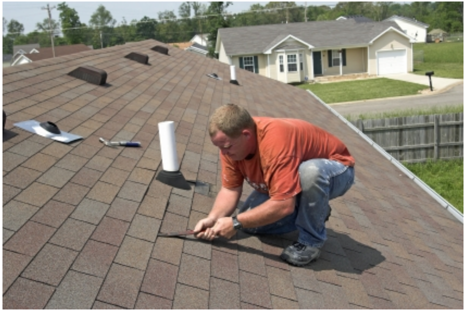
CSCI 4022 Fall 2021
Shingling and Minhashing



Example: prove that the edit distance is a proper distance measure.

Announcements and To-Dos

Announcements:

1. HW 1 up!

Minute form comments: mostly seemed all good! Please don't hesitate to me. Couple implementation comments, especially on .index. Yes, dictionaries would have maye been better! We'll formally introduce dictionaries soon - no prior knowledge is assumed.

Other non-Euclidean distances

1) $d \geq 0$ ✓

2) $d(x,y) = d(y,x)$

3) $d(x,x) = 0$

A if $d(x,y) = 0 \Rightarrow x = y$

B if $x = y \Rightarrow d(x,y) = 0$

Example: prove that the edit distance is a proper distance measure.

Other non-Euclidean distances

Example: prove that the edit distance is a proper distance measure.

Solution:

1. No negative distances.
2. Distances are only zero from a point to itself.
3. Distance is symmetric.
4. Distances satisfy the **triangle inequality**.

Other non-Euclidean distances

Example: prove that the edit distance is a proper distance measure.

Solution:

1. $d(x, y) \geq 0$: $\boxed{\checkmark}$. We're counting operations: obviously can't be negative!
2. $d(x, y) = 0 \leftrightarrow x = y$: $\boxed{\checkmark}$. Two statements: if $x = y$, clearly the edit distance is zero. On the other hand, if we don't edit a string x then we get the same string, so no edit distance also implies $x = y$.
3. $d(x, y) = d(y, x)$ $\boxed{\checkmark}$ Given a sequence of edits to turn x to y, we can turn this into a same-length sequence of edits to get y to x by changing each insertion to a deletion and vice versa. (in other words: each operation is invertible!)
4. $d(x, y) \leq d(x, z) + d(z, y)$ $\boxed{\checkmark}$. If we turn x to y by first turning x to z, then it the number of edits can't be less than going directly to y.

So edit distance is a proper distance measure.

Better Proofs

To be quite honest, the argument on the prior slide for the **triangle inequality** was a little hand-wavy. What's a more formal proof for $d(x, y) \leq d(x, z) + d(z, y)$?

Proof by Contradiction (for a $p \to q$ proof)

1. Assume p *and* $\neg q$: "Suppose edit distance is distance measure (*p*) that so that $d(x, y) > d(x, z) + d(z, y)$. (*not q*)"
2. By definition of edit distance, the *minimum* edits to get from string x to string z is $d(x, z)$, and the *minimum* edits to get from string z to string y is $d(y, z)$.
3. We can get from string x to string y by changing x to z and then z to y, so **one** way to get from x to y is via z, at a total distance of $d(x, z) + d(z, y)$.
4. Because edit distance is the *minimum* edits necessary, the actual distance $d(x, y)$ can not be more than the *possible* distance $d(x, z) + d(z, y)$. In other words, $d(x, y) \leq d(x, z) + d(z, y)$ which contradicts our assumptions of p *and* $\neg q$, so we must conclude that $p \to q$ (or if the metric is edit distance, it satisfies the triangle inequality.)

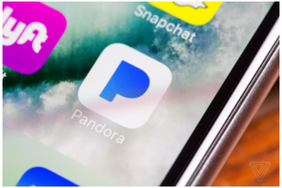
Last time: Similarity

Many problems - both in this course and in general - can be expressed as the task of finding **similar** elements.

We often phrase this as finding "near-neighbors," which may exist in **high-dimensional** spaces.

Examples:

1. Documents with similar words/code
2. Users who watch similar movies
3. Songs that have similar attributes
4. Images with similar features
5. etc., etc.



Last time: Distance Measures

We first considered distances in **Euclidean** spaces, like the general L_r-norm:

$$d(x, y) = \left(\sum_{i=1}^{n} (x_i - y_i)^r \right)^{1/r}$$

which worked well for vectors in a continuous space.

For comparing *sets*, we saw **Jaccard similarity**,

$$\text{sim(S,T)} = \frac{|S \cap T|}{|S \cup T|}$$

and the associated **distance**: $d = 1 - \text{sim}$.

