

Singular Value Decomposition

Lecture Notes for Big Data Analytics

Imdad Ullah Khan

April 2019

Contents

1	Rank of a matrix	2
1.1	Rank Factorization of a matrix	2
1.1.1	Equivalence between these definitions	3
1.1.2	Singular and non-singular matrices	3
2	Low Rank Approximation	3
2.1	Why Low Rank Approximability is Expected	5
3	SVD Definition	7
4	Low Rank Approximation from SVD (Truncated SVD)	10
4.1	Spectral decomposition of a matrix - Decomposition into rank-1 matrices	10
4.2	Truncated SVD	11
4.2.1	How to choose k	12
5	Application of Low Rank Approximation	12
5.1	Data Compression	12
5.2	Recommender System	12
5.2.1	Using SVD to factorize the rating matrix	14
5.3	Latent Semantic Analysis and Word Embeddings	15
5.4	Denosing	18
6	Relation of SVD and eigen-decomposition	18
7	Computing the SVD: The power method	19

1 Rank of a matrix

Given a $n \times m$ matrix A

Column Rank of A , $\text{col-rank}(A)$ is the maximum number of linearly independent columns of A

Row Rank of A , $\text{row-rank}(A)$ is the maximum number of linearly independent rows of A For any matrix A , $\text{rank}(A) := \text{col-rank}(A) = \text{row-rank}(A)$ Looking at A as a linear transformation, $\text{rank}(A)$ is the (true) dimensionality of the range of A (dimensions of the output). Note that $n \times m$ matrix maps $m \times 1$ vectors to $n \times 1$ vectors, but the true dimensionality of output may be very low. As we discussed, suppose $\text{rank}(A) = 1$ (all columns are linear dependent), then A maps all vectors to a line (one-dimensional subspace of \mathbb{R}^n)

For example $\begin{bmatrix} 2 & 4 \\ 3 & 6 \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} 2x + 3y \\ 4x + 6y \end{bmatrix}$, thus all vectors are mapped to the line $y = 2x$

Another equivalent definition of rank also called the decomposition rank of a matrix.

Rank-0 Matrices: A $n \times m$ matrix has rank 0 if all it's entries are 0.

Rank-1 Matrices: A $n \times m$ matrix A has rank 1 if it can be written as an outer product of a $n \times 1$ matrix \mathbf{u} and a $m \times 1$ matrix \mathbf{v} , i.e. $A = \mathbf{u}\mathbf{v}^T$

$$A = \mathbf{u}\mathbf{v}^T = \begin{bmatrix} | \\ \mathbf{u} \\ | \end{bmatrix} \begin{bmatrix} \text{---} & \mathbf{v}^T & \text{---} \end{bmatrix} = \begin{bmatrix} | & | & \dots & | \\ v_1\mathbf{u} & v_2\mathbf{u} & \dots & v_m\mathbf{u} \\ | & | & \dots & | \end{bmatrix} = \begin{bmatrix} \text{---} & u_1\mathbf{v}^T & \text{---} \\ \text{---} & u_2\mathbf{v}^T & \text{---} \\ \vdots & \ddots & \vdots \\ \text{---} & u_n\mathbf{v}^T & \text{---} \end{bmatrix}$$

Notice that the j th column of A is just v_j/v_i times the i th column of A . Similarly all rows are linearly dependent, thus the row and column ranks are 1.

Rank-2 Matrices: A $n \times m$ matrix A has rank 2 if it is the (non-trivial) sum of two rank-1 matrices. i.e.

$$A = \mathbf{u}\mathbf{v}^T + \mathbf{w}\mathbf{x}^T$$

(By non-trivial we mean that that A is not rank-0 or rank-1).

$$A = \mathbf{u}\mathbf{v}^T + \mathbf{w}\mathbf{x}^T = \begin{bmatrix} | & | & \dots & | \\ v_1\mathbf{u} + x_1\mathbf{w} & v_2\mathbf{u} + x_2\mathbf{w} & \dots & v_m\mathbf{u} + x_m\mathbf{w} \\ | & | & \dots & | \end{bmatrix} = \begin{bmatrix} \text{---} & u_1\mathbf{v}^T + w_1\mathbf{x}^T & \text{---} \\ \text{---} & u_2\mathbf{v}^T + w_2\mathbf{x}^T & \text{---} \\ \vdots & \ddots & \vdots \\ \text{---} & u_n\mathbf{v}^T + w_n\mathbf{x}^T & \text{---} \end{bmatrix} = \begin{bmatrix} | & | \\ \mathbf{u} & \mathbf{w} \\ | & | \end{bmatrix} \begin{bmatrix} \text{---} & \mathbf{v}^T & \text{---} \\ \text{---} & \mathbf{x}^T & \text{---} \end{bmatrix}$$

Rank-k Matrices A $n \times m$ matrix A has rank k if it is the sum of k rank-1 matrices and cannot be written as a sum of $k - 1$ or fewer rank-1 matrices.

1.1 Rank Factorization of a matrix

The above definition of is in terms of sum of matrices. Rephrased in terms of matrix multiplication, an equivalent definition is that A can be written as, or “factored into” the product of a

long and skinny ($n \times k$) matrix U (whose columns are the columns of rank-1 factors \mathbf{u}_i 's) and a short and long ($k < n$) matrix V^T (whose rows are the rows of the rank-1 factors \mathbf{v}_i 's). And that A cannot be likewise factored into the product $n \times (k - 1)$ and $(k - 1) \times m$ matrices.

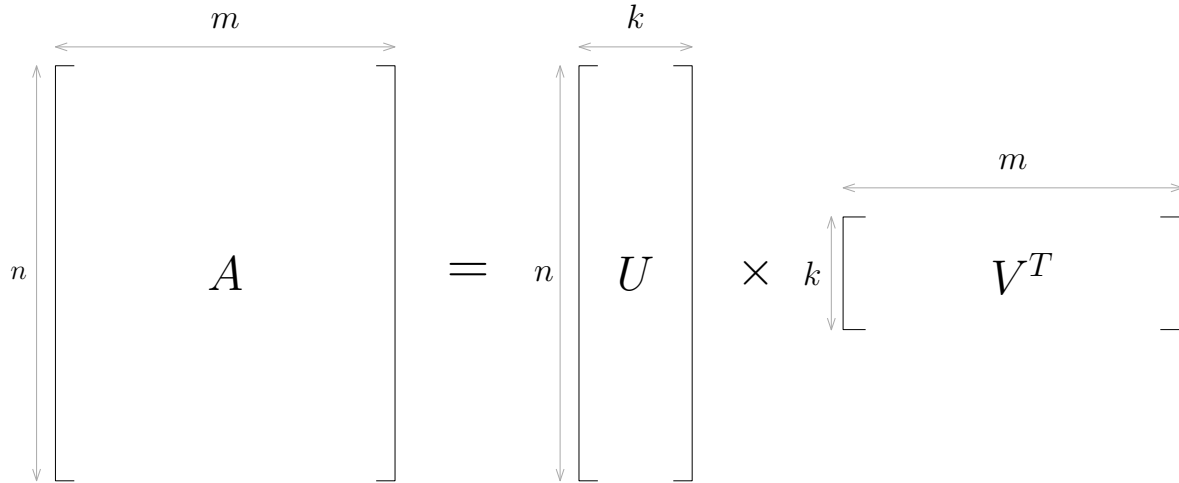


Figure 1: Rank factorization of a matrix

1.1.1 Equivalence between these definitions

All these definitions are equivalent. Any condition implies the other two in terms of row-rank and col-rank. It is clear that each condition implies the other two

if $A = UV^T$, then all of A 's columns are linear combinations of the k columns of U , and all of A 's rows are linear combinations of the k rows of V^T . The proofs are straight-forward.

