COLT: Linear Algebra in Machine Learning

Lecturer: Ravi Kannan Scribe: Kiran Vodrahalli

Contents

1 Introduction 1

2 Low-rank Approximation with Additive Error 2
2.1 Sketch of a matrix . . . 3

3 Continuing AA^T Story 4

4 Spectral sparsifiers of AA^T 4

5 Randomized Algorithm for general tensors 5

6 The random sign matrix 5
6.1 Distributed Data . . . 5

7 Answers to some questions from audience 6

1 Introduction

Ravi Kannan was one of the pioneers of random methods, the real shocker was approximating the volumes of convex bodies, he's worked on spectral methods, sketching, linear algebraic methods, and a wonderful book on the foundations of Data Science.

I'll give a high-level picture of the use of linear algebra. I'm impressed the amount of detail this audience can absorb online. I have one or two semi-complete stories, and otherwise it's a survey.

Let's start with a flashy intro. If I went back 25 years and asked a randomed theory person which methods would get scaled and applied to problems in 25 years: SVD and other linear algebra algorithms, network flows, shortest paths, other graph algorithms; sophisticated data structures; optimization – the answer would often have been the middle two in the 90s. But what has panned out is (1) and (4). The most importance reason for this: So many decades of continuous mathematics and developments in randomized algorithms. The theory community rather more focused on discrete than continuous. Machine learning stuff has turned out to be very important. Working on discrete problems you're handicapped! There's a lot more you can draw on for continuous techniques.

I'm going to focus a lot on randomized algorithms; this helps quite a bit. It helps in discrete problems too, but not as much.

As for SVD in learning, it is widely used (topic modeling, clustering, etc.). It's folklore that if you have amixture of spherical Gaussians and you want to find the subspace spanned by the means, the SVD subspace is the subspace of component means. So SVD helps you find that space, then you project to it and do some algorithm. Non-negative Matrix Factorization (NMF) is similar in spirit to SVD, but you have non-negative values in the factors.

For people in theory, algorithms can toss coins (randomized algorithms), but the data is worst-case. You want to assert expected running time for the worst case input is at most something.

The other case is the data tosses coins, the algorithm may be deterministic, this is average-case analysis. Here we'll focus on (1) more than (2). Random sampling will be used for very large matrices. They tend to be peculiar matrices for a specific model, so it better work for the worst case - we don't care about many random matrices.

Why randomized algorithms? Modern data matrices are massive, and you want to sample to reduce size to save on time. Space is also important. You assume that the data doesn't fit into random-access memory (this is big data). A simple form of random algorithm computes on samples of the rows/columns of a matrix. You need a proven error guarantee, and secondly, you must be able to sample quickly. It defeats the purpose if the sampling itself takes more time or space than the original problem. We'll be slightly more relaxed and allow multiple passes, not just one pass. Towards the end, we'll see situations where the data is distributed among various servers. Randomization will help reducing communication between servers. There are two scenarios: The entire matrix exists somewhere, and the algorithm draws samples from the existing matrix. In the other situation, only a sample of entries may be known (i.e. Netflix challenge). So you in fact only have a sample of the matrix. You'd think that the same methods apply to both, but you need to know something about the probability distribution via which things were chosen. If it were adversarial, not as much you can do. So we pretend the matrix is somewhere and we're drawing samples. You have an $m \times n$ data matrix A with n, m large. You do s i.i.d. trials, pick a random column of A and scale it. Then we throw away the big matrix and only compute the sampled and scaled $n \times s$ matrix.

I won't talk about every problem every time. The first and simplest problelm is matrix multiplication. We will talk about calculating AA^T; this is a very nice question. This is one of our stories. More generally we want to multiply AB. We also want to do low-rank SVD. We'll look at matrix sketches (compact representations of a matrix). We'll look at graph sparsification, linear regression, and the spectral norm of tensors. Now recall there's no free lunch, samples can only get approximate answers. We want to pin down what the approximation is. We'll prove bounds.

2 Low-rank Approximation with Additive Error

We seek a rank k approximation A^* to A with $\|A - A^*\|_F \leq$ best possible $+ \epsilon\|A\|_F$. The first is the best possible error, due to SVD, and the second error is due to sampling.

The first theorem is

Theorem 2.1. *$s = \text{poly}(k/\epsilon)$ samples are sufficient provided that sampling is done with probabilities proportional to the* ***squared length*** *of the columns.*

Now, this is only interested if $\epsilon\|A\|_F <$ best possible SVD error. This holds for PCA matrices. This is from the paper "Length-squared sampling" by Frieze, Kannan, Vempala (1998). There are many improvements via Drineas, Mahoney, Sarlos, Deshpande, Rademacher.

An alternative scheme is to draw a sample of entries, set others to zero. Sparsity gain rather than reduction in dimensions (Achlioptas, McSherry).

