2018-05-29

1 Some factorization tools

We generally seek to approximately factor a data matrix $A \in \mathbb{R}^{m \times n}$ as

$$A \approx LMR, \quad L \in \mathbb{R}^{m \times r}, M \in \mathbb{R}^{r \times r}, R \in \mathbb{R}^{r \times n}.$$

In this view, we can think of R (or MR) as a map from the original attributes to a smaller set of *latent factors*. Different factorization methods differ both in the structural constraints on L, M, and R and on how the approximation is chosen. For today, we will discuss:

- The *symmetric eigendecomposition*

  $$A = Q\Lambda Q^T$$

  where Q is an orthogonal matrix ($Q^TQ = I$) of eigenvectors and Λ is the diagonal matrix of eigenvalues. We will also consider the decomposition for the *generalized* problem, in which $Q^TMQ = I$ where M is a symmetric and positive definite matrix.

- The *singular value decomposition*

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

  where U and V are orthogonal matrices (in the full version) or matrices with orthonormal columns (in the economy version), and Σ is the diagonal matrix of singular values. The SVD is a veritable Swiss Army knife of matrix factorizations; we will use it on its own and as a building block for other methods.

- The *pivoted QR factorization*

  $$A\Pi = QR$$

  where Π is a permutation of the columns of A, Q is orthogonal, and R is upper triangular. The permutation Π is chosen to guarantee that the magnitude of the diagonal entries of R appear in descending order. The pivoted QR factorization also has a geometric interpretation that is useful in several settings.

- The *interpolative decomposition* (ID)

$$A\Pi \approx C \begin{bmatrix} I & T \end{bmatrix}$$

  where C is drawn from the columns of A and the entries of T are bounded in size (no more than two). The factorization is exact when A is low rank. In many cases, the columns in C are the columns that would be chosen by pivoted QR factorization.

- The *CUR factorization*

$$A \approx CUR$$

  where C and R are drawn from the columns and rows of A.

The symmetric eigendecomposition and the SVD are both closely tied to "nice" continuous optimization problems, and this connection allows us to make very strong statements about them. In contrast, pivoted QR, ID, and CUR involve discrete choices involving column and row selection, and it is much more difficult to analyze the properties that arise from these discrete choices. At the same time, these latter factorizations are often *interpretable* in a way that the eigendecomposition and SVD are not.

2 The symmetric eigenvalue problem

Among their many other uses, in data analysis tasks a real *symmetric* matrix A often represent interactions between unordered pairs of objects. We use them to represent edges in undirected graphs, similarities between objects, covariances of pairs of random variables, counts of pairs of words that occur together across sets of documents, and in many other settings.

From the linear algebra perspective, a symmetric matrix also represents a *quadratic form*, i.e. a purely quadratic function of many variables. For a concrete vector space, we write this as

$$\phi(x) = \frac{1}{2} x^T A x.$$

We say ϕ (and A) is *positive definite* if ϕ is positive for any nonzero argument; ϕ is *positive semidefinite* if ϕ is always non-negative. Any positive semidefinite matrix can be represented (not uniquely) as a *Gram matrix* $A = B^T B$; in this case, we have $\phi(x) = \|Bx\|^2/2$. If A is a positive semidefinite matrix

that represents similarities between objects, we can think of the rows of the matrix B as feature vectors for different objects, so that the similarity between objects is encoded as the dot product between their feature vectors. This is a particularly useful perspective if we want to use matrix factorization methods to cluster similar objects.

A real symmetric matrix is always diagonalizable with real eigenvalues, and has an orthonormal basis of eigenvectors $q_1, \ldots, q_n$, so that we can write the eigendecomposition

$$A = Q\Lambda Q^T.$$

Broadly speaking, I tend to distinguish between two related perspectives on eigenvalues. The first is the linear map perspective: A represents an operator mapping a space to itself, and an eigenvector corresponds to an *invariant direction* for the operator; that is, we read the decomposition as

$$Aq_i = q_i\lambda_i.$$

The second perspective is the quadratic form perspective: if A is a symmetric matrix representing a quadratic form, then we read the decomposition as

$$\begin{aligned}(Qx)^T A(Qx) &= \sum_{i=1}^n \lambda_i x_i^2 \\ x^T Ax &= \sum_{i=1}^n \lambda_i (q_i^T x)^2.\end{aligned}$$