1.1.2 Singular and non-singular matrices

If $n \times n$ matrix A has rank n , we say it is a “full rank” matrix. It uniquely maps $n \times 1$ vectors to $n \times 1$ vectors (thus it is a bijection) we say the matrix is invertible. If $rank(A) < n$, then A is called a singular matrix (rank deficient matrix). The resulting dimensionality is $\leq n - 1$, we cannot get pre-images from images and A is not invertible. There cannot be any inverse for non-square matrices.

2 Low Rank Approximation

Suppose our data matrix A is a $n \times m$ matrix (each row is a data point and each column is a feature). If all data points $\mathbf{x}_1, \dots, \mathbf{x}_n \in X$ can be represented as a linear combination of some other k basis vectors $\mathbf{v}_1, \dots, \mathbf{v}_k$ (rather than the given features), i.e. $\mathbf{x}_i = \sum_{j=1}^k c_{ij} \mathbf{v}_j$, where the coefficients, u_{ij} are lengths of projections of \mathbf{x}_i on \mathbf{v}_j . This means we can write A as $A = UV^T$. Geometrically, this mean that all data lie in a k -dimensional subspace (spanned by $\mathbf{v}_1, \dots, \mathbf{v}_k$).

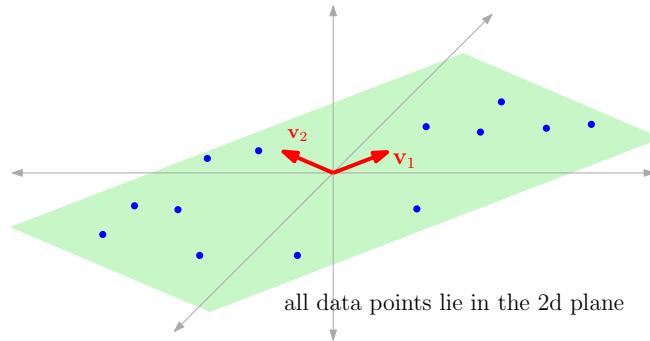


Figure 2: Data points lie in 2d plane

In this case we can just store the matrix U of coefficients and the matrix V of with basis vectors $\mathbf{v}_1, \dots, \mathbf{v}_k$ as columns. Each row of U can be thought of a reduced dimensional version of rows in A

Generally, we might not be able to get all the data lying in a k dimensional subspace, but what if data is ‘close by’ a low dimensional subspace (see below on why should we expect this). That is datasets may be approximately low ranks, i.e. while we may not get $A = UV^T$ but we would like to find U and V such that $A \simeq UV^T$

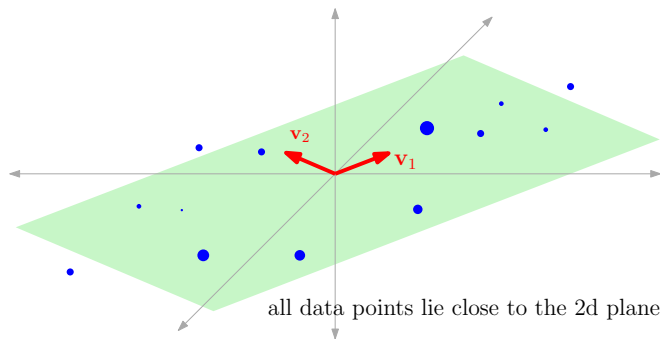


Figure 3: Data points lie close to the 2d plane

The primary goal of this lecture is to identify the “best” way to approximate a given matrix A with a rank- k matrix, for a target rank k . Such a matrix is called a low-rank approximation.

Naturally, we need a measure of goodness of a low-rank approximation UV^T for a data matrix A . The most natural and easy to deal with error measure is the sum of squared errors i.e.

$$\sum_{i=1}^n \|\mathbf{x}_i - \sum_{j=1}^k u_{ij} \mathbf{v}_j\|^2 =: \|A - UV^T\|_F^2$$

Where for a matrix M , $\|M\|_F = \sqrt{\sum_{i,j} M_{ij}^2}$ is called the Frobenius norm of a M . Note that many other error measures (norms of the error matrix) can be used, while the Frobenius norm is the most popular, spectral norm is also quite common.

Formally, the optimization problem of finding the best low-rank approximation w.r.t to Frobenius norm error measure is as follows

$$\arg \min_{V \in \mathbb{R}^{k \times m}, U \in \mathbb{R}^{n \times k}} \|A - UV^T\|_F^2$$

This looks like a very difficult problem (it is non-linear and not convex) but can nonetheless be solved surprisingly efficiently. We will discuss the power iteration method for this.

2.1 Why Low Rank Approximability is Expected

I am going to present two toy examples these datasets are made up but are very realistic and based on slight adjustments to real datasets.

Housing Data Example Consider the following data about for sale houses

ID	Beds	Baths	Living sq-ft	Lot sq-ft	Floors	Garage Cars	List Price	Sale Price
1	1	1	870	1100	1	0	31630	31544
2	1	1	1080	1400	1	0	35920	35916
3	2	1	1250	1500	1	0	48250	48025
4	2	1	1285	1550	1	0	48965	48738
5	2	2	1460	1800	2	1	67540	67633
6	3	2	1560	1800	1	0	68440	68763
7	3	2	1630	1900	2	1	79870	79533
8	3	2	2050	2500	2	1	88450	88054
9	3	2.5	2120	2600	2	2	102380	102576
10	4	2	2360	2800	2	1	103640	103892
11	4	2.5	2500	3000	2	1	109000	109523
12	4	2.5	2570	3100	2	1	110430	110393
13	4	3	2710	3300	3	2	125790	125945
14	5	2	2880	3400	2	2	133120	133503
15	5	2.5	2880	3400	3	2	135620	136124
16	5	2.5	3300	4000	3	2	144200	144365
17	5	3	3650	4500	3	2	153850	154444
18	5	3	3720	4600	3	3	165280	165439

But the rank of this matrix is not really eight (the features describing it). There are linear dependencies among features. Note that there may be many more non-linear dependencies.

$$\text{List-Price} = 10k \times \text{bed} + 5k \times \text{baths} + 9 \times \text{liv-sqFT} + 8 \times \text{Lot} + 10k \times \text{Cars}$$


