2021-03-11

1 Non-negative Matrix Factorization (NMF)

In the last lecture, we considered low rank approximations to data matrices. We started with the "optimal" rank k approximation to $A \in \mathbb{R}^{m\times n}$ via the SVD, then moved on to approximations that represent A in terms of the rows and columns of A rather than in terms of the left and right singular vectors. We argued that while these latter factorizations may not minimize the Frobenius norm of the error for a given rank, they are easier to interpret because they are expressed in terms of the factors in the original data set. We continue with our theme of finding interpretable factorizations today by looking at *non-negative matrix factorizations* (NMF).

Let $\mathbb{R}_+$ denote the non-negative real numbers; for a non-negative data matrix $A \in \mathbb{R}_+^{m\times n}$, we seek

$$A \approx WH, \quad \text{where } W \in \mathbb{R}_+^{m\times k}, H \in \mathbb{R}_+^{k\times n}.$$

Non-negative matrix factorizations are convenient because they express the columns of A (the data) in terms of positively weighted sums of the columns of W, which we interpret as "parts." This type of decomposition into parts makes sense in many different domains; for example:

Meaning of columns of A	Meaning of columns of W
Word distributions for documents	Word distributions for topics
Pictures of faces	Pictures of facial features
Connections to friends	Communities
Spectra of chemical mixtures	Spectra of component molecules

Unfortunately, non-negative matrix factorizations are generally much more difficult to compute than the factorizations we considered in the last lecture. There are three fundamental difficulties:

- We do not know how big k must be to get a "good" representation. Compare this to ordinary factorization, where we can hope for error bounds in terms of $\sigma_{k+1}, \ldots, \sigma_{\min(m,n)}$.
- The optimization problem is non-convex, and there may generally be many local minima. Again, compare this with the optimal approximation problem solved by singular value decomposition, which has saddle points, but has no local minimizers that are not also global minimizers.

- NMF is not *incremental*: the best rank k approximation may have little to do with the best rank $k+1$ approximation. Again, we can compare with the unconstrained problem, for which the best rank $k+1$ approximation is a rank-one update to the best rank k approximation.

Faced with this hard optimization problem, we consider two tactics. First, we might seek efficient optimization methods that at least converge to a local minimizer[1]; we will spend the first part of the lecture discussing this approach. Second, we might seek common special cases where we can prove something about the approximation. In particular, the NMF problem is much more tractable when we make a *separability* assumption which is appropriate in some applications.

2 Going with gradients

2.1 Projected gradient descent

We begin with the *projected gradient descent* algorithm for minimizing a function ϕ subject to simple constraints. Let $\mathcal{P}(x)$ be a projection function that maps x to the nearest feasible point; in the case of a simple non-negativity constraint, $\mathcal{P}(x) = [x]_+$ is the elementwise maximum of x and zero. The projected gradient descent iteration is then

$$x^{k+1} = \mathcal{P}\left(x^{k+1} - \alpha_k \nabla \phi(x^k)\right).$$

The convergence properties of projected gradient descent are similar to those of the unprojected version: we can show reliable convergence for convex (or locally convex) functions and sufficiently short step sizes, though ill-conditioning may make the convergence slow.

In order to write the gradient for the NMF objective without descending into a morass of indices, it is helpful to introduce the *Frobenius inner product*: for matrices $X, Y \in \mathbb{R}^{m\times n}$,

$$\langle X, Y \rangle_F = \sum_{i,j} y_{ij} x_{ij} = \operatorname{tr}(Y^T X).$$

[1] In most cases, we can only show convergence to a stationary point, but we are likely to converge to a minimizer for almost all initial guesses.

The Frobenius inner product is the inner product associated with the Frobenius norm: $\|X\|_F^2 = \langle X, X \rangle_F$, and we can apply the usual product rule for differentiation to compute directional derivatives of $\phi(W,H) = \|A - WH\|_F^2/2$ with respect to W and H:

$$\begin{aligned}
\delta\phi &= \delta\left[\frac{1}{2}\langle A - WH, A - WH\rangle_F\right] \\
&= \langle \delta(A - WH), A - WH\rangle_F \\
&= -\langle (\delta W)H, A - WH\rangle_F - \langle W(\delta H), A - WH\rangle_F.
\end{aligned}$$

We let $R = A - WH$, and use the fact that the trace of a product of matrices is invariant under cyclic permutations of the matrices:

$$\begin{aligned}
\langle (\delta W)H, R\rangle_F &= \operatorname{tr}(H^T(\delta W)^T R) = \operatorname{tr}((\delta W)^T R H^T) = \langle \delta W, RH^T\rangle_F \\
\langle W(\delta H), R\rangle_F &= \operatorname{tr}((\delta H)^T W^T R) = \langle \delta H, W^T R\rangle_F.
\end{aligned}$$

Therefore, the projected gradient descent iteration for this problem is

$$\begin{aligned}
W^{\text{new}} &= \left[W + \alpha R H^T\right]_+ \\
H^{\text{new}} &= \left[H + \alpha W^T R\right]_+,
\end{aligned}$$

where in the interest of legibility we have suppressed the iteration index on the right hand side.

