PRINCETON UNIV. F'18 COS 521: ADVANCED ALGORITHM DESIGN

Lecture 13: Low Rank Approximation and the Singular Value Decomposition

Lecturer: *Christopher Musco*

This unit is about finding compressed data representations – e.g. compressed representations of vectors $a_1, a_2, \ldots a_n \in \mathbb{R}^d$. The techniques we have discussed so far (Johnson-Lindenstrauss sketching and hashing for similarity search) are *oblivious* to any structure in the dataset. They compress each point a_i without looking at any other points. In some ways, this obliviousness is a strength. It makes the methods fast and, as we saw in the previous two lectures, we can obtain strong guarantees without making assumptions about our data.

At the same time, obliviousness can be a weakness. Oblivious dimensionality reduction methods don't take advantage of structure which might make better compression possible. Moreover, finding such structure might be interesting in its own right.

1 Low-rank structure

Today we will discuss a type of structure that can 1) allow for better dimensionality reduction, 2) lead to very interesting scientific discoveries and insights about data, and 3) is remarkably common across diverse application areas.

In particular, we are interested in datasets where most of our vectors $a_1, \ldots, a_n$ can be well approximated as a linear combination of a small ground set of vectors in $\mathbb{R}^d$, $\{b_1, \ldots, b_k\}$. I.e. for some set of k coefficients $\{C_{1i}, C_{2i}, \ldots, C_{ki}\}$ we approximate a_i by:

$$a_i \approx \sum_{j=1}^{k} C_{ji} b_j.$$

Let $A \in \mathbb{R}^{d \times n}$ be a *data matrix* which contains each a_i as a column. If A is a rank k matrix (i.e. a "low-rank" matrix), then it would be possible to find some ground set and coefficients so that, for all i, this is actually an equality: $a_i = \sum_{j=1}^{k} C_{ji} b_j$. Geometrically, this would mean that all points lie on a low dimensional hyperplane in $\mathbb{R}^d$ (see Figure 1)

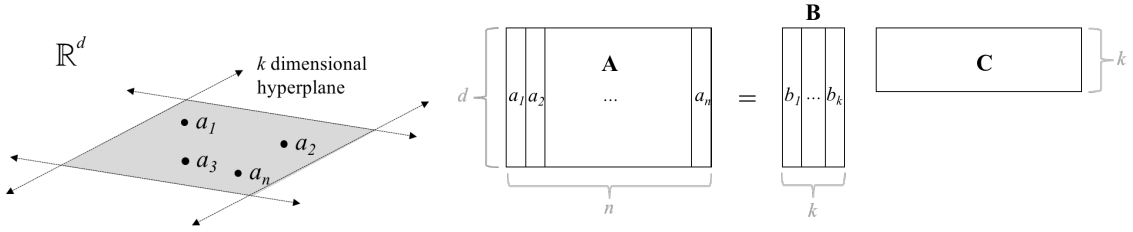

Figure 1: If $a_1, \ldots, a_n$ lie on a low dimensional hyperplane, then $A = [a_1, \ldots, a_n]$ is exactly low rank, so it can be written as the product of two matrices, $B \in \mathbb{R}^{d \times k}$ and $C \in \mathbb{R}^{k \times n}$

We can view the columns of $C = [c_1, \ldots, c_n]$, which are each in $\mathbb{R}^k$, as dimensionality reductions of $a_1, \ldots, a_n$. In the case when A is low-rank, consider choosing $b_1, \ldots, b_k$ to be an orthogonal span for the hyperplane containing $a_1, \ldots, a_n$. Then each c_i is simply a_i represented in a coordinate basis for the hyperplane. So, for example, $\|c_i\|_2 = \|a_i\|_2$ for all i, $\|c_i - c_j\|_2 = \|a_i - a_j\|_2$ for all i, j, and in general, $[c_1, \ldots, c_n]$ captures all the geometric information about our original dataset.

