Smallest font

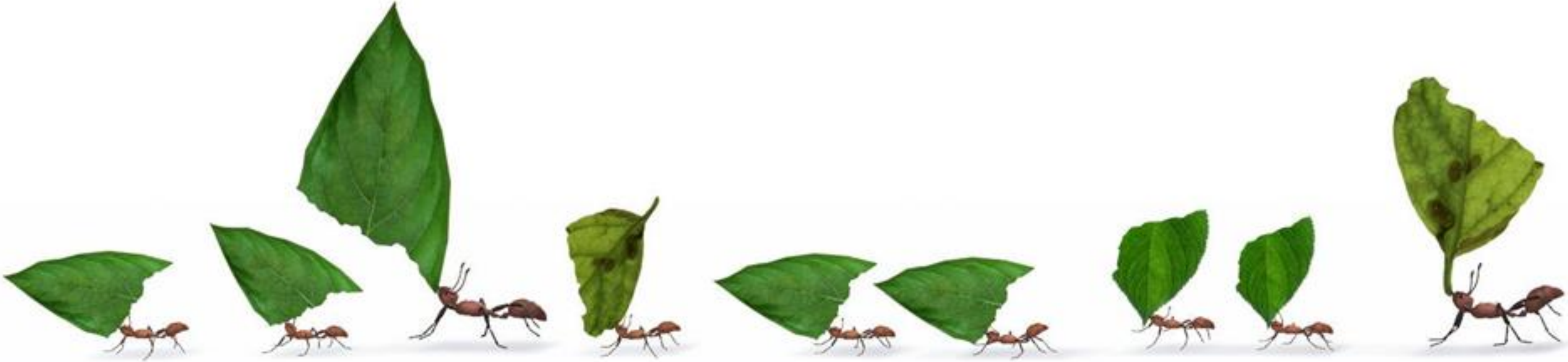
Please turn off and put away your cell phone

Calibration slide

These slides are meant to help with note-taking
They are no substitute for lecture attendance

Smallest font

Big Data



NYU | Center for Data Science

Week 09: Similarity based search

DS-GA 1004: Big Data

We are at a pivotal point in this class

- We spent 2/3 of the time in this class on various frameworks to distribute work efficiently, if there is too much work to do for a single worker.
- This is sound (both in terms of distributed data storage and suitably distributed computing) in terms of scaling, reliability, maintainability.
- But there is another approach (which we will spend the last 1/3 of this class on): Working *smart*(er).
- However, what “smart” means is often application specific (e.g. search, recommendation, etc.)

If you have a big problem: Looking for a small needle in a large haystack

Is it your only option to hire more workers and/or distribute the large stack into smaller piles?

You could also work smarter:

Big Data = Big opportunity

- Big Data = Big problem
- Scale brings its own inherent challenges
- Potential solutions:
- Divide the problem into smaller problems, then solve them in parallel.
- Use a more suitable data structure / format
- Work smarter, not harder

Finding items in a large collection

- **Search** and **recommendation** rely on similarity calculation

- User provides a "**query**", e.g.:
  - Search string
  - Example document
  - Latent representation

- System returns a list of matching "**documents**" from the database

Examples abound

- **Text search**: search string ⇔ the web (documents)
- **Recommender systems**: user representation ⇔ item representation
- **Reverse image search**: photo ⇔ library
- **Copyright detection**: uploaded video ⇔ all of youtube
- **Plagiarism detection**: uploaded document ⇔ all documents

Basic approach:
Conceptually very straightforward

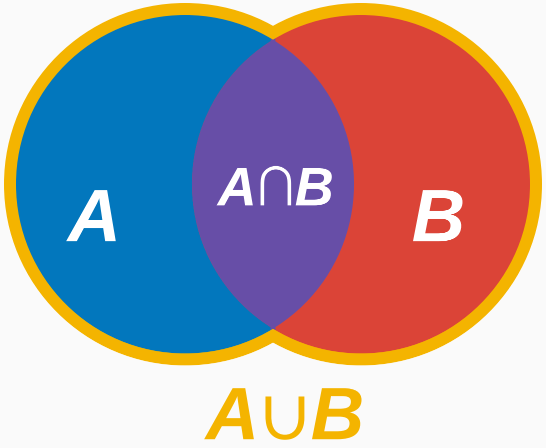
- Given a query *q*
- For each document *d* in collection
  - Compute **similarity**(*q*, *d*)
- Order collection by decreasing **similarity**
- Return top *k* documents

How can we do this efficiently?

First, we need to quantify the similarity of sets: **Jaccard similarity**

- Items are represented as **sets**, *e.g.*:
    - ***A*** = {words contained in document *A*}
    - ***B*** = {users who interacted with item *B*}
- ***Jaccard similarity*** is the ratio

  $J(A, B) = |A \cap B| / |A \cup B|$



- D(A, B) = 1 - J(***A***, ***B***) is the ***Jaccard distance metric***

But scale could pose problems:

- Size of collection (*N*) – at least N comparisons needed
- Dimensionality of representation (could be high k)
- More generally: complexity of computing similarity
- ***Can we do better (=faster) than brute force search?***

Instead: Approximate search as a first pass/filter

- Use a **fast method** to identify $n \ll N$ **candidate nearest neighbors**
- Use **true similarity** on **candidate set** to discard any false positives
- This is common sense/typical for many applications / outside of big data
- Within Data Science, a **fast method** for similarity search of documents usually requires some kind of data structure
  - Search time should be strictly sub-linear: $n \sim o(N)$
  - e.g. **log(*N*)** or **sqrt(*N*)**

Approximate search as a preliminary filter is just common sense

- For instance, imagine a large tech company (or an academic department, for that matter) receives many applications (e.g. 1k) for an advertised position.
- That's a good problem to have for the people doing the hiring, as sample size is large – an ideal candidate is probably in the set of applications, but:
- The ideal candidate will be a good match on many dimensions, which has to be revealed by a multitude of interviews (=many perspectives).
- Is it at all feasible to put all 1k applicants though this process?
- Takes too long / is too much work for one interviewer or committee.
- One solution: Hire more interviewers and interview the candidates in parallel.
- Another solution: Filter by scanning resumes, then only interview (far fewer) candidates that are *likely* a good match.
- Note: This is an approximate search. It does not guarantee success – you might be missing good candidates and also have false positives.

