Lecture 24: Sketching and Nearest Neighbor Search

David Woodruff
Carnegie Mellon University

Slides mostly from Alex Andoni

Sketching

• Random linear projection \( M: \mathbb{R}^n \rightarrow \mathbb{R}^k \)
  that preserves properties of any \( v \in \mathbb{R}^n \)
  with high probability , where \( k \ll n \)

\[
\begin{pmatrix}
M \\
\end{pmatrix}
\begin{pmatrix}
v \\
\end{pmatrix} = \begin{pmatrix}
Mv \\
\end{pmatrix} \rightarrow \text{answer}
\]

• Matrix \( M \) doesn’t depend on \( v \), e.g., \( M \) is a random matrix (typically, we require the entries of \( M \) be \( O(\log n) \) bits)

Estimating the Norm of a Vector

• For a vector \( x \in \mathbb{R}^n \), its (squared) Euclidean norm is \( |x|^2 = \sum_i x_i^2 \)
• Want to output a number \( Z \) for which \( (1 - \epsilon)|x|^2 \leq Z \leq (1 + \epsilon)|x|^2 \)
• Choose a 2-universal independent hash function \( h: [n] \rightarrow [k] \)
• Choose a 4-universal independent hash function \( \sigma: [n] \rightarrow \{-1,1\} \)

\[
\sum_{i:j=h(i)} \sigma_i \cdot x_i = \sum_{i:j=h(i)} \sigma_i \cdot x_i \\
\]

\[
E[Sx^2] = E[\sum_{i:j=h(i)} \delta(h(i) = j)\sigma(i) x_i^2] \\
= \sum_{i,j} x_i x_j E[\delta(h(i) = j)\sigma(i)\sigma(j)] \\
= \sum_{i,j} x_i x_j E[\delta(h(i) = j)\delta(h(j) = j)\sigma(i)\sigma(j)] \\
= \sum_{i,j} x_i x_j E[\delta(h(i) = j)\delta(h(j) = j)] E[\sigma(i)\sigma(j)] \\
= \sum_{i,j} x_i x_j \delta(h(i) = j) \delta(h(j) = j) \frac{x_i x_j}{k} = |x|^2
\]

CountSketch

• CountSketch is a linear map \( S: \mathbb{R}^n \rightarrow \mathbb{R}^k \)
• A row \( i \) of \( S \) is a hash bucket, and \( (Sx)_i \) is the value in the bucket
• Output \( |Sx|^2 \)

\[
E[|Sx|^2] = E[\sum_{i,j} \delta(h(i) = j)\sigma(i) x_i^2] \\
= \sum_{i,j} x_i x_j E[\delta(h(i) = j)\sigma(i)\sigma(j)] \\
= \sum_{i,j} x_i x_j E[\delta(h(i) = j)\delta(h(j) = j)] E[\sigma(i)\sigma(j)] \\
= \sum_{i,j} x_i x_j \delta(h(i) = j) \delta(h(j) = j) \frac{x_i x_j}{k} = |x|^2
\]
Estimating the Norm from CountSketch

- In recitation, you will show $\text{Var}[|Sx|^2] = O(|x|^4/k)$
- By Chebyshev’s inequality,
  $$\Pr\left[ | |Sx|^2 - |x|^2 | > \epsilon |x|^2 \right] \leq \frac{\text{Var}[|Sx|^2]}{\epsilon^2 |x|^4} \leq \frac{1}{10} \text{ if } k = \Theta\left(\frac{1}{\epsilon^2}\right)$$
- If $S$ has $k = \Theta\left(\frac{1}{\epsilon^2}\right)$ rows, can estimate $|x|^2$ from $Sx$ up to a $(1 + \epsilon)$-factor with probability at least $9/10$

Measuring similarity between objects

<table>
<thead>
<tr>
<th>000000</th>
<th>001100</th>
<th>000100</th>
<th>000100</th>
<th>110100</th>
<th>111111</th>
</tr>
</thead>
</table>

objects ⇒ high-dimensional vectors
similarity ⇒ distance b/w vectors

Problem: Nearest Neighbor Search (NNS)

- Preprocess: a set $P$ of points
- Query: given a query point $q$, report a point $p^* \in P$ with the smallest distance to $q$
- Useful for clustering problems, and many other problems on large sets of multi-feature objects
- Applications:
  - speech/image/video/music recognition, signal processing, bioinformatics, etc...

