COMPSCI 650 Applied Information Theory April 19, 2016

Lecture 22

Instructor: Arya Mazumdar *Scribe: Names Redacted*

1 Compressed Sensing

As in the last class, we are discussing compressed sensing, sometimes also known as compressive sampling. We assume the existence of a "signal" vector $\mathbf{x} = (x_1\ x_2\ \cdots\ x_N)$, and our goal is to design an $m \times N$ matrix Φ where m is much smaller than N, so that we can sample $\mathbf{x}$ by computing $\Phi\mathbf{x} = b$, where b has length m. We also want to design Φ in such a way that we can easily reconstruct $\mathbf{x}$ from b.

Without some assumptions on $\mathbf{x}$, this is not possible; there are more vectors of length N than vectors of length m, so we cannot hope to recover every vector in $\mathbb{R}^N$ from a vector in $\mathbb{R}^m$. Thus we assume that $\mathbf{x}$ is "compressible" in some way. Our first order of business will be to define what this means more rigorously.

1.1 Defining Compressibility

The issue of how compressible a sequence is has been well-studied, and is known as the *Kolmogorov complexity* (or Minimum Description Length) of the sequence. Informally, the Kolmogorov complexity of the sequence is the length of the shortest program which produces the sequence as output. Every sequence of length n has at most $O(n)$ Kolmogorov complexity, since a program can always just store the uncompressed sequence and output it, and we can think of sequences with linear Kolmogorov complexity as being nearly random. On the other hand, a sequence like "000000000000000000000000" is highly repetitive, and can be compressed significantly (with, say, a loop).

While this definition of compressibility is theoretically nice, it is hard to work with in practice. It can be very difficult to tell whether a particular sequence has high or low Kolmogorov complexity. As such, we will use a more restrictive definition of compressibility.

One method of compression that we have examined previously is difference coding. Given a vector $\mathbf{x} = (x_1\ x_2\ \cdots\ x_n)$, we encode to the vector $\mathbf{y} = (x_1\ x_2 - x_1\ x_3 - x_2\ \cdots\ x_n - x_{n-1})$, where each entry y_i is the difference of x_i and x_{i-1}. The hope is that adjacent coordinates of $\mathbf{x}$ will have similar values, in which case the values of $\mathbf{y}$ will tend to be small, or even zero. We can represent this encoding process as a linear transformation:

$$\begin{pmatrix} 1 & 0 & 0 & \cdots & 0 & 0 \\ -1 & 1 & 0 & \cdots & 0 & 0 \\ 0 & -1 & 1 & \cdots & 0 & 0 \\ 0 & 0 & \ddots & \ddots & 0 & 0 \\ \vdots & \vdots & \vdots & \ddots & \ddots & \vdots \\ 0 & 0 & 0 & \cdots & -1 & 1 \end{pmatrix} \mathbf{x} = \mathbf{y}.$$

This process is easily decodable by setting $x_1 = y_1$ and then performing an addition at each step. We

can also write this decoding process as the inverse linear transformation:

$$\begin{pmatrix} 1 & 0 & 0 & 0 & \cdots & 0 \\ 1 & 1 & 0 & 0 & \cdots & 0 \\ 1 & 1 & 1 & 0 & \cdots & 0 \\ 1 & 1 & 1 & \ddots & \cdots & 0 \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & 1 & 1 & 1 & \cdots & 1 \end{pmatrix} \mathbf{y} = \mathbf{x}.$$

A method more commonly used in practice than difference coding is to use a Discrete Fourier Transform (DFT). The DFT is an invertible linear transformation, and the matrix has complex entries of a specific form. It turns out to be the case that the DFT is good for compressing many real-world data sources, in that often many coordinates of the compressed vector are close to zero. The matrix of the DFT has the nice property that its columns form an orthonormal basis for $\mathbb{C}^n$. It also has another desirable property that will be important later.

