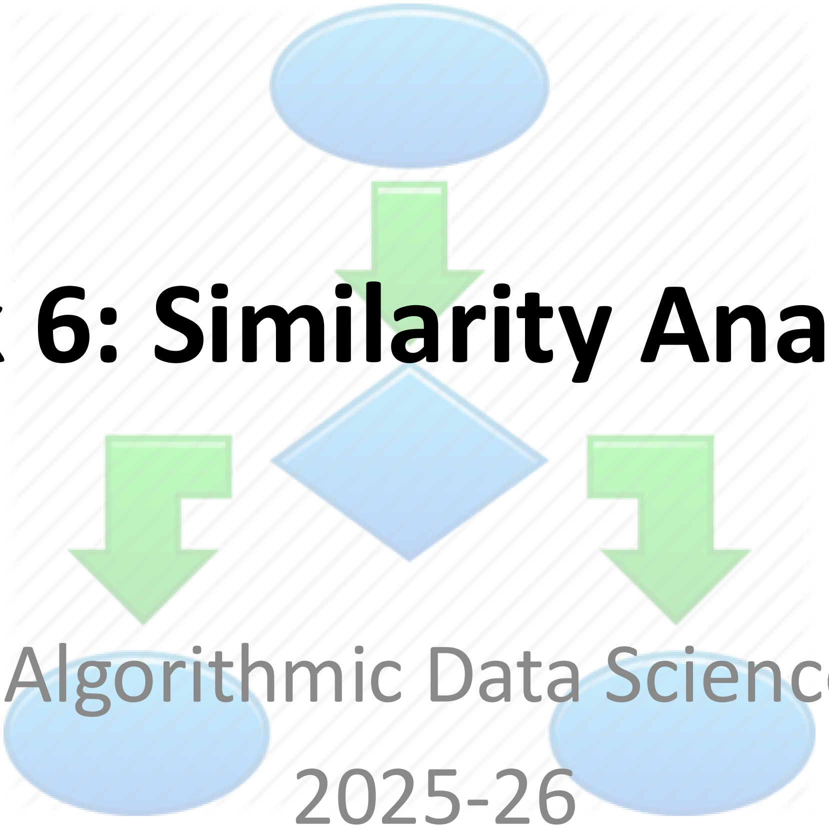


Week 6: Similarity Analysis

Algorithmic Data Science

2025-26

UNIVERSITY OF SUSSEX

Dr Adam Barrett

Warm-up

- Computers store numbers (well everything) in binary
- In decimal, 45 means $4 \times 10^1 + 5 \times 10^0$
- In binary, 1011 means $1 \times 2^3 + 0 \times 2^2 + 1 \times 2^1 + 1 \times 2^0$
- Complete the table below with binary / decimal equivalences

Binary	Decimal
1011	11
101	
1001101	
	32
	100

A byte is 8 bits (where a bit is a 0 or 1). What's the largest number which can be stored in 1 byte? What's the largest number which can be stored in 4 bytes?

Warm-up

- Computers store numbers (well everything) in binary
- In decimal, 45 means $4 \times 10^1 + 5 \times 10^0$
- In binary, 1011 means $1 \times 2^3 + 0 \times 2^2 + 1 \times 2^1 + 1 \times 2^0$
- Complete the table below with binary / decimal equivalences

Binary	Decimal
1011	11
101	5
1001101	1+4+8+64=77
100000	32
1100100	100

A byte is 8 bits (where a bit is a 0 or 1). What's the largest number which can be stored in 1 byte? What's the largest number which can be stored in 4 bytes?

Main topics per week

Week	Topic
1	Data structures and data formats
2	Algorithmic complexity. Sorting.
3	Matrices: Manipulation and computation
4	Processes and concurrency
5	Distributed computation
6	**Similarity**
7	Map/reduce
8	Graphs/networks
9	Graphs/networks, PageRank algorithm
10	Databases
11	*independent study*

Overview

- applications of similarity / near-neighbour search
- similarity measures
- string similarity
- shingling
- Minhashing
- Locality sensitive hashing (LSH)

Applications of similarity

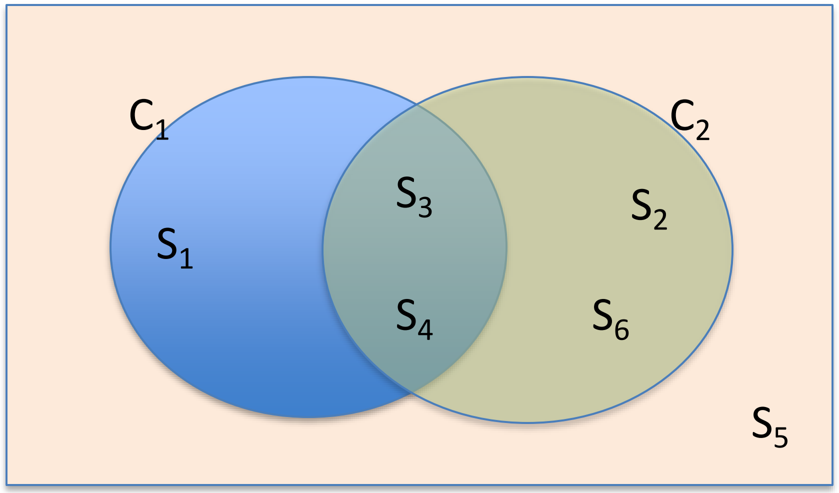
- similarity of documents
  - plagiarism
  - mirror pages
  - articles from the same source
- collaborative filtering
  - online purchases
  - movie ratings
- clustering
  - grouping objects in such a way that objects in the same group (called a **cluster**) are **more similar** to each other than to those in other clusters.