Why length-squared? We have $AA^T = \sum_j \mathrm{col}_j(A)\mathrm{row}_j(A)$. You can estimate the whole sum by taking a sample of j and summing them. So you do i.i.d. trials. Each trial is not uniform. What are good p_j (number of samples to choose)? It turns out that uniform sampling is no good (E.g. all but one column of A is all zeros). An unbiased estimator $X = \frac{1}{p_j}\mathrm{col}_j(A)\mathrm{row}_j(A)$. By calculus length-squared minimizes the variance of this estimator. With an average of s samples:

$$\mathbf{E}[\|AA^T - \hat{AA^T}\|_F] \leq \frac{\|A\|_F^2}{\sqrt{\epsilon}}$$

We'll see in a minute how we actually know the lengths. This whole thing holds if we know p_j with constant factors. In general, the data handling picture is the data is too large to be stored in RAM. We measure RAM time, space and the number of passes (the way to access the matrix: a sequential read of the entire matrix). You'd rather not read through the whole matrix too many times though. So the first path, you can compute probabilities by taking sum of squares of each row / column, and the second pass you sample. So in RAM you only hold a sample number. If you have a picture of the matrix you can do it with one pass, but we won't go into that.

An approximate low-rank approx. can be caried out even in the streaming model. Also, one can first do length-squared sampling to pick s columns, then again do length-squared smapling to pick s rows to form $s \times s$ matrix and only use $O(1)$ RAM space. This proof gets more complicated.

2.1 Sketch of a matrix

How do we sketch a matrix? Suppose we form a sample of rows; can't form a sketch in this way since don't know anything about other rows. But we can sample rows and sample columns, then this gives a complete sketch of the matrix. Suppose A has rank k, small. Then a sample of $100k$ rows should pin down the row space of A. But still don'tknow for an unsampled row what linear cominbation it is. Suppose we pick a sample of $100k$ columns, then if the rows are in general position, then each row should be a unique linear combination, so this pins down linear combinations.

Theorem 2.2. *Length-squared samples of rows and length-squared sample of columns suffices to form a sketch of the matrix $A \in \mathbb{R}^{m\times n}$. C is $m \times s$ matrix formed by sampling/scaling the columns of A. R is a $\sqrt{s} \times n$ is formed by sampling $\sqrt{s}$ rows according to length-squared. Given just C, R, we can find a $s \times \sqrt{s}$ matrix U such that*

$$\boldsymbol{E}[\|A - CUR\|_2^2] \leq \frac{c\|A\|_F^2}{\sqrt{s}}$$

You can't get the error in terms of the spectral norm. This kind of thing fails for high-rank matrices like identity which cannot get anything good. We can say something for the Frobenius norm too (something else), but we talk about the spectral norm here. There are a lot of papers by Bourtsides and Woodruff (2015). *All this is saying is that a sample of rows and sample of columns is good enough.*

Traditional SVD, given data matrix A, finds best rank k approximation A_k to A. Two issues are computation time and the fact it's not interpolative.

3 Continuing AA^T Story

Say you have a probability distribution P on $\mathbb{R}^d$ with mean 0. The variance-covariance matrix is $M : M_{ij} = \mathbf{E}_P[x_i x_j]$. A is a $d \times \infty$ matrix with each column a sample weighted by probability. The variance along v is $v^T AA^T v = |v^T A|^2$.

Now our question is the sample complexity: How many i.i.d. samples according to P suffice to estaimte variance to relative error along every direction?

We want to sample a finite, scaled submatrix B of A so that for all v, $|v^T B| = (1 \pm \epsilon)|v^T A|$. So you want a direction along which the error is very small. It becomes difficult to do this if the entire variance lies in one plane.

Previously we saw that length-squared sampling gives a certain error. Rudelson and Vershynin using a beautiful technique of "decoupling" from probability and functional analysis. They proved that

$$\mathbf{E}[\|AA^T - \text{Estimate}\|_2] \leq \frac{c\|A\|_F\|A\|_2}{\sqrt{s}}$$

Now there are some works by Tropp which use the matrix Hoffding-Chernoff inequalities which give you this result in a more elementary way. For log-concave probability densities on $\mathbb{R}^d$, $O(d)$ samples suffice (Pisier and Bourgain).

4 Spectral sparsifiers of AA^T

Spielman and Srivatsava get a graph G with n nodes and m edges, where A_G is a node-edge incidence matrix. We want a sparser sub-graph H so that all cuts are approximately right (Karger). A stronger condition than cuts is the same question as approximating variance-covariance matrix from before. Rudelson-Vershynin implies that $||v^T A_G|^2 - |v^T A_H|^2| \leq \frac{cb\|A\|_2^2\|v\|^2}{\sqrt{s}}$.

