PART B. GEAR SESSIONS
SESSION 5: ALGORITHMIC TECHNIQUES FOR BIG DATA

Sangmi Lee Pallickara
Computer Science, Colorado State University
http://www.cs.colostate.edu/~cs535

Topics of Today's Class

- Part 1: Locality Sensitive Hashing for Minhash Signatures and The Theory of Locality Sensitive Functions
- Part 2: LSH Families for Other Distance Measures
- Part 3: Geohash and Bloom filter

Planning the computation

- Creating DataFrames
- Generating Hash values
- Calculating signature

<table>
<thead>
<tr>
<th>Data element</th>
<th>S1</th>
<th>S2</th>
<th>S3</th>
<th>S4</th>
<th>H+ and -</th>
<th>2x+1 and -</th>
</tr>
</thead>
<tbody>
<tr>
<td>X</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>2</td>
<td>4</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>3</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>4</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>4</td>
<td>0</td>
<td>0</td>
<td>5</td>
<td>0</td>
<td>3</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>H+ and -</th>
<th>1</th>
<th>3</th>
<th>0</th>
<th>1</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>2x+1 and -</td>
<td>h1</td>
<td>1</td>
<td>3</td>
<td>0</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td></td>
<td>h2</td>
<td>0</td>
<td>2</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
</tbody>
</table>

GEAR Session 5. Algorithmic Techniques for Big Data
Lecture 2. Locality Sensitive Hashing
Locality Sensitive Hashing for Minhash Signatures

General LSH Operations in Apache Spark

- Feature Transformation
  - Add hashed values as a new column
  - Users can specify input and output column names by setting inputCol and outputCol to adjust the dimensionality
- Supports multiple LSH hash tables
  - Users can specify the number of hash tables by setting numHashTables
- Approximate Similarity Join
  - Takes two datasets and approximately returns pairs of rows in the datasets whose distance is smaller than a user-defined threshold
- Approximate Nearest Neighbor Search
  - Takes a dataset (of feature vectors) and a key (a single feature vector), and it approximately returns a specified number of rows in the dataset that are closest to the vector
  - A distance column will be added to the output dataset to show the true distance between each output row and the searched key
Calculating MinHash values with Apache Spark

- https://spark.apache.org/docs/2.2.3/ml-features.html#minhash-for-jaccard-distance

Input sets for MinHash
- Both dense and sparse vectors are supported.
- Sparse vectors are recommended for efficiency.

```scala
val key = Vectors.sparse(6, Seq((1, 1.0), (3, 1.0)))
val mh = new MinHashLSH().setNumHashTables(5).setInputCol("features").setOutputCol("hashes")
val model = mh.fit(dfA)
```

The theory of Locality Sensitive Functions

- The LSH for Minhash signatures is one example of a family of functions (in this case the minhash functions) that can be combined (by the banding technique).
- Distinguish the closer pairs.
- The steepness of the S-curve reflects how effectively we can avoid false positives and false negatives among the candidate pairs.
- How about other families of functions? Can we apply similar approaches?

Conditions for LSHs

- There are three conditions that we need for a family of functions:
  1) They must be more likely to make close pairs be candidate pairs than distant pairs.
  2) They must be statistically independent.
  3) They must be efficient, in two ways:
     - They must be able to identify candidate pairs in time much less than the time it takes to look at all pairs.
     - The combined functions must also take time that is much less than the number of pairs.

Locality-Sensitive Functions

- Purpose:
  - Consider functions that take two items and render a decision about whether these items should be a candidate pair.
- For "hash" items:
  - The decision will be based on whether or not the result is equal.

http://www.cs.colostate.edu/~cs535
Locality-Sensitive Functions --continued
- Let \( d_1 < d_2 \) be two distances according to a target distance measure \( d \)
- A family \( F \) of functions is said to be \( (d_1, d_2, p_1, p_2) \)-sensitive if for every \( f \) in \( F \):
  1. If \( d(x,y) \leq d_1 \), then the probability that \( f(x) = f(y) \) is at least \( p_1 \)
  2. If \( d(x,y) \geq d_2 \), then the probability that \( f(x) = f(y) \) is at most \( p_2 \)

