CS4225/CS5425 Big Data Systems for Data Science

MapReduce & Data Mining

Bingsheng He
School of Computing
National University of Singapore
hebs@comp.nus.edu.sg

Learning Objectives

- Learn data mining algorithms including finding similar items and clustering.
- Understand large-scale data mining implementation with MapReduce.

Overview

1. **Similarity Search**
2. Clustering

Source: www.flickr.com/photos/rheinitz/6158837748/

A Common Subroutine

- Many problems can be expressed as finding “similar” objects:
- Examples:
  - **Pages with similar words**
    - For duplicate detection, classification by topic
  - **Customers who purchased similar products**
    - Products with similar customer sets
  - **Users who visited similar websites**
    - Users with similar interests.

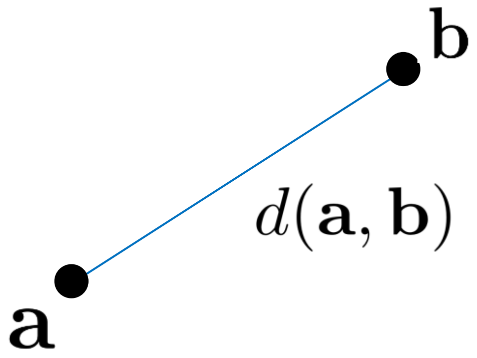
Distance / Similarity Measures

- We define “near neighbors” as points that are a “small distance” apart
- To measure the distance between objects x and y, we need a function d(x, y) which we call a “**distance measure**”
- **Similarity measures** are the opposite: lower distance = higher similarity, and vice versa

Common distance / similarity metrics

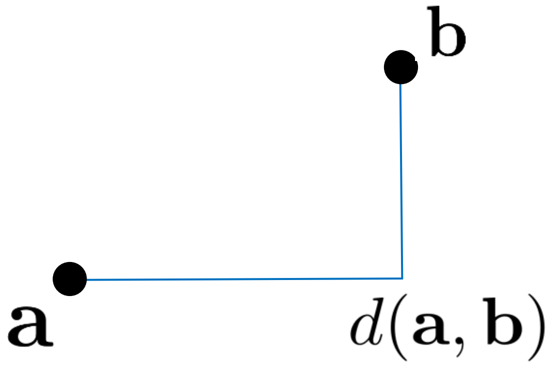
- **Euclidean distance**

$$d(\mathbf{a},\mathbf{b}) = \|\mathbf{a}-\mathbf{b}\|_2 = \sqrt{\sum_{i=1}^{D}(a_i - b_i)^2}$$

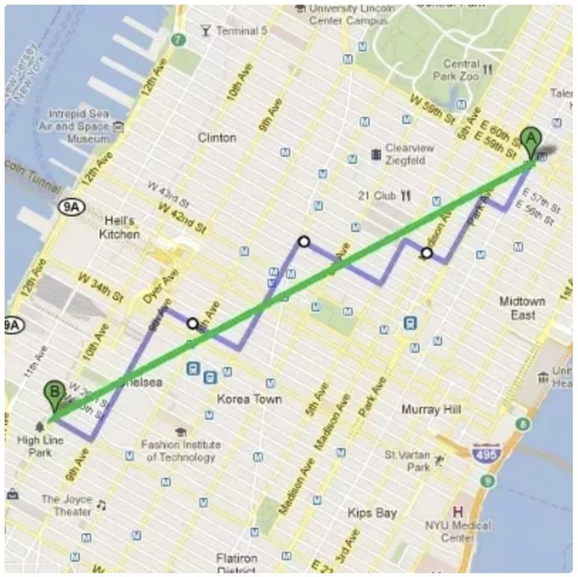


Common distance / similarity metrics

- **Manhattan distance**



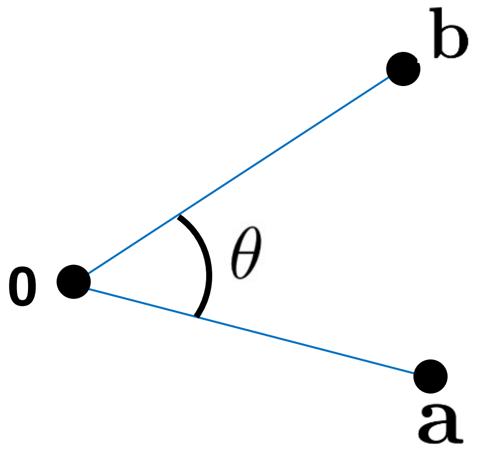
$$d(\mathbf{a}, \mathbf{b}) = \|\mathbf{a} - \mathbf{b}\|_1 = \sum_{i=1}^{D} |a_i - b_i|$$



Common distance / similarity metrics

- **Cosine similarity**

$$s(\mathbf{a}, \mathbf{b}) = \cos\theta = \frac{\mathbf{a} \cdot \mathbf{b}}{\|\mathbf{a}\| \cdot \|\mathbf{b}\|}$$

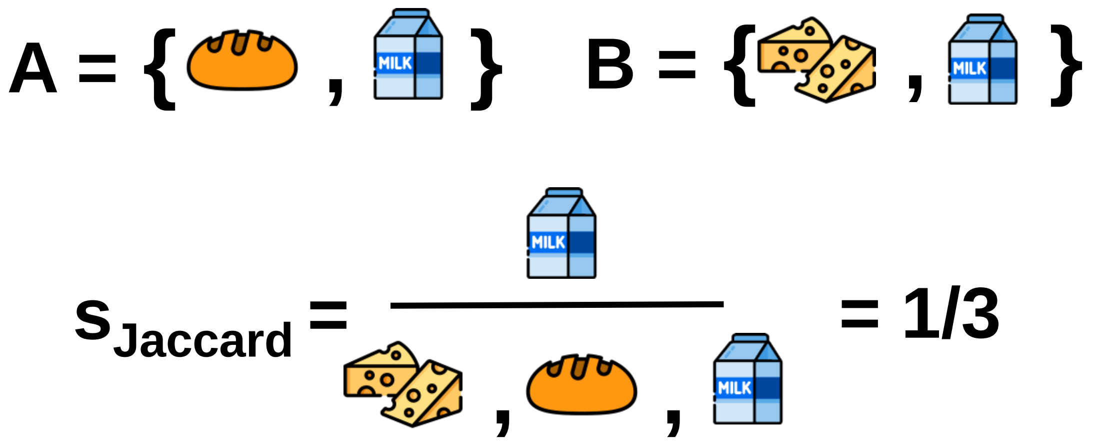


Only considers direction: cosine similarity doesn’t change if we scale a or b (i.e. multiplying them by a constant)

Common distance / similarity metrics

- **Jaccard Similarity**
  (between **sets** A and B)

$$s_{\text{Jaccard}}(A, B) = \frac{|A \cap B|}{|A \cup B|}$$



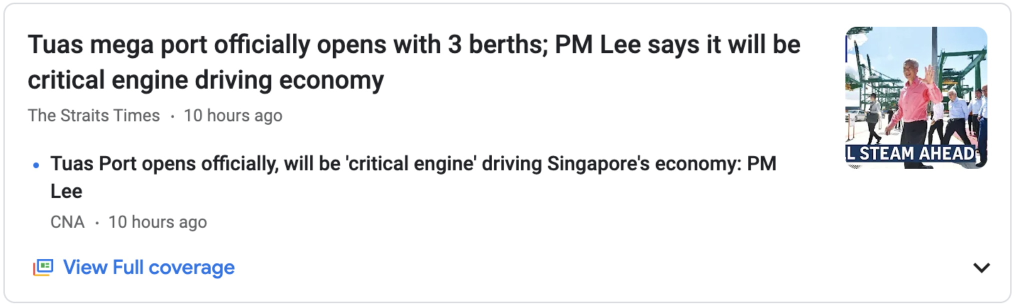
- **Jaccard Distance**

$$d_{\text{Jaccard}}(A, B) = 1 - s_{\text{Jaccard}}(A, B)$$

Task: Finding Similar Documents

- **Goals:**
  - **All pairs similarity:** Given a large number N of documents, find all "near duplicate" pairs, e.g. with Jaccard distance below a threshold
  - **Similarity search:** **Or**: given a query document D, find all documents which are "near duplicates" with D
- **Applications:**
  - Mirror websites, or approximate mirrors
    - Don't want to show both in search results
  - Similar news articles
    - Cluster articles by "same story"

In order to show you the most relevant results, we have omitted some entries very similar to the 101 already displayed.
If you like, you can repeat the search with the omitted results included.



Tuas mega port officially opens with 3 berths; PM Lee says it will be critical engine driving economy
The Straits Times · 10 hours ago

- **Tuas Port opens officially, will be 'critical engine' driving Singapore's economy: PM Lee**
  CNA · 10 hours ago