ID	Beds	Baths	Living sq-ft	Lot sq-ft	Floors	Garage Cars	List Price	Sale Price
1	1	1	870	1100	1	0	31630	31544
2	1	1	1080	1400	1	0	35920	35916
3	2	1	1250	1500	1	0	48250	48025
4	2	1	1285	1550	1	0	48965	48738
5	2	2	1460	1800	2	1	67540	67633
6	3	2	1560	1800	1	0	68440	68763
7	3	2	1630	1900	2	1	79870	79533
8	3	2	2050	2500	2	1	88450	88054
9	3	2.5	2120	2600	2	2	102380	102576
10	4	2	2360	2800	2	1	103640	103892
11	4	2.5	2500	3000	2	1	109000	109523
12	4	2.5	2570	3100	2	1	110430	110393
13	4	3	2710	3300	3	2	125790	125945
14	5	2	2880	3400	2	2	133120	133503
15	5	2.5	2880	3400	3	2	135620	136124
16	5	2.5	3300	4000	3	2	144200	144365
17	5	3	3650	4500	3	2	153850	154444
18	5	3	3720	4600	3	3	165280	165439

$$\text{Sale Price} = (1 \pm 0.02) \times \text{List Price}$$

Since I made up this data, it is not very hard to see that Living Sq-ft feature can be derived as a linear combination of Lot sq-ft and Beds feature.

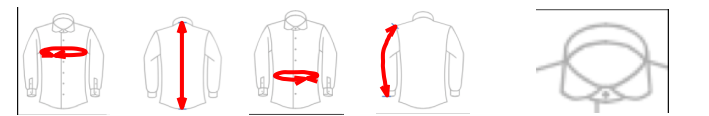
Shirt Dimension Example When you buy a shirt, there are many measurements involved, chest and waist circumferences, sleeve and back lengths. One can easily think of many more. But generally, in the market shirts are marked with collar measurement only and it fits reasonably well in most cases. Here is a made up data, (the data is quite realistic only minor adjustments are made to a real dataset to make the formula simpler)

The collar feature is a new dimension (linear combination of other features). So the data actually lies in a one dimensional space and true rank of the data is 1.



Chest	Back	Waist	Sleeve
104	81	98	67
107	81	100	67
110	82	102	67
113	82	104	67
116	83	106	68
120	83	110	68
124	84	114	68
128	84	118	68
132	85	122	68
136	85	126	68

$$\text{Collar} = 0.44 \times \text{Chest} + 0.015 \times \text{Back} - 0.2 \times \text{Waist} + 0.153 \times \text{Sleeve}$$



Chest	Back	Waist	Sleeve	Collar
104	81	98	67	37
107	81	100	67	38
110	82	102	67	39
113	82	104	67	40
116	83	106	68	41
120	83	110	68	42
124	84	114	68	43
128	84	118	68	44
132	85	122	68	45
136	85	126	68	46

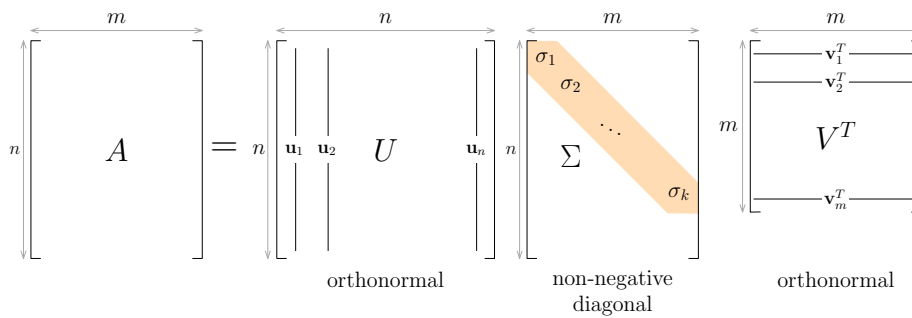
3 SVD Definition

Our high-level plan for computing a rank- k approximation of a matrix A is: (i) express A as a list of its ingredients, ordered by “importance” (ii) keep only the k most important ingredients. The

non-trivial step (i) is made easy by the singular value decomposition, a general matrix operation discussed in the next section.

Theorem 1. Any $n \times m$ matrix can be written as a product of three matrices, $A = U\Sigma V^T$

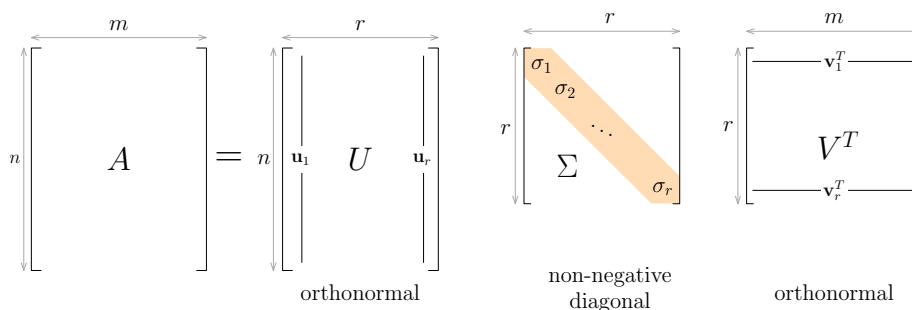
- U is a $n \times n$ orthogonal matrix (columns are orthonormal)
- V is a $m \times m$ orthogonal matrix
- Σ is a $n \times m$ diagonal matrix, with non-negative entries and entries at the main diagonal are sorted from highest value to lowest



The SVD theorem is depicted in this diagram. Another version sometimes called compact SVD is used often (as we will do). You should have asked this question, that what if $rank(A)$ is much smaller than $\min\{m, n\}$, then how can we have n or m orthonormal columns and rows in U and V^T , respectively. Well the answer is that we never said they only have real vectors. Anyway, the compact SVD version addresses this issues.

Theorem 2. Any $n \times m$ matrix with $rank\ r \leq \min\{m, n\}$ can be written as a product of three matrices, $A = U\Sigma V^T$

- U is a $n \times r$ orthogonal matrix (columns are orthonormal)
- V is a $r \times r$ orthogonal matrix
- Σ is a $r \times m$ diagonal matrix, with non-negative entries and entries at the main diagonal are sorted from highest value to lowest



- Columns of U are called left singular vectors
- Columns of V are called right singular vectors (note that these are rows of V^T)
- Diagonal entries of Σ are called singular values

The proof of these theorems are not very deep and covered in any standard linear algebra text. We discuss what does it mean geometrically. We know that an $n \times m$ matrix maps vectors in \mathbb{R}^m to vectors in \mathbb{R}^n (in a linear fashion, $\mathbf{0}$ is mapped to $\mathbf{0}$ and straight lines are mapped to straight lines). As we discussed with many examples that linear transformations can achieve many things (scaling, rotation, reflections, shear). The SVD theorem says that no matter how weird the linear transform A looks like it only performs a rotation, followed by scaling, followed by another rotation. Note that orthogonal matrices achieve rotations (and reflections and permutations we noted this in the lecture on PCA too). While diagonal matrices do only scaling.

