The Implementation of Singular Value Decomposition in Machine Learning

Lily Seebach[1], Dr. Nicholas Kevlahan[2]

Abstract

Machine learning is a subset of artificial intelligence (AI) concerning machines that learn and adapt through "experiences" in the form of data. AI is the larger field which focuses on the ability of machines to mimic and go beyond the capabilities of intelligent human behaviour using mathematical algorithms. With the rapid expansion of the big data industry, an essential part of machine learning is reducing the amount of data a computer needs to store and analyze. Efficient storage and analysis can be achieved using singular value decomposition (SVD) since machine learning data can often be represented as matrices. SVD provides a robust method for the dimension reduction of any matrix by decomposing it into the product of three simpler matrices. Using this decomposition, a low-rank (i.e., compressed) approximation of the data matrix can be found, saving computational memory and reducing analyzation time. The necessity of this simplification in machine learning algorithms leads to many applications of SVD in the field. This research paper focuses on these applications and provides illustrative examples using MATLAB via GitHub. In particular, it gives an overview of machine learning and SVD, and explanations of how SVD is implemented in latent semantic analysis, image compression, and facial recognition. This paper aims to bridge the gap between general public articles and scientific research articles in the current literature by providing annotated code for these algorithms and explaining the necessary linear algebra concepts. The goal of this research is to allow those without a computer science or mathematics background to understand the intersection of numerical linear algebra and AI.

1. Introduction

The field of artificial intelligence (AI) aims to design algorithms which allow computers to mimic and go beyond the abilities of intelligent human behaviour (Columbia Engineering, 2023). Arguably, one of the most intelligent human behaviours is learning—adapting one's behaviour based on experiences and

[1]McMaster University, Hamilton, Ontario L8S 4L8
[2]Supervisor, McMaster University, Hamilton, Ontario L8S 4L8

knowledge acquisition. Machine learning is a sub-discipline of AI which focuses on programming machines to automatically learn and improve their performance based on experiences, just like humans. Machine learning aims to develop algorithms that create models based on data, the computer equivalent to human experiences (Zhou, 2021).

Machine learning is already becoming ubiquitous in society's everyday life. Implemented in facial recognition, medical diagnoses, self-driving cars, and, more recently, ChatGPT, machine learning is no longer constrained to just the fields of computer science and statistics. Now, in the age of big data, machines are being tasked with reading and analyzing more data than ever before. Consequently, reducing the amount of data an algorithm needs to review is essential in creating effective and efficient machine learning programs. Singular value decomposition is a common approach for dimension reduction in machine learning and reduces the amount of data that needs to be stored and analyzed.

This paper elucidates the different ways in which singular value decomposition (SVD) and the related technique principal component analysis (PCA) are used in machine learning algorithms. Specifically, it focuses on latent semantic analysis, image compression, and facial recognition. In addition, sample MATLAB code for each algorithm has been created and is available through GitHub. These files can be downloaded and edited for the reader's enrichment. This paper has been written with the goal that, even with limited knowledge of linear algebra or machine learning, the reader will be able to understand the concepts implemented in machine learning techniques and gain knowledge on the benefits and drawbacks of SVD and PCA within the setting of machine learning algorithms.

2. Machine Learning Overview

In order for a machine to learn from data, it first needs to be fed a large data set from which it can create a model. This data is known as *training data*. Numerical training data can be represented by a data matrix. The data matrix $D = \begin{bmatrix} \vec{x}_1 & \vec{x}_2 & \dots & \vec{x}_n \end{bmatrix}^T$ has rows $\vec{x}_i$ which represent the individual *instances* of each "experience". The columns of D are the *features* or *variables* of the corresponding instance. For example, when creating a data matrix regarding a set of humans' physical traits, the rows (or instances) would represent the individuals and the features would represent quantitative characteristics of the individuals such as age, weight, height, and blood pressure. Each entry of a data matrix, x_{ij}, is known as a *feature value* and represents the value of the j^{th} feature for the i^{th} instance. In the previous example, the feature values for the first individual could be 37 years, 173 cm, 60 kg, and 120 mm Hg, respectively. With m instances and n features, the resulting data matrix will take the shape $m \times n$. Depending on the purpose of the algorithm, just this information may not be enough for a machine to build a model. If the algorithm should be able to predict information of new instances based on the training data, it must be told the *outcome* or *label* of each instance. In the previous example, the outcome

could be lifespan. Then the final form of the instance will be $(\vec{x}_i, y_i)$. The goal of machine learning algorithms is to find the underlying rules of the data. These rules are known as the *ground-truth*. (Zhou, 2021).

Machine learning algorithms can be classified as supervised or unsupervised based on the presence of labels. Supervised learning algorithms are those that detect patterns within labelled training data to predict the outcome of new, unlabelled cases. In contrast, unsupervised learning algorithms group unlabelled data into clusters by "learning" the underlying features of the data and grouping instances with similar features together (Alloghani et al., 2020). SVD is one of the most commonly used unsupervised machine learning algorithms and is central to many dimension reduction-based algorithms (Luboobi, 2018).

3. Singular Value Decomposition and Principal Component Analysis

Since data used in machine learning can be represented as a matrix, linear algebra techniques can be used within algorithms. Further, numerical linear algebra can be used to create efficient and accurate algorithms. Numerical linear algebra is the study of how to optimize linear algebra techniques on computers. Computers have finite memory, so they are unable to represent every number exactly. For example, π cannot be exactly expressed on a computer since it is an irrational number. This imprecision can be exacerbated by linear algebra techniques and lead to large errors in solutions. Thus, numerical linear algebra attempts to minimize these errors using carefully designed algorithms.