Example

- The ‘objects’ in which we are interested are customers
- We want to consider two customers similar if they have purchased similar items.
- If we have each customer’s purchase history, how do we represent each customer?

Set-theoretic notions of similarity

- Boolean features (a customer C_i either has or hasn't purchased some item S_j) lead naturally to set–theoretic notions of similarity.



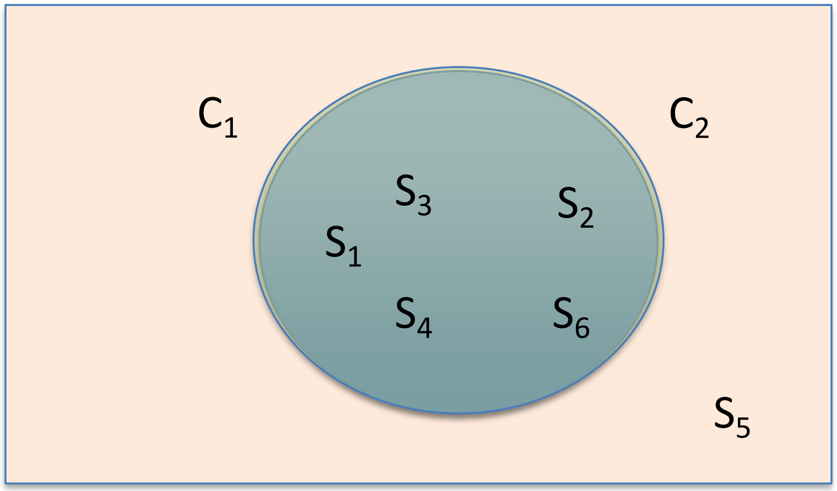
$C_1=\{S_1, S_3, S_4\}$

$C_2=\{S_2, S_3, S_4, S_6\}$

Jaccard's measure is the ratio of the cardinality (size) of the intersection of two sets to the cardinality of the union of two sets

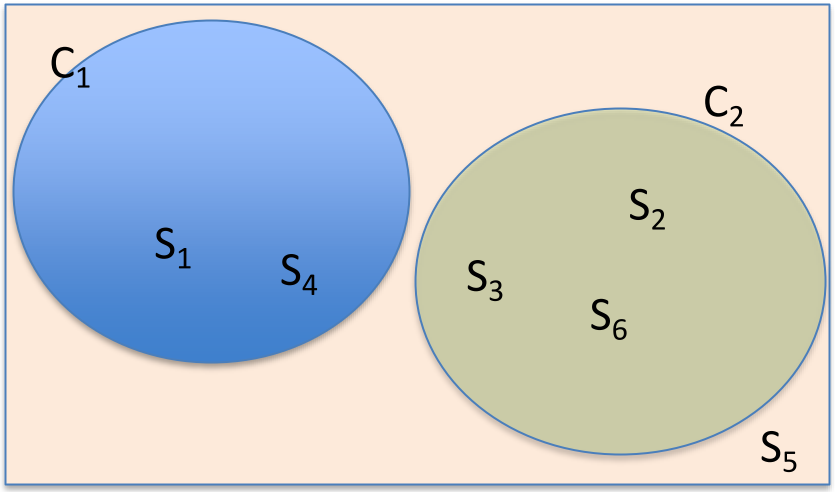
$$Jacc(C_1, C_2) = \frac{|C_1 \cap C_2|}{|C_1 \cup C_2|} = \frac{|C_1 \cap C_2|}{|C_1| + |C_2| - |C_1 \cap C_2|} = \frac{2}{5}$$

Special cases



$C_1=C_2=\{S_1, S_2, S_3, S_4, S_6\}$

$\mathrm{Jacc}(C_1,C_2) = 5/5 = 1$



$C_1=\{S_1, S_4\}$

$C_2=\{S_2, S_3, S_6\}$

$\mathrm{Jacc}(C_1,C_2) = 0/5 = 0$

Algorithm for Jaccard’s Measure

- What does the run-time for computing Jaccard similarity depend on?
- What are the possible worst-case performances in O notation?

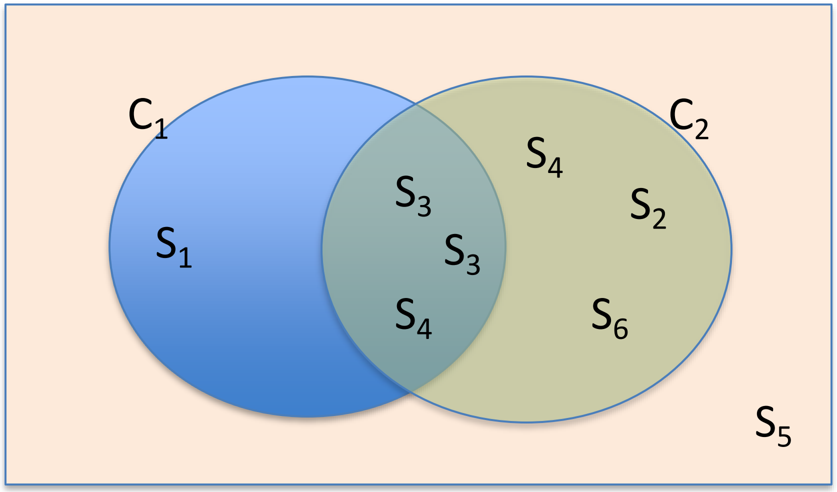
Algorithm for Jaccard's measure

Data stored in Python lists:

```
def jaccard(C1, C2):
        int=0
        union=0
        for item in C1:
                if item in C2:
            int+=1
        union=len(C1)+len(C2)-int
        return int/union
```