Since by SVD, $A = U\Sigma V^T$, where U and V have orthonormal columns and rows, respectively, and Σ is a diagonal matrix. The SVD says that we can replace any transformation by a rotation (or reflection) from “input” coordinates into convenient coordinates, followed by a simple scaling operation, followed by a rotation into “output” coordinates. Furthermore, the diagonal scaling Σ comes out with its elements sorted in decreasing order.

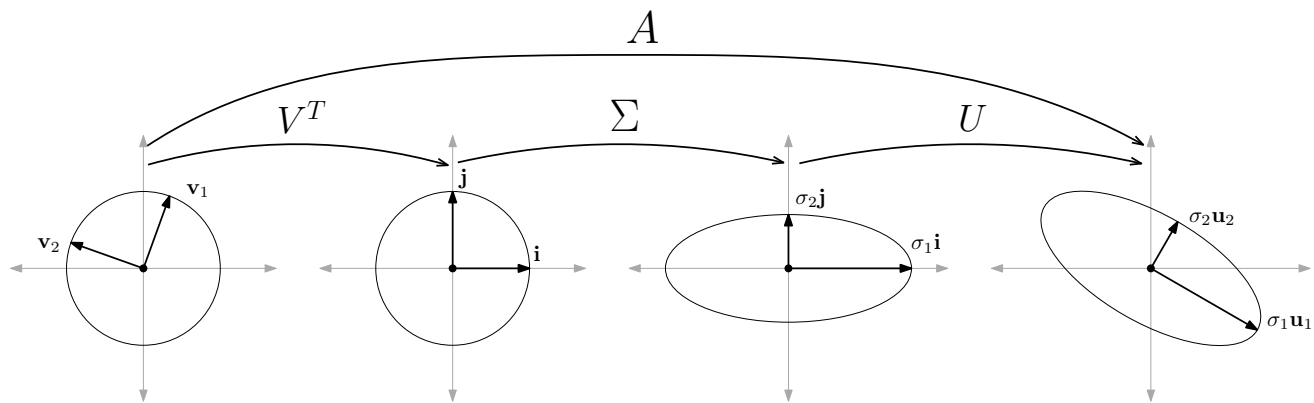


Figure 4: Any transformation is a rotation followed by scaling followed by a rotation

If we take the unit circle and transform it by A , we get an ellipse (because A is a linear transformation). The left singular vectors $\mathbf{u}_1, \mathbf{u}_2$ are the major and minor axes of that ellipse (being on the left they live in the “output” space). The right singular vectors $\mathbf{v}_1, \mathbf{v}_2$ are the vectors that get mapped to the major and minor axes (being on the right they live in the “input” space). If we break the transformation down into these three stages we see a circle being rotated to align the \mathbf{v} ’s with the coordinate axes, then scaled along those axes, then rotated to align the ellipse with the \mathbf{u} ’s:

4 Low Rank Approximation from SVD (Truncated SVD)

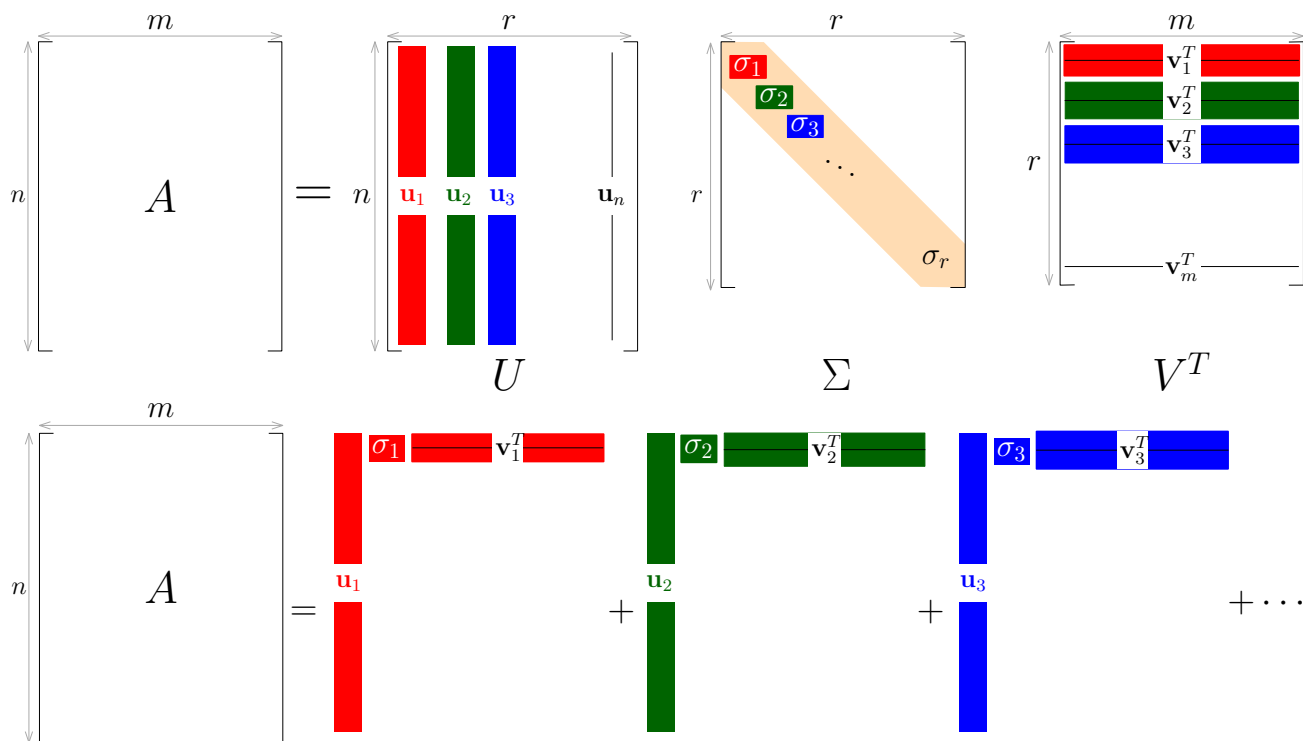
The SVD theorem says that no matter how weird the linear transform A looks like it only performs a rotation, followed by scaling, followed by another rotation. In other words A has an equivalent representation as the product of 3 simple matrices two orthogonal spans and a diagonal scaling. One of the most useful application of SVD (and the one most relevant to us) is that the factorization of A into $U\Sigma V^T$ can be used to get the best k -rank approximation of A in terms of Frobenius norm, for any k .

4.1 Spectral decomposition of a matrix - Decomposition into rank-1 matrices

SVD expresses the $n \times m$ matrix A as a linear combination of $\min\{m, n\}$ rank-1 matrices, where the coefficients are the singular values and the rank-1 matrices are the outer product of the corresponding left and right singular vectors.