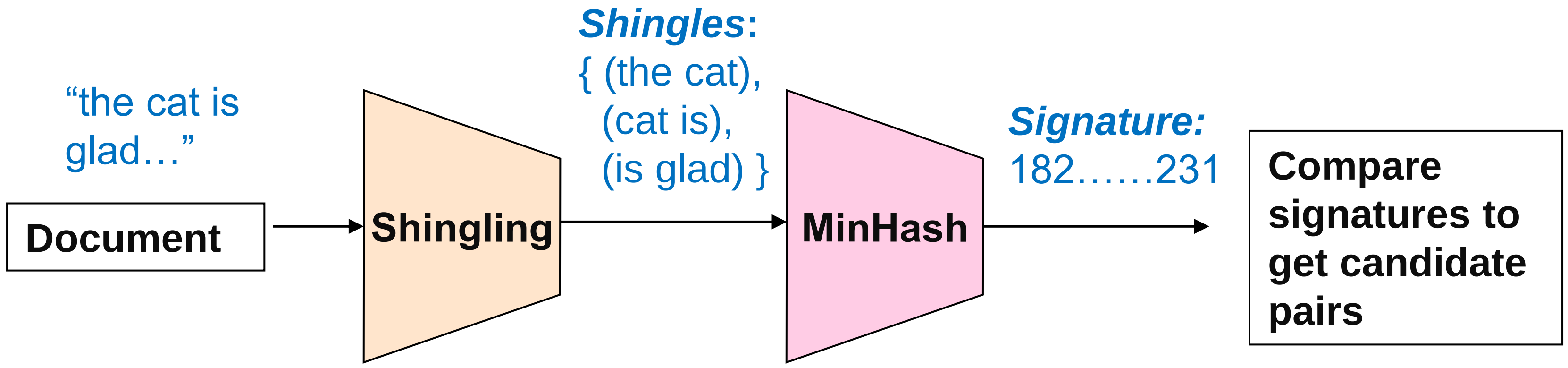
View Full coverage

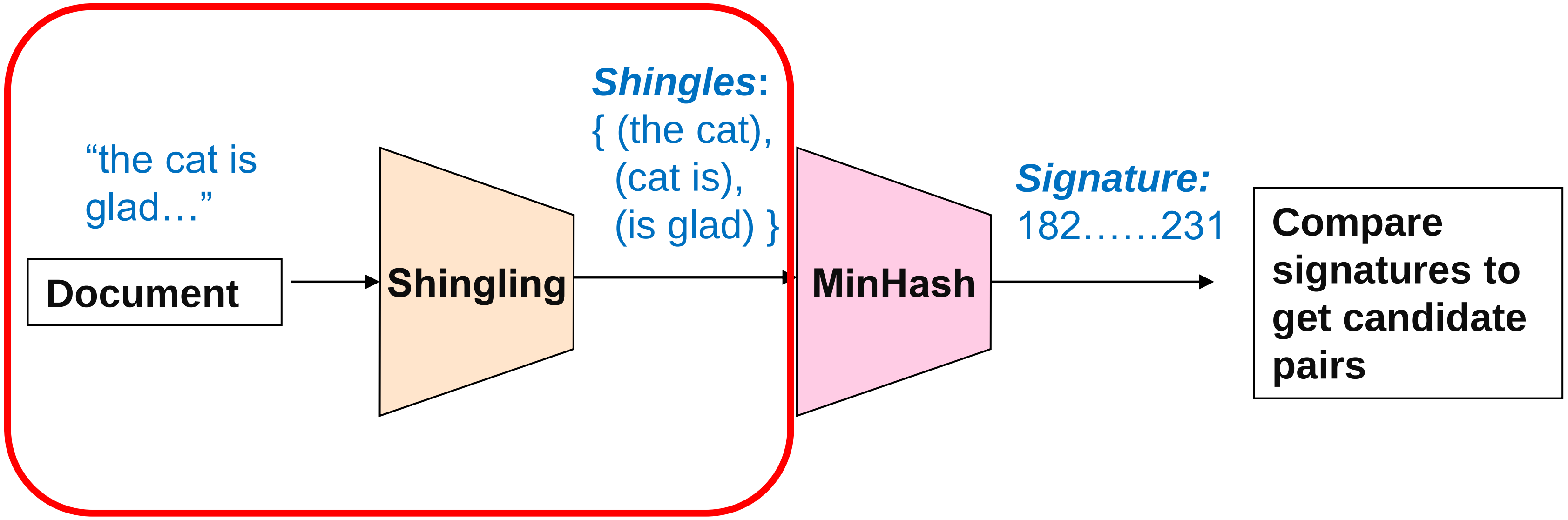
2 Essential Steps for Similar Docs

1. ***Shingling:*** Convert documents to sets of short phrases ("shingles")

2. ***Min-Hashing:*** Convert these sets to short "signatures" of each document, while preserving similarity
   - A "signature" is just a block of data representing the contents of a document in a compressed way
   - Documents with the same signature are candidate pairs

The Big Picture





Shingling

Step 1: ***Shingling:*** Convert documents to sets of phrases

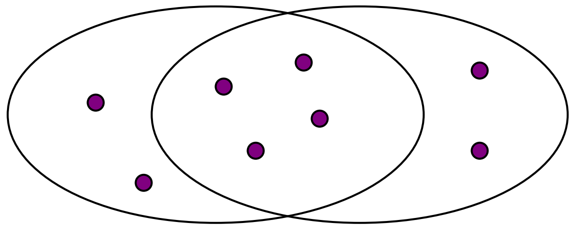
Define: Shingles

- A *k*-shingle (or *k*-gram) for a document is a sequence of *k* tokens that appears in the doc
- **Example: k=2**; document **D_1** = “the cat is glad”
  Set of 2-shingles: **S(D_1)** = {“the cat”, “cat is”, “is glad”}

Similarity Metric for Shingles

- Each document D_1 can be thought of as a set of its k-shingles C_1
  - E.g. D = "the cat is" ⇒ C = {'the cat', 'cat is'}
- Often represented as a matrix, where columns represent documents, and shingles represent rows
- We measure similarity between documents as **Jaccard similarity**:

$$sim(D_1, D_2) = |C_1 \cap C_2| / |C_1 \cup C_2|$$



Documents

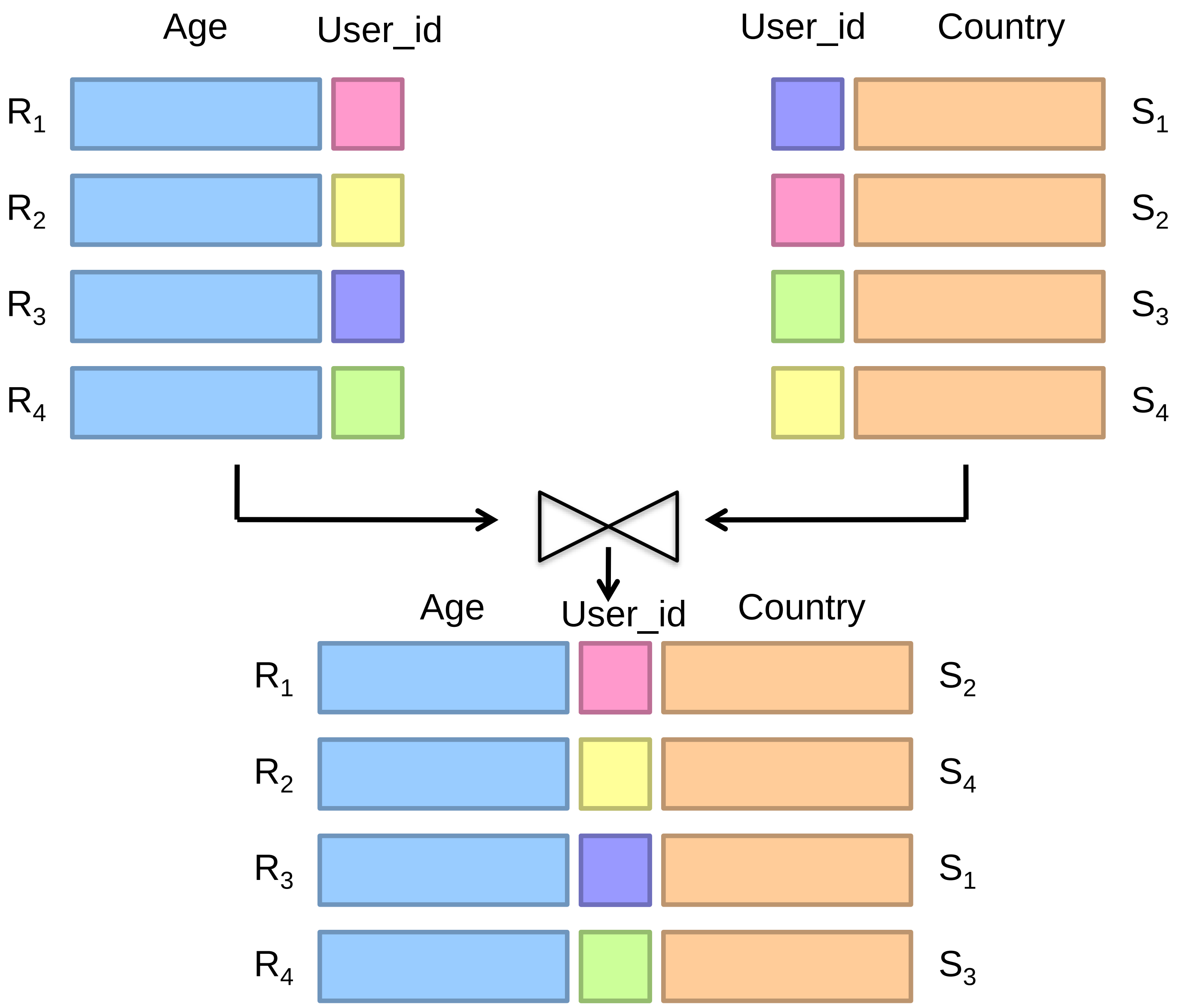
Shingles

$$\begin{array}{r|c|c|c|c|}
 & D_1 & D_2 & \dots & \\
\text{"the cat"} & 1 & 1 & 1 & 0 \\
\text{"cat is"} & 1 & 1 & 0 & 1 \\
\dots & 0 & 1 & 0 & 1 \\
 & 0 & 0 & 0 & 1 \\
 & 1 & 0 & 0 & 1 \\
 & 1 & 1 & 1 & 0 \\
 & 1 & 0 & 1 & 0
\end{array}$$

Announcement

- Midterm seating plan is already uploaded Canvas.
  - Canvas > Files > MidtermMatters
- Past year test paper
  - Canvas > Files > PastYearTestPaper
  - The past year test paper was for the entire semester of the past year. Please attempt only relevant questions in mid-term.
- More consultation hours
  - March 9/16 Thursday 2-3pm in my office (COM3-02-12).
  - Or, post your questions in forum/email me (try to avoid 6pm March 17 afterwards till March 18)

Recap: Relational Joins ('Inner Join')



Recap: An Overview

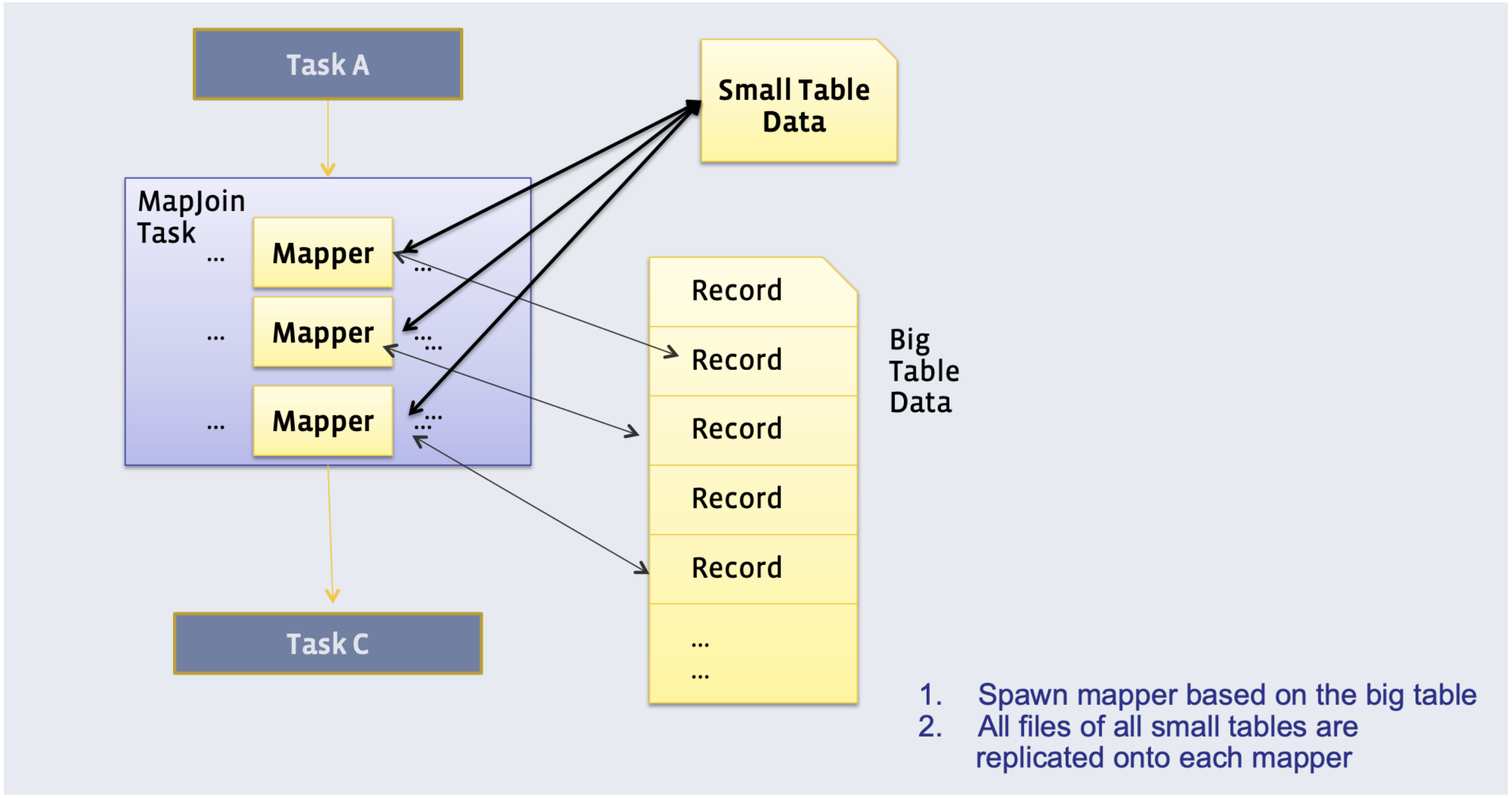
- Join implementations are complex
- "One size does not fit all"
- We mainly talk about two methods.
  - If (one of the input tables is **small**):

