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

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

FAQs

• Please check the announcement for the term project deadlines
### 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>Row (element)</th>
<th>S1</th>
<th>S2</th>
<th>S3</th>
<th>S4</th>
<th>X+1 mod 5</th>
<th>3x +1 mod 5</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>2</td>
<td>4</td>
</tr>
<tr>
<td>2</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>3</td>
<td>2</td>
</tr>
<tr>
<td>3</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>4</td>
<td>0</td>
</tr>
<tr>
<td>4</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>3</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>S1</th>
<th>S2</th>
<th>S3</th>
<th>S4</th>
</tr>
</thead>
<tbody>
<tr>
<td>h1</td>
<td>1</td>
<td>3</td>
<td>0</td>
</tr>
<tr>
<td>h2</td>
<td>0</td>
<td>2</td>
<td>0</td>
</tr>
</tbody>
</table>

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

Input sets for MinHash

- Binary vectors
- vector indices represent the elements themselves
- non-zero values in the vector represent the presence of that element in the set
- Both dense and sparse vectors are supported,
  - sparse vectors are recommended for efficiency

- `Vectors.sparse(10, Array((2, 1.0), (3, 1.0), (5, 1.0)))`
  - There are 10 elements in the space
  - elem 2, elem 3 and elem 5
  - All non-zero values are treated as binary "1" values

Example

```scala
val dfA = spark.createDataFrame(Seq(  
  (0, Vectors.sparse(6, Seq((0, 1.0), (1, 1.0), (2, 1.0))))),  
  (1, Vectors.sparse(6, Seq((2, 1.0), (3, 1.0), (4, 1.0))))),  
  (2, Vectors.sparse(6, Seq((0, 1.0), (2, 1.0), (4, 1.0)))))).toDF("id", "features")

val dfB = spark.createDataFrame(Seq(  
  (3, Vectors.sparse(6, Seq((1, 1.0), (3, 1.0), (5, 1.0))))),  
  (4, Vectors.sparse(6, Seq((2, 1.0), (3, 1.0), (5, 1.0))))),  
  (5, Vectors.sparse(6, Seq((1, 1.0), (2, 1.0), (4, 1.0)))))).toDF("id", "features")

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)
```
GEAR Session 5. Algorithmic Techniques for Big Data
Lecture 2. Locality Sensitive Hashing
The Theory of Locality Sensitive Functions

The LSH for Minhash signatures

- 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
   - They must be combinable to build functions that are better at avoiding false positives and negatives, and 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
   - \( f(x) \) will "hash" items
     - The decision will be based on whether or not the result is equal

A family of LSFs
   - A collection of these LSFs
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 \( p_2 \)

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 \( 0 \leq d_1 < d_2 \leq 1 \)
• if \( d(x,y) \leq d_1 \), where \( d \) is the Jaccard distance, then \( \text{SIM}(x,y) = 1 - d(x,y) \geq 1 - d_1 \)
• Jaccard similarity of \( x \) and \( y \) is equal to the probability that a minhash function will hash \( x \) and \( y \) to the same value
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 \(r\) members of \(F\)
  - \(F'\) is a \((d_1, d_2, (p_1)^r, (p_2)^r)\)-sensitive family
    - 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 \(r\) members of \(F\)
  - \(F'\) is a \((d_1, d_2, 1-(1-p_1)^r, 1-(1-p_2)^r)\)-sensitive family

---

**GEAR Session 5. Algorithmic Techniques for Big Data**

**Lecture 2. Locality Sensitive Hashing**

**LSH Families for Other Distance Measures**

1. Hamming Distance
2. Cosine Distance
3. Euclidean Distance
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 $f_i(x)$ is the $i$th bit of vector $x$
  - $f_i(x) = f_i(y)$ if and only if vectors $x$ and $y$ agree in the $i$th position
  - Probability that $f_i(x) = f_i(y)$ for a randomly chosen $i$ is exactly $1 - h(x, y)/d$

- The family $F$ consisting of the functions $\{f_1, f_2, \ldots, f_d\}$
  - $(d_1, d_2, 1 - d_1/d, 1 - d_2/d)$-sensitive family of hash functions for any $d_1 < d_2$

2. Random Hyperplanes and the Cosine Distance

- 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
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 \( \mathbf{v} \) to the hyperplane
  - The hyperplane is then the set of points whose dot product with \( \mathbf{v} \) is 0
- **Case 1.** Pick a vector \( \mathbf{v} \) that is normal to the hyperplane whose projection is represented with \( l_1 \)
  - The vector \( x \) and \( y \) are on different side of hyperplane
    - Dot products \( \mathbf{v} \cdot x \) and \( \mathbf{v} \cdot y \) will have different signs

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

- **Case 2.** Pick a vector \( \mathbf{v} \) that is normal to the hyperplane whose projection is represented with \( l_2 \)
  - The vector \( x \) and \( y \) are on the same side of hyperplane
    - Dot products \( \mathbf{v} \cdot x \) and \( \mathbf{v} \cdot y \) will have the same signs
  
- What is the probability that the randomly chosen vector is normal to a hyperplane that looks like \( l_1 \)?
  - \( \frac{\theta}{180} \)
- Assume that we can extend the vectors
Locality-sensitive family $F$ for the cosine distance

- For a randomly chosen vector $v_f$, given two vectors $x$ and $y$, say $f(x) = f(y)$ if and only if the dot products $v_f \cdot x$ and $v_f \cdot y$ 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.