Low-rank approximation

Now of course, it's very rare to find matrices that are *actually* low rank. On the other hand, it's very common to find matrices that are *approximately* low rank. I.e. where there exists some set of ground set of vectors $\{b_1, \ldots, b_k\}$ and some coefficients C_{ij} so that, for example, the following error measure is small:

$$\sum_{i=1}^{n} \left\| a_i - \sum_{j=1}^{k} c_{ij} b_j \right\|_2^2 = \|A - BC\|_F^2. \tag{1}$$

Here $\|\cdot\|_F$ denotes the Frobenius norm of a matrix (i.e. the square root of its sum of squared entries):

$$\|M\|_F = \sqrt{\sum_{i,j} M_{ij}^2}.$$

Using the Frobenius norm gives a convenient way of using matrices to express our error measure. There are other ways to measure if a matrix is "close to a low-rank matrix", but Frobenius norm distance is a popular measure and the one we will focus on today.

Computation

If A were exactly low-rank, BC could be found using any standard orthogonalization procedure for finding a span for A's columns. On the other, when A is only approximately low-rank, we ideally want to solve:

$$\min_{B \in \mathbb{R}^{d \times k}, C \in \mathbb{R}^{k \times n}} \|A - BC\|_F^2. \tag{2}$$

This is a non-linear, non-convex optimization problem, but surprisingly it can be solved very efficiently (in low polynomial time). We will discuss one particular method for doing so in this lecture.

2 Example Applications

Before getting into algorithms, let's see a few places where low-rank approximation is important in practice.

Latent Semantic Analysis (LSA)

Let's return to the bag-of-words model discussed in previous lectures. Here each a_i represents the i^{th} document in a database of n documents. The vector has an entry for all possible words in a given language. At that location, it contains a count for the number of times that word appeared in document i.

$A = [a_1, \ldots, a_n]$ is called the "term-document" matrix – each row corresponds to a term and each column a document. Term document matrices tend to be close to low-rank, and algorithms for low-rank approximation can recover that structure. In particular, we can find an approximation BC for the term-document matrix where B has k columns $b_1, \ldots, b_k \in \mathbb{R}^d$ (here d is the number of words in our language) and $\|A - BC\|_F^2$ is small.

There are a number of justifications for why this might be the case. For example, one simple generative model for documents would be to simply look at all of the words in a language, look at the empirical frequency with which those words occur (e.g. "the" is used much more frequently than "president") and then assume that a typical document is generated by drawing words at random according to their relative frequencies. Of course this isn't a very good model, but we can think about how to make it better.

For example, we might notice that documents about politics tend to use the word "president" much more frequently than those about sports. Moreover, political documents written in the US tend to use the word "president" more than political articles written in the UK. Similarly, sports articles written in the US tend to use "touchdown" more than sports articles written in the UK. We can construct a better model by assuming a document can be assigned to a mix of potentially overlapping categories – for example, a document about US environmental policy might have categories "politics","for US audience", and "environment". Would could predict the words contained in that document by looking at the global distribution of words used in the "politics", "for US audience", and "environment" categories, and drawing words at random according to a mixture of those distributions.

This model turns out to be very powerful in predicting the frequency of words in a document. Since it does so in a linear way (drawing from a linear combination of word distributions), we expect that A will have a good k-rank approximation. Instead of hand choosing categories, columns in B can be seen as representing a set of "optimal" categories.

This is the idea behind what's known as "latent semantic analysis" in natural language processing. Each column $c_i \in C$ is viewed as an indicator vector for the presence of different categories in document a_i. As a "semantic representation" for a document, c_i's can be used to compare or cluster documents. For a more in-depth discussion, check out Chapter 18 in [2] (freely available online) or [3].

Word embeddings

If columns in C give some sort of meaningful representation of words, what about rows in B? We have one row for each word in our language. These rows are sometimes called "word embeddings" – they can give very powerful semantic representations of words. Words that tend to appear in similar categories of documents (e.g. in a similar context) tend to have similar word embedding vectors. These embeddings can be used for finding similar words or synonyms, but also for tasks like solving analogy problems.

