week 10: don't be so negative

goals this week

This week we're going to study one paper in depth: Lee and Seung, 1999, *Learning the parts of objects by non-negative matrix factorization*.

We'll follow a pattern that we've been developing throughout the course:

- what's the generative probability model?
- write down the probability of everything (data, parameters, model)
- use probability calculus to infer what we want
- write code to generate synthetic control data
- write code to implement the algorithm

Maybe you can't invest this much time in *every* paper you read, but it's good to know how to do it for ones that matter to you.

Non-negative matrix factorization (NMF) is an attractive testbed for our purposes, for several reasons.

- It's a promising approach for analyzing RNA-seq data. Maybe you'll want to use it in your work.

- It wasn't developed for RNA-seq analysis. Lee and Seung developed it for image analysis, and also showed examples from text analysis (word/document classification). There is a lot of arbitrage across different data analysis communities (genomics, neuroscience, text, images...), and an important skill is to be able to read papers outside yours.

- The Lee and Seung paper is surprisingly deep, compact, and beautiful. It's very rewarding to think through it carefully, because you'll see things that aren't immediately apparent on its surface.
- NMF is an example of a method that sometimes seems to get used without understanding it. Descriptions of the NMF algorithm often seem to have copied and pasted bits of explanation that aren't self-consistent. This makes it a nice example for how satisfying it is to really understand what a method is doing.