    Choose method 1: Broadcast Join
  - Else:

    Choose method 2: Reduce-side Join

Recap: Method 1: Broadcast Join

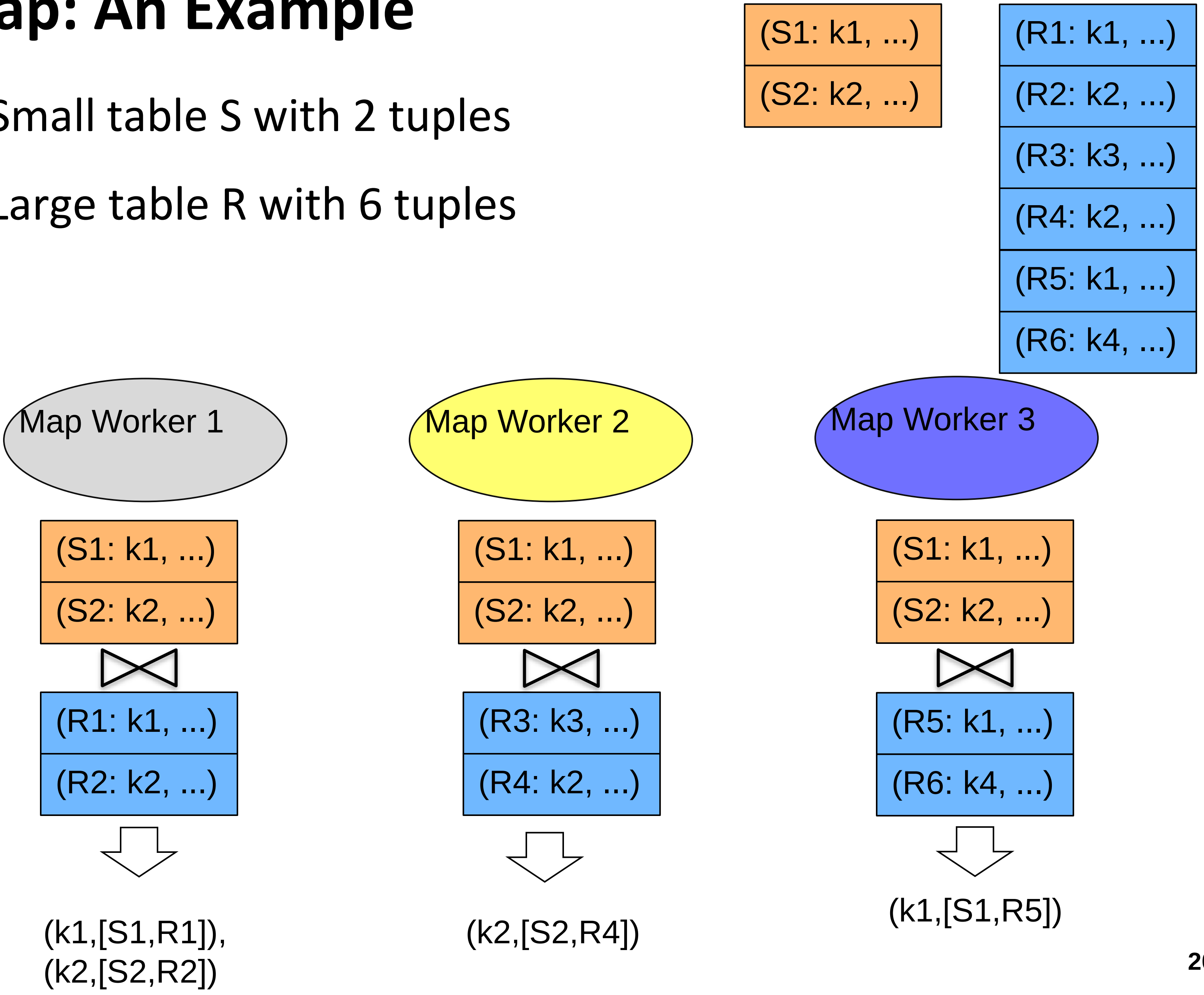
- Requires one of the tables to fit in main memory of individual servers
  - All mappers store a copy of the small table
  - They iterate over the big table, and join the records with the small table



1. Spawn mapper based on the big table
2. All files of all small tables are replicated onto each mapper

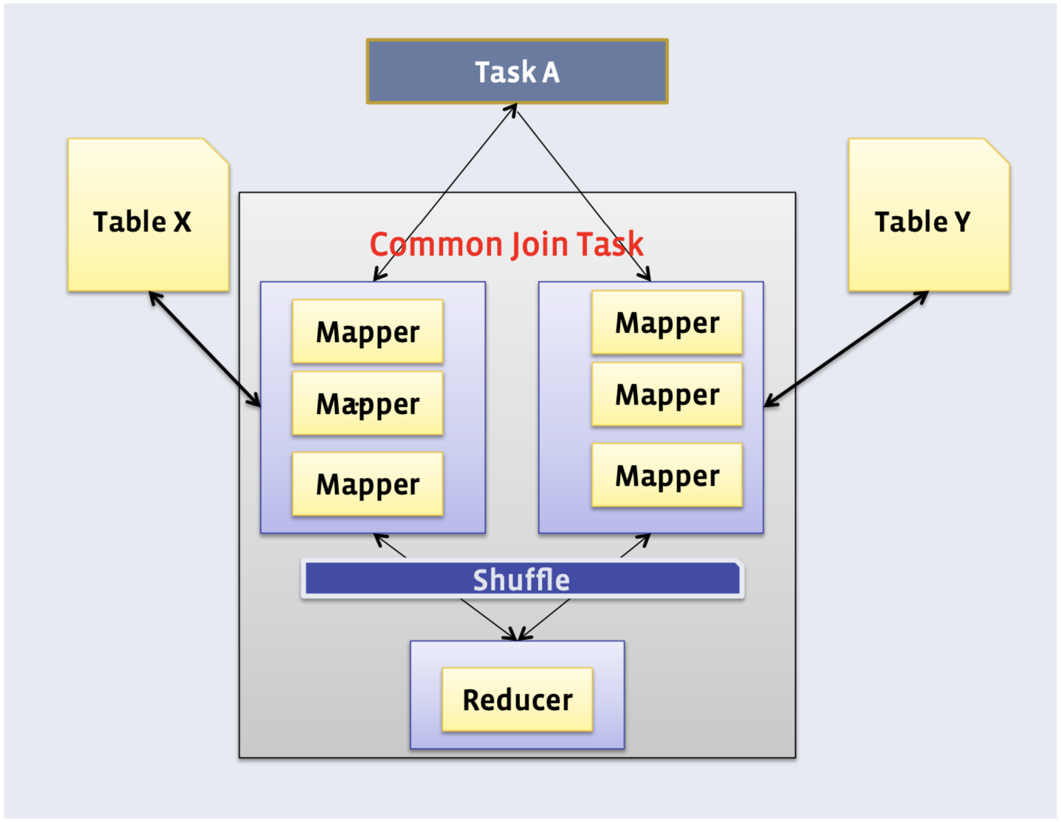
Recap: An Example

- Small table S with 2 tuples
- Large table R with 6 tuples



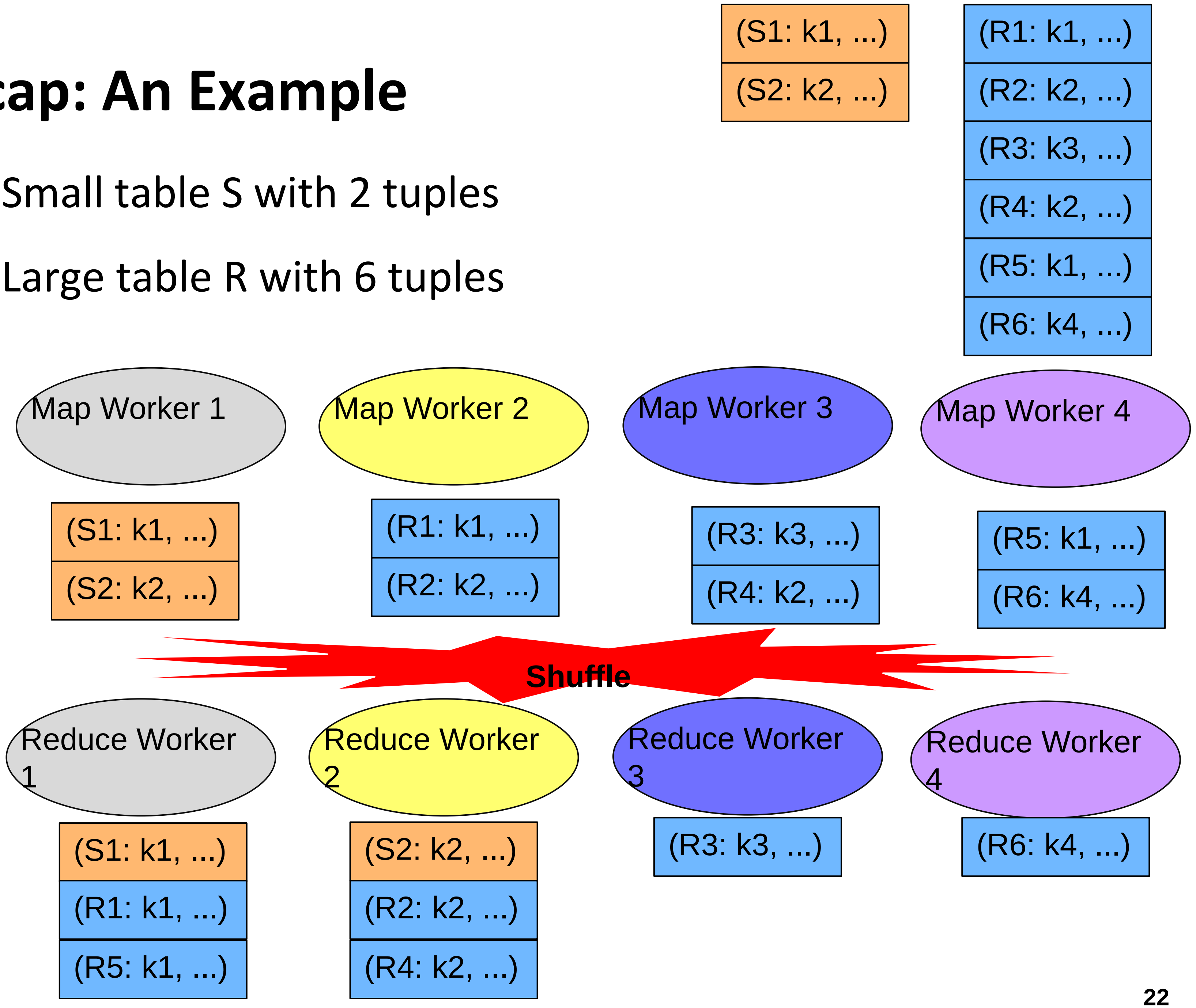
Recap: Method 2: Reduce-side (‘Common’) Join