3. LSH Families for Euclidean Distance

- 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 $\theta$ is large, the chance that the two points will fall in the same bucket becomes greater.
  - If $\theta$ is 90 degrees, then the two points fall in the same bucket.
3. LSH Families for Euclidean Distance

If $d$ is greater than $a$,
- To have two points fall in the same bucket
  - $d \cos \theta \leq a$
- 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 $= (a/2, 2a, 1/2, 1/3)$-sensitive family of hash function
For distances up to $a/2$ the probability is at least $1/2$ that two points at that distance will fall in the same bucket
For distances at least $2a$ 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

![World Map with Geohash Resolution Examples](http://www.cs.colostate.edu/~cs535)
Resolutions with length of geohash string

01
0
00

Resolutions with length of geohash string

01001011
0
00

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

Spring 2020 Colorado State University, page 14
Resolutions with length of geohash string

Geohash Encoding

- 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<=22.5 set geobits[3] as 0
  • If 22.5< LAT<45 set geobits[3] as 1
**Geohash Encoding**

- LAT: 40.5747652 LON:-105.0865006 (CSU)
- Repeat this process until your precision requirements are met
- Currently geohash bit string is 01001

**Phase 2.**
- Encode every five bits from the left side, by mapping to the table,

\[
\text{9xjqbdqm5h3y1}
\]

First five bits "01001" are mapped to "9"

<table>
<thead>
<tr>
<th>Decimal</th>
<th>0-9</th>
<th>10</th>
<th>11</th>
<th>12</th>
<th>13</th>
<th>14</th>
<th>15</th>
<th>16</th>
<th>17</th>
</tr>
</thead>
<tbody>
<tr>
<td>Base 32</td>
<td>0-9 b c d e g g j</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Decimal</th>
<th>18</th>
<th>19</th>
<th>20</th>
<th>21</th>
<th>22</th>
<th>23</th>
<th>24</th>
<th>25</th>
<th>26</th>
</tr>
</thead>
<tbody>
<tr>
<td>Base 32</td>
<td>k m n p s r t u</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Decimal</th>
<th>27</th>
<th>28</th>
<th>29</th>
<th>30</th>
<th>31</th>
</tr>
</thead>
<tbody>
<tr>
<td>Base 32</td>
<td>v w x y z</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Bloom filter?

• Checking the membership of a set

• Known uses
  • Removing most of the non-membership values
  • Prefiltering 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
Building a Bloom filter

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

<table>
<thead>
<tr>
<th>Initial bloom filter</th>
<th>0</th>
<th>0</th>
<th>0</th>
<th>0</th>
<th>0</th>
<th>0</th>
<th>0</th>
</tr>
</thead>
</table>

Building a Bloom filter

- $m = 8$, $n = 3$ target set $T = \{5, 10, 15\}$
- $k = 3$
  - $h_1(x) = 3x \mod 8$
  - $h_2(x) = (2x + 3) \mod 8$
  - $h_3(x) = x \mod 8$
  - $h_1(5) = 7$, $h_2(5) = 5$, $h_3(5) = 5$

<table>
<thead>
<tr>
<th>Initial bloom filter</th>
<th>0</th>
<th>0</th>
<th>0</th>
<th>0</th>
<th>0</th>
<th>0</th>
<th>0</th>
</tr>
</thead>
</table>

After $h_1(5) = 7$

<table>
<thead>
<tr>
<th>After $h_1(5) = 7$</th>
<th>1</th>
<th>0</th>
<th>0</th>
<th>0</th>
<th>0</th>
<th>0</th>
<th>0</th>
</tr>
</thead>
</table>

After $h_2(5) = 5$

<table>
<thead>
<tr>
<th>After $h_2(5) = 5$</th>
<th>1</th>
<th>0</th>
<th>1</th>
<th>0</th>
<th>0</th>
<th>0</th>
<th>0</th>
</tr>
</thead>
</table>

