CS 598: Theoretical Machine Learning

Lecturer: Pranjal Awasthi
Scribe: Jeremy Bierema

Lecture #18
Friday, November 10, 2017

In this lecture, we wrap up our discussion of the matrix completion problem.

1 Review

In the last lecture, we defined the matrix completion problem. We then showed what weak guarantees can be achieved.

1.1 Definition

In the matrix completion problem, we are given a matrix $M_{n\times n}$ of rank r, where the entries are bounded as $|M_{ij}| \leq M_{\max}$. We are also given a set $\Omega = \{(i_1, j_1), (i_2, j_2), \dots\}$, where each (i, j) is in Ω with probability p. And finally, we receive the matrix $P_\Omega(M)$, where $P_\Omega(M)_{ij} = \begin{cases} M_{ij}, & (i,j) \in \Omega \\ 0 & \text{otherwise,} \end{cases}$ the matrix whose sampled entries are identical to those in M and whose non-sampled entries are 0. The goal is to recover M.

1.2 Weak recovery

Last time, we saw an algorithm for doing weak (approximate) recovery, and the accompanying theorem for what it guarantees. Our algorithm was regularized SVD. We first looked at $\hat{M} = \frac{1}{p}P_\Omega(M)$ and regularized it if needed, where regularization is needed if p is small. Regularization consists of dropping dense rows and columns. We defined M' to be the regularized version of $\hat{M}$. Finally, we output the first r terms of the SVD representation, that is, we approximate M by $\tilde{M}$, the best rank-r approximation of M'.

The theorem we showed was that with high probability, $\frac{1}{n}||\tilde{M} - M||_{\mathrm{F}} \leq M_{\max} O\left(\sqrt{\frac{r}{np}}\right)$, where we typically assume $M_{\max}$ to be a small constant. This is the error we get for weak recovery.

To make this error small, we would typically need $p \approx \frac{r}{n}\log n$, so that the number of entries sampled from the matrix is roughly $rn \log n$. And in fact, these results can be generalized to cases where we are given $rn \log n$ entries from each row and column, without the assumptions about randomness. The property that we need for this to work is for the graph represented by the matrix to be an expander graph. That is, if we treat the matrix as the adjacency matrix for a graph so that each entry is nonzero iff there is an edge from the node corresponding to the entry's row to the node corresponding to the entry's column, then we want this graph to be an expander graph. What this means for the matrix (at an intuitive level) is that the entries are spread out. For example, if we had a matrix where all the nonzero entries were in the first few columns and the other columns had all zeros, that would not satisfy this property.

2 Scope of exact recovery

In this lecture, we show how to do exact recovery, rather than approximate recovery.

In practice, we do not care about truly exact recovery, but we want to be able to get the error as small as anyone wants. So there are two meanings of exact recovery, one meaning absolute exact recovery, and one that works in terms of an error parameter $\epsilon > 0$.

2.1 Hardness of exact recovery

First, we note that the approximate algorithm we described last time does not work, because there could be really bad low-rank matrices. For example, on the matrix

$$\begin{bmatrix} 1 & 0 & 0 & \cdots & 0 \\ 0 & 0 & 0 & \cdots & 0 \\ 0 & 0 & 0 & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \cdots & 0 \end{bmatrix}$$

with a single nonzero entry, the algorithm will just see all 0s with high probability, so the error will be $\frac{1}{n}$. And importantly, this is an issue with *any* algorithm. If we are given randomly sampled entries, the probability of seeing the 1 entry is $\frac{1}{n^2}$. To succeed, an algorithm would basically need to see the entire matrix, and so we would need p close to 1.

Thus, we see that we need to make some assumptions to rule out these kinds of matrices. So what is a good assumption to make? The problem in the previous example is that the useful information is localized, and so random sampling will not hit it. To fix this, we should look at matrices where information is spread out. This is the intuition behind what are called *incoherent matrices*, which we will describe somewhat more precisely after we perform some motivational setup.

2.2 Incoherent matrices

The matrix that we gave as an example of the hardness of exact recovery above is given by $e_1 e_1^{\mathrm{T}}$, where e_1 is the first standard basis vector. The standard basis vectors are examples of what we call *spiky vectors*, which are vectors where most of the concentration is located on one or a few of the coordinates. And when we take the outer product of a spiky vector with itself, we get a matrix where a majority of the mass is localized. So to restrict ourselves to matrices without localized information, we will look at matrices with non-spiky singular vectors, something that we achieve by bounding the incoherence of a matrix.

