princeton univ. F'13 cos 521: Advanced Algorithm Design

Lecture 13: Intrinsic dimensionality of data and low-rank approximations: SVD

Lecturer: *Sanjeev Arora* Scribe:

Today's topic is a technique called singular value decomposition. We'll take two views of it, and then see that there is a surprising algorithm for it.

1 View 1: Inherent dimensionality of a dataset

In many settings we have a set of m vectors $v_1, v_2, \ldots, v_m$ in $\Re^n$. Think of n as large, and maybe m also. We would like to represent v_i's using fewer number of dimensions. We saw one technique in Lecture 11, namely, dimension reduction. Unfortunately, as we mentioned there, dimension reduction is only known in the setting where we only care about the pairwise ℓ_2 distance of the vectors.

But in many settings the v_i's have a special structure: they are well-approximated by some low-dimensional set of vectors. By this we mean that for some small k, there are vectors $u_1, u_2, \ldots, u_k \in \Re^n$ such that every v_i is close to the *span* of $u_1, u_2, \ldots, u_k$. This means that there are numbers $\alpha_{i1}, \ldots, \alpha_{ik} \in \Re$ such that

$$\left| v_i - \sum_j \alpha_{ij} u_j \right|_2^2 \approx \text{small} \tag{1}$$

Example 1 (Understanding shopping data) Suppose a marketer is trying to assess shopping habits. They observe the shopping behaviour of m shoppers with respect to n goods: how much of each good did they buy? This gives m vectors in $\Re^n$.

The simplest model for this would be: every shopper starts with a budget, and allocates it equally among all m items. Then if B_i is the budget of shopper i and p_j is the price for item j the ith vector is $\frac{1}{n}(\frac{B_i}{p_1}, \frac{B_i}{p_2}, \ldots, \frac{B_i}{p_n})$. Denoting by $\vec{u}$ the vector of price inverses, that is $(1/p_1, 1/p_2, \ldots, 1/p_n)$ this is just $\frac{B_i}{n}\vec{u}$. We conclude that the data is 1-dimensional: just scalar multiples of $\vec{u}$.

Now maybe the above model is too unrealistic and doesn't quite fit the data well. Then one could try another model. We assume that the goods partition into k categories like produce, canned goods, etc. $S_1, S_2, \ldots, S_k$. These categories are unknown to us. Assume furthermore that the ith shopper designates a budget B_{it} for the tth category, and then divides this budget equally among goods in that category. Let $u_t \in \Re^n$ denotes the vector that is 0 for goods not in S_t and the inverse prices for goods in S_t. The the quantities of each good purchased by shopper i are given by the vector $\sum_{t=1}^k \frac{B_{it}}{|S_t|} u_t$. In other words, this model predicts that the dataset is k-dimensional.

Of course, no model is exact so the data set will only be approximately k-dimensional in the sense of (1).

One can consider alternative probabilistic models of data generation where the shopper picks items randomly from each category. You'll analyse that in the next homework.

Example 2 (Understanding microarray data in biology) The number of genes in your cell is rather large, and their activity levels —which depend both upon your genetic code and environmental factors— determine your body's functioning. *Microarrays* are tiny "chips" of chemicals sites that can screen the activity levels —aka *gene expression* levels—of a large number of genes in one go, say $n = 10,000$ genes. Typically these genes would have been chosen because they are suspected to be related to the phenomenon being studied, say a particular disease, immune reaction etc. If one tests m individuals, one ends with with m vectors in $\Re^n$.

In practice it is found that this gene expression data is low-dimensional. This means that there are say 4 directions u_1, u_2, u_3, u_4 such that most of the vectors are close to their span. These new axis directions usually have biological meaning; eg groups of genes whose expression (up or down) is controlled by common regulatory mechanisms.

We conclude that finding the α_{ij}'s and u's as in (1) would be nice. It seems like a difficult nonlinear optimization problem. Surprisingly, it is actually fairly easy it is a built-in primitive in Matlab. Let's discuss why its easy, which I find one of the miracles of math (one of few natural nonlinear problems that are solvable in polynomial time).