- Doesn’t require a dataset to fit in memory, but slower than map-side join
  - Different mappers operate on each table, and emit records, with key as the variable to join by

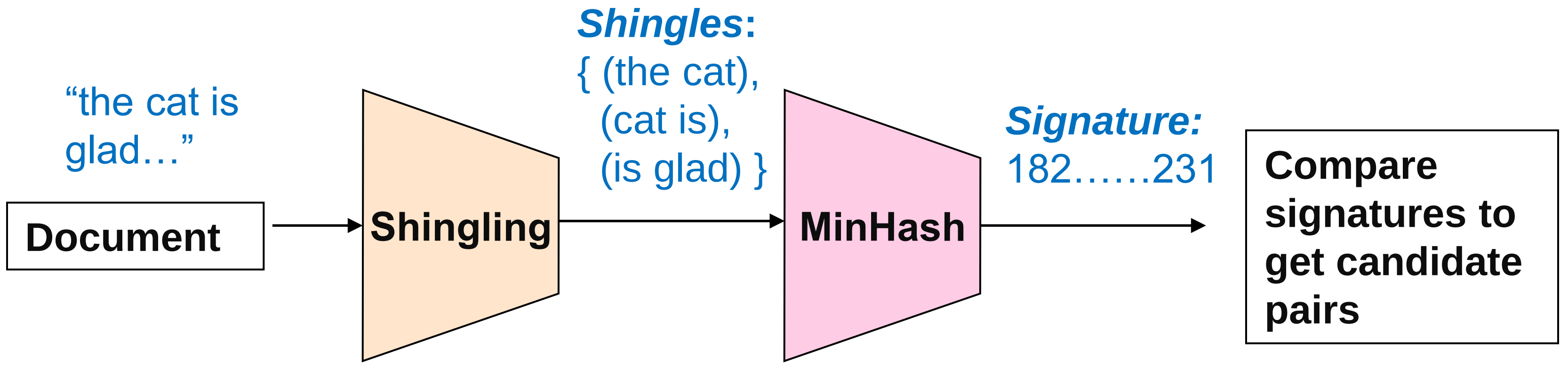


Recap: An Example

- Small table S with 2 tuples
- Large table R with 6 tuples



Recap: The Big Picture


"the cat is glad…"
Document
Shingling
Shingles: { (the cat), (cat is), (is glad) }
MinHash
Signature: 182……231
Compare signatures to get candidate pairs

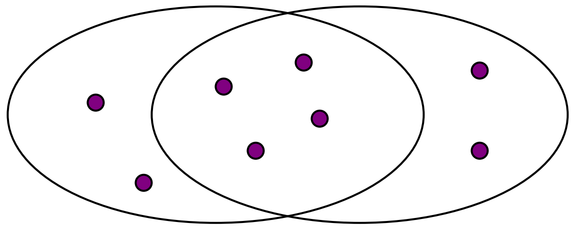
Recap: Define: Shingles

- A *k*-shingle (or *k*-gram) for a document is a sequence of *k* tokens that appears in the doc
- **Example: k=2**; document $\mathbf{D_1}$ = “the cat is glad”
  Set of 2-shingles: $\mathbf{S(D_1)}$ = {“the cat”, “cat is”, “is glad”}

Recap: Similarity Metric for Shingles

- Each document D_1 can be thought of as a set of its k-shingles C_1
  - E.g. D = “the cat is” ⇒ C = {’the cat’, ‘cat is’}
- Often represented as a matrix, where columns represent documents, and shingles represent rows
- We measure similarity between documents as **Jaccard similarity**:

$$sim(D_1, D_2) = |C_1 \cap C_2| / |C_1 \cup C_2|$$



Documents

Shingles

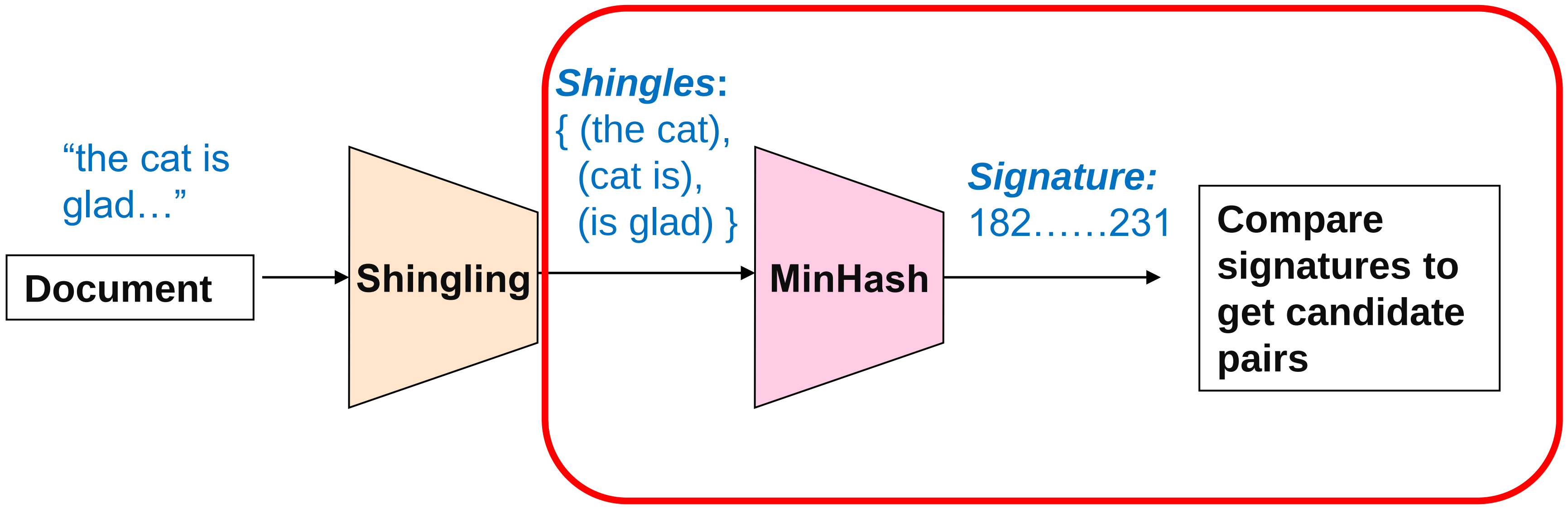
$$\begin{array}{c|cccc}
 & D_1 & D_2 & \dots & \\
\hline
\text{“the cat”} & 1 & 1 & 1 & 0 \\
\text{“cat is”} & 1 & 1 & 0 & 1 \\
\dots & 0 & 1 & 0 & 1 \\
 & 0 & 0 & 0 & 1 \\
 & 1 & 0 & 0 & 1 \\
 & 1 & 1 & 1 & 0 \\
 & 1 & 0 & 1 & 0
\end{array}$$

Motivation for MinHash

- Suppose we have $N = 1$ million documents

- Naively, we would have to compute **pairwise Jaccard similarities** for **every pair of docs**
  - $\boldsymbol{N(N-1)/2}$ ≈ $5*10^{11}$ comparisons: too slow!

- MinHash gives us a fast approximation to the result of using Jaccard similarities to compare all pairs of documents


"the cat is glad…"
Document
Shingling
Shingles: { (the cat), (cat is), (is glad) }
MinHash
Signature: 182……231
Compare signatures to get candidate pairs

MinHashing

Step 2: ***MinHash:*** Convert **large sets** to **short signatures**, while **<u>preserving similarity</u>**

MinHash: Overview

- **Key idea:** hash each column ***C*** to a small ***signature*** ***h(C)***, such that:
  - **(1)** ***h(C)*** is small enough that the signature fits in RAM
  - **(2)** highly similar documents usually have the same signature
- **Goal: Find a hash function *h(·)* such that:**
  - If $sim(C_1,C_2)$ is high, then with high probability, $h(C_1) = h(C_2)$
  - If $sim(C_1,C_2)$ is low, then with high probability, $h(C_1) \neq h(C_2)$

MinHash: Steps

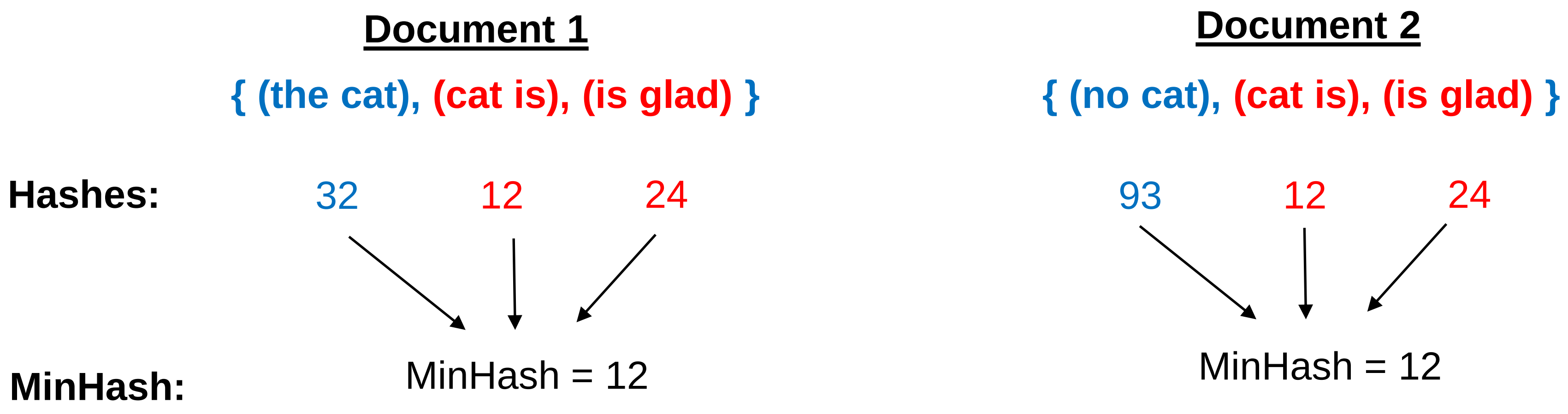
- Given a set of shingles, { (the cat), (cat is), (is glad) }