Incoherent matrices come up in many machine learning applications, not just matrix completion. You can see them used in sparse coding and in dictionary learning. They also arise in deep learning, where it appears that when the matrices describing neural net weights are incoherent matrices, we are able to give good guarantees.

We will now begin to define what an incoherent matrix is. Let $M_{n\times n} = \sum_{i=1}^{r} \sigma_i u_i v_i^{\mathrm{T}} = U\Sigma V^{\mathrm{T}}$, where

$$U = \begin{bmatrix} | & | & & | \\ u_1 & u_2 & \cdots & u_r \\ | & | & & | \end{bmatrix}$$

has as its columns the u_is, where

$$V = \begin{bmatrix} | & | & & | \\ v_1 & v_2 & \cdots & v_r \\ | & | & & | \end{bmatrix}$$

has as its columns the v_is, and where

$$\Sigma = \begin{bmatrix} \sigma_1 & 0 & \cdots & 0 \\ 0 & \sigma_2 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & \sigma_r \end{bmatrix}$$

is the diagonal matrix formed from the singular values. We restrict $||u_i||_2 = 1$ and $||v_i||_2 = 1$ so that this is the SVD of M.

To build some intuition before we actually give our definition, suppose the u_is and v_is were random unit-length vectors. We would expect typical coordinates of u_i to be roughly $\frac{1}{\sqrt{n}}$. If we understand that, now suppose alternatively that U and V were random orthonormal bases. We would expect the typical row norm to be roughly $\sqrt{\frac{r}{n}}$. Incoherent matrices are ones with this latter property. More precisely, M is said to be μ-incoherent if $\forall i$, $||e_i^{\mathrm{T}} U||_2^2 \leq \frac{\mu r}{n}$ and $\forall j$, $||e_j^{\mathrm{T}} V||_2^2 \leq \frac{\mu r}{n}$. Decoding this notation, $||e_i^{\mathrm{T}} U||$ is just the row norm of the ith row of U. So the condition is that these norms are close to what we would expect for the random case.

It turns out that this is a very strong condition. When it holds for a matrix, it implies strong structure on the matrix.

In the case that U and V each give a random orthonormal basis, $M = U\Sigma V^{\mathrm{T}}$ is $O(\log n)$-incoherent. This is the range that we will mainly be interested in, where μ is either constant or $O(\log n)$. It should be noted that every matrix is incoherent for some value of μ, though it might be large. We are interested in those with small μ, since their singular vectors will be non-spiky.

In our discussion we mentioned a random orthonormal basis. An orthonormal basis will not have independently chosen basis vectors. We can think of it as giving an r-dimensional feature space. The incoherence property we have defined is saying that all coordinates within this feature space are important. In a setting with r chosen appropriately, all coordinates will contain important information, and so we cannot delete a feature without losing data. This is an acceptable assumption, since we are free to make r smaller if there are some coordinates without important information.

3 Applicability of strategies

We mention two algorithms for performing exact recovery for incoherent matrices. The first uses semidefinite programming. This strategy allows us to achieve the right value of p. The second strategy is that of alternate minimization. This strategy is more practical, in particular, faster. However, it achieves a higher value of p rather than the best possible. As it turns out, alternate minimization is a strategy that is useful in many machine learning problems.

3.1 Semidefinite programming

The theorem we can prove for the semidefinite programming (SDP) approach is that if $p \geq \frac{\mu^2 r}{n} \log^2 n$, then with high probability, SDP outputs $\tilde{M}$ such that $\tilde{M} = M$, assuming $|M_{ij}| \leq 1$. Note that here $\tilde{M} = M$ is exact equivalence, not approximate as before. If we leave off the condition $|M_{ij}| \leq 1$, then the maximum $|M_{ij}|$ value factors into the bound on p.

In expectation, the number of entries we need to see is roughly $\mu^2 r n \log n$.