Singular value decomposition (SVD) is one of the most common applications of linear algebra in machine learning. SVD offers a robust method for matrix dimension reduction by factoring the data matrix into the product of three simpler matrices. It is a generalization of the more "standard" diagonalization of a matrix called *eigendecomposition*. SVD borrows key steps and concepts from the standard diagonalization method and thus it is important to first understand how to compute the eigendecomposition of a matrix.

3.1. Eigendecomposition

The term *eigendecomposition* arises from the use of eigenvalues and eigenvectors in the decomposition of an $n \times n$ matrix, A. A vector $\vec{v}$ is considered an eigenvector of A if it satisfies

$$A\vec{v} = \lambda\vec{v} \tag{1}$$

for λ scalar. When the equation is satisfied, λ is called the eigenvalue of A corresponding to eigenvector $\vec{v}$. Equation 1 reduces to solving the *characteristic polynomial* $\det(A - \lambda I_{n\times n})=0$, where $I_{n\times n}$ is the $n \times n$ identity matrix. Finding the solution of the characteristic polynomial equates to solving for the roots of an n^{th} order polynomial. These roots are the eigenvalues of A, λ_i for $i = 1, 2, ..., n$. To solve for the corresponding eigenvectors, $\vec{v_i}$ for $i = 1, 2, ..., n$, we solve the equation

$$(A - \lambda I_{n\times n})\vec{v} = 0 \tag{2}$$

for $\vec{v}$, which is equivalent to equation 1. Numerical linear algebra provides methods for solving both the characteristic polynomial and equation 2. Matrix A can then be diagonalized by the decomposition

$$A = P^{-1}DP \tag{3}$$

if there are n linearly independent eigenvectors for A. In this case, matrix D is a diagonal matrix with diagonal entries λ_i, ordered from largest to smallest. P is an $n \times n$ matrix with the i^{th} column equal to v_i, the corresponding eigenvector to λ_i.

The drawback of this method is it only works for square matrices with n linearly independent eigenvectors. In machine learning data matrices, there is too much variability to guarantee these properties. SVD solves this issue by providing a method to diagonalize any matrix, regardless of size or rank.

3.2. Singular Value Decomposition

SVD diagonalizes any $m \times n$ matrix A by decomposing it into three matrices, $U\Sigma V^T$, and takes the form of Figure 1. Finding the decomposed matrices for the SVD of a matrix follows similar steps as finding the eigendecomposition.

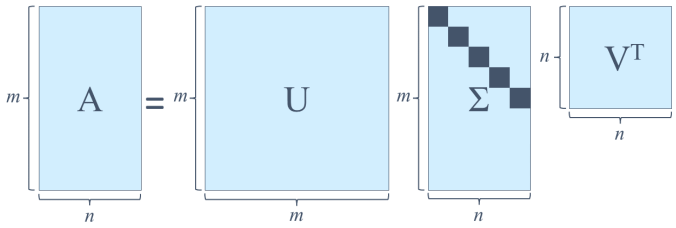

Figure 1: The singular value decomposition of an $m \times n$ matrix A into the product of $U\Sigma V^T$. A is an $m \times n$ matrix, U is an $m \times m$ matrix, Σ is an $m \times n$ matrix, and V^T is an $n \times n$ matrix. The blue boxes within Σ represent the singular values of A and show Σ is a diagonal matrix.

The columns of U are the eigenvectors of AA^T and are called the m *left singular vectors* of A. Similarly, the columns of V are the n *right singular vectors* of A and are the eigenvectors of A^TA. The inverse of U and V are equal to their transpose, making them *unitary* and *orthogonal* matrices. Finally, Σ is a diagonal matrix where the diagonal entries, σ_i, are the *singular values* of A. The singular values of A are defined as the square roots of the non-zero eigenvalues of the matrix A^TA and AA^T. Typically, the singular values are ordered such that $\sigma_i \geq \sigma_{i+1}$ and the eigenvectors (columns of U and V) are ordered to their corresponding singular value. The same method for finding eigenvectors and eigenvalues described in eigendecomposition can be used on A^TA and AA^T to find the necessary matrices for SVD.

SVD is also used to define the *numerical condition number* of a matrix A. A larger condition number indicates that solving a linear system involving the matrix will be ill-conditioned and very sensitive to small changes to the input. Typically, the 2-norm condition number of a matrix is defined by

$$cond_2(A) = \frac{\sigma_{max}}{\sigma_{min}}. \tag{4}$$

However, in the numerical computation of the SVD of a matrix, it is common to replace any $\sigma_i < \varepsilon\sigma_1$ with 0, where ε is the machine precision of the computer. As such, the numerical condition number is defined as

$$cond_2(A) = \frac{\sigma_1}{\sigma_k} \tag{5}$$

where σ_k is the smallest singular value larger than $\varepsilon\sigma_1$. This practice improves the conditioning of A.

The two most common uses for the SVD of a matrix are finding a low-rank (i.e., compressed) approximation of A and finding a pseudo-inverse A^+ of a singular matrix. A low-rank approximation allows a computer to retain a good estimate of A while saving computational memory. To begin, we write A as a linear combination of rank-one matrices:

$$A = U\Sigma V^T = \sum_{i=1}^{r} \sigma_i E_i = \sigma_1 E_1 + \sigma_2 E_2 + ... + \sigma_r E_r \tag{6}$$

where $E_i = u_i v_i^T$ are outer products of the columns of U and V, respectively. Further, σ_i are ordered largest to smallest. The *best rank* $k \leq r$ *approximation of* A is then defined by $\hat{A}_k = \sum_{i=1}^{k} \sigma_i E_i = U_k \hat{\Sigma}_k V_k^T$, where U_k is the matrix of the first k columns of U, V_k^T is the matrix of the first k rows of V^T, and $\hat{\Sigma}_k$ is the first k columns and rows of Σ, as seen in Figure 2. This method is also called *truncated SVD* since we are truncating the matrices from the typical SVD.