2 View 2: Low rank matrix approximations

We have an $m \times n$ matrix M. We suspect it is actually a noisy version of a rank-k matrix, say $\tilde{M}$. We would like to find out $\tilde{M}$. One natural idea is to solve the following optimization problem

$$\min \sum_{ij} \left| M_{ij} - \tilde{M}_{ij} \right|^2 \qquad \text{s.t.} \quad \tilde{M} \text{ is a rank-}k \text{ matrix} \tag{2}$$

Again, seems like a hopeless nonlinear optimization problem. Peer a little harder and you realize that a rank-k matrix is just one whose rows are linear combinations of k independent vectors. So this is nothing but problem (1)!

Example 3 (Planted bisection/Hidden Bisection) Graph bisection the problem where we are given a graph $G = (V, E)$ and wish to partition V into two equal sets $S, \overline{S}$ such that we minimize the number of edges between $S, \overline{S}$. It is NP-complete. Let's consider the following average case version.

Create a random graph on n nodes. Partition nodes into S_1, S_2. Within S_1, S_2 put each edge with prob. p, and between S_1, S_2 put each edge with prob. q where $q < p$. Now give this to the algorithm. Note that the algorithm doesn't know S_1, S_2. It has to find the optimum bisection.

It is possible to show using Chernoff bounds that if $q = \Omega(\frac{\log n}{n})$ then with high probability the optimum bisection in the graph is the planted one, namely, S_1, S_2. How can the algorithm recover this partition?

The observation in Figure 1 suggests that the adjacency matrix is close to a rank 2 matrix shown there: the block within S_1, S_2 have value p in each entry; the blocks between S_1, S_2 have q in each entry.

Maybe if we can solve (2) with $k = 2$ we are done? This turns out to be correct as we will see in next lecture.

content...

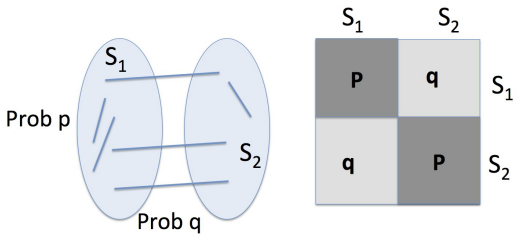

Figure 1: Planted Bisection problem: Edge probability is p within S_1, S_2 and q between S_1, S_2 where $q < p$. On the right hand side is the adjacency matrix. If we somehow knew S_1, S_2 and grouped the corresponding rows and columns together, and squint at the matrix from afar, we'd see more density of edges within S_1, S_2 and less density between S_1, S_2. Thus from a distance the adjacency matrix looks like a rank 2 matrix.

One can study planted versions of many other NP-hard problems as well.

Many practical problems involve graph partitioning. For instance, image recognition involves first partitioning the image into its component pieces (sky, ground, tree, etc.); a process called *image segmentation* in computer vision. This is done by graph partitioning on a graph defined on pixels where edges denote *pixel-pixel similarity*. Perhaps planted graphs are a better model for such real-life settings than worst-case graphs.

3 Singular Value Decomposition

Now we describe the tool that lets us solve the above problems.

For simplicity let's start with a symmetric matrix M. Suppose its eigenvalues are $\lambda_1, \ldots, \lambda_n$ in decreasing order by absolute value, and the corresponding eigenvectors (scaled to be unit vectors) are $e_1, e_2, \ldots, e_n$. (These are column vectors.) Then M has the following alternative representation.

Theorem 1 (Spectral decomposition)
$M = \sum_i \lambda_i e_i e_i^T$.

Proof: At first sight, the equality does not even seem to pass a "typecheck"; a matrix on the left and vectors on the right. But then we realize that $e_i e^T$ is actually an $n \times n$ matrix, albeit of rank 1. So the right hand side is indeed a matrix. Let us call it B.

