CS4225/CS5425 Big Data Systems for Data Science

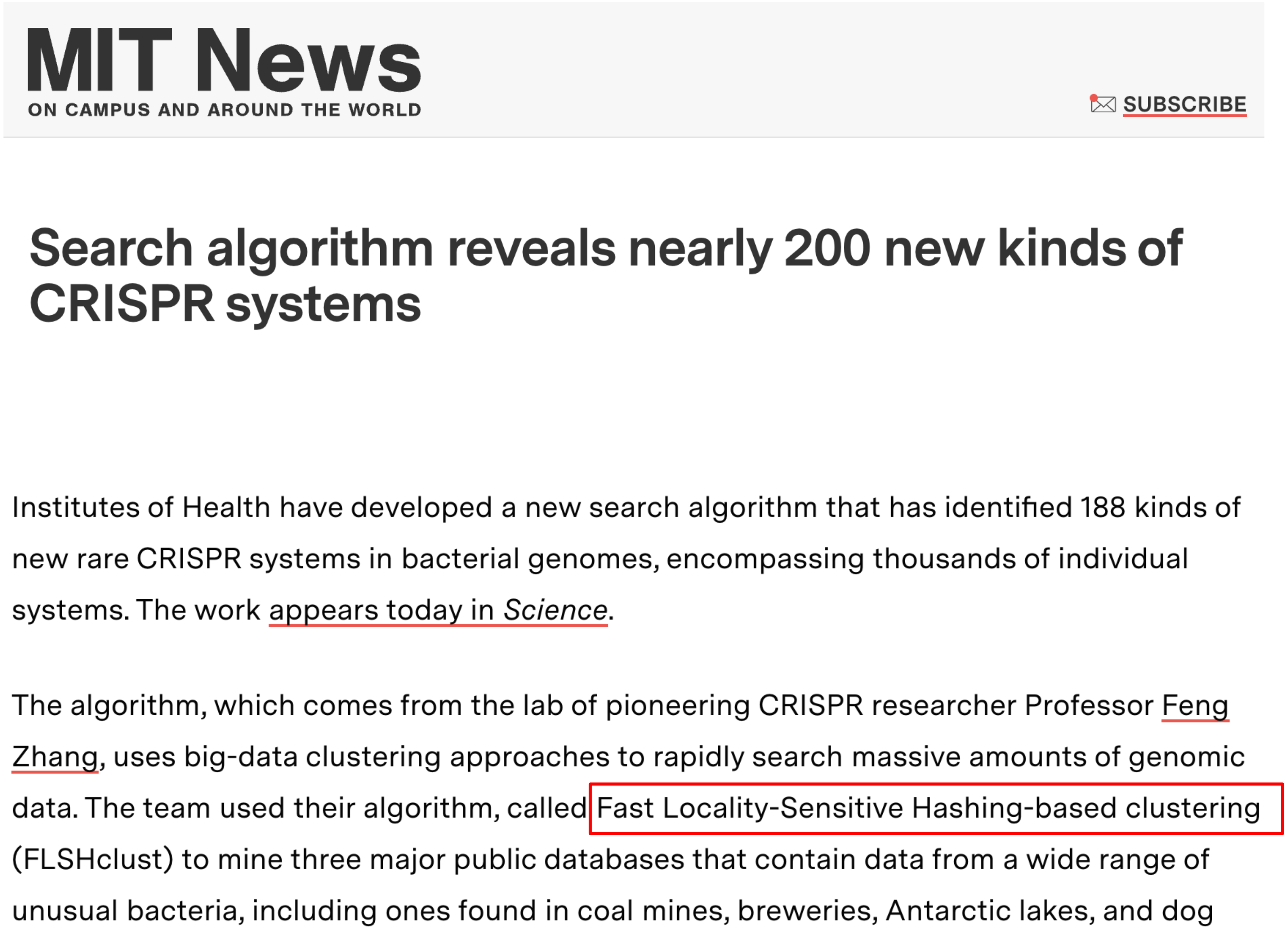
MapReduce & Data Mining

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

Overview

1. **Similarity Search**
2. Clustering

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



MIT News

ON CAMPUS AND AROUND THE WORLD

SUBSCRIBE

Search algorithm reveals nearly 200 new kinds of CRISPR systems

Institutes of Health have developed a new search algorithm that has identified 188 kinds of new rare CRISPR systems in bacterial genomes, encompassing thousands of individual systems. The work appears today in *Science*.

The algorithm, which comes from the lab of pioneering CRISPR researcher Professor Feng Zhang, uses big-data clustering approaches to rapidly search massive amounts of genomic data. The team used their algorithm, called Fast Locality-Sensitive Hashing-based clustering (FLSHclust) to mine three major public databases that contain data from a wide range of unusual bacteria, including ones found in coal mines, breweries, Antarctic lakes, and dog

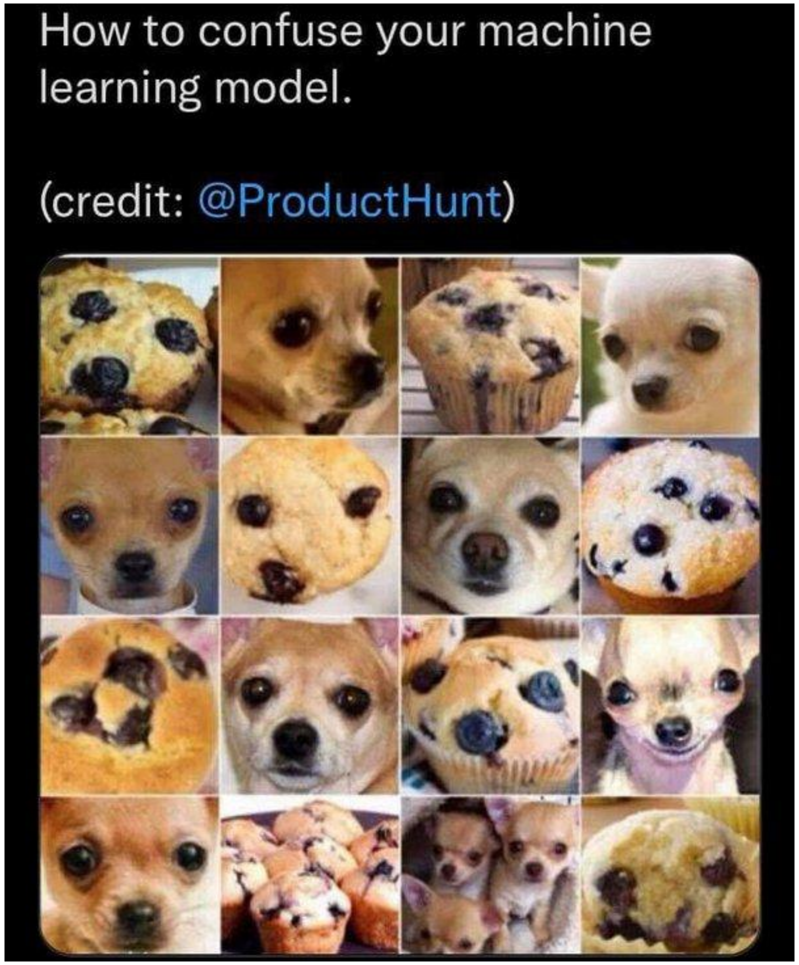
https://news.mit.edu/2023/search-algorithm-reveals-nearly-200-new-kinds-crispr-systems-1123