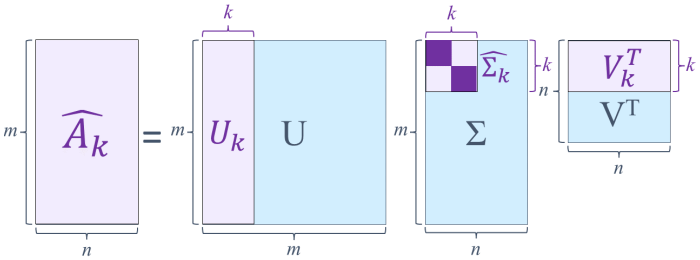

Figure 2: The low-rank approximation of an $m \times n$ matrix A using singular value decomposition compared to the original singular value decomposition. $\hat{A}_k$ is the same size as A but the factorized matrices decrease in size. The dark purple squares in $\hat{\Sigma}_k$ represent the singular values in the diagonal matrix. U is $m \times k$, $\hat{\Sigma}_k$ is $k \times k$ and V^T is $k \times n$. This truncation means the computer needs to stores $k(m + n + 1)$ values instead of $m \times n$ values.

By truncating these matrices and limiting the number of singular values needed, the computer is required to store less data. Specifically, it only needs to store $\{\sigma_i, u_i, v_i\}$ for $i = 1, 2, ..., k$. Using a low-rank approximation also improves the numerical condition number of A (see equation 5), meaning it is less sensitive to changes in the input data.

The second application of SVD finds the least-squares solution to any linear system $A\vec{x} \approx \vec{b}$ even in the rank-deficient case. The first case we consider is the overdetermined case, where there are more equations m than unknowns n. An example of an overdetermined system is trying to fit a parabola to seven data points since there are seven equations and only three unknowns, $ax^2 + bx + c$. We begin by finding the SVD of A in the form of block matrices: $A = U\Sigma V^T = \begin{bmatrix} U_1 & U_2 \end{bmatrix} \begin{bmatrix} \Sigma_1 \\ 0 \end{bmatrix} V^T = U_1 \Sigma_1 V^T$. Then

$$A\vec{x} \approx \vec{b} \Leftrightarrow U_1 \Sigma_1 V^T \vec{x} \approx \vec{b}. \tag{7}$$

Equation 7 implies that $\vec{x} = V\Sigma_1^+ U_1^T \vec{b}$ is the least squares solution (since U and V are unitary matrices). Then the Moore-Penrose pseudo-inverse of A is defined as

$$A^+ = V\Sigma_1^+ U_1^T \tag{8}$$

where the n diagonal entries of Σ_1^+ are

$$\sigma_i^+ = \begin{cases} 1/\sigma_i & \text{if } \sigma_i \neq 0 \\ 0 & \text{if } \sigma_i = 0 \end{cases}. \tag{9}$$

In the underdetermined case, there are fewer equations m than unknowns n. An example of the underdetermined case is trying to fit a parabola to two data points, since there are two equations but three unknowns. We can still use A^+ to find a least-squares solution to $A\vec{x} \approx \vec{b}$. The solution in this case differs from the overdetermined case due to the formation of the block matrices. In the underdetermined case we have $A = U\Sigma V^T = U \begin{bmatrix} \Sigma_1 & 0 \end{bmatrix} \begin{bmatrix} V_1 & V_2 \end{bmatrix}^T = U_1 \Sigma_1 V^T$. Then

$$A\vec{x} \approx \vec{b} \Leftrightarrow U\Sigma_1 V_1^T \vec{x} \approx \vec{b}. \tag{10}$$

Equation 10 implies $V_1 \Sigma_1 U^T \vec{b}$ is the least squares solution and the Moore-Penrose pseudo-inverse of A is

$$A^+ = V_1 \Sigma_1 U^T \tag{11}$$

Using SVD to find the pseudo-inverse ensures we can always find a solution to the matrix equation, even in the rank-deficient case. If A is square and non-singular, $A^+ = A^{-1}$ and the solution for $A\vec{x} = b$ is exact. Within numerical linear algebra, methods exist to improve the conditioning of the least-squares solution by finding the pseudo-inverse and limiting the values of σ_i, in turn improving the numerical condition number of A, just like in the low-rank approximation. Examples of MATLAB code to find the SVD of a matrix, a low-rank approximation, and the pseudo-inverse of a matrix can be found in the GitHub repository under SVD_and_PCA.m.

