Bloom Filters, Minhashes, and Other Random Stuff

Brian Brubach

University of Maryland, College Park

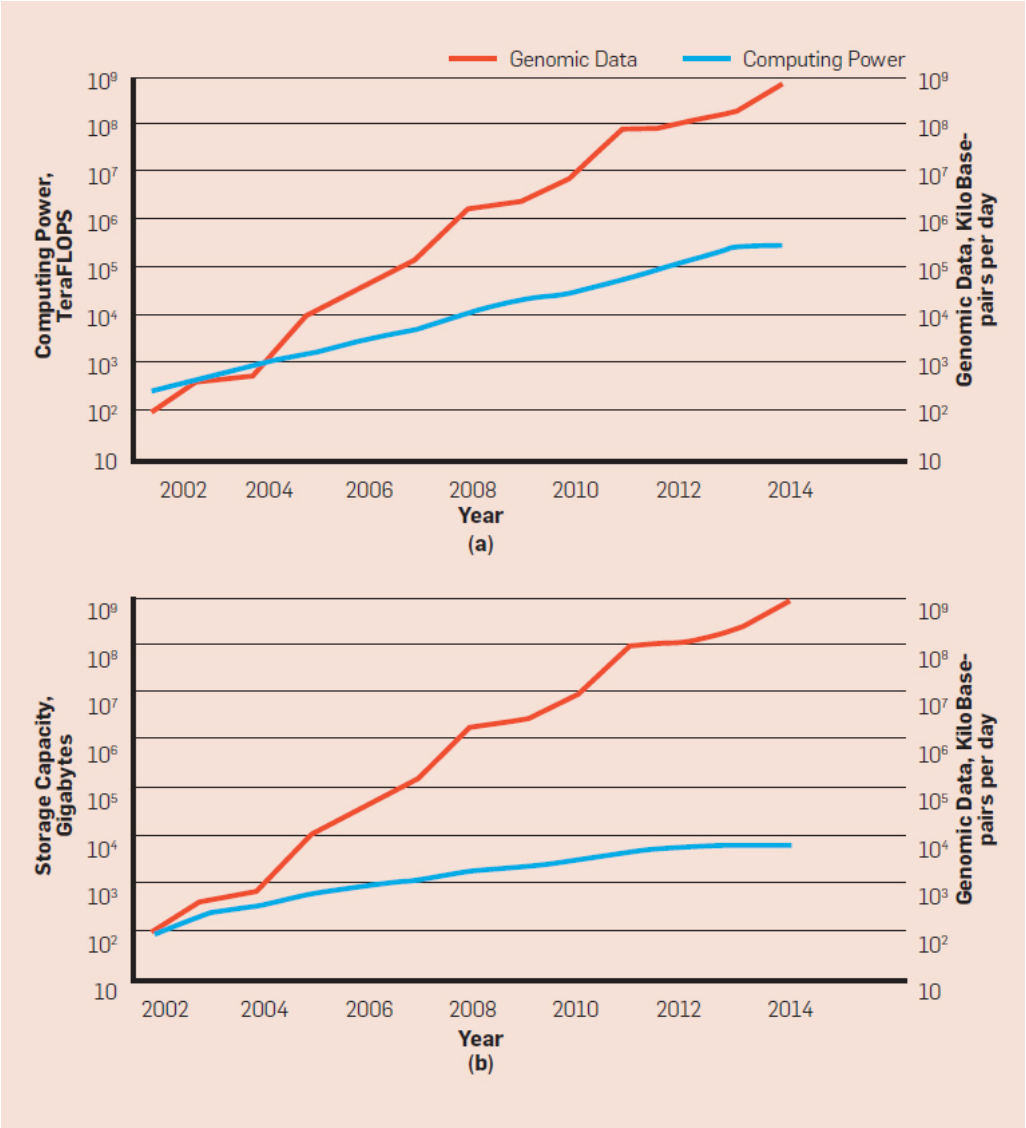
StringBio 2018, University of Central Florida

What?

- Probabilistic
- Space-efficient
- Fast
- Not exact

Why?

- Data deluge/Big data/Massive data
- Millions or billions of sequences
- Human genome: 3 Gbp
  - 1 giga base pairs = 1 billion characters
- Microbiome sample of 1.6 billion 100 bp reads generated in 10.8 days (Caporaso, et al., 2012)
- Medium data, but on a laptop
  - Lots of bioinformatics happens here
- Beyond scalability of BWT, FM-index, etc.



(Berger, Daniels, and Yu, 2016)

Curse of Dimensionality

- Sequences are compared in high dimensional space
- Comparing N sequences takes N^2 time
- Computing edit distance between two sequences of length n takes n^2 time
  - Allegedly

Curse of Dimensionality

- ATGATCGAGGCTATGCGACCGATCGATCGATTCGTA
- ATGATGGAGGCTATGGGAACGATCGATCGACTCGTA
- ATGATCGAGGCTATGCCACCGATCGAACGATTCGTA
- ATCATCGAGGCTATGCGACCGTTCGATCGATTCCTA
- GTGATCGTGGCTATGCGACCGATCGATCGATTCGTC
- ATGATCGAGGCTATGCCACCGATCGAACGATTCGTA
- ATGATCCAGGCTATGCGACCGATCGATGCATTCGTA

Why Stay in High Dimensions?

- 4^{100} possible DNA strings of length 100
- $4^{15} \approx$ 1 billion reads

k-mers of a Sequence

- All substrings of length k
- Canonical: lexicographically smallest among forward and reverse complement
  - Forget this for now

```
Reverse complement:
ATCTGAGGTCAC
GTGACCTCAGAT
```

```
All 7-mers:
ATCTGAGGTCAC
ATCTGAG
 TCTGAGG
  CTGAGGT
   TGAGGTC
    GAGGTCA
     AGGTCAC
```

Hash function

- Will assume idealized model of hashing for this talk
- Lots of research in this area

Bloom Filter Example Problem

- Store a large set of N k-mers
- Query k-mers against it for exact matches
- Want speed and space-efficiency

Bloom Filter Example Problem

- Store a large set of N k-mers
- Query k-mers against it for exact matches
- Want speed and space-efficiency
- How can we address this with hashing?

