COS 598D: Overcoming intractability in machine learning.
Sanjeev Arora, Princeton University. Spring 2015.

Lecture 1: A whirlwind survey of machine learning and ML theory.

Various meanings of learning. Usual assumption: data consists of iid samples from some distribution. (Philosophical aside: De Finetti's theorem, exchangeability.)
A running example in this lecture will be linear classifiers.

- Unsupervised vs Supervised.

  Unsupervised: Unlabeled data. Usually need some kind of model for how the data was generated (the "story") and then recover the model parameters. Examples: k-means and other forms of clustering, loglinear models, bayes nets, ..
  Often NP-hard.

  Supervised: Training data is labeled by a human (labels could be binary, or in [1..k]). Algorithm needs to predict labels on future data. Examples: Decision trees, SVMs, k-NN.

  Generalization bounds: connect performance on training data with that on unseen data (i.e. the entire distribution).

  Rough idea: Suppose there is a classifier that is representable with M bits and has error at most \epsilon on the full distribution. Then if the training set has size at least f(M + K, \epsilon), then any classifier describable in K bits that has error at most \epsilon on the training set will have error at most 2\epsilon on the entire distribution. Furthermore, if any classifier has error at most \epsilon/2 on the entire distribution, it has error at most \epsilon on the sample. (Thus the method is complete and sound: if there exists a good classifier, it can be found by examining a small set of training points, and conversely every classifier that is good enough on the samples is also good enough for the entire distribution.)

  Proof sketch: Chernoff bounds. Only 2^M classifiers that can be represented using M bits. If any M-bit classifier has error more than 2\epsilon fraction of points of the distribution, then the probability it has error only \epsilon fraction of training set is $< 2^{-M}$. Hence whp no bad classifier can be good on the training points.

  There is a more general theory for computing the number of training points that involves VC dimension. Pls see online sources.

  The classical philosophical principle Occam's razor is related to this.

  Example of training: Perceptron algorithm for linear classifier. (Can think of as a way to determine weights of features.) Completely nonbayesian description.

Can also turn into convex program using the notion of a margin.

- Discriminative vs Generative.

  Discriminative: Only know P(label| data). (Example 1: label = linear threshold corresponds to SVM. Note this is deterministic. Example 2: Logistic regression. Smoothened version of SVM.) Examples of Discriminative learners: decision trees, SVMs, kernel SVMs, deep nets, logistic regression etc. SVMs and logistic regression can be solved by convex optimization.

  Generative: Know an expression for P(label, data). Estimate P(label|data) by Bayes rule and calculating P(label, data)/P(data). For an example see the application to spam classification in Lecture notes by Cynthia Rudin on Naïve Bayes.

  Also see the chapter in Mitchell's book, which shows that logistic regression corresponds to a naïve bayes estimator where coordinates are iid Gaussian.

  For a more hands-on viewpoint with worked out examples, see Chris Manning's lecture notes.
  https://web.stanford.edu/class/cs124/lec/Maximum_Entropy_Classifiers.pdf

- (Aside: Logistic regression is great right? How about if we stack up logistic regression units in a circuit? We get deep nets. Training these is a nontrivial task and we only know of heuristic algorithms. We don't know of a good bayes interpretation of such deep net classifiers.)

- There can be advantages to each. Discriminative needs fewer assumptions. Generative is easier to adapt to semisupervised settings.

  See

  Relevant chapter on generative vs discriminative in Tom Mitchell's book.

  On discriminative vs generative: A comparison of logistic regression and naïve bayes, by Ng and Jordan in NIPS 2001.

  Generative or Discriminative? Getting the best of both worlds by Bishop and Lasserre. Bayesian Statistics 2007.

- *Regularization*. Technique used to avoid overfitting. For this it is better to use a less complicated solution, and adding a regularizer to the objective can help with this. (related to the generalization theory; a rough analogy is to restrict yourself

to solutions that can be described with fewer # of bits. This is only a rough intuition)

We will focus a lot on unsupervised learning.

Max Likelihood and Maximum Entropy Principle.

Given a choice between many possible distributions that fit the data, pick the one with maximum entropy.

Example: We are given a die that, when thrown, produces an expected value 4.7. What is the chance that it produces 5? Solution: Let
p_i = prob it produces i. Then average of i p_i is 4.7. Compute values of p_i 's that maximize entropy subject to this average.

Example 2: If we are only given the mean of a distribution, the max entropy distribution consistent with this is the exponential. (If the variable is n-variate, the distribution is loglinear.)

Example 3: If we are only given the mean and the covariances of a distribution then the max entropy distribution consistent with that is the gaussian.

Max likelihood method

Find the parameter vector Theta that maximizes the likelihood of seeing the data.

(Aside: Amazingly, Shannon in 1948 also invented NLP in addition to information theory by describing n-gram models for languages and suggesting measuring them using his entropy measure, which we can think of as max likelihood.
http://cm.bell-labs.com/cm/ms/what/shannonday/shannon1948.pdf)

Example: Max log likelihood expression for logistic regression from
http://ufldl.stanford.edu/wiki/index.php/Softmax_Regression

Recall that in logistic regression, we had a training set $\{(x^{(1)}, y^{(1)}), \ldots, (x^{(m)}, y^{(m)})\}$ of m labeled examples, where the input features are $x^{(i)} \in \Re^{n+1}$. (In this set of notes, we will use the notational convention of letting the feature vectors x be $n + 1$ dimensional, with $x_0 = 1$ corresponding to the intercept term.) With logistic regression, we were in the binary classification setting, so the labels were $y^{(i)} \in \{0, 1\}$. Our hypothesis took the form:

$$h_\theta(x) = \frac{1}{1 + \exp(-\theta^T x)},$$

and the model parameters θ were trained to minimize the cost function

$$J(\theta) = -\frac{1}{m}\left[\sum_{i=1}^{m} y^{(i)} \log h_\theta(x^{(i)}) + (1 - y^{(i)}) \log(1 - h_\theta(x^{(i)}))\right]$$

This expression is convex, and so gradient descent methods find the best fit very fast. In fact, this computational ease is the reason for the popularity of logistic regression ---this popularity dates back to pre-computer days when people were using slide rules etc.

Unfortunately, when you compute the log likelihood for most other settings, the expression turns out to be nonconvex. Such nonconvex optimization often turns out to be NP-hard (as has been proved for many settings).

Simple example: mixture of spherical gaussians of the same radius. Maximising the log likelihood is tantamount to k-means clustering.

(From Arora-Kannan: Learning Mixtures of Separated Nonspherical Gaussians; https://www.cs.princeton.edu/~arora/pubs/gaussians.pdf)

Trying to overcome this intractability is a major goal in this course.

4. Max-likelihood estimation. Now we describe an algorithm for max-likelihood fit of a mixture of k spherical Gaussians of equal radius to (possibly) unstructured data. First we derive a combinatorial characterization of the optimum solution in terms of the *k-median* (*sum of squares*, *Steiner version*) problem. In this problem, we are given M points $x_1, x_2, \ldots, x_M \in \Re^n$ in $\Re^n$ and an integer k. The goal is to identify k points $p_1, p_2, \ldots, p_k$ that minimize the function

$$\sum_{i=1}^{M} |x - p_{c(j)}|^2, \tag{26}$$

where $p_{c(j)}$ is the point among $p_1, \ldots, p_k$ that is closest to j and $|\cdot|$ denotes Euclidean distance.

THEOREM 13. *The mixture of k spherical Gaussians that minimizes the log-likelihood of the sample is exactly the solution to the above version of k-median.*

PROOF. Recall the density function of a spherical Gaussian of variance σ (and radius $\sigma\sqrt{n}$) is

$$\frac{1}{(2\pi\sigma)^{n/2}} \exp\left(-\frac{|x-p|^2}{2\sigma^2}\right).$$

Let $x_1, x_2, \ldots, x_M \in \Re^n$ be the points. Let $p_1, p_2, \ldots, p_k$ denote the centers of the Gaussians in the max-likelihood solution. For each data point x_j let $p_{c(j)}$ denote the closest center. Then the mixing weights of the optimum mixture $w_1, w_2, \ldots, w_k$ are determined by considering, for each i, the fraction of points whose closest center is p_i.