Any matrix can be specified completely by describing how it acts on an orthonormal basis. By definition, M is the matrix that acts as follows on the orthonormal set

$\{e_1, e_2, \ldots, e_n\}$: $Me_j = \lambda_j e_j$. How does B act on this orthonormal set? We have

$$\begin{aligned} Be_j &= (\sum_i \lambda_i e_i e_i^T) e_j \\ &= \sum_i \lambda_i e_i (e_i^T e_j) \qquad \text{(distributivity and associativity of matrix multiplication)} \\ &= \lambda_j e_j \end{aligned}$$

since $e_i^T e_j =< e_i, e_j >$ is 1 if $i = j$ and 0 else. We conclude that $B = M$. □

THEOREM 2 (BEST RANK k APPROXIMATION)
The solution $\tilde{M}$ to (2) is simply the sum of the first k terms in the previous Theorem.

The proof of this theorem uses the following, which is not too hard to prove from the spectral decomposition using definitions.

THEOREM 3 (COURANT-FISHER)
If $e_1, e_2, \ldots, e_n$ are the eigenvectors as above then:

1. *e_1 is the unit vector that maximizes $|Mx|_2^2$.*
2. *e_{i+1} is the unit vector that is orthogonal to $e_1, e_2, \ldots, e_i$ and maximizes $|Mx|_2^2$.*

Let's prove Theorem 2 for $k = 1$ by verifying that the first term of the spectral decomposition gives the best rank 1 approximation to M. Denoting the rows of M as $M_1, M_2, \ldots, M_n$, we are trying to find a unit vector x such that the M_i's have small squared distances to the line defined by x. Thus we are trying to minimize

$$\sum_i |M_i - < M_i, x > x|^2 = \sum_i |M_i|^2 - \sum_i |< M_i, x >|^2 .$$

This minimization is tantamount to maximising

$$\sum_i |< M_i, x >|^2 = |Mx|^2 , \tag{3}$$

which by the Courant-Fisher theorem is minimized for $x = e_1$. Thus the best rank 1 approximation to M is the matrix whose ith row is $< M_i, e_1 > e_1^T$, which of course is $\lambda_1 e_{1i} e_1^T$. Thus the rank 1 matrix approximation is $\lambda_1 e_1^T e_1$, which proves the theorem for $k = 1$. The proof of Theorem 2 for general k follows similarly by induction.

3.1 General matrices: Singular values

Now we look at general matrices that are not symmetric. The notion of eigenvalues and eigenvectors have to be modified. The following theorem is proved similarly as in the symmetric case but with a bit more tedium.

THEOREM 4 (SINGULAR VALUE DECOMPOSITION AND BEST RANK-k-APPROXIMATION) *An $m \times n$ real matrix has $t \leq \min\{m, n\}$ nonnegative real numbers $\sigma_1, \sigma_2, \ldots, \sigma_t$ (called singular values) and two sets of unit vectors $U = \{u_1, u_2, \ldots, u_t\}$ which are in $\Re^m$ and $V = v_1, v_2, \ldots, v_t \in \Re^n$ (all vectors are column vectors) where U, V are orthogonormal sets and*

$$u_i^T M = \sigma_i v_i \qquad \text{and} \qquad M v_i = \sigma_i u_i^T \tag{4}$$

Furthermore, M can be represented as

$$M = \sum_i \sigma_i u_i v_i^T. \tag{5}$$

The best rank k approximation to M consists of taking the first k terms of (5) and discarding the rest.

This solves problems (1) and (2). Next time we'll go into some detail of the algorithm for computing them. In practice you can just use matlab or another package.

4 View 3: Directions of Maximum Variance

The above proof of Theorem 2, especially the subcase $k = 1$ we proved, also shows yet another view of SVD which is sometimes useful in data analysis. Let us again see this in the case of symmetric matrices. Suppose we shift the given points $M_1, M_2, \ldots, M_n$ so that their mean $\frac{1}{n}\sum_i M_i$ is the origin. Then the rank-1 SVD corresponds to the direction x where the projections of the given data points —a sequence of n real numbers— have maximum *variance*. Note that this variance is exactly the quantity in (3). The second SVD direction corresponds to directions with maximum variance after we have removed the component along the first direction, and so on.

BIBLIOGRAPHY

1. O. Alter, P. Brown, and D. Botstein. Singular value decomposition for genome-wide expression data processing and modeling. *PNAS* August 29, 2000 vol. 97 no. 18

2. Relevant chapter of Hopcroft-Kannan book on data science. (link on course website)