Bloom Filter Example Problem

- Store a large set of N k-mers
- Query k-mers against it for exact matches
- Want speed and space-efficiency
- How can we address this with hashing?
  - Put k-mers in hash table

Bloom Filter Example Problem

- Store a large set of N k-mers
- Query k-mers against it for exact matches
- Want speed and space-efficiency
- How can we address this with hashing?
  - Put k-mers in hash table
  - At least $2Nk$ bits for data plus table overhead

Bloom Filter Example Problem

- Store a large set of N k-mers
- Query k-mers against it for exact matches
- Want speed and space-efficiency
- How can we address this with hashing?
  - Put k-mers in hash table
  - At least $2Nk$ bits for data plus table overhead
- What if we just store one bit at each hash for presence/absence?
  - Simple Bloom filter, potentially suboptimal

Bloom Filter

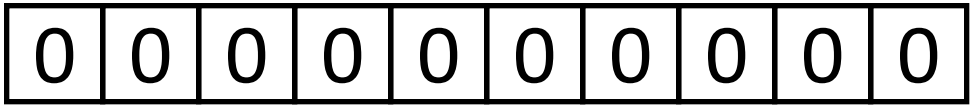
- Probabilistic data structure
- Fast and space-efficient
- False positives, but no false negatives
- Insert and contains, but no delete
- Due to Burton Howard Bloom in 1970
  - Gave example of automatic hyphenation
  - Identify the 10% of words that require special hyphenation rules

Bloom Filter

- N items to store: $x_1, x_2, \ldots, x_N$
- m-bit vector
- d hash functions: $h_1, h_2, \ldots, h_d$
- Insert(x): set bits $h_1(x), h_2(x), \ldots, h_d(x)$ to 1
- Contains(y):
  - Yes if bits $h_1(y), h_2(y), \ldots, h_d(y)$ are 1
  - No if any are 0