Let $A = U\Sigma V^T$, then

$$A = \sum_{\ell=1}^{\min\{m,n\}} \sigma_{\ell} \mathbf{u}_{\ell} \circ \mathbf{v}_{\ell}^T$$



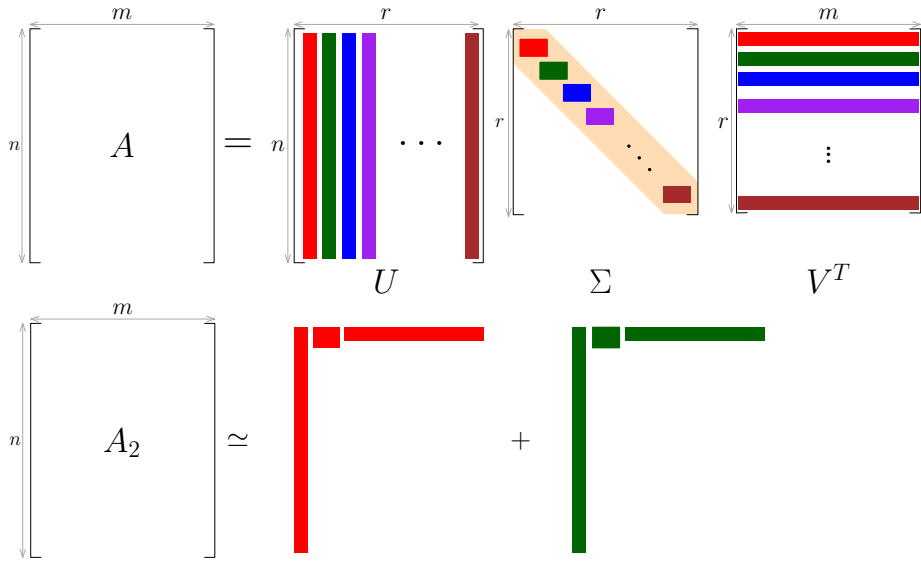
We picture this view as writing A as a sum of its ingredients in decreasing importance of ingredients. The truncated SVD keeps the top k important ingredients and drop the remaining.

4.2 Truncated SVD

Define A_k to be the sum of the first k terms in the SVD decomposition of A , i.e.

$$A_k = \sum_{\ell=1}^k \sigma_\ell \mathbf{u}_\ell \circ \mathbf{v}_\ell^T + \sum_{\ell=k+1}^r \sigma_\ell \mathbf{u}_\ell \circ \mathbf{v}_\ell^T$$

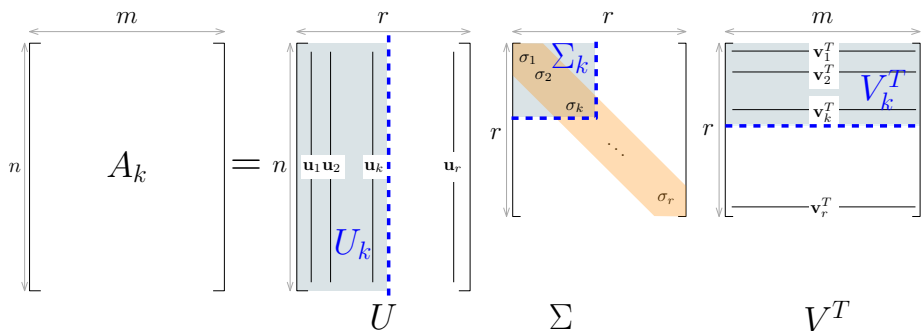
$$A_k = \sum_{\ell=1}^k \sigma_\ell \mathbf{u}_\ell \circ \mathbf{v}_\ell^T + \cancel{\sum_{\ell=k+1}^r \sigma_\ell \mathbf{u}_\ell \circ \mathbf{v}_\ell^T}$$



This only corresponds to setting the last $r - k$ singular values σ_{k+1} to σ_r to 0.

For $1 \leq k \leq r$, let $U_k \in \mathbb{R}^{n \times k}$ contain the first k left singular vectors (the first k columns of U), $\Sigma_k \in \mathbb{R}^{k \times k}$ be a diagonal matrix containing the first k singular values (the top-left $k \times k$ principal submatrix of Σ), and V_k^T be the first k right singular vectors (the first k rows of V^T). Let $A_k = U_k \Sigma_k V_k^T$.

One can easily prove that A_k is the best rank- k approximation to A , i.e. it is the solution to the above optimization problem.



4.2.1 How to choose k

What is the right number of singular values and vectors that we should keep and discard the remaining. This is a similar question that we tackled in choosing the right number of clusters, and then in choosing the right number of principal components.

If k is not given as part of input, then we select k by looking at singular values. We can choose k (as the elbow-point) a point after which the remaining singular values are significantly smaller than the preceding ones. Alternately, we can select the first k singular values such that sum of singular values (or there squares) is at least a certain fraction of total sum. This threshold can be chosen with trial and error or can be domain-dependent. Note that the sum of squares of a signal (and array) is called the energy of the signal. Thus a threshold can be that choose k such that $\sum_{i=1}^k \sigma_i^2 \sim 0.85 (\sum_{i=1}^r \sigma_i^2)$, i.e. that is choose k so as 85% of the energy in singular values is retained.

5 Application of Low Rank Approximation

5.1 Data Compression

We use SVD as a data dependent linear dimensionality reduction method. It provides in depth insights into the data and is applicable in a wide range of field. Just like PCA it is also linear, so especially when there is reason that original features are related in a linear fashion SVD provides a very good dimensionality reduction.

A low-rank approximation provides a (lossy) compressed version of the matrix. The original matrix A is described by nm numbers, while describing U_k and V_k^T requires only $kn + km = k(n + m)$ numbers. When k is significantly smaller than n and m , $k(n + m) \ll nm$. For example, when A represents an image (with entries = pixel intensities), m and n are typically in the 100s. In other applications, m and n might well be in the tens of thousands or more. With images, a modest value of k (say 100 or 150) is usually enough to achieve approximations that look a lot like the original image. Thus low-rank approximations are a matrix analog to notions of dimensionality reduction for vectors.

5.2 Recommender System

A natural interpretation of the truncated SVD of of a data matrix A that of approximating A in terms of k “concepts”. The singular vectors are a numeric representations of rows and columns of the concepts, while Σ_k measure the strength of these concepts.

The i th row of U represents the i th data item (i th row of A) as a linear combination of the rows of concepts (with coefficients Σ). The j th column of V^T represents the j th dimension (j th columns of A) as a linear combination of the columns of concepts (with coefficient Σ).