1. We have a **hash function** h that maps each shingle to an integer:

   h(“the cat”) = 12, h(“cat is”) = 74, h(“is glad”) = 48

2. Then compute the **minimum** of these: min(12, 74, 48) = 12

Example: MinHash on 2 sets of shingles



- Recall that we want to ensure that highly similar documents have high probability to have the same MinHash signature

Key Property of MinHash

- **Key property**: the probability that two documents have the same MinHash signature is equal to their Jaccard similarity!
- Formally: $\mathbf{Pr}[h(C_1) = h(C_2)] = \textit{Jaccard-Sim}(C_1, C_2)$
- **Proof** (not required knowledge):

Document 1

{ (the cat), (cat is), (is glad) }

Document 2

{ (no cat), (cat is), (is glad) }

Hashes:

$$\begin{tikzcd}
32 \arrow[dr] & 12 \arrow[d] & 24 \arrow[dl] \\
 & \text{MinHash} = 12 &
\end{tikzcd}$$

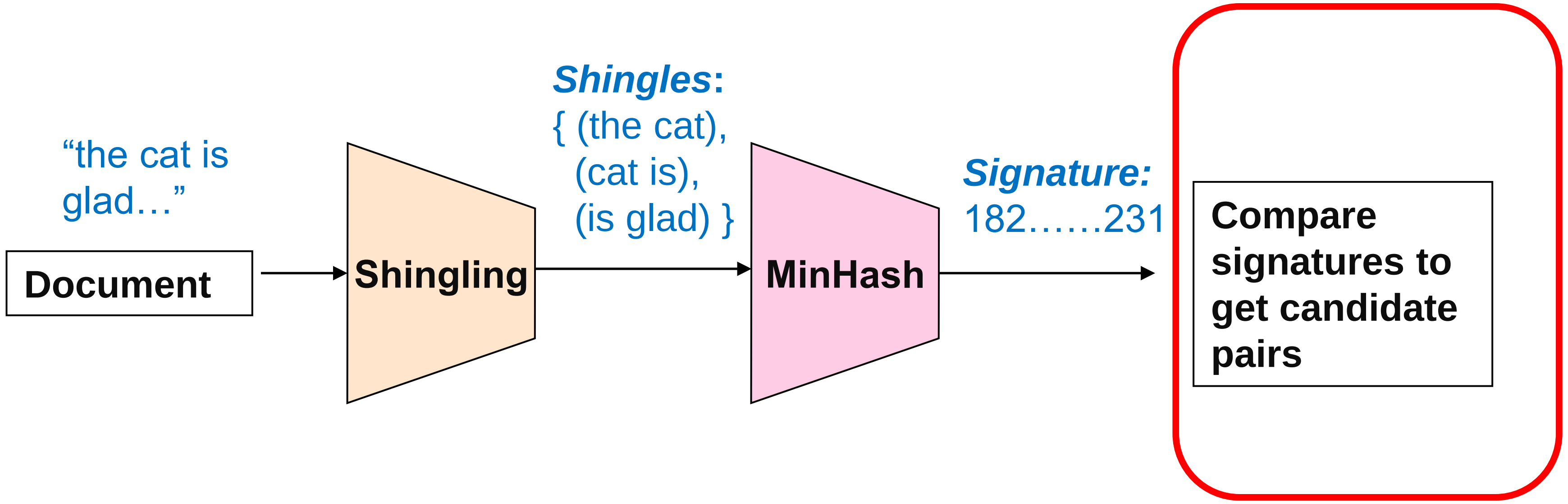
$$\begin{tikzcd}
93 \arrow[dr] & 12 \arrow[d] & 24 \arrow[dl] \\
 & \text{MinHash} = 12 &
\end{tikzcd}$$

Set of all our tuples: $\{\underbrace{\text{(the cat), (no cat)}}_{\text{(2 non-shared shingles)}}, \underbrace{\text{(cat is), (is glad)}}_{\text{(2 shared shingles)}}\}$

Jaccard sim = 2/4 = 0.5

Among these 4 tuples, each has the same probability of having the smallest hash value.
if one of the red shingles has the smallest hash, the documents will have the same MinHash.
Otherwise, they will have different MinHashes.
=> $\mathbf{Pr}[h(C_1) = h(C_2)]$ = (red shingles / total shingles) = (intersection / union) = Jaccard similarity

Putting it all together



"the cat is glad…"
Document
Shingling
Shingles: { (the cat), (cat is), (is glad) }
MinHash
Signature: 182……231
Compare signatures to get candidate pairs

- **Candidate pairs**: the documents with the same final signature are "candidate pairs". We can either directly use them as our final output, or compare them one by one to check if they are actually similar pairs.
- **Extension to multiple hashes:** in practice, we usually use multiple hash functions (e.g. N=100), and generate N signatures for each document. "Candidate pairs" can be defined as those matching a 'sufficient number' (e.g. at least 50) among these signatures.

One Example

- We have three statistically independent hash functions: H1, H2, H3.
- If we define 'sufficient number' of matchings for candidate pairs to be 2. What are the candidate pairs?
  - (D1, D2): YES, matchings on H1 and H3
  - (D3, D4): NO, matching only on H1
  - (D1, D3), NO, there is no matching.

	D1	D2	D3	D4
H1	10	10	32	32
H2	20	25	25	20
H3	23	23	15	25

MapReduce Implementation

- Map
  - Read over the document and extract its shingles
  - Hash each shingle and take the min of them to get the MinHash signature
  - Emit <signature, document_id>
- Reduce
  - Receive all documents with a given MinHash signature
  - Generate all candidate pairs from these documents
  - (Optional): compare each such pair to check if they are actually similar

Overview

1. Similarity Search
2. **Clustering**

Source: www.flickr.com/photos/rheinitz/6158837748/

SFGIANTS.COM/VO
at&t
at&t
FRA
TEMI

mariners.com
T · Mobile
Mobile
IWAKUMA 33 PITCHES
TB 0
SEA 1 0-2
PITCHER GIFS

FOR TICKETS VISIT WHITESOX.COM
OD Household Services
KIDS SWEATSHIRT JUNE 6
KLUBER 15 PITCHES
CLE 0
CHW 0 2-2
PITCHER LIST

KIA 4 5 삼성
삼성카드 삼성증권 삼성생명
SPOTV
KCW
대구은행
치킨더홈
땅땅치킨

FOR TICKETS VISIT WHITESOX.COM
U.S. Cellular
CROSSTOWN CUP MAY 7-8 TICKETS AVAILABLE
SCHERZER 24 PITCHES
DET 0
CHW 0 0-2
PITCHER GIFS

SFGIANTS.COM/VOT
at&t
FRA
TEMI

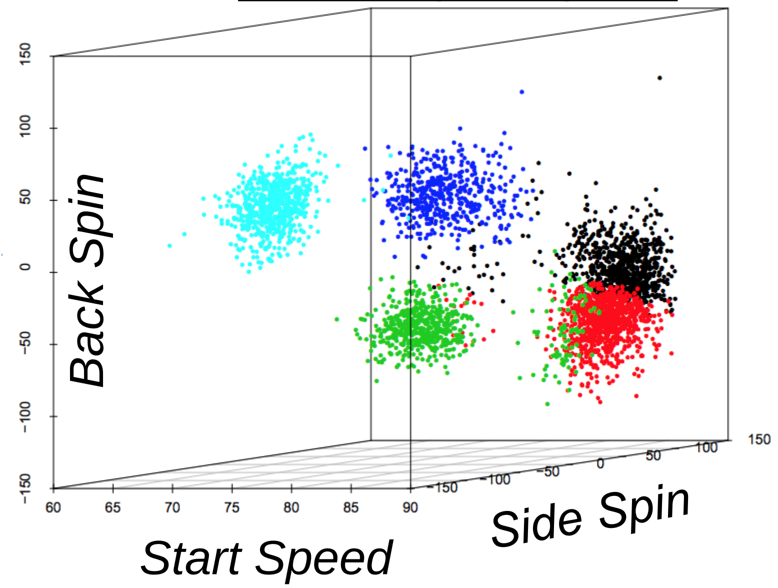
FOR TICKETS VISIT WHITESOX.COM
OD Household Services
KIDS SWEATSHIRT JUNE 6
KLUBER 15 PITCHES
CLE 0
CHW 0 2-2
PITCHER LIST

KIA 4 5 삼성
SPOTV
대구은행
치킨더홈

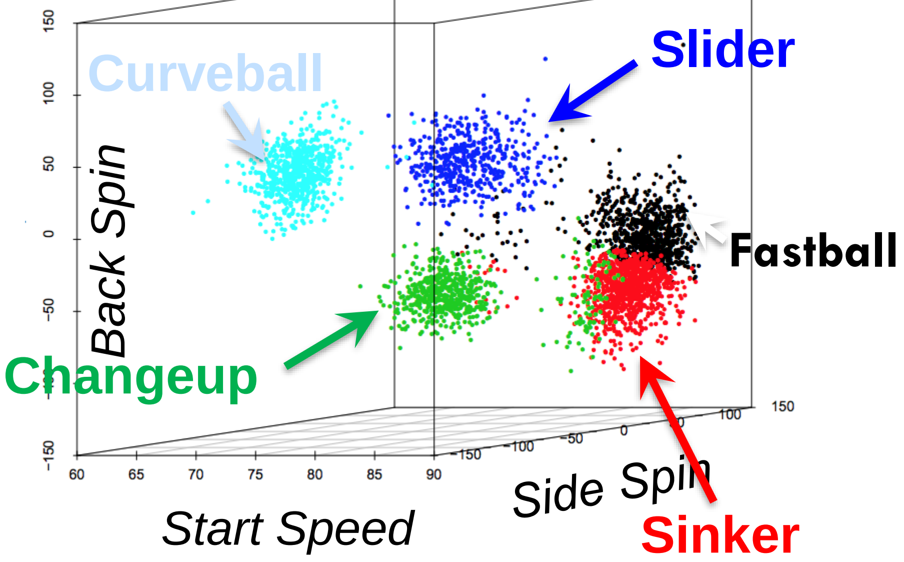
mariners.com
T-Mobile
IWAKUMA 33 PITCHES
TB 0
SEA 1 0-2
PITCHER GIFS

FOR TICKETS VISIT WHITESOX.COM
U.S. Cellular
CROSSTOWN CUP MAY 7-8 TICKETS AVAILABLE
SCHERZER 24 PITCHES
DET 0
CHW 0 0-2
PITCHER GIFS

