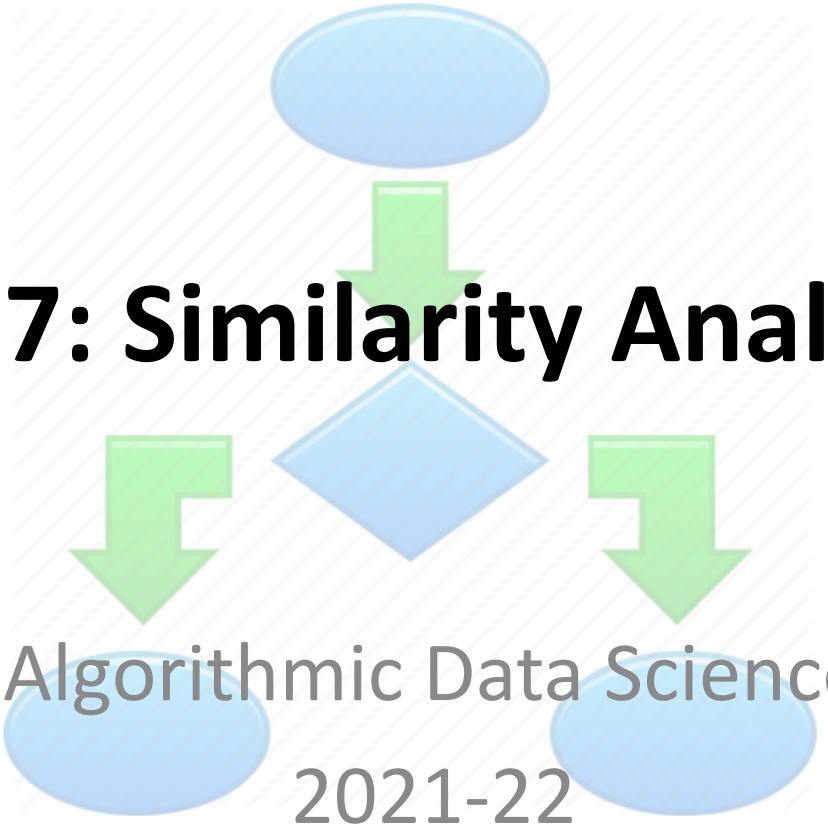


Week 7: Similarity Analysis I

Algorithmic Data Science

2021-22

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?

Midterm feedback response

- I'm glad many of you are finding that the teaching on this module helps to make the content clear.
- **Re pace.** I'm sorry some of you think it is too fast. I agree there was a jump in difficulty of labs a few weeks in. But I think it now levels off. It is not expected that you understand everything completely the first time. At first exposure, just aim to get the gist of more advanced material.
  - *I want to do the lab as normal this week, because it will prepare you for the assignment. But next week, there will be no new worksheet.*
  - *Next week, I will devote half the lecture to questions.*
  - *I have set up a Padlet for questions.*
  - *Students' backgrounds are very diverse, and it is expected that some students will need to use more initiative to keep up: use office hours, and speak to each other via Canvas Discussions and Discord.*
- **Re lab exercises.** I have provided you with a lot of code and a lot of solutions. You should look through it all. Remember this is MSc level- you have to choose a little bit which questions to focus your energy and time on, and some questions are open-ended.
  - *Going forward, we will devote more time to going through lab exercise solutions.*
  - *If you are behind, prioritise this week's lab, and labs 1 and 2 to prepare for the assignment.*
- **Re diverse learning materials**. I try to post blogs / YouTube videos when I can. Please also search yourselves and share good things in Canvas Discussions or Discord.