MinHash

[Broder, 1997]

Before we go into the details of minHash

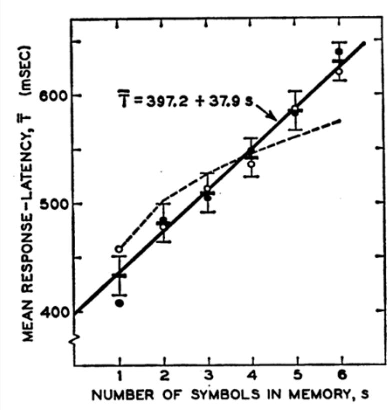
- Some necessary review:
- What is a hash function?
- A function that takes any input and returns fixed-size bytes.
- What is a hash collision?
- When two different inputs yield the same output hash value.
- Example: The digit root (DR) function
- DR(12) = 3, DR (15) = 6, DR(18) = 9, DR (21) = 3
- Digit root of multiples of 9 *always* collide (in base 10):
- DR(9) = DR (18) = DR (27) = DR (36) = DR (45) = DR (54) = … = 9

The problem with similarity search of large collections, e.g. for plagiarism detection

Serial scans are inherently slow

High-Speed Scanning in Human Memory

Abstract. *When subjects judge whether a test symbol is contained in a short memorized sequence of symbols, their mean reaction-time increases linearly with the length of the sequence. The linearity and slope of the function imply the existence of an internal serial-comparison process whose average rate is between 25 and 30 symbols per second.*



is retrieved. Of par- the study of retrieval e number of elements the response latency. t memorizes a short

nuli. On each trial of subject (4) saw a ran- om one to six different ingly at a fixed locus each. The length, *s*, of

the selection of a response requires the use of information that is in memory, the latency of the response will reveal something about the process by which

the series varied at random from trial to trial. There followed a 2.0-second delay, a warning signal, and then the test digit. As soon as one of the levers

Results are shown in Fig. 1. Linear regression accounts for 99.4 percent of the variance of the overall mean response-latencies (6). The slope of the fitted line is 37.9 ± 3.8 msec per symbol (7); its zero intercept is 397.2 ± 19.3 msec. Lines fitted separately to the mean latencies of positive and negative responses differ in slope by 9.6 ± 2.3 msec per symbol. The difference is attributable primarily to the fact that for *s* = 1, positive responses were 50.0 ± 20.1 msec faster than negative responses. Lines fitted to the data for

On trials requiring negative responses, *s* comparisons must be made. If positive responses were initiated as soon as a match had occurred (as in a self-terminating search), the mean number of comparisons on positive trials would be $(s + 1)/2$ rather than *s*. The latency function for positive responses would then have half the slope of the function for negative responses. The

lar to that used in more conventional experiments on choice-reaction time. In experiment 1, the set of symbols associated with the positive response changed from trial to trial. In contrast to this varied-set procedure, a fixed-set procedure was used in experiment 2. In each of three parts of the session, a set of digits for which the

Sternberg, Science, 1966

Many pairwise comparisons are needed

N (documents)	# pairwise comparisons
2	1
10	45
100	4950
1000	499,500
1e4	49.995e6
1e5	49.9995e8
1e6	49.9999e10

How does MinHash work?

Note: π is not 3.14159… here.
They just needed a Greek letter

- Fix a random ordering π of the items (items = words)
- Here is a table of set memberships
- For each set $\boldsymbol{S}$, its hash is:

$$h(\boldsymbol{S} \mid \pi) = \mathbf{min}\ \{k \mid \pi(k) \in \boldsymbol{S}\}$$

  - The index of the first (permuted) item belonging to set $\boldsymbol{S}$

	$\pi(k)$	Doc 1	Doc 2	Doc 3	Doc 4	...
1	Item 3	1	0	0	0	
2	Item 75	0	0	1	0	
3	Item 21	0	1	1	0	
4	Item 1	0	0	1	0	
5	Item 2004	0	1	0	1	

$$h(\mathbf{Doc\ 1} \mid \pi) = 1$$
$$h(\mathbf{Doc\ 2} \mid \pi) = 3$$
$$h(\mathbf{Doc\ 3} \mid \pi) = 2$$
$$h(\mathbf{Doc\ 4} \mid \pi) = 5$$

A specific example of permutation indexing in practice

A = {“T.rex”, “Stegosaurus”, “PDP-11”}

B = {“Apples”, “Bananas”, “Pine cones”}

C = {“T.rex”, “Bananas”, “Penguins”}

D = {“Apples”, “Turtles”, “ Pine cones”}

$\pi(k)$		A	B	C	D
1	PDP-11	1	0	0	0
2	Penguins	0	0	1	0
3	Pine cones	0	1	0	1
4	Turtles	0	0	0	1
5	Apples	0	1	0	1
6	T.rex	1	0	1	0
7	Bananas	0	1	1	0
8	Stegosaurus	1	0	0	0

A specific example of permutation indexing in practice

A = {“T.rex”, “Stegosaurus”, “PDP-11”} $\rightarrow h(\mathbf{A}|\pi) = 1$

B = {“Apples”, “Bananas”, “Pine cones”} $\rightarrow h(\mathbf{B}|\pi) = 3$

C = {“T.rex”, “Bananas”, “Penguins”} $\rightarrow h(\mathbf{C}|\pi) = 2$

D = {“Apples”, “Turtles”, “ Pine cones”}. $\rightarrow h(\mathbf{D}|\pi) = 3$

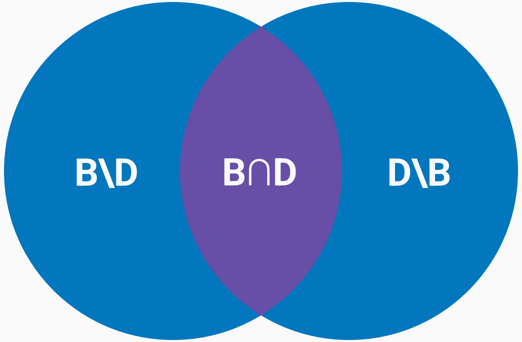
Hash collision is more likely when sets overlap.
Let’s analyze this more formally!