Assuming C1 and C2 have length O(n), then this is O(n^2) because the if statement takes O(n) to execute (check every element).

Extending to Bags



Can model duplicate items or even real-valued scores using bags.

The shared (minimum) part of the score goes in the intersection. All of it goes in the union

$C_1=\{S_1, S_3, S_3, S_4\}$
$C_2=\{S_2, S_3, S_3, S_4, S_4, S_6\}$

$$\begin{array}{c|cc} & C_1 & C_2 \\ S_1 & 1 & 0 \\ S_2 & 0 & 1 \\ S_3 & 2 & 2 \\ S_4 & 1 & 2 \\ S_5 & 0 & 0 \\ S_6 & 0 & 1 \end{array}$$

characteristic matrix

$$Jacc(C_1,C_2)=\frac{|C_1\cap C_2|}{|C_1|+|C_2|-|C_1\cap C_2|}=\frac{\sum_i \min(S_{i1},S_{i2})}{\sum_i S_{i1}+S_{i2}-\min(S_{i1},S_{i2})}=\frac{3}{7}$$

Algorithm for Jaccard's measure

Data stored in a dictionary, bags version of Jaccard:

```
def maketotal(dict1):
    total=0
    for item in dict1:
        total += dict1[item]
    return total

def jaccard(dict1,dict2):
    intersection={}
    for item in dict1.keys():
        if item in dict2.keys():
            intersection[item]=min(dict1[item],dict2[item])

    intersectiontot=maketotal(intersection)
    union = maketotal(dict1)+maketotal(dict2)-intersectiontot
    return intersectiontot/union
```

Assuming dict1 and dict2 have O(n) item, then **if we have no hash collisions:**
the if statement takes O(1) to execute- just look at what is stored at the hash of the item.

Note

- Take care when using Jaccard similarity, to be clear whether you are using the measure applied to **sets** or to **bags**. In general, this choice affects the value you get.

Exercise

- Consider the following 2 sets of items S1 and S2.

S1={A1, A2, A5, A6}
S2={A2, A3, A5}

What is the Jaccard similarity of sets S1 and S2?

(a) 3/4
(b) 2/5
(c) 2/7
(d) 1/3

Exercise (Solution)

- Consider the following 2 sets of items S1 and S2.

S1={A1, A2, A5, A6}
S2={A2, A3, A5}

Compute the Jaccard similarity of sets S1 and S2.

Intersection = {A2, A5}   size of intersection = 2
Union = {A1,A2,A3,A5,A6} size of union = 5
Jaccard similarity = |I| / |U| = 2/5 =0.4

Compute the cosine similarity of sets S1 and S2.

	S1	S2
A1	1	0
A2	1	1
A3	0	1
A5	1	1
A6	1	0

S1.S2 = 0 + 1 + 0+1+0 =2
S1.S1=4
S2.S2=3

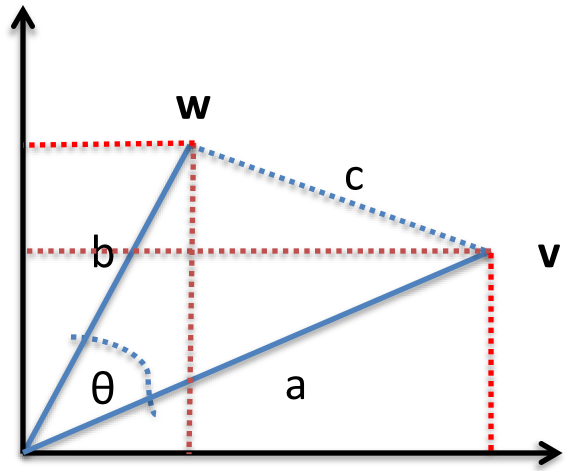
$$Cos = \frac{S1.S2}{\sqrt{(S1.S1)(\ S2.S2\ )}} = \frac{2}{\sqrt{12}} = \frac{2}{2\sqrt{3}} = \frac{1}{\sqrt{3}} \approx 0.58$$

Cosine similarity

This makes use of the **dot product** for vectors:

