Outline

• Randomized methods - so far
  – SGD with the hash trick
  – Bloom filters
  – count-min sketches

• Today:
  – Review and discussion
  – More on count-min
  – Morris counters
  – locality sensitive hashing
Locality Sensitive Hashing (LSH)

• **Bloom filters:**
  – **set of objects** mapped to a **bit vector**
  – **allows**: add to set, check containment

• **Countmin sketch:**
  – **sparse vector, x** mapped to **small dense matrix**
  – **allows**: recover approximate value of $x_i$
    especially useful for largest values

• **Locality sensitive hash:**
  – **feature vector, x** mapped to bit vector, $b x$
  – **allows**: compute approximate similarity of $b x$
    and by
LSH: key ideas

• Goal:
  – map feature vector $\mathbf{x}$ to bit vector $\mathbf{b}$
  – ensure that $\mathbf{b}$ preserves “similarity”
Random Projections
Random projections

\[ \mathbf{u} \cdot \mathbf{u} + \gamma + \gamma + \gamma + \gamma - \gamma - \gamma - \gamma - \gamma - \gamma \]
Random projections

To make those points “close” we need to project to a direction orthogonal to the line between them.
Any other direction will keep the distant points distant.

So if I pick a random \( \mathbf{r} \) and \( \mathbf{r} \cdot \mathbf{x} \) and \( \mathbf{r} \cdot \mathbf{x}' \) are closer than \( \gamma \) then \textit{probably} \( \mathbf{x} \) and \( \mathbf{x}' \) were close to start with.
LSH: key ideas

• Goal:
  – map feature vector $\mathbf{x}$ to bit vector $\mathbf{b}_x$
  – ensure that $\mathbf{b}_x$ preserves “similarity”

• Basic idea: use random projections of $\mathbf{x}$
  – Repeat many times:
    • Pick a random hyperplane $\mathbf{r}$ by picking random weights for each feature (say from a Gaussian)
    • Compute the inner product of $\mathbf{r}$ with $\mathbf{x}$
    • Record if $\mathbf{x}$ is “close to” $\mathbf{r}$ ($\mathbf{r}.\mathbf{x} \geq 0$)
      – the next bit in $\mathbf{b}_x$
    • Theory says that is $\mathbf{x}'$ and $\mathbf{x}$ have small cosine distance then $\mathbf{b}_x$ and $\mathbf{b}_x'$ will have small Hamming distance
Online Generation of Locality Sensitive Hash Signatures

Benjamin Van Durme and Ashwin Lall
Hamming Distance := $h = 1$
Signature Length := $b = 6$

$$\cos(\theta) \approx \cos\left(\frac{h}{b} \pi\right)$$
$$= \cos\left(\frac{1}{6} \pi\right)$$
32 bit signatures

Approximate Cosine

True Cosine

Cheap

256 bit signatures

Approximate Cosine

True Cosine

Accurate
LSH applications

• Compact storage of data
  – and we can still compute similarities
• LSH also gives very fast ...
  – approx nearest neighbor method
    • just look at other items with \( b_x' = b_x \)
    • also very fast nearest-neighbor methods for Hamming distance
  – approximate clustering/blocking
    • cluster = all things with same \( b_x \) vector
Locality Sensitive Hashing (LSH) and Pooling Random Values
**LSH algorithm**

- Naïve algorithm:
  - Initialization:
    - For \( i = 1 \) to \( \text{outputBits} \):
      - For each feature \( f \):
        » Draw \( r(f,i) \sim \text{Normal}(0,1) \)
  - Given an instance \( \mathbf{x} \)
    - For \( i = 1 \) to \( \text{outputBits} \):
      \[
      \text{LSH}[i] = \begin{cases} 
      \text{sum}(\mathbf{x}[f] \cdot r[i,f] \text{ for } f \text{ with non-zero weight in } \mathbf{x}) > 0 \ ? & 1 : 0 
      \end{cases}
      \]
    - Return the bit-vector LSH
**LSH algorithm**