$\pi(k)$		A	B	C	D
1	PDP-11	1	0	0	0
2	Penguins	0	0	1	0
3	Pine cones	0	1	0	1
4	Turtles	0	0	0	1
5	Apples	0	1	0	1
6	T.rex	1	0	1	0
7	Bananas	0	1	1	0
8	Stegosaurus	1	0	0	0

$P[h(S_1) = h(S_2)] = \text{Jaccard}(S_1, S_2)$

- For two sets S_1 and S_2, there are three types of rows:
  - **Type 1**: $\pi(k) \in S_1 \cap S_2$
  - **Type 2**: $\pi(k) \in S_1 \Delta S_2$
  - Type 3: $\pi(k) \notin S_1 \cup S_2$

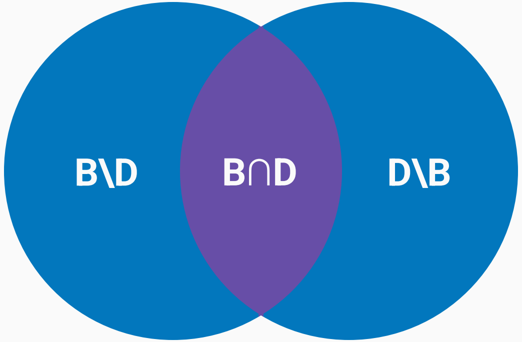
- **Insight:**
  Collision $\Leftrightarrow$ **type 1 row** occurs before all **type 2 rows**



$\pi(k)$		A	B	C	D
1	PDP-11	1	0	0	0
2	Penguins	0	0	1	0
3	**Pine cones**	0	**1**	0	**1**
4	**Turtles**	0	**0**	0	**1**
5	**Apples**	0	**1**	0	**1**
6	T.rex	1	0	1	0
7	**Bananas**	0	**1**	1	**0**
8	Stegosaurus	1	0	0	0

$P[h(S_1) = h(S_2)] = \text{Jaccard}(S_1, S_2)$

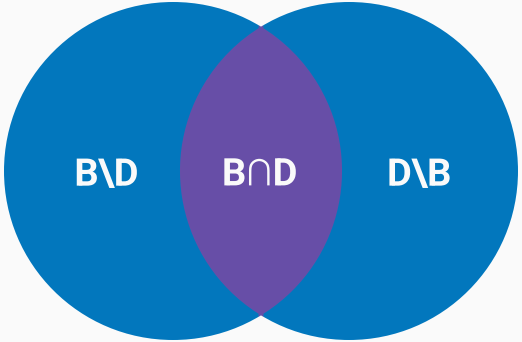
- For two sets S_1 and S_2, there are three types of rows:
  - **Type 1**: $\pi(k) \in S_1 \cap S_2$
  - **Type 2**: $\pi(k) \in S_1 \Delta S_2$
  - Type 3: $\pi(k) \notin S_1 \cup S_2$
- **Collision** $\Leftrightarrow$ **type 1 row** occurs before all **type 2 rows**
- P[**Collision**] = (# Type 1) / (# Type 1 + # Type 2)



$\pi(k)$		A	B	C	D
1	PDP-11	1	0	0	0
2	Penguins	0	0	1	0
3	**Pine cones**	0	**1**	0	**1**
4	**Turtles**	0	**0**	0	**1**
5	**Apples**	0	**1**	0	**1**
6	T.rex	1	0	1	0
7	**Bananas**	0	**1**	1	**0**
8	Stegosaurus	1	0	0	0

$P[h(S_1) = h(S_2)] = \text{Jaccard}(S_1, S_2)$

- For two sets S_1 and S_2, there are three types of rows:
  - **Type 1**: $\pi(k) \in S_1 \cap S_2$
  - **Type 2**: $\pi(k) \in S_1 \Delta S_2$
  - Type 3: $\pi(k) \notin S_1 \cup S_2$
- **Collision** ⇔ **type 1 row** occurs before all **type 2 rows**
- P[Collision] $= (\#\ \text{Type 1}) / (\#\ \text{Type 1} + \#\ \text{Type 2})$
  $= |S_1 \cap S_2| / (|S_1 \cap S_2| + |S_1 \Delta S_2|)$

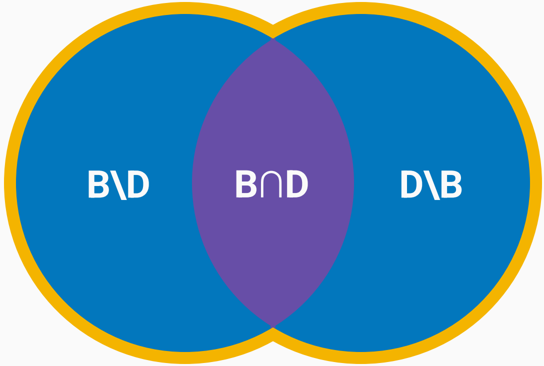


$\pi(k)$		A	B	C	D
1	PDP-11	1	0	0	0
2	Penguins	0	0	1	0
3	**Pine cones**	0	**1**	0	**1**
4	**Turtles**	0	**0**	0	**1**
5	**Apples**	0	**1**	0	**1**
6	T.rex	1	0	1	0
7	**Bananas**	0	**1**	1	**0**
8	Stegosaurus	1	0	0	0

$P[h(S_1) = h(S_2)] = \text{Jaccard}(S_1, S_2)$

- For two sets S_1 and S_2, there are three types of rows:
  - **Type 1**: $\pi(k) \in S_1 \cap S_2$
  - **Type 2**: $\pi(k) \in S_1 \Delta S_2$
  - Type 3: $\pi(k) \notin S_1 \cup S_2$
- **Collision** $\Leftrightarrow$ **type 1 row** occurs before all **type 2 rows**
- P[**Collision**] $= (\#\ \text{Type 1}) / (\#\ \text{Type 1} + \#\ \text{Type 2})$

$$= |S_1 \cap S_2| / (|S_1 \cap S_2| + |S_1 \Delta S_2|)$$
$$= |S_1 \cap S_2| / (|S_1 \cup S_2|)$$
$$= J(S_1, S_2) \blacksquare$$



$\pi(k)$		A	B	C	D
1	PDP-11	1	0	0	0
2	Penguins	0	0	1	0
3	**Pine cones**	0	**1**	0	**1**
4	**Turtles**	0	**0**	0	**1**
5	**Apples**	0	**1**	0	**1**
6	T.rex	1	0	1	0
7	**Bananas**	0	**1**	1	**0**
8	Stegosaurus	1	0	0	0