Similarity: a Common Subroutine

- Many problems can be expressed as finding "similar" objects:
- Examples:
  - Documents with similar words
    - For duplicate detection, or classification by topic
  - Customers who purchased similar products
    - For recommendation
  - Users who visited similar websites
    - For grouping users with similar interests

Similarity (or Distance) 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

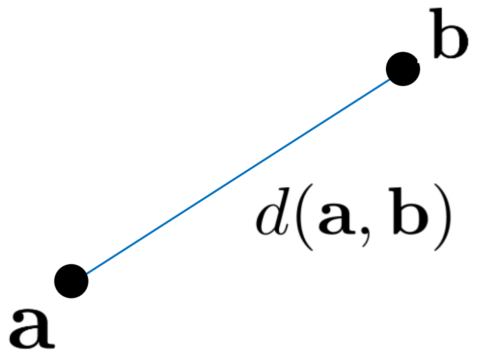

How to confuse your machine learning model.

(credit: @ProductHunt)

Common Distance / Similarity Metrics

- Euclidean distance

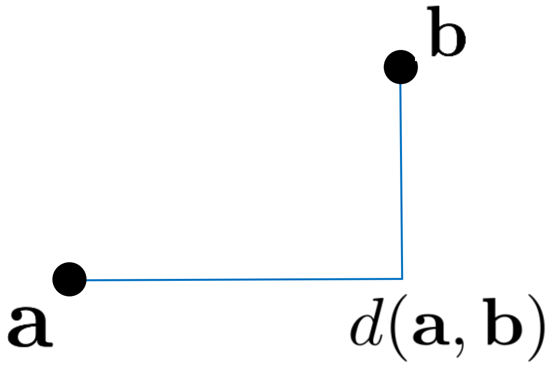
$$d(\mathbf{a}, \mathbf{b}) = \|\mathbf{a} - \mathbf{b}\|_2 = \sqrt{\sum_{i=1}^{D} (a_i - b_i)^2}$$

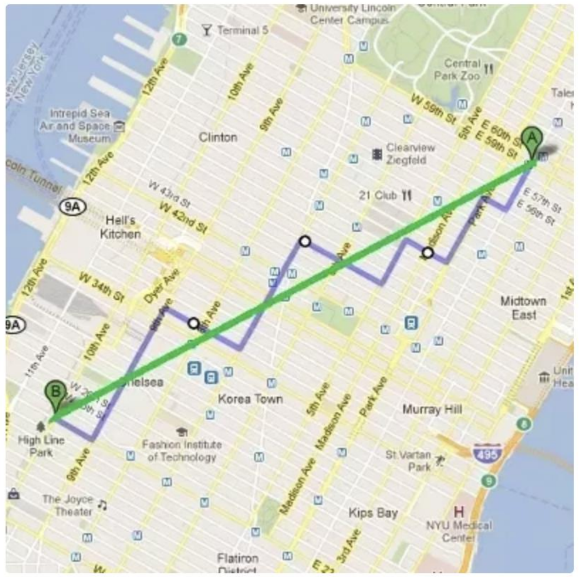


Common Distance / Similarity Metrics

- Manhattan distance

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

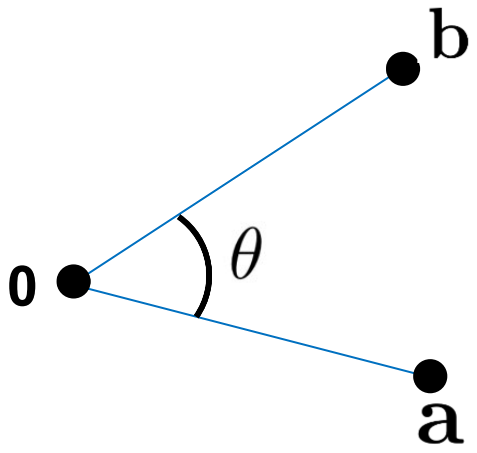




Common Distance / Similarity Metrics

- Cosine similarity

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

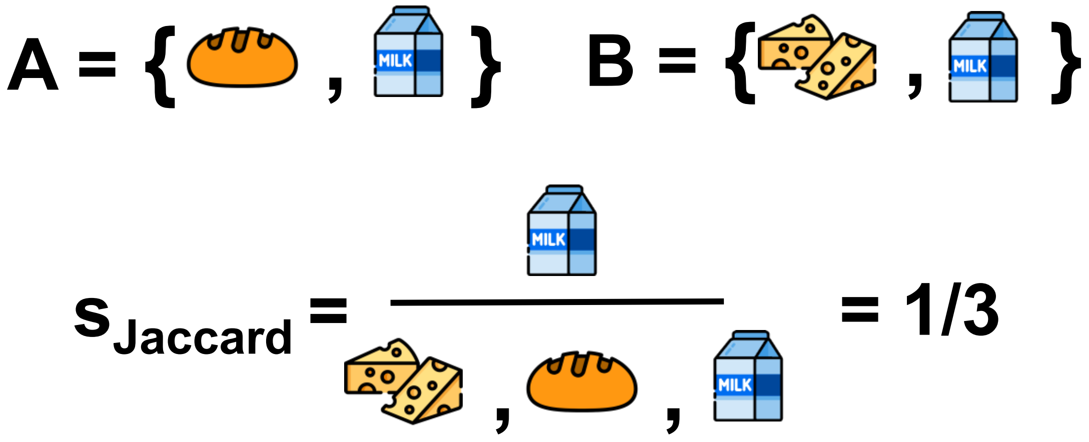


Only considers direction: cosine similarity doesn’t change if we scale a or b (i.e. multiplying them by a constant)