The log-likelihood expression is obtained by adding terms for the individual points to obtain

$$-\left[\text{Constant} + \frac{Mn}{2}\log\sigma + \sum_j \frac{|x_j - p_{c(j)}|^2}{2\sigma^2}\right].$$

The optimum value $\hat{\sigma}$ is obtained by differentiation,

$$\hat{\sigma}^2 = \frac{2}{Mn}\sum_j |x_j - p_{c(j)}|^2, \tag{27}$$

which simplifies the log-likelihood expression to

$$\text{Constant} + \frac{Mn}{2}\log\hat{\sigma} + \frac{Mn}{4}.$$

Thus the goal is to minimize $\hat{\sigma}$, which from (27) involves minimizing the familiar objective function from the sum-of-squares version of the k-median problem. □

princeton univ. Spr'15 cos 598D: Overcoming intractability in machine learning

Lecture 2: Linearized models and SVD

Lecturer: *Sanjeev Arora* Scribe:

1 Concluding thoughts about last time and segue to today's topics

The dominant model for supervised learning from a theory viewpoint is SVM or Kernel SVM. We discussed SVM's last time; just a linear classifier.

A kernel is a mapping from datapoint x to a point $\phi(x)$ in a higher-dimensional space. Example: map $(x_1, x_2, \ldots, x_n)$ to the n^3-dimensional vector $(x_{i_1}x_{i_2}x_{i_3})$. The coordinates of $\phi(x)$ can be seen as *features*.

What makes kernel SVMs reasonably practical are the following two facts: (i) the standard algorithm for fitting SVM's only needs the ability to compute *inner products* of pairs of data points. Thus they also work for kernel SVMs if the kernel is such that $\langle \phi(x), \phi(y) \rangle$ is easy to compute given x, y. This is true for all the popular kernels. Thus in the above example, there is no need to work with the explicit n^3 size representation. (ii) the running time and the sample complexity of the algorithm is proportional to $1/\lambda$, where λ is the *margin* between the 0 examples and the 1 examples. Thus the running time can be small even if the kernel implicitly is mapping to a very high dimensional space.

The theoretical justification for kernel SVMs is the *margin hypothesis*: For every classification task there is a reasonable kernel for it that also has large margin (hence, can be fitted efficiently)

The lure of kernel SVMs is that they promise to fold in the *feature learning* with the classification task. Unfortunately, in practice people need to use more explicit ways of learning features, which gives a new representation for the data and then one can fit a classifier on top.

This brings us to today's topic, which is feature learning. Just as with classfication, this can be done in a *generative* and *nongenerative* setting.

Example 1 In the k-means problem one is given points $x_1, x_2, \ldots, \in \Re^d$ and one is trying to find k points (called *means*) $c_1, c_2, \ldots, c_k$ so as to minimize

$$\sum_i |x_i - p_i|^2,$$

where p_i is the closest mean to x_i.

After learning such means, each point has been labeled from 1 to k, corresponding to which mean it is closest to.

The generative analog of this would be say mixture of k gaussians. Each datapoint could be labeled with a k-tuple, describing its probability of being generated from each of the gaussians.

The nongenerative feature learning problems (such as k-means) are often NP-hard, whereas the generative version seems potentially easier (being average case). So I am attracted to the generative setting.

2 Linear Algebra++

The mathematical problems most directly useful to feature learning have to do with extensions of linear algebra. Recall that your freshman linear algebra class consisted of the following three main methods. (a) Solving linear equations. $Ax = b$. (b) Computing rank, which we can also think of as *matrix factorization:* Given $n \times m$ matrix M rewrite it as $M = AB$ where A is $n \times r$, B is $r \times m$ and r is as small as possible. (c) Eigenvalues/eigenvectors and singular values/singular vectors. For instance, every symmetric matrix M can be rewritten as

$$\sum_i \lambda_i u_i u_i^T,$$

where u_i's are eigenvectors and λ_i's are eigenvalues. This is called the *spectral decomposition* (if the matrix is not symmetric, the analogous expression is called *Singular Value Decomposition*).

Linear Algebra++ is my name for the above problems with any subset of the following extensions: (i) require some coordinates of the solution to be nonnegative. (ii) require a solution with a specified number or pattern of nonzeroes. (iii) solve in the presence of some kind of noise.

EXAMPLE 2 If we have to solve $Ax = b$ subject to x being nonnegative, then that is tantamount to linear programming, which was only discovered to be solvable in poly time in 1979.

If we have to solve $Ax = b$ subject to x having only k nonzeroes then this is the *sparse recovery problem* that is NP-hard.

If we have to solve $M = AB$ in presence of coordinate-wise noise, we may be looking for desired A, B such that we minimize

$$\sum_{ij} |M_{ij} - (AB)_{ij}|^2.$$

This is minimized by truncating the SVD to the first r terms. We will denote the best rank k approximation to M by M_k.

You never saw most of these extensions in your freshman linear algebra because they are NP-hard. But they are ubiquitous in ML settings.

3 Linearized models

The notion of features is often in some linearized setting: topic models, sparse coding, sparse recovery. We will see these in later lectures.

Rest of the lecture covered sparse recovery (Moitra's lecture notes); SVD (my lecture notes from COS 521) and use of SVD for clustering (Hopcroft-Kannan book).

COS 598C: Detecting overlapping communities, and theoretical frameworks for learning deep nets and dictionaries

Lecturer: Sanjeev Arora
Scribe: Max Simchowitz

April 8, 2015

Today we present some ideas for provable learning of deep nets and dictionaries, two important (and related) models. The common thread is a simple algorithm for detecting overlapping communities in networks. While community detection is typically thought of as a way to discover structure in, say, large social networks, here we use as a general purpose algorithmic tool to understand structure of latent variable models. The algorithm for learning deep nets and dictionaries starts by identifying correlations among variables, and represent these pairwise correlations using a graph. Then it uses community-finding to uncover the underlying connection structure.

1 Detecting Overlapping Communities in Networks

Community detection has been well studied in planted settings where the communities are disjoint. We discussed the *stochastic block model* in an earlier lecture. The concrete setting was that we are given $G = (V, E)$, where the vertices of V are partitioned into two sets S and S^c, and edges within S and S^c are drawn with probability p, and between S and S^c are drawn with probability q, such that $p - q = \Omega(1)$. Then, as long as $\min(S, S^c) = \Omega(\sqrt{|V|})$, we can easily recover S and S^c using an SVD or semi-definite programming [6].

However, when communities overlap, this problem does not seem doable via SVD. To recap the notation, let $G = (V, E)$ be our graph, and lets assign users to (perhaps more than one) communities $C_1, \ldots, C_m$. In the simplest setting - such as the one that arises in the dictionary learning problem - $(v_1, v_2) \in E$ if and only if there is a community C_j such that $v_1 \in C_j$ and $v_2 \in C_j$. In this case, we can identity C_j are subgraphs of G, all of which are cliques, and G is precisely the union of these cliques. I don't know of an algorithm to find the cliques given the graph, if the graph is a union of arbitrary cliques.

Luckily, in the dictionary learning setting, the structure of G is not determined adversarially. Instead, we assume that the vertices $v \in V$ are distributed across the communities fairly evenly, and that each v doesnt belong to too many communities (say, there are no hubs). We formalize the generative process for G as follows:

Definition 1.1 (Planted Problem Corresponding to Overlaping Communities)**.** Let $G = (V, E)$, where $|V| = N$. Suppose that there are m communities $C_1, \ldots, C_m$, and each vertex is assigned to k communities uniformly at random. Finally, if u, v belong to the same

community, then $Pr[(u,v) \text{ is an edge }] = p \geq 0.9$. If they do not have any overlapping community, then there are no edges.

Remark. If $p = 1$, then $C_1, \dots, C_m$ are cliques, and we are back in the clustering setting for dictionary learning.