Monte Carlo approximation of all possible permutations

$\pi(k)$		A	B	C	D
1	PDP-11	1	0	0	0
2	Penguins	0	0	1	0
3	Pine cones	0	1	0	1
4	Turtles	0	0	0	1
5	Apples	0	1	0	1
6	T.rex	1	0	1	0
7	Bananas	0	1	1	0
8	Stegosaurus	1	0	0	0

- **B** and **D** had **P[collision]** = ½ for a **single permutation** π
- But we want the ***probability of collision*** over **all choices** of π

Table of set memberships of one particular permutation

- **Idea**:

Table of minhash signatures across permutations

  - Generate m **random permutations** $\pi_1, \pi_2, \ldots, \pi_m$
  - **Count hash collisions** between A and B over all π_i's
  - $J(\boldsymbol{A}, \boldsymbol{B}) \approx$ # collisions / m

$h(S \mid \pi_i)$	A	B	C	D
π_1	7	1	2	1
π_2	2	1	2	5
π_3	4	5	3	6
π_4	9	5	1	1
π_5	3	2	6	2
...				

MinHash signatures

But full permutations or large sets is still costly
Can we approximate the permutations too?

- Computing and storing random permutations of large collections is not practical, as it is simply too costly

- Instead, we can replace ***permutations*** π_i with ***hashes*** H_i
  - A permutation is a **perfect hash**: distinct elements cannot collide
  - Approximate this by an **imperfect hash**: distinct elements ***may collide***

- As long as H_i are unlikely to collide, this can still work

- Note: This is an approximation to an approximation (2^{nd} order).

Computing minhash signatures with hash functions (approximate minhash)

x	$H_1(x)$	$H_2(x)$	A	B	C	D
PDP-11	0	0	1			
Penguins	1	2			1	
Pine cones	2	4		1		1
Turtles	3	0				1
Apples	0	1		1		1
T.rex	1	3	1		1	
Bananas	2	5		1	1	
Stegosaurus	3	1	1			

	A	B	C	D
H_1	∞	∞	∞	∞
H_2	∞	∞	∞	∞

Signature table is initialized to infinity for each entry

Computing minhash signatures with hash functions (approximate minhash)

x	$H_1(x)$	$H_2(x)$	A	B	C	D
PDP-11	0	0	1			
Penguins	1	2			1	
Pine cones	2	4		1		1
Turtles	3	0				1
Apples	0	1		1		1
T.rex	1	3	1		1	
Bananas	2	5		1	1	
Stegosaurus	3	1	1			

	A	B	C	D
H_1	$0\ \cancel{\infty}$	∞	∞	∞
H_2	$0\ \cancel{\infty}$	∞	∞	∞

A, H_1: $0 < \infty \rightarrow$ update
A, H_2: $0 < \infty \rightarrow$ update

Computing minhash signatures with hash functions (approximate minhash)

x	$H_1(x)$	$H_2(x)$	A	B	C	D
PDP-11	0	0	1			
Penguins	1	2			1	
Pine cones	2	4		1		1
Turtles	3	0				1
Apples	0	1		1		1
T.rex	1	3	1		1	
Bananas	2	5		1	1	
Stegosaurus	3	1	1			

	A	B	C	D
H_1	0	∞	$1\ \cancel{\infty}$	∞
H_2	0	∞	$2\ \cancel{\infty}$	∞

C, H_1: $1 < \infty \rightarrow$ update
C, H_2: $2 < \infty \rightarrow$ update

Computing minhash signatures with hash functions (approximate minhash)

x	$H_1(x)$	$H_2(x)$	A	B	C	D
PDP-11	0	0	1			
Penguins	1	2			1	
Pine cones	2	4		1		1
Turtles	3	0				1
Apples	0	1		1		1
T.rex	1	3	1		1	
Bananas	2	5		1	1	
Stegosaurus	3	1	1			

	A	B	C	D
H_1	0	2 $\cancel{\infty}$	1	2 $\cancel{\infty}$
H_2	0	4 $\cancel{\infty}$	2	4 $\cancel{\infty}$

B, H_1: $2 < \infty \rightarrow$ update
B, H_2: $4 < \infty \rightarrow$ update
D, H_1: $2 < \infty \rightarrow$ update
D, H_2: $4 < \infty \rightarrow$ update

Computing minhash signatures with hash functions (approximate minhash)

x	$H_1(x)$	$H_2(x)$	A	B	C	D
PDP-11	0	0	1			
Penguins	1	2			1	
Pine cones	2	4		1		1
Turtles	3	0				1
Apples	0	1		1		1
T.rex	1	3	1		1	
Bananas	2	5		1	1	
Stegosaurus	3	1	1			

	A	B	C	D
H_1	0	2	1	2
H_2	0	4	2	0 4

D, H_1: $3 > 2 \rightarrow$ no update
D, H_2: $0 < 4 \rightarrow$ update

Computing minhash signatures with hash functions (approximate minhash)

x	$H_1(x)$	$H_2(x)$	A	B	C	D
PDP-11	0	0	1			
Penguins	1	2			1	
Pine cones	2	4		1		1
Turtles	3	0				1
Apples	0	1		1		1
T.rex	1	3	1		1	
Bananas	2	5		1	1	
Stegosaurus	3	1	1			

	A	B	C	D
H_1	0	0 ~~2~~	1	0 ~~2~~
H_2	0	1 ~~3~~	2	0

B, H_1: $0 < 2 \rightarrow$ update
B, H_2: $1 < 3 \rightarrow$ update
D, H_1: $0 < 2 \rightarrow$ update
D, H_2: $1 > 0 \rightarrow$ no update

Computing minhash signatures with hash functions (approximate minhash)

x	$H_1(x)$	$H_2(x)$	A	B	C	D
PDP-11	0	0	1			
Penguins	1	2			1	
Pine cones	2	4		1		1
Turtles	3	0				1
Apples	0	1		1		1
T.rex	1	3	1		1	
Bananas	2	5		1	1	
Stegosaurus	3	1	1			