- But: storing the *k classifiers* is expensive in high dimensions
  - For each of 256 bits, a dense vector of weights for every feature in the vocabulary
- Storing seeds and random number generators:
  - Possible but somewhat fragile
LSH: “pooling” (van Durme)

• Better algorithm:
  – Initialization:
    • Create a pool:
      – Pick a random seed $s$
      – For $i=1$ to poolSize:
        » Draw $\text{pool}[i] \sim \text{Normal}(0,1)$
    • For $i=1$ to outputBits:
      – Devise a random hash function $\text{hash}(i,f)$:
        » E.g.: $\text{hash}(i,f) = \text{hashcode}(f) \text{ XOR } \text{randomBitString}[i]$
  – Given an instance $\mathbf{x}$
    • For $i=1$ to outputBits:
      $\text{LSH}[i] = \text{sum}(\mathbf{x}[f] \times \text{pool}[\text{hash}(i,f) \mod \text{poolSize}]$ for $f \in \mathbf{x}) > 0 \ ? \ 1 : 0$
    • Return the bit-vector LSH
The Pooling Trick

![Graph showing the relationship between mean absolute error and pool size](image)

- **Y-axis**: Mean Absolute Error
- **X-axis**: Pool Size

The graph illustrates how the mean absolute error decreases as the pool size increases.
LSH: key ideas: pooling

• Advantages:
  – with pooling, this is a compact re-encoding of the data
    • you don’t need to store the r’s, just the pool
Locality Sensitive Hashing (LSH) in an On-line Setting
**LSH: key ideas: online computation**

- Common task: distributional clustering
  - for a word $w$, $x(w)$ is sparse vector of words that co-occur with $w$
  - cluster the $w$'s
$\vec{v} \in \mathbb{R}^d$

$\vec{r}_i \sim N(0, 1)^d$

$h_i(\vec{v}) = \begin{cases} 
1 & \text{if } \vec{v} \cdot \vec{r}_i \geq 0, \\
0 & \text{otherwise.}
\end{cases}$

if $\vec{v} = \sum_j \vec{v}_j$
then $\vec{v} \cdot \vec{r}_i = \sum_j \vec{v}_j \cdot \vec{r}_i$

$h_{it}(\vec{v}) = \begin{cases} 
1 & \text{if } \sum_j \vec{v}_j \cdot \vec{r}_i \geq 0, \\
0 & \text{otherwise.}
\end{cases}$
Algorithm 1 Streaming LSH Algorithm

Parameters:
- \( m \): size of pool
- \( d \): number of bits (size of resultant signature)
- \( s \): a random seed
- \( h_1, \ldots, h_d \): hash functions mapping \( (s, f_i) \) to \( \{0, \ldots, m-1\} \)

Initialization:
1: Initialize floating point array \( P[0, \ldots, m-1] \)
2: Initialize \( H \), a hashtable mapping words to floating point arrays of size \( d \)
3: for \( i := 0 \ldots m-1 \) do
4: \( P[i] := \) random sample from \( N(0, 1) \), using \( s \) as seed

Online:
1: for each word \( w \) in the stream do
2: for each feature \( f_i \) associated with \( w \) do
3: for \( j := 1 \ldots d \) do
4: \( H[w][j] := H[w][j] + P[h_j(s, f_i)] \)

Signature Computation:
1: for each \( w \in H \) do
2: for \( i := 1 \ldots d \) do
3: if \( H[w][i] > 0 \) then
4: \( S[w][i] := 1 \)
5: else
6: \( S[w][i] := 0 \)
Experiment

• Corpus: 700M+ tokens, 1.1M distinct bigrams
• For each, build a feature vector of words that co-occur near it, using on-line LSH
• Check results with 50,000 actual vectors

similar to problem we looked at Tuesday using sketches
Experiment
Closest based on true cosine

London
Milan.97, Madrid.96, Stockholm.96, Manila.95, Moscow.95
ASHER0, Champaign0, MANS0, NOBLE0, come0
Prague1, Vienna1, suburban1, synchronism1, Copenhagen2

Closest based on 32 bit sig.'s

Cheap