We will take our definition of compressibility for a vector $\mathbf{x}$ to be that when some specific known orthogonal transformation (meaning the columns form an orthonormal basis) is applied to $\mathbf{x}$, the resulting vector has many coordinates that are 0. Specifically, we say that for the known transformation F, $F\mathbf{x}$ is k-sparse, meaning at most k entries of $F\mathbf{x}$ are nonzero. This will lead to a compression rate of $\frac{k}{n}$, if we can decode correctly.

1.2 Spark Condition

Now that we have established a definition of compressibility, we return to the task of constructing a good matrix Φ for compressed sensing. Note that the linear transformation F, while it transforms x to a k-sparse vector, does not reduce the length of $\mathbf{x}$ as we desired. We can take our matrix $\Phi = \Psi F$ for some Ψ, in which case $\Phi\mathbf{x} = \Psi F\mathbf{x} = \Psi\mathbf{y}$, where by our definition of compressibility, $\mathbf{y}$ is k-sparse. Thus we can assume the signal vector $\mathbf{x}$ is k-sparse without any loss of generality.

As a sidebar, this situation is very close to the situation of designing the parity check matrix of a linear code. Recall that in that case, we receive a vector $\mathbf{y} = \mathbf{c} + \mathbf{e}$, where $\mathbf{c}$ is a codeword and $\mathbf{e}$ is the error vector. Then we computed $H\mathbf{y} = H(\mathbf{c} + \mathbf{e}) = H\mathbf{c} + H\mathbf{e} = H\mathbf{e}$, since by definition of the parity check matrix $H\mathbf{c} = 0$. When there is only one error, then $H\mathbf{e}$ is the column of the parity check matrix corresponding to the bit that was flipped from the codeword. In general, we assume that $\mathbf{e}$ is a sparse vector, otherwise decoding is not possible. Then for both compressed sensing and linear decoding, the goal is to design a matrix for recovering some sparse vector. The main difference is that for linear codes the vectors are over finite fields, so there are only a finite number of such vectors, whereas for compressed sensing the vectors are over $\mathbb{R}^n$. In general similar techniques can be used in the two situations, such as the bipartite graph decoding via belief propagation discussed in previous classes. However, the use of real numbers tends to make things more difficult.

In order to recover $\mathbf{x}$ from $\Phi\mathbf{x}$, we need it to be the case that for every other k-sparse vector $\mathbf{x}'$, $\Phi\mathbf{x} \neq \Phi\mathbf{x}'$. If this condition holds (though note that it may be difficult to show, as there are infinitely many vectors $\mathbf{x}'$), then $\mathbf{x}$ will be recoverable, although this does not take into account the potential complexity of the decoding process. Equivalently, we want $\Phi(\mathbf{x} - \mathbf{x}') \neq 0$ for all pairs $\mathbf{x}, \mathbf{x}'$ of k-sparse vectors. $\mathbf{x} - \mathbf{x}'$ is at most $2k$-sparse, so if we can guarantee that $\Phi\mathbf{z} \neq 0$ for every $2k$-sparse vector $\mathbf{z}$ we can be sure that $\Phi\mathbf{x} \neq \Phi\mathbf{x}'$ for any pair. If a $2k$-sparse vector $\mathbf{z}$ has $\Phi\mathbf{z} = 0$, then this corresponds to a linear dependence between at most $2k$ columns of Φ. As such this condition is equivalent to requiring that every set of $2k$ columns of Φ is linearly independent. This condition is sometimes referred to as the *spark condition*, as the *spark* of Φ is the minimum number of linearly dependent columns of Φ.