From there we introduced two other non-Euclidean distances meant to describe strings or lists:
edit distance:, # of deletions/insertions required to move from one string to another;
Hamming distance, number of components by which two strings differ

Hi!

users (document)

item (in set)

1	0
0	1
1	1

Today's plan

Today we discuss the plan of how to implement Jaccard Similarity on a *large data* scale. This is a three part-process.

1. First, we **shingle:** we map a document into a set of numbers.
2. Second, we discuss **approximate Jaccard similarity**: by considering the properties of reshuffling/permuting bitstrings representing sets.
3. Finally, we discuss **approximate permutations** via modular transformations.

Document Similarity

Jaccard similarity was a measure for the similarity between *sets*. Consider the task of measure similarity between **documents.** Applications might include:

1. plagiarism detection
2. mirroring web pages
3. finding news articles from the same sources or authors

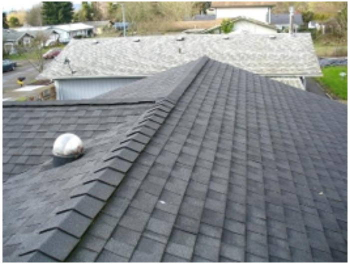
The first question: how can we represent an entire document as a set?

Document Similarity

Jaccard similarity was a measure for the similarity between *sets*. Consider the task of measure similarity between **documents.** Applications might include:

1. plagiarism detection
2. mirroring web pages
3. finding news articles from the same sources or authors

The first question: how can we represent an entire document as a set?



A document is just a long string of *characters*. We can

- Represent a document as a collection of *substrings* of those characters.
- these substrings of characters are called *shingles*

k-shingles

Definition: a *k-shingle* for a document is any subtring of length k found in that document.

Example: Suppose we have a document D that consists only of the word "remember" and we pick $k = 2$. What is the entire set of all 2-shingles for D?

re
em
me
mb
be
er

k-shingles

Definition: a *k-shingle* for a document is any subtring of length k found in that document.

Example: Suppose we have a document D that consists only of the word "remember" and we pick $k = 2$. What is the entire set of all 2-shingles for D?

Solution: $\{$re, em, me, mb, be, er$\}$.
Note:

1. "em" would appear twice... but this is a *set*, so we don't include duplicates?
2. A variant would consider a *bag* of 2-shingles, which might include duplicates.
3. We generally want characters to be case-insensitive and not using punctuation. So we pre-process by lowering cases and replace any spaces, tabs, newlines, etc. with a general white-space character.

k-shingles

Suppose document D_1 consists only of the word "remember" and document D_2 consists only of the word "emperor." Representing the documents as their sets of 2-shingles (call them S_1 and S_2), what is the Jaccard similarity of D_1 and D_2?

6 6

re

em em

me mp

mb pe

be er

er ro

or

overlap: 2

total: 6 + 6 − 2

$\Rightarrow J(D_1, D_2) = 2/10$

k-shingles

Suppose document D_1 consists only of the word "remember" and document D_2 consists only of the word "emperor." Representing the documents as their sets of 2-shingles (call them S_1 and S_2), what is the Jaccard similarity of D_1 and D_2?

Solution:
$S_1 =$\{re, em, me, mb, be, er\}.
$S_2 =$\{em, mp, pe, er, ro, or\}.
$|S_1 \cup S_2| = |$\{re, em, me, mb, be, er, mp, pe, ro , or\}$|$=10.
$|S_1 \cap S_2| = |$\{em, er\}$|$=2.
So the similarity is 2/10=2.

k-shingles

The next task: how do we choose k to represent a document?

1. If k is too small, most sequences of k chars might appear in most documents. As a result, everything looks similar.
2. If k is too large, many documents might share no sequences as all and everything will look different.

General rule: choose k just large enough such that the probability of any given shingle appearing in any given document is low.

k-shingles

The next task: how do we choose k to represent a document?

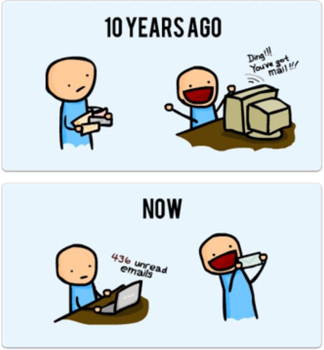
1. If k is too small, most sequences of k chars might appear in most documents. As a result, everything looks similar.
2. If k is too large, many documents might share no sequences as all and everything will look different.

General rule: choose k just large enough such that the probability of any given shingle appearing in any given document is low.

Cool, but what does “low” mean?

k-shingles

General rule: choose k just large enough such that the probability of any given shingle appearing in any given document is low.