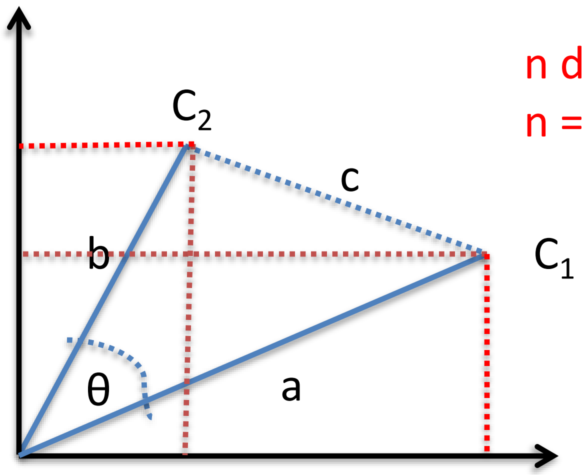
$$\boldsymbol{v} \cdot \boldsymbol{w} = \sum_{\boldsymbol{i}=1}^{\boldsymbol{n}} \boldsymbol{v}_{\boldsymbol{i}} \boldsymbol{w}_{\boldsymbol{i}}$$

$$\cos(\theta) = \frac{\boldsymbol{v} \cdot \boldsymbol{w}}{\sqrt{(\boldsymbol{v} \cdot \boldsymbol{v})(\boldsymbol{w}.\boldsymbol{w})}}$$



Cosine similarity

- Real valued 'vector' representations of objects lead naturally to geometric notions of similarity



n dimensions,
n = 6 here

	C_1	C_2	$C_1.C_2$
S_1	1	0	0
S_2	0	1	0
S_3	2	2	4
S_4	3	1	3
S_5	1	1.5	1.5
S_6	0	1	0
			Σ=8.5

$$\cos(C_1, C_2) = \frac{C_1.C_2}{\sqrt{C_1.C_1 \square C_2.C_2}}$$

$C_1.C_2$ is the dot product. Also known as the inner product $<C_1,C_2>$ or the scalar product

Algorithm for Cosine Measure

```
def naiveCosine (a , b):
    num=0
    d1=0
    d2=0
    for i in range len( a ) :
        num += a [ i ] *b [ i ]
        d1 += a [ i ] *a [ i ]
        d2 += b [ i ] *b [ i ]
    return num / ( d1*d2 ) **0.5
```

- What does the running time of this algorithm depend on?
- Give an estimate of its worst-case performance in O notation

Finding similarity of two strings

- Strings can be modelled as bags-of-characters
- Long strings (documents!) can be modelled as bags-of-words

"colour" -> {c:1, o:2, l:1, u:1, r:1}
"color" -> {c:1, o:2, l:1, r:1}

a dictionary is a compact sparse representation for a bag but it is equivalent to the dense matrix representation

Exercise (at home): Show that Jacc("colour", "color") = 5/6

Question:
cos("colour", "color") = ?

$(a)\ \sqrt{\frac{8}{7}}$ $\quad (b)\ \sqrt{\frac{7}{8}}$

$(c)\ \frac{8}{7}$ $\quad (d)\ \frac{7}{8}$

	S1	S2
C	1	1
O	2	2
L	1	1
U	1	0
R	1	1

S1.S2 1+4+1+1=7

S1.S1=1+4+1+1+1=8

S2.S2=1+4+1+1=7

$$Cos = \frac{S1.S2}{\sqrt{(S1.S1)(\ S2.S2\ )}} = \frac{7}{\sqrt{7 * 8}} = \sqrt{\frac{7}{8}}$$

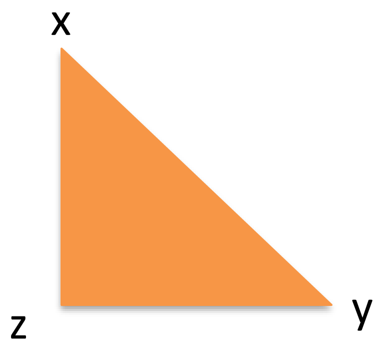
SUPPLEMENTARY

Similarity vs Distance

- Distance measures measure dissimilarity

distance measures	similarity measures
$d(x,y) \geq 0$	$0 \leq sim(x,y) \leq 1$
$d(x,y) = 0$ iff $x=y$	$sim(x,y)=1$ iff $x=y$
$d(x,y) = d(y,x)$	$sim(x,y) = sim(y,x)$
$d(x,y) \leq d(x,z) + d(z,y)$	

triangle inequality



Supplementary material

Correlation vs Cosine Similarity

To compute correlation of 2 variables X and Y

- Subtract the mean of X from each value X_i and the mean of Y from each value Y_i
- Compute the dot-product of the transformed X and Y. This is the **covariance** of X and Y
- Divide by the square root of the product of cov(X,X) and cov(Y,Y)

To compute cosine similarity between 2 vectors X and Y

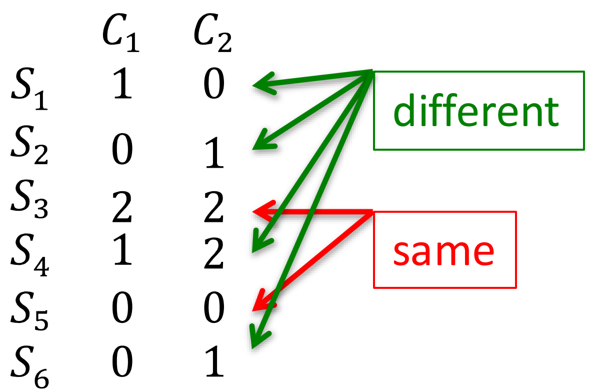
- Compute the dot product of X and Y : <X,Y>
- Divide by the square root of the product of <X,X> and <Y,Y>

The only difference is that when computing correlation, we compute covariance rather than a simple dot product i.e., we standardize by subtracting the means first.

