University of Wrocław: Algorithms for Big Data (Fall'19) 09/12/2019

Lecture 9: Sparse Recovery

Lecturer: *Przemysław Uznański* Scribe: -

1 Sparse FFT - fast for random signal

Assume we have fourier-sparse signal, where frequencies were sampled uniformly at random, that is $\hat{a}$ support is $\{u_1, \ldots, u_k\}$ where each u_i was picked independently.

We use following:

Theorem 1 (Aliasing Theorem)**.** *Given signal $a = (a_0, \ldots, a_{n-1})$, let L divide n, and consider $b = (a_0, a_L, a_{2L}, \ldots, a_{n-L})$. Then $\hat{b}_i = \frac{1}{L}\sum_{l=0}^{L-1} \hat{a}_{i+l\cdot n/L}$.*

Proof.

$$\begin{aligned}
\sum_{l=0}^{L-1} \hat{a}_{i+l\cdot n/L} &= \sum_{l=0}^{L-1}\sum_{j=0}^{n} a_j \omega^{(i+l\cdot n/L)j} \\
&= \sum_{j=0}^{n} a_j \omega^{ij} \sum_{l=0}^{L-1} \omega^{lj\cdot n/L} \\
&= \sum_{j=0}^{n} a_j \omega^{ij} \sum_{l=0}^{L-1} (\omega^{j\cdot n/L})^l \\
&= \sum_{j=0}^{n} a_j \omega^{ij} \cdot L \cdot [j = 0 \bmod L] \\
&= L \sum_{j=0}^{n/L-1} a_{jL} \omega^{ijL} \\
&= L \sum_{j=0}^{n/L-1} b_j (\omega^{iL})^j \\
&= L\hat{b}_i
\end{aligned}$$

□

We thus use a following algorithm:

- $L = \mathcal{O}(n/k^2)$.
- Let $a' = (a_0, a_L, \ldots, a_{n-L})$ of length $\mathcal{O}(k^2)$.
- Let $a'' = (a_1, a_{L+1}, \ldots, a_{n-L+1})$ of length $\mathcal{O}(k^2)$.

- Compute $\hat{a}'$ and $\hat{a}''$.
- Iterate through non-zero elements of $\hat{a}'$, say $\hat{a}'_u$ and apply two-point algorithm for $\hat{a}'_u$ and $\hat{a}''_u$.

Why does it work (and what is two-point algorithm)? First, $\hat{a}'_u = \frac{1}{L}\sum_{l=0}^{L-1} \hat{a}_{u+ln/L}$. On the other hand, by using timeshift-frequencyshift theorem, a'' is a sampling from shifted-by-one a, so $\hat{a}''_u = \frac{1}{L}\sum_{l=0}^{L-1} \hat{a}_{u+ln/L}\omega^{u+ln/L}$.

Any non-zero element of $\hat{a}$ falls (with 2/3 ppb) into distinct "buckets" (birthday paradox), so $\hat{a}'_u = \frac{1}{L}\hat{a}_{u+in/L}$ for some unknown l, and $\hat{a}''_u = \frac{1}{L}\hat{a}_{u+in/L}\omega^{u+in/L}$. Thus $\hat{a}''_u/\hat{a}'_u = \omega^{u+in/L}$ which reveals i.

Thus total runtime is $\mathcal{O}(k^2 \log k)$ for 2/3 success probability.

2 Sparse recovery

A related problem to what we considered so far. Consider $A \in \mathbb{R}^{m\times n}$ for some small m.

Definition 2 (Exact sparse recovery)**.** *Given Ax where x is k-sparse ($\|x\|_0 \le k$), recover x.*

Definition 3 (Noisy sparse recovery)**.** *Given Ax, find x' that is k-sparse such that*

$$\|x - x'\| \le C \min_{z:\|z\|_0\le k} \|x - z\|.$$

(Note that we do not specify what norms to use in the definition.) Our leverage is that *we can choose* matrix A.

2.1 Exact sparse recovery

Assume the signal is non-negative, that is $x \ge 0$. Assume the simplest scenario: signal is 1-sparse.

Then we just use the simplest matrix, with $m = \log_2 n$, and have it so $A_{i,j} = 1$ iff i-th bit of j is 1 and 0 otherwise. It is easy to see that x_j is "copied" to unique combination (depending on binary representation of j) of coordinates in $y = Ax$.

Example:

$$\begin{bmatrix} 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 0 & 1 & 1 & 0 & 0 & 1 & 1 \\ 1 & 0 & 1 & 0 & 1 & 0 & 1 \end{bmatrix}$$

Property: In such binary matrix columns are unique, and no column is all-0.

Can we do have matrix such that it can distinguish between 1-sparse and 2-sparse? (Doesn't need to uniquely decode 2-sparse, just be able to say that signal is not 1-sparse.)

Have $m = 2\log_2 n$ output values. We just look at non-zero values at the output, ignoring the magnitude. For each bit of position of input, we encode one output for 1 and one for 0 there.

$$\begin{bmatrix} 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 \\ 1 & 1 & 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 \\ 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 \end{bmatrix}$$

Property: In such matrix, no column covers any other column, and any two distinct columns have at least one intersection (common 1).

It is actually better to talk in terms of combinatorial properties. Instead of matrices and columns, we want to talk about sets (column $\to$ set, positions of 1's $\to$ elements of set). So in this setting, we have n sets $F_1, \ldots, F_n \subseteq [m]$, and we require some combinatorial properties of sets. E.g. $\forall_{i \neq j} F_i \not\subseteq F_j$.