Common Distance / Similarity Metrics

- Jaccard Similarity
  (between sets A and B)

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

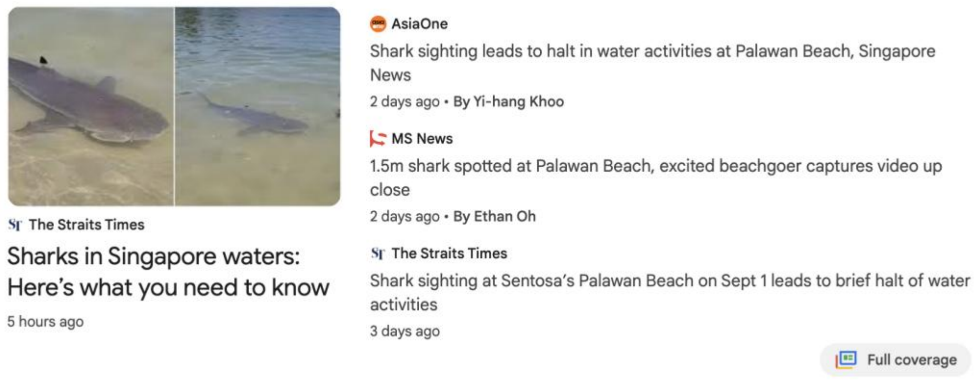


- Jaccard Distance

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

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"



The Straits Times
Sharks in Singapore waters: Here's what you need to know
5 hours ago

AsiaOne
Shark sighting leads to halt in water activities at Palawan Beach, Singapore News
2 days ago • By Yi-hang Khoo

MS News
1.5m shark spotted at Palawan Beach, excited beachgoer captures video up close
2 days ago • By Ethan Oh

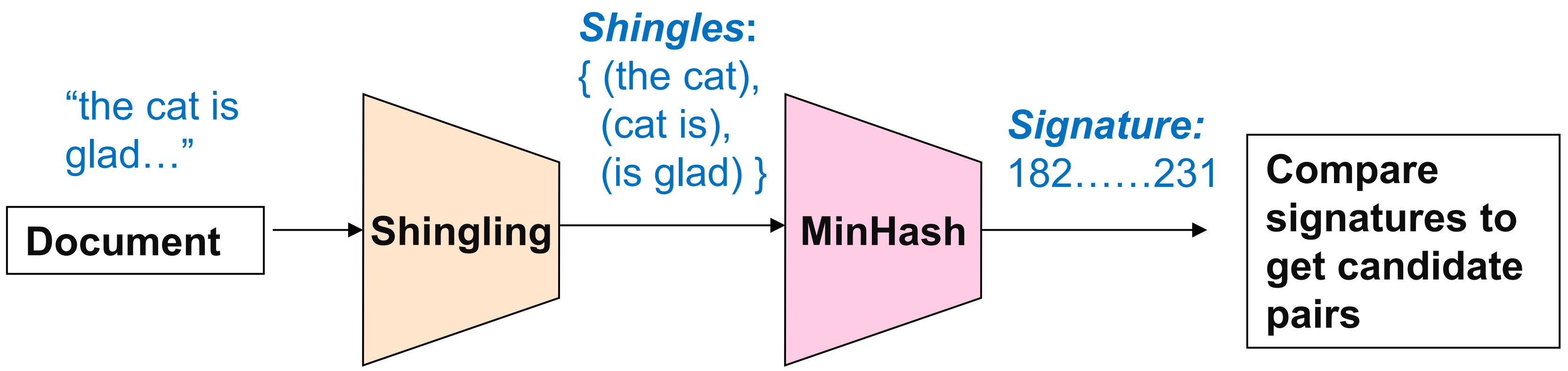
The Straits Times
Shark sighting at Sentosa's Palawan Beach on Sept 1 leads to brief halt of water activities
3 days ago

2 Essential Steps for Similar Docs

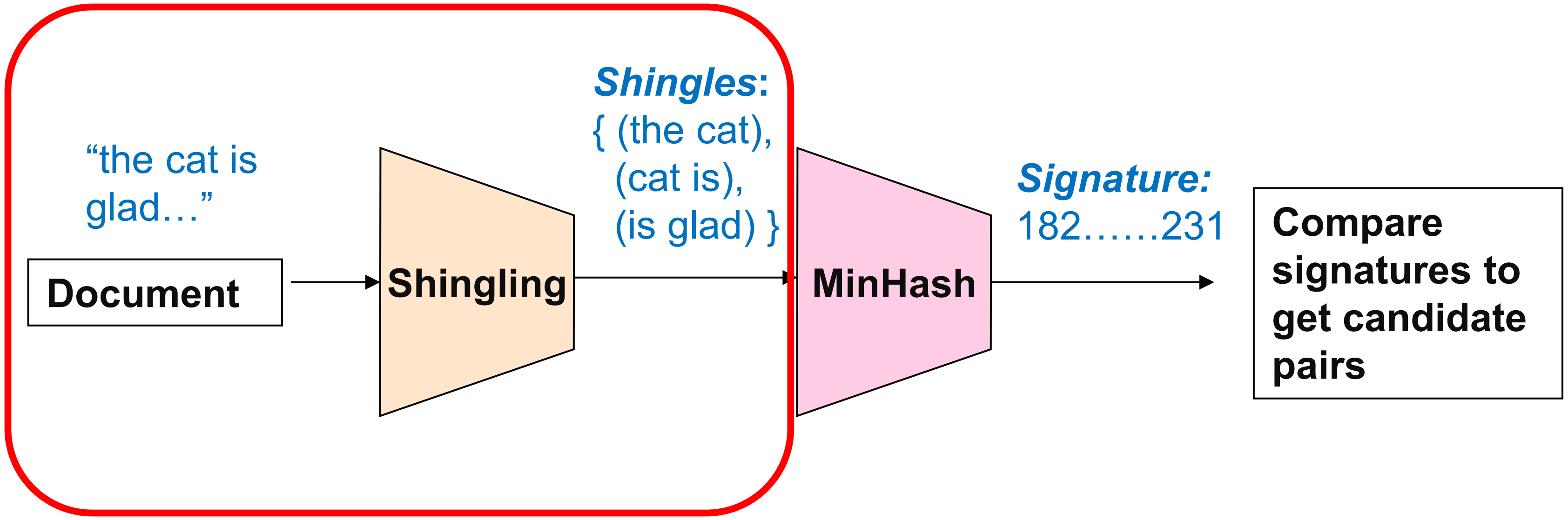
1. **Shingling**: Convert documents to sets of short phrases ("shingles")