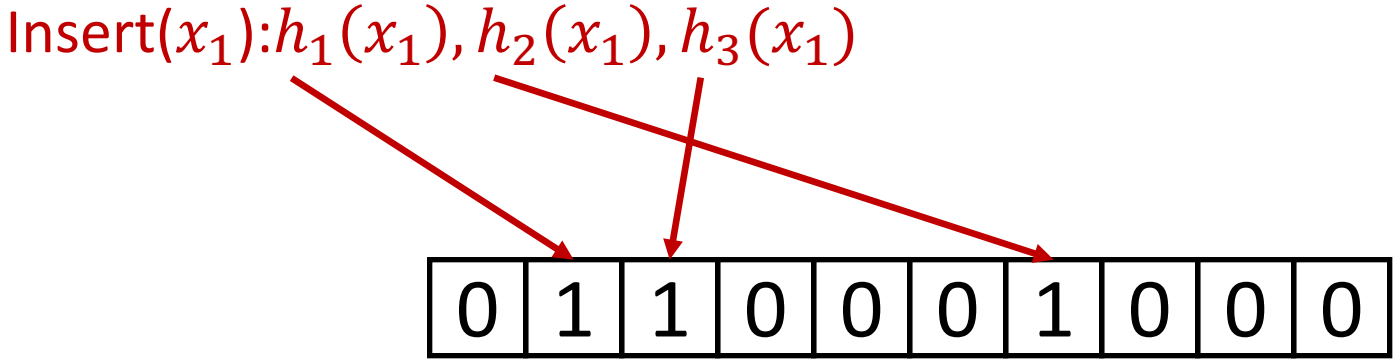
Bloom Filter Example

- $m = 10$, $d = 3$, hash functions: h_1, h_2, h_3



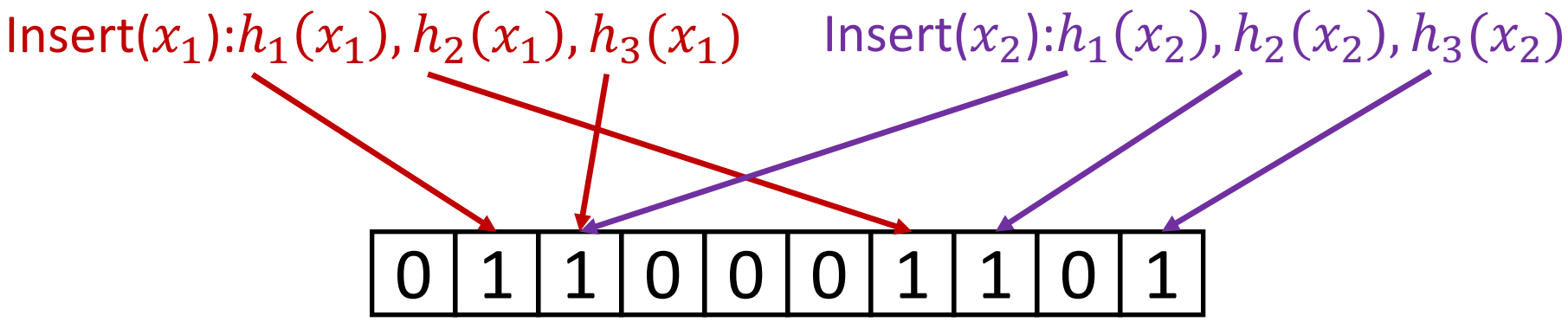
Bloom Filter Example

- $m = 10$, $d = 3$, hash functions: h_1, h_2, h_3



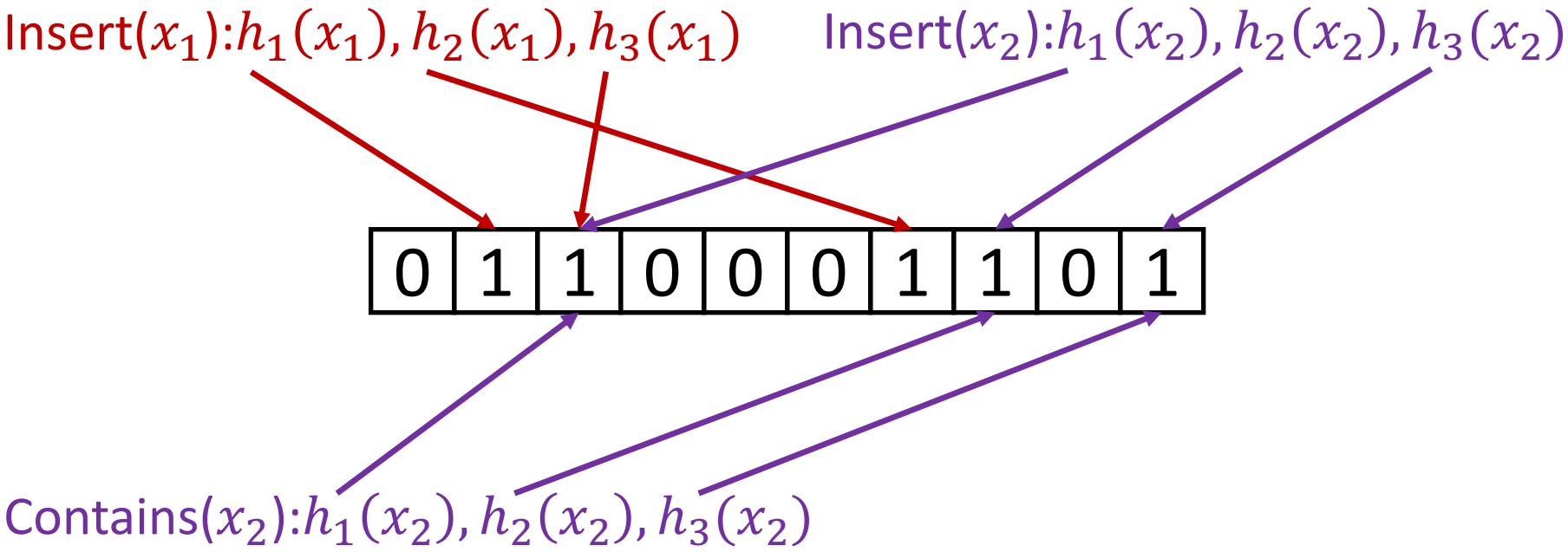
Bloom Filter Example

- $m = 10$, $d = 3$, hash functions: h_1, h_2, h_3



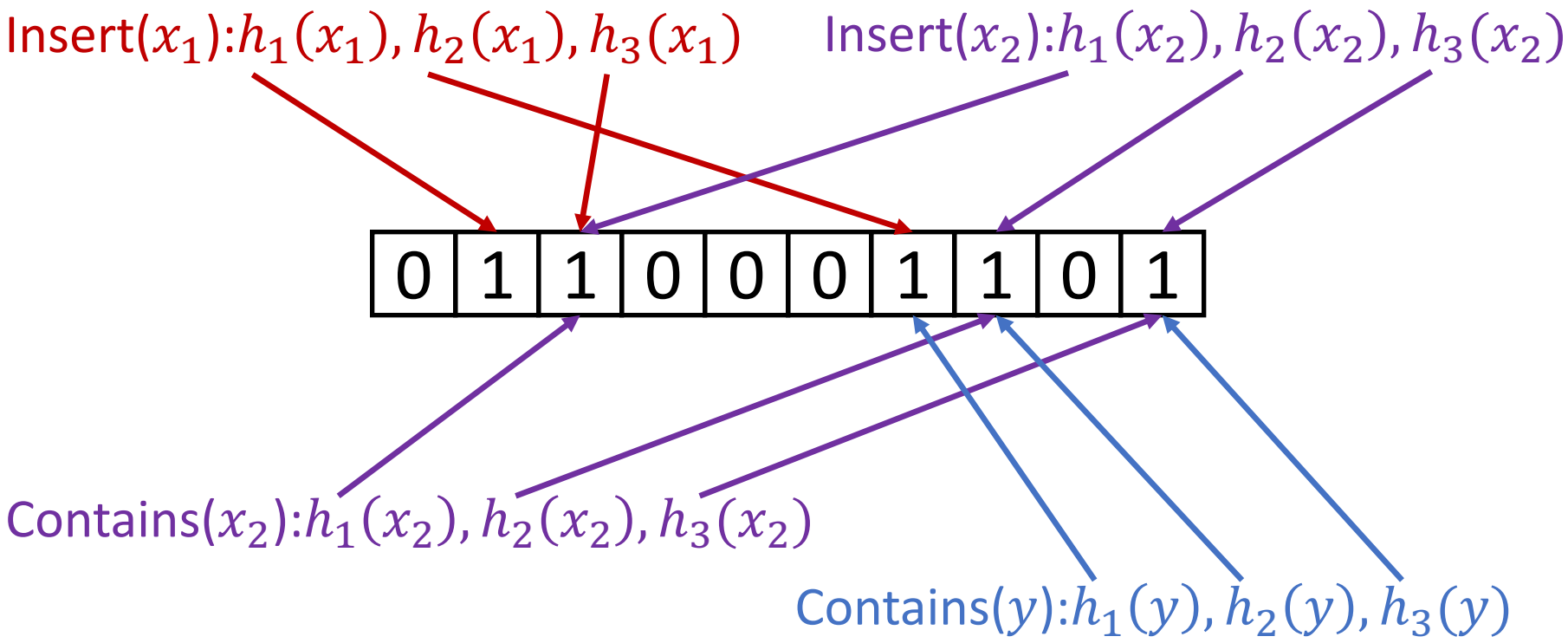
Bloom Filter Example

- $m = 10$, $d = 3$, hash functions: h_1, h_2, h_3



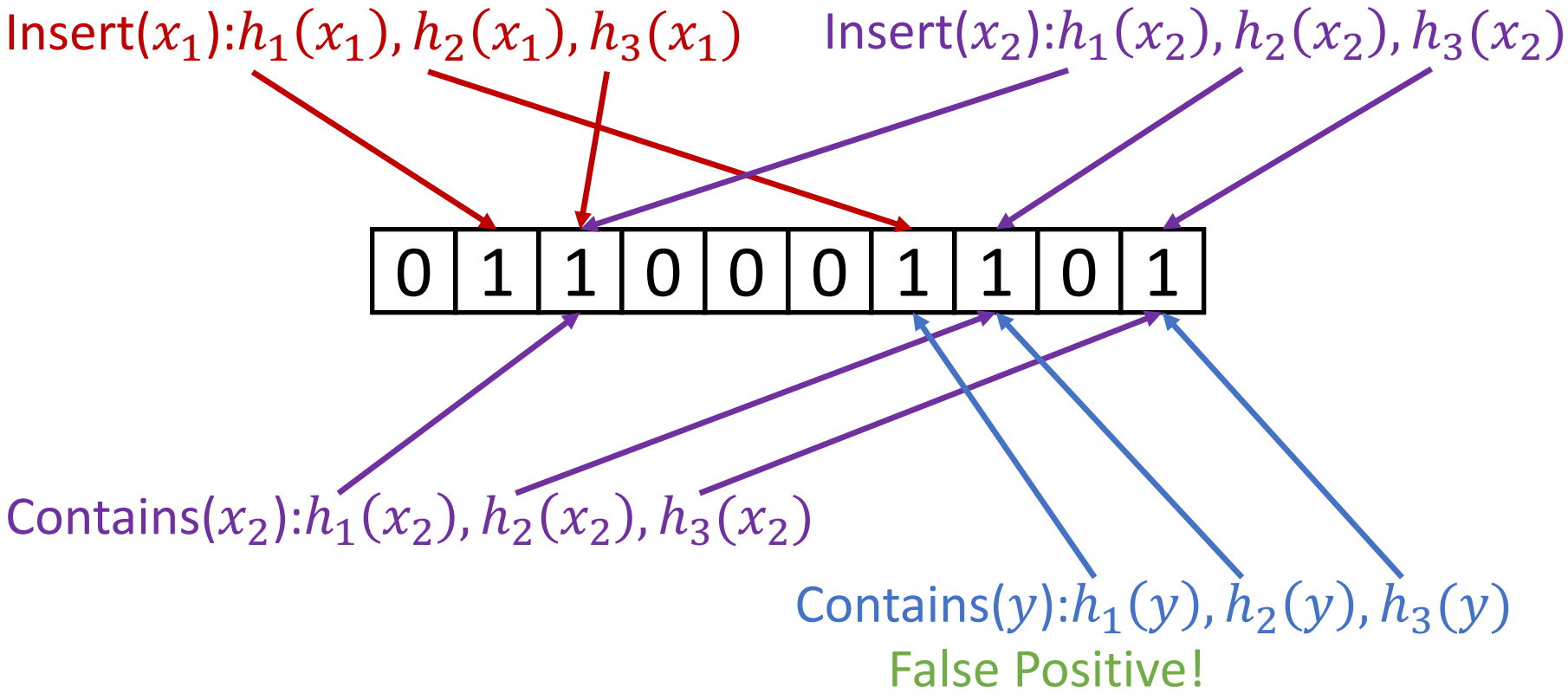
Bloom Filter Example

- $m = 10$, $d = 3$, hash functions: h_1, h_2, h_3



Bloom Filter Example

- $m = 10$, $d = 3$, hash functions: h_1, h_2, h_3



False Positive probability

- Pr[one hash misses a bit]
  - $1 - \frac{1}{m}$
- Pr[one insertion misses a bit]
  - $\left(1 - \frac{1}{m}\right)^{d}$
- Pr[all insertions miss a bit]
  - $\left(1 - \frac{1}{m}\right)^{dn}$
- Pr[a single bit flipped to 1]
  - $1 - \left(1 - \frac{1}{\mathrm{m}}\right)^{\mathrm{dn}} \approx 1 - e^{-dn/m}$
- False positive probability (assuming independence)
  - $\left(1 - e^{-dn/m}\right)^{d}$

Optimal parameters

- False positive rate $p \approx \left(1 - e^{-dn/m}\right)^d$
- False positives minimized at $d = \frac{m}{n} \ln 2$
- Bits per item $\frac{m}{n} \approx -\frac{\log_2 p}{\ln 2} \approx -1.44 \log_2 p$
  - Approximate: assuming asymptotic, independence, and integrality of d
  - $p = 0.01$, needs 9.59 bits per item
  - $p = 0.001$, needs 14.38 bits per item
- Number of hashes $d \approx -\log_2 p$

Properties

- Insert and check in $O(d)$ time
  - Independent of number of items inserted
- Fast and parallel to compute hashes
- Can do union and intersection with OR and AND of bit vectors
- Can estimate N if unknown

Endless Variations

- Deletions
- Counting
- Bloomier filters: storing values
- Cache optimizations
- Distance sensitive: is x close to the set