3.2 Alternate minimization

In the alternate minimization (AM) approach, we need a slightly stronger assumption. Our theorem is that if $p \geq \frac{\mu^2 r}{n} \log^2 n (\frac{\sigma_1}{\sigma_r})^2 \log^2 \frac{1}{\epsilon}$, then with high probability, AM outputs $\tilde{M}$ such that $||\tilde{M} - M||_{\mathrm{F}} \leq \epsilon$. We do not need to assume $|M_{ij}| \leq 1$ here, since we are already paying in the $(\frac{\sigma_1}{\sigma_r})^2$ term.

p is higher here than in the previous theorem. The most significant difference is the $(\frac{\sigma_1}{\sigma_r})^2$ factor. The ratio $\frac{\sigma_1}{\sigma_r}$ is called the condition number of a matrix. This factor is inherent to the AM technique and cannot be avoided. If it is problematic, the solution is to use SDP instead.

AM almost always solves examples in practice. This is because in most applications, information is concentrated in the first few singular values, and after these first few, the singular values decay very rapidly. So for example, we could have σ_1, σ_2, and σ_3 being significant and then σ_4, σ_5, and so on all being very close to 0.

Both of the bounds that we have given are tight. Note that we have made no assumption on the incoherence of the matrix; the incoherence of the matrix factors into the bound on p.

4 Algorithms

Now that we know the bounds, we want to know what the algorithms are and how to analyze them. We will frame the discussion by first looking at the ideal problem and how to solve it, and subsequently by relaxing it to give tractable solutions.

4.1 Ideal optimization

For approximate recovery, the ideal solution was to minimize the Frobenius norm of the error matrix. For exact recovery, the optimal solution is the minimized rank representation of the matrix. We want to find $X_{n \times n}$ such that $P_\Omega(X) = P_\Omega(M)$ while minimizing $\mathrm{rank}(X)$.

As an optimization problem, the constraints are $x_{ij} = m_{ij}$. The objective function is matrix rank. This is non-convex, and in general, optimizing non-convex functions is NP-hard.

4.2 Semidefinite programming

The SDP solution to this apparent hardness is to replace matrix rank by a convex function and optimize that. Recall that the rank of a matrix X is the number of nonzero singular values. Minimizing this means finding a matrix with the number of nonzero values as small as possible. SDP replaces this with the nuclear norm $||X||_* = \sum_{i=1}^{n} \sigma_i(x)$. One can think

of this as using the ℓ_1 norm of the vector of singular values instead of the ℓ_0 "norm." This is normally not a bad replacement.

So the SDP solution is to minimize $||X||_*$ subject to $P_\Omega(X) = P_\Omega(M)$. We are optimizing a relaxation of the original problem.

Why is this objective function convex? In fact, it is not obvious. It turns out that it is possible to show that $||X||_* = \sup_{Q:||Q||_2 \leq 1} \langle X, Q \rangle$, where $\langle X, Q \rangle$ denotes the dot product of the two matrices treated as n^2-dimensional vectors. If two norms satisfy this equation that the nuclear norm and the spectral norm are here stated to satisfy, they are called duals of each other. So the nuclear norm is the dual of the spectral norm.

Proving that this convex program solves exact recovery involves some nice math and beautiful techniques, especially if you want to get the right bounds.

4.3 Alternate minimization

However, what we will study in this lecture is the alternate minimization algorithm.

We will first see what relaxation AM makes relative to the ideal problem. If the true matrix is X, we want to enforce that the matrix we output is low-rank or close to low-rank. The SDP solution was to replace the rank by the sum of the singular values in the objective function. In AM, we write $X = \underset{n \times r}{U} \cdot \underset{r \times n}{V}^{\mathrm{T}}$. Here the variables are entries of U and V, not entries of X. We do not need to worry about minimizing the rank, since by this representation, it is forced to be r. We just need to enforce $P_\Omega(X) = P_\Omega(M)$. AM uses a smooth version of this, in which we minimize $\sum_{(i,j) \in \Omega} \big(M_{ij} - (UV^{\mathrm{T}})_{ij}\big)^2$. However, this is still not convex, since the $(UV^{\mathrm{T}})_{ij}$ part involves products of variables.

The idea of AM is that if someone told us what U should be, the problem becomes convex. It is a standard ℓ_2 regression problem, and standard techniques can be used to minimize over V.