	A	B	C	D
H_1	0	0	1	0
H_2	0	1	2	0

A, H_1: $1 > 0 \rightarrow$ no update
A, H_2: $3 > 0 \rightarrow$ no update
C, H_1: $1 = 1 \rightarrow$ no update
C, H_2: $3 > 2 \rightarrow$ no update

Computing minhash signatures with hash functions (approximate minhash)

x	$H_1(x)$	$H_2(x)$	A	B	C	D
PDP-11	0	0	1			
Penguins	1	2			1	
Pine cones	2	4		1		1
Turtles	3	0				1
Apples	0	1		1		1
T.rex	1	3	1		1	
Bananas	2	5		1	1	
Stegosaurus	3	1	1			

	A	B	C	D
H_1	0	0	1	0
H_2	0	1	2	0

B, H_1: $2 > 0 \rightarrow$ no update
B, H_2: $5 > 1 \rightarrow$ no update
C, H_1: $2 > 1 \rightarrow$ no update
C, H_2: $5 > 2 \rightarrow$ no update

Computing minhash signatures with hash functions (approximate minhash)

x	$H_1(x)$	$H_2(x)$	A	B	C	D
PDP-11	0	0	1			
Penguins	1	2			1	
Pine cones	2	4		1		1
Turtles	3	0				1
Apples	0	1		1		1
T.rex	1	3	1		1	
Bananas	2	5		1	1	
Stegosaurus	3	1	1			

	A	B	C	D
H_1	0	0	1	0
H_2	0	1	2	0

A, H_1: $3 > 0 \rightarrow$ no update
A, H_2: $1 > 0 \rightarrow$ no update

Computing minhash signatures with hash functions (approximate minhash)

x	$H_1(x)$	$H_2(x)$	A	B	C	D
PDP-11	0	0	1			
Penguins	1	2			1	
Pine cones	2	4		1		1
Turtles	3	0				1
Apples	0	1		1		1
T.rex	1	3	1		1	
Bananas	2	5		1	1	
Stegosaurus	3	1	1			

	A	B	C	D
H_1	0	0	1	0
H_2	0	1	2	0

Collisions:

- H_1: $\mathbf{A} \equiv \mathbf{B} \equiv \mathbf{D} \not\equiv \mathbf{C}$
- H_2: $\mathbf{A} \equiv \mathbf{D} \not\equiv \mathbf{B} \not\equiv \mathbf{C}$

Big picture: Similarity search and minHash

- Similarity search is straightforward – just compare the similarity of documents in a collection with a metric like Jaccard similarity.
- However, this is too costly for large collections, so an approximation that identifies candidates is indicated, we can then compute full similarity for candidate subset.
- To identify candidates, we can use minHash, as the probability of a hash collision corresponds to the Jaccard similarity.
- Conceptually, we can use random permutations, as the number of collisions in the table of minHash signatures corresponds to the Jaccard similarity (scaled by the number of permutations we do) to identify a candidate set.
- There is a 2nd layer. In addition to using minHash, we can also use hash functions as a proxy for permutations, as full permutations are too costly for large collections.
- AFTER minHash has identified a – much reduced - candidate set, we can do the full similarity search (as it will be a small data problem at that point).

Failure modes of MinHash

- Permutation MinHash:
  - **Collisions** are **more likely** when a **small set of items** are shared across **many documents**
  - ⇒ “Stop-words” can be “deadly”! *“The”, “and”, “or”, etc…*

- **Hashing approximation** only makes things ***worse***
  - Two distinct items can now hash to the same value
  - Collision probability only increases with the hashing approximation
  - High collision likelihood ⇒ **large candidate sets** ⇒ **slow retrieval**

Key / solution: Keep the candidate set as small as possible, but that might lower recall

Now:

Can we reduce the size of candidate sets, but maintain high recall?

Locality sensitive hashing

[Indyk and Motwani, 1998]
[Charikar 2002]

LSH: Improving the efficiency of MinHash

- Traditional hash functions scatter data "**randomly**"
  - Probability of collision is **independent of similarity** between inputs

- **The big idea:** Locality-sensitive hashes have a higher probability of collision for inputs that are ***near each other*** *(relative to inputs that are far)*

- LSH has a huge literature, this part will be introductory

LSH + MinHash

- Carve signature matrix into ***b*** blocks of *R* rows each
- Hash each sub-column with a standard (non-local) hash function ***W***
  - Pick ***W*** such that collisions are rare for non-identical inputs
- Candidate set = items that collide in *any* row block

	A	B	C	D
H_1	0	0	1	0
H_2	0	1	2	0
H_3	1	2	0	1
H_4	0	1	0	0
H_5	2	2	0	0
H_6	1	2	1	1

	A	B	C	D
Block 1	0	5	3	0
Block 2	3	0	1	3
Block 3	0	7	2	2

$W([0, 1]) \rightarrow 2$

LSH + MinHash

- Carve signature matrix into ***b*** blocks of ***R*** rows each
- Hash each su … function ***W***
  - Pick ***W*** such …
- Candidate set = items that collide in *any* row block

What's the probability that we have at least one block where the hashes of all rows match?

	A	B	C	D
H_1	0	0	1	0
H_2	0	1	2	0
H_3	1	2	0	1
H_4	0	1	0	0
H_5	2	2	0	0
H_6	1	2	1	1

	A	B	C	D
Block 1	0	5	3	0
Block 2	3	0	1	3
Block 3	0	7	2	2

$W([0, 1]) \rightarrow 2$

LSH vs direct MinHash analysis

- MinHash:
  - **P**[single row collision] = J(A, B) = j
- LSH:
  - **P**[all R rows in a block collide] = j^R

	A	B	C	D
H_1	0	0	1	0
H_2	0	1	2	0
H_3	1	2	0	1
H_4	0	1	0	0
H_5	2	2	0	0
H_6	1	2	1	1

	A	B	C	D
Block 1	0	5	3	0
Block 2	3	0	1	3
Block 3	0	7	2	2

LSH vs direct MinHash analysis

- MinHash:
  - **P**[single row collision] = J(A, B) = j
- LSH:
  - **P**[all ***R*** rows in a block collide] = j^R
  - **P**[at least one **non-collision** in a block] = 1 - j^R