2.2 Multiplicative updates

One of the earliest and most popular NMF solvers is the *multiplicative update* scheme of Lee and Seung. This has the form of a *scaled* gradient descent iteration where we replace the uniform step size α_k with a different (nonnegative) step size for each entry of W and H:

$$\begin{aligned}
W^{\text{new}} &= \left[W + S \odot \left(AH^T - WHH^T\right)\right]_+ \\
H^{\text{new}} &= \left[H + S' \odot \left(W^T A - W^T W H\right)\right]_+,
\end{aligned}$$

where $\odot$ denotes elementwise multiplication. We similarly let $\oslash$ to denote elementwise division to define the nonnegative scaling matrices

$$S = W \oslash (WHH^T), \quad S' = H \oslash (W^T W H).$$

With these choices, two of the terms in the summation cancel, so that

$$W^{\text{new}} = S \odot (AH^T) = W \oslash (WHH^T) \odot (AH^T)$$
$$H^{\text{new}} = S' \odot (W^T A) = H \oslash (W^T WH) \odot (W^T A).$$

At each step of the Lee and Seung scheme, we scale the (non-negative) elements of W and H by non-negative factors, yielding a non-negative result. There is no need for a non-negative projection because the step sizes are chosen increasingly conservatively as elements of W and H approach zero. But because the steps are very conservative, the Lee and Seung algorithm may require a large number of steps to converge.

3 Coordinate descent

The *(block) coordinate descent* method (also known as *block relaxation* or *nonlinear Gauss-Seidel*) for solving

$$\text{minimize } \phi(x_1, x_2, \ldots, x_p) \text{ for } x_i \in \Omega_i$$

involves repeatedly optimizing with respect to one coordinate at a time. In the basic method, we iterate through each i and compute

$$x_i^{k+1} = \text{argmin}_\xi \, \phi(x_1^{k+1}, \ldots, x_{i-1}^{k+1}, \xi, x_{i+1}^k, \ldots, x_p^k).$$

The individual subproblems are often simpler than the full problem. If each subproblem has a unique solution (e.g. if each subproblem is strongly convex), the iteration converges to a stationary point[2]; this is the situation for all the iterations we will discuss.

3.1 Simple coordinate descent

Perhaps the simplest coordinate descent algorithm for NMF sweeps through all entries of W and H. Let $R = A - WH$; then for the (i, j) coordinate of W, we compute the update $w_{ij} = w_{ij} + s$ where s minimizes the quadratic

$$\frac{1}{2}\|A - (W + se_ie_j^T)H\|_F^s = \frac{1}{2}\|R\|_F^2 - s\langle (e_ie_j^T), RH^T\rangle_F + \frac{1}{2}s^2\|e_ie_j^T H\|_F^2,$$

[2] For non-convex problems, we may converge to a saddle; as an example, consider simple coordinate descent for $\phi(x_1, x_2) = x_1^2 + 4x_1x_2 + x_2^2$.

subject to the constraint that $s \geq -w_{ij}$. The solution to this optimization is

$$s = \max\left(-w_{ij}, \frac{(RH^T)_{ij}}{(HH^T)_{jj}}\right).$$

Therefore, the update for w_{ij} is

$$s = \max\left(-w_{ij}, \frac{(RH^T)_{ij}}{(HH^T)_{jj}}\right), \quad w_{ij} := w_{ij} + s, \quad R_{i,:} := R_{i,:} - sH_{j,:}$$

A similar computation for the elements of H gives us the update formulas

$$s = \max\left(-h_{ij}, \frac{(W^T R)_{ij}}{(W^T W)_{ii}}\right), \quad h_{ij} := h_{ij} + s, \quad R_{:,j} := R_{:,j} - sW_{:,i}.$$

Superficially, this looks much like projected gradient descent with scaled step lengths. However, where in gradient descent (or the multiplicative updates of Lee and Seung) the updates for all entries of W and H are independent, in this coordinate descent algorithm we only have independence of updates for a single column of W or a single row of H. This is a disadvantage for efficient implementation.