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 for finding near-duplicates

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


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

Shingling

Step 1: Shingling: Convert documents to sets of phrases

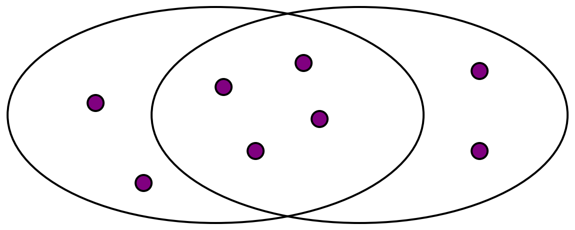
Define: Shingles

- 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_i can be thought of as a set of its k-shingles C_i
  - E.g. if D = “the cat is”, then its set of k-shingles 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:

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



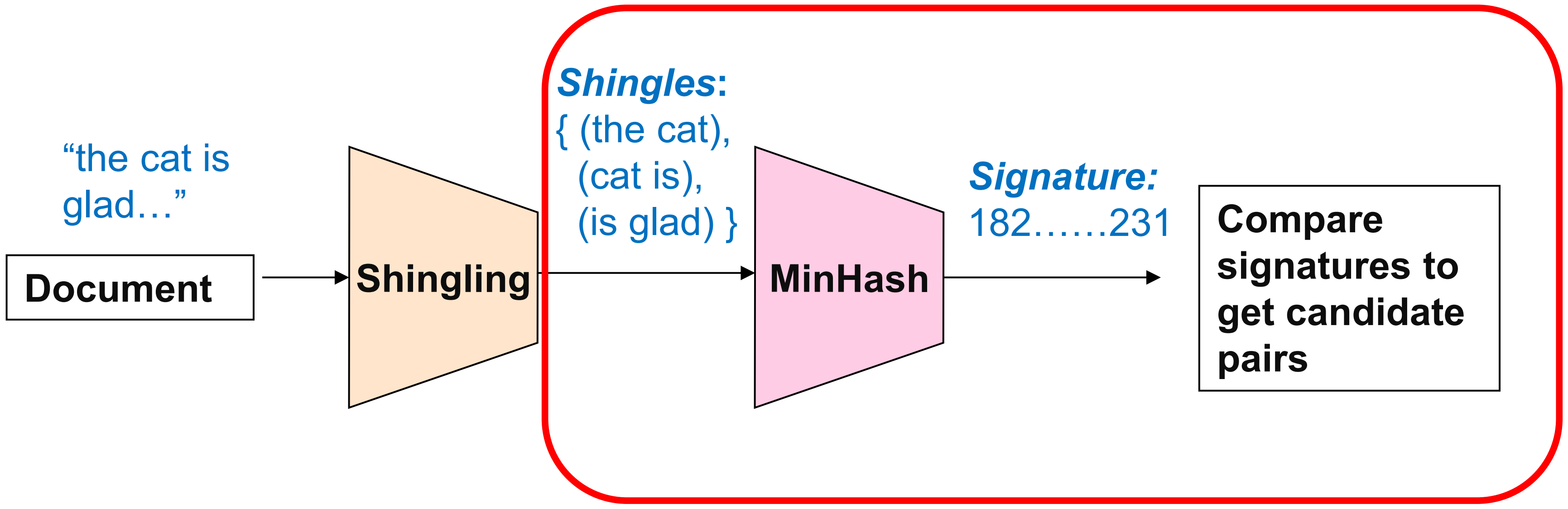
Documents

Shingles

	D_1	D_2	…	
“the cat”	1	1	1	0
“cat is”	1	1	0	1
…	0	1	0	1
	0	0	0	1
	1	0	0	1
	1	1	1	0
	1	0	1	0

Motivation for MinHash

- Suppose we have $N = 1$ million documents
- Naively, we would have to compute pairwise Jaccard similarities for every pair of docs
  - $\boldsymbol{N(N-1)/2} \approx 5*10^{11}$ 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) ≠ 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

Document 1

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

Hashes:

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

MinHash:

Document 2

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

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

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

[Optional] Key Property of MinHash

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

Document 1

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

Document 2

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

Hashes:

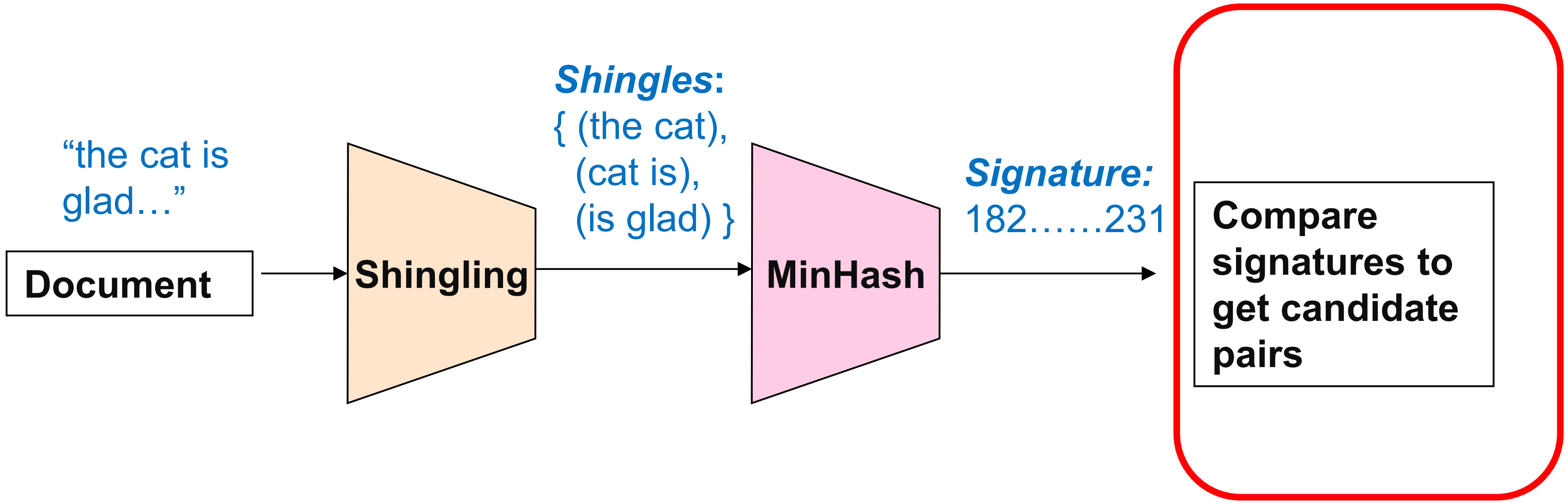
$$\begin{tikzcd}
32 \arrow[dr] & 12 \arrow[d] & 24 \arrow[dl] & & 93 \arrow[dr] & 12 \arrow[d] & 24 \arrow[dl] \\
& \text{MinHash} = 12 & & & & \text{MinHash} = 12 &
\end{tikzcd}$$