	A	B	C	D
H_1	0	0	1	0
H_2	0	1	2	0
H_3	1	2	0	1
H_4	0	1	0	0
H_5	2	2	0	0
H_6	1	2	1	1

	A	B	C	D
Block 1	0	5	3	0
Block 2	3	0	1	3
Block 3	0	7	2	2

LSH vs direct MinHash analysis

- MinHash:
  - **P**[single row collision] = J(A, B) = j
- LSH:
  - **P**[all R rows in a block collide] = j^R
  - **P**[at least one **non-collision** in a block] = 1 - j^R
  - **P**[at least one non-collision in all b blocks] = $(1 - j^R)^b$

	A	B	C	D
H_1	0	0	1	0
H_2	0	1	2	0
H_3	1	2	0	1
H_4	0	1	0	0
H_5	2	2	0	0
H_6	1	2	1	1

	A	B	C	D
Block 1	0	5	3	0
Block 2	3	0	1	3
Block 3	0	7	2	2

LSH vs direct MinHash analysis

- MinHash:
  - P[single row collision] = J(A, B) = j
- LSH:
  - P[all R rows in a block collide] = j^R
  - P[at least one **non-collision** in a block] = 1 - j^R
  - P[at least one non-collision in all b blocks] = $(1 - j^R)^b$
  - P[at least one block **collides on all rows**] = $1 - (1 - j^R)^b$

	A	B	C	D
H_1	0	0	1	0
H_2	0	1	2	0
H_3	1	2	0	1
H_4	0	1	0	0
H_5	2	2	0	0
H_6	1	2	1	1

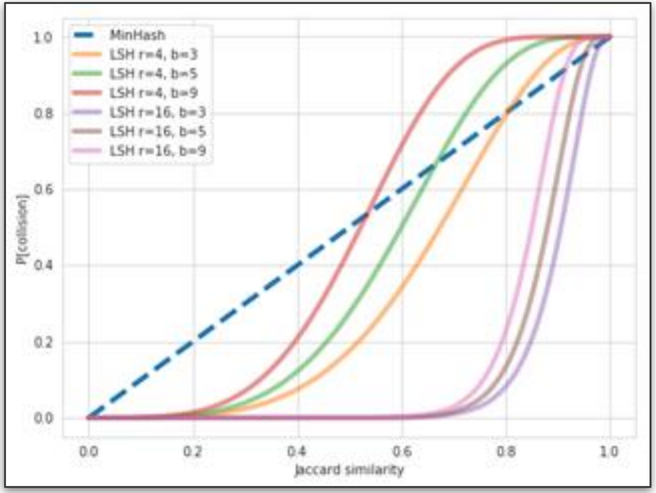
	A	B	C	D
Block 1	0	5	3	0
Block 2	3	0	1	3
Block 3	0	7	2	2

LSH vs direct MinHash analysis

- MinHash:
  - P[single row collision] = J(A, B) = j
- LSH:
  - P[all R rows in a block collide] = j^R
  - P[at least one **non-collision** in a block] = 1 - j^R
  - P[at least one non-collision in all b blocks] = $(1 - j^R)^b$
  - P[at least one block **collides on all rows**] = $1 - (1 - j^R)^b$

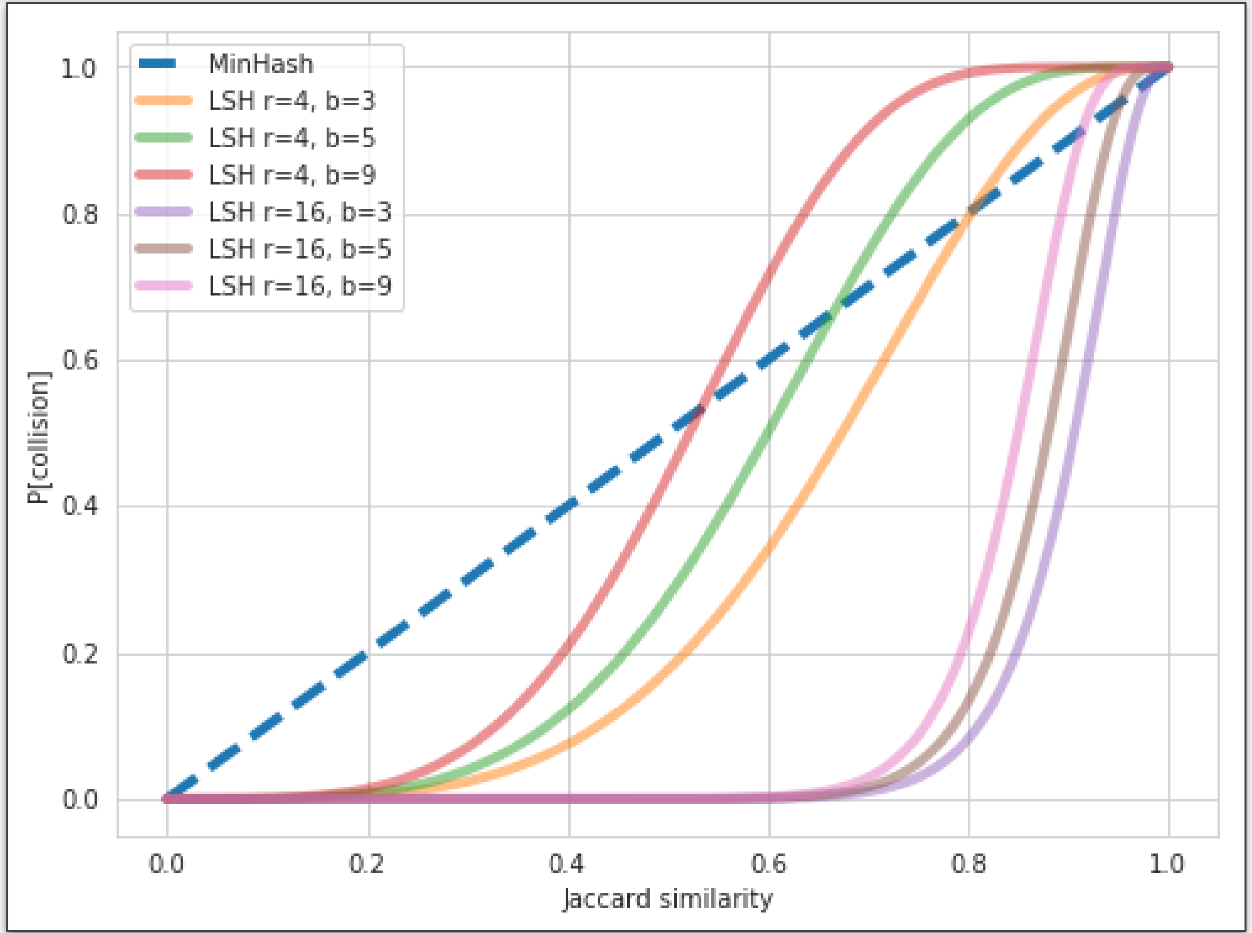
	A	B	C	D
H_1	0	0	1	0
H_2	0	1	2	0
H_3	1	2	0	1
H_4	0	1	0	0
H_5	2	2	0	0
H_6	1	2	1	1