Supplementary material

Other measures: Hamming distance

- The number of vector components (dimensions) in which two objects differ.
- Usually only applied to Boolean vectors (e.g., sets) but can be applied to bags

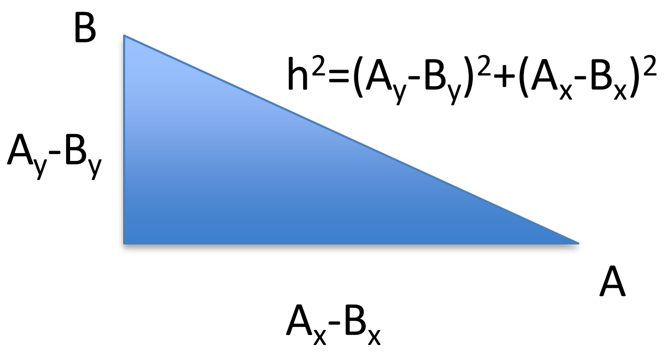


	C_1	C_2
S_1	1	0
S_2	0	1
S_3	2	2
S_4	1	2
S_5	0	0
S_6	0	1

Hamming distance(C_1,C_2) = 4

L Norms

Most people are familiar with the L_2 Norm (also known as the **Euclidean distance**), which is Pythagoras theorem in n-dimensions:



$$L_2(A,B) = \sqrt{\sum_{i=1}^{n}(A_i - B_i)^2}$$

In general, the L_k Norm is given by

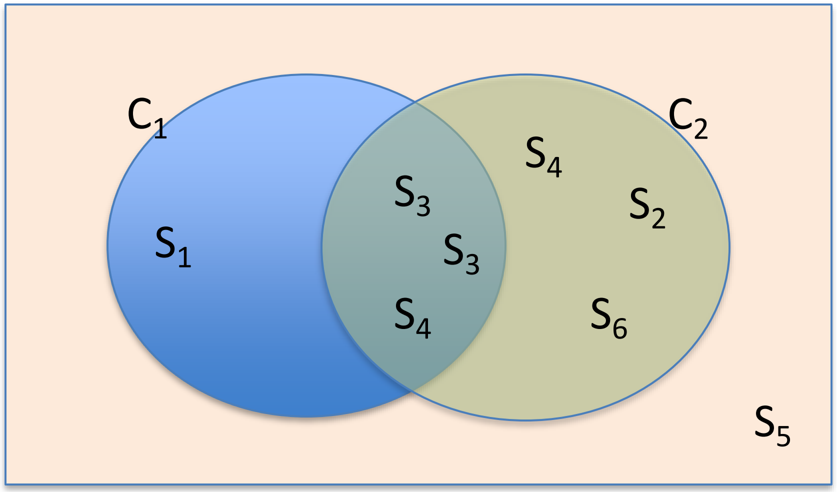
$$L_k(A,B) = \sqrt[k]{\sum_{i=1}^{n}|A_i - B_i|^k}$$

The L_1 Norm (or Manhattan or City Block) distance is

$$L_1(A,B) = \sum_{i=1}^{n}|A_i - B_i|$$

Probabilistic measures of similarity

- Frequencies can be easily converted into probabilities



$$\begin{array}{ccc} & C_1 & C_2 \\ S_1 & 1 & 0 \\ S_2 & 0 & 1 \\ S_3 & 2 & 2 \\ S_4 & 1 & 2 \\ S_5 & 0 & 0 \\ S_6 & 0 & 1 \end{array} \rightarrow \begin{array}{ccc} & C_1 & C_2 \\ S_1 & 0.25 & 0 \\ S_2 & 0 & 0.166 \\ S_3 & 0.5 & 0.333 \\ S_4 & 0.25 & 0.333 \\ S_5 & 0 & 0 \\ S_6 & 0 & 0.166 \end{array}$$

What is the probability that a randomly chosen item is S_i given it is in the set / bag C_j?

Supplementary material

Probabilistic measures of similarity

Most well-known ‘distance’ measure for probability distributions is the Kullback-Leibler divergence measure

$$D_{KL}(C_1 \| C_2) = \sum_i p_{i1} \cdot \log \frac{p_{i1}}{p_{i2}}$$

What is the average penalty (i.e., difference in log probabilities) if you use the distribution for C_1 in place of the distribution for C_2?

This is not strictly a distance measure because it is not symmetric. The Jenson-Shannon divergence measure is the symmetric version, which measures distance as the average Kullback-Leibler divergence to the centroid of the distributions.

$$\boxed{JS(A,B) = ½ (KL(A,M) + KL(B,M))}$$

where $M = \frac{1}{2}(A + B)$

Disadvantages of Using Bags for Text

- Not sensitive to order
  - "brag" = "grab"
- If applied to documents where the atomic units are words ....
  - does not capture relationships between different words

The old man chased the small dog that bit a naughty child.

?
=

The old dog chased the naughty small child that bit a man.

Shingling

The old man chased the small dog that bit a naughty child.

?=

The old dog chased the naughty small child that bit a man.