Example: suppose our corpus of documents to consider is a bunch of emails.

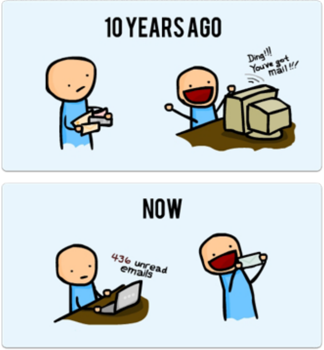
Suppose we consider only letters, all converted to lower case.

With 27 total characters (white space, too!) we'd have $27^k \approx$ 14.3 million total possible shingles.

Since typical e-mails are typically much shorter than this, we expect $k = 5$ to work well... and it does!

k-shingles

Example: suppose our corpus of documents to consider is a bunch of emails.



But are all characters (letters/spaces) equally likely in an e-mail? **Definitely not!**

Common letters and blanks are far more common!

As a result, we might revise our heuristic: if only 20 of the characters are "common," we'd think of only around 20^k common shingles. Still, $20^5 \approx$ 3.2 million, which means $k = 5$ for e-mails is likely ok. Longer documents like essays, code, etc. might require e.g. $k = 9$.

hashing

Example: Suppose document D_1 is the word "banana"; document D_2 is the word "bandit,"; and document D_3 is the work "brand." Representing the documents only as their **characteristic matrix**, what are the Jaccard similarities of the 3 documents?

Jaccard: $\dfrac{|S \cap T|}{|S \cup T|} = \dfrac{\text{"1" \& "1"}}{\text{at least 1 "1"}}$

	banana	bandit	brand
ba	1	1	0
an	1	1	1
na	1	0	0
nd	0	1	1
di	0	1	0
it	0	1	0
br	0	0	1
ra	0	0	1

$\dfrac{2}{2+4} = 2/6$

hashing

Example: Suppose document D_1 is the word “banana”; document D_2 is the word “bandit,”; and document D_3 is the work “brand.” Representing the documents only as their **characteristic matrix**, what are the Jaccard similarities of the 3 documents?

	banana	bandit	brand

Definition: The *characteristic matrix* of a set of shingles is the matrix where each row is a shingle and each column is a document. Row i and col j is a 1 if shingle i appears in document j.

Solution:

hashing

Example: Suppose document D_1 is the word "banana"; document D_2 is the word "bandit,"; and document D_3 is the work "brand." Representing the documents only as their **characteristic matrix**, what are the Jaccard similarities of the 3 documents?

Definition: The *characteristic matrix* of a set of shingles is the matrix where each row is a shingle and each column is a document. Row i and col j is a 1 if shingle i appears in document j.

	banana	bandit	brand
ba	1	1	0
an	1	1	1
na	1	0	0
nd	0	1	1
di	0	1	0
it	0	1	0
br	0	0	1
ra	0	0	1

Solution: The Jaccard similarity for columns 1 and 2 is 2/6=1/3.

hashing

Often, we *hash* the shingles instead of storing the substrings themselves.
A hash:

1. Map each substring of length k to an integer
2. Each document is now represented by a set of integers or bucket numbers/indices
3. We have 8 shingles here, so we need at least 8 buckets. We want more to be safe! Suppose we choose 19.

	banana	bandit	brand
ba	1	1	0
an	1	1	1
na	1	0	0
nd	0	1	1
di	0	1	0
it	0	1	0
br	0	0	1
ra	0	0	1

Definition: *A hash function* is a modular function to map string characters to buckets. It may use e.g. ASCII values of those characters. We could use:

$$h(k) := (\text{sum of ASCII values of characters in shingle}) \mod 19$$

hashing

Often, we *hash* the shingles instead of storing the substrings themselves. We could use:

$$h(k) := \text{(sum of ASCII values of characters in shingle)} \mod 19$$

Which results in...

1. ba $\rightarrow$ ord(b) + ord(a) $\mod 19 = 5$
2. an $\rightarrow$ ord(a) + ord(n) $\mod 19 = 17$
3. na $\rightarrow$ ord(n) + ord(a) $\mod 19 = 17$

	banana	bandit	brand
ba	1	1	0
an	1	1	1
na	1	0	0
nd	0	1	1
di	0	1	0
it	0	1	0
br	0	0	1
ra	0	0	1

hashing

Often, we *hash* the shingles instead of storing the substrings themselves. We could use:

$$h(k) := \text{(sum of ASCII values of characters in shingle)} \mod 19$$

	banana	bandit	brand
ba	1	1	0
an	1	1	1
na	1	0	0
nd	0	1	1
di	0	1	0
it	0	1	0
br	0	0	1
ra	0	0	1

Which results in...