The nice thing about our generative process is that it admits for a local search heuristic, as described in [2]. First, set $T = kN/m$, which is roughly the expected size of each community. Now, if (u,v) are in a community C_i, then the expected number of edges shared between them is about pT, so by a Chernoff bound, there are at least $.9pT$ vertices w connected to u and v with high probability.

On the otherhand, suppose (u,v) are not in the same community. Then, while they are not necessarily joined by an edge, there may be vertices w such that (u,w) are in one community, say C_i, and (u,v) are in another community, say C_j, giving rise to edges from both u and v to w.

Now, how many such spurious edges are there? That, is, what is the probability that edges occur between any two vertices, neglecting shared community structure? Well, there are $\binom{N}{2}$ ways to pick pairs of vertices, and the number of edges in the graph is no more than the sum of the number of edges in one community, which concentrates around $p\binom{T}{2}$ using a standard Chernoff argument. Taking the union over all m communities, we see that the probability of an edge between two vertices is no more than

$$p_0 := pm\binom{T}{2}\binom{N}{2}^{-1} \tag{1.1}$$

Hence, the probability of a spurious edge between w and u and w and v is about p_0^2, and thus the number of spurious edges between (u,w) concentrats around $p_0^2 N$. Hence, to distinguish between u and v sharing only spurious edges and sharing non-trivial edges due to common community memberships, we want to ensure

$$p_0^2 N \ll pT \tag{1.2}$$

This amounts to imposing the requirement that

$$\frac{N \cdot T^4 m^2}{N^4} \ll T \iff m \ll (N/T)^{3/2} \iff m \ll (m/k)^{3/2} \tag{1.3}$$

that most of the edges between u and v will be because they are in the same community. Hence, we can greedily assign vertices to communities by considering the number of common edges.

1.1 Notation

Given a vector $x \in \mathbb{R}^n$, we will denote its i-th entry by $x(i)$. We will denote the inner product between two vectors $x, y \in \mathbb{R}^n$ by $\langle x, y \rangle$, or $x^T y$ interchangably. Given a matrix $A \in \mathbb{R}^{n \times m}$, we denote its i-th column by A_i.

2 Neural Netowrks

Before expounding on the applications of the community finding algorithms described in [2], we will take a brief detour into the world of neural networks: perhaps one of the most popular tools in contemporary machine learning. At a very basic expert, neural networks mimic the structure of physical brains. One abstraction for emulating a brain is to view a network of biological neurons as large graphs, whose vertices are neurons and whose edges are synapses (or other forms of connections). The state of such a neural network is described by the potential with which each neuron is activited (and by other factors like current), and the synapses determine how much potential is transferred from one neuron-node to the next.

Motivated by both common implementation practices and theoretical feasibility, we will consider study artificial neural networks which decompose in L-layers. We can therefore describe the state of this network at a given time by an L-tuple of vectors $x^{(1)}, \dots, x^{(L)}$, where the entries of the vector $x^{(l)} \in R^{N_l}$ record the potentials of a corresponding neuron in the l-th layer. For example $x^{(2)}(1)$ is the potential of the first neuron in layer two. We refer to $x^{(1)}$ as the top layer and $x^{(L)}$ as the bottom layer.

What makes neural networks fascinating is the way the potential vector $x^{(l)}$ in different layers relate to one another. In biological neural tissue, electrical potential and chemical signals are being exchanged continuously. In our setting, we instead imagine that, at discrete interviews $t = 1, \dots, T$, nature draws top layer - potential vectors $x_t^{(1)}$. Then, the potentials in each succesive layer $x^{(l)}$ is given by a noisy objection of a deterministic function of the potentials $x^{(l-1)}$ in layer $x_t^{(1)}$.

We model this transfer of potentials as

$$x^{(l+1)} = h(A^{(l)} x^{(l)}) \tag{2.4}$$

where h is an (often nonlinear) function which which operates entrywise and identically on each entry, and $A^{(l)} \in \mathbb{R}^{N^{(l)} \times N^{(l+1)}}$ is a matrix specifying how the potentials in one layer feed into the next. Equivalently, we can think of $A^{(l)}$ as the adjacency matrix of a bipartite graph $G^{(l)}$, whose edges represent the connection between neurals. In what follows, we will interchange between identifying the vertices of $G^{(l)}$ with the entries of $x^{(l)}$, both of which semantically correspond to the neurons in the l-th layer.

To lighten up the notation and facillitate exposition, the majority of these notes will focus on learning networks with only two layers: one encoded by a sparse vector x, hidden to the observer and drawn from a suitably behaved generative process, and a dense layer encoded by a dense vector y, which can be observed. We will use G and A to refer to the connection graph between x and y, and its adjacency matrix, respectively.

3 Dictionary Learning, Neural Nets, and Community Finding

We can imagine that even the two layer problem is rather difficult for arbitrary, nonlinear h. Thus, it makes sense to start off by considering the simpler case where h is just the identity; that is $Ax = y$. This problem is known as *Dictionary Learning*, and the adjacency matrix A is called the dictionary.

In the Dictionary Learning problem, we are given samples $y_1, \dots, y_N$ samples of observed potentials, and our goal is to reconstruct both A, and the hidden samples $x_1, \dots, x_N$ so as to minimize the error

$$\min_{A,\{x_i\}} \|Ax_i - y_i\|_2^2 \tag{3.5}$$

In general, this problem is extremeley over-determined. Indeed, if the x's have dimension greater than the y's, then it is trivial to reconstruct A and x_i for which $Ax_i = y_i$ exactly. In order to make the problem both meaningful and tractable, we need to posit that the x_i have some additional structure. Here, we will assume that the samples x are sparse.

There are two motivations for recovering sparse x_i. The first is empirical - biological neurons tend to show sparse activation patterns. More broadly, sparsity is a rather intuitive assumption for capturing a sense of "latent simplicity" or "hidden structure" in otherwise very high dimensional data. The second motivation is that, assuming sparsity, we can leverage insights from sparse recovery and compressed sensing, under certain conditions on the dictionary matrix A. Recall that matrices that have low column inner products are called incoherent:

Definition 3.1. Let A be a matrix with columns A_i, such that $\|A_i\| = 1$. We call A $\mu/\sqrt{n}$ incoherent if $|A_i^T A_j| \leq \frac{\mu}{\sqrt{n}}$

Now, if we knew A exactly, and A is sufficiently incoherent, then we have the following result

Theorem 3.1 (Compressed Sensing, Stated Loosely)**.** *Let A be a matrix with unit norm columns, such that $|\langle A_i, A_j\rangle| \leq \frac{1}{2k}$. Suppose given $y = Ax$ where x is k-sparse. Then, x is the unique k-sparse vector for which $y = Ax$. Hence, x can be recovered in polynomial time.*

The guiding insight is that, for $\mu/\sqrt{n}$ incoherent dictionaries, $A^T A \approx I$, since the of diagonals are bounded above by $\mu/\sqrt{n}$. Note that this approximation is not necessarily a great one in the spectral sense, since $A^T A - I$ can have $n^2 - n$ entries of size $\Omega(\mu/\sqrt{n})$, and thus $\|A^T A - I\|$ might be $\Omega(\mu\sqrt{n})$.

