Lecture 22: Linearity Testing
Sparse Fourier Transform
• Thu: no class. Happy Thanksgiving!

• Tue, Dec 1\textsuperscript{st}:
  – Sergei Vassilvitskii (Google Research) on MapReduce model and algorithms

• I’m away until next Thu, Dec 3rd
  – Office hours: Tue 2:30-4:30, Wed 4-6pm

• Plan:
  – Linearity Testing (finish)
  – Sparse Fourier Transform
Last lecture

• Linearity Testing:
  – $f: \{-1, +1\}^n$ is linear iff for any $x, y \in \{-1, +1\}^n$, we have:
    • $f(x) \cdot f(y) = f(x \oplus y)$

• Test: repeat $O(1/\epsilon)$ times
  – Pick random $x, y$
  – Verify that $f(x) \cdot f(y) = f(x \oplus y)$

• Main Theorem:
  – If $f$ is $\epsilon$-far from linearity, then $\Pr[\text{test fails}] \geq \epsilon$
Linearity Testing

• Remaining Lemma:
  – Let $T_{xy} = 1$ iff $f(x) \cdot f(y) = f(x \oplus y)$
  – $\Pr[T_{xy} = 1] = \frac{1}{2} + \frac{1}{2} \sum_{S \subseteq [n]} \hat{f}_S^3$
  – Where $\hat{f}_S = \langle f, \chi_S \rangle$ for $\chi_S(x) = \prod_{i \in S} x_i$
Discrete Fourier Transform

• Consider
  – $f: [n] \rightarrow \mathbb{R}$
    • also special case of more general setting:
      • $f: [n]^d \rightarrow \mathbb{R}$

• Will call such function: $x = (x_1, \ldots, x_n)$

• Fourier transform:
  – $\hat{x}_i = \frac{1}{n} \sum_{j \in [n]} x_j \omega^{-ij}$,
  – where $\omega = e^{-2\pi i / n}$ is the $n^{th}$ root of unity
  – $x_i = \sum_{j \in [n]} \hat{x}_j \omega^{ij}$

– Assume: $n$ is power of 2
Why important?

- Imaging
  - MRI, NMR
- Compression:
  - JPEG: retain only high Fourier coefficients
- Signal processing
- Data analysis
- ...

\[ \hat{x}_i = \frac{1}{n} \sum_{j \in [n]} x_j \omega^{-ij}, \]
\[ \omega = e^{-2\pi i/n} \ (n^{th} \ root \ of \ unity) \]
\[ x_i = \sum_{j \in [n]} \hat{x}_j \omega^{ij} \]
Computing

• Naively:
  – $O(n^2)$

• Fast Fourier Transform:
  – $O(n \log n)$ time
  – [Cooley-Tukey 1964]
  – [Gauss 1805]

• One of the biggest open questions in CS:
  – Can we do in $O(n)$ time?

\[
\hat{x}_i = \frac{1}{n} \sum_{j \in [n]} x_j \omega^{-ij},
\]
\[
\omega = e^{-2\pi i/n} \text{ ($n^{th}$ root of unity)}
\]
\[
x_i = \sum_{j \in [n]} \hat{x}_j \omega^{ij}
\]
Sparse Fourier Transform

• Many signals represented well by sparse Fourier transform
• If $\hat{x}$ is sparse,
  – Can we do better?
• YES!
  – $O(k \cdot \log^2 n)$ time possible, assuming $k$ non-zero Fourier coefficients!
  – Sublinear time: just sample a few positions in $x$
• Even when $\hat{x}$ is approximately sparse

\[
\hat{x}_i = \frac{1}{n} \sum_{j \in [n]} x_j \omega^{-ij}, \quad \omega = e^{-2\pi i/n} (n^{th} \text{ root of unity})
\]
\[
x_i = \sum_{j \in [n]} \hat{x}_j \omega^{ij}
\]
Similar to Compressed Sensing