Pitches by Barry Zito



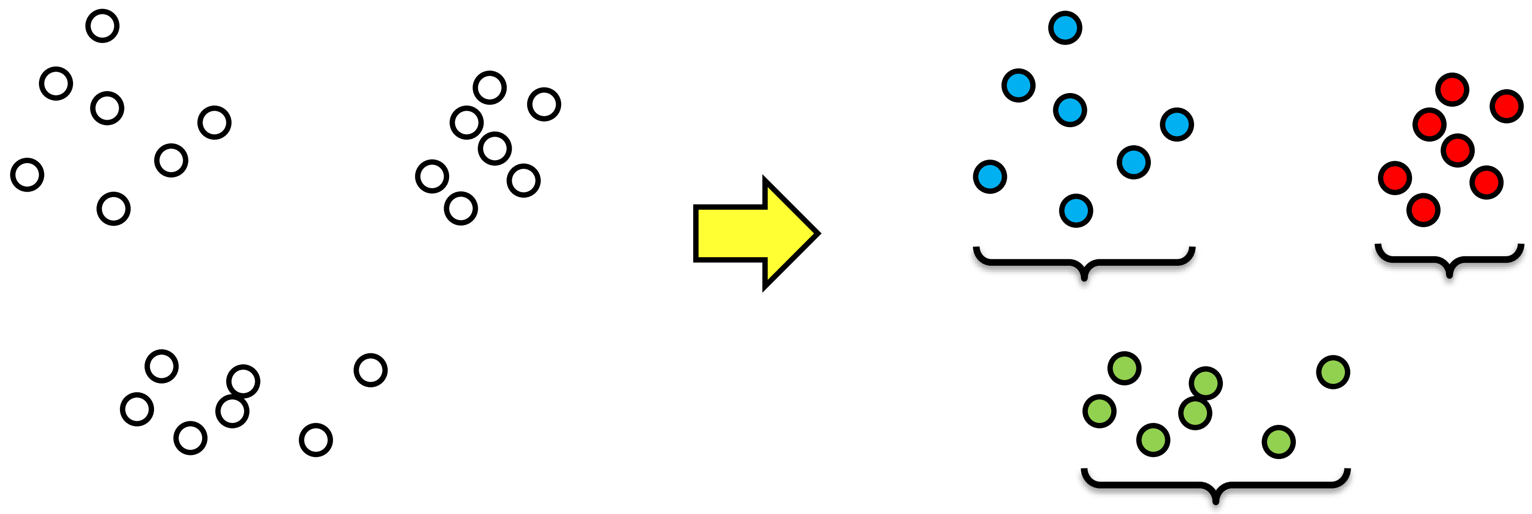
Pitches by Barry Zito



Goal of clustering

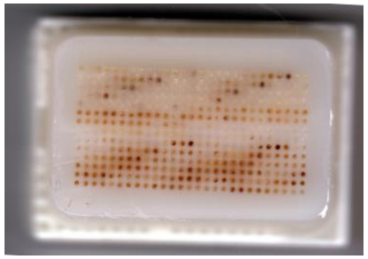
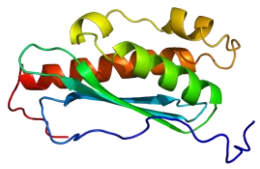
Clustering separates unlabelled data into groups of similar points.

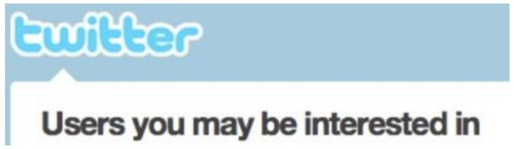
Clusters should have high intra-cluster similarity, and low inter-cluster similarity.



Applications of clustering

Many applications:

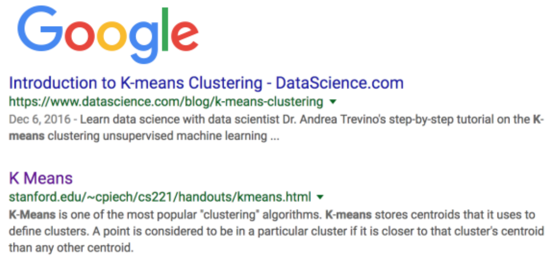


Users you may be interested in

Trump, North Korea's Kim to hold second summit in late February
Channel NewsAsia · today

Trump to hold second summit with Kim Jong Un in February
The Straits Times · today

View more



Introduction to K-means Clustering - DataScience.com
https://www.datascience.com/blog/k-means-clustering
Dec 6, 2016 - Learn data science with data scientist Dr. Andrea Trevino's step-by-step tutorial on the K-means clustering unsupervised machine learning ...

K Means
stanford.edu/~cpiech/cs221/handouts/kmeans.html
K-Means is one of the most popular "clustering" algorithms. K-means stores centroids that it uses to define clusters. A point is considered to be in a particular cluster if it is closer to that cluster's centroid than any other centroid.

Microbiology: find groups of related genes (or proteins etc.)

Recommendation & Social Networks: find groups of similar users

Search & Information Retrieval: grouping similar search (or news etc.) results

K-Means Algorithm: Steps

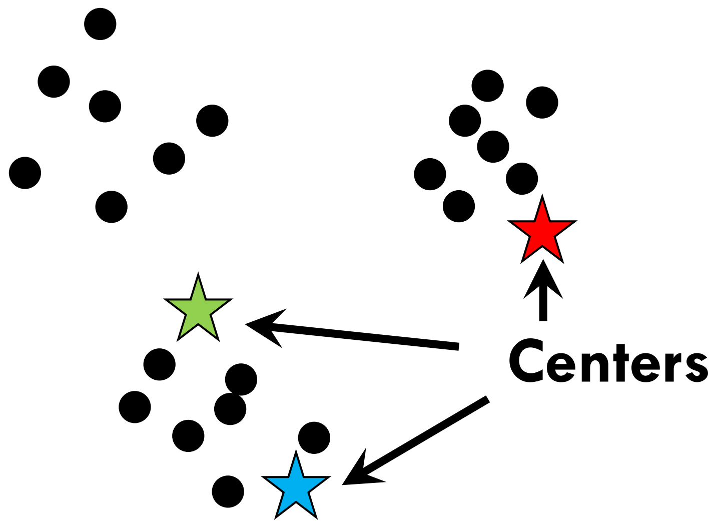
1. Initialization: Pick K random points as centers

2. Repeat:

a) **Assignment:** assign each point to nearest cluster

b) **Update:** move each cluster center to average of its assigned points

Stop if no assignments change



K-Means Algorithm: Steps

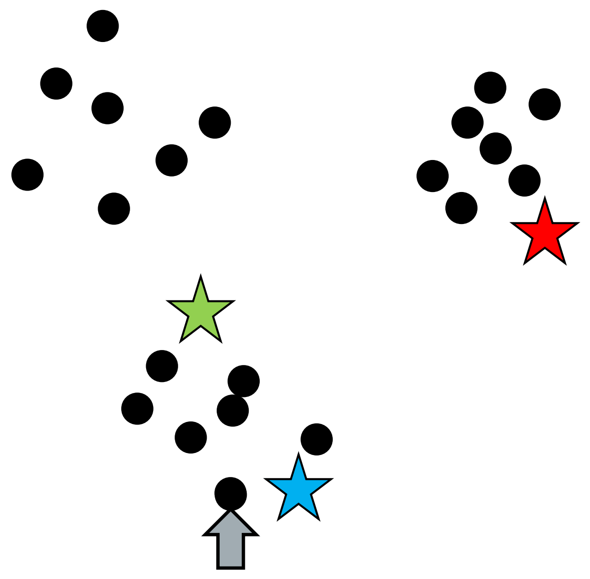
1. **Initialization**: Pick K random points as centers

2. Repeat:

a) **Assignment:** assign each point to nearest cluster

b) **Update:** move each cluster center to average of its assigned points

Stop if no assignments change



K-Means Algorithm: Steps

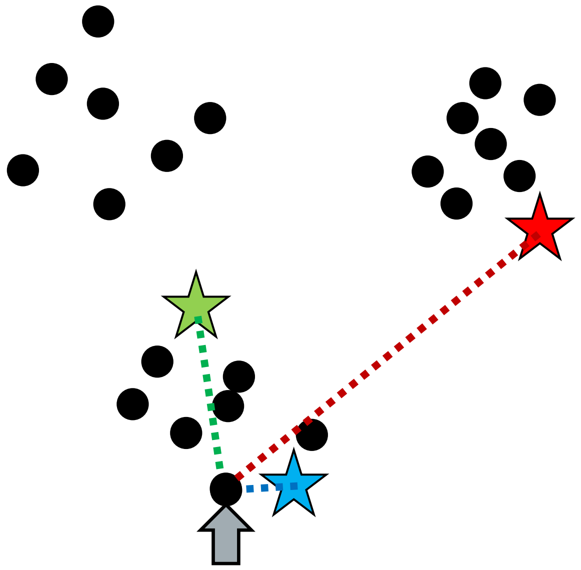
<u>**1. Initialization**</u>: Pick K random points as centers

2. Repeat:

a) **Assignment:** assign each point to nearest cluster

b) **Update:** move each cluster center to average of its assigned points

Stop if no assignments change



K-Means Algorithm: Steps

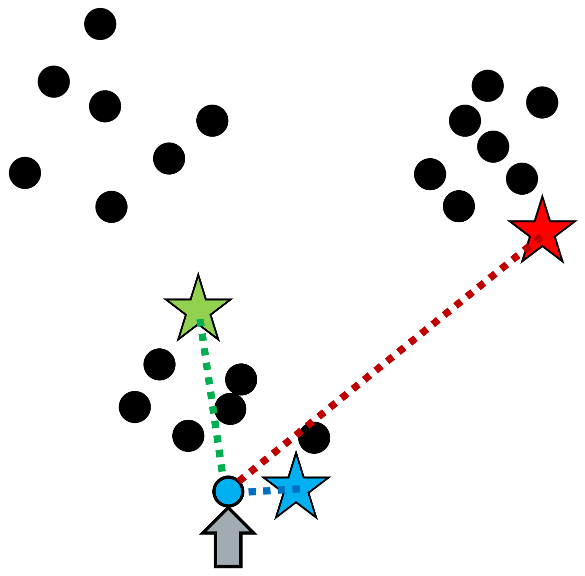
1. **Initialization**: Pick K random points as centers

2. Repeat:

   a) **Assignment:** assign each point to nearest cluster

   b) **Update:** move each cluster center to average of its assigned points

   Stop if no assignments change



K-Means Algorithm: Steps

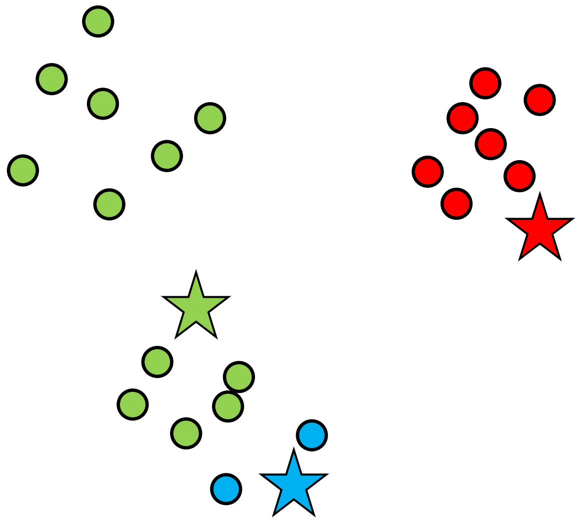
1. **Initialization**: Pick K random points as centers