So the algorithm is to start with some U_0 and V_0, and then for $t = 1, 2, \ldots$, compute first $U_t = \arg\min_U ||P_\Omega(M - UV_{t-1}^{\mathrm{T}})||^2$ and then $V_t = \arg\min_V ||P_\Omega(M - U_t V^{\mathrm{T}})||^2$. More informally, we start with some (perhaps arbitrary) guess for U and V. We fix V and then update U, where this problem is a squared loss setup. Then we fix U and update V. We repeat this process.

It is a simple algorithm. It is easy to describe and implement. And there are good algorithms for least squares kinds of problems. This is one of the central insights in machine learning: simple iterative algorithms do surprisingly well. A big challenge of research then is to characterize when/why they do well. Now for matrix completion, we have some idea of why AM works.

Now that we have specified the algorithm, we can state a more complete version of our theorem for AM, where we can specify the (very fast) convergence speed. If $p \geq \frac{\mu r}{n} \log^2 n (\frac{\sigma_1}{\sigma_r})^2 \log^2 \frac{1}{\epsilon}$, then after $T = \log \frac{1}{\epsilon}$ iterations (where each iteration includes an update to U and an update to V) we get $||M - U_T V_T^{\mathrm{T}}||_{\mathrm{F}} \leq \epsilon$ provided U_0 and V_0 are initialized appropriately. This care in initialization is the caveat. Typically we want a fast, cheap way to initialize. Usually this comes from the weak SVD guarantee.

So if this method does not give the best p, where do we lose out? It is in our optimization method. We do not know how to optimize properly.

5 Alternate minimization analysis

The proof we will give will reveal the general ideas of the full proof from a special case: rank-1 matrices. The general case requires more care.

So we have $M = \sigma_1 v v^{\mathrm{T}}$. If the matrix is μ-incoherent, this implies that every entry of v has to be small, since every row norm has to be small and the vector of squared row norms of vv^{T} is just v itself. So we get $|v_i| \leq \sqrt{\frac{\mu}{n}}$, which is what you would expect if we were choosing randomly. We will try to prove the theorem for this rank-1 matrix.

There are two steps to the analysis: initialization, and then alternate minimization.

5.1 Initialization

The first step is initialization. We use SVD, outputting the rank-1 term. The theorem is that SVD guarantees that with high probability, $||v_0 - v|| \leq \frac{\mu}{\sqrt{\log n}}$. This by itself is already a pretty good guarantee.

We have already proved this sort of bound a couple of times, so there is little point in proving it again. It is proved using the same random matrix bounds based on the bound on p. We use Davis–Kahan to get $||v_0 - v|| \leq \frac{\mu}{\sqrt{\log n}}$ from $||\frac{1}{p}P_\Omega(M) - M||_2 \leq \frac{\sigma_1 \mu}{\sqrt{\log n}}$.

5.2 Updates

The interesting part is to analyze the updates. Note that because we are proving this special case, we only have one update during each iteration of AM. We can replace the second part of the update with $v_t = \frac{u_t}{||u_t||}$. Note that we will analyze the rank-1 case in such a way as to bring out the strategies used in the general analysis, even though it is possible to analyze the rank-1 case more directly.

We want to show that as we update v_0 it is converging to the right vector, v. It is already $\frac{\mu}{\sqrt{\log n}}$-close. We want to show that this distance is getting smaller, as small as we want. To demonstrate this, let us decompose v_{t-1} as $(v_{t-1} \cdot v)v + v_\perp^{t-1}$, where $(v_{t-1} \cdot v)v$ is the component of v_{t-1} parallel to v and $v_\perp^{t-1}$ is the component perpendicular to v. We will show that $||v_\perp^t|| \leq \frac{1}{4}||v_\perp^{t-1}||$. If we establish this, then we are done; the perpendicular component is decreasing geometrically. The rest of the mass is going in the right direction, parallel to v.