There are many ways to generate better word embeddings. For example, instead of looking at term-document co-occurrence, it's more common to look at more local measures, like term-sentence co-occurence. Current algorithms are also based on more complex models than the simple linear model discussed above (see e.g. https://nlp.stanford.edu/projects/glove/ or https://code.google.com/archive/p/word2vec/), but it all started with basic LSA and low-rank approximation!

Visualizing and understanding genetic data

Genetic data tends to be low-rank. Consider a vector a_i for each individual in a population that holds a numerical representation of that individual's genome (e.g. a bit vector, with every two bits representing the expression of a single nucleotide, which can take values A, T, G, or C.). Why might this be the case?

At a course level genetics are controlled largely by ancestry – historically isolated populations (geographically, culturally, etc.) tend to have very similar genomes, with the exception of a relatively small number of genes that distinguish individuals. Accordingly, if we let our set of ground vectors $b_1, \ldots, b_k$ contain representative individuals from different ancestral populations, we can do a pretty good job reconstructing every a_i vector up to small error. If we take the best low-rank approximation, we can do even better.

One particularly dramatic exposition of the low-rank natural of genetic data is given in [1]. After a few basic data transformations (mean centering, removing outliers, etc.) they took a rank-2 approximation of a genetic data set from populations in Europe. This produced a set of two dimensional vectors, $c_1, \ldots, c_n$ for each individual. When plotting these points on a two-dimensional grid, the location of each point roughly reflects the ancestral origin of each individual!

In other words, each a_i is well represented by a two dimensional linear model where each dimension represents the East/West and North/South coordinates of where that individual is from. In fact, since it was derived from an optimal rank-2 approximation, this is the best two dimensional linear model for reconstructing A, indicating how important geography is in genetic variation. I would encourage you to check out the paper (e.g. at https://www.researchgate.net/publication/23469983_Genes_Mirror_Geography_within_Europe).

3 The Singular Value Decomposition

It turns out that optimal low-rank approximations can be computed using what's known at the "singular value decomposition".

Theorem 1 (Singular Value Decomposition (SVD))**.** *Consider $A \in \mathbb{R}^{n \times d}$ and let $r = \min(d, n)$. A can always be written as the product of three matrices, $A = U\Sigma V^T$, where:*

- *$U \in \mathbb{R}^{n \times r}$ is a matrix with orthonormal columns,*
- *$\Sigma = \begin{bmatrix} \sigma_1 & & \\ & \ddots & \\ & & \sigma_r \end{bmatrix}$ is a non-negative diagonal matrix with entries $\sigma_1 \geq \ldots \geq \sigma_r \geq 0$,*

- $V \in \mathbb{R}^{d \times r}$ *is a matrix with orthonormal columns.*

U's columns are called the "left singular vectors" of A, V's columns are its "right singular vectors", and $\sigma_1, \ldots, \sigma_r$ are its "singular values". When the SVD is computed after mean centering A's columns or rows, the singular vectors are sometimes call "principal components".

The singular value decomposition is closely related to eigendecomposition: $U\Sigma^2 U^T$ is the eigendecomposition of AA^T and $V\Sigma^2 V^T$ is the eigendecomposition of $A^T A$. Using eigendecomposition algorithms (e.g. the QR algorithm), it can be computed in approximately $O(nd^2)$ time (assuming $d < n$).

The existence of the SVD may be surprising – it says that no matter what A looks like, it can be represented at the product of 3 simple matrices – two orthogonal spans and a diagonal scaling. The SVD has countless applications in linear algebra, but one of the most useful is that it can be used to read-off the optimal k-rank approximation for A, for any k.