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.
=> $\Pr[h(C_1) = h(C_2)]$ = (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 may 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.
- (optional) More advanced algorithms: Locality Sensitive Hashing (LSH)

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

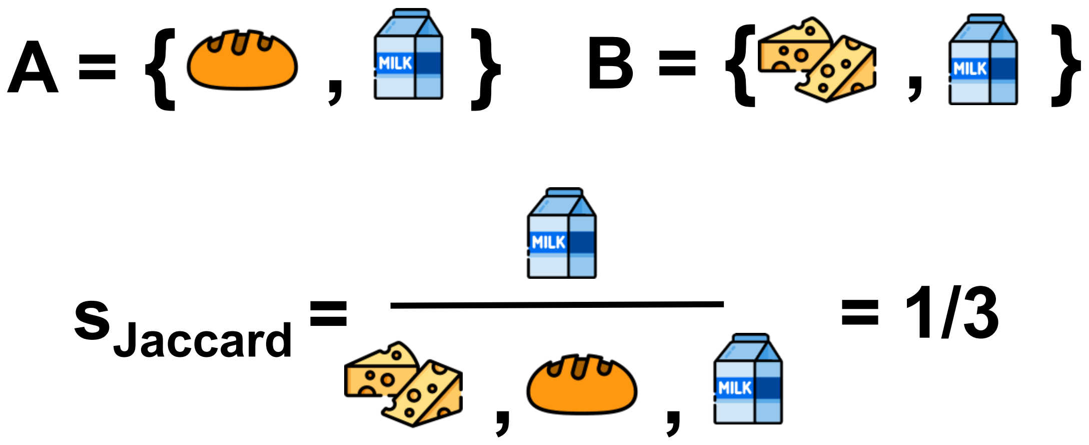
MapReduce Implementation

- 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

[Recap] Common Distance / Similarity Metrics

- Jaccard Similarity (between sets A and B)

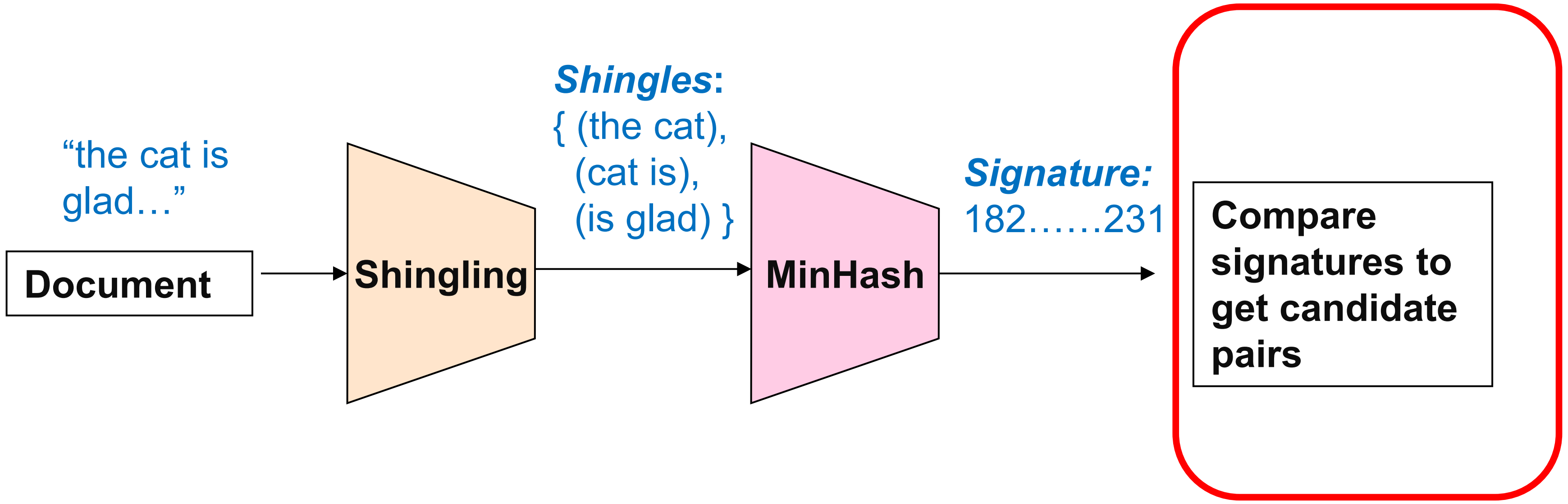
$$s_{\text{Jaccard}}(A, B) = \frac{|A \cap B|}{|A \cup B|}$$



- Jaccard Distance

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

[Recap] 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 may 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.
- (optional) More advanced algorithms: Locality Sensitive Hashing (LSH)

[Recap] 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. **<u>Clustering</u>**

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

SFGIANTS.COM/VOTE
at&t

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

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

KIA 4 5 삼성
대구은행
치킨더홈
SPOTV

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

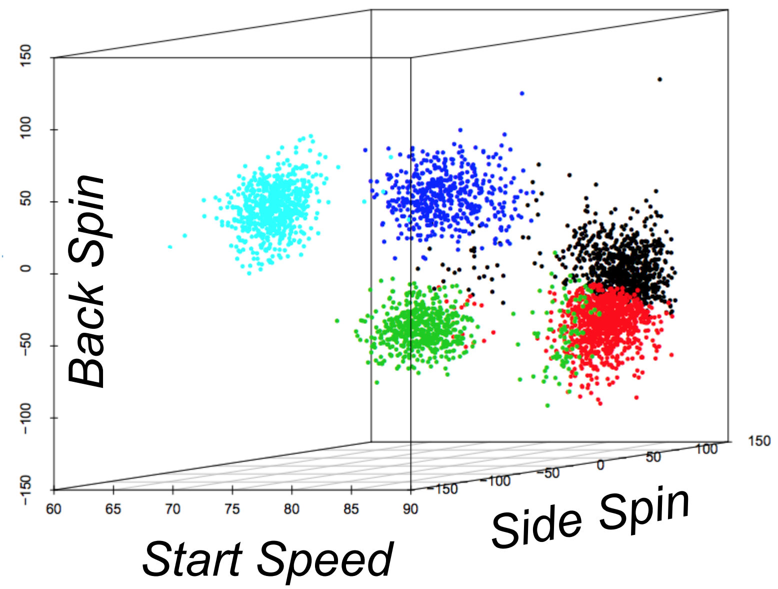
mariners.com
T-Mobile
T-Mobile
IWAKUMA 33 PITCHES
TB 0
SEA 1 0-2
PITCHER GIFS