• Sparse Fourier Transform
  – Sparse: $\hat{x} = Fx$
  – Access: $x = F^{-1}\hat{x}$ of dimension $n$
  – $F, \hat{F} =$ concrete matrix

• Compressed Sensing
  – Sparse: $x \in \mathbb{R}^n$
  – Access: $y = Ax \in \mathbb{R}^m$, where $m = O(k \log n)$
  – $A$ is usually designed (though sometimes: random rows of the Fourier matrix)

\[
\hat{x}_i = \frac{1}{n} \sum_{j \in [n]} x_j \omega^{-ij},
\]

\[
\omega = e^{-2\pi i/n} (n^{th} \text{ root of unity})
\]

\[
x_i = \sum_{j \in [n]} \hat{x}_j \omega^{ij}
\]
Warm-up: $k = 1$

- Assume $\hat{x}$ is exactly 1-sp.
  - $\hat{x}_f \neq 0$

- Problem:
  - How many queries into $x$?

- Algorithm:
  - Sample $x_0, x_1$

- $x_0 = a\omega^0 = a$

- $x_1 = a\omega^f$

- $\omega^f = x_1/x_0$
  - Can read off the frequency $f$
What about noise?

• \( x_i = a \omega^{if} + \text{noise} \)

• Problem:
  – Find \( y \) s.t. 1-sparse (in Fourier domain)
  – Best approximation to \( x \)
  \[
  ||\hat{x} - \hat{y}||_2 \leq C \cdot \min_{\hat{y}: \text{1-sparse}} ||\hat{x} - \hat{y}||
  \]
  \[
  ||x - y||_2 \leq C \cdot \min_{y: \text{1-sparse}} ||x - y||
  \]

• Will assume “error” is \( \epsilon \) fraction:
  \[
  \min_{\hat{y}: \text{1-sparse}} ||\hat{x} - \hat{y}||^2 = \sum_{j \neq f} |\hat{x}_j|^2 \leq \epsilon^2 \cdot \hat{x}_f^2 = \epsilon^2 a^2
  \]
  \[
  E_j \left[ |x_j - y_j|^2 \right] \leq \epsilon^2 a^2 \quad \text{(Parseval’s)}
  \]
  – Interesting when \( \epsilon \ll 1 \)
Re-use $k = 1$ algorithm?

- Suppose: $a = 1$
- $x_0 = 1 + \epsilon$
- $x_1 = \omega^f + \epsilon \omega^q$
- So: $\frac{x_1}{x_0} = \frac{1}{1+\epsilon} (\omega^f + \epsilon \omega^q)$

- Error in frequency!
  - Will recover $y = \omega^g$ for $g \neq f$
  - Thus $||\hat{x} - \hat{y}|| \geq ||\hat{x}_f|| = 1$ instead of $O(\epsilon)$...

- Good news: error bounded, up to $\epsilon n$
Algorithm for $k = 1 + \text{noise}$

- $x_i = \omega^{if} + \text{noise}$
- Will find $f$ by binary search!
- Bit 0:
  - $f = 2f_1 + b$ for $b \in \{0,1\}$
- **Claim**: for pure signal $y_i = \omega^{if}$:
  - $y_{n/2} = y_0 \cdot (-1)^b$
  - $y_{n/2+r} = y_r (-1)^b$
- **Proof**:
  - $y_{n/2} = \omega^{f \cdot n/2} = (-1)^f = (-1)^{2f_1} \cdot (-1)^b = (-1)^b y_0$
  - $y_{n/2+r} = \omega^{f \cdot n/2 + fr} = (-1)^f \omega^{fr} = (-1)^b y_r$
- What about noise?
Bit 0 with noise

We have:
- \( x_i = \omega^i(2f_1 + b) + \text{noise} \)
- \( y_i = \omega^i(2f_1 + b) \)
- **Claim**: \( y_{n/2+r} = y_r(-1)^b \)
- \( E_j \left[ |x_j - y_j|^2 \leq \epsilon^2 \right] \) (Parseval’s)