Claim 2 (Truncated SVD)**.** *For any $k \in 1, \ldots, \min(n, d)$, let $U_k \in \mathbb{R}^{n \times k}$ contain the first k columns of U, let $V_k \in \mathbb{R}^{d \times k}$ contain the first k columns of V, and let Σ_k be a $k \times k$ diagonal matrix containing A's first singular values. Then:*

$$\|A - U_k \Sigma_k V_k^T\|_F^2 = \min_{B \in \mathbb{R}^{d \times k}, B \in \mathbb{R}^{k \times n}} \|A - BC\|_F^2.$$

In other words, there is no better rank k approximation for A than $U_k \Sigma_k V_k^T$.

Note that, a solution to our original low-rank approximation problem, (2), can be obtained either by setting $B = U_k \Sigma_k$ and $C = V_k^T$, or by setting $B = U_k$ and $C = \Sigma_k V_k^T$ – the product BC is the same.

One thing surprising about Claim 2 is that it implies that we can find a basis set for an optimal k rank approximation in a *greedy* way. The best ground set of a rank-k approximation, $u_1, \ldots, u_k$, just adds one vector to the best basis set for a rank-$(k-1)$ approximation, $u_1, \ldots, u_{k-1}$.

I'm going to give a proof of Claim 2, but if you have already seen this before in a linear algebra class, feel free to skip it. Or try to reprove it on your own.

Proof. **Case: $k = 1$.**

This statement is easiest to prove for $k = 1$. For rank 1 approximation our goal is to choose $b \in \mathbb{R}^d$ and $c \in \mathbb{R}^n$ to minimize:

$$\|A - bc^T\|_F^2 = \sum_{i=1}^{n} \|a_i - c_i \cdot b\|_2^2,$$

where c_i is the i^{th} entry of c. Without loss of generality, we may assume that b is a unit vector. For a given b, it's clear that we should choose c_i so that $c_i \cdot b$ is the projection of a_i onto b. I.e. we should set $c_i = \langle a_i, b \rangle$ (see Figure 2 for the geometric intuition). Equivalently, we should set:

$$c = b^T A. \tag{3}$$

So, our rank 1 optimization problem actually reduces to:

$$\min_{b\in\mathbb{R}^n, \|b\|_2=1} \|A - bb^T A\|_F^2. \tag{4}$$

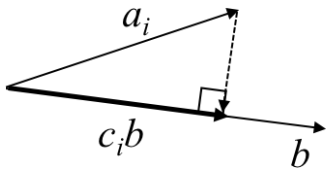

Figure 2: For an optimal rank 1 approximation with a fixed b, we should always choose c_i so that $c_i b$ is the projection of a_i onto b.

In this case, by Pythagorean theorem, $\sum_{i=1}^n \|a_i - c_i \cdot b\|_2^2 = \sum_{i=1}^n \|a_i\|_2^2 - \|c_i b\|_2^2 = \|A\|_F^2 - \sum_{i=1}^n \|c_i b\|_2$. So, in fact, solving (4) is actually equivalent to solving:

$$\max_{b\in\mathbb{R}^n, \|b\|_2=1} \|bb^T A\|_F^2. \tag{5}$$

From this point of view, it is clear that u_1 is the optimal choice for b. Writing A using the SVD, we have

$$\|bb^T A\|_F^2 = \|b^T A\|_2^2 = \|b^T U\Sigma V^T\|_2^2 = \|b^T U\Sigma\|_2^2 = \sum_{i=1}^k (b^T u_i)^2 \sigma_i^2.$$

Since U is orthonormal and $\|b\|_2^2 = 1$, $\sum_{i=1}^k (b^T u_i)^2 = 1$. Accordingly, since $\sigma_1 \geq, \ldots, \geq \sigma_r$, $\sum_{i=1}^k (b^T u_i)^2 \sigma_i^2$ is maximized when $(b^T u_1)^2 = 1$, which can be accomplished by setting $b = u_1$.

Case: $k > 1$**.**

The proof for $k > 1$ is similar. It will be helpful to use the following:

Claim 3 (Matrix Pythagorean Theorem)**.** *If M and N are matrices with mutually orthogonal columns, i.e. $M^T N = 0$, then,*

$$\|M + N\|_F^2 = \|M\|_F^2 + \|N\|_F^2.$$

This is a direct consequence of writing $\|M + N\|_F^2 = \sum_i \|m_i + n_i\|_2^2$ and applying the Pythagorean theorem to each column $m_i + n_i$ separately.