But looking only at the spectral norm *does not take advantage of sparsity*: Indeed, $\|A^T A - I\| = \max_{\|z\|:1} z^T(A^T A - I)z$, and if $A^T A - I$ has entries all around $\mu/\sqrt{n}$, this maximimum will be attained for $z^* \approx \frac{1}{\sqrt{n}}(1, \dots, 1)$. However, if we impose that z^* is k-sparse, things are a bit different. Define the seminorm $\|z\|_0 := \sum_i I(z_i \neq 0\}$, and let $B_0(k) := \{z \in \mathbb{R}^n : \|z\| \leq 1, \|z\|_0 \leq k\}$. It is rather easy to show that

$$\sup_{z \in B_0(k)} z^T(A^T A - I)z \leq k\mu/\sqrt{n} \tag{3.6}$$

This restriction to the subset of k=sparse vectors gives rise to the notion of the "Restricted Isometry Property" in the compressed sensing literature [4]. Indeed, if $k\mu/\sqrt{n} < 1/2$, then A is effictively "invertible" for all $2k$ sparse vectors z, and if $k\mu/\sqrt{n} = o(1)$, then $\langle Az, Az\rangle = \|z\|^2 + z^T(A^T A - I)z \approx \|z\|^2$ for all k-sparse z. More precisely, we can prove the following lemma:

Lemma 3.2. *Let z_1 and z_2 be two k-sparse vectors, and let A have unit norm columns, Then Then $\langle Az_1, Az_2\rangle = \langle z_1, z_2\rangle \pm \frac{2k\mu}{\sqrt{n}}\|z_1\|\|z_2\|$.*

Proof. By relableing the columns of A and then entries of z_1 and z_2, we can imagine that z_1 and z_2 are both supported on the indices $[2k] := \{1, \dots, 2k\}$. Hence,

$$\begin{aligned} \langle Az_1, Az_2 \rangle &= \sum_{i \in [2k]} \|A_i\|^2 z_1(i) z_2(i) + \sum_{i \in [2k]} \sum_{j \neq 1 \in [2k]} z_1(i) z_2(j) \langle A_i, A_j \rangle & (3.7) \\ &= \langle z_1, z_2 \rangle + E & (3.8) \end{aligned}$$

where $E := \sum_{i \in [2k]} \sum_{j \in [2k]} z_1(i) z_2(j) \langle A_i, A_j \rangle$.

$$\begin{aligned} |E| &\leq \sum_{i \in 2k} \sum_{j \neq i \in [2k]} |z_1(i)||z_2(j)| \cdot |\langle A_i, A_j \rangle| & (3.9) \\ &\leq \sum_{i \in 2k} \sum_{j \in [2k]} |z_1(i)||z_2(j)| \cdot |\langle A_i, A_j \rangle| & (3.10) \\ &\leq \frac{\mu}{\sqrt{n}} \sum_{i \in [2k]} \sum_{j \in} |z_1(i)||z_2(j)| \leq \frac{\mu}{\sqrt{n}} \|w_1 w_2^T\|_F & (3.11) \end{aligned}$$

where $w_1 \in \mathbb{R}^{2k}$ has $w_1(i) = |z_1(i)|$ for all $i \in [2k]$, and w_2 is defined similarly for z_2, and $\|\cdot\|_F$ denotes the Frobenius norm. Because $w_1 w_2^T$ is a $2k \times 2k$ matrix, we have $\|w_1 w_2^T\|_F \leq 2k \|w_1 w_2^T\|$, where $\|\cdot\|$ denotes the spectral norm. But $\|w_1 w_2^T\| = \|w_1\| \|w_2\| = \|z_1\| \|z_2\|$, whence

$$|E| \leq \frac{2k\mu}{\sqrt{n}} \|z_1\| \|z_2\| \quad (3.12)$$

□

3.1 Formal Models for Dictionary Learning

To encourage sparsity, Olshausen and Field [5] designed an alternating gradient descent algorithm to minimize the following objective:

$$\min \sum_{i=1}^{N} |y_i - Ax_i|^2 + \sum_{i=1}^{N} \text{penalty}_K(x) \quad (3.13)$$

As we remarked about, an unpenalized Dictionary learning is highly underdetermined. Hense, Olshausen and Field introduced the penalties - for example, l_1-regularization - to encourage sparsity and ensure (or at least promote) model indentifiability [5]. In [3], Arora, Ge, et al. describe an alternating minimization algorithm based on Olshausen and Field to learning the objective in Equation 3.13. In these notes, we will restrict our attention to the "overlapping community methods" to be described shortly. In either case, both the alternating minimization algorithms in [3] and the overlapping community detection methods from [2] will make use of roughly the same assumptions, which we formalize as follows:

1. The dictionary $A \in \mathbb{R}^{n \times m}$ has unit norm columns, and has $\frac{\mu}{\sqrt{n}}$-incoherent columns, that is: $|\langle A_i, A_j \rangle| \leq \frac{\mu}{\sqrt{n}}$.

2. We are concerned with the regime $m \geq n$, and we require that $\|A\| = O(\sqrt{m}/\sqrt{n})$.

3. Each x has exactly k nonzero coordinates, drawn uniformly from $\{1, \dots, m\}$ (this can be relaxed somewhat, as in [3])

4. x each coordinate is independent conditioned on its support, and $x_i | x_i \neq 0$ is subgaussian with $O(1)$ variance proxy, and there is a constant C - universal across all $i \in [m]$ - such that $|x_i||x_i \neq 0 \geq C$ almost surely. For example, we can think of $x_i|x_i \neq 0$ as being drawn uniformly from $[1, 10]$, or from $[-10, -1] \cup [1, 10]$.

5. We will start off by assumping that $x_i \geq 0$ almost surely. An adpatation of the arguments in this paper will also hold for the case where $\mathbb{E}[x_i] = 0$

Given samples $y_1 = Ax_1$ and $y_2 = Ax_2$, it follows from Lemma 3.2 that

$$\langle y_1, y_2 \rangle = \langle x_1, x_2 \rangle \pm \|x_1\|\|x_2\| \frac{k\mu}{\sqrt{n}} \tag{3.14}$$

By subgaussian concentration, it holds with high probability that $\|x_1\|\|x_\| = \tilde{O}(k)$, so as long as $k^2\mu/\sqrt{n}$ is roughly $o(1)$, then

$$\langle y_1, y_2 \rangle = \langle x_1, x_2 \rangle \pm o(1) \tag{3.15}$$

If we assume that x_1 and x_2 are entrywise non-negative, then

$$\begin{aligned} \langle x_1, x_2 \rangle &= \sum_{i \in \mathrm{Supp}(x_1) \cap \mathrm{Supp}(x_2)} x_1(i)x_2(i) && (3.16) \\ &\geq C|\mathrm{Supp}(x_1) \cap \mathrm{Supp}(x_2)| && (3.17) \\ &\geq CI(\mathrm{Supp}(x_1) \cap \mathrm{Supp}(x_2) \neq \emptyset) && (3.18) \end{aligned}$$

Hence, with high probability, it holds that $\langle y_1, y_2 \rangle \geq C/2$ if and only iff x_1 and x_2 have a nonzero entry in common. We will state this in an informal lemma:

Lemma 3.3. *If $k^2\mu/\sqrt{n}$ is roughly $o(\log n)$, then with very high probability $\langle x_1, x_2 \rangle \geq C/2$ if and only x_1 and x_2 share a nonzero entry.*

This observation allows us to transform the problem from an analytic one to a combinatorial one. Indeed, given N observations $y_1, \dots, y_N$, let each observation y_i correspond to a vertex i in a graph $G = (V, E)$. We draw an edge between the vertices i and j if and only if $\langle y_i, y_j \rangle \geq C/2$. By the above discussion, it holds high probability that edges are drawn between i and j if and only if x_i and x_j share a common non-zero entry. Taking a union bound, the following claim holds:

Lemma 3.4. *With high probability that $G \simeq \tilde{G}$, where $\tilde{G}$ is the graph over the vertices $i \in [N]$ with edges connected all indices i, j for which x_i and x_j have non-disjoint support*

We now give a more intuitive way to characterize $\tilde{G}$: Let $C_1, \dots, C_m$ be sets defined so that $C_j := \{i \in [N] : x_i(j) \neq 0\}$. We will call the sets "communities", in the sense that all $i \in C_j$ share a nonzero entry in common. By the assumption that there are exactly k nonzero entries of each sample x_i selected uniformly at random, each vertex i is assigned

to exactly k communities $C_{j_1}, \dots, C_{j_k}$. Moreover, it follows directly from the definition of the sets C_j that x_i and x_j share a common nonzero if and only if they both lie in the same community: $\tilde{G}$ is precisely the graph constructed by drawing edges between vertices which belong to at least one of the same community. Hence, to recover the sparsity patterns of the x_i with high probability, Lemma 3.4 tells us that the graph G, whose edges are constructed from the inner products of samples y_i and y_j, is precisely generated by the community assignments of its vertices.