We can write the eigenvalues and eigenvectors as the critical points for the objective function $x^T Ax$ subject to $\|x\|^2 = 1$; in terms of the Lagrangian

$$L(x, \lambda) = \frac{1}{2} x^T Ax - \frac{\lambda}{2}(x^T x - 1),$$

we have the KKT conditions

$$\begin{aligned}Ax - \lambda x &= 0 \\ \|x\|^2 - 1 &= 0\end{aligned}$$

The eigenvalues and eigenvectors are also the stationary values and vectors for the *Rayleigh quotient*

$$\rho_A(x) = \frac{x^T Ax}{x^T x}.$$

If we differentiate $x^T Ax - \rho_A x^T x = 0$, we have

$$2\delta x^T (Ax - \rho_A x) - \delta\rho_a (x^T x) = 0$$

which means that setting $\delta\rho_A = 0$ implies

$$Ax - \rho_A(x)x = 0.$$

The largest eigenvalue is the maximum of the Rayleigh quotient, and the smallest eigenvalue is the minimum of the Rayleigh quotient.

If M is symmetric and positive definite, it defines an inner product and an associated Euclidean norm:

$$\langle x, y\rangle_M = y^T Mx \quad \text{and} \quad \|x\|_M^2 = x^T Mx = \langle x, x, \rangle_M.$$

If A is a symmetric matrix and M is symmetric and positive definite, we might also consider minimizing the quadratic form for A subject to the constraint $\|x\|_M = 1$. This gives us the KKT equations

$$\begin{aligned} Ax - Mx\lambda &= 0 \\ \|x\|_M^2 - 1 &= 0. \end{aligned}$$

This is an example of a *generalized* eigenvalue problem for the matrix pencil (A, M). We can write down a full eigendecomposition for the generalized problem as

$$U^T AU = \Lambda \text{ where } U^T MU = I.$$

The eigenvalues of the pencil (A, M) are the same as those of $M^{-1}A$ or $B^{-T}AB^{-1}$ for any B such that $M = B^T B$. We have already described one example where this type of generalized decomposition is useful when we discussed "fisherfaces" in the last lecture.

The optimization problem associated with the symmetric eigenvalue problem is *not* convex – far from it! But it is a nice optimization problem nonetheless. It has saddle points, but the only local minima and maxima are also global minima and maxima, so even standard optimizers are not prone to getting stuck in a local minimum. But there are also specialized iterations for solving the symmetric eigenvalue problem that are very efficient. While the details of these methods are a topic for a different class, it is worth sketching just a few ideas in order to understand the different types of solvers available and their complexities:

- Most of the *direct* solvers[1] for the symmetric eigenvalue problem go through two stages. First, we compute

$$A = UTU^T$$

  where U is an orthogonal matrix and T is tridiagonal. Then we compute the eigenvalues and eigenvectors of the resulting tridiagonal matrix. The asymptotically fastest of the tridiagonal eigenvalue solvers (the "grail" code) takes $O(n)$ time to find all eigenvalues and $O(n^2)$ to find all eigenvectors; but the cost of reducing A to a tridiagonal form is $O(n^3)$. When we call `eig` in MATLAB, or related solvers in other languages, this is the algorithm we use.

- There are also several *iterative* methods for computing a few eigenvalues and eigenvectors of a symmetric matrix. The simplest methods are the *power iteration* and *subspace iteration*, which you may have seen in a previous class; unfortunately, these methods do not converge very quickly. The *Lanczos* method is the main workhorse of sparse eigenvalue solvers, and the method you will use if you call `eigs` in MATLAB, or related subroutines in other languages. Like power iteration and subspace iteration, the Lanczos method requires only matrix-vector multiplications, and so it can be very efficient when we want to compute matrix-vector products.

The Lanczos method is good at computing a few of the largest and smallest eigenvalues of a symmetric matrix (the *extremal eigenvalues*), together with the corresponding eigenvectors. It is not as good at computing *interior* eigenvalues without some help, usually in the form of a *spectral transformation*. But for many applications in data science, we are content to look at the extremal eigenvalues and vectors, so we will not discuss the more difficult interior case any further.