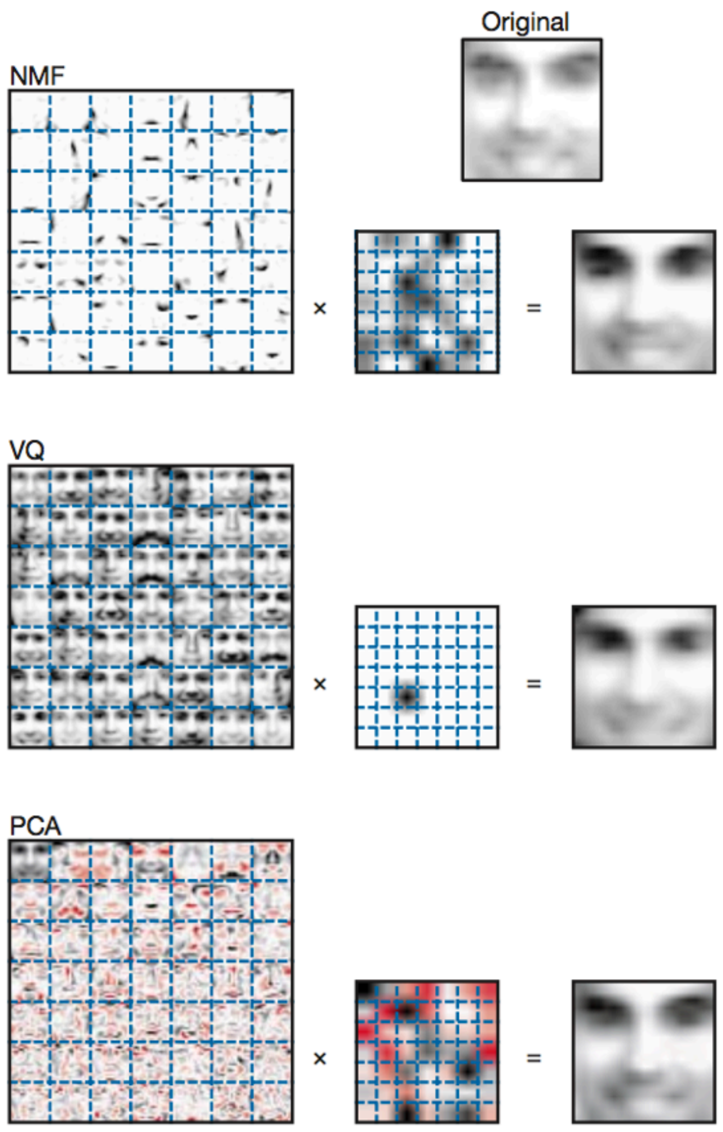

Figure 1 from [Lee and Seung, 1999], comparing three different matrix

gene modules, gene

decomposition techniques. VQ (vector quantization) is sort of like k-means: it aims to assign an observed data point to *one* hidden component, so each component looks like a different kind of face. PCA (principal component analysis) finds an *eigenface* as its main component, then uses additional components to add and subtract from that, so the hidden components tend not to be easily interpretable. NMF (nonnegative matrix factorization), by enforcing nonnegative coefficients, forces a parts-based representation on the hidden components, where the resulting output image is a simple weighted sum of component parts.

batteries

Typically, several gene products work together to get something done, whether it's building a synapse, assembling a ribosome, or metabolizing sugar. Genes that function together tend to be regulated together. By analyzing RNA-seq experiments across many conditions, we can detect sets of genes that are co-regulated, and this gives us clues about gene function. For example, if we observe a cluster of co-regulated genes that includes most of the known ribosomal protein genes and several genes of unknown function, it'd be a good bet that those unknown genes have a role in ribosome synthesis.

The language around this is squishy. We talk almost interchangeably about *pathways*, *modules*, and *networks*. I'm fond of the term gene *batteries*, a term introduced in the 1930's by Thomas Hunt Morgan. All these terms are about co-expression and functional coordination, with shades of subtle distinction. As a rough thing, it's still useful to conceive of genes in terms of *regulatory* genes versus *structural* genes. A smaller number of regulatory genes (like transcription factors) encode products that turn other genes on and off; a larger number of structural genes encode products that get stuff done. We often imagine a regulatory network as a fan-out, with one or a few regulatory gene products turning on a larger set of target structural genes -- a *regulon*, in bacterial genetics. A "wiring diagram" of regulatory relationships (who's turning who on and off) is a *regulatory network*. If we're thinking of it in temporal terms and cartoon arrows (first this guy turns that one on, then that one does this...) we might prefer to talk about a *regulatory pathway*. If we can talk about a piece of the network as if it's relatively self-contained, we might call it a *module*. By *battery*, we tend to be emphasizing an array of target genes with a coordinated function, not the regulatory layer that turned them on.

It's the general concept -- that co-expression is a signal for co-function -- that motivates the development of methods for detecting correlated sets of genes in expression data.

There are a plethora of such techniques: hierarchical clustering, k-means clustering, PCA, and many more. How to think about these methods; how to understand them deeply; how to assess their assumptions, strengths, and weaknesses? Let's choose one, and understand it step by step: an interesting method called **nonnegative matrix factorization**.

Derivation of NMF as a parts-based representation

The more that genes are organized in *disjoint* modules -- each gene goes to one and only one module -- the more that *global* clustering methods will shine, such as hierarchical clustering or PCA. But genes can play multiple roles, in different contexts. The "modules" in one cell type can be different than those in another cell type. It's attractive to consider clustering methods that could allow one gene to be a member of different modules. NMF is an example of such a method.

We observe RNA-seq data (mapped read counts) for N genes. Call the observed count data in one sample V_i. (I'm going to use nomenclature that sticks close to the Lee and Seung paper.)

Imagine that there are R different modules that organize the co-expression of N genes.

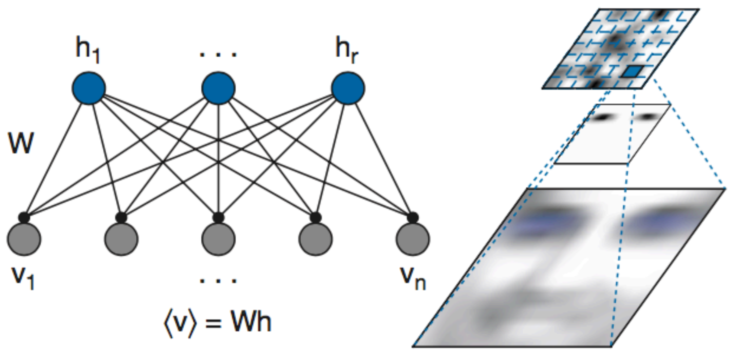

Figure 3 from [Lee and Seung, 1999] showing the generative probability model.