Week	Who	Topic
1	Barrett	Data structures and data formats
2	Barrett	Algorithmic complexity. Sorting.
3	Barrett	Matrices: Manipulation and computation
4	Wakeman	Processes and concurrency
5	Wakeman	Distributed computation
6	Barrett	Map/reduce
7	**Barrett**	**Similarity analysis**
8	Barrett	Similarity (bit more), graphs/networks 1, Q&A
9	Barrett	Graphs/networks 2, PageRank algorithm
10	Barrett	Databases
11		*independent study / Q&A*

Space complexity / Memory requirements

```
:
    1 def insertion_sort(alist):
    2     for index in range(1,len(alist)):
    3         item=alist[index]
    4         sofar=index-1
    5         while sofar>-1 and alist[sofar]>item:
    6             alist[sofar+1]=alist[sofar]
    7             sofar-=1
    8         alist[sofar+1]=item
    9     return alist
```

For each element in the list, find its correct place in the already sorted list to the left. Insert it (by shifting everything up) and proceed to next element.

No new lists created, so (asymptotic) memory requirement is just the size of the inputted list:

a*n

where a is memory per list item. (Can neglect variables of size O(1).)

Naïve matrix multiplication

```
Matrix-Multiply (A,B):
  # Assume A and B are nxn matrices:
    let C be an nxn matrix
    for i in range(1,n):
       for j in range(1,n):
          c_ij = 0
          for k in range(1,n):
             c_ij += a_ik * b_kj
  return C
```

Whole new matrix created, so (asymptotic) memory requirement is the size of the inputted matrices plus the size of the output

$3*a*n^2$

where a is memory per matrix component. (Can neglect variables of size O(1) and O(n).)

Overview

- applications of similarity / near-neighbour search
- similarity and distance measures
- string similarity
- shingling
- minhashing

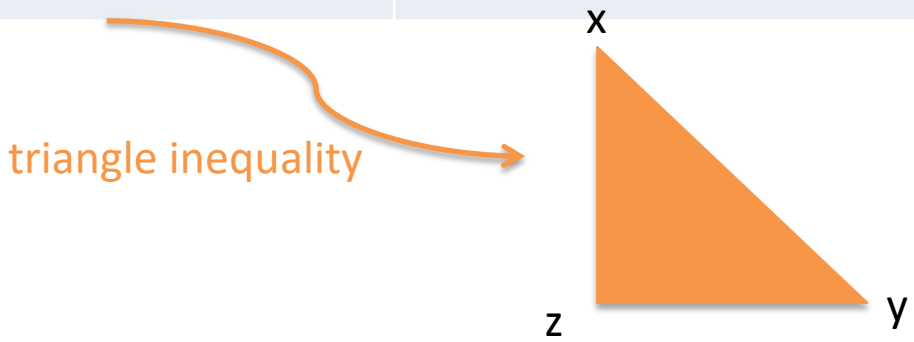
Applications of similarity

- 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.

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 $x=y$
$d(x,y) = d(y,x)$	$sim(x,y) = sim(y,x)$
$d(x,y) \leq d(x,z) + d(z,y)$	

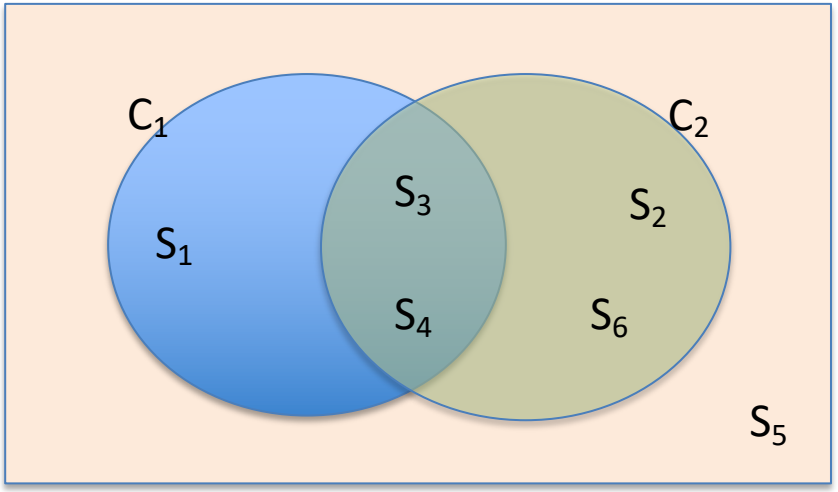


Example

- 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.