- A bag-of-words (or bag of characters) representation will lead to these strings being considered identical.
- We could use a bag-of-*ngrams* :
  - unigram = 1 word, bigram = 2 words, trigram = 3 words, ngram = n words
  - ***Exercise:*** *Show that the Jaccard similarity if we used a bag of bigrams is 2/9.*
- Alternative is to use a set or bag of shingles. A *k*-shingle for a document is any string of length *k* found in the document

Shingling

Example: What are the sets of 3-shingles for the strings “john loves mary” and “mary loves john”?

If a string has m characters, it will have at most m-k distinct shingles

A: john loves mary			B: mary loves john		
joh	ohn	hn_	mar	ary	ry_
n_l	_lo	lov	y_l	_lo	lov
ove	ves	es_	ove	ves	es_
s_m	_ma	mar	s_j	_jo	joh
ary			ohn		

Jacc(A,B) = 9/(13+13-9) = 9/17

Supplementary material

Edit distance

- The edit distance between strings X and Y is the smallest number of operations required to transform X into Y, where the operations allowed are insertion and deletion (and also sometimes transposition and mutation).
- There are variants where the different operations have different costs but lets assume cost of each operation = 1

X	**Y**		**d(X,1)**
colour	color	delete c_5	1
doggy	daddy	delete c_2, delete c_3, delete c_4, insert 'a' at c_2, insert 'd' at c_3, insert 'd' at c_4	6
brag	grab	??	4
house	home	??	?

Similarity for large collection of documents

documents

	C1	C2	C3	C4
S1	1	0	0	1
S2	0	0	1	0
S3	0	1	0	1
S4	1	0	1	1
S5	0	0	1	0

. . .   .

shingles

. . .
. . .
. . .

- We're going to see an efficient way of analysing similarity for all pairs of documents in a collection.
- It will use the **set version of Jaccard similarity**.

  - So want to choose a shingle length where most shingles don't occur in most documents, so very different documents have very different shingle sets.

Choosing the shingle size

- *k* should be picked large enough that the probability of any given shingle appearing in any given document is low
- depends on the length of the typical document and the size of the character set

Example: if our corpus of documents is emails then k=5 is probably appropriate. Why?

- Assume number of characters is 27.
- Number of shingles = 27^5 = 14, 348, 907
- Typical email length << 14, 348 907 characters ☺
- In practice, there are more than 27 characters but many are very rare which increases probability of shingles of more common letters
- So actually better to assume number of characters for English ≈ 20

Shingles vs Bags-of-words

Representation	parameters	dimensionality

shingles	characters = 27
	k = 5
		$27^5 \approx 1.4 \times 10^7$

shingles	characters = 20
	k = 9
		$20^9 = 5.12 \times 10^{11}$

bag-of-words	vocabulary = 500K	5×10^5

bag-of-ngrams	vocabulary = 500K
	n = 2
		$500{,}000^2 = 2.5 \times 10^{11}$

- Shingles are fixed length whereas words are variable in length

Hashing shingles

- The ASCII character set has 128 characters
- If we use 1 byte per character, a 9-shingle will take 9 bytes
- And many of the possible shingles will never occur
- Use a hash function which maps 9-shingles to numbers in range 0 -> $2^{32} – 1$
- Then likelihood of each hashed value occurring much more equal.
- And such a number can be stored in 4 bytes
- However, still have several times more shingles per document as individual characters – need compression if lots of documents to analyse.

Minhashing

- A technique for constructing small **signatures** from large sets whilst preserving estimates of similarity.

Algorithm for minhashing a set represented by a column of a characteristic matrix:

1. pick a permutation of the rows
2. The minhash value of any column is the first row in the permuted order in which the column has a 1.
3. Repeat m times to get a minhash signature of length m

	C1	C2	C3	C4
S1	1	0	0	1
S2	0	0	1	0
S3	0	1	0	1
S4	1	0	1	1
S5	0	0	1	0

permutation 1

	C1	C2	C3	C4
S2	0	0	1	0
S5	0	0	1	0
S3	0	1	0	1
S1	1	0	0	1
S4	1	0	1	1
MH	**4**	**3**	**1**	**3**

permutation 2

	C1	C2	C3	C4
S4	1	0	1	1
S3	0	1	0	1
S2	0	0	1	0
S1	1	0	0	1
S5	0	0	1	0
MH	**1**	**2**	**1**	**1**

How many permutations?

- In practice, m will be much less than the dimensionality of the matrix (and much much less than the number of possible permutations)
- How large should m be? Say m = 100

- By minhashing we have reduced the dimensionality of the characteristic matrix :
- Originally: e.g. 2^{32} Boolean values, each Boolean takes 1 bit so 2^{24} bytes (=17MB) per column
- In minhash signature: each integer < 2^{32} so can be stored in 4 bytes so 400 bytes per column

Computing Minhash Signatures

- Not feasible to permute a large matrix explicitly
  - would have to pick a random permutation of billions of rows
  - then sort all of those rows ...
- Simulate the effect of a random permutation using a hash function, *h(r)*
- Same number of buckets as rows
- Whilst there will be some collisions, we can maintain the fiction that our hash function *h* permutes row *r* to position *h(r)*
- So instead of m random permutations, randomly choose m hash functions on the rows

Computing Minhash signatures

1. LET $M_{r,c}$ be element of the characteristic matrix for the rth element for cth set.
2. Let $SIG_{i,c}$ be the element of the signature matrix for the *i*th hash function and column *c*.
3. Initialise $SIG_{i,c}$ to ∞ for all *i* and *c*
4. FOR each row *r*:

```
 FOR each hash function $h_i$:
  compute $h_i(r)$
  FOR each column $c$:
   IF c has a 0 in row $r$: do nothing
   ELSE: $SIG_{i,c} = MIN(SIG_{i,c}, h_i(r))$
```

Minhashing and Jaccard Similarity

The probability that the minhash function for a random permutation of rows produces the same value for two sets equals the Jaccard similarity of those sets.