As in the rank 1 case, without loss of generality we can view the low rank approximation problem as choosing an optimal orthonormal matrix B to minimize $\|A - BC\|_F^2$. Using an identical projection argument, the optimal C for a given B is $B^T A$. So our goal is to solve:

$$\min_{B\in\mathbb{R}^{d\times k}, B^T B - I} \|A - BB^T A\|_F^2. \tag{6}$$

By matrix Pythagorean theorem applied to $\|(A - BB^TA) + BB^TA\|_F^2$, we have

$$\|A - BB^TA\|_F^2 = \|A\|_F^2 - \|BB^TA\|_F^2$$

and thus (6) is equivalent to

$$\max_{B \in \mathbb{R}^{d\times k}, B^TB - I} \|BB^TA\|_F^2. \tag{7}$$

Since A can be written as $U\Sigma V^T$, this is equivalent to solving:

$$\max_{B \in \mathbb{R}^{d\times k}, B^TB=I} \|BB^TU\Sigma V^T\|_F^2 = \max_{B \in \mathbb{R}^{d\times k}, B^TB=I} \|B^TU\Sigma\|_F^2$$

$Q = B^TU \in \mathbb{R}^{k\times r}$ has orthonormal rows, so it's columns cannot have norm great than 1. Also the sum of Q's squared column norms is k (its Frobenius norm squared). It follows that $\|Q\Sigma\|_F^2 = \sum_{i=1}^r \|q\|_i^2\sigma_i^2 \leq \sigma_1^2 + \ldots + \sigma_k^2$. This maximum is obtained when Q's first k columns are the standard basis vector – i.e. when $B = U_k$. □

4 Greedily constructing low-rank approximations

As mentioned, one thing that's interesting about the SVD and Claim 2 is that it implies that we can construct an optimal low-rank approximation in a greedy way: if b_1 is the best basis vector for a rank 1 approximation, then there's an optimal rank 2 approximation that maintains b_1 as one of its basis vectors. In fact, this observation gives an approach to *proving* that the SVD exists for all matrices. Again, if you've already seen this proven in another class, feel free to skip this section.

Consider the following iterative routine, which we will prove constructs a singular value decomposition for any matrix A:

- Let $A^{(1)} = A$.
- For $i = 1, \ldots, r$:
  - Let $b_i, c_i = \arg\min_{b,c} \|A^{(i)} - bc^T\|_F^2$.
  - Let $A^{(i+1)} \leftarrow A^{(i)} - b_ic_i^T$ and set $u_i = b_i/\|b_i\|^2, v_i = c_i/\|v_i\|^2, \sigma_i = \|b_i\|_2 \cdot \|c_i\|_2$.
- Set $U = [u_1, \ldots, u_r]$, $V = [v_1, \ldots, v_r]$, and $\Sigma = diag(\sigma_1, \ldots, \sigma_r)$.

Note that step one of the procedure requires an algorithm for computing an optimal *rank 1* approximation to a given matrix. At least to prove *existence* of the SVD, we do not need an actual implementation of this procedure. However, because we do care about computing the SVD and rank-k approximations, we will eventually see an algorithm for solving this rank 1 problem.