k-mer Bloom Filter

- Can we do better if we know the items are k-mers from a genome?

k-mer Bloom Filter

- Can we do better if we know the items are k-mers from a genome?
- Observation: the “items” are overlapping substrings from a 4 letter alphabet

k-mer Bloom Filter

- Can we do better if we know the items are k-mers from a genome?
- Observation: the "items" are overlapping substrings from a 4 letter alphabet
- After getting positive,
  - Check all 4 preceding k-mers and all 4 following k-mers
  - One must be in the set for a true positive
  - False positive next to another positive less likely
- Can reduce false positives or space
- (Pellow, Filippova, and Kingsford, 2017)

```
ATCC
xATC
 TCCx
```

Bio Applications

- Pan-genome storage
  - Bloom filter trie (Holley, Wittler, and Stoye, 2015)
- Short-read RNA-seq database
  - Split Sequence Bloom tree (Solomon and Kingsford, 2016)
- Succinct de Bruijn graphs
  - Probabilistic de Bruijn graph (Pell, et al., 2011)
  - Exact version (Chikhi and Rizk, 2012)
    - Human genome: 3 Gbp, $k = 27$, 3.7 GB, 13.2 bits per vertex

Locality Sensitive Hashing (LSH)

- What do we typically want to avoid when hashing?

Locality Sensitive Hashing (LSH)

- What do we typically want to avoid when hashing?
  - Collisions!
- Approximate nearest neighbors: towards removing the curse of dimensionality (Indyk and Motwani, 1998)
  - Idea: get similar elements to hash together
  - “Its key ingredient is the notion of *locality-sensitive hashing* which may be of independent interest;…”

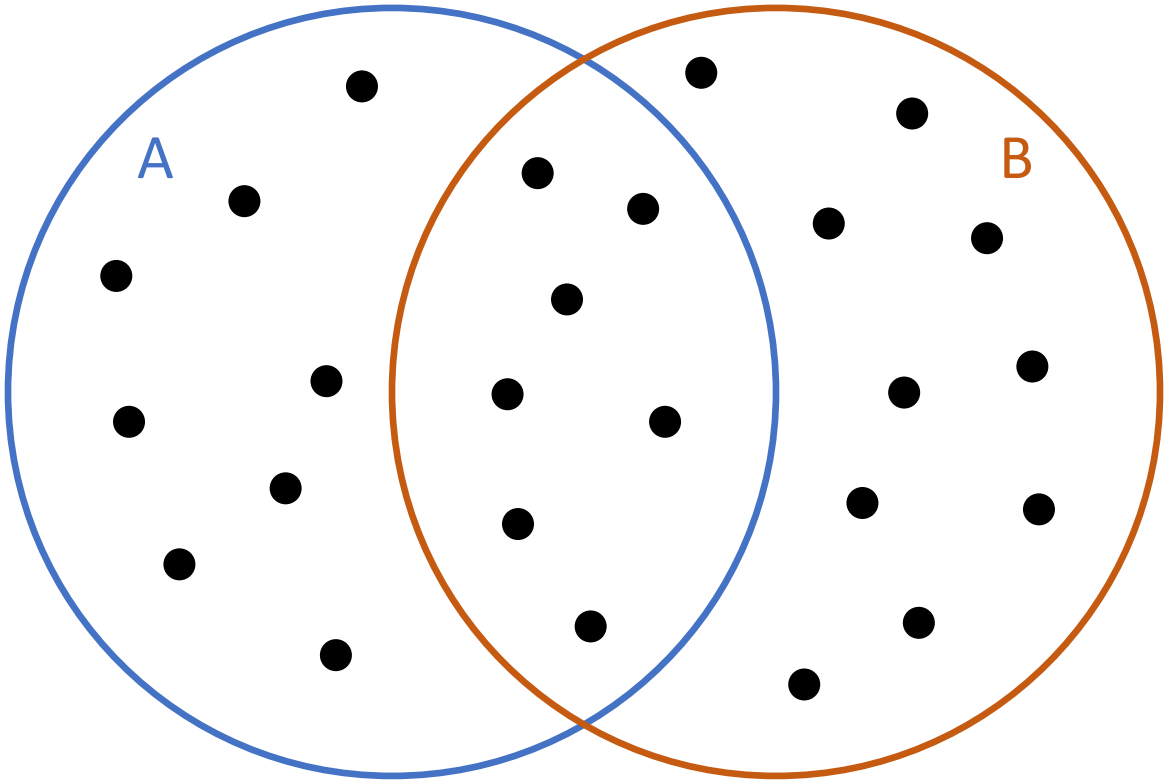
Comparing Two Sequences

- Mash: fast genome and metagenome distance estimation using MinHash (Ondov et al., 2016)
- Let A and B be two DNA sequences to compare
- Construct k-mer sets A and B
  - Assume $|A| = |B|$ for now (not true)
- Compare the sets somehow
- Not faster yet, but we'll get there…