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

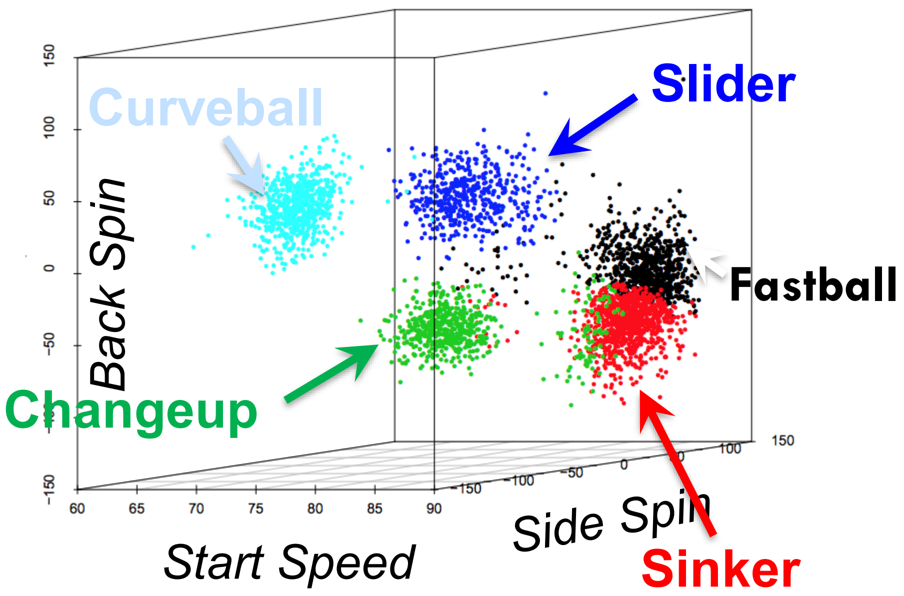
KIA 4 5 삼성
SPOTV
삼성카드 삼성증권 삼성생명
KCW
대구은행
치킨더홈

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



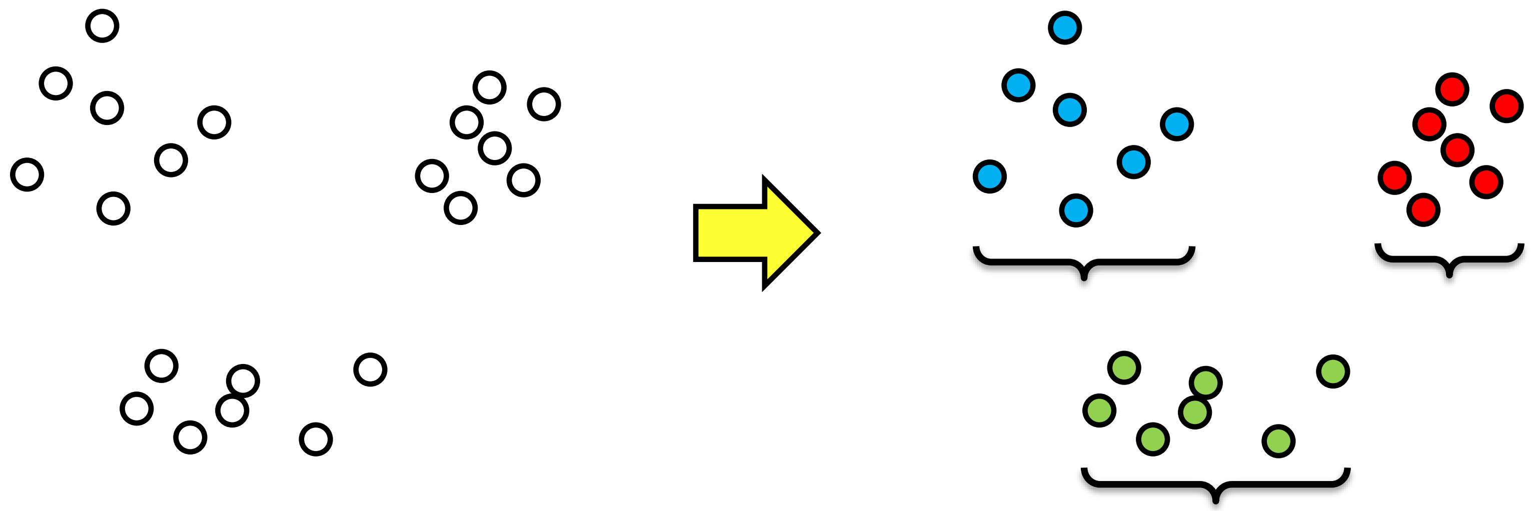
Pitches by Barry Zito



Goal of Clustering

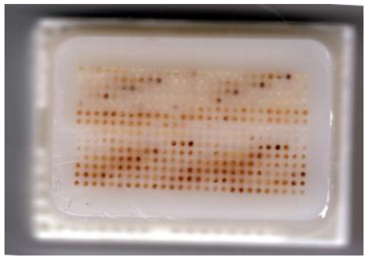
Clustering separates unlabelled data into groups of similar points.

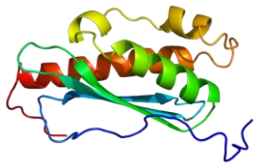
Clusters should have high intra-cluster similarity, and low inter-cluster similarity.