We give an example for recommendation systems. Recall the recommendation system problem.

	p_1	p_2	p_3	p_j					p_m				
u_1	1		2	1	4		2	3	2	5			2
u_2		1			2	1		2		1			3
u_3	1	1	2			1					1	2	
			3	2			5		2		3	4	
	1			2						5			
u_i		3	2	1	4	5	?	1	3	1	2		1
		4										4	
		5		1								5	
	1		4					1	3	5	1	2	
u_n		3		1	1	2	1			4			5

In the matrix factorization approach to recommender, factorizes the $n \times m$ rating matrix R as $R = PQ$ by solving the following optimization problem

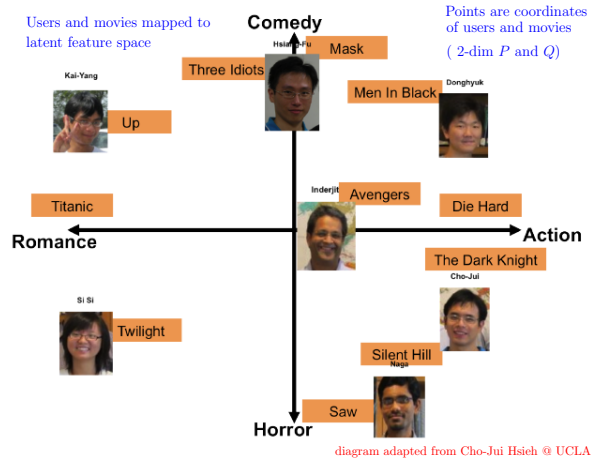
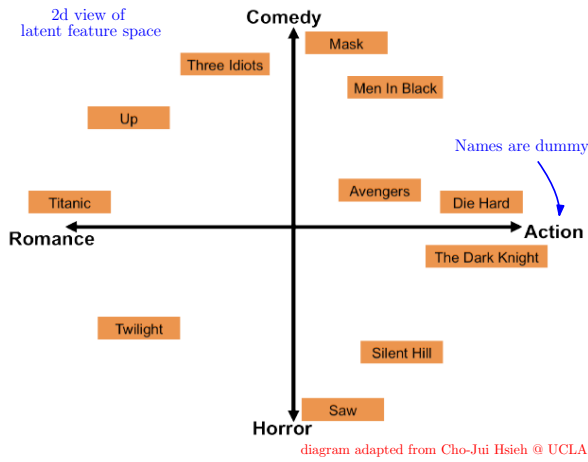
$$\min_{\substack{P \in \mathbb{R}^{n \times k} \\ Q \in \mathbb{R}^{m \times k}}} \sum_{(i,j)} (R_{ij} - P_i Q_j^T)^2 + \lambda (\|P\|_F^2 + \|Q\|_F^2)$$

In this setting the matrix P is a k -dim representation of users (i th row is user i in a latent/hidden feature space \mathbb{R}^k) and the matrix Q is the k -dim representation of movies, (i th row of P is the latent feature space representation of movie i , the i th column of R). $P_i Q_j^T$ is the interaction between user i and movie j , the approximation of R_{ij} . Thus we aim to find the best k dimensional feature representation for users and movies.

The second term in the optimization objective is the so-called regularization term, with the weight λ it keeps the value of P and Q from growing too large in order to avoid over-fitting.

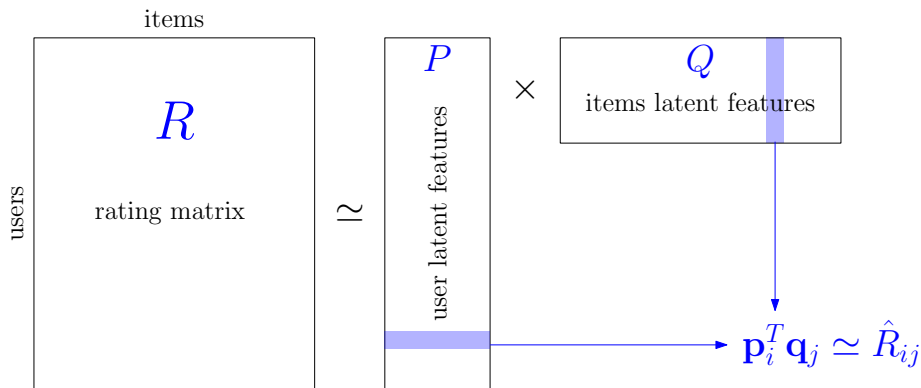
	p_1	p_2	p_3	p_j					p_m				
u_1	1		2	1	4		2	3	2	5			2
u_2		1			2	1		2		1			3
u_3	1	1	2			1					1	2	
			3	2			5		2		3	4	
	1			2						5			
u_i		3	2	1	4	5	?	1	3	1	2		1
		4										4	
		5		1								5	
	1		4					1	3	5	1	2	
u_n		3		1	1	2	1			4			5

A dummy example is as follows:

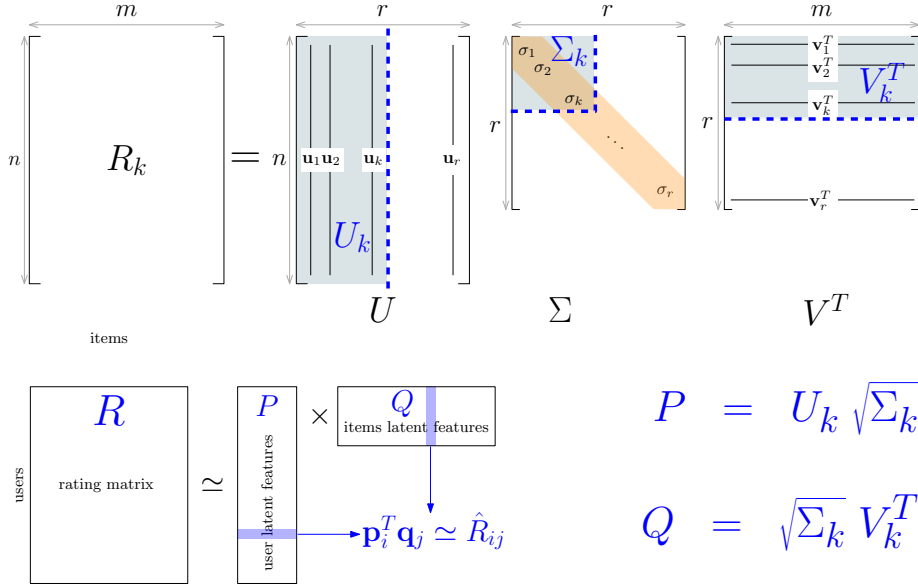


5.2.1 Using SVD to factorize the rating matrix

A schematic view of matrix factorization method for recommenders is given in the following diagram.



We can use the SVD to factor the rating matrix R (one method is given in the following diagram)



However, the main limitation of this approach to factorize R is that SVD has to use certain value for the missing entries (which we expect to be numerous, as rating matrix generally are very sparse.) Thus it is not very hard to see that SVD will tend to find embeddings for users and movies to better fit the empty cells in the matrix. As in the optimization objective the majority of errors will be due to the empty cells, thus to minimize the error SVD would tend to give $\mathbf{p}_i^T \mathbf{q}_j$ equal to 0 (if empty cells were filled with 0's). Thus the approximation on the non-zero entries (known ratings) is not going to be good. One can try other default values that could even be data dependent such as matrix averages, row averages, column averages or with the ANOVA. This approach works well when the matrix is close to a rank- k matrix and there are not too many missing entries.