1. ba $\rightarrow$ ord(b) + ord(a) $\mod 19 = 5$
2. an $\rightarrow$ ord(a) + ord(n) $\mod 19 = 17$
3. na $\rightarrow$ ord(n) + ord(a) $\mod 19 = 17$

Ack! A **collision!**.

hashing

To avoid collisions, we often use hash functions that stretch the strings out to much larger numbers.

$10 \cdot \#1 + 7 \cdot \#2 + 4 \cdot \#3 + \#4$

$$h(k) := 2 \cdot \text{first ASCII value} + \text{second ASCII value} \mod 19$$

Which results in less overlaps, especially if those coefficients get further apart!

We can also replace the substrings for each row in the characteristic matrix with the hash bucket numbers.

Then, if we sort by bucket number, we get much easier indexing!

	banana	bandit	brand
8	1	1	0
0	1	1	1
13	1	0	0
16	0	1	1
1	0	1	0
3	0	1	0
6	0	0	1
2	0	0	1

hashing

The sorted, renumbered characteristic matrix is to the left. The missing numbers correspond to possibly *many* missing shingles.

	banana	bandit	brand
0	1	1	1
1	0	1	0
2	0	0	1
3	0	1	0
6	0	0	1
8	1	1	0
13	1	0	~~10~~ 1
16	0	1	1

The data science question:

Is there a fast way to efficiently determine which pairs are most similar? Idea:

1. Replace each set/document with a smaller representation called a *signature.*
2. Use the signatures to *estimate* Jaccard similarity.

minhashing

So how do we find "signatures?"

	banana	bandit	brand
0	1	1	1
1	0	1	0
2	0	0	1
3	0	1	0
6	0	0	1
8	1	1	0
13	1	0	10
16	0	1	1

Definition: To *minhash* a set represented by a column of the characteristic matrix, pick a permutation of the rows. The *minhash value* of any column is the number of the first row, in the permuted order, in which the column has a 1.

Definition: A row in the *signature matrix* is built up from the minhash values of all columns under a given permutation

Permutations and Jaccard

Why does this make any sense?

	banana	bandit	brand
0	1	1	1
1	0	1	0
2	0	0	1
3	0	1	0
6	0	0	1
8	1	1	0
13	1	0	10
16	0	1	1

2/10

4/10

2/8

Idea: if we pick a row *at random* and look at 2 columns, what happens?

1. Type "X" is both 1s. This should be in the numerator *and* denominator of Jaccard similarity.
2. Type "Y" is exactly 1 1. This should be in the denominator of Jaccard similarity.
3. Type "Z" is both 0s. This should not be included in Jaccard similarity at all.

If counted these types over *all* rows, that's sim $= \frac{X}{X+Y}$.
We're going to use permutations to *randomly* select a row, and only count the first row that's either X-type or Y-type.

hashing Example

Hi!

8 1 1 0
6 0 0 1
2 0 0 1

Let's generate some permutations.

	banana	bandit	brand
0	1	1	1
1	0	1	0
2	0	0	1
3	0	1	0
6	0	0	1
8	1	1	0
13	1	0	10
16	0	1	1

1. Consider the permutation of rows: 8, 6, 2, 16, 0, 13, 1, 3
2. The resulting minhash values are:
   2.1 banana: 1 (b/c first 1 is in row 1)
   2.2 bandit: 1
   2.3 brand: 2 (b/c first 1 is in row 2)
3. *Save* those values into the new, *signature matrix*:

$$\begin{bmatrix} banana & bandit & brand \\ 1 & 1 & 2 \end{bmatrix}$$

Hi!

6 0 0 1
16 0 1 1
3 0 1 0
1 0 1 0
8 1 1 0

hashing Example

Let's generate some permutations.

	banana	bandit	brand
0	1	1	1
1	0	1	0
2	0	0	1
3	0	1	0
6	0	0	1
8	1	1	0
13	1	0	10
16	0	1	1

1. Next, consider the permutation of rows: 6, 16, 3, 1, 8, 0, 13, 2
2. The resulting minhash values are:
   - 2.1 banana: 5
   - 2.2 bandit: 2
   - 2.3 brand: 1
3. Finally let's also add the original permutation of rows: 3, 6, 16, 1, 8, 0, 13, 2, and the resulting minhash values... banana: 5, bandit: 1, brand: 2.
4. Append those values into the *signature matrix*:

$$\begin{array}{ccc} banana & bandit & brand \\ \end{array}$$
$$\begin{bmatrix} 1 & 1 & 2 \\ 5 & 2 & 1 \\ 5 & 1 & 2 \end{bmatrix}$$

hashing Example

are they identical =1
" not " =0

The Jaccard similarity of the document is approximated by the Jaccard similarity of the rows of the *signature matrix*

	banana	bandit	brand
0	1	1	1
1	0	1	0
2	0	0	1
3	0	1	0
6	0	0	1
8	1	1	0
13	1	0	10
16	0	1	1

10 million rows

$$\begin{bmatrix} banana & bandit & brand \\ 1 & 1 & 2 \\ 5 & 2 & 1 \\ 5 & 1 & 2 \end{bmatrix}$$

1000 rows?