By itself, each module $a = 1..R$ co-expresses a set of genes i at relative expression levels W_{ia}. W_{ia} are relative expression levels, so they are probabilities (frequencies) that sum to one over all genes: $\sum_i W_{ia} = 1$. We imagine that they are *sparse*, with many $W_{ia} = 0$, because the module only co-expresses a subset of genes.

In a given cell type (or condition), we imagine that each module a is itself expressed at some level, relative to the other modules: call this H_a, the relative expression level of module a. The H_a are also probabilities: $\sum_a H_a = 1$. (As we'll see, this is a notational difference from Lee and Seung, who have their H_a in units of counts, not probabilities. Their focus is on the matrix algebra; mine is on the probabilistic generative model, so it makes sense from the probabilistic perspective to make the H_a probabilities.)

In different samples (cell types, experiments), it's the relative level of each *module* that changes; the W_{ia} are constant across samples. The expected expression level of each gene i is then a weighted sum of the gene's expression level in each module, over all modules. We collect some number of total mapped read counts C, so the expected counts $\langle V_i \rangle$ is:

$$\langle V_i \rangle = C \sum_a W_{ia} H_a$$

The RNA-seq count data V_i we actually observe for gene i have noise around this expected mean.

To make a generative probability model, we have to state an assumption about the noise. Let's suppose for example that the only noise in the experiment is Poisson counting noise, and to simplify notation a bit, let's define the mean $\lambda_i = \langle V_i \rangle = C \sum_a W_{ia} H_a$. Now the probability of observing V_i counts is a Poisson distribution:

$$P(V_i \mid W, H) = \frac{\lambda_i^{V_i} e^{-\lambda_i}}{V_i\,!}$$

the choice of noise model

We could've made a different assumption about the noise, perhaps a more sophisticated one. For example, another simple option would be to assume that we're adding Gaussian noise to the mean. (If we're being careful, we distinguish *applying* Poisson noise from *adding* Gaussian noise: adding Gaussian noise is literally adding a noise term, the way we did with linear regression, whereas Poisson noise sticks the mean into the Poisson equation.) There are two main branches of how the NMF algorithm is described: one implicitly assumes Poisson count noise, and the other implicitly assumes additive Gaussian noise. Least-squares-like equations arises in the additive Gaussian noise case, just as we've seen before when we thought about linear regression as a generative model.

Sometimes the two are confused. You can easily find references that give the algorithm for the Poisson version, but show the objective function for the Gaussian version, and vice versa. It's easy to mix them up if you think of NMF as something arbitrary. It's more difficult to get mixed up if you think about it from first principles as a generative model.

Thinking about it this way also helps us see how we might tune NMF to our problem better. There's nothing sacred about Lee and Seung's derivation in terms of Poisson noise, and we're allowed to state a different model. In RNA-seq analysis, where observed data are overdispersed, I may not want to assume simple Poisson noise; I might want to fold a common dispersion and a mean-variance relationship into my model of how observed RNA-seq counts are dispersed around a mean expression level $\langle V_i \rangle$.

But Lee and Seung's paper makes the Poisson noise assumption, so we'll follow that along.

the log likelihood

Imagine that you did many different experiments, in different cell types or conditions that varied the expression level of the modules. Now you'd see correlations in expression levels of genes i, induced by the up- and down-regulation of the hidden modules that control them. If we had enough data, we should be able to solve for the hidden parameters of our model, especially the W_{ia}, which will tell us which genes are co-expressed, at what level, by each module.