M	C1	C2	C3	C4
S1	1	0	0	1
S2	0	0	1	0
S3	0	1	0	1
S4	1	0	1	1
S5	0	0	1	0

SIG	C1	C2	C3	C4
MH1	4	3	1	3
MH2	1	2	1	1

Why?

Minhashing and Jaccard

	C1	C2	C3	C4
S1	1	0	0	1
S2	0	0	1	0
S3	0	1	0	1
S4	1	0	1	1
S5	0	0	1	0

For any given pair of columns:

- Type X rows have a 1 in both
- Type Y rows have different values
- Type Z rows have a 0 in both

For sparse matrices, most rows for most pairings will be type Z

It is the ratio of type X to type Y rows that determine Jacc(C_i,C_j) and also the probability that h(C_i) = h(C_j)

$$Jacc(C_3, C_4) = \frac{X_{3,4}}{X_{3,4} + Y_{3,4}} = \frac{1}{5}$$

	C1	C2	C3	C4
S4	1	0	1	1
S3	0	1	0	1
S2	0	0	1	0
S1	1	0	0	1
S5	0	0	1	0
MH	**1**	**2**	**1**	**1**

In a random permutation, the probability that we meet a type X row before we meet a type Y row is also $X_{3,4}/(X_{3,4}+Y_{3,4})$

If we do meet a type X row before we meet a type Y row, then we get $MH(C_3) = MH(C_4)$

	C1	C2	C3	C4
S2	0	0	1	0
S5	0	0	1	0
S3	0	1	0	1
S1	1	0	0	1
S4	1	0	1	1
MH	**4**	**3**	**1**	**3**

However, if we meet a type Y row before we meet a type X row then we get $MH(C_3) \neq MH(C_4)$

Minhashing and Jaccard Similarity

SIG	C1	C2	C3	C4
MH1	4	3	1	3
MH2	1	2	1	1

So, if we carried out ALL random permutations, the proportion of matches in the minhash signatures for 2 objects would equal Jaccard similarity

For a random selection of permutations, proportion of matches will estimate the Jaccard similarity

estimate of Jaccard	C1	C2	C3	C4
C1	1	0	0.5	0.5
C2	0	1	0	0.5
C3	0.5	0	1	0.5
C4	0.5	0.5	0.5	1

Efficient similarity analysis: Summary

1. Construct a characteristic matrix:
   - columns are population members e.g. documents, customers
   - rows are items, e.g. hashed k-shingles, items for sale
2. Compute *m* minhash signatures.

Jaccard similarity of Ci and Cj is approximately the proportion of minhash signatures which agree in column i and column j

But if n is the size of the population (number of documents / customers etc), it is still $O(n^2)$ to do all pairs-similarity.

Locality-Sensitive Hashing (LSH)

- A technique for efficiently finding nearest neighbours without computing all-pairs similarities.
- General approach is to hash items several times in such a way that similar items are more likely to be hashed to the same bucket than dissimilar items
- Any pair hashed to the same bucket for any of the hashings is then considered a **candidate pair**
- Only compute similarities for candidate pairs.
- There will be false positives but these will be found whilst computing similarities
- There will be false negatives (candidate pairs completely missed) – need to minimise these as no way of recovering them.

LSH for a Minhash Signature

	C1	C2	C3	C4	C5
band 1	4	3	1	3	1
	1	2	1	1	1
	0	1	3	1	3
band 2					
band 3					
band 4					

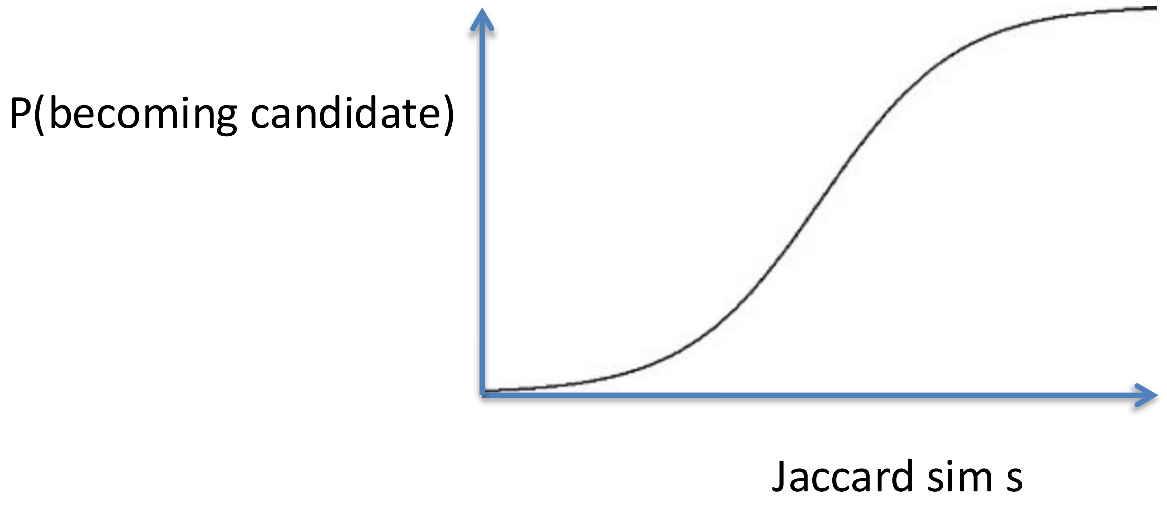
- Each hash function only considers a **band** of rows.
- Columns which are identical in the rows of a particular band must be hashed to the same bucket for that band
- There will be accidental collisions (leading to false positives)
- However, similar items will probably be identical in at least 1 band