Definition 4. *k-separable set family: for any $I_1, I_2 \subseteq [n]$, such that $|I_1| = |I_2| = k$ and $I_1 \neq I_2$, we have*

$$\bigcup_{i \in I_1} F_i \neq \bigcup_{i \in I_2} F_i$$

Essentially: any k-sparse signal gets unique support on the image side, and can be decoded.

Issue: size of codes (value of m), time to decode (better than $\mathcal{O}(n^k)$, can be easily done in $\mathcal{O}(n)$, but there are ways to do it in time $\text{poly}(k, \log n)$).

Equivalent property:

Definition 5. *k-disjoint set family: for any $I \subseteq [n], |I| = k$ and $t \notin I$, there is*

$$F_t \not\subseteq \bigcup_{i \in I} F_i$$

Explicit construction: (c.f. Reed-Solomon codes):

consider polynomials of degree d, modulo p. Each polynomial $f(x) \to$ its graph $\{(0, f(0)), (1, f(1)), \ldots, (p-1, f(p-1))\} \subseteq [p] \times [p]$. There are p^d distinct polynomials. We thus create family of sets, where with each set (indexed by polynomial) we associate its graph. Thus $m = p^2$, $n = p^d$.

What property do we have? Each two sets overlap in at most d values, that is for $f \neq g$ there is $|F_f \cap F_g| \leq d$. Thus we need $dk + 1 \leq p$.

We have $d = \frac{\log n}{\log p}$ and $d \leq p/k$, so best is to set $p \approx \frac{k}{\log k} \log n$ and $d \approx \frac{\log n}{\log k}$. Then $m \approx \frac{k^2}{\log^2 k} \log^2 n$.

Also: randomized construction with $m = \mathcal{O}(k^2 \log n)$. Also, any slow-decoding can be made fast-decoding, at the cost of increasing m by a factor of $\log n$.

Literature: super-imposed codes, k-cover-free families, nonadaptive group testing.

2.2 Count Min as sparse recovery

Recall count min: $t = \mathcal{O}(\varepsilon^{-1})$, $r = \mathcal{O}(\log n)$ hash functions $[n] \to [t]$.

- $x[i]$ is mapped to $\forall_j y[j \cdot t + h_j(i)]$ (linearly)
- $x[i]$ value is queried from median of $\forall_j y[j \cdot t + h_j(i)]$

This of course easily can be written in the matrix form, with $m = rt = \mathcal{O}(\frac{\log n}{\varepsilon})$. The guarantee is that query for $x[i]$ return x' such that $x' = x[i] \pm \varepsilon \|x\|_1$, whp.

Slow sparse recovery Run Count Min with $\varepsilon = \frac{1}{3k}$, and query *every* $x[i]$ for $i \in [n]$. Create a list $L = \{i : |x'[i]| \geq 2\varepsilon \cdot \|x\|_1\}$. (This assumes knowledge of $\|x\|_1$, but this can also be recovered using *linear sketching* from previous lectures.)

All heavy elements (such that $|x[i]| \geq 3\varepsilon \|x\|_1$) are on the list, and every element in L has the property that $|x[i]| \geq \varepsilon \|x\|_1$). Thus $|L| \leq 1/\varepsilon = 3k$ (is sparse). We return sparse vector v where $v[i] = x'[i]$ iff $i \in L$, and otherwise $v[i] = 0$. The guarantee is

$$\|x - v\|_\infty \leq 3\varepsilon \|x\|_1 = \frac{1}{k} \|x\|_1.$$

Improved analysis of Count Min: in the original analysis, we look at expected mass of elements from x colliding with particular hash function, being $\sim \varepsilon\|x\|_1$. However, if we split x into two vectors, $x_{(k)}$ and $x_{\text{tail}(k)}$, where first gets top-k heaviest values, and second gets all the rest, we have the following:

- When hashing $x[i]$, it has $\mathcal{O}(1/k)$ ppb of colliding with each of elements from $x_{(k)}$ over a single hash function.
- So with constant probability (lets say 7/8), $x[i]$ has no collision with any element from $x_{(k)}$.
- The expected mass of collision with $x_{\text{tail}(k)}$ is $\mathcal{O}(1/k)\|x_{\text{tail}(k)}\|$.
- So with ppb lets say 7/8, its at most $\mathcal{O}(1/k)\|x_{\text{tail}(k)}\|$ (with some larger constant than above).
- By union bound, the error is at most $\mathcal{O}(1/k)\|x_{\text{tail}(k)}\|$ with ppb at least 3/4.
- Taking median over $\mathcal{O}(\log n)$ hash function gives whp bound.

This gives us better guarantee:

$$\|x - v\|_\infty \leq \frac{C}{k}\|x_{\text{tail}(k)}\|_1 = \frac{C}{k} \min_{z:\|z\|_0 \leq k} \|x - z\|_1,$$

where v is $3k$-sparse.

Better runtime (polylog): hierarchical structure (binary tree).

2.3 Count-sketch as sparse recovery

Similar analysis (with tail and heavy part of vectors) gives us

$$\|x - v\|_\infty \leq \frac{C'}{\sqrt{k}}\|x_{\text{tail}(k)}\|_2 = \frac{C'}{\sqrt{k}} \min_{z:\|z\|_0 \leq k} \|x - z\|_2$$