Amplifying a Locality-Sensitive Family
- Suppose that we are given a \( (d_1, d_2, p_1, p_2) \)-sensitive family \( F \)
- Construct a new family \( F' \) by the AND-construction on \( F \)
  - Each member of \( F' \) consists of \( d \) members of \( F \)
  - The members of \( F' \) are independently chosen to make a member of \( F' \)
- Construct a new family \( F'' \) by the OR-construction on \( F' \)
  - Each member of \( F'' \) consists of \( d \) members of \( F' \)
  - \( F'' \) is a \( (d_1, d_2, p_1, p_2) \)-sensitive family

Locality-Sensitive Families for Jaccard Distance
- From the previous example in Week 13-B
- A minhash function \( h \)
  - \( x \) and \( y \) are a candidate pair if and only if \( h(x) = h(y) \)
  - The family of minhash functions is a \( (d_1, d_2, 1 - d_1, 1 - d_2) \)-sensitive family for any \( d_1 \) and \( d_2 \), where \( 0 \leq d_1 \leq d_2 \)
  - If \( d(x,y) \leq d \), where \( d \) is the Jaccard distance, then \( SIM(x, y) = 1 - d(x, y) \leq 1 - d \)
  - Jaccard similarity of \( x \) and \( y \) is equal to the probability that a minhash function will hash \( x \) and \( y \) to the same value

1. LSF Families for Hamming Distance
- Suppose we have a space of \( d \)-dimensional vectors
  - \( h(x,y) \) denotes the Hamming distance between vectors \( x \) and \( y \)
  - The function \( f(x) \) is the \( i \)th bit of vector \( x \)
  - \( f(x) = f(y) \) if and only if vectors \( x \) and \( y \) agree in the \( i \)th position
  - Probability that \( f(x) = f(y) \) for a randomly chosen \( i \) is exactly \( 1 - h(x,y) \)
- The family \( F \) consisting of the functions \( \{f_1, f_2, \ldots, f_d\} \)
  - \( \{d, d, 1 - d, d, 1 - d, \ldots\} \)-sensitive family of hash functions for any \( d \)-valued

2. Random Hyperplanes and the Cosine Distance [1/3]
- What if we use the cosine distance?
  - Two vectors \( x \) and \( y \) that make an angle \( \theta \) between them
  - These vectors may be in a space of many dimensions
- The angle between them is measured in the plane defined by these two vectors
  - Hyperplane through the origin
    - Intersects the plane of \( x \) and \( y \) in a line

http://www.cs.colostate.edu/~cs535

Spring 2020 Colorado State University, page 3
2. Random Hyperplanes and the Cosine Distance [2/3]

- Hyperplane through the origin
  - Intersects the plane of $x$ and $y$ in a line
  - Pick the normal vector $(v)$ to the hyperplane
  - The hyperplane is the set of points whose dot product with $v$ is 0
- Case 1. Pick a vector $v$ that is normal to the hyperplane whose projection is represented with $l$
  - The vector $x$ and $y$ are on different side of hyperplane
  - Dot products $v.x$ and $v.y$ will have different signs

Locality-sensitive family $F$ for the cosine distance

- For a randomly chosen vector $v$, given two vectors $x$ and $y$, say $f(x) = f(y)$ if and only if the dot products $v.x$ and $v.y$ have the same sign
- The parameters are same as the Jaccard-distance family
  - The scale of distances is 0–180 rather than 0–1
- Now, $F$ can be defined as
  - $(d_1, d_2, (180 - d_1) / 180, (180 - d_2) / 180)$-sensitive family of hash functions

2. Random Hyperplanes and the Cosine Distance [3/3]

- Case 2. Pick a vector $v$ that is normal to the hyperplane whose projection is represented with $l$
  - The vector $x$ and $y$ are on the same side of hyperplane
  - Dot products $v.x$ and $v.y$ will have the same signs
  - What is the probability that the randomly chosen vector is normal to a hyperplane that looks like $l$?
    - Assume that we can extend the vectors

3. LSH Families for Euclidean Distance [1/2]

- Let’s start with a 2-dimensional Euclidean space
- Each hash function $f$ in our family $F$ will be associated with a randomly chosen line in this space
- The segments of the line (with the width of $a$) are the buckets into which function $f$ hashes points
  - If $d$ is small than $a$
    - there is a good chance that the two points hash to the same bucket
  - If $d$ is large, the chance that the two points will fall in the same bucket becomes greater
  - If $d$ is 90 degrees, then the two points fall in the same bucket!