We can write the objective function in explicit form as

$$u_t = \arg\min_u \sum_{(i,j)\in\Omega} (M_{ij} - u_i v_{t-1,j})^2.$$

Denote the argument, $\sum_{(i,j)\in\Omega}(M_{ij} - u_i v_{t-1,j})^2$, by L. How should we minimize this? Since it is convex, to minimize it, set the derivatives to 0. We get

$$\frac{\partial L}{\partial u_i} = 0 \implies \sum_{j:(i,j)\in\Omega} 2(M_{ij} - u_i v_{t-1,j})(-v_{t-1,j}) = 0$$

for all i. This explicitly gives what u_i will look like:

$$u_i = \frac{\sum_{j:(i,j)\in\Omega} M_{ij} v_{t-1,j}}{\sum_{j:(i,j)\in\Omega} v_{t-1,j}^2}.$$

Let us try to write this in a nicer form. We can divide the numerator and denominator by p, and then add and subtract identical terms:

$$u_i = \frac{\sum_{j:(i,j)\in\Omega} \frac{M_{ij}}{p} v_{t-1,j}}{\sum_{j:(i,j)\in\Omega} \frac{v_{t-1,j}^2}{p}} + \sum_j M_{ij} v_{t-1,j} - \sum_j M_{ij} v_{t-1,j}.$$

Note that the added summations are over all j. With a little bit of rearrangement, we get

$$u_i = \sum_j M_{ij} v_{t-1,j} + \frac{\sum_{j:(i,j)\in\Omega} \frac{M_{ij}}{p} v_{t-1,j} - (\sum_{j:(i,j)\in\Omega} \frac{1}{p} v_{t-1,j}^2)(\sum_j M_{ij} v_{t-1,j})}{\sum_{j:(i,j)\in\Omega} \frac{v_{t-1,j}^2}{p}}.$$

If we let $B_i := \sum_{j:(i,j)\in\Omega} \frac{v_{t-1,j}^2}{p}$, then we can write this as

$$u_i = \sum_j M_{ij} v_{t-1,j} + \frac{\sum_{j:(i,j)\in\Omega} \frac{M_{ij}}{p} v_{t-1,j} - B_i(\sum_j M_{ij} v_{t-1,j})}{B_i},$$

and we can rewrite the sums as the ith components of vectors:

$$u_i = (Mv_{t-1})_i + \frac{\Big(\big(\frac{1}{p} P_\Omega(M) - MB_i\big) v_{t-1}\Big)_i}{B_i}.$$

We have an equivalence between the ith coordinates of two vectors, so the entire vectors must be equal:

$$u_t = Mv_{t-1} + B^{-1}\Big(\frac{1}{p} P_\Omega(M) - MB\Big) v_{t-1},$$

where B is the diagonal matrix

$$\begin{bmatrix} B_1 & 0 & \cdots & 0 \\ 0 & B_2 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & B_n \end{bmatrix}.$$

If the $B^{-1}\big(\frac{1}{p} P_\Omega(M) - MB\big) v_{t-1}$ term were missing, this would be a familiar expression. Recall the power method from the SVD lecture. We start with a random vector, and we keep iteratively multiply a matrix by it and normalize. This converges to the first singular vector.

The extra term here is a noise component. So AM can be thought of as a noisy power method. This idea can be formalized in more detail. To analyze AM, then, we need to get a handle on what the noise looks like. Identifying this decomposition makes the problem much easier to analyze.

So if we want to use the power method, let us try to understand the noise. If we add and subtract M inside the parentheses, we can write the noise as the sum of two components, $B^{-1}\big(\frac{1}{p} P_\Omega(M) - M\big) v_{t-1} + B^{-1} M(I - B) v_{t-1}$. We have given arguments before that the spectral norm of $\frac{1}{p} P_\Omega(M) - M$ is small. This implies that the whole first term of this is in fact small.

To understand the second term, let us try to understand B. Since each j is paired with i in Ω with probability p, we have $E[B_i] = \sum_j p \frac{v_{t-1,j}^2}{p} = \sum_j v_{t-1,j}^2 = 1$, where the last

equality holds since we defined v_{t-1} to have unit length. So $E[B] = I$. After this we want to use concentration bounds to show that B is in fact close to I. In expectation, $I - B$ is quite small.

Time is short, but the conclusion is that the component of v_t orthogonal to v comes from these two noise terms, so if we argue that they are small, our power method works.

A link to the completed argument will be distributed online. The key point of this discussion is that AM is a noisy power method, and it is analyzed as we have been analyzing the other problems in this module, using bounds on random matrices, generated here as the difference between a sampled matrix and the true matrix.