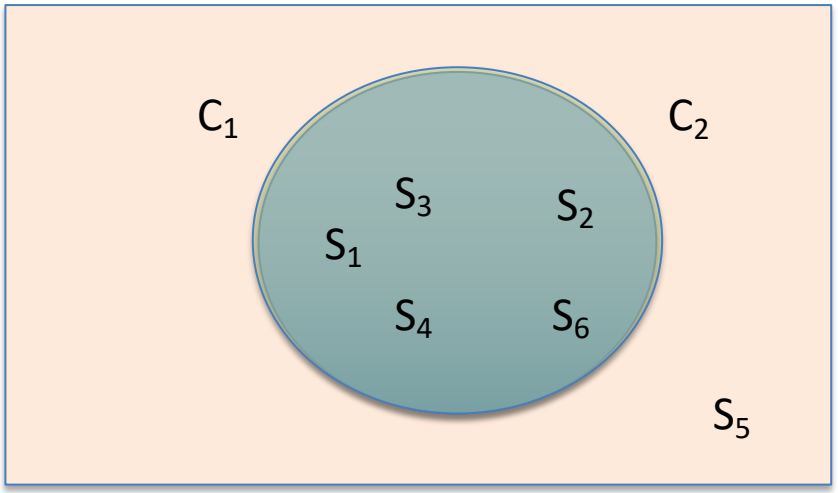
$C_1=\{S_1, S_3, S_4\}$

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

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

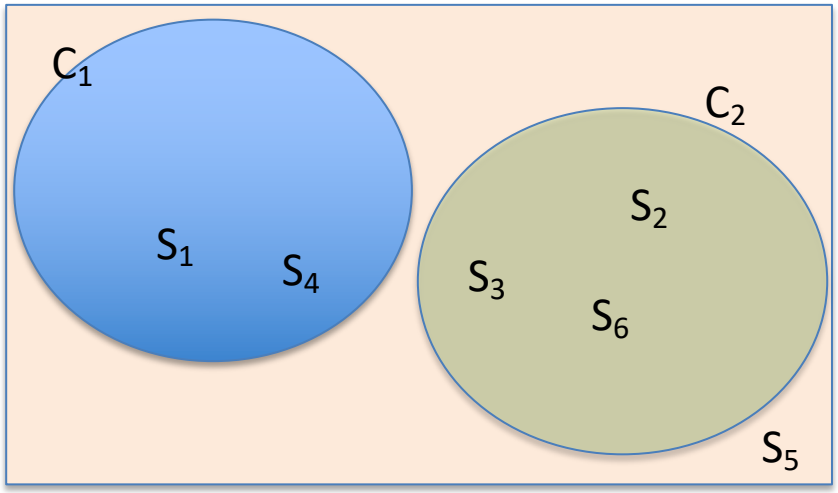
$$Jacc(C_1,C_2)=\frac{|C_1\cap C_2|}{|C_1\cup C_2|}=\frac{|C_1\cap C_2|}{|C_1|+|C_2|-|C_1\cap C_2|}=\frac{2}{5}$$

Special cases



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

$Jacc(C_1, C_2) = 5/5 = 1$



$C_1 = \{S_1, S_4\}$

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

$Jacc(C_1, C_2) = 0/5 = 0$

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?