3.3. Principal Component Analysis

Another method of dimension reduction closely related to SVD is principal component analysis (PCA). The basis of PCA is reducing the dimension of the data as much as possible while retaining as much of the data's variability as possible (Jolliffe and Cadima, 2016). Mathematically, retaining the data's variability is equivalent to finding new uncorrelated variables that are linear functions of the original variables in the dataset. These new variables successively maximize variance. The first step in conducting PCA on an $m \times n$ matrix X is column-centering X by subtracting the column mean from each element in the column:

$$x^*_{ij} = x_{ij} - \bar{x}_j. \tag{12}$$

Sometimes this first step also includes dividing by the column's standard deviation. Next, the covariance matrix is computed. The covariance of two variables x, y is defined as $cov(x, y) = \frac{\Sigma_{i=1}^{N}(x_i - \bar{x})(y_i - \bar{y})}{N-1}$, where N is the number of data points. The covariance of a variable with itself is the variance of the variable. In the case of data matrices, the features (columns) are the variables. The covariance matrix is $n \times n$ and is calculated by

$$C = \begin{bmatrix} var(x_1) & cov(x_1, x_2) & ... & cov(x_1, x_n) \\ cov(x_2, x_1) & var(x_2) & ... & cov(x_2, x_n) \\ ... & ... & ... & ... \\ cov(x_n, x_1) & cov(x_n, x_2) & ... & var(x_n) \end{bmatrix}.$$

Then, we solve for the eigenvalues λ and eigenvectors $\vec{v}$ of the covariance matrix C using the technique described in the Eigendecomposition section (Jolliffe and Cadima, 2016). The eigenvector $\vec{v_i}$ corresponding to the largest eigenvalue λ_i is said to be the first principal component, the eigenvector corresponding to the second largest eigenvalue is the second principal component, and so on. Each eigenvector will be orthogonal to the others. These eigenvectors successively maximize variance since they are the directions of the axes where there is the most variance, as seen in Figure 3. The eigenvalues represent the amount of variance carried in each principal component and the percent of variance an eigenvector contains can be calculated by dividing the corresponding eigenvalue by the sum of the eigenvalues.

Similarly to truncated SVD, we can create a *feature vector* by disregarding principal components corresponding to the smaller eigenvalues and thus contain little of the data's variance. Finally, we project the standardized data on to the feature vector. To do this, we multiply the feature vector by the original standardized data set (Jolliffe and Cadima, 2016). The feature vector of a matrix can also be obtained from the SVD of the column-centered data matrix as in Equation 12. From here, the matrix is subject to SVD and the columns of V are considered the principal components. The comparison between SVD and PCA can be seen in the SVD_and_PCA.m file in the linked GitHub. Note that the columns of V and the principal components may differ by a factor of -1. This factor just rotates the vector 180° so they are still representing the same axis, just oriented in the opposite direction.

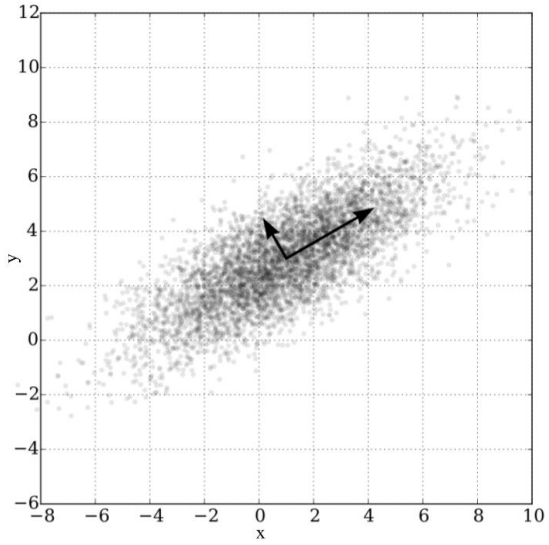

Figure 3: An example of two principal components for a Gaussian distribution. The first principal component is pointing up and to the right, representing the axes of most of the data's variability. The second principal component is pointing up and to the left and contains less of the data's variability. The principal components are orthogonal (Adapted from Nicoguaro, 2016).

4. Singular Value Decomposition in Machine Learning Algorithms

With this understanding of SVD and PCA, several machine learning algorithms can be understood. All algorithms discussed in this paper focus on dimension reduction and either use truncated SVD, as in latent semantic analysis and image compression, or a feature vector from PCA, as in facial recognition.

4.1. Latent Semantic Analysis

First, SVD can be used in a technique known as *information retrieval* (IR). IR is the process of selecting unstructured material stored within large collections to satisfy an information need (Manning, Raghavan, and Schütze, 2009). One common example of IR is choosing which documents to select from a database based on a search term known as a query. This process is known as latent semantic analysis (LSA). Without LSA, documents are retrieved by matching terms in documents to terms in a query directly. This technique is known as lexical matching (Berry et al., 1995). However, using this approach can cause errors due to the complexity of the human language. Synonyms can cause documents which relate to the query not to be retrieved since the words do not match directly. Conversely, polysemy (words having multiple meanings) can cause irrelevant documents to be retrieved. LSA attempts to overcome these issues by accounting for the underlying (i.e., latent) structure of language. SVD is frequently used in LSA to estimate this structure (Berry et al., 1995).

Before understanding the role SVD plays in LSA, we must understand what matrix we are decomposing. This matrix is the term-document matrix, S, and identifies which words appear in which documents. The features (columns) are documents and the instances (rows) are terms. In a simple example, let's say we have two documents. The first reads "I like dogs," and the second reads "You do not like cats." A term-document matrix for these two documents would be:

	D1	D2
i	1	0
you	0	1
do	0	1
not	0	1
like	1	1
dogs	1	0
cats	0	1

Like the example, each entry of the term-document matrix, s_{ij}, is the frequency of the i^{th} term in the j^{th} document. Retrieval performance can be further improved by applying local and/or global weightings to the terms. A global weight represents the importance of a term in the collection while a local weight represents the importance of a term within a singular document. Global weights are applied to terms (rows) while local weights are applied to specific entries (Dumais, 1991). Entries of S are thus represented as $s_{ij} = L(i,j) \times G(i)$ where $L(i,j)$ is the local weight for the i^{th} term in the j^{th} document and $G(i)$ is the global weight for the i^{th} term. Many different weighting schemes have been proposed and their accuracy have been tested (Dumais, 1991). The simplest local weighting scheme is raw term frequency where s_{ij} is given by the frequency of the term in the corresponding document. The most commonly used global weighting scheme is $\sqrt{\frac{1}{\Sigma_j tf_{ij}^2}}$, where tf_{ij} is the frequency of term i in document j. This global weighting scheme, along with many others, places less weight on more common terms (Dumais, 1991).