3.1.1 Mean Zero Case

If the entries of x_i are mean zero, then the argument is a little different: indeed,

$$\sum_{i \in \text{Supp}(x_1) \cap \text{Supp}(x_2)} x_1(i)x_2(i) \tag{3.19}$$

can have absolute value much smaller than C due to cancellations from the entries of x_1 and x_2 having cancelling signs. However, with probability $\Omega(k^2/m^2)$, $\text{Supp}(x_1)$ and $\text{Supp}(x_2)$ will overlap at at most one entry, so we can neglect these correlations if we are willing to accept a small (but not as small as $n^{-\omega(1)}$) probability of missing an edge (which will also not be independent of the common support of and x_2 x_1). On the other hand, we can improve the bound in Lemma 3.2 due to cancellations. Indeed, we have

$$\langle y_1, y_2 \rangle - \langle x_1, x_2 \rangle = \sum_{i \neq j} \langle A_i, A_j \rangle x_1(i)x_2(j) \tag{3.20}$$

Using the bound $|\langle A_i, A_j \rangle| \leq \mu/\sqrt{n}$, this term has mean zero and moment roughly $O(\sqrt{k}\mu/\sqrt{n})$ due to cancellations. Hence, $\langle y_1, y_2 \rangle = \langle x_1, x_2 \rangle + \tilde{O}(\sqrt{k}\mu/\sqrt{n})$, so our error drops by roughly a factor of $\sqrt{k}$.

3.2 Reduction of Dictionary Learning to Community Detection

Given our community detection algorithm, we have given a sktech of how to efficiently recover the sparsity patterns of the latent samples x_i. We show how to use this technique to recover the dictionary A, folllowing [2]. Note that, once A has been retrieved, we can use more standard techniques from sparse recovery to (approximately) recover the latent signal vectors x.

The basic idea is that the j-th column of A, A_j, should roughly resembly the average of all sampes y_i for which the j-th entry is active: that is, $y_i = Ax_i$ where $x_i(j) \geq 0$. Thus, a first first attempt at recovering A would be simply to compute the following average:

$$A_j := \frac{1}{|C_j|} \sum_{i : y_i \in C_j} y_i \tag{3.21}$$

Unfortunately, in the case were the x_i have mean zero, we have $\mathbb{E}[y_i] = \mathbb{E}[Ax_i] = A\mathbb{E}[x_i] = 0$, In the case where the x_i have do not have mean zero, then we get a lot of spurious contributions from the nonzero entries of the samples at indices not equal to j: that is, $y_i = A_j x_i(j) + \sum_{j' \neq j} A_{j'} x_i(j')$.

A better idea is to instead look at the best rank 1 approximation to $E[yy^T] : y \in C_j$. For simplicity, we will first handle the mean zero case. Note first that, because the problem is invariant to permutation of the columns of A, it suffices to prove an algorithm which recovers A_1, the column of A which corresponds to community C_1. Our strategy will be to compute the best rank-one approximation to the empirical covariance matrix of all samples y_i which have an active first column, that is $y_i \in C_1$. Let

$$M_1 := \frac{1}{\#y : y \in C} \sum_{y \in C} [yy^T] \tag{3.22}$$

that is, the empirical average of all yy^T for $y \in C$. First, lets show that M_1 is a good approximate of $A_1 A_1^T$ up to a constant factor:

$$\begin{aligned} M_1 &= \mathbb{E}[x(1)^2 A_1 A_1^T] + \mathbb{E}[\sum_{i \geq 2} x(i)^2 A_i A_i^T] \\ &+ \mathbb{E}[\sum_{i \geq 2} x(i)x(j)(A_1 A_i + A_i A_1)] + E[\sum_{i,j \geq 2} x(i)x(j) A_i A_j] + \text{ statistical error} \\ &\approx \Theta(A_1 A_1^T) + O(\frac{k}{m} \sum_{i \geq 1} A_i A_i^T) + \tilde{O}(k^2/\sqrt{N}) \end{aligned}$$

Here N is the number of samples used, the $O(f)$ notation is means a quantity whose spectral norm is bounded by Cf for some $C > 0$, and $O(M)$ (resp $\Theta(M)$) means a quantity which is less than CM (resp less than CM and greater than cM) according to the cannonical ordering $\preceq$ of the semidefinte cone . The first term comes from the fact that $\mathbb{E}[x(1)^2 | y \in C] = \Theta(1)$, the second term comes from the fact that $\mathbb{E}[x(i)^2 | y \in C] = O(k/m)$. Note that the second error term is systemic - it does not depend on the number of samples used by the algorithm.

The remaining error term $\tilde{O}(k^2/\sqrt{N})$ is statistical in nature, and comes from the deviation of all the terms from their exptation. It is easy to establish the $\tilde{O}(k^2/\sqrt{n})$ by conditioning on the very high probability even that all the x_i are small, and then using Chernoff bounds to finish up. The bound can be improved to $\tilde{O}(k/\sqrt{N})$, but this improved bounded affects the sample complexity of the algorithm. On the other hand, the systemic error of $O(\frac{k}{m} \sum_i A_i A_i^T)$ determines the conditions on k and m under which the best-rank-one approximation algorithm accurately retrives the underlying dictionary.

First, we will use a standard assumption in the dictionary learning literature that $\|A\| = O(\sqrt{m}/\sqrt{n})$. Under this condition, it holds that $\|(\frac{k}{m} \sum_{i \geq 2} A_i A_i^T)\| = O(k/n)$. We will assume that the number of samples is large enough that the statistical error is also dominated by $O(k/n)$. Thus, $M_1 \propto A_1 A_1^T + E$, where E has norm $O(k/n)$. N

Now let $\hat{A}_1$ be the top eigenvector of M_1. We can show that $\hat{A}_1$ is a good estimate of A_1 by appealing to Wedin's Theorem, an elementary result from linear algebra, which bounds the distance of the top eigenvector of a PSD matrix A to that of $A + E$, where E is a small perturbation. Because A_1 is the top eigenvecotr of $A_1 A_1^T$, Wedin's Theorem will help us show that the top eigenvector of M_1 should be close to A_1 as well:

Theorem 3.5. *Let v_1 be the top eigenvector of PSD matrix A and let v_2 be the top eigenvector of $A + E$. Let θ be angle between v_1 and v_2. Then $\sin\theta \leq \frac{2\|E\|}{\sigma_1(A) - \sigma_2(A)}$.*

As a corrolary, we get a clean bounded on the Euclidean distance between the (normalized) top eigenvectors of A and $A + E$

Corollary. *Let A be a rank one matrix of norm 1 with top eigenvector v_1, let v_2 be the top eigenvector of $A + E$. Then as long as $\|E\| = o(1)$ $\|v_1 - v_2\| \leq \sqrt{1/2}$, $\|v_1 - v_2\| \leq 2\|E\|$*

Proof. Because $\sigma_1(A_1 A_1^T) = 1$, and $\sigma_2(A_1 A_1^T) = 0$, we have that $\sin\theta(v_1, v_2) \leq 2\|E\|$. Because $v_1^T v_2 = 1 - \|v_1 - v_2\|^2$, we have

$$\begin{aligned} \sin\theta(v_1, v_2) &= \sin\arccos(v_1^T v_2) = \sqrt{1 - (v_1^T v_2)^2} = \sqrt{2\|v_1 - v_2\|^2 - \|v_1 - v_2\|^4} && (3.23) \\ &= \sqrt{2}\|v_1 - v_2\|\sqrt{1 - \|v_1 - v_2\|^2} && (3.24) \end{aligned}$$

Because $E = o(1)$, it follows that $\sin\theta(v_1, v_2)$, and hence $\|v_1 - v_2\|$, must be $o(1)$ as well. Hence, $\|v_1 - v_2\| \leq \sqrt{2}\|E\|/\sqrt{1 - \|v_1 - v_2\|^2} \leq 2\|E\|$. □

From this corrolary, it follows immediately that $\|\hat{A}_1 - A_1\| \leq O(k/n)$. Hence, given enough (but still polynomially many) samples, we can recover easy the columns of A up to an error of k/n.