We first need to prove the following, which implies that our choice of U is orthonormal:

Claim 4. *If $b_1, \ldots, b_k$ are chosen as above, then $b_i^Tb_j = 0$ for all i, j.*

Proof. Let's remove indices to keep notation simple, and consider an optimal rank 1 approximation bc^T for a matrix A. We claim:

1. b is always in the column span of A.
2. $b^T(A - bc^T) = 0$

The first point follows from a contradiction argument. If b is not in A's column span, it can be written it as $Ax + y$ for some y orthogonal to all of A's columns. Then:

$$\|A - bc^T\|_F^2 = \|A - Axc^T - yc^T\|_F^2 = \|A - Axc^T\|_F^2 + \|yc^T\|_F^2 \geq \|A - Axc^T\|_F^2.$$

The second equality follows from Claim 3 because every column in yc^T is orthogonal to every column in $A - Axc^T$. This is a contradiction because Axc^T is a rank 1 matrix that clearly achieves better error than bc^T, which we claimed was chosen to be optimal. So we conclude that b must in fact lie in A's column span. The second claim follows from our earlier projection argument: $c(i)$ is chosen so that $c(i)b$ is the projection of a_i onto b, and thus $b^T(a_i - c(i)b)$ for all i.

From these two claims, it follows that, for any i, $b_i^T b_{i+1} = 0$, because b_{i+1} is in the column span of $A^{(i+1)}$ but b_i is orthogonal to that span. Then, by induction $b_i^T b_j = 0$ for all $j > i+1$ as well. We will just argue one step of the induction: $b_i^T A^{(i+2)} = b_i^T(A^{(i+1)} - b_{i+1}c_{i+1}^T) = 0 - 0$. So b_i is orthogonal to anything in the column span of $A^{(i+2)}$, and is thus orthogonal to b_{i+2}. □

The same exact argument (applied to rows instead of columns) also let's us establish:

Claim 5. *If $c_1, \ldots, c_k$ are chosen using our greedy procedure above, then $c_i^T c_j = 0$ for all i, j. In other words, our ground basis is orthogonal.*

Finally, we note that, since $b_1, \ldots, b_r$ are all orthogonal to the column span of $A^{(r+1)}$, then it must be that $A^{(r+1)} = 0$. So $\sum_{i=1}^r b_i c_i^T = A$, and thus $\sum_{i=1}^n u_i \sigma_i v_i^T = A$ for the $U = [u_1, \ldots, u_r]$, $V = [v_1, \ldots, v_r]$, and $\Sigma = diag(\sigma_1, \ldots, \sigma_r)$ produced by the iterative algorithm. Combined with Claim 4 and Claim 5 this proves Theorem 1.

5 Computing the best rank-1 approximation

The SVD gives one way of obtaining an optimal low-rank approximation for any rank parameter k. It can be computed in essentially time $O(ndr)$ time, where $r = \min(n, d)$.[1]

In general, this is too slow from many large data matrices. Next lecture we will discuss the power method, which gives a much faster way of finding just the top singular vector of a matrix, which is typically the problem we want to solve in data applications. The power method runs in approximately $O(nd)$ time. It can thus find find k singular vectors iteratively in $O(ndk)$ time, which is much faster than a full SVD when $k \ll rank(A)$.

[1] We say "roughly" because technically there is no "exact" algorithm for the SVD, even in the Real RAM model of computation. This is consequence of the Abel-Ruffini theorem. Thus, all SVD algorithms are technically approximation algorithms. However, standard methods obtain *very* good ϵ dependence. E.g. the QR algorithm can compute a factorization $U\Sigma V^T$ with $\|U\Sigma V^T - A\| \leq \epsilon$ in $O(nd^2 + d^2 \log\log(1/\epsilon))$ time. The second term is ignored because it's always lower order in practice.

References

[1] John Novembre, Toby Johnson, Katarzyna Bryc, Zoltan Kutalik, Adam R Boyko, Adam Auton, Amit Indap, Karen King, Sven Bergmann, Matthew Nelson, Matthew Stephens, Carlos Bustamante, . (2008). Genes Mirror Geography within Europe. Nature. 456. 274.

[2] Christopher D. Manning, Prabhakar Raghavan and Hinrich Schtze, Introduction to Information Retrieval, Cambridge University Press. 2008. https://nlp.stanford.edu/IR-book/information-retrieval-book.html.

[3] T.K. Landauer, P.W. Foltz, and D. Laham. Introduction to Latent Semantic Analysis. Discourse Processes, 25, 259-284. (1998)