87.2% similar

So we have:

1. sim(banana, bandit)$\approx 1/3$
2. sim(banana, brand)$\approx 0/3$
3. sim(bandit, brand)$\approx 0/3$

which is uh... not great compared to the true values of (2/6, 1/6, 2/7). But in reality we'd do this many many times!

Permuting

So we're at this point:

1. Collapse documents into smaller matrices using shingles, and possible hashing to reduce the total number of rows.
2. Permuting the rows around allows us to save on *memory*. Instead of loading all the documents at once, we can create their *signatures*, or the first first row with a "1" under a given permutation. We then apply the *same permutation* to other documents and compare *signatures*

But did this actually save memory? Now we have to save permutations of huge matrices to use the same permutations on new documents!

Worse... aren't truly random permutations pretty computationally expensive??

Hi!

$1000x + 14 \mod 7$

x grows

Not Permuting

It turns out we don't *actually have to permute*. Instead, another hash function allows us to **approximate** permutations. If we use a hash function we can randomly grab rows in a way that *looks* like a random permutation.

This is called a **universal hash**. For *random* integers a and b and a large prime number p (much greater than the total number of rows N,

$$h_{a,b}(x) = ((a \cdot x + b) \mod p) \mod N$$

is a function that in practice *scans the rows* of the **characteristic matrix** in an approximately random order.

The full minhash:

$10x+7 \mod 19$
$5x+2 \mod 19$
$3x+8 \mod 19$

1. Step 1: pick n hash functions.
2. Step 2: initialize the *signature matrix* with all infinities. We'll replace the terms once we find "first 1's".
3. Step 3: For each row r of the characteristic matrix, compute $h_i(r)$ for that row for each and every one of the i hash functions. This represents "pick a random row" for n different permutations.

next time!

4. For each document or column x:
   4.1 If the char. matrix has a 0 in column c, row r: do nothing.
   4.2 If the char. matrix has a 1 in column c, row r, then take each of the hash functions and replace $sig(i, c)$ by $\min(sig(i, c), h_i(r))$

Why this last bit? We're looking for the *first row* of a 1, which will be the minimum time that it occurred.

Acknowledgments

We'll pick up with an example of this next time.. and start discussing clustering!

Some material is adapted/adopted from Mining of Massive Data Sets, by Jure Leskovec, Anand Rajaraman, Jeff Ullman (Stanford University) `http://www.mmds.org`

Special thanks to Tony Wong for sharing his original adaptation and adoption of slide material.

Minhash example slides follow: expect to do next time in class.

The full minhash:

1. Step 1: pick n hash functions.
2. Step 2: initialize the *signature matrix* with all infinities. We'll replace the terms once we find "first 1's".
3. Step 3: For each row r of the characteristic matrix, compute $h_i(r)$ for that row for each and every one of the i hash functions. This represents "pick a random row" for n different permutations.
4. Step 4: For each document or column x:
   4.1 If the char. matrix has a 0 in column c, row r: do nothing.
   4.2 If the char. matrix has a 1 in column c, row r, then take each of the hash functions and replace $sig(i, c)$ by $\min(sig(i, c), h_i(r))$

Why this last bit? We're looking for the *first row* of a 1, which will be the minimum time that it occurred.

The full minhash: Example

So here's our data, with row labels now indices:

	banana	bandit	brand
0	1	1	1
1	0	1	0
2	0	0	1
3	0	1	0
4	0	0	1
5	1	1	0
6	1	0	0
7	0	1	1

Step 1:
Suppose we choose a few hash functions. We use a prime of 11 since it's more than the buckets we have (8). Our functions:

$h_1(r) = (r+3) \mod 11$

$h_2(r) = (2r+5) \mod 11$

$h_3(r) = (3r+7) \mod 11$

Step 2: Initialize: where was the first 1?
We set *sig*=

$$\begin{array}{c c} & \begin{array}{ccc} banana & bandit & brand \end{array} \\ \begin{array}{c} h_1 \\ h_2 \\ h_3 \end{array} & \begin{bmatrix} \infty & \infty & \infty \\ \infty & \infty & \infty \\ \infty & \infty & \infty \end{bmatrix} \end{array}$$

The full minhash: Example

	banana	bandit	brand
0	1	1	1
1	0	1	0
2	0	0	1
3	0	1	0
4	0	0	1
5	1	1	0
6	1	0	0
7	0	1	1

$h_1(r) = (r+3) \mod 11$

$h_2(r) = (2r+5) \mod 11$

$h_3(r) = (3r+7) \mod 11$

Step 3: For each row r of the characteristic matrix, compute $h_i(r)$ for that row for each and every one of the i hash functions. This represents "pick a random row" for n different permutations.