Once the term-document matrix has been created, the SVD of S is computed. In this case, U represents the term vectors and V represents the document vectors. By finding the best k-rank approximation of S and ensuring this matrix is not exactly the original matrix S, truncated SVD maps S from a word-based vector space to a lower-dimensional, semantic vector space (Berry et al., 1995; Dong and Liu, 2018). It is important the low-rank approximation matrix does not exactly match S as to remove the variability (noise) in word choice (Berry et al., 1995). For the individual matrices, U_k maps each *term* into a k-dimensional space while V_k maps each *document* into a k-dimensional space. Geometrically, each dimension in the k-dimensional space can be thought to represent a latent concept (Dong and Liu, 2018), and terms which occur in similar documents will be near each other in this space (Berry et al., 1995). These compressed matrices are the ones used for document retrieval based on queries in LSA. Queries can be represented similarly to documents, as a set of words translated into a matrix. We define q as the vector of terms in the query, multiplied by the appropriate

weights. Transforming q to

$$\hat{q} = q^T U_k \Sigma_k^{-1}$$

changes the original vector to a weighted sum of its term vectors and maps q to the k-dimensional semantic vector space. Finally, the similarity of the query to each document is computed to determine the most relevant documents. A common similarity measure for this purpose is the cosine similarity between the query and document vectors represented as the columns of V_k^T (Berry et al., 1995). The documents can then be returned to the inquirer in order of relevance to their query. The number of documents returned can either be pre-determined (i.e., the program will return the p closest documents) or a minimum threshold for similarity will be set (e.g., the program will return the documents with a cosine similarity above 0.9)(Berry et al., 1995).

A LSA algorithm also needs to have the ability to update the lower-dimension space as new documents are added since it is rare for a database to be complete when it is first created. As new documents are added, new terms will also have to be added. There are currently two prominent options for updating: *SVD-updating* or *folding-in* the new documents. There are benefits and drawbacks to each method. Another option is recomputing the SVD of the new term-document matrix but this is computationally expensive and may be unfeasible depending on storage amount. This technique is also not considered an updating method since it creates a new semantic vector space with a different approximation matrix. We will consider only the folding-in updating method and details for SVD-updating can be found in O'Brien, 1994. The folding-in method is less computationally expensive, but may misrepresent new documents by placing them in the pre-existing semantic vector space, meaning the placement of the original documents will not change. Folding-in documents or terms is a similar process to finding a query representation. The document vector d is projected onto the span of the pre-existing term vectors by $\hat{d} = d^T U_k \Sigma_k^{-1}$, then appended as a column to V_k^T, the set of document vectors. Adding new terms is done similarly; a row term vector t is projected onto the span of the pre-existing documnet vectors by $\hat{t} = t V_k \Sigma_k^{-1}$, and this vector is appended as a row to the existing U_k, the set of term vectors (Berry et al., 1995).

An example of LSA can be found in the GitHub repository under the file LSA.m. This example uses first- and second-year mathematics textbook titles as documents and folds-in two third-year mathematics textbooks. The original titles can be seen in Table 1. To create the term-document matrix, words with less than four letters were removed and raw term frequency weighting was used.

We can plot the documents in two-dimensions to see documents near each other when a 2-rank approximation of S is used. This plot can be seen in Figure 4. We can see some groupings of common concepts. For example, d2, d6, and d10 are near each other, which are all about linear algebra. Similarly, d4 and d5 are near each other which are both about calculus. The transformed query vector for the search term "statistics" is also shown in red.

Label	Textbook Title
D1	Calculus Early Transcendentals
D2	Linear Algebra and its Applications
D3	The Tools of Mathematical Reasoning
D4	Applied Calculus for Business, Economics, and the Social and Life Sciences
D5	Calculus for the Life Sciences Modelling the Dynamics of Life
D6	Elementary Linear Algebra Applications Version
D7	Finite Mathematics and Its Applications
D8	Elementary Differential Equations with Boundary value Problems
D9	Mathematics of Investment and Credit
D10	Linear Algebra Done Right
D11	99 Numbers: Mathematics of Everyday Life
D12	Biostatistics for the Biological and Health Sciences
D13	Introduction to Probability
D14	Probability and Statistics for Engineering and the Sciences

Table 1: Textbook titles used in the MATLAB example of LSA and their corresponding labels.

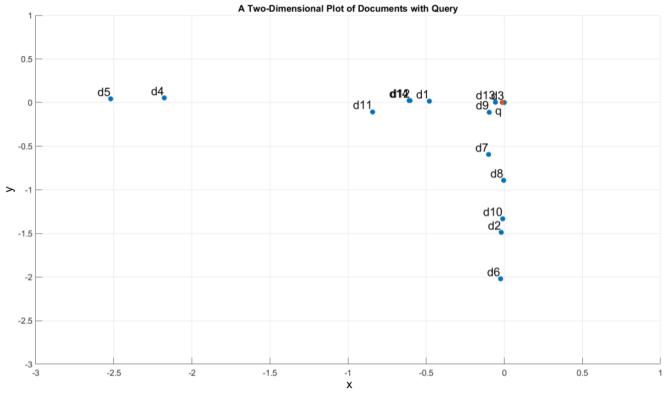

Figure 4: A plot produced by MATLAB showing the two-dimensional approximation of the document titles shown in Table 1. The query vector q for "statistics" mapped to the two-dimensional space is also shown in red.