	A	B	C	D
Block 1	0	5	3	0
Block 2	3	0	1	3
Block 3	0	7	2	2



Result:
Collisions are **more likely** for high Jaccard similarity and **less likely** for low Jaccard similarity

Tradeoffs everywhere…



Beyond simple sets:
1) Bags
2) Spatial similarity search (as in a vector database)

What if you don't just want to consider whether an item appears (in a set), but also how often it does so (in a multi-set/bag): **Ruzicka similarity**

[Chen, Philbin, Zisserman 2008]

- Idea: reduce bags to sets by uniquely identifying each repetition

$\{dog\} \rightarrow \{dog_1\}$

$\{dog, dog\} \rightarrow \{dog_1, dog_2\}$

$\{dog, dog, dog\} \rightarrow \{dog_1, dog_2, dog_3\}$

- Jaccard on expanded sets = **Ruzicka similarity** on original bags

$$R(A, B) = \frac{\sum_i \min(A[i], B[i])}{\sum_j \max(A[j], B[j])}$$

$$\begin{array}{lc} & \mathbf{A}\ \ \mathbf{B}\ \ \mathbf{C}\ \ \mathbf{D} \\ X = & [1, 3, 5, 7] \\ Y = & [2, 3, 1, 6] \end{array}$$

Σ(min(x_i, y_i)) = 1 + 3 + 1 + 6 = 11

Σ(max(x_i, y_i)) = 2 + 3 + 5 + 7 = 17

R(X, Y) = 11 / 17 ≈ 0.65

min(x_1, y_1) = min(1, 2) = 1

max(x_1, y_1) = max(1, 2) = 2

min(x_2, y_2) = min(3, 3) = 3

max(x_2, y_2) = max(3, 3) = 3

min(x_3, y_3) = min(5, 1) = 1

max(x_3, y_3) = max(5, 1) = 5

min(x_4, y_4) = min(7, 6) = 6

max(x_4, y_4) = max(7, 6) = 7

Cosine similarity is often a suitable metric to compare documents (and other high-dimensional objects)

- For instance, one could express documents in terms of the magnitude of their embedding dimensions.
- Cosine similarity conceptualizes the similarity of two vectors **a** and **b** in terms of the angle θ between them, regardless of their length.
- This makes intuitive sense:

$$cosine\ similarity = \cos(\theta) = \frac{\boldsymbol{a^T b}}{\|\boldsymbol{a}\|\|\boldsymbol{b}\|} = \frac{\boldsymbol{a^T b}}{\sqrt{\boldsymbol{a^T a}}\sqrt{\boldsymbol{b^T b}}}$$

cos(0) = 1 cos(90) = 0 cos(180) = -1

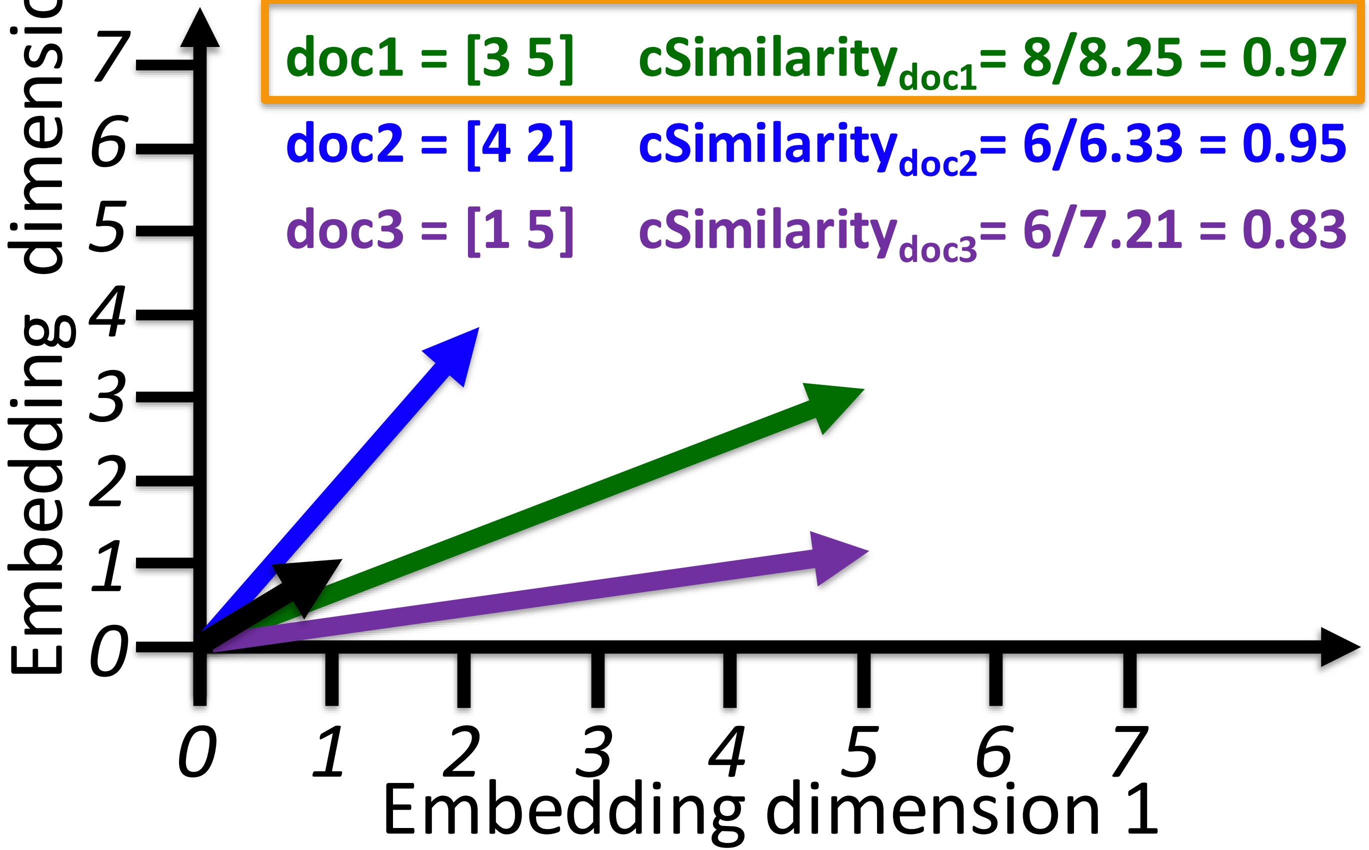
Determining relevance with cosine similarity