**Algorithm**:
- For \( t \) times:
  - Pick random \( r \in [n] \)
  - Check \( |x_{n/2+r} + x_r| > |x_{n/2+r} - x_r| \): then \( b = 0 \)
  - Otherwise \( b = 1 \)
- Take majority vote

**Claim**: output the right \( b \) with \( 1 - 2^{-\Omega(t)} \) probability

**Proof**:
- Each test:
  - \( x_{n/2+r}, x_r \) are within \( 5\epsilon^2 \) of \( y_{n/2+r}, y_r \) with probability \( 1 - 2 \cdot 1/5 \) (Markov)
  - Hence test works with at least 0.6 probability
- Majority of \( t \) tests work with \( 1 - 2^{-\Omega(t)} \) probability (Chernoff bound concentration)
Bit 1

- Reduce to bit 0 case!
- We have
  - \( x_i = \omega^{i(2f_1+b)} + \text{noise} \)
  - \( y_i = \omega^{i(2f_1+b)} \)
- Suppose \( b = 0 \):
  - \( y_i = \omega^{i \cdot 2f_1} = (\omega^2)^{if_1} = (\omega_{n/2})^{if_1} \)
    - where \( \omega_{n/2} \) is the \((n/2)^{th}\) root of unity
  - Same problem as for Fourier transform over \([n/2]\)!
- Suppose \( b = 1 \):
  - Define \( y'_i = y_i \omega^{-i} \)
    - Then \( y'_i = \omega^{if-i} = \omega^{i(2f_1+1-1)} = \omega^{i \cdot 2f_1} \)
    - Just shifts all frequencies down by one!
  - Continue as above for \( x'_i = x_i \omega^{-i} \)
    - Note: we compute \( x'_i \) on the fly when whenever we query some \( x_i \)
Overall algorithm to recover $f$

- $x_i = \omega^{if} + \text{noise}$
  - Where $f = b_0 + b_1 \cdot 2^1 + b_2 \cdot 2^2 + \cdots + b_{\lg \frac{n}{2}} \cdot \frac{n}{2}$

- Algorithm:
  - Learn $b_0$: take majority of $t$ trials of
    - Pick random $r$
    - Check: $|x_{n/2+r} + x_r| > |x_{n/2+r} - x_r|$
      - Then set $b_0 = 0$
  - Learn $b_1$: take majority of $t$ trials of
    - Pick random $r$
    - Check: $|\omega^{n/4 \cdot b_0} x_{n/4+r} + x_r| > |\omega^{n/4 \cdot b_0} x_{n/4+r} - x_r|$
      - Then set $b_1 = 0$
  - Learn $b_2$: take majority of $t$ trials of
    - Pick random $r$
    - Check: $|\omega^{n/8 \cdot (b_0 + 2b_1)} x_{n/8+r} + x_r| > |\omega^{n/8 \cdot (b_0 + 2b_1)} x_{n/8+r} - x_r|$
      - Then set $b_2 = 0$
  - ...
Wrap-up of the algorithm $k = 1$

- **Correctness:**
  - We learn $O(\log n)$ bits
  - Each needs to succeed with probability $1 - O(1/\log n)$
  - Hence set $t = O(\log \log n)$

- **Overall performance:**
  - Number of samples: $O(\log n \cdot \log \log n)$
  - Same run-time
\( k > 1 \)

- \( x_i = a_1 \omega^{if_1} + a_2 \omega^{if_2} + \cdots a_k \omega^{if_k} + \text{noise} \)

- **Main ideas:**
  - Isolate each frequency
    - Like in CountSketch or compressed sensing!
    - “Throw” frequencies in “buckets”
    - Hope have 1 frequency per “bucket”
  - Throw in buckets:
    - permute the frequencies (pseudo-)randomly
      - Can have frequencies go as \( i \rightarrow ai + b \) for random \( a, b \)
    - partition in blocks: \( \left[ 1, \frac{n}{k} \right], \left[ \frac{n}{k} + 1, \frac{2n}{k} \right], \ldots \)
    - Apply a filter that keeps only the correct block