When using a best 5-rank approximation and a cosine similarity threshold of 0.9, the algorithm returns d12, d13, and d14 based on the query. These textbook titles have a cosine similarity to the query of 0.9714, 0.9987, and 0.9873, respectively. Textbooks d12 and d13 would not have been retrieved using lexical matching since they do not contain the word "statistics", suggesting LSA is an improvement on lexical matching since these textbooks do relate to statistics. The example then updates the term-document matrix by adding two third-year mathematics textbooks, d15: Introduction to Mathematical Statistics, and d16: Applied Statistics and Probability for Engineering. The documents go through the same pre-processesing as the original textbooks, are mapped to latent space

and are appended to the matrix V_k^T. The two-dimensional plot is updated with these textbooks, as seen in Figure 5. Note that the original documents did not change from Figure 4. This property arises from the algorithm of folding-in and is a drawback since all documents will be placed into the pre-determined semantic vector space.

The low-rank approximation matrices can also be used to compute the similarity between documents or terms by computing the similarity of the columns of V_k^T or the rows of U_k, respectively. The ability to accurately retrieve documents and locate similar documents has applications in many fields, such as healthcare and research.

Figure 5: A plot produced by MATLAB showing the two-dimensional approximation of the document titles shown in Table 1 and d15 (red) and d16 (yellow).

While SVD is by far the most common algorithm for LSA, other algorithms are used, such as non-negative matrix factorization which is a non-exact matrix factoring (Peter et al., 2009). Other variations of LSA depend on the term weighting schemes or the similarity measures.

4.2. Image Compression

SVD and PCA are also used in a variety of image-based machine learning algorithms. In particular, truncated SVD can be used for image compression. Image compression reduces the amount of unimportant information, or redundancy, in an image (Kahu and Rahate, 2013). Redundancy in an image can exist in many forms. One of the most recognized redundancies is spatial redundancy, which is similarity between nearby pixels (Kahu and Rahate, 2013). Image compression can be further categorized into lossy image compression, like the SVD technique, and lossless image compression (Mounika, Sri Navya Lakshmi, and Alekya, 2015). If there is any loss of details due to the compression, the technique is known as a lossy technique, even if the details are not detectable by the human eye (Prasantha, Shashidhara, and Balasubramanya Murthy, 2007). These algorithms are usually used for everyday photographs where perfect quality

is not crucial. Lossy techniques also cannot reconstruct the original image after compression (Prasantha, Shashidhara, and Balasubramanya Murthy, 2007). In contrast, a lossless technique will produce an exact copy of the original photo, so the image can be reconstructed after compression. These techniques are more applicable to medical images, where every detail needs to be present (Prasantha, Shashidhara, and Balasubramanya Murthy, 2007).

The simplest example of image compression is a grayscale image. In this case, each pixel is assigned a value from 0 (black) to 255 (white) and thus can be represented as a single matrix. By finding a low-rank approximation of this matrix, we can discard the singular values containing little information about the image. Visually, this truncation should not cause great changes to the image, however, the memory required for the storage of the compressed image is much less than that of the original. Specifically, the uncompressed image requires mn values to be storage for an $m \times n$ image, while the compressed image only requires $k(m+n+1)$ values be stored, where k is the number of singular values kept (Kahu and Rahate, 2013). The compression ratio of this process is $\frac{mn}{k(m+n+1)}$. Because singular values tend to decrease quickly, smaller singular values contain very little information about the image, so after retaining some k singular values, retaining more does not impact the quality of the image to the human eye. This trend can be seen in Figure 6. This figure and Figure 7 were created using the code in Image_Compression.m. Other grayscale images (from Zhang et al., 2016) and other colour images (from Afifi et al. 2021) can be found in the repository and downloaded to use in this file.

Figure 6: A grayscale image with varying compression ratios. The original image from Suwal (2021) is shown in the top left. The original image has 1120 singular values and the numbers of singular values used for the compressed images are shown above. Very little differences can be seen between the image created using 150 singular values and 300 singular values. This is because the singular values added hold very little information about the image.

The more complex, but more applicable, case is a colour image. Since a colour image is composed of red, green, and blue layers, it is represented by three matrices instead of one matrix like grayscale images. Each pixel is then assigned three values, one for the saturation level of each colour. These entries range from 0 (not saturated) to 255 (fully saturated) (Swathi et al., 2017). In this case, all three colour matrices must be truncated to the same degree. Objects in compressed colour images can be easier to identify than in grayscale images since the colours can help differentiate items, as seen in Figure 7.

Figure 7: A colour image with varying compression ratios. The original image from Afifi et al. (2021) is shown in the top left. Each colour layer has 684 singular values and the numbers of singular values used for the compressed images are shown above. Very little differences can be seen between the image created using 100 singular values and 200 singular values. This is because the singular values added hold very little information about the image.