Let's introduce an additional index μ over M different samples (conditions, experiments), $\mu = 1..M$. Each sample has its own *mixing coefficients* $H_{a\mu}$. There is still a common set of module-specific relative expression levels W_{ia}. (Lee and Seung call the W_{ia} the *basis images*, and the $H_{a\mu}$ the *encodings*.) Our observed count data are now an $N \times M$ matrix $V_{i\mu}$. Our expected means are $\lambda_{i\mu} = C_\mu \sum_a W_{ia} H_{a\mu}$, where C_μ is the total number of mapped reads in experiment μ.

The total log likelihood is the sum of our Poisson probability over all experiments and all genes:

$$\log P(V \mid W, H) = \sum_{\mu=1}^{M} \sum_{i=1}^{N} \left[V_{i\mu} \log \lambda_{i\mu} - \lambda_{i\mu} - \log(V_{i\mu}!) \right]$$

Compare this to equation (2) in [Lee and Seung, 1999]. They don't bother with the $\log(V_{i\mu}!)$ term because it's a constant independent of W and H, so it's about to disappear when we take the gradient of the log likelihood. They show our $\lambda_{i\mu}$ as $(WH)_{i\mu}$ because they're using matrix notation that we haven't introduced yet (we will soon). Also, as we'll see, they're assuming that the $H_{a\mu}$ are in units of counts, like the V are; in contrast, my derivation assumes that the $H_{a\mu}$ are probabilities, and I'm keeping track of the total counts per experiment, C_μ, explicitly and separately.

maximum likelihood estimation of W and H

So we have the log likelihood; but what we want to do is to infer optimal parameters W and H given observed data V. As usual a reasonable definition of "optimal" is "maximum likelihood", and to find the parameters that maximize the log likelihood, we take its gradient: its partial derivative with respect to each parameter.

If you're not adept at this: an important trick is that you have to consider each term in this large summation individually, and consider whether it depends on a *particular* parameter W_{ia} or $H_{a\mu}$ or not.

We'll use the partial derivatives of our shorthand $\lambda_{i\mu}$:

$$\frac{\partial \lambda_{i\mu}}{\partial W_{ia}} = C_\mu H_{a\mu}$$
$$\frac{\partial \lambda_{i\mu}}{\partial H_{a\mu}} = C_\mu W_{ia}$$

and after some pleasant differential calculus, we'll arrive at the gradient of the log likelihood:

$$\frac{\partial}{\partial W_{ia}} = \sum_\mu \left(\frac{V_{i\mu}}{\lambda_{i\mu}} \cdot C_\mu H_{a\mu} \right) - \sum_\mu C_\mu H_{a\mu}$$
$$\frac{\partial}{\partial H_{a\mu}} = \sum_i \left(\frac{V_{i\mu}}{\lambda_{i\mu}} \cdot C_\mu W_{ia} \right) - \sum_i C_\mu W_{ia}$$

A key observation: the ratio $\frac{V_{i\mu}}{\lambda_{i\mu}}$ is the ratio of observed to expected counts. As we improve our model for the expected counts, this ratio goes to 1.

We could also note that in the gradient with respect to $H_{a\mu}$ we've left C_μ's inside summations over i. They're constant with respect to i, so we could pull them outside the summation, but we left them where they are just for symmetry between the two equations in the gradient. We'll deal with them soon.

gradient ascent optimization

We've got the log likelihood and we've got its gradient, so we could use standard optimization techniques to maximize the likelihood of our W_{ia} and $H_{a\mu}$. For example if we wanted to do steepest ascent optimization, we would guess (perhaps random) initial values of all the W_{ia} and $H_{a\mu}$, then iterate small steps of improving them along their gradient:

$$W'_{ia} = W_{ia} + \Delta_{ia} \left[\sum_\mu \left(\frac{V_{i\mu}}{\lambda_{i\mu}} \cdot C_\mu H_{a\mu} \right) - \sum_\mu C_\mu H_{a\mu} \right]$$

$$H'_{a\mu} = H_{a\mu} + \Delta_{a\mu} \left[\sum_i \left(\frac{V_{i\mu}}{\lambda_{i\mu}} \cdot C_\mu W_{ia} \right) - \sum_i C_\mu W_{ia} \right]$$

But to implement this, we have to worry about what all those "small steps" Δ_{ia} and $\Delta_{a\mu}$ are. They have to be large enough that we reach a solution reasonably quickly, but small enough that we don't overshoot an optimum.