3 The singular value decomposition

The singular value decomposition of $A \in \mathbb{R}^{m \times n}$ is

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

[1] This is a bit of a fib in that all eigenvalue solvers are iterative. We say these methods are "direct" because we only need a constant number of iterations per eigenvalue to find the eigenvalues to nearly machine precision.

where U and V have orthonormal columns and Σ has the non-negative *singular values* of A in descending order on the diagonal (and zeros elsewhere). We sometimes distinguish between the "full" SVD in which U and V are square and Σ is the same size as A; and the "economy" SVD, in which Σ is square and one of U or V is the same size as A.

Like the symmetric eigenvalue decomposition, the singular value decomposition can be viewed in terms of an associated optimization problem:

$$\text{minimize } \|Av\|_2 \text{ s.t. } \|v\|_2 = 1.$$

We change nothing by squaring the norms to get

$$\text{minimize } \|Av\|_2^2 \text{ s.t. } \|v\|_2^2 = 1,$$

but we can rewrite this as

$$\text{minimize } v^T A^T A v \text{ s.t. } v^T v = 1,$$

which we recognize as exactly the optimization formulation of the maximum eigenvalue (and eigenvector) of $A^T A$. More generally, we have

$$A^T A = V \Sigma^2 V^T.$$

where the eigenvectors of $A^T A$ are the constrained critical points for the optimization problem and the (non-negative) eigenvalues in Σ^2 are the corresponding eigenvectors. Having computed the vectors V, we write

$$AV = U\Sigma$$

where we know the columns of U each have unit Euclidean norm. To see that the columns of U are actually orthonormal, we write

$$AA^T = AVV^T A^T = U\Sigma^2 U^T.$$

And this is exactly the eigenvalue decomposition of AA^T!

We say a function $f : \mathbb{R}^{m \times n} \to \mathbb{R}$ is *orthogonally invariant* (or *unitarily invariant*[2] if $f(Q_1 A Q_2) = f(A)$ for any orthogonal matrices Q_1 and Q_2. Any unitarily invariant function can be written in terms of the singular values of A. Among the most important unitarily invariant functions are the *Ky-Fan* norms, which are just the ℓ^p norms of the vector of singular values. We generally only care about three of these norms:

[2]Unitarily invariant covers the complex case

- The *operator 2-norm* (or *spectral norm*) is $\|A\|_2 = \sigma_1$; this is the Ky-Fan norm for $p = \infty$.
- The *Frobenius norm* satisfies $\|A\|_F^2 = \sum_i \sigma_i^2$; that is, $\|A\|_F$ is the Ky-Fan norm for $p = 2$.
- The *nuclear norm* satisfies $\|A\|_* = \sum_i \sigma_i$; that is, $\|A\|_*$ is the Ky-Fan norm for $p = 1$. We will see more of this norm when we talk about matrix completion on Friday.

The Eckart-Young theorem tells us that the *best* rank k approximation to A in the spectral norm or the Frobenius norm is the truncated singular value decomposition. In fact, the theorem holds for any of the Ky-Fan norms (so it is true in the nuclear norm as well).

The singular value decomposition goes by many names and has many close relatives. It is sometimes called the *proper orthogonal decomposition* of a data matrix; in statistics, it is the basis for *principal component analysis*; and in the study of stochastic processes, it is the *Karhuenen-Loève* decomposition. But in some contexts, the SVD is not applied directly to our data matrix A, but is instead applied to a transformed matrix. For example, if the columns of A represent samples of different random variables (and the rows represent experiments), we would typically look at the SVD of the *centered* matrix $A - e\mu^T$ where μ is the vector of column means and e is the vector of all ones. And in other settings, we might look at the matrix of z-scores, which are obtained by normalizing the Euclidean lengths of the vectors of the centered matrix.

4 Pivoted QR and pivoted Cholesky

The SVD provides optimal low-rank approximations to data matrices, but those approximations are not particularly easy to interpret. We therefore turn now to low-rank factorizations in which the factors are formed from subsets of the columns or rows of the data matrix.