3. LSH Families for Euclidean Distance [2/2]

- If $d$ is greater than $a$,
  - To have two points fall in the same bucket
    - it must be $\leq a$
  - If $d \geq 2a$, there is no more than a $1/3$ chance the two points fall in the same bucket
- The family $F$ for the Euclidean distance
  - $(\alpha_2, 2\alpha, 12, 1/3)$-sensitive family of hash function
- For distances up to $\alpha_2$ the probability is at least $1/2$ that two points at that distance will fall in the same bucket
- For distances at least $2\alpha$ the probability points at that distance will fall in the same bucket is at most $1/3$
Geohash

- Latitude/longitude geocode system
- Provides arbitrary precision
- By selecting the length of the code gradually
- All of the geospatial points within a bounding box will be mapped to the same hash output
- You can specify the resolution as well

Colorado State University
- 40.5748° N, 105.0810° W
- Geohash: 9xjqbdqm5h3y1

Resolutions with length of geohash string

http://www.cs.colostate.edu/~cs535

Geohash Encoding

01001011
001

- LAT: 40.5747652 LON: -105.0865006 (CSU)
- Phase 1. Create interleaved bit string geobits[]
  - Even bits are from longitude code, LON
  - Odd bits are from latitude code, LAT
- Step 1.
  - If -180 <= LON <= 0 set geobits[0] as 0
  - If 0 < LON < 180 set geobits[0] as 1
- Step 2.
  - If -90 <= LAT <= 0 set geobits[1] as 0
  - If 0 < LAT < 90 set geobits[1] as 1
Geohash Encoding

- LAT: 40.5747652 LON: -105.0865006 (CSU)

  - Step 3.
    - Since geobits[0] = 0,
      - If -180 <= LON <= 90 set geobits[2] as 0
      - If -90 < LON < 0 set geobits[2] as 1

  - Step 4.
    - Since geobits[1] = 1
      - If 0 <= LAT <= 45 set geobits[3] as 0
      - If 45 < LAT < 90 set geobits[3] as 1

  Repeat this process until your precision requirements are met
  - Currently geohash bit string is 01001

GEAR Session 5. Algorithmic Techniques for Big Data
Lecture 2. Memory Efficient Data Sketch
Bloom Filter

Bloom filter?
- Checking the membership of a set
- Known uses
  - Removing most of the non-membership values
  - Pre-filtering a data set for an expensive set membership check
- Created by Burton Howard Bloom in 1970
- Probabilistic data structure used to test whether a member is an element of a set
- Strong space advantage

Building a Bloom filter
- m
  - The number of bits in the filter
- n
  - The number of members in the set
- p
  - The desired false positive rate
- k
  - The number of different hash functions used to map some element to one of the m bits with a uniform random distribution

http://www.cs.colostate.edu/~cs535
Building a Bloom filter

- \( m = 8 \)
- The number of bits in the filter
- \( n = 3 \)
- The number of members in the set \( T = \{ 5, 10, 15 \} \)
- \( k = 3 \)
- \( h_1(x) = 3x \mod 8 \)
- \( h_2(x) = (2x +3) \mod 8 \)
- \( h_3(x) = x \mod 8 \)

Initial bloom filter: 0 0 0 0 0 0 0 0

Building a Bloom filter

- \( m = 8 \), \( n = 3 \) target set \( T = \{ 5, 10, 15 \} \)
- \( k = 3 \)
- \( h_1(x) = 3x \mod 8 \)
- \( h_2(x) = (2x +3) \mod 8 \)
- \( h_3(x) = x \mod 8 \)

- \( h_1(5) = 7 \), \( h_2(5) = 5 \), \( h_3(5) = 5 \)