Jaccard Index

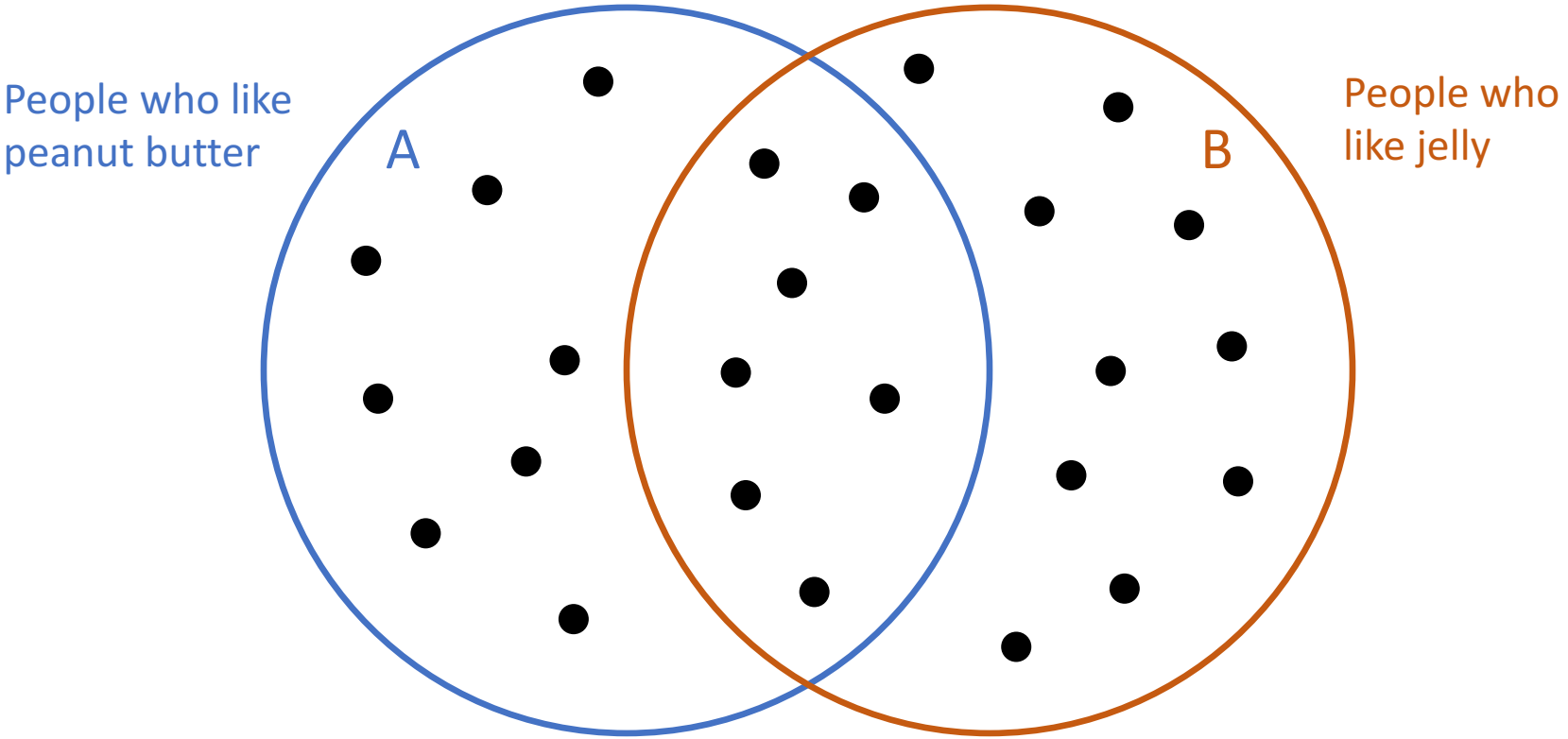
- Similarity between sets A and B
  - $\frac{|A \cap B|}{|A \cup B|}$
- Correlated with Average Nucleotide Identity (ANI)
  - Empirical support, but debatable

Jaccard Index: $\frac{|A \cap B|}{|A \cup B|}$



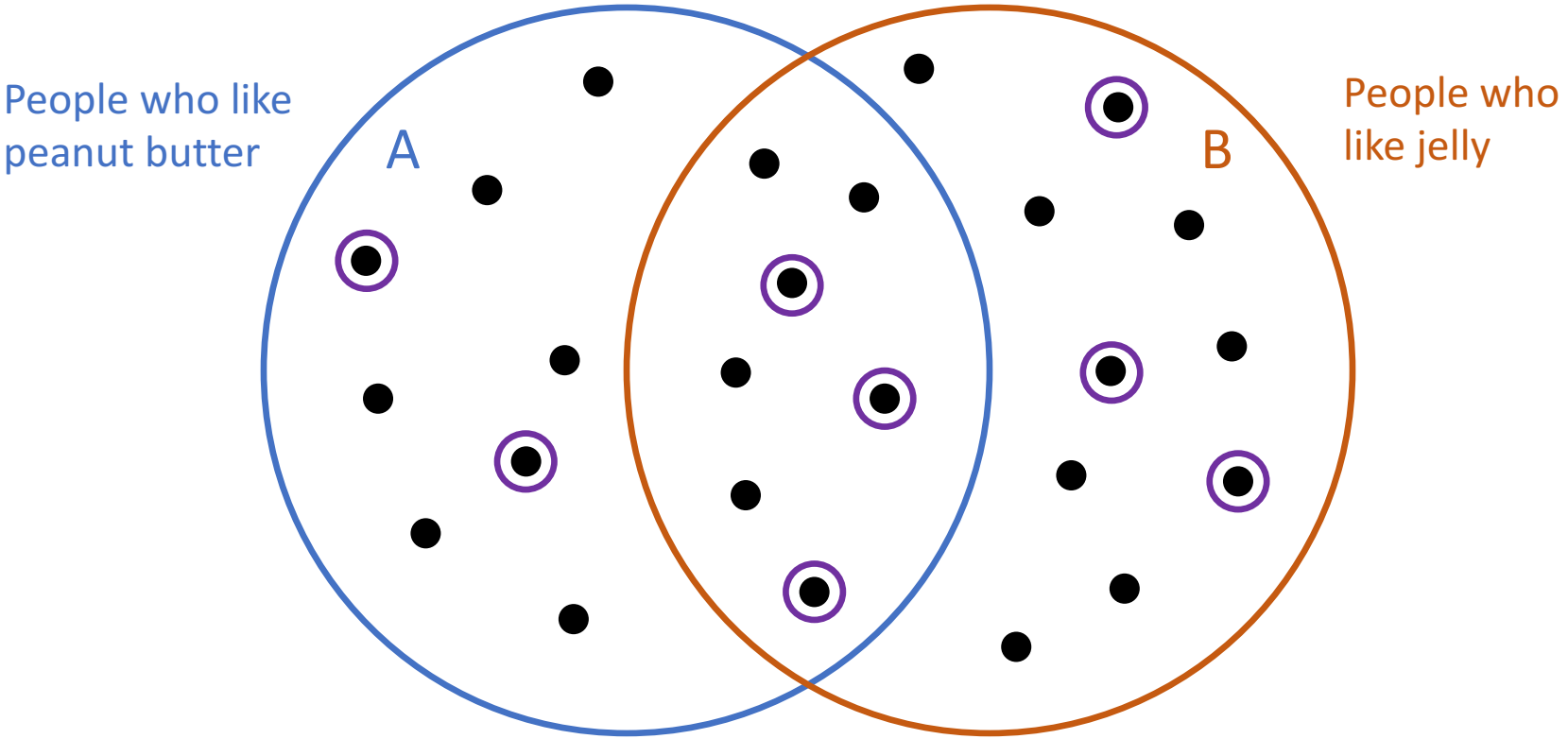
Jaccard Index: $\frac{|A \cap B|}{|A \cup B|}$