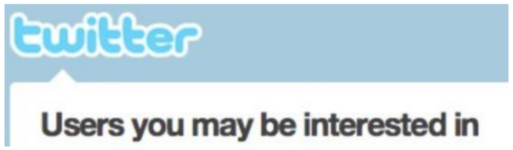


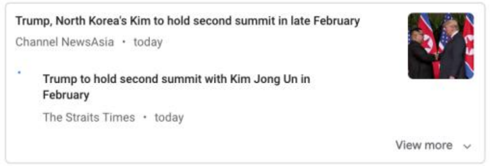
Applications of Clustering

Many applications:



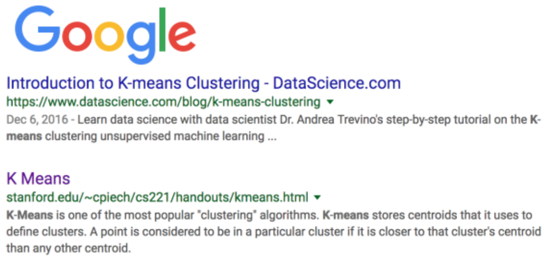



Users you may be interested in


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

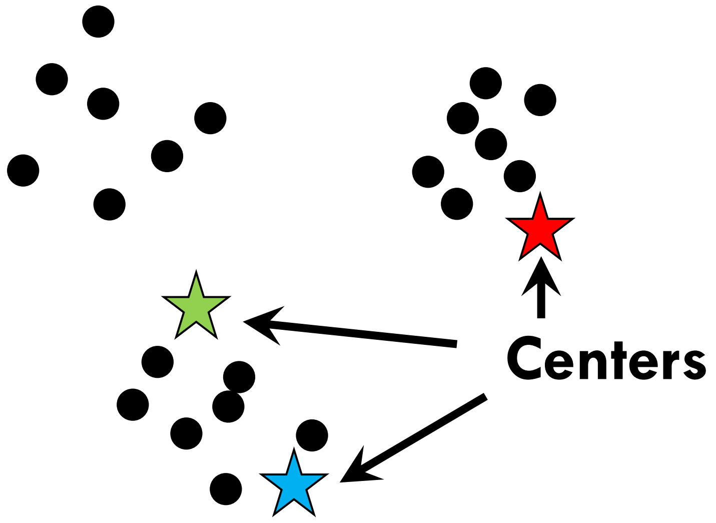
1. Initialization: Pick K random points as centers

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

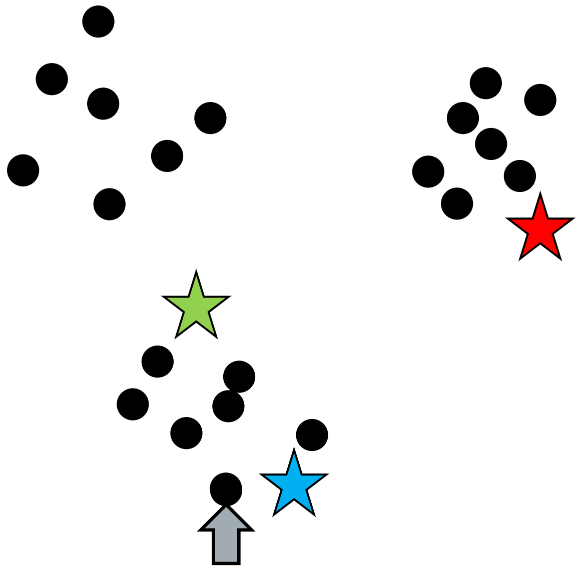
1. Initialization: Pick K random points as centers

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

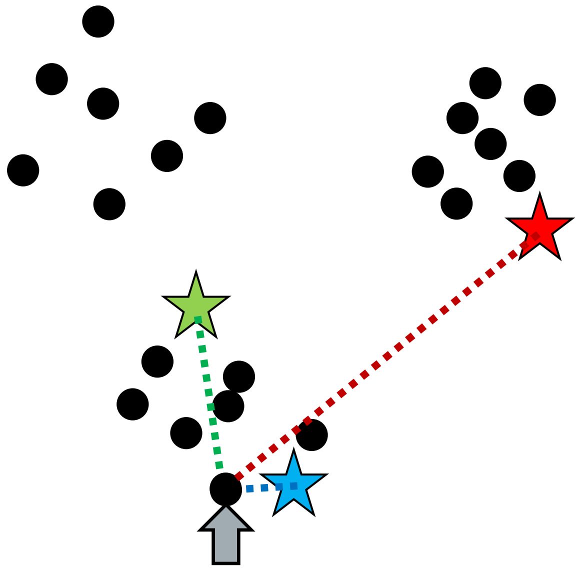
1. **Initialization**: Pick K random points as centers

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

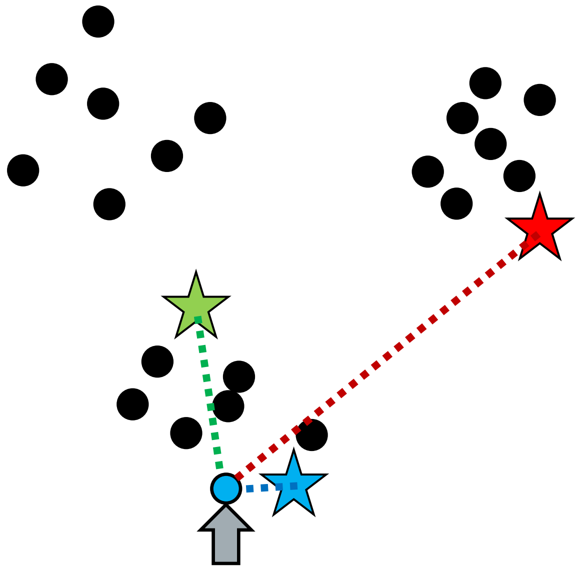
1. Initialization: Pick K random points as centers

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

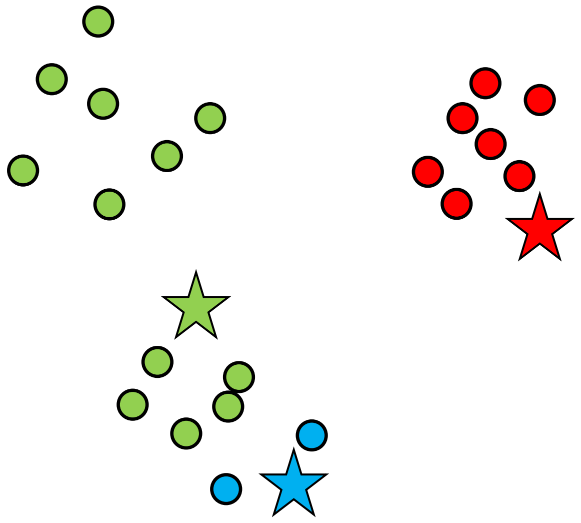
1. **Initialization**: Pick K random points as centers