doc_0 = [1 1]

doc1 = [3 5] $cSimilarity_{doc1}$ = 8/8.25 = 0.97

doc2 = [4 2] $cSimilarity_{doc2}$ = 6/6.33 = 0.95

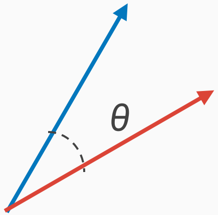
doc3 = [1 5] $cSimilarity_{doc3}$ = 6/7.21 = 0.83



LSH for cosine similarity
[Charikar 2002]

- What if we want to compare vectors $u, v \in \mathbf{R}^d$ by cosine similarity?

$$\text{sim}(u, v) = \cos(\theta)$$

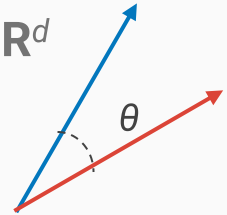


LSH for cosine similarity
[Charikar 2002]

- What if we want to compare vectors $u, v \in \mathbf{R}^d$ by cosine similarity?

$$\mathrm{sim}(u, v) = \cos(\theta)$$

- Pick a vector w randomly but uniformly from the unit sphere in $\mathbf{R}^d$
  - $h_w(x)$ $= 1$ if $w^T x >= 0$
    $= -1$ if $w^T x < 0$

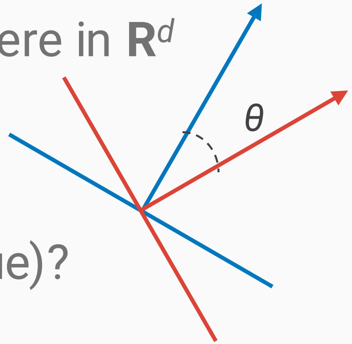


LSH for cosine similarity
[Charikar 2002]

- What if we want to compare vectors u, $v \in \mathbf{R}^d$ by cosine similarity?

  $$\text{sim}(u, v) = \cos(\theta)$$

- Pick a vector w randomly but uniformly from the unit sphere in $\mathbf{R}^d$
  - $h_w(x) = 1$ if $w^T x >= 0$
    $= -1$ if $w^T x < 0$



- What's the probability of a collision (both same hash value)?
  - $\mathbf{P}[h_w(u) = h_w(v)] = 1 - \mathbf{P}[h_w(u) \neq h_w(v)]$

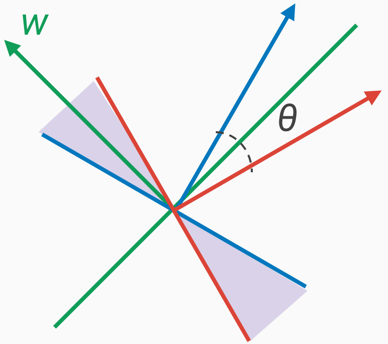
LSH for cosine similarity
[Charikar 2002]

- What if we want to compare vectors u, $v \in \mathbf{R}^d$ by cosine similarity?

$$\mathrm{sim}(u, v) = \cos(\theta)$$

- Pick a vector w uniformly from the sphere in $\mathbf{R}^d$
  - $h_w(x) = 1$ if $w^T x >= 0$
    $= -1$ if $w^T x < 0$

- What's the probability of collision?
  - $\mathbf{P}[h_w(u) = h_w(v)] = 1 - \mathbf{P}[h_w(u) \neq h_w(v)]$
    $= 1 - \mathbf{P}[w$ in **shaded region**$]$

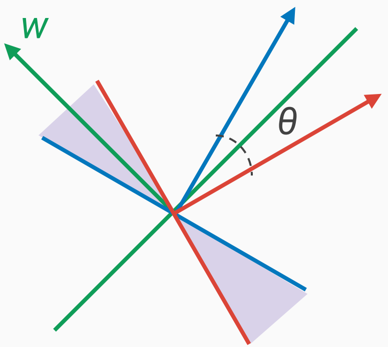


LSH for cosine similarity
[Charikar 2002]

- What if we want to compare vectors $u, v \in \mathbf{R}^d$ by cosine similarity?

$$\text{sim}(u, v) = \cos(\theta)$$

- Pick a vector w uniformly from the sphere in $\mathbf{R}^d$
  - $h_w(x) = 1$ if $w^T x >= 0$
    $= -1$ if $w^T x < 0$



- What's the probability of collision?
  - $\mathbf{P}[h_w(u) = h_w(v)] = 1 - \mathbf{P}[h_w(u) \neq h_w(v)]$
    $= 1 - \mathbf{P}[w$ in **shaded region**$]$
    $= 1 - 2 \cdot |\theta| / 2\pi$
    $= 1 - |\theta|/\pi$

Not exactly cos(θ), but monotonically decreasing with |θ| ⇒ same rank-ordering

Multiple projections

- P[**No collision** | single projection] = θ/π
- P[**All projections do not collide** | *m* projections] = $(\theta/\pi)^m$
- P[At least one **Collision** | *m* projections] = $1 - (\theta/\pi)^m$

Wrap-up

- Similarity search is straightforward, but scale can be overwhelming
- MinHash is simple, but low-precision
- LSH can improve precision, while retaining recall
- This approach generalizes to spatial similarity metrics (beyond sets)

Outlook

- This week:
- Release of HW4
- Release of Capstone project

- Next week:
- Graph search

Q&R