4 Unsupervised learning of Deep Nets

Let's return from the restricted setting of dictionary learning to the more general setting of neural nets. Aside from their success, one of the major reasons for the popularity of deep nets is that the last layer seems to capture "meaningful features". For example, in vision problems, the pixel-representations of an object learned by neural nets often represents the shape of that object very closely. And, in many applications, one can train very effective clasifiers (using, say, an SVM or Logistic Regression) on the features learning by the last layer. In fact, if we train a multilayer neural network for a classification task - say, distinguishing between cats and dogs - and then retrain the last layer to learn a new task - say, distinguish between birds and bees - *without retraining the parameters of most of the hidden layers*, the retrained network is still remarkably succesful at its new classification task. This suggest that the representations learned in the deeper layers of the neural network capture most of the relevant information, or at least enough information to build effective classifiers.

This suggests that deep nets are capturing some inherent structure in the images themselves, raising hope that the hidden layers correspond to natural "features" that could be learnt from just unlabeled data. (By contrast, the recent successes involve leveraging large amounts of labeled images.) Unsupervised training of deep nets is a holy grail of this area, and major researchers in this area have tried to define a generative model corresponding to deep nets. This quest is very much in the spirit of the discriminative-generative pairs we discussed in an earlier lecture (eg naive bayes classifier is a generative analog of logistic regression).

If we move from the descriminative perspective to the generative perspective, we might wonder - what is the structure that neural nets can extract structure from the data? Let's now consider a two layer neural net whose top layer is encoded in a vector x and bottom layer is encoded in a vector y.

Rather than imagining a linear map A which takes a sparse input x and maps it to a dense y, we now imagine an *encoding function* $E(\cdot)$ which *encodes* a dense y as a sparse x. In the linear case, we had that $y = Ax$, so that $x \approx A^T y$. In the general case, we model $x = E(y) = h(A'y + b)$, where b is an offset function and $h(\cdot)$ is a nonlinear map which acts identically and independently on each coordinate, for example, $h(\cdot)$ can be the function which returns the sign of the entries of its arguments. Again, we can imagine that eancy entry of x and y are treated as vertices in a bipartite graph, and that A is the adjacency matrix which captures the edge weights between the entries of x and y.

The hope is that we can now invert the encoding function $E(\cdot)$, and in fact perform the inversion in the presence of noise. This motivates the following definition:

Definition 4.1 (Denoising Auto-Encoder)**.** Give an adjacency matrix A, an *autoencoder* consists of a pair of an encoding function of the form $E(y) = h(A'y + b)$ and a decoding function $D(x) = h(Ax + b')$. The autoencoder is called *denoising* for a noise model $\xi \sim \mathcal{D}$ if the the decoding robust to noise in the sense that:

$$E(D(h) + \xi) = h \quad \text{with high probability} \tag{4.25}$$

and said to be weight tying if $A' = A^T$. Here $D(h) + \xi$ is shorthand for D corrupted with the noise vector ξ. This corruption might not necessarily be additive.

The following theorem states that, if the entrywise nonlinear function $h(\cdot)$ is the sign function, and A is sufficiently sparse, then [1] show that the two layer neural network is in fact a denoising autoencoder:

Theorem 4.1 (stated loosely)**.** *Consider a two layer neural network with sparse bi-partite graph G with adjancey matrix A with edge weights drawn uniformly in $[-1, 1]$. Suppose that the latent sample x are binary with support S. Finally, suppose that $y = sign(Ax)$. Then there is a b' for which the pair $E(\cdot)$ and $D(\cdot)$ form an denoising autoencoder, where $E(\cdot) = sign(A^T y + b')$.*

In fact, we can learning the encoding/decoding function with high probability:

Theorem 4.2. *Under some regularity assumptions, there is a polynomial time algorithm to learn the encoding and decoding functions for a two layer neural with sparse edge weights drawn uniformly in $[-1, 1]$*

Proof. To preserve intuition, we assume that we have an unweighted bipartite graph which is drawn unifromly from all d-regular bipartite graphs on vertex set given by the entries of x and y. We assume that thare $E(\cdot)$ and $D(\cdot)$ are chosen with no thresholding function, so that $b = b' = 0$. We also assume that the x_i are uniformly drawn, k-sparse binary vectors, where $x = \rho n$ for some small ρ.

Let's begin by learning the adjacency matrix A, or equivalently, the graph G. What are the communities? They are subsets of nodes with a common neighbor. So what happens when two nodes have a common neighbor. If u, v have a common neighbor, then $Pr[u, vare1] \geq \rho$. So $Pr[u, v \text{ are both } 1] \leq (pd)^2$. So if $\rho \gg (\rho d)^2$, we can recover the communities with high probability.

Now lets describe how to recover the entries of samples x. The key intuitition is that, if an entry of x, say x_1 is active, then some number of its neighbors y_i will be active as

well. Hence, we can recover x_1 by determining if above a certain threshold of its neighboring indicies y are 1. The guarantees behind these algorithm come from the following observation: that uniformly drawn sparse bipartite graphs are expanders with high probability. Let's be more specific:

Let U denote the vertex set corresponding to the entries of x, and V the vertex set corresponding to the entries if y. Given $u \in U$, let $F(u)$ denote all of its neighbors in V. Fiallly for some set $S \subset U$, let $UF(u, S)$ be the set of unique neighbors of u with respect to S: that is

$$UF(u, S) := \{v \in V : v \in F(u), v \notin F(S - \{u\})\} \tag{4.26}$$

It turns out that, for a randomly generated bipartite graph and sufficiently small set S, then for every $u \in U$, the total number of u's neighbors in $UF(u, S)$is at least $9d/10$ of its total number of neighbors. Hence, if an entry x_i is not active, we expect no more than, say $2d/10$ of its neighbors in V to be active. Hence, we can recover the vector x with high probability by setting

$$x_i = \text{threshold}_{2d/10}\left(\#\text{neighbors of } x_i \text{ active }\right) \tag{4.27}$$

□

Perhaps even more surprisingly, [1] show that one can learn the connect graphs $G^{(l)}$ in a multilayer neural network by learning the bottom-most layers first, and then moving upward thhrough the graph:

Theorem 4.3 (Generalization to Deep Nets)**.** *Given a deep neural network with layers $x^{(l)}$ and weighted connection graphs $G^{(l)}$ drawn with expected degree $d^{(l)}$, and edge weights uniformly in $[-1, 1]$, and where the samples in the top layer are binary vectors with uniform sparse support of size ρn. Then, if ρ is sufficiently small, and the degrees $d^{(l)}$ do not grow too quickly, then the ground truth graphs $G^{(l)}$ and corresponding samples x can be learned with high probability. In fact, they can be learned my infering the second to bottom layer from the bottom layer, and then moving up layerwise through the network.*

References

[1] Sanjeev Arora, Rong Ge, Aditya Bhaskara, Tengyu Ma. "Provable Bounds for Learning Some Deep Representations." *Journal of Machine Learning*, volume 32, 2014.

[2] Sanjeev Arora, Rong Ge, Ankur Moitra. "New Algorithms for Learning Incoherent and Overcomplete Dictionaries" *Conference on Learning Theory*, 2014.

[3] Arora, Sanjeev, et al. "Simple, Efficient, and Neural Algorithms for Sparse Coding" *arXiv preprint arXiv:1503.00778*, 2015

[4] Candes, Emmanuel J. "The restricted isometry property and its implications for compressed sensing." *Comptes Rendus Mathematique* 346.9 (2008): 589-592.

[5] Bruno A. Olshausen and David J. Field. "Sparse coding with an overcomplete basis set: a strategy employed by v1." *Vision Research*, 37:3311–3325, 1997a.

[6]

Tensor decomposition + Another method for topic modeling

Lecturer: Sanjeev Arora
Scribe: Holden Lee

April 13, 2015

Today we talk about *tensor decomposition*, a general purpose tool for learning latent variable models. Then we switch gears and talk about a recent improvement of the topic modeling algorithm we saw in an earlier lecture.