Step 4: For each column:

1. If the char. matrix has a 0 in column c, row r: do nothing.
2. If the char. matrix has a 1 in column c, row r, then take each of the hash functions and replace $sig(i,c)$ by $\min\left(sig(i,c), h_i(r)\right)$

Solution: in row 0, only *bandit* has a 1. It's eligible for replacement. **The minimum "location"** we've observed a non-zero for *bandit* is at the hashed value of "3".

$$\begin{array}{c c} & \begin{array}{ccc} banana & bandit & brand \end{array} \\ h_1 & \begin{bmatrix} \infty & \infty & \infty \end{bmatrix} \end{array}$$

minhash Example, row 0

	banana	bandit	brand
0	1	1	1
1	0	1	0
2	0	0	1
3	0	1	0
4	0	0	1
5	1	1	0
6	1	0	0
7	0	1	1

$h_1(r) = (r + 3) \mod 11$

$h_2(r) = (2r + 5) \mod 11$

$h_3(r) = (3r + 7) \mod 11$

Step 3: Compute $h_i(r)$ for that row.

Solution : $h_1(0) = 3,\ h_2(0) = 5,\ h_3(0) = 7$

Step 4: For each document or column x:

1. If the char. matrix has a 0 in column c, row r: do nothing.
2. If the char. matrix has a 1 in column c, row r, then take each of the hash functions and replace $sig(i, c)$ by $\min(sig(i, c), h_i(r))$

Solution: in row 1, all docs have a 1. For all 3 hash fns and all 3 docs, we have an estimate for the "first location of a 1."

$$\begin{array}{c} \\ h_1 \\ h_2 \\ h_3 \end{array} \begin{array}{c} \begin{array}{ccc} banana & bandit & brand \end{array} \\ \begin{bmatrix} 3 & 3 & 3 \\ 5 & 5 & 5 \\ 7 & 7 & 7 \end{bmatrix} \end{array}$$

minhash Example, row 1

	banana	bandit	brand
0	1	1	1
1	0	1	0
2	0	0	1
3	0	1	0
4	0	0	1
5	1	1	0
6	1	0	0
7	0	1	1

$h_1(r) = (r+3) \mod 11$

$h_2(r) = (2r+5) \mod 11$

$h_3(r) = (3r+7) \mod 11$

Step 3: Compute $h_i(r)$ for that row.

Solution : $h_1(1) = 4,\ h_2(1) = 7,\ h_3(1) = 10$

Step 4: For each document or column x:

1. If the char. matrix has a 0 in column c, row r: do nothing.
2. If the char. matrix has a 1, replace $sig(i,c)$ by $\min\left(sig(i,c), h_i(r)\right)$

Solution: in row 1, only *bandit* has a 1. It's eligible for replacement.

$$\begin{array}{c} \\ h_1 \\ h_2 \\ h_3 \end{array} \begin{array}{c} \begin{array}{ccc} banana & bandit & brand \end{array} \\ \begin{bmatrix} 3 & 3\, vs.\, 4 & 3 \\ 5 & 5\, vs.\, 7 & 5 \\ 7 & 7\, vs.\, 10 & 7 \end{bmatrix} \end{array}$$

minhash Example, row 1

	banana	bandit	brand
0	1	1	1
1	0	1	0
2	0	0	1
3	0	1	0
4	0	0	1
5	1	1	0
6	1	0	0
7	0	1	1

$$h_1(r) = (r+3) \mod 11$$
$$h_2(r) = (2r+5) \mod 11$$
$$h_3(r) = (3r+7) \mod 11$$

Step 3: Compute $h_i(r)$ for that row.