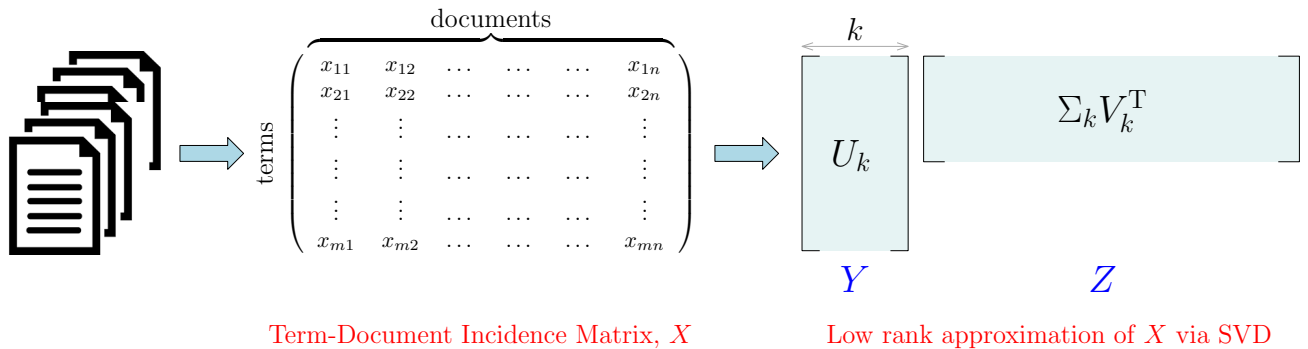
However, my intention was to introduce to matrix factorization (in particular for recommenders) and that SVD can be used for it. It will work provably better for the general problem of matrix completion if only a few entries in the table are missing.

5.3 Latent Semantic Analysis and Word Embeddings

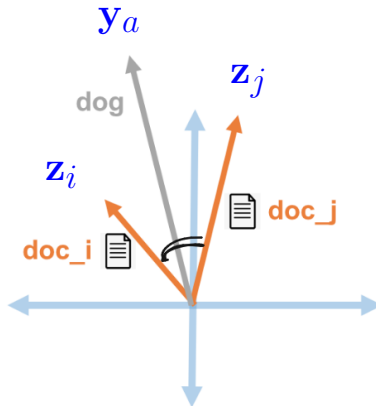
A classic use of SVD is Latent Semantic Analysis, which is the basis of Word Embeddings and Latent Semantic Indexing (when used in information retrieval). [1].

Recall our discussion on vector space modeling of text where we represented text documents by vectors (set of words, bag of words or tf-idf vectors.). We discussed word2vec [2, 3] where words are embedded into high dimensional Euclidean space such that words appearing in similar context are likely to be close by (in terms of ℓ_2 distance).

Note that YZ^T is the best low rank approximation of X in terms of the Frobenius norm i.e. $\|X - YZ^T\|_F$ is the smallest (amongst rank k matrices). This implies that on average we expect every entry X_{ij} of X to be close to $\mathbf{y}_a \mathbf{z}_i^T$. In other words $\mathbf{y}_a \mathbf{z}_i^T \simeq 1$ when doc_i contains $term_a$. Here \mathbf{y}_a and \mathbf{z}_i refer to the a th row of Y and i th columns of Z^T , respectively.



If two documents doc_i and doc_j both contains $term_a$, then $\mathbf{y}_a \mathbf{z}_i^T \simeq \mathbf{y}_a \mathbf{z}_j^T \simeq 1$. This basically means if two documents doc_i and doc_j contain the same word, then the corresponding vectors \mathbf{z}_i and \mathbf{z}_j both have high dot product with the vector \mathbf{y}_a (low angle or cosine distance).



In general, if two documents doc_i and doc_j contain many of the same terms, they will have very small angle between them (high dot-product). Similarly, if two terms appear in many of the same documents, then the corresponding vectors, rows \mathbf{y}_a and \mathbf{y}_b of Y will tend to have higher dot products.

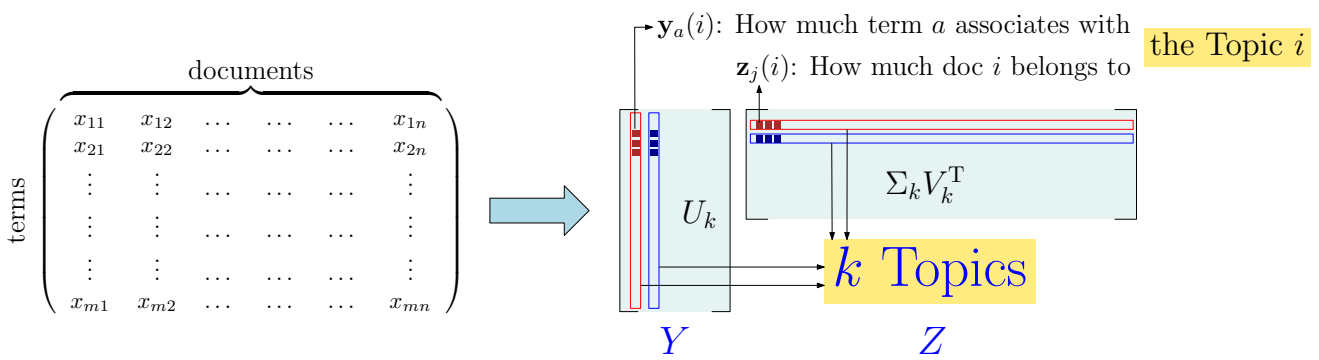
One can look at it as follows: The matrices Y and Z represent k abstract concepts (latent factors)- the a th row of Y represent the term a as linear combination of the k concepts. Similarly, each column of Z^T represent the corresponding document as linear combination of the k concepts.

Latent Semantic Analysis (LSA) gives a way to embed terms (in the corpus) into the k -dimensional space, where rows of Y are the representation.

Note that as we discuss later (see below), SVD of X gives us eigendecomposition of XX^T as