0.1 Tensor decomposition

Tensor decomposition is the analog of spectral decomposition for tensors.

The nice thing about eigenvalues/eigenvectors is that they exist (ok, singular values/vectors in case of nonsymmetric matrices) and you can efficiently compute them. For M a symmetric $n \times n$ matrix, we can write

$$M = \sum \lambda_i u_i u_i^T.$$

A 3-D tensor M is a $n \times n \times n$ array. Extending linear algebra to tensors is nontrivial. Many problems regarding tensors are NP-hard, like rank (which is not straightforward to define).

Today we are interested in tensors that we are guaranteed have a representation like $M = \sum \lambda_i u_i^{\otimes 3}$, where the u_i are orthogonal. We don't know the u_i's and are trying to recover them. We can actually recover these similarly to the **power method**. (Recall that the power method repeatedly sets $x \leftarrow \frac{Mx}{\|Mx\|_2}$; it gives the top eigenvector if there is a gap between the top 2 eigenvalues. The running time is inversely proportional to this gap.)

Definition 0.1: The **tensor-vector product** (aka *flattening* by x) is defined as follows: Mx is the matrix where

$$(Mx)_{ij} = \sum_k M_{ijk} x_k.$$

Now

$$Mx = \sum \lambda_i (u_i \cdot x) u_i^{\otimes 2}.$$

This looks like a spectral decomposition: it takes the orthogonal directions u_i and boosts them by $\lambda_i(u_i \cdot x)$. (Under the isomorphism $V \otimes V \cong V \otimes V^*$, $u_i^{\otimes 2}$ corresponds to $u_i u_i^T$.)

Why does this work? From inspection, the eigenvalues of Mx are $u_i \cdot x$ since the u_i's are orthonormal and spectral decomposition is unique. The Mx's are approximately Gaussian, and there is a good chance that Mx has a top eigenvalue, with a significant gap to the next eigenvalue.

0.1.1 Method of moments

In topic modeling, etc., what is really going on is that we are using the method of moments. The general setup is that we sample

$$x \sim D := D(A)$$

where A is the matrix of hidden parameters; given observed X we try to recover A. We can consider the moments

$$\begin{aligned} \mathbb{E}X &= f_1(A) \\ \mathbb{E}(X^{\otimes 2}) &= f_2(A) \\ \mathbb{E}(X^{\otimes 3}) &= f_3(A) \\ &\vdots \end{aligned}$$

Then we try to solve this nonlinear system of equations. A lot of machine learning can be thought of in this way.

Mathematicians and statisticians have studied questions like: What distributions can we identify from the third moments, or up to the kth moments?

Recall that in topic modelling, under the separability assumption, a document is sampled from A with $w \in \mathrm{Dir}(\alpha)$. We considered

$$XX^T = A \underbrace{\mathbb{E}[ww^T]}_{R} A^T$$

and used separable matrix factorization. We were exactly using second moments to recover the distribution.

See [AGH+14] for more on this framework.

Dictionary learning was not method of moments; we drew edges between X, X' when $|\langle X, X' \rangle| \geq \frac{1}{2}$ and used community detection on the resulting graph.

0.1.2 Example: Mixtures of identical spherical gaussians

Consider k Gaussians $N(\mu_i, \sigma^2)$ in n dimensions ($\mu_i \in \mathbb{R}^n$) where σ^2 *is known.* Let the mixing weights w_i be such that $\sum_{i=1}^k w_i = 1$. To pick a sample, pick i with probability w_i,

and output a sample from $N(\mu_i, \sigma^2)$. We have

$$\mathbb{E}[X] = \sum_{i=1}^{k} w_i \mu_i$$
$$\mathbb{E}[X^{\otimes 2}] = \sum_{i=1}^{k} w_i \mu_i^{\otimes 2} + \sigma^2 I$$
$$\mathbb{E}[X^{\otimes 3}] = \sum_{i=1}^{k} w_i \mu_i^{\otimes 3}$$

Assume we shift coordinates so that $\mathbb{E}[X] = 0$*, and that the* μ_i *are linearly independent.* If we can do a tensor decomposition of $\mathbb{E}[X^{\otimes 3}]$ then we will obtain the μ_i and weights w_i. However, we can't do tensor decomposition yet because the μ_i are in general not orthogonal. We must first whiten the vectors.

0.1.3 Whitening

The idea of **whitening** is to change tensors of the form $\sum w_i \mu_i^{\otimes 3}$ to $\sum w_i \nu_i^{\otimes 3}$ where the ν_i's are orthogonal. Letting $U = (\mu_1, \ldots, \mu_n)$, we have

$$P = \sum_{i=1}^{k} w_i \mu_i^{\otimes 2} = U \operatorname{diag}(w_i) U^T.$$

(This is not the spectral decomposition, because U is not orthogonal.) The spectral decomposition is, say

$$P = V D V^T$$

where V is orthogonal. *Assume* U, V *are full rank.* We would like to find a matrix A such that the vectors $\nu_i := A\sqrt{w_i}\mu_i$ are orthogonal, i.e., $AU \operatorname{diag}(\sqrt{w_i})$ are orthogonal. This is equivalent to

$$[AU \operatorname{diag}(\sqrt{w_i})][\operatorname{diag}(\sqrt{w_i}) U^T A^T] = 1 \iff APA^T = 1.$$

Thus, take $A = W^T$ where $W = VD^{-\frac{1}{2}}$. Then

$$APA^T = D^{-\frac{1}{2}} V (VDV^T) V^T D^{-\frac{1}{2}} = I$$

as needed.

In the Gaussian case, if we applied W to $\sum_{i=1}^{k} w_i \mu_i^{\otimes 3}$, we would get

$$\sum w_i (W^T \mu_i)^{\otimes 3} = \sum \frac{1}{\sqrt{w_i}} \nu_i^{\otimes 3}.$$

(Of course, we actually get the noisy versions of $\sum_{i=1}^{k} w_i \mu_i^{\otimes 2}$, $\sum_{i=1}^{k} w_i \mu_i^{\otimes 3}$, so if we want to do proper analysis we'll have to take error into account.)

(See also [BCMV13] for a somewhat different setting, overcomplete tensor decomposition.)

0.2 SVD-based approaches for topic models (presentation by Andrej Risteski)

We explain a paper by Bansal, Bhattacharyya, and Kannan [BBK], which uses SVD plus some other tricks. They develop and prove a SVD-based algorithm that learns topic models with L^1 error under certain assumptions including the catch words assumption (a weakening of the anchor words assumption).

We set up notation. Let k be the number of topics and n be the number of words. Let A be the words×topics matrix, giving the distribution of words for each topic, and W be the topics×documents matrix. Let $M = AW$. If $W_{\bullet i}$ is a column of W, then $\widetilde{M}_{\bullet i}$ is generated according to m draws on the distribution given by $M_{\bullet i}$. (m is the number of words in each document.)

The goal is to recover A with L^1 error. Previous works such as Arora et al. recovered with L^2 error. Note that L^2 error ignores words with small frequency, and empirically, a lot of words have small frequency. Moreover, columns are distributions so the natural norm is L^1.

0.2.1 Assumptions

We make the following assumptions. See the paper for the precise parameters.

1. (Dominating topic) We assume there is a **dominating topic** in each document:
   (a) for each document d there exists a topic $t(d)$ such that $W_{t(d),d} > \alpha$. For all other topics $t' \neq t(d)$, $W_{t',d} \leq \beta$, where $\beta - \alpha$ is large enough.
   (b) (Each topic appears as a dominating topic enough times) For each topic t there are $\geq \varepsilon_0 w_0 s$ documents d in which $W_{t,d} \geq 1 - \delta$.
2. **Definition 0.2:** w is a **catch word** for topic t if for all $t' \neq t$, $A_{wt'} \leq \rho A_{wt}$, and the probability of appearing is not too small, $A_{wt} \geq \frac{8}{m\delta^2\alpha} \ln\left(\frac{20}{\varepsilon w_0}\right)$.