4.3. Facial Recognition

An image-based machine learning algorithm which uses PCA instead of SVD is facial recognition. First the machine is fed a training set of images of faces. Assuming each image is $m \times n = M$ pixels, we can turn each image into an $M \times 1$ vector, $\vec{f_i}$. If there are N images of the same size, vectors are concatenated to create an $M \times N$ matrix, such $F = [\vec{f_1}, \vec{f_2}, ..., \vec{f_N}]$ where each $\vec{f_i}$ is an image of a face transformed to a vector. Then, the mean face of the set is calculated by $\bar{f} = \frac{1}{N}\Sigma_{i=1}^{N}\vec{f_i}$. We re-define F by subtracting the mean image from each vector image:

$$F = [\vec{f_1} - \bar{f}, \vec{f_2} - \bar{f}, ..., \vec{f_N} - \bar{f}].$$

The next step in PCA is solving for the eigenvectors of the covariance matrix defined as $C = FF^T$. However, depending on computational limitations, this step may result in a matrix of an unfeasible size (Turk and Pentland, 1991). For example, the image in Figure 6 is 1120 $\times$ 1120 pixels. If this were the size of all images in F, the covariance matrix C would be 1254400×1254400. To keep computations efficient, we instead find the eigenvectors of F^TF, multiply them

by F, and normalize them to find the eigenvectors of the covariance matrix (Kim, n.d.). These steps are essentially PCA. The eigenvectors u_i form an orthonormal basis for the column subspace of F, meaning every column of F can be expressed as a linear combination of the eigenvectors (Zeng, 2007). This gives rise to the concept of *eigenfaces*. Eigenfaces are the images (faces) that are created from the eigenvectors when they are reshaped to the original image shape and this set of eigenvectors make up the *face space* (Zeng, 2007). Furthermore, the eigenvalues can be used to order the eigenfaces by the amount of variance they represent in the images and consequently, how useful they are in characterizing faces (Turk and Pentland, 1991). As such, the face space can be defined by a subset of the eigenfaces by using the first k eigenfaces. The face space and eigenfaces can be used to detect which face matches a new, unknown face, assuming it is a face present in the database. First, the new image, F', is projected onto the face space by computing $\omega_i = u_i^T(F' - \bar{f})$ for $i = 1, 2, ..., k$. These calculations form the vector $\Omega^T = [\omega_1\ \omega_2\ ...\ \omega_k]$ where each ω_i is the weight of the contribution of each selected eigenface in representing the new image. This vector is the one used to recognize the face. Specifically, if $\epsilon = ||\Omega - \Omega_l||$, where Ω_l is the vector describing the l^{th} face class, is lower than a set threshold, then the face is said the belong to such face class. If the threshold is not met for any class, the face is deemed unknown (Turk and Pentland, 1991).

A small-scale example of a facial recognition algorithm can be found in Facial_Recognition.m. This code uses images from Thakur (2022) which are available in the folders Celebrity-Images and Test-Images. The images were chosen to be front-facing profiles and were cropped and aligned so each face was approximately in the same place in the photo. The images were then shrunk to 100 × 100 pixels. Small discrepancies may arise due to errors in the centering of the faces while removing their backgrounds. The code uses grayscale celebrity images to find the average face of a dataset, find the eigenfaces of a data set, reconstruct a face using varying amount of eigenfaces (Theodoridis, 2015), and create a simple facial recognition algorithm.

The average face of the set was calculated by finding the average value of each row and reshaping the resulting mean vector to the original image size (100 × 100). This face is shown in Figure 8. The eigenfaces for the set was also found using the method described above and the first eight eigenfaces are shown in Figure 9.

Figure 8: The average face of the celebrities used for this algorithm.

These eigenfaces may look unexpected in comparison to other eigenfaces available in the literature. The reason for this discrepancy is thought to be attributed to the limited number of images used in the algorithm, making the lines more distinct, as well as imprecise alignment of the faces during pre-processing, causing facial features of different images to not align.

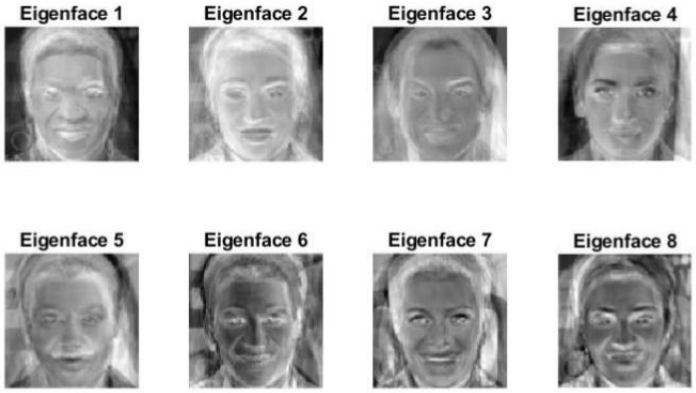

Figure 9: The first eight eigenfaces for the data set.

This facial recognition algorithm is not created with the intent to work for any image of the included celebrities but rather was written to demonstrate the steps of facial recognition using eigenfaces and PCA.

5. Conclusion and Future Directions

Singular value decomposition and principal component analysis clearly have many applications within machine learning, including textual and image-based algorithms. LSA and image compression both rely on SVD to find a low-rank approximation of the matrices of interest. In contrast, facial recognition uses an adaptation of PCA to find a basis for the columns of the matrix of interest. This paper presents the most basic algorithms for these three techniques. Future research should look into what proposed modifications to these algorithms exist within the literature and how they impact the output. Other research could focus on the different forms of SVD and PCA used in machine learning, some of which are non-linear. Two examples are k-SVD, which is used for sparse dictionary modelling, and kernel PCA, which is a method on non-linear dimension reduction. As the research in this area expands, it is important to ensure the resulting literature is accessible to everyone, regardless of their academic background. Overall, numerical linear algebra is crucial to machine learning and, as the AI industry continues to expand, it is important to continue searching for new methods and improvements upon existing methods. It is in this manner that we will be able to continue developing technology which has implications in areas outside computer science.

