intercal - CC^\intercal\|_F$$

and thus we get:

$$\|A - H_k H_k^\intercal A\|_2^2 \leq \|A - A_k\|_2^2 + 2\|AA^\intercal - CC^\intercal\|_F \tag{3.33}$$

Therefore, 3.27 and 3.28 will follow naturally, through proofs almost identical to the ones for the Frobenius norm inequalities, using the appropriate values of c defined in the theorem:

$$\mathbf{E}\big[\|A - H_k H_k^\intercal A\|_2^2\big] \leq \|A - A_k\|_2^2 + \epsilon\|A\|_F^2$$

for $c \geq 4/\beta\epsilon^2$. Moreover, if $c \geq 4\eta^2/\beta\epsilon^2$, with probability at least $1 - \delta$, we get:

$$\|A - H_k H_k^\intercal A\|_2^2 \leq \|A - A_k\|_2^2 + \epsilon\|A\|_F^2$$

□

Chapter 4

Conclusions

In this thesis we have presented an algorithm that computes a low-rank approximation of a matrix A in a more efficient way than the regular SVD method. The efficiency is regarded to the space and time complexities which are linear in $m + n$ and not super-linear, as for SVD. The main idea behind the `LinearTimeSVD` algorithm is that it uses Monte Carlo methods to sample columns from the input matrix A and then to approximate the left singular vectors of it. These are stored in the output matrix H_k. Using this matrix, it then constructs a low-rank approximation of A, given by $H_k H_k^\intercal A$. The sampling process brings in an additional error, besides the one that comes naturally from the regular SVD method. We proved that the additional error is bounded by $\epsilon \|A\|_F^2$ and that, by sampling enough columns (of the order $\Theta(1/\epsilon^2)$ [7, Chapter 6]) it is possible to decrease this bound. Even though this method is a computational improvement to SVD, the authors of [7] describe an additional method, the `ConstantTimeSVD` algorithm, which provides even better results. This one approximates the top k regular singular values and the corresponding singular vectors in a constant number of passes through the data and additional space and time that are $O(1)$ [7, Chapter 5]. The main idea behind this algorithm is the use of the sampling process step twice, once for A and once for C, thus removing the additional time and space required for computing the singular values and corresponding singular vectors of matrix C in `LinearTimeSVD`.

In the end of this paper, we want to go back to the application that represented the motivation for this study and also provide a more visual application of SVD in order to aid the reader with understanding of what SVD is doing by approximating matrix A with a lower rank-k matrix.

As we mentioned in Chapter 1, SVD and methods for approximating large matrices are used in recommender algorithms [14] [15]. Examples of these are the *movie suggestion* algorithms. This type of algorithms was probably made famous by the *Netflix Prize* competition that awarded in 2009 a prize of $1M to the team that was able to come up with an algorithm that produced a better recommendation than what Netflix was already using. The algorithm that won had an important part based on low-rank factorization. We presented an efficient way of doing this through `LinearTimeSVD`, but there is a small detail that we want to touch on. As noted from Chapter 3, the input matrix A was a full matrix, i.e. had all the entries already known. In the case of recommendation systems, many entries of the input matrix are unknown though. This is due to the fact that, in an example similar to Netflix, users vote only on a small subset of the entire movie database. Therefore, SVD or `LinearTimeSVD` can not be directly applied to such a sparse matrix A. In order to be able to apply these algorithms, some preprocessing of the entries of A has to be done. This involves using functions that will weight the entries of matrix A based on whether or not they are known. A more detailed explanation on how these methods work can be found in [16].

We want to conclude with a short illustration of SVD and `LinearTimeSVD`. The algorithm produces a lower rank-k approximation of an input matrix A. If we consider this matrix to be the grayscale representation of an image, where the entries of A represent the intensity of the pixel at position (i, j) in the image grid, then Figure 4.1 depicts the output of an SVD based algorithm for various values of k.