Nearest Neighbor Search (NNS)

- Preprocess: a set $P$ of points
- Query: given a query point $q$, report a point $p \in P$ with the smallest distance to $q$

$n$: number of points
$d$: dimension

Hamming distance
Approximate NNS

• r-near neighbor problem:
  given a new point \( q \), report a point \( p \in D \) s.t. \( d(p,q) \leq cr \)
  if there exists a point at distance \( \leq r \)

• Randomized: a point \( p \) returned with 90% probability

Sketching

• \( W: \mathbb{R}^d \rightarrow \text{short bit-strings} \)
  – given \( W(x) \) and \( W(y) \), can distinguish between:
    • Close: \( d(x,y) \leq r \)
    • Far: \( d(x,y) > cr \)
  – With high success probability: only \( \delta = 1/n^\alpha \) failure prob.

• Hamming distance of bitstrings: \( O(\epsilon^{-2} \cdot \log n) \) bits

NNS: approaches

• Sketch \( W \): uses \( k = O(\epsilon^{-2} \cdot \log n) \) bits
  1: Linear scan
    – Precompute \( W(p) \) for \( p \in D \)
    – Given \( q \), compute \( W(q) \)
    – For each \( p \in D \), estimate distance using \( W(q), W(p) \)

• 2: Exhaustive storage
  – For each possible \( \sigma \in \{0,1\}^k \)
    • compute \( A[\sigma] = \text{point } p \in D \text{ such that } d(W(p), \sigma) < t \)
    – On query \( q \), output \( A[W(q)] \)
  – Space: \( 2^k = \Omega(1/\epsilon^2) \)

Near-linear space and sub-linear query time?

Locality Sensitive Hashing

Random hash function \( h \) on \( \mathbb{R}^d \)
satisfying:
  for close pair (when \( d(q,p) \leq r \))
  \( P_1 = \Pr[h(q) = h(p)] \) is "not-so-small"
  for far pair (when \( d(q,p') > cr \))
  \( P_2 = \Pr[h(q) = h(p')] \) is "small"

Use several hash tables

\( n^p \), where \( p = \frac{\log 1/P_1}{\log 1/P_2} \)
LSH for Hamming space

- **Fact 1:** \( p_g = p_h \)
- **Example:** Hamming space \((0,1)^d\)
  - \( h(p) = p_j \), i.e., choose \( j \)th bit for a random \( j \)
  - \( g(p) \) chooses \( k \) bits at random

\[
P[r = r'] = 1 - e^{-r/d}
\]

\[
P[cr < r < dr] = 1 - e^{-cr/d} - e^{-(d-1)r/d}
\]

\[
g(p) = \{h_1(p), h_2(p), ..., h_k(p)\}
\]

Full Algorithm

- **Data structure** is just \( L = n^p \) hash tables:
  - Each hash table uses a fresh random function \( g_i(p) = \{h_{i,1}(p), ..., h_{i,k}(p)\} \)
  - Hash all dataset points into the table
- **Query:**
  - Check for collisions in each of the hash tables
  - until we encounter a point within distance \( cr \)
- **Guarantees:**
  - Space: \( O(nL\log n) = O(n^{1+p} \log n) \) bits, plus space to store original points
  - Expected Query time: \( O(L \cdot (k + d)) = O(n^p \cdot d) \)
  - 50% probability of success

Choice of parameters \( k, L \)?

- **L** hash tables with \( g(p) = \{h_1(p), ..., h_k(p)\} \)
  - set \( k \) s.t. \( \Pr[\text{collision of far pair}] = P_1^k = 1/n \)
  - \( \Pr[\text{collision of close pair}] = P_2^k = (P_2^k)^k = 1/n^p \)
  - Success probability for a hash table: \( P_1^k \)
  - \( L = O(1/P_1^k) \) tables should suffice
  - **Runtime as a function of** \( P_1, P_2 \) ?
  - \( O\left(\frac{1}{P_1} \text{timeToHash} + nP_2^k d\right) \)
  - **Hence** \( L = O(n^p) \)

Analysis: correctness

- **Let** \( p^* \) be an \( r \)-near neighbor
  - If does not exists, algorithm can output anything
- **Algorithm fails when:**
  - near neighbor \( p^* \) is not in the searched buckets \( g_1(q), g_2(q), ..., g_L(q) \)
- **Probability of failure:**
  - Probability \( q, p^* \) do not collide in a hash table: \( \leq 1 - P_1^k \)
  - Probability they do not collide in \( L \) hash tables at most

\[
\left(1 - P_1^k\right)^L = \left(1 - \frac{1}{n^p}\right)^n \leq 1/e
\]
Analysis: Runtime

- Runtime dominated by:
  - Hash function evaluation: $O(L \cdot k)$ time
  - Distance computations to points in buckets
- Distance computations:
  - Care only about far points, at distance $\geq cr$
  - In one hash table, we have
    - Probability a far point collides is at most $P_{\frac{k}{L}} = \frac{1}{n}$
    - Expected number of far points in a bucket: $n \cdot \frac{1}{n} = 1$
  - Over $L$ hash tables, expected number of far points is $L$
- Total: $O(Lk) + O(Ld) = O(n^p d)$ in expectation