After $h_3(5) = 5$

| After $h_3(5) = 5$ | 1 | 0 | 1 | 0 | 0 | 0 | 0 |
Building a Bloom filter

- \( m = 8 \), \( n = 3 \) target set \( T = \{ 5, 10, 15 \} \)
- \( k = 3 \)
  - \( h1(x) = 3x \mod 8 \)
  - \( h2(x) = (2x + 3) \mod 8 \)
  - \( h3(x) = x \mod 8 \)
  - \( h1(10) = 6 \), \( h2(10) = 7 \), \( h3(10) = 2 \)

After encoding 5

\[
\begin{array}{cccccccc}
1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\
\end{array}
\]

After \( h1(10) = 6 \)

\[
\begin{array}{cccccccc}
1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 \\
\end{array}
\]

After \( h2(10) = 7 \)

\[
\begin{array}{cccccccc}
1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 \\
\end{array}
\]

After \( h3(10) = 2 \)

\[
\begin{array}{cccccccc}
1 & 1 & 1 & 0 & 0 & 1 & 0 & 0 \\
\end{array}
\]

Building a Bloom filter

- \( m = 8 \), \( n = 3 \) target set \( T = \{ 5, 10, 15 \} \)
- \( k = 3 \)
  - \( h1(x) = 3x \mod 8 \)
  - \( h2(x) = (2x + 3) \mod 8 \)
  - \( h3(x) = x \mod 8 \)
  - \( h1(15) = 5 \), \( h2(15) = 7 \), \( h3(15) = 7 \)

After encoding 5 and 10

\[
\begin{array}{cccccccc}
1 & 1 & 1 & 0 & 0 & 1 & 0 & 0 \\
\end{array}
\]

After \( h1(15) = 5 \)

\[
\begin{array}{cccccccc}
1 & 1 & 1 & 0 & 0 & 1 & 0 & 0 \\
\end{array}
\]

After \( h2(15) = 1 \)

\[
\begin{array}{cccccccc}
1 & 1 & 1 & 0 & 0 & 1 & 1 & 0 \\
\end{array}
\]

After \( h3(15) = 7 \)

\[
\begin{array}{cccccccc}
1 & 1 & 1 & 0 & 0 & 1 & 1 & 0 \\
\end{array}
\]
Applying a Bloom filter

- Is 5 part of set $T$?
  - $h1(5), h2(5), h3(5)$th bits are 1
    - 5 is probably a part of set $T$

<table>
<thead>
<tr>
<th>After encoding 5, 10 and 15</th>
<th>1</th>
<th>1</th>
<th>1</th>
<th>0</th>
<th>0</th>
<th>1</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>Check $h1(5) = 7$</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>Check $h2(5) = 5$</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>Check $h3(5) = 5$</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

Applying a Bloom filter

- Is 8 part of set $T$?
  - $h1(8), h2(8), h3(8)$
    - 8 is NOT a part of set $T$

<table>
<thead>
<tr>
<th>After encoding 5, 10 and 15</th>
<th>1</th>
<th>1</th>
<th>1</th>
<th>0</th>
<th>0</th>
<th>1</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>Check $h1(8) = 0$</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>Check $h2(8) = 3$</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>Check $h3(8) = 0$</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>
Applying a Bloom filter

- Is 9 part of set $T$?
- $h1(9)$, $h2(9)$, $h3(9)$
- 9 is NOT a part of set $T$

After encoding 5, 10 and 15

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>1</th>
<th>1</th>
<th>0</th>
<th>0</th>
<th>1</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
</table>

Check $h1(9) = 3$

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>1</th>
<th>1</th>
<th>0</th>
<th>0</th>
<th>1</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
</table>

Check $h2(9) = 5$

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>1</th>
<th>1</th>
<th>0</th>
<th>0</th>
<th>1</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
</table>

Check $h3(9) = 1$

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>1</th>
<th>1</th>
<th>0</th>
<th>0</th>
<th>1</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
</table>

Applying a Bloom filter

- Is 7 part of set $T$?
- $h1(7)$, $h2(7)$, $h3(7)$th bits are 1
  - 7 is probably a part of set $T$

After encoding 5, 10 and 15

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>1</th>
<th>1</th>
<th>0</th>
<th>0</th>
<th>1</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
</table>

Check $h1(7) = 7$

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>1</th>
<th>1</th>
<th>0</th>
<th>0</th>
<th>1</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
</table>

Check $h2(7) = 1$

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>1</th>
<th>1</th>
<th>0</th>
<th>0</th>
<th>1</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
</table>

Check $h3(7) = 7$

|   | 1 | 1 | 1 | 0 | 0 | 1 | 1 | 0 |
**False positive rate**

\[
fpr = \left(1 - \left(1 - \frac{1}{m}\right)^{kn}\right)^k \approx \left(1 - e^{-kn/m}\right)^k
\]

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

The false positive probability \( p \) as a function of number of elements \( n \) in the filter and the filter size \( m \).

https://en.wikipedia.org/wiki/Bloom_filter

**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 $((1-e^{-kn/m})^k)$

- 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?