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

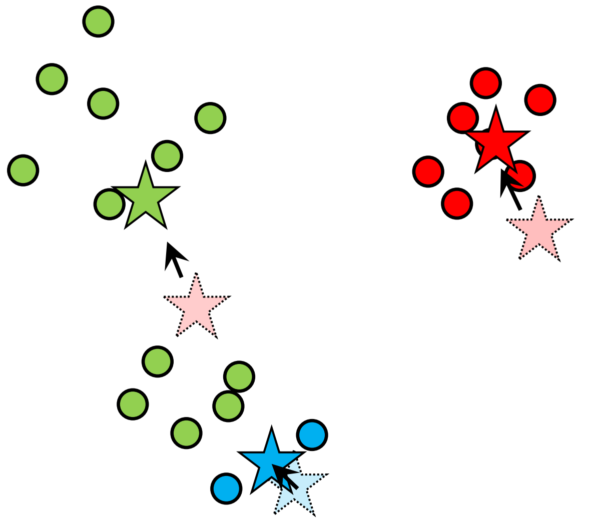
1. **Initialization**: Pick K random points as centers

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

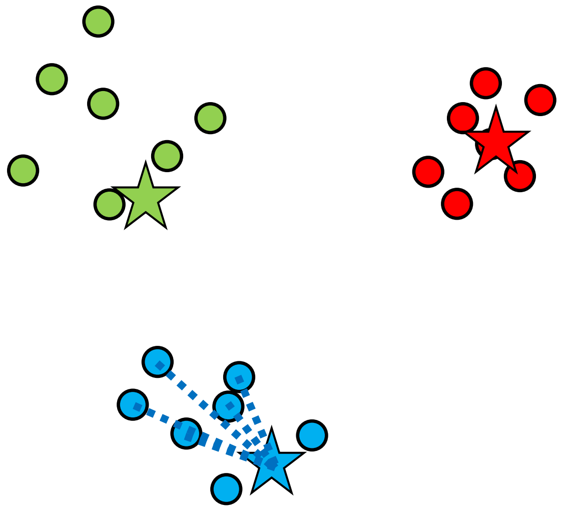
1. **Initialization**: Pick K random points as centers

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

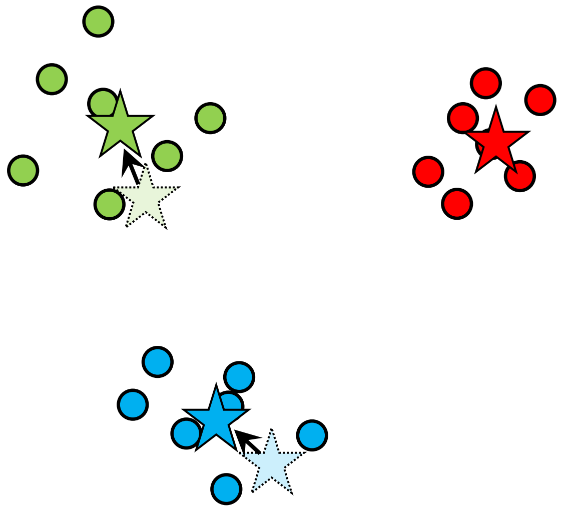
1. **Initialization**: Pick K random points as centers

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



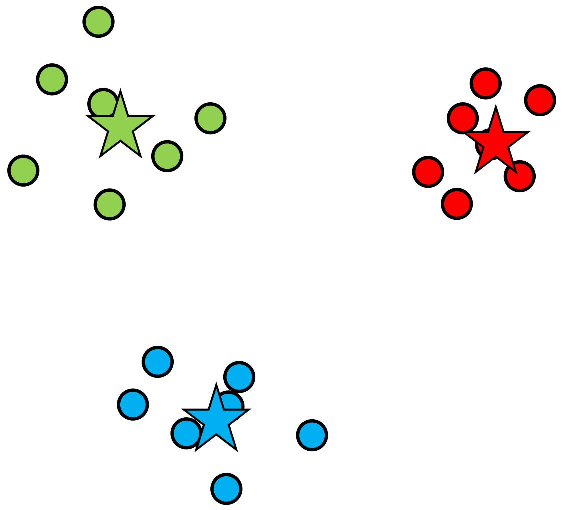
K-Means Algorithm: Steps

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



K-Means Algorithm: Steps

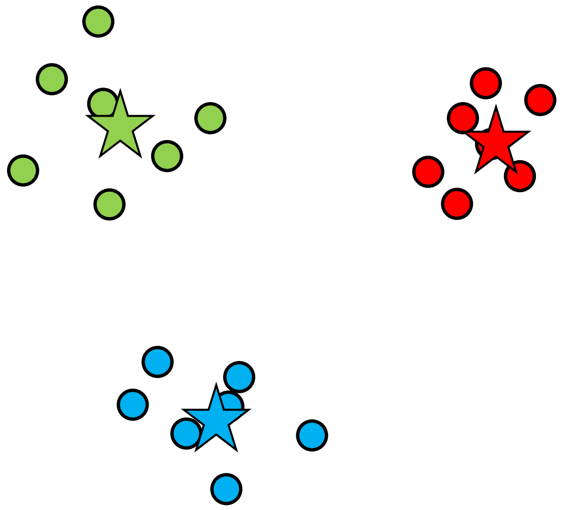
1. Initialization: Pick K random points as centers

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

```
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!

This MapReduce job performs a single iteration of k-means.

Configure() reads in the current positions of the cluster centers (i.e. stars) from a file.

Map() assigns each point to the closest cluster.

Reduce() aggregates the points in each cluster.

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 $m$)
```

This MapReduce job is similar to v1, but uses an in-mapper combiner to combine points in the same map task using a data structure H.

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)
(you may assume there is only 1 map task)

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)

(you may assume there is only 1 map task)

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

Take-away

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

Take-away in the AI Era

- (Why data mining at scale still needs human responsibility)
- 🧠 What AI cannot decide for you (yet)
  - Whether an algorithm is **scalable at data-center scale**
  - Whether the system cost is dominated by **computation or data movement**
  - Whether performance optimizations change the *asymptotic cost*
- **Your design matters (AI can assist)**
  - Translating algorithms into scalable system designs
  - Reasoning about approximation vs exactness
  - Understanding iterative algorithm costs
  - Designing system-level optimizations

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

Acknowledgement

- Slides adopted/revised from
  - Jimmy Lin, http://lintool.github.io/UMD-courses/bigdata-2015-Spring/
  - Bryan Hooi
- Some slides are also 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/