- What would you do if you were studying a population?



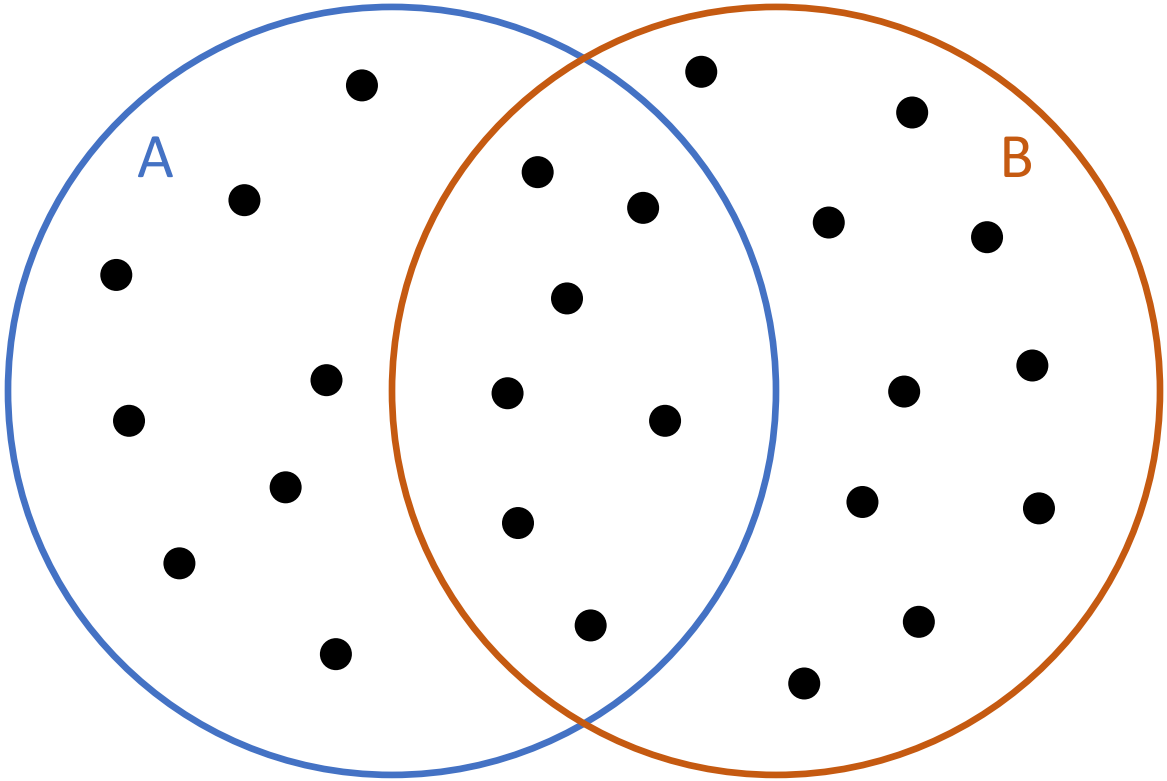
Jaccard Index: $\frac{|A \cap B|}{|A \cup B|}$

- What would you do if you were studying a population? Sample!



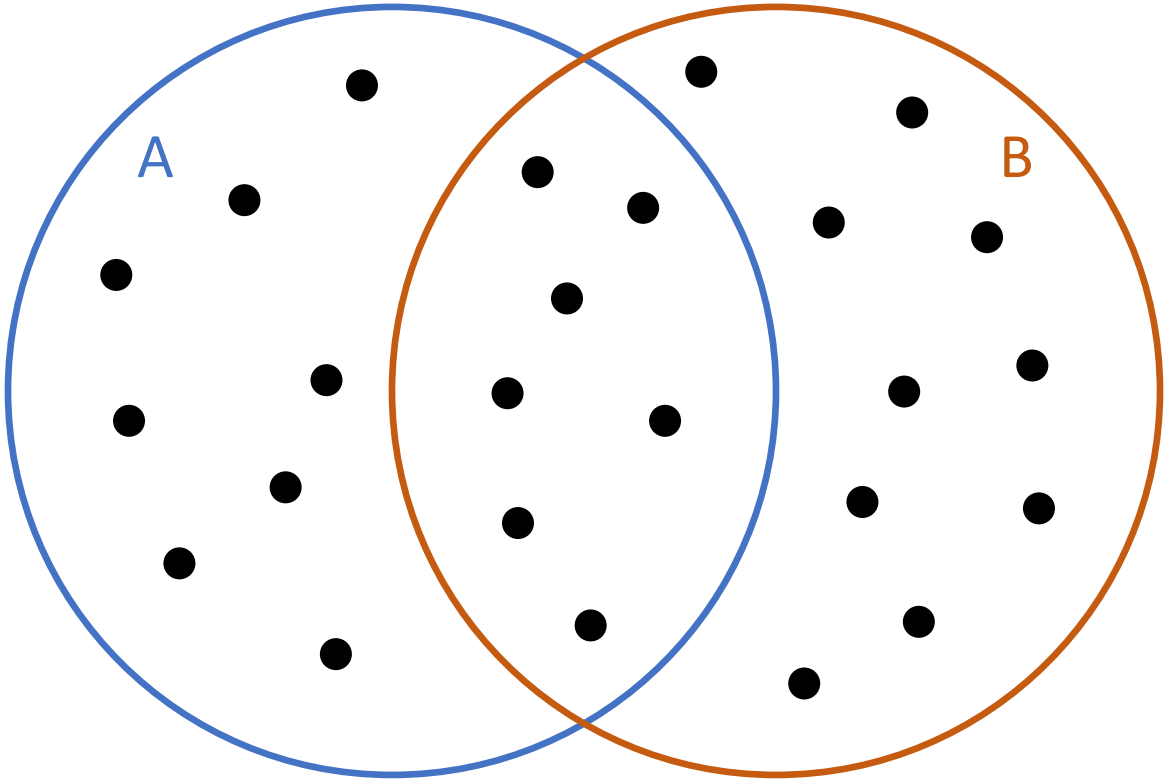
Sketch

- Small “fingerprint” of a data point (string)