Moreover, all our W_{ia} and $H_{a\mu}$ are probabilities: they have to stay nonnegative, and they have to sum to one. If we're adding terms that might be negative, we could create a negative "probability".

Fine, we've already seen that there are various techniques for enforcing nonnegativity and normalization constraints on parameters during an optimization - such as using change of variables trickery. We could go back and rewrite our equations in terms of auxiliary unconstrained real-valued variables, and our W and H in terms of functions of those.

But it's here that the real substance of the Lee and Seung paper lurks, which isn't really described in the main paper, but instead gets relegated to an accompanying short paper. They describe a counterintuitive solution to the nonnegativity constraint.

multiplicative rather than additive updates

Lee and Seung propose a particular special case for the update step sizes Δ_{ia} and $\Delta_{a\mu}$:

$$\Delta_{ia} = \frac{W_{ia}}{\sum_\mu C_\mu H_{a\mu}}$$
$$\Delta_{a\mu} = \frac{H_{a\mu}}{\sum_i C_\mu W_{ia}}$$

Plugging those into the gradient ascent update equations gives:

$$W'_{ia} = W_{ia} \cdot \frac{\sum_\mu \frac{V_{i\mu}}{\lambda_{i\mu}} \cdot C_\mu H_{a\mu}}{\sum_\mu C_\mu H_{a\mu}}$$
$$H'_{a\mu} = H_{a\mu} \cdot \frac{\sum_i \frac{V_{i\mu}}{\lambda_{i\mu}} \cdot C_\mu W_{ia}}{\sum_i C_\mu W_{ia}}$$

The update equations are now **multiplicative**, not **additive**; and all the parameters in the multiplicative term are nonnegative (V, W, H, C), so we're always multiplying by a nonnegative number. Thus the new W'_{ia} and $H'_{a\mu}$ **must be nonnegative**.

Lee and Seung were able to prove that this update rule can never decrease the likelihood. Therefore it will reach a local optimum.

We still haven't done anything to constrain the W and H to sum to one. Lee and Seung propose an ad hoc renormalization at each update, $W''_{ia} = \frac{W'_{ia}}{\sum_j W'_{ja}}$, which seems to suffice. With the W thus constrained, normalization of the optimal H (over a) follows automatically.

et voila: figure 2

$$\boxed{\begin{gathered} W_{ia} \leftarrow W_{ia} \sum_{\mu} \frac{V_{i\mu}}{(WH)_{i\mu}} H_{a\mu} \\ W_{ia} \leftarrow \frac{W_{ia}}{\sum_j W_{ja}} \end{gathered}} \quad \boxed{H_{a\mu} \leftarrow H_{a\mu} \sum_i W_{ia} \frac{V_{i\mu}}{(WH)_{i\mu}}}$$

Figure 2 from [Lee and Seung, 1999] showing their update equations for the NMF algorithm.

As we already noted, the C_μ in the H update equation are constant with respect to a summation over i; we can pull them outside the summations in both the numerator and the denominator and they cancel.

Then note that $\sum_i W_{ia}$ term in the denominator of the H update is just one, because we defined the W_{ia} as probabilities that way. So it disappears too.

The $\sum_\mu C_\mu H_{a\mu}$ in the denominator of the W update does *not* disappear that way though. So at first glance that term looks like it has to stay in the denominator of the update equation for W_{ia}.

But! Lee and Seung's update equations forces a renormalization of the new W'_{ia} *anyway*, so the denominator of the W' doesn't matter -- so we can drop it for *that* reason.