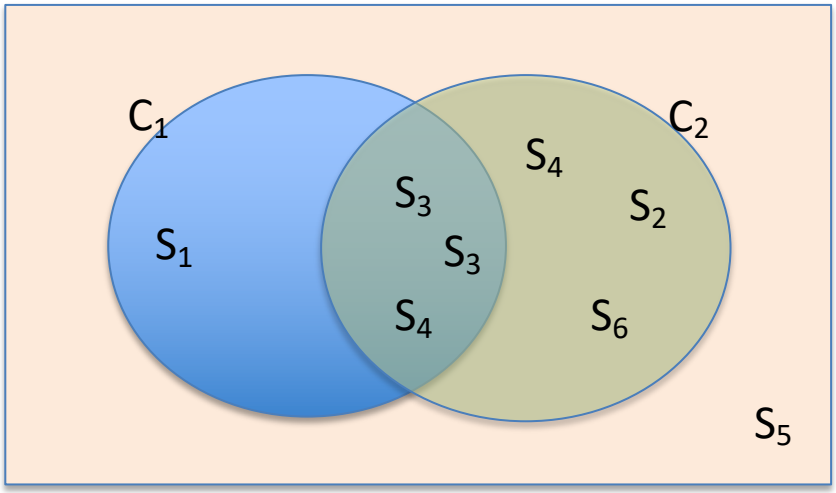
Algorithm for Jaccard's measure

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}

	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

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:**

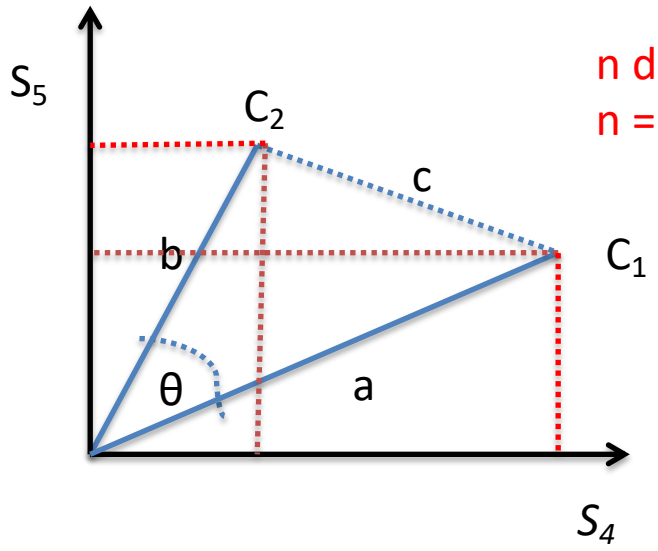
the if statement takes O(1) to execute- just look at what is stored at the hash of the item.

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.

Cosine similarity

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



n dimensions,
n = 6 here

	C_1	C_2
S_1	1	0
S_2	0	1
S_3	2	2
S_4	3	1
S_5	1	1.5
S_6	0	1

$C_1.C_2$

0
0
4
3
1.5
0
Σ=8.5

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

$C_1.C_2$ is the dot product. Also known as the inner product $\langle C_1, C_2 \rangle$ 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

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)

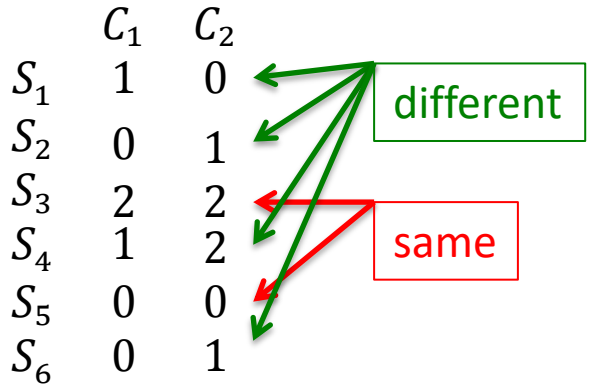
To compute cosine similarity between 2 vectors X and Y

- 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.