Solution : $h_1(1) = 4,\ h_2(1) = 7,\ h_3(1) = 10$

Step 4:
Solution: in row 1, only *bandit* has a 1. It's eligible for replacement, but its current values of 3,5,7 are smaller than the current evaluations of 4, 7, 10. **The minimum "location"** we've observed a non-zero for *bandit* is still at the hashed value of "3" for function 1.

$$\begin{array}{cc} & \begin{array}{ccc} banana & bandit & brand \end{array} \\ \begin{array}{c} h_1 \\ h_2 \\ h_3 \end{array} & \left[\begin{array}{ccc} 3 & 3 & 3 \\ 5 & 5 & 5 \\ 7 & 7 & 7 \end{array}\right] \end{array}$$

minhash Example, row 2

	banana	bandit	brand
0	1	1	1
1	0	1	0
2	0	0	1
3	0	1	0
4	0	0	1
5	1	1	0
6	1	0	0
7	0	1	1

$$h_1(r) = (r+3) \mod 11$$
$$h_2(r) = (2r+5) \mod 11$$
$$h_3(r) = (3r+7) \mod 11$$

Step 3: Compute $h_i(r)$ for that row.

Solution : $h_1(2) = 5,\ h_2(2) = 9,\ h_3(2) = 2$

Step 4: For each document or column x:

1. If the char. matrix has a 0 in column c, row r: do nothing.
2. If the char. matrix has a 1, replace $sig(i, c)$ by $\min(sig(i, c), h_i(r))$

Solution: only *brand* is eligible now.

$$\begin{array}{cc} & \begin{array}{ccc} banana & bandit & brand \end{array} \\ \begin{array}{c} h_1 \\ h_2 \\ h_3 \end{array} & \begin{bmatrix} 3 & 3 & 3\, {\color{blue} vs.\, 5} \\ 5 & 5 & 5\, {\color{blue} vs.\, 9} \\ 7 & 7 & 7\, {\color{red} vs.\, 2} \end{bmatrix} \end{array}$$

minhash Example, row 3

	banana	bandit	brand
0	1	1	1
1	0	1	0
2	0	0	1
3	0	1	0
4	0	0	1
5	1	1	0
6	1	0	0
7	0	1	1

$h_1(r) = (r+3) \mod 11$

$h_2(r) = (2r+5) \mod 11$

$h_3(r) = (3r+7) \mod 11$

Step 3: Compute $h_i(r)$ for that row.

Solution : $h_1(3) = 6,\ h_2(3) = 0,\ h_3(3) = 5$

Step 4: For each document or column x:

1. If the char. matrix has a 0 in column c, row r: do nothing.
2. If the char. matrix has a 1, replace $sig(i,c)$ by $\min(sig(i,c), h_i(r))$

Solution: only *bandit* is eligible now.

$$\begin{array}{c} \\ h_1 \\ h_2 \\ h_3 \end{array}\begin{array}{c} \begin{array}{ccc} banana & bandit & brand \end{array} \\ \left[\begin{array}{ccc} 3 & 3 & 3 \\ 5 & \color{red}{0} & 5 \\ 7 & \color{red}{5} & 2 \end{array}\right] \end{array}$$

minhash Example, row 4

	banana	bandit	brand
0	1	1	1
1	0	1	0
2	0	0	1
3	0	1	0
4	0	0	1
5	1	1	0
6	1	0	0
7	0	1	1

$$h_1(r) = (r+3) \mod 11$$
$$h_2(r) = (2r+5) \mod 11$$
$$h_3(r) = (3r+7) \mod 11$$

Step 3: Compute $h_i(r)$ for that row.

Solution : $h_1(4) = 7,\ h_2(4) = 2,\ h_3(4) = 8$

Step 4: For each document or column x:

1. If the char. matrix has a 0 in column c, row r: do nothing.
2. If the char. matrix has a 1, replace $sig(i, c)$ by $\min(sig(i,c), h_i(r))$

Solution: only *brand* is eligible now.

$$\begin{array}{cc} & \begin{array}{ccc} banana & bandit & brand \end{array} \\ \begin{array}{c} h_1 \\ h_2 \\ h_3 \end{array} & \left[\begin{array}{ccc} 3 & 3 & 3 \\ 5 & 0 & \color{red}{2} \\ 7 & 5 & 2 \end{array}\right] \end{array}$$

minhash Example, row 5

	banana	bandit	brand
0	1	1	1
1	0	1	0
2	0	0	1
3	0	1	0
4	0	0	1
5	1	1	0
6	1	0	0
7	0	1	1

$$h_1(r) = (r+3) \mod 11$$

$$h_2(r) = (2r+5) \mod 11$$

$$h_3(r) = (3r+7) \mod 11$$

Step 3: Compute $h_i(r)$ for that row.