Supplementary material

Analysis of Banding Technique

- Suppose we use b bands of r rows each and that a particular pair of documents have Jaccard similarity s
- P(sigs agree in all rows of a particular band) = s^r
- P(sigs do not agree in all rows of a particular band) = $1 – s^r$
- P(sigs do not agree in all rows of any band) = $(1 – s^r)^b$
- P(sigs agree in all rows of at least one band) = $1-(1-s^r)^b$

Analysis of Banding Technique



Curve given by $1-(1-s^r)^b$

for b bands of r rows each

- The threshold Jaccard similarity at which it becomes likely that the pair will become a candidate depends on b and r.
- The more rows per band, the higher this threshold is.
- An approximation to the threshold (where Prob=1/2) is

  threshold similarity = $(1/b)^{1/r}$

- If 100 rows are divided into 20 bands of 5, what will the threshold similarity be?
- What if we use 5 bands of 20?

Complexity of LSH

- Assuming that parameters have been chosen so that only 10% of pairs are considered to be candidate pairs by LSH, how much efficiency saving do you get for finding the k-nearest neighbours?

If n is now number of documents,
- O(n) to minhash.
- O(n) to do LSH (apply hash to each of n documents, and then look at what is in each bucket).
- So, still $O(n^2)$, but with factor of 10 saving on the constant

Efficient similarity analysis: Summary

1. Construct a characteristic matrix:
   - columns are population members e.g. documents, customers
   - rows are items, e.g. hashed k-shingles, items for sale
2. Compute *m* minhash signatures.

Jaccard similarity of C_i and C_j is approximately the proportion of minhash signatures which agree in column i and column j

1. Choose a similarity threshold, *t*
2. Construct candidate pairs by applying LSH
3. Compute similarities for candidate pairs from minhash signatures
4. Check similarity for a few original documents (to verify nothing went wrong).

Bonus material: Other Similarity Measures and LSH

- No guarantee that a particular distance / similarity measure has a locality-sensitive family of hash functions
- However possible to do so for:-
  - Hamming distance
  - Cosine distance
  - Euclidean distance

Bonus material: LSH for Cosine

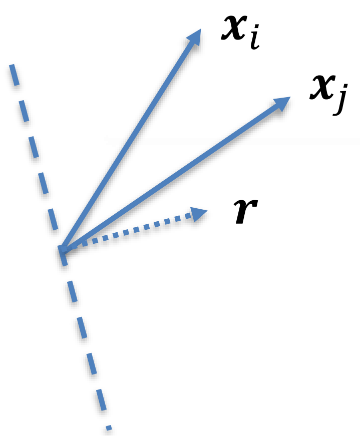
- Consider two points (described by position vectors) and a random hyperplane through the Origin
- The two points are either on the same side of the hyperplane or on different sides of the hyperplane
- If we take the dot product of each vector with the normal vector to the plane
  - Same sign -> same side of hyperplane
  - Different signs -> different sides of hyperplane

Bonus material: LSH for Cosine

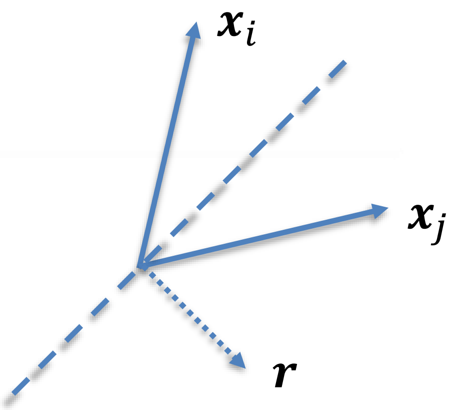
- The probability that a *random hyperplane* separates two unit vectors depends on the angle between them:

$$\Pr[\mathrm{sign}(\boldsymbol{x}_i^T \boldsymbol{r}) = \mathrm{sign}(\boldsymbol{x}_j^T \boldsymbol{r})] = 1 - \frac{1}{\pi}\cos^{-1}(\boldsymbol{x}_i^T \boldsymbol{x}_j)$$

High dot product: unlikely to split



Lower dot product: likely to split



Corresponding hash function:

$$h_{\boldsymbol{r}}(\boldsymbol{x}) = \begin{cases} 1, & \text{if } \boldsymbol{r}^T \boldsymbol{x} \geq 0 \\ 0, & \text{otherwise} \end{cases}$$

What have you learnt about the following topics?

- applications of near-neighbour search
- similarity and distance measures
- string similarity
- shingling
- min-hashing
- LSH