$$XX^T = U\Sigma^2U^T$$

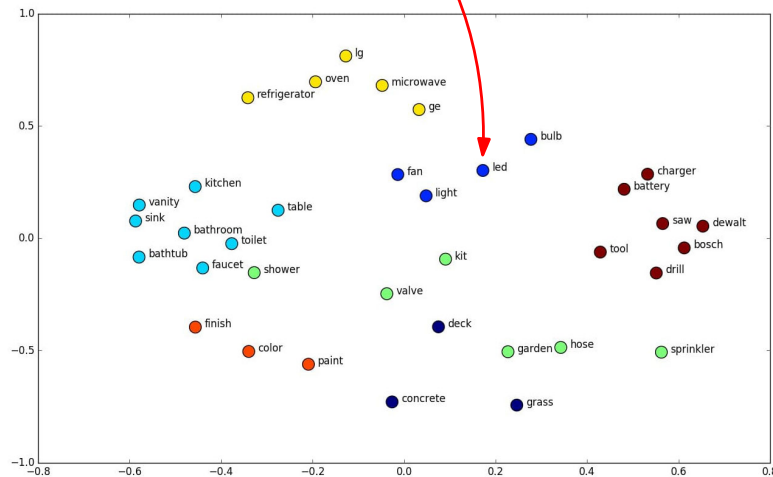
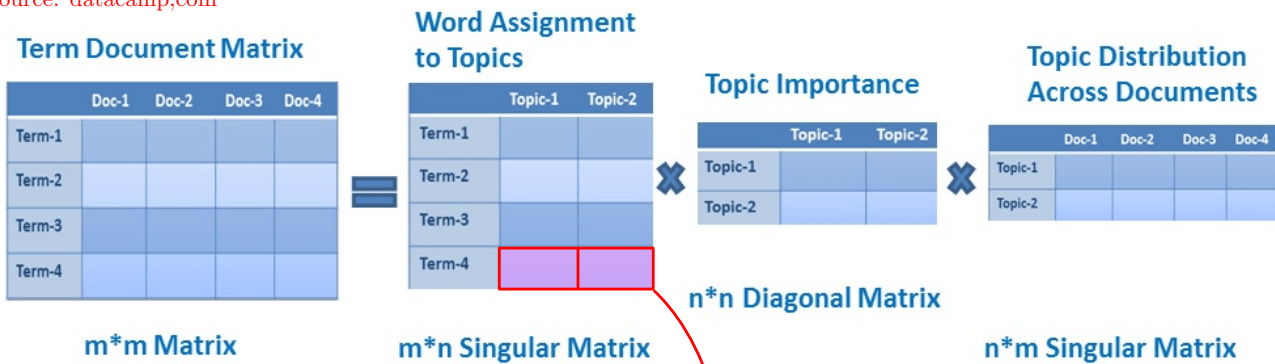
In our setting XX^T is the term co-occurrence matrix (if X is the term document incidence matrix.) If X is the (normalized) term-document frequency matrix (each column is bag of words representation of a document), then XX^T is the correlation matrix between terms. Columns of U are the eigenvectors of XX^T .



Term-Document Incidence Matrix, X

Low rank approximation of X via SVD

source: datacamp.com



The matrix XX^T is somewhat of a matrix of pairwise similarity between terms (called gram matrix or kernel matrix). Substituting some definition of similarity measure between terms XX^T with the corresponding similarity matrix, we get the different word embeddings such as WORD2VEC, GLOVE etc.

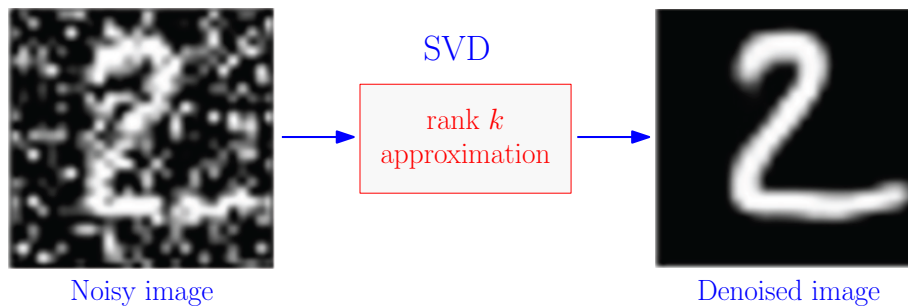
Another interesting application of SVD is to represent electricity consumers based on their hourly consumption in the training dataset. In this setting given consumer load data as a matrix,

where each row is a consumer and each column is a time-period (e.g. an hour). We perform truncated SVD on this load matrix and use rows of $U\Sigma$ as a low dimensional representation of each consumer. This succinct load data-driven representation of consumers (and time periods) is used for customer segmentation, identifying similar time periods by load patterns to predict future consumers load.

5.4 Denoising

Suppose the matrix A represents some data with noise added to it (e.g. images with noise). If the true underlying data is actually of low rank, then a low-rank approximation via SVD of the A might throw out a significant amount of noise and very little ground truth data (the signal).

Thus, the resulting approximate data might be a cleaner, more informative and better version of the given data. This will in particular help, if the singular values exhibit a good elbow structure, since smaller singular values will more likely correspond to the added noise in data.



6 Relation of SVD and eigen-decomposition

SVD and eigen-decomposition are related but there are quite a few differences between them.

- Not every matrix has an eigen-decomposition (not even every square matrix). Any matrix (even rectangular) has an SVD.
- In eigen-decomposition $A = X\Lambda X^{-1}$, that is, the eigen-basis is not always orthogonal. The basis of singular vectors is always orthogonal
- In SVD we have two singular-spaces (right and left)
- Computing the SVD of a matrix is more numerically stable

For $n \times m$ matrix SVD gives us

$$A = U\Sigma V^T$$

While eigendecomposition, when possible gives us

$$A = X\Lambda X^{-1}$$

In this case we must have the following

- A is real symmetric matrix (we used covariance matrix only), otherwise eigendecomposition may not exist
- U, V, X are orthonormal matrices
- Λ and Σ are diagonal matrices with values in decreasing orders (respectively eigenvalues and singular values)
- U and V are respectively the left and right singular matrices of A
- X are eigenvectors of A

We can compute AA^T using SVD of A as follows.

$$AA^T = (U\Sigma V^T)(U\Sigma V^T)^T = U\Sigma V^T ((V^T)^T \Sigma^T U^T) = U\Sigma V^T V \Sigma U^T = U\Sigma^2 U^T$$

Thus, U is the eigenvectors of the covariance matrix of the data $A^T A$, notice that we get it directly from $A = U\Sigma V^T$, without having to explicitly compute the covariance matrix. If dataset is large and dimensionality is also large, computing the covariance matrix is already computationally expensive. Thus, SVD provides another way of computing the principal components.

Recall that principal components are eigenvectors of $A^T A$ (if A is the data matrix with each data point as a row and each dimension as a column). Eigenvalues are just the square roots of the singular values.

7 Computing the SVD: The power method

There are pretty good algorithms for computing the SVD of a matrix; details are covered in any numerical analysis course. The running time of the algorithm is the smaller of $O(n^2m)$ and $O(m^2n)$, and the standard implementations of it have been heavily optimized. If you just want to compute the top k singular values and their associated singular vectors (which is generally what we want for low rank approximation), this can be computed significantly faster, in time roughly $O(kmn)$ using the power iteration method.

References

- [1] Susan T. Dumais. Latent semantic analysis. *Annual Review of Information Science and Technology*, 38(1):188–230, 2004.
- [2] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. Efficient estimation of word representations in vector space. In Yoshua Bengio and Yann LeCun, editors, *1st International Conference on Learning Representations, ICLR 2013, Scottsdale, Arizona, USA, May 2-4, 2013, Workshop Track Proceedings*, 2013.

- [3] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg Corrado, and Jeffrey Dean. Distributed representations of words and phrases and their compositionality. In *Proceedings of the 26th International Conference on Neural Information Processing Systems - Volume 2*, NIPS'13, page 3111–3119. Curran Associates Inc., 2013.