1.3 Vandermonde Matrices

With the above information, we must next determine what the minimum size of m can be. Let's first consider the following $m \times N$ matrix, consisting of N values $\alpha_i \in \mathbb{R}$, where each α_i is unique. This is known as a Vandermonde matrix.

$$\begin{pmatrix} 1 & 1 & \cdots & 1 \\ \alpha_1 & \alpha_2 & \cdots & \alpha_N \\ \alpha_1^2 & \alpha_2^2 & \cdots & \alpha_N^2 \\ \vdots & \vdots & \cdots & \vdots \\ \alpha_1^{m-1} & \alpha_2^{m-1} & \cdots & \alpha_N^{m-1} \end{pmatrix}$$

A selection of any m columns from this matrix forms a square $m \times m$ submatrix β of the following form:

$$\begin{pmatrix} 1 & 1 & \cdots & 1 \\ \beta_1 & \beta_2 & \cdots & \beta_m \\ \beta_1^2 & \beta_2^2 & \cdots & \beta_m^2 \\ \vdots & \vdots & \cdots & \vdots \\ \beta_1^{m-1} & \beta_2^{m-1} & \cdots & \beta_m^{m-1} \end{pmatrix}$$

We want to show that β is non-singular, so has no linearly dependent columns.

Recall that $\det(\beta) = \prod_{1 \le i < j \le m} (\beta_i - \beta_j)$. Because each β value is a unique real number, this product will never be equal to zero. So, with $m = 2k$ (to satisfy the spark condition), this matrix will work. However, this is an unstable model, as it requires vectors that are *exactly* k-sparse, whereas when working with real world data this will typically not be the case. We are instead interested in vectors which have many entries that are nearly 0.

1.4 Approximate k-Sparsity

In order to avoid this instability, we can hope to approximately recover $\mathbf{x}$ with a degree of error dependent on how close $\mathbf{x}$ is to being k-sparse. Let $\mathbf{x}_k$ be the vector that contains the k largest coefficients in $\mathbf{x}$. We want to approximate $\Phi\mathbf{x} = \mathbf{b}$ with an estimator $\hat{\mathbf{x}}$ such that (for some constant c):

$$\|\hat{\mathbf{x}} - \mathbf{x}\|_2 \le c\|\mathbf{x} - \mathbf{x}_k\|_2$$

Then our estimation error would be upper-bounded by the difference between $\mathbf{x}$ and $\mathbf{x}_k$. From the above, if $\mathbf{x}$ is k-sparse, then the right-hand side of the inequality is 0, and therefore our estimator $\hat{\mathbf{x}}$ is simply $\mathbf{x}$. It turns out that this inequality will hold if Φ satisfies the Restricted Isometry Property, which states that for any $2k$-sparse signal $\mathbf{x}$:

$$(1 - \delta)\|\mathbf{x}\|_2^2 \le \|\Phi\mathbf{x}\|_2^2 \le (1 + \delta)\|\mathbf{x}\|_2^2$$

Where $\delta = \sqrt{2} - 1$. This states that Φ must approximately preserve the ℓ_2 norm in order for the inequality to hold. It is known that a random Gaussian matrix will work for Φ, where $m = O(k \log \frac{N}{k}) \approx O(k \log N)$, but the question of designing good explicit matrices is still largely open.

2 Clustering

We now turn our attention to applications of information theory in machine learning, specifically with regards to clustering. Consider the following Gaussian mixture model:

$$\begin{aligned} f_1(x) &= \frac{1}{\sqrt{2\pi\sigma^2}} e^{\frac{-(x-\mu_1)^2}{2\sigma^2}}, \\ f_2(x) &= \frac{1}{\sqrt{2\pi\sigma^2}} e^{\frac{-(x-\mu_2)^2}{2\sigma^2}}. \end{aligned}$$

Given N i.i.d. samples, we must determine which distribution the samples came from. In other words, given $x_1, x_2, \cdots, x_N$, we want to label each x_i either 1 or 2 to indicate from which distribution it came. We assume the samples come from f_1 with probability p_1, and f_2 with probability p_2.

First we consider the overall model $f(x)$:

$$f(x) = \sum_{i=1}^{2} p_i \frac{1}{\sqrt{2\pi\sigma^2}} e^{\frac{-(x-\mu_i)^2}{2\sigma^2}}.$$

For a given x_i, we can calculate the probability that the label is 1 using Bayes' Rule:

$$p(l_i = 1|x_i) = \frac{f(x_i|l_i = 1)p(l_i = 1)}{f(x)}.$$

To be continued in the next lecture.