   The catch words for t occupy a significant proportion of the words for topic t

$$\sum_{w \text{ is catch word for topic } t} A_{wt} > \frac{1}{2}.$$

   (You can replace $\frac{1}{2}$ by p_0, and get dependence on p_0 in later parameters. For simplicity we don't do this. There is some absolute lower bound on p_0.).
3. (Almost pure documents) There is a small fraction of almost pure documents. For all i, $\geq \varepsilon_0 w_0 D$ of the documents are such that $W_{td} > 1 - \delta$.
4. (No-local-minima assumption) Let $p_j(\zeta, t)$ be the probability that t is the dominant topic in the document, word j appears ζ time, i.e., with proportion $\frac{\zeta}{m}$. Then

$$p_j(\zeta, t) > \min(p_j(\zeta - 1, t), p_j(\zeta + 1, t)).$$

The motivation is that there are two possibilities: either the probability of the word appearing ζ times decays as ζ gets larger (e.g. as a power law), or it's a catch word, and it keeps rising until some frequency, and then decays.

5. (Dominant admixture) The proportion of documents where topic i is dominant is $\frac{D}{k}$, where k is the total number of documents.

0.2.2 Algorithm

The intuition is that topic models is like soft clustering, soft because each document doesn't belong to 1 cluster exclusively.

Intuitively, what is the obstacle? Suppose the frequency of a certain word in cluster 1 is in $[0, \sigma]$ and in cluster 2 is $[\mu, 1]$, with the spread much larger in cluster 2. Then clustering could split the second cluster into two.

This is solvable with the trick of *thresholding before clustering*. If μ is known, threshold by μ: if a coordinate is $> \mu$, then set it to be 1, and 0 otherwise. If you directly apply SVD, you can handle less noise than if you threshold first.

Consider the following problem.

Problem 0.3: Given a random $n \times n$ matrix A where some $m \times m$ submatrix has $\mathbb{P}(A_{ij} \geq \mu) \geq \frac{1}{2}$, and the other entries are $N(0, \sigma)$, find the submatrix (planted clique).

Solution. First consider the naive SVD solution.

The idea is that the spectral norm of the $m \times m$ matrix is significantly larger than the spectral norm of the rest of the matrix.

1. Let C be the subset (clique); let $\mathbb{1}_C$ be the characteristic vector. Then (assuming there is not a significant negative contribution)

$$\frac{\|A\mathbb{1}_C\|}{\|\mathbb{1}_C\|} \sim \frac{\sqrt{K(K\frac{\mu}{2})^2}}{\sqrt{K}} = O(K\mu)$$

2. The spectral norm of the random part is $\sqrt{n}\sigma$.

SVD will work whenever $K\mu \gg \sqrt{n}\sigma$,

$$\frac{\mu}{\sigma} \gg \frac{\sqrt{n}}{k}. \tag{1}$$

Now consider thresholding first:

1. If $A_{ij} > \mu$ then set $\widetilde{A}_{ij} = 1$; if $A_{ij} < \mu$ set $\widetilde{A}_{ij} = 0$. In the planted clique the entries are 1 with probability $\frac{1}{2}$; away from it entries are 1 with probability $\sim e^{-\frac{\mu^2}{s\sigma^2}}$.

2. Now we shift back so the mean on the non-clique part is 0. Set $\widetilde{\widetilde{A}} = \widetilde{A} - e^{-\frac{\mu^2}{2\sigma^2}}J$, where J the all 1's matrix.

The planted part has spectral norm $\left(\frac{1}{\sqrt{k}}\right)^2 k^2 = k$. The random part has spectral norm $\precsim \sqrt{n}e^{-\frac{\mu^2}{2\sigma^2}}$.

Thus, after thresholding, we can solve the problem whenever $k \gg \sqrt{r}e^{-\frac{\mu^2}{2\sigma^2}}$, i.e.

$$e^{\frac{\mu^2}{\sigma^2}} \gg \frac{\sqrt{n}}{k}.$$

which is a larger range than in (1). □

The algorithm is the following (informally).

1. (Pick thresholds) For all words j, pick a threshold ζ_j as follows. Take $\zeta_j \in \{0, 1, \ldots, m\}$,

$$\zeta_j = \text{argmax}_j \left\{ \left| \left\{ d : \widetilde{M_{wd}} > \frac{\zeta}{m} \right\} \right| \geq \frac{D}{k} \text{ and } \left| \left\{ d : \widetilde{f_{jd}} = \frac{\zeta}{m} \right\} \right| \leq \varepsilon \frac{D}{k} \right\}.$$

   Then define the threshold matrix

$$T_{wd} := \begin{cases} \sqrt{\zeta_w}, & \text{if } \widetilde{A_{wd}} > \frac{\zeta_w}{m} \text{ and } \zeta_w \text{ is not too small} \\ 0, & \text{otherwise.} \end{cases}$$

2. Now use the Swiss army knife [KK10].[1]
   (a) Take T, do a rank k-SVD, and produce $T^{(k)}$.
   (b) Run a 2-approximation for k-means to get tentative cluster centers.
   (c) Run Lloyd's algorithm on columns S of B, with starting points and centers above.
3. Determine catchwords. (See the paper for details.)
4. Determine the $(1-\delta)$-pure documents and get the topic-word mix.

A key point in the analysis is to show that the thresholding doesn't break the clusters. We need to use the non-local-min assumption.

Proposition 0.4 (Lemma A1 in [BBK])**:** If $\sum_{\zeta \geq \zeta_0} p_j(\zeta, i) \geq \nu$ and $\sum_{\zeta \leq \zeta_0} p_j(\zeta, i) \geq \nu$, then $p_j(\zeta_0, i) \geq \frac{\nu}{m}$.

Proof. Let $f(\zeta) := p_j(\zeta, i)$. One of the following happens.

1. $f(\zeta) \geq f(\zeta - 1)$ for all $n \leq \zeta_i \leq \zeta_0$
2. $f(\zeta + 1) \leq f(\zeta)$ for all $m - 1 \geq \zeta \geq \zeta_0$.

[1] The theorem says that the algorithm works when $> (1-\varepsilon)$ of points satisfy the proximity condition. M_i in cluster T_r satisfies the proximity condition if for any $s \neq r$, the projection of A_i onto the μ_r-to-μ_s line is at least Δ_{rs} closer to μ_r than μ_s. Here $\Delta_{rs} = ck\left(\frac{1}{\sqrt{n_r}+\sqrt{n_s}}\right)\|M - C\|$ where C consists of the cluster centers.

Let's assume (1). Then

$$\zeta_0 p_j(\zeta_0, i) \geq \sum_{\zeta \geq \zeta_0} p_j(\zeta, i) \geq \nu \implies p_j(\zeta_0, i) \geq \frac{\nu}{m}.$$

The other case is similar. □

Lemma 0.5 (Thresholding does not separate dominating topics, Lemma A3 in [BBK])**:** With high probability, for a fixed word w and topic t,

$$\min(\mathbb{P}(\widetilde{A_{wd}} \leq \frac{\zeta_w}{m}; d \in T_t), \mathbb{P}(\widetilde{A_{wd}} > \frac{\zeta_w}{m}, d \in T_t) \leq O(m\varepsilon w_0).$$

where T_t consists of the documents with dominant topic t.

References

[AGH+14] Animashree Anandkumar, Rong Ge, Daniel Hsu, Sham M Kakade, and Matus Telgarsky. Tensor decompositions for learning latent variable models. *arXiv preprint arXiv:1210.7559*, 15:1–55, 2014.

[BBK] Trapit Bansal, C Bhattacharyya, and Ravindran Kannan. A provable SVD-based algorithm for learning topics in dominant admixture corpus. pages 1–22.

[BCMV13] Aditya Bhaskara, Moses Charikar, Ankur Moitra, and Aravindan Vijayaraghavan. Smoothed Analysis of Tensor Decompositions. *arXiv:1311.3651 [cs, stat]*, 2013.

[KK10] Amit Kumar and Ravindran Kannan. Clustering with spectral norm and the k-means algorithm. *Proceedings - Annual IEEE Symposium on Foundations of Computer Science, FOCS*, pages 299–308, 2010.