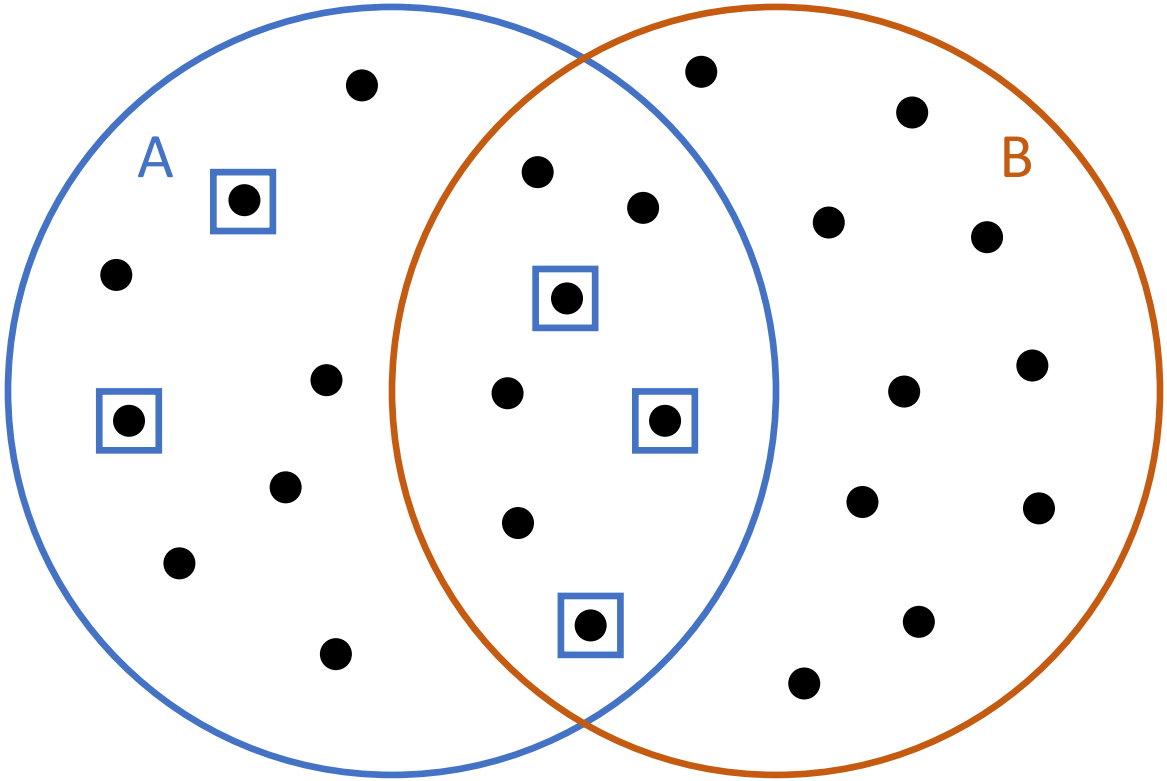
Warm-up: Naïve Sketch

- Sample each string independently (don't want to do N^2 sketches for comparing all pairs of N strings)



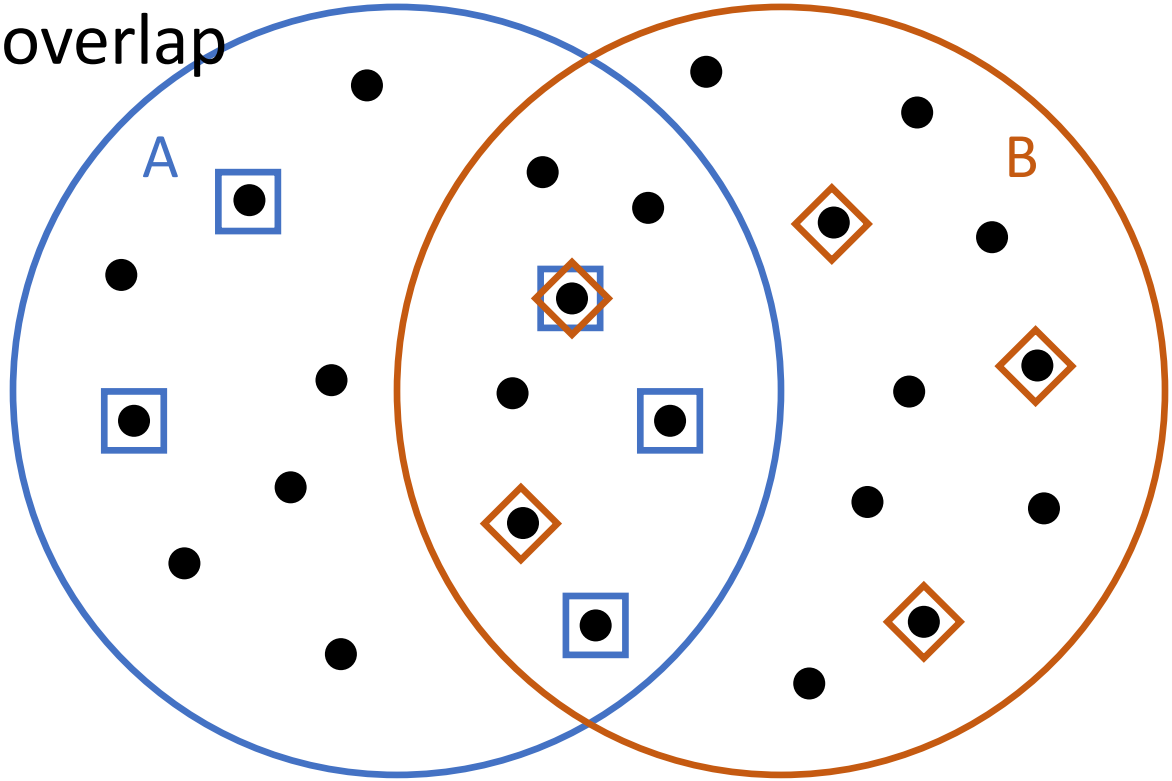
Warm-up: Naïve Sketch

- Sample each string independently (don't want to do N^2 sketches for comparing all pairs of N strings)