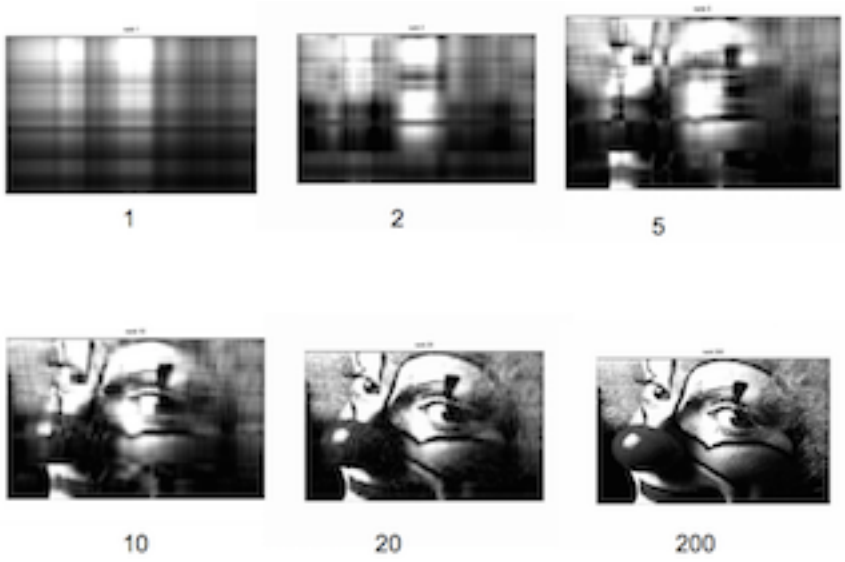

Figure 4.1: Ilustration of the SVD output based on the rank-k approximation [17]

Bibliography

[1] G. Linden, B. Smith, and J. York. Amazon.com recommendations. Item-to-Item collaborative filtering. *IEEE Internet Computing*, 7(1):76–80, January/February 2003. URL http://www.cs.umd.edu/~samir/498/Amazon-Recommendations.pdf.

[2] G. D. Clifford. Singular value decomposition and independent component analysis for blind source separation [MIT Course 6.222j - lectures], April 2005. URL http://www.mit.edu/~gari/teaching/6.222j/ICASVDnotes.pdf. Accessed: 2014-03-11.

[3] G. D. Clifford. Blind source separation [MIT Course 6.555 - lectures]. URL http://www.mit.edu/~gari/teaching/6.555/SLIDES/BSShandouts.pdf. Accessed: 2014-03-12.

[4] O. Troyanskaya et al. Missing value estimation methods for DNA microarrays. *Bioinformatics Journal*, 17(6):520–525, 2001. URL http://bioinformatics.oxfordjournals.org/content/17/6/520.full.pdf.

[5] J.D. Ullman and A.Rajaraman. *Mining of Massive Datasets*. Cambridge University Press, 2011. URL http://infolab.stanford.edu/~ullman/mmds/ch9.pdf.

[6] Łukasz Kidziński. Statistical foundations of recommender systems. Master's thesis, University of Warsaw, September 2011. URL http://homepages.ulb.ac.be/~lkidzins/works/mgr-math-05.pdf.

[7] P. Drineas, R. Kannan, and M.W. Mahoney. Fast Monte Carlo Algorithms for Matrices II: Computing a Low-Rank Approximation to a Matrix. *SIAM Journal of Computing*, 36(1):158–183, 2006. URL http://cs-www.cs.yale.edu/homes/mmahoney/pubs/matrix2_SICOMP.pdf.

[8] P. Drineas, R. Kannan, and M.W. Mahoney. Fast Monte Carlo Algorithms for Matrices I: Approximating Matrix Multiplication. *SIAM Journal of Computing*, 36(1):132–157, 2006. URL http://cs-www.cs.yale.edu/homes/mmahoney/pubs/matrix1_SICOMP.pdf.

[9] Jr. Richard O. Hill. *Elementary Linear Algebra with Applications*. Harcourt Brace Jovanovich, 2 edition, 1991.

[10] C. Moler. Numerical computing with MATLAB. URL http://www.mathworks.com/moler/eigs.pdf. Accessed: 2014-04-02.

[11] Mason A. Porter Carla D. Martin. The extraordinary SVD. URL http://arxiv.org/pdf/1103.2338.pdf. Accessed: 2014-04-03.

[12] Big O notation [MIT Course 16.070 - lectures], . URL http://web.mit.edu/16.070/www/lecture/big_o.pdf. Accessed: 2014-04-04.

[13] Singular value decomposition, . URL http://campar.in.tum.de/twiki/pub/Chair/TeachingWs05ComputerVision/3DCV_svd_000.pdf. Accessed: 2014-04-17.

[14] Serdar Sari. Using SVD to predict movie ratings. URL http://classes.soe.ucsc.edu/cmps242/Winter08/proj/serdar_talk.pdf. Accessed: 2014-04-20.

[15] Michael Percy. Using SVD to recommend movies. URL http://classes.soe.ucsc.edu/cmps242/Fall09/proj/mpercy_svd_talk.pdf. Accessed: 2014-04-20.

[16] K. Csalogány M. Kurucz, A. A. Benczúr. Methods for large scale SVD with missing values. *Proceedings of KDD Cup and Workshop*, 12:31–38, 2007. URL http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.122.266&rep=rep1&type=pdf.

[17] Kevin Murphy. Eigenvectors and SVD. URL http://www.cs.ubc.ca/~murphyk/Teaching/Stat406-Spring08/Lectures/linalg1.pdf. Accessed: 2014-04-20.