Now take W the $n \times n$ left pseudo-inverse of A: WA is an isometry on the column space of A. Sample columns of A according to length-squared probabilities from WA. Let p_j be the length squared of col j of WA and divide by $\|WA\|_F^2$. Repeat this s times with probability p_j and scale by $1/p_j$ to form $n \times s$ matrix B. Then for all v, $|v^T B|^2 = (1 \pm \frac{\sigma\sqrt{n}}{\sqrt{s}})|v^T A|^2$ by Rudelson and Vershynin. Then you can sparsify for any matrix A.

But computing p_j involves finding W which is time consuming. Spielman and Srivatsava showed that for Laplacian matrices this can be done in linear time. For graphs, these results seem to be electrical resistances. An open question is are there other classes of interesting matrices for which p_j can be computed easily?

Pre-conditioned length-squared sampling leverage scores. Rudelson-Vershynin theorem can be used to assert that $O^*(\text{rank}(A))$ samples suffice. Can we save on rank(A)? Yes. First if we can do SVD to find A_k, the best rank k approximation to A then use pre-conditioned length-squared probabilities of A_k, we can do with $O^*(k)$ samples. Drineas, Mahoney and Muthukrishnan say that with poly(k/ϵ) sample columns of A drawn according to pre-conditioned length-squared probabilities on A_k, we can get an interpolative approximation A' to A with the following relative error: $\|A - A'\|_F \leq (1+\epsilon)\|A - A_k\|_F$. An alternative way to get the same theorem: Draw a sample $r = \text{poly}(k/\epsilon)$ sample columns of A with probabilities proportional to the square of the volume of the simplex spanned by them. Deshpande, Rademacher and Vemapala did Volume Spanning via Determinental processes. Suppose you'er searching for "jaguar" - there are two senses, the animal and the car. The car is more often.

If you did length-squared sampling you'd get the car a lot. The trouble with this algorithm is that it may take n^r time. There was a linear time algorithm yesterday now!

5 Randomized Algorithm for general tensors

Suppose A is $n_1 \times n_2 \times \cdots \times n_r$ symmetric. We want to maximize $\sum_{i,j,k,\ldots} A_{ijk\ldots} x_i x_j x_k \ldots$ over unit length vector x.

Theorem 5.1. *For any fixed $\epsilon > 0$, can find in polynomial time a y satisfying*

$$\sum_{i,j,k,\ldots} A_{ijk\ldots} y_i y_j y_k \ldots \geq \text{MAX possible} - \epsilon\|A\|_F$$

The algorithm requires length-squared sampling (Kannan and Vempala).

6 The random sign matrix

A nice result in the last few years. Let's look at a particular matrix called the count-sketch matrix. It has a single non-zero entry in each column which is ± 1 with probability 1/2 each in a random row. This was studied in the context of streaming. Clarkson and Woodruff in STOC (2013) (won best paper) showed that for A any $m \times n$ matrix, $m >> n$. Then S is a $t \times m$ count-sketch matrix with $t = \text{poly}(n/\epsilon)$ independent of m. With high probability simultaneously for all $x \in \mathbb{R}^n$. Then $|SAx| = (1 \pm \epsilon)|Ax|$. SA can be computed in linear time. Then LRA just on SA suffices. Ss are subspace embedding matrices. They're preserved for every x.

Another paper by Clarkson and Woodruff showed that low-rank approximation, regression, matrix multiplication, etc which is linear in the number of non-zeros. Many problems can be done like this. There's another paper by Clarkson and Woodruff in the space optimal streaming algorithm. Last year, there were optimal algorithms for CUR.

6.1 Distributed Data

Now, how to deal with distributed data? Suppose r servers each has an $m \times n$ matrix with $m >> n$, server i has matrix $A^{(i)}$. We want to compute with $A = A^{(1)} + A^{(2)} + \cdots + A^{(r)}$. We want to compute with a random projection of A (use PA where $P \in \mathbb{R}^{s \times m}$ random matrix). Server i can find $PA^{(i)}$ locally and communicate this $s \times n$ matrix to a central processer. Communication is $O(snr)$ avoiding m except servers need to compute on the same P, which is random! So you need $O(smr)$. Alon, Mataias and Szegedy suggests the use of pseudo-random P. Then in this context, Kane, Mekha, Nelson let x be a fixed vector, then to get $|PAx| \approx |Ax|$, you only need $O(\log n)$-way independence. Need to communicate only the $O(\log n)$ bit-seed to all servers. To ensure $|PAx| \approx |Ax|$ for all $x \in \mathbb{R}^n$, poly(n)-way independence suffices.

7 Answers to some questions from audience

For linear regression, you project using these methods and then do regression in the lower dimensional space. One of the big open questions is to get a $\log(1/\epsilon)$ dependence using these random methods. It's not known that we can't do it.