2. Repeat:

a) **Assignment:** assign each point to nearest cluster

b) **Update:** move each cluster center to average of its assigned points

Stop if no assignments change



K-Means Algorithm: Steps

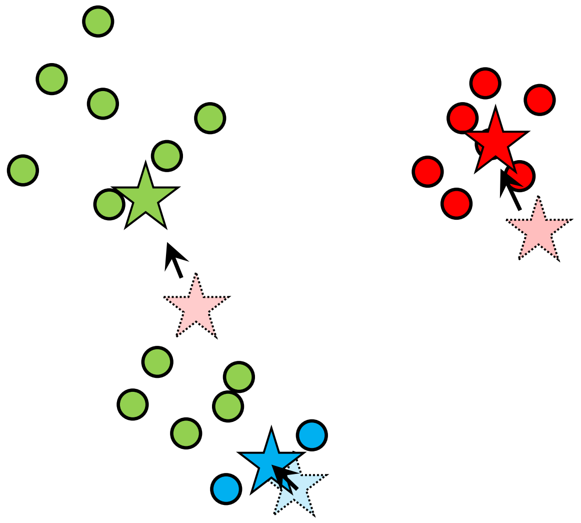
1. **Initialization**: Pick K random points as centers

2. Repeat:

a) **Assignment:** assign each point to nearest cluster

b) Update: move each cluster center to average of its assigned points

Stop if no assignments change



K-Means Algorithm: Steps

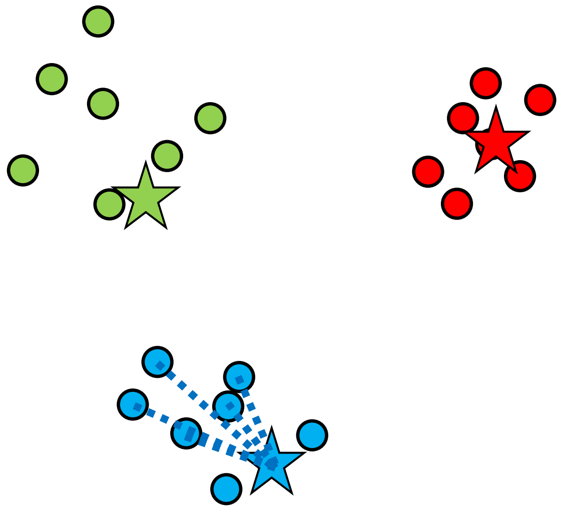
1. **Initialization**: Pick K random points as centers

2. **Repeat:**

    a) **Assignment:** assign each point to nearest cluster

    b) **Update:** move each cluster center to average of its assigned points

    Stop if no assignments change



K-Means Algorithm: Steps

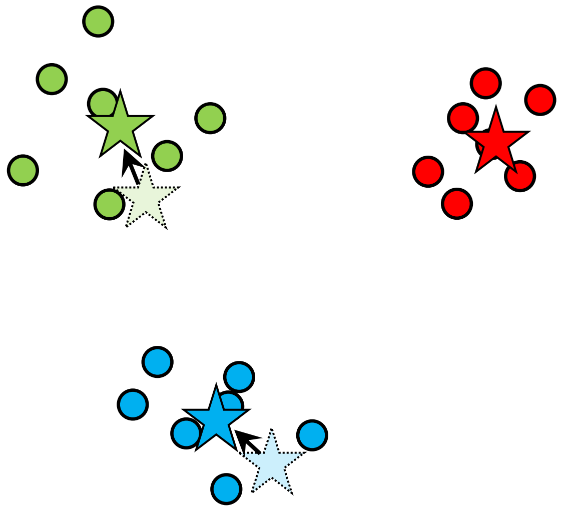
1. Initialization: Pick K random points as centers

2. Repeat:

a) **Assignment:** assign each point to nearest cluster

b) **Update:** move each cluster center to average of its assigned points

Stop if no assignments change



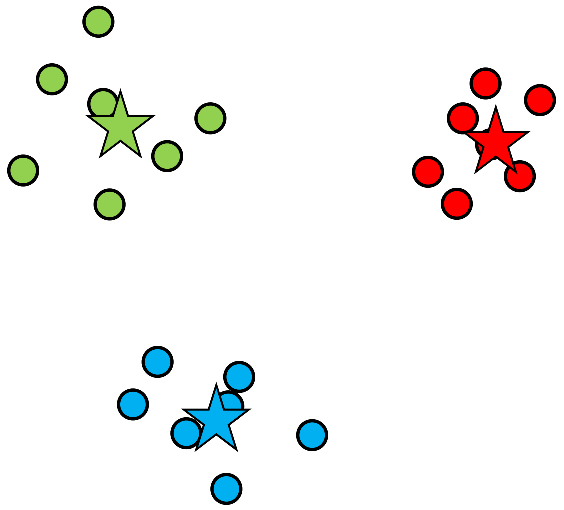
K-Means Algorithm: Steps

1. Initialization: Pick K random points as centers

2. Repeat:

- **a) Assignment:** assign each point to nearest cluster
- **b) Update:** move each cluster center to average of its assigned points

Stop if no assignments change



K-Means Algorithm: Steps

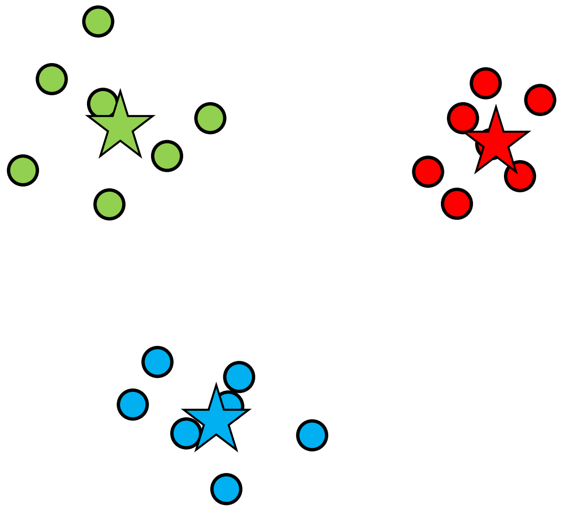
1. Initialization: Pick K random points as centers

2. Repeat:

a) **Assignment:** assign each point to nearest cluster

b) **Update:** move each cluster center to average of its assigned points

Stop if no assignments change



MapReduce Implementation v1

This MapReduce job performs a **single iteration** of k-means. It receives as input the current positions of the cluster centers (i.e. stars)

```
1: class Mapper
2:     method Configure()
3:         $c \leftarrow$ LoadClusters()
4:     method Map(id $i$, point $p$)
5:         $n \leftarrow$ NearestClusterID(clusters $c$, point $p$)
6:         $p \leftarrow$ ExtendPoint(point $p$)
7:         Emit(clusterid $n$, point $p$)
1: class Reducer
2:     method Reduce(clusterid $n$, points $[p_1, p_2, \ldots]$)
3:         $s \leftarrow$ InitPointSum()
4:         for all point $p \in$ points do
5:             $s \leftarrow s + p$
6:         $m \leftarrow$ ComputeCentroid(point $s$)
7:         Emit(clusterid $n$, centroid $m$)
```

Network traffic!

What is the problem?

Q: How much disk I/O is exchanged between the mappers and reducers, in MapReduce K-means v1?

(let n = no. of points, m = no. of iterations, d = dimensionality, k = no. of centers)

1. O(n)
2. O(md)
3. O(nmd)
4. O(kmd)

Q: How much disk I/O is exchanged between the mappers and reducers, in MapReduce K-means v1?

(let n = no. of points, m = no. of iterations, d = dimensionality, k = no. of centers)

1. O(n)
2. O(md)
3. [✓ O(nmd)]
4. O(kmd)

MapReduce Implementation v2 (with in-mapper combiner)

```
 1: class Mapper
 2:    method Configure()
 3:       $c \leftarrow$ LoadClusters()
 4:       $H \leftarrow$ InitAssociativeArray()
 5:    method Map(id $i$, point $p$)
 6:       $n \leftarrow$ NearestClusterID(clusters $c$, point $p$)
 7:       $p \leftarrow$ ExtendPoint(point $p$)
 8:       $H\{n\} \leftarrow H\{n\} + p$
 9:    method Close()
10:       for all clusterid $n \in H$ do
11:          Emit(clusterid $n$, point $H\{n\}$)
 1: class Reducer
 2:    method Reduce(clusterid $n$, points $[p_1, p_2, \ldots]$)
 3:       $s \leftarrow$ InitPointSum()
 4:       for all point $p \in$ points do
 5:          $s \leftarrow s + p$
 6:       $m \leftarrow$ ComputeCentroid(point $s$)
 7:       Emit(clusterid $n$, centroid $m$)
```

Q: How much disk I/O is exchanged between the mappers and reducers, in MapReduce K-means v2?

(let n = no. of points, m = no. of iterations, d = dimensionality, k = no. of centers)

1. O(n)
2. O(md)
3. O(nmd)
4. O(kmd)

Q: How much disk I/O is exchanged between the mappers and reducers, in MapReduce K-means v2?

(let n = no. of points, m = no. of iterations, d = dimensionality, k = no. of centers)

1. O(n)
2. O(md)
3. O(nmd)
4. [✓ O(kmd)]

Implementation Notes

- Standard setup of iterative MapReduce algorithms
  - Driver program sets up MapReduce job
  - Waits for completion
  - Checks for convergence
  - Repeats if necessary
- Must be able keep cluster centroids in memory
  - With large *k*, large feature spaces, potentially an issue
  - Memory requirements of centroids grow over time!
- Can we have a better solution?

Further Reading

- Chapter 6, "Data-Intensive Text Processing with MapReduce", by Jimmy Lin. http://lintool.github.io/MapReduceAlgorithms/ed1n/MapReduce-algorithms.pdf

- White et al. Web-scale computer vision using MapReduce for multimedia data mining. In Proceedings of the Tenth International Workshop on Multimedia Data Mining (MDMKDD '10). http://www.umiacs.umd.edu/~lsd/papers/brandyn-kdd-cloud.pdf

- Chu et al. Map-reduce for machine learning on multicore. In Proceedings of the 19th International Conference on Neural Information Processing Systems (NIPS'06). http://papers.nips.cc/paper/3150-map-reduce-for-machine-learning-on-multicore.pdf

Take-away

- MapReduce algorithms for BIG data mining:
  - Finding similar items
  - Clustering
- MapReduce may not be the ideal efficient solution for iterative algorithms.

Questions?

OUR CONSULTANT HAS BEEN MINING DATA ALL DAY.
THE RESULTS ARE QUITE SHOCKING.
ACCORDING TO THE DATA, SALES ARE ALWAYS HIGHEST WHEN I DO THIS...
www.dilbert.com
scottadams@aol.com
© 1999 United Feature Syndicate, Inc.