Building a Bloom filter

- After \( h_1(5) = 7 \) the \( 7^{th} \) bit is set to 1

Building a Bloom filter

- \( h_2(5) = 5 \)
- After \( h_2(5) = 5 \) the \( 5^{th} \) bit is set to 1

Building a Bloom filter

- \( h_3(5) = 5 \)
- After \( h_3(5) = 5 \) the \( 5^{th} \) bit is set to 1

Building a Bloom filter

- After encoding 5, \( h_1(10) = 6 \), \( h_2(10) = 7 \), \( h_3(10) = 2 \)

Building a Bloom filter

- After \( h_1(10) = 6 \) the \( 6^{th} \) bit is set to 1

Building a Bloom filter

- After \( h_2(10) = 7 \) the \( 7^{th} \) bit is set to 1

Building a Bloom filter

- After \( h_3(10) = 2 \) the \( 2^{nd} \) bit is set to 1

Building a Bloom filter

- After encoding 5, \( h_1(15) = 6 \), \( h_2(15) = 7 \), \( h_3(15) = 7 \)

Building a Bloom filter

- After \( h_1(15) = 5 \) the \( 5^{th} \) bit is set to 1

Building a Bloom filter

- After \( h_2(15) = 1 \) the \( 1^{st} \) bit is set to 1

Building a Bloom filter

- After \( h_3(15) = 7 \) the \( 7^{th} \) bit is set to 1

Applying a Bloom filter

- Is 5 part of set \( T \)?
- \( h_1(5), h_2(5), h_3(5) \) all bits are 1
- 5 is \textit{probably} a part of set \( T \)

Applying a Bloom filter

- Is 8 part of set \( T \)?
- \( h_1(8), h_2(8), h_3(8) \)
- 8 is \textit{NOT} a part of set \( T \)

http://www.cs.colostate.edu/~cs535

Spring 2020 Colorado State University, page 7
Applying a Bloom filter

- Is 9 part of set $T$?
  - $h_1(9), h_2(9), h_3(9)$
  - 9 is NOT a part of set $T$

| After encoding 5, 10 and 15 | 1 | 0 | 1 | 1 | 0 | 1 |
| Check $h_1(9)$ = 3 | 1 | 1 | 1 | 0 | 1 | 0 |
| Check $h_2(9)$ = 5 | 1 | 1 | 1 | 0 | 1 | 0 |
| Check $h_3(9)$ = 1 | 1 | 1 | 1 | 0 | 1 | 0 |

After encoding 5, 10 and 15

Applying a Bloom filter

- Is 7 part of set $T$?
  - $h_1(7), h_2(7), h_3(7)$
  - Th bits are 1
  - 7 is probably a part of set $T$

| After encoding 5, 10 and 15 | 1 | 0 | 1 | 1 | 0 | 1 |
| Check $h_1(7)$ | 1 | 1 | 1 | 0 | 1 | 0 |
| Check $h_2(7)$ | 1 | 1 | 1 | 0 | 1 | 0 |
| Check $h_3(7)$ | 1 | 1 | 1 | 0 | 1 | 0 |

False positive rate

$$fp_r = \left(1 - \left(1 - \frac{m}{n}\right)^k\right)^k = \left(1 - e^{-kn/m}\right)^k$$

- $m$=number of bits in the filter
- $n$=number of elements
- $k$=number of hashing functions

The false positive probability $p_r$ as a function of number of elements $n$ in the filter and the filter size $m$.

False positive rate

- A bloom filter with an optimal value for $k$ and 1% error rate only needs 9.6 bits per key.
- Add 4.8 bits/key and the error rate decreases by 10 times
- 10,000 words with 1% error rate and 7 hash functions
  - ~12KB of memory
- 10,000 words with 0.1% error rate and 11 hash functions
  - ~18KB of memory

How big should I make my Bloom Filter?

- Try various values of $k$ and $m$
  - To achieve target false-positive rate $(1-fp_r)$
- Then, how many hash functions should I use?
  - The more hash functions you have
    - The slower your bloom filter
    - The quicker it fills up
    - If you have few hash functions
      - Too many false positives
  - Given an $m$ and an $n$, the optimal value of $k$
    - $(m/n)\ln(2)$

Questions?