We described the *pivoted QR decomposition* briefly in our discussion of least squares. The idea in pivoted QR is to permute the columns of the data matrix so that the diagonal entries of the QR factorization of the pivoted matrix appear in descending magnitude, i.e.

$$A\Pi = QR, \quad r_{ii} \geq r_{i+1,i+1} \text{ for all } i.$$

The column order is computed in a greedy fashion as follows. At the first step of the iteration, we choose the column of A with the largest Euclidean norm; we then scale it by the norm r_{11} in order to get the first column of Q:

$$A\Pi_{:,1} = q_1 r_{11}.$$

Next, we find the column of A with the largest component orthogonal to q_1, and write it as

$$A\Pi_{:,2} = q_1 r_{12} + q_2 r_{22}$$

where r_{12} is the extent to which the vector projects onto q_1, q_2 is the direction of the residual component orthogonal to q_1, and r_{22} is the magnitude of that second component. And we proceed in a similar fashion until we run out of columns or until all the remaining residuals are tiny.

Though we have described this decomposition in terms of explicit orthogonalizations at each step (as in the Gram-Schmidt procedure), in practice we would usually use an alternate algorithm based on orthogonal transformations. However, the algorithm we have described is useful for reasoning about a special property of the algorithm: the columns it selects are all on the *convex hull* of the set of columns of A. In general, for any vector u, the column for which $|u^T A|$ is largest will lie on the convex hull of the columns of A; and by design, $q_i^T A\Pi = R_{i,:}$ has its largest entry on the diagonal. This fact is key to the use of pivoted QR in some algorithms for *separable* non-negative matrix factorization, as we will discuss next time.

Closely related to the pivoted QR algorithm is the *pivoted Cholesky* algorithm: for a positive semi-definite matrix A, pivoted Cholesky computes

$$\Pi^T A \Pi = R^T R$$

where the diagonal entries of R have descending magnitude. Like pivoted QR, pivoted Cholesky is a greedy algorithm, and at each step it chooses the next pivot row/column based on the magnitude of a "residual" diagonal[3]. In exact arithmetic, pivoted Cholesky on a Gram matrix $B^T B$ computes the same permutation and R factor as pivoted QR on B.

[3]This is really the diagonal of the Schur complement, for those of you who may have seen Schur complements in a discussion of Gaussian elimination in an earlier class.

5 Interpolative decomposition and CUR

An *interpolative decomposition* is a decomposition of the form

$$A\Pi \approx C \begin{bmatrix} I & T \end{bmatrix}$$

where C is a subset of the columns of A. One way to get the interpolative decomposition is via truncated pivoted QR; is we write

$$A\Pi \approx Q \begin{bmatrix} R_1 & R_2 \end{bmatrix}$$

where $R_1 \in \mathbb{R}^{k \times k}$ is the leading submatrix in the rectangular R factor, then

$$A\Pi = QR_1 \begin{bmatrix} I & R_1^{-1} R_2 \end{bmatrix} = C \begin{bmatrix} I & T \end{bmatrix}$$

Unfortunately, if we only use truncated pivoted QR, we cannot guarantee that we are very close to the best rank k approximation; nor can we guarantee that the entries of T are nice. It is possible to show that there is *some* permutation such that the entries of T are at most 2 in magnitude, the singular values of $\begin{bmatrix} I & T \end{bmatrix}$ lie between 1 and $1 + \sqrt{k(n-k)}$, and the approximation error is within a factor of $1 + \sqrt{k(n-k)}$ of the best possible. But pivoted QR might not find the best choice. Fortunately, we can get close to optimal by either a post-processing algorithm that iteratively swaps new columns for the original selection of columns in order to reduce the size of elements in T; or we can use randomized algorithms.

In the interpolative decomposition, we choose a subset of the columns of A as the basis for our approximation. In the CUR decomposition, we choose both columns *and rows*, i.e.

$$A \approx CUR$$

where C and R are drawn from the rows and columns of A. Given a choice of C and R, we find the optimal choice of U by least squares:

$$U = C^\dagger A R^\dagger.$$

Again, though, we have a problem: how should we select the columns and rows to use? A simple approach is to run pivoted QR on both the rows and columns, potentially with "swapping" algorithms of the type used in ID. An alternative approach is to choose rows and columns based on a randomized scheme using approximate "leverage scores" to determine the importance of choosing a given column or row for the final factorization.