Other measures: Hamming distance

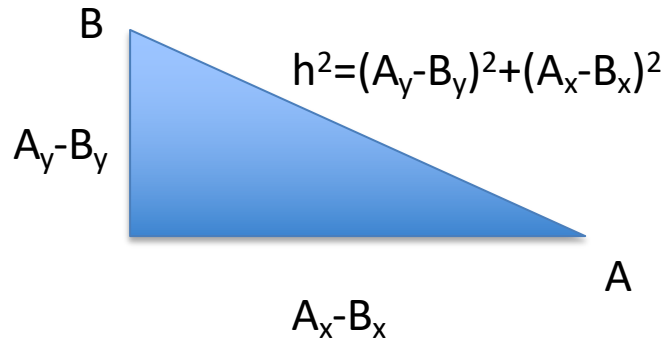
- The number of vector components (dimensions) in which two objects differ.
- Usually only applied to Boolean vectors (e.g., sets) but can be applied to bags



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

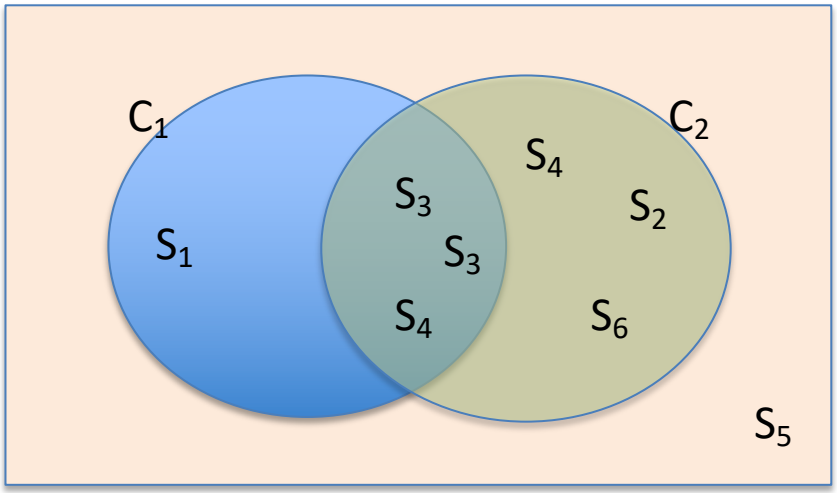
$$L_k(A,B) = \sqrt[k]{\sum_{i=1}^{n}|A_i - B_i|^k}$$

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



	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

$\rightarrow$

	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

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

Probabilistic measures of similarity

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

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

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?

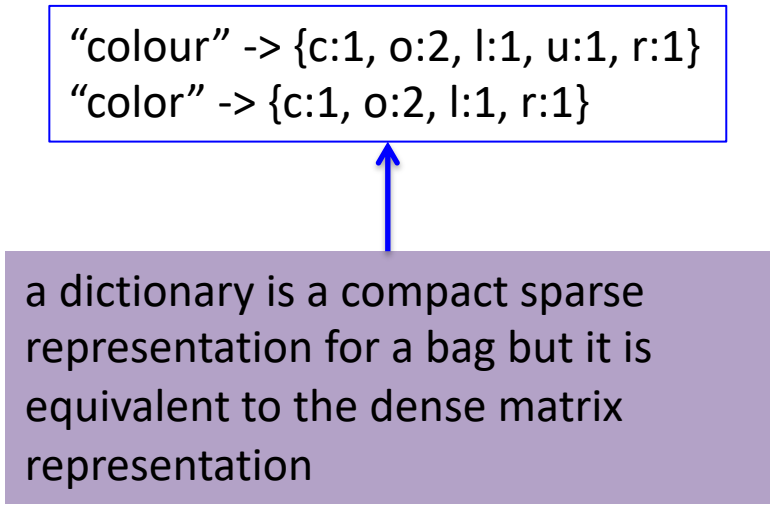
This is not strictly a distance measure because it is not symmetric. The Jenson-Shannon divergence measure is the symmetric version, which measures distance as the average Kullback-Leibler divergence to the centroid of the distributions.

JS(A,B) = ½ (KL(A,M) + KL(B,M))

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

Finding similarity of two strings

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



Jacc(“colour”, “color”) = 5/6

cos(“colour”, “color”) = ?

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.

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	??	?

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
  - *What would be the Jaccard similarity if we used a bag of bigrams?*
- 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

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

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?

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

permutation 2

	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

	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

What have you learnt about the following topics?

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