Acknowledgement

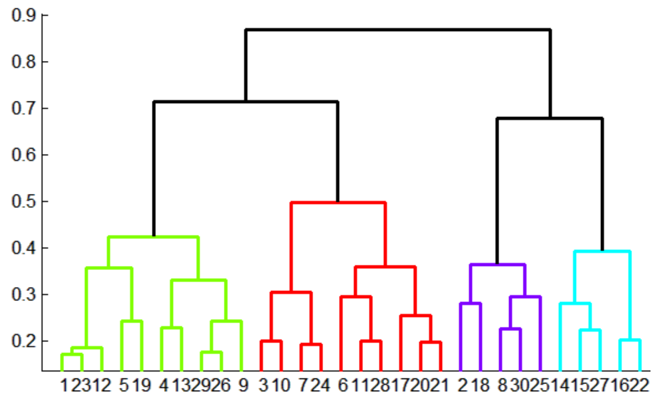
- Some slides are adopted/revised from
  - Jure Leskovec, Anand Rajaraman, and Jeffrey David Ullman. 2014. Mining of Massive Datasets (2nd ed.). Cambridge University Press. http://www.mmds.org/
  - Bryan Hooi
- Some slides are also adopted/revised from
  - Jimmy Lin, http://lintool.github.io/UMD-courses/bigdata-2015-Spring/

Supplementary Slides for Clustering

Hierarchical Clustering

- **Key operation:**
  Repeatedly combine two nearest clusters

```
WHILE it is not time to stop DO
    pick the best two clusters to merge;
    combine those two clusters into one cluster;
END;
```

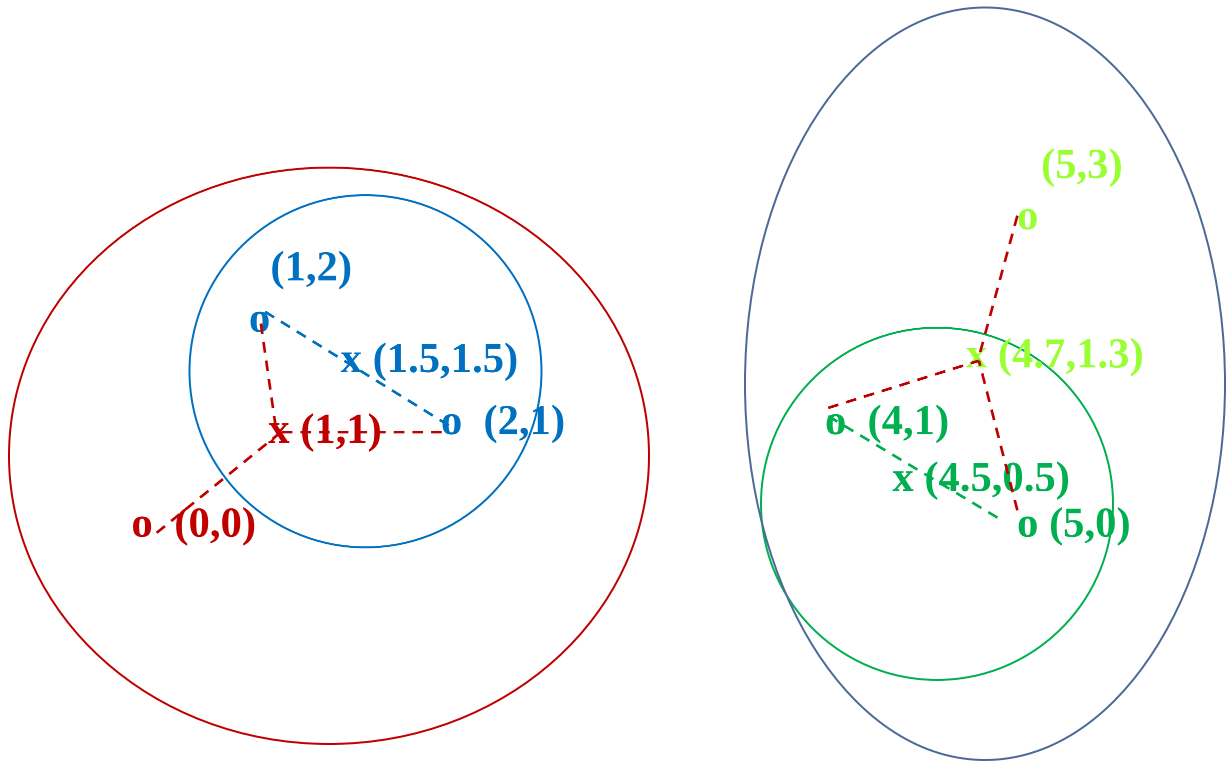


- **Three important questions:**
  - **1)** How do you represent a cluster of more than one point?
  - **2)** How do you determine the “nearness” of clusters?
  - **3)** When to stop combining clusters?

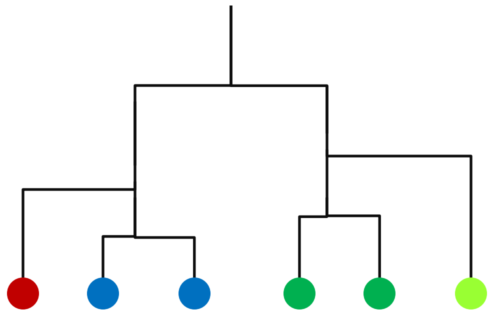
Hierarchical Clustering

- **Key operation: Repeatedly combine two nearest clusters**
- **(1) How to represent a cluster of many points?**
  - **Key problem:** As you merge clusters, how do you represent the "location" of each cluster, to tell which pair of clusters is closest?
- **Euclidean case:** each cluster has a ***centroid*** = average of its (data)points
- **(2) How to determine "nearness" of clusters?**
  - Measure cluster distances by distances of centroids

Example: Hierarchical clustering



Data:
o … data point
x … centroid



Dendrogram

And in the Non-Euclidean Case?

What about the Non-Euclidean case?

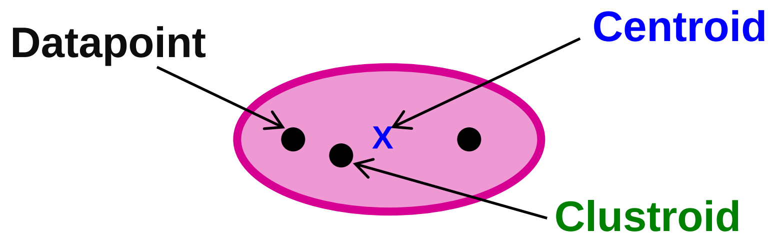
- The only "locations" we can talk about are the points themselves
  - i.e., there is no "average" of two points

- **Approach 1:**
  - **(1) How to represent a cluster of many points?**
    clustroid = (data)point "***<u>closest</u>***" to other points
  - **(2) How do you determine the "nearness" of clusters?** Treat clustroid as if it were centroid, when computing inter-cluster distances

"Closest" Point?

- **(1) How to represent a cluster of many points?**
  clustroid = point "***closest***" to other points
- **Possible meanings of "closest":**
  - Smallest maximum distance to other points
  - Smallest average distance to other points
  - Smallest sum of squares of distances to other points
    - For distance metric ***d*** clustroid ***c*** of cluster ***C*** is:

$$\min_{c} \sum_{x \in C} d(x,c)^2$$



Cluster on 3 datapoints

Centroid is the avg. of all (data)points in the cluster. This means centroid is an "artificial" point.
Clustroid is an existing (data)point that is "closest" to all other points in the cluster.

One Example of String Clustering

- Suppose we are using edit distance, and a cluster consists of the four points abcd, aecdb, abecb, and ecdab.

	ecdab	abecb	aecdb
abcd	5	3	3
aecdb	2	2	
abecb	4		

- According to the different meanings of "closest", we have:

Point	Sum	Max	Sum-Sq
abcd	11	5	43
aecdb	7	3	17
abecb	9	4	29
ecdab	11	5	45

Defining "Nearness" of Clusters

- **(1) How to represent a cluster of many points?**
  nil
- **(2) How do you determine the "nearness" of clusters?**
  - **Approach 2:**
    Intercluster distance = minimum of the distances between any two points, one from each cluster
  - **Approach 3:**
    Pick a notion of "**cohesion**" of clusters, *e.g.*, maximum distance from the clustroid
    - Merge clusters whose *union* is most cohesive

Cohesion

- **Approach 3.1:** Use the **diameter** of the merged cluster = maximum distance between points in the cluster
- **Approach 3.2:** Use the **average distance** between points in the cluster
- **Approach 3.3:** Use a **density-based approach**
  - Take the diameter or avg. distance, e.g., and divide by the number of points in the cluster

Implementation

- **Naïve implementation of hierarchical clustering:**
  - At each step, compute pairwise distances between all pairs of clusters, then merge
  - O(N^3)

- Careful implementation using priority queue can reduce time to O($N^2 \log N$)
  - **Still too expensive for really big datasets that do not fit in memory**

Supplementary Slides for LSH

2	1	4	1
1	2	1	2
2	1	2	1

Example of Bands

Assume the following case:

- Suppose 100,000 columns of ***M*** (100k docs)
- Signatures of 100 integers (rows)
- Therefore, signatures take 40Mb
- Choose ***b*** = 20 bands of ***r*** = 5 integers/band
- **Goal:** Find pairs of documents that are at least ***s*** *= 0.8* similar

2	1	4	1
1	2	1	2
2	1	2	1

C_1, C_2 are 80% Similar

- **Find pairs of ≥ *s*=0.8 similarity, set b=20, r=5**
- **Assume:** sim(C_1, C_2) = 0.8
  - Since sim(C_1, C_2) ≥ **s**, we want C_1, C_2 to be a **candidate pair**: We want them to hash to at **least 1 common bucket** (at least one band is identical)
- **Probability C_1, C_2 identical in one particular band:** $(0.8)^5 = 0.328$
- Probability C_1, C_2 are ***not*** similar in all of the 20 bands: $(1-0.328)^{20} = 0.00035$
  - i.e., about 1/3000th of the 80%-similar column pairs are **false negatives** (we miss them)
  - **We would find 99.965% pairs of truly similar documents**

C_1, C_2 are 30% Similar

2	1	4	1
1	2	1	2
2	1	2	1

- **Find pairs of ≥ *s*=0.8 similarity, set b=20, r=5**
- **Assume:** sim(C_1, C_2) = 0.3
  - Since sim(C_1, C_2) < **s** we want C_1, C_2 to hash to **NO common buckets** (all bands should be different)
- **Probability C_1, C_2 identical in one particular band:** $(0.3)^5$ = 0.00243
- Probability C_1, C_2 identical in at least 1 of 20 bands: $1 - (1 - 0.00243)^{20} = 0.0474$
  - In other words, approximately 4.74% pairs of docs with similarity 0.3 end up becoming **candidate pairs**
    - They are **false positives** since we will have to examine them (they are candidate pairs) but then it will turn out their similarity is below threshold **s**

2	1	4	1
1	2	1	2
2	1	2	1

LSH Involves a Tradeoff

- **Pick:**
  - The number of Min-Hashes (rows of ***M***)
  - The number of bands ***b***, and
  - The number of rows ***r*** per band

  to balance false positives/negatives

- **Example:** If we had only 15 bands of 5 rows, the number of false positives would go down, but the number of false negatives would go up