Warm-up: Naïve Sketch

- Sample each string independently (don't want to do N^2 sketches for comparing all pairs of N strings)
- Small overlap

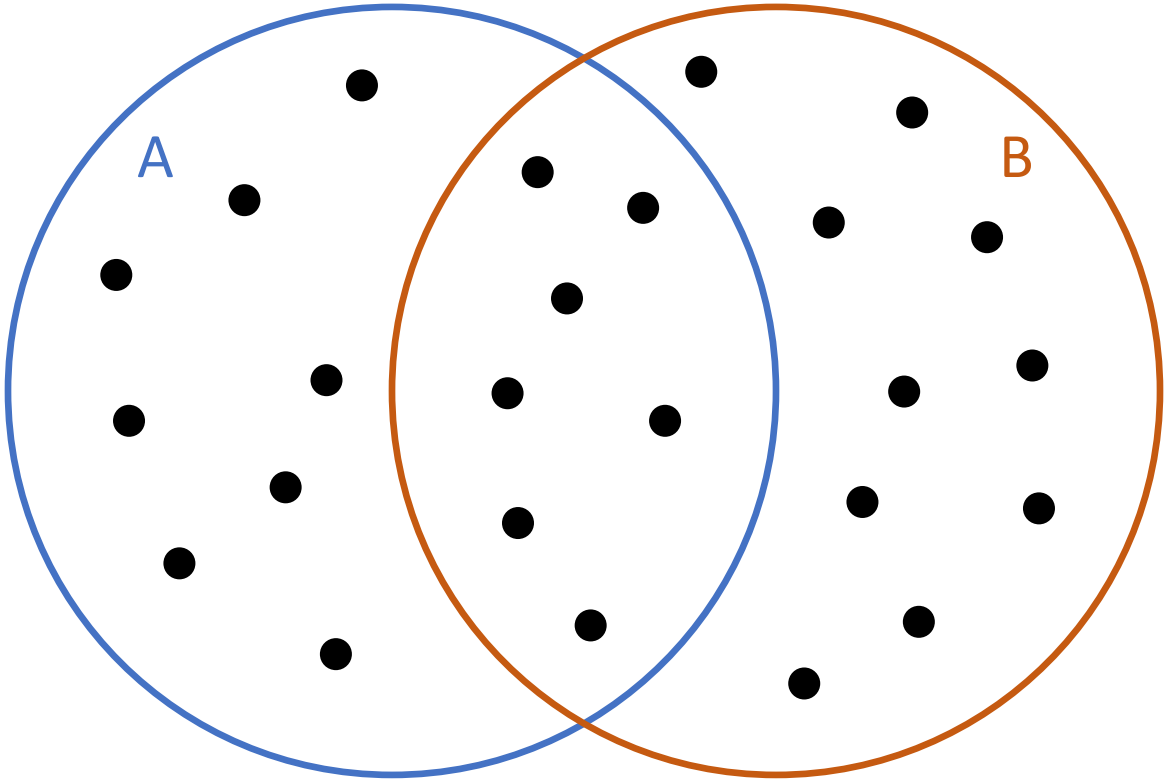


Minhashing/Bottom-d Sketch

- On the resemblance and containment of documents (Broder, 1997)
  - For comparing documents
- Hash each k-mer in a sequence
- Sketch $S(A)$: smallest d hash values in A
  - Or take min for each of d different hash function
- Use same hash function for $S(A)$ and $S(B)$
- Lets us sketch each string, but "simulate" sketching the union $S(A \cup B)$
- Canonical k-mers, A and B could be reverse comps

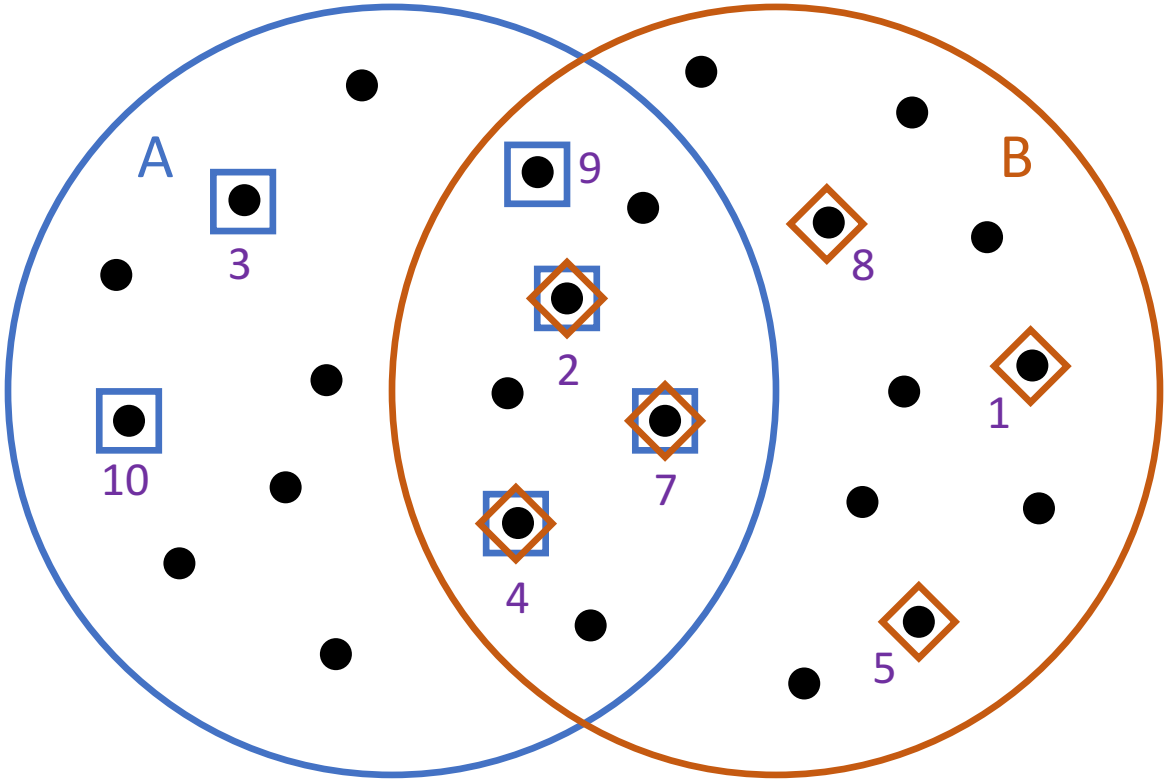
Minhashing/Bottom-d Sketch

- Sample smallest $d = 6$ hashes of k-mers in each set