3.2 HALS/RRI

The simple algorithm in the previous algorithm relaxed on each element of W and H independently. In the *hierarchical alternating least squares* or *rank-one residual iteration*, we treat the problem as consisting of $2k$ vector blocks, one for each column of W and row of H. To update a column $W_{:,j} := W_{:,j} + u$, we must solve the least squares problem

$$\text{minimize } \|R - uH_{j,:}\|_F^2 \text{ s.t. } u \geq -W_{:,j},$$

which is equivalent to solving the independent *single variable* least squares problems

$$\text{minimize } \|R_{i,:} - u_i H_{j,:}\|_2^2 \text{ s.t. } u_i \geq -w_{ij}.$$

The u_i must satisfy the normal equations unless it hits the bound constraint; thus,

$$u_i = \max\left(-w_{ij}, \frac{R_{i,:}H_{j,:}^T}{H_{j,:}H_{j,:}^T}\right) = \max\left(-w_{ij}, \frac{(RH^T)_{ij}}{(HH^T)_{jj}}\right).$$

Thus, updating a column of W at a time is *equivalent* to updating each of the elements in the column in sequence in scalar coordinate descent. The same is true when we update row of H.

3.3 ANLS

The alternating non-negative least squares (ANLS) iteration updates all of W together, then all of H:

$$W := \text{argmin}_{W \geq 0} \|A - WH\|_F^2$$
$$H := \text{argmin}_{H \geq 0} \|A - WH\|_F^2$$

We can solve for each row of W (or column of H) independently by solving a *non-negative least squares problem*. Unfortunately, these non-negative least squares problems cannot be solved in a simple closed form!

The non-negative least squares problem has the general form

$$\text{minimize } \|Ax - b\|^2 \text{ such that } x \geq 0;$$

it is a convex optimization problem that can be solved using any constrained optimization solver. An old class of solvers for this problem is the *active set* methods. To derive these methods, we partition the variables into a free set $\mathcal{I}$ and a constrained set $\mathcal{J}$, and rewrite the KKT equations in the form

$$\begin{aligned} x_{\mathcal{I}} &= A_{\mathcal{I}}^{\dagger} b & x_{\mathcal{I}} &\geq 0 \\ A_{\mathcal{J}}^T (Ax - b) &\geq 0 & x_{\mathcal{J}} &= 0. \end{aligned}$$

If the partitioning into $\mathcal{I}$ and $\mathcal{J}$ is known, we can compute x via an ordinary least squares solve. The difficult part is to figure out which variables are free! The simplest approach is to guess $\mathcal{I}$ and $\mathcal{J}$ and then iteratively improve the guess by moving one variable at a time between the two sets as follows. Starting from an initial non-negative guess x, $\mathcal{I}$, $\mathcal{J}$, we

- Compute $p = A_{\mathcal{I}}^{\dagger} b - x$.
- Compute a new point $x := x + \alpha p$ where $\alpha \leq 1$ is chosen to be as large as possible subject to non-negativity of the new point.
- If $\alpha < 1$, we move the index for whatever component became zero from the $\mathcal{I}$ set to the $\mathcal{J}$ set and compute another step.
- If $\alpha = 1$ and $g_{\mathcal{J}} = A_{\mathcal{J}}^T (Ax - b)$ has any negative elements, we move the index associated with the most negative element of $r_{\mathcal{J}}$ from the $\mathcal{J}$ set to the $\mathcal{I}$ set and compute another step.

- Otherwise, we have $\alpha = 1$ and $g_{\mathcal{J}} \geq 0$. In this case, the KKT conditions are satisfied, and we may terminate.

The problem with this approach is that we only change our guess at the free variables by adding or removing one variable per iteration. If our initial guess is not very good, it may take many iterations to converge. Alternate methods are more aggressive about changing the free variable set (or, equivalently, the active constraint set).

4 Separable NMF

In the general case, non-negative matrix factorization is a hard problem. However, there are special cases where it becomes easier, and these are worth exploring. In a *separable* problem, we can compute

$$\Pi^T A = \begin{bmatrix} I \\ W_2 \end{bmatrix} H;$$

that is, every row of A can be expressed as a positively-weighted combination of k columns of A. Examples where we might see this include:

- In topic modeling, we might have "anchor words" that are primarily associated with just one topic.
- In image decomposition, we might have "pure pixels" that are active for just one part of an image.
- In chemometrics, we might see that a component molecule produces a spike at a unique frequency that is not present for other components.

Assuming that this separability condition occurs, how are we to find the k rows of A that go into H? What we will do is to compute the normalized matrix $\bar{A}$ by scaling each row of A so that it sums to 1. With this normalization, all rows of $\bar{A}$ are positively weighted combinations of the anchor rows *where the weights sum to 1*; that is, if we view each row as a point in m-dimensional space, then the anchor rows are points on the convex hull. As discussed in the last lecture, we can find the convex hull with the pivoted QR algorithm.

$$\bar{A}^T \Pi = QR.$$

Variants of this approach also work for *nearly* separable problems.