Solution : $h_1(5) = 8,\ h_2(5) = 4,\ h_3(5) = 0$

Step 4: For each document or column x:

1. If the char. matrix has a 0 in column c, row r: do nothing.
2. If the char. matrix has a 1, replace $sig(i, c)$ by $\min(sig(i, c), h_i(r))$

Solution: banana and bandit are eligible now.

$$\begin{array}{c} \\ h_1 \\ h_2 \\ h_3 \end{array} \begin{array}{c} \begin{array}{ccc} banana & bandit & brand \end{array} \\ \begin{bmatrix} 3 & 3 & 3 \\ 4 & 0 & 2 \\ 0 & 0 & 2 \end{bmatrix} \end{array}$$

minhash Example, row 6

	banana	bandit	brand
0	1	1	1
1	0	1	0
2	0	0	1
3	0	1	0
4	0	0	1
5	1	1	0
6	1	0	0
7	0	1	1

$$h_1(r) = (r+3) \mod 11$$

$$h_2(r) = (2r+5) \mod 11$$

$$h_3(r) = (3r+7) \mod 11$$

Step 3: Compute $h_i(r)$ for that row.

Solution : $h_1(6) = 9,\ h_2(6) = 6,\ h_3(6) = 3$

Step 4: For each document or column x:

1. If the char. matrix has a 0 in column c, row r: do nothing.
2. If the char. matrix has a 1, replace $sig(i,c)$ by $\min\left(sig(i,c), h_i(r)\right)$

Solution: only *banana* is eligible now.

$$\begin{array}{cc} & \begin{array}{ccc} banana & bandit & brand \end{array} \\ \begin{array}{c} h_1 \\ h_2 \\ h_3 \end{array} & \begin{bmatrix} 3 & 3 & 3 \\ 4 & 0 & 2 \\ 0 & 0 & 2 \end{bmatrix} \end{array}$$

minhash Example, row 7

	banana	bandit	brand
0	1	1	1
1	0	1	0
2	0	0	1
3	0	1	0
4	0	0	1
5	1	1	0
6	1	0	0
7	0	1	1

$$h_1(r) = (r+3) \mod 11$$

$$h_2(r) = (2r+5) \mod 11$$

$$h_3(r) = (3r+7) \mod 11$$

Step 3: Compute $h_i(r)$ for that row.

Solution : $h_1(7) = 10,\ h_2(7) = 8,\ h_3(7) = 6$

Step 4: For each document or column x:

1. If the char. matrix has a 0 in column c, row r: do nothing.
2. If the char. matrix has a 1, replace $sig(i,c)$ by $\min(sig(i,c), h_i(r))$

Solution: bandit and brand are eligible.

$$\begin{array}{cc} & \begin{array}{ccc} banana & bandit & brand \end{array} \\ \begin{array}{c} h_1 \\ h_2 \\ h_3 \end{array} & \begin{bmatrix} 3 & 3 & 3 \\ 4 & 0 & 2 \\ 0 & 0 & 2 \end{bmatrix} \end{array}$$

minhash wrapup

Here's our final signature matrix:

$$
\begin{array}{c c}
 & \begin{array}{ccc} banana & bandit & brand \end{array} \\
\begin{array}{c} h_1 \\ h_2 \\ h_3 \end{array} & \left[\begin{array}{ccc} 3 & 3 & 3 \\ 4 & 0 & 2 \\ 0 & 0 & 2 \end{array} \right]
\end{array}
$$

which leads to similarities of $2/3$ between *banana* and *bandit* and $1/3$ for either other word with *brand*. But to be clear: we would do this *at least* hundreds of times more!

This would consolidate each document - which may have hundreds of thousands of rows in the characteristic matrix - into just the hundreds of rows in the signature matrix, matching our number of hash functions.

minhash wrapup

There are a lot of approximations, here

1. Shingles simplify language
2. Hashing buckets simplify/collapse shingles
3. Random row comparisons approximate Jaccard similarity
4. Permutations approximate random rows
5. Hash functions approximate permutations

... all to solve one major problem: not loading the characteristic matrices for multiple documents into memory at once.

It turns out there are some more things that can approximate/streamline the process for huge parallel data sets. See the text for how LSH (locally sensitive hashing) uses **bands** on the signature matrix to compare documents rather than computing their “full” Jaccard similarities.

minhash wrapup

There are a lot of approximations, here

1. Shingles simplify language
2. Hashing buckets simplify/collapse shingles
3. Random row comparisons approximate Jaccard similarity
4. Permutations approximate random rows
5. Hash functions approxomate permutations

... all to solve one major problem: not loading the characteristic matrices for multiple documents into memory at once.

It turns out there are some more things that can approximate/streamline the process for huge parallel data sets. See the text for how LSH (locally sensitive hashing) uses **bands** on the signature matrix to compare documents rather than computing their "full" Jaccard similarities.