References

[1] Afifi, M., Brubaker, M. A., Brown, M. S., 2021. Histogan: Controlling colors of gan-generated and real images via color histograms. In: 2021 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR). pp. 7937–7946.

[2] Alloghani, M., Al-Jumeily, D., Mustafina, J., Hussain, A., Aljaaf, A. J., 2020. Supervised and Unsupervised Learning for Data Science. In: Berry, M. W., Mohamed, A., Yap, B. W. (Eds.), A Systematic Review on Supervised and Unsupervised Machine Learning Algorithms for Data Science, 1st Edition. Unsupervised and Semi-Supervised Learning. Springer Cham, Cham, pp. 3–21.
URL https://doi.org/10.1007/978-3-030-22475-2_1

[3] Berry, M. W., Dumais, S. T., O'Brien, G. W., 1995. Using linear algebra for intelligent information retrieval. SIAM Review 37 (4), 573–595.
URL http://www.jstor.org/stable/2132906

[4] Columbia Engineering, 2023. Artificial Intelligence (AI) vs. Machine Learning.
URL
https://ai.engineering.columbia.edu/ai-vs-machine-learning/

[5] Dong, G., Liu, H. (Eds.), Apr. 2018. Feature Engineering for Machine Learning and Data Analytics, 1st Edition. CRC Press, Boca Raton.
URL http://doi.org/10.1201/9781315181080

[6] Dumais, S. T., Jun. 1991. Improving the retrieval of information from external sources. Behavior Research Methods, Instruments, & Computers 23 (2), 229–236.
URL https://doi.org/10.3758/BF03203370

[7] Jolliffe, I. T., Cadima, J., Apr. 2016. Principal component analysis: a review and recent developments. Philosophical Transactions of the Royal Society A 374 (2065), 20150202, publisher: Royal Society.
URL
https://royalsocietypublishing.org/doi/10.1098/rsta.2015.0202

[8] Kahu, S., Rahate, R., Aug. 2013. Image Compression using Singular Value Decomposition. International Journal of Advancements in Research & Technology 2 (8), 244–248.

[9] Kim, K., n.d. Face Recogntion using Principal Component Analysis.
URL http://staff.ustc.edu.cn/~zwp/teach/MVA/pcaface.pdf

[10] Luboobi, P., Feb. 2018. Foundations of Machine Learning : Singular Value Decomposition (SVD).
URL https://medium.com/the-andela-way/foundations-of-machine-learning-singular-value-decomposition-svd-162ac796c27d

[11] Manning, C. D., Raghavan, P., Schütze, H., Apr. 2009. An Introduction to Information Retrieval. Cambridge University Press, Cambridge, England.

[12] Mounika, K., Sri Navya Lakshmi, D., Alekya, K., Apr. 2015. SVD Based Image Compression. International Journal of Engineering Research and General Science 3 (2), 1271–1278.

[13] Nicoguaro, Feb. 2016. Gaussian Scatter PCA.

[14] O'Brien, G. W., 1994. Information management tools for updating an svd-encoded indexing scheme. Tech. rep., University of Tennessee, USA.

[15] Peter, R., Shivapratap, G., Divya, G., Soman, K., May 2009. Evaluation of SVD and NMF Methods for Latent Semantic Analysis. International Journal of Recent Trends in Engineering 1 (3), 308–310.

[16] Prasantha, H., Shashidhara, H., Balasubramanya Murthy, K., 2007. Image compression using svd. In: International Conference on Computational Intelligence and Multimedia Applications (ICCIMA 2007). Vol. 3. pp. 143–145.

[17] Suwal, M. S., 2021. bnw.
URL https://www.kaggle.com/datasets/merishnasuwal/grayscale-images

[18] Swathi, H. R., Sohini, S., Surbhi, Gopichand, G., Nov 2017. Image compression using singular value decomposition. IOP Conference Series: Materials Science and Engineering 263 (4), 042082.
URL https://dx.doi.org/10.1088/1757-899X/263/4/042082

[19] Thakur, V., Jul. 2022. Celebrity Face Image Dataset.
URL https://www.kaggle.com/datasets/vishesh1412/celebrity-face-image-dataset

[20] Theodoridis, S., 2015. Machine Learning: A Bayesian and Optimization Perspective. Elsevier Science & Technology, Kent.
URL http://ebookcentral.proquest.com/lib/mcmu/detail.action?docID=2007481

[21] Turk, M., Pentland, A., Jan. 1991. Eigenfaces for Recognition. Journal of Cognitive Neuroscience 3 (1), 71–86, _eprint: https://direct.mit.edu/jocn/article-pdf/3/1/71/1932018/jocn.1991.3.1.71.pdf.
URL https://doi.org/10.1162/jocn.1991.3.1.71

[22] Zeng, G., Aug. 2007. Facial Recognition with Singular Value Decomposition. In: Elleithy, K. (Ed.), Advances and Innovations in Systems, Computing Sciences and Software Engineering. Springer Dordrecht, Dordrecht, Netherlands, pp. 145–148.

[23] Zhang, K., Zuo, W., Chen, Y., Meng, D., Zhang, L., Aug. 2016. Beyond a Gaussian Denoiser: Residual Learning of Deep CNN for Image Denoising. Institute of Electrical and Electronics Engineers 26 (7), 3142–3155.

[24] Zhou, Z.-H., 2021. Machine Learning, 1st Edition. Springer Singapore, Singapore.
URL https://doi.org/10.1007/978-981-15-1967-3