This leaves us with the NMF update equations:

$$W'_{ia} = W_{ia} \sum_{\mu} \frac{V_{i\mu}}{\lambda_{i\mu}} \cdot C_\mu H_{a\mu}$$

$$W''_{ia} = \frac{W'_{ia}}{\sum_j W'_{ja}}$$

$$H'_{a\mu} = H_{a\mu} \sum_{i} \frac{V_{i\mu}}{\lambda_{i\mu}} \cdot W_{ia}$$

And these are *almost* the equations in Figure 2 of [Lee and Seung, 1999]. The differences are that I have $C_\mu H_{a\mu}$ where they only have $H_{a\mu}$, and I have $\lambda_{i\mu} = C_\mu \sum_a W_{ia} H_{a\mu}$ where they have $(WH)_{ia} = \sum_a W_{ia} H_{a\mu}$. This is precisely the difference between having H as probabilities (my way, with C_μ separate and explicit) or having H in units of counts (implicitly how Lee and Seung have it).

Either way, again, the key intuition here is that ratio $\frac{V_{i\mu}}{\lambda_{i\mu}}$: when we've matched our observed counts $V_{i\mu}$ to our expected counts $\lambda_{i\mu}$, this ratio goes to one, thus the whole multiplicative update term goes to one, and our W and H stop changing.

As with other iterative algorithms, we start with random parameter choices and iterate to convergence. A good measure of convergence is whether the log likelihood is still increasing. Because the algorithm is a local optimizer, not a global one, we want to try different initial guesses of W and H.

Some sources talk about this as if it is an expectation-maximization algorithm in which you alternate between updating the W and the H. If all you had to go by are the two boxed equations in Lee and Seung's Figure 2, that might seem reasonable (and it shouldn't hurt to do it that way, except to slow the algorithm down). But as we've seen in the derivation, the derivation is not an EM algorithm, it comes from gradient ascent equations in which we simultaneously update all the parameters. This isn't an EM problem where if you told me the H I'd be able to infer the W and vice versa.

and finally, in matrix notation

We've come at this from our favorite direction: first the model, then the likelihood, then turning the crank of probabilistic inference (in this case, using gradient ascent to fit maximum likelihood parameters to the data).

NMF, as you can tell from its name "non-negative matrix factorization", is usually presented in terms of linear algebra. We're looking for a "decomposition" of the $N \times M$ data matrix V in terms of a product of a $N \times R$ *basis feature* matrix W and an $R \times M$ *coefficient* matrix H:

$$V \simeq WH$$

If we have H in units of counts, then the expected counts $\lambda_{i\mu} = \sum_a W_{ia} H_{a\mu}$, and that's just matrix multiplication, defined for each individual matrix element. λ is an $N \times M$ matrix of expected counts, with $\lambda = WH$. Then the $V \simeq \lambda$ part is how we get the observed data V from the expected means λ, given Poisson count noise, which a likelihood approach forces us to be explicit about.

So the matrix multiplication emerges here just as a convenient notational shorthand -- *not* as some magical driving force in the algorithm. I did the whole derivation in terms of expected counts $\lambda_{i\mu}$, to help make the Poisson distribution explicit; Lee and Seung express these elements as $(WH)_{i\mu}$, elements of a matrix multiplication, which is the exact same thing.

The magic, if there is any, is in the $\simeq$: what do we mean, exactly, by $V \simeq WH$? That's where the actual substance is. That's where the likelihood model is, that gets you from expected data to observed data. That's where you made important assumptions. That's where you could specify a different, more appropriate, perhaps more sophisticated model. That's where you write down a log likelihood and optimize.

There's magic also in the clever multiplicative update. But a key observation is that the parameters we're trying to optimize -- the W and H -- were defined as probabilities *to begin with*. So it is absolutely nonsurprising, and indeed imperative that they're nonnegative. "Nonnegative matrix factorization" may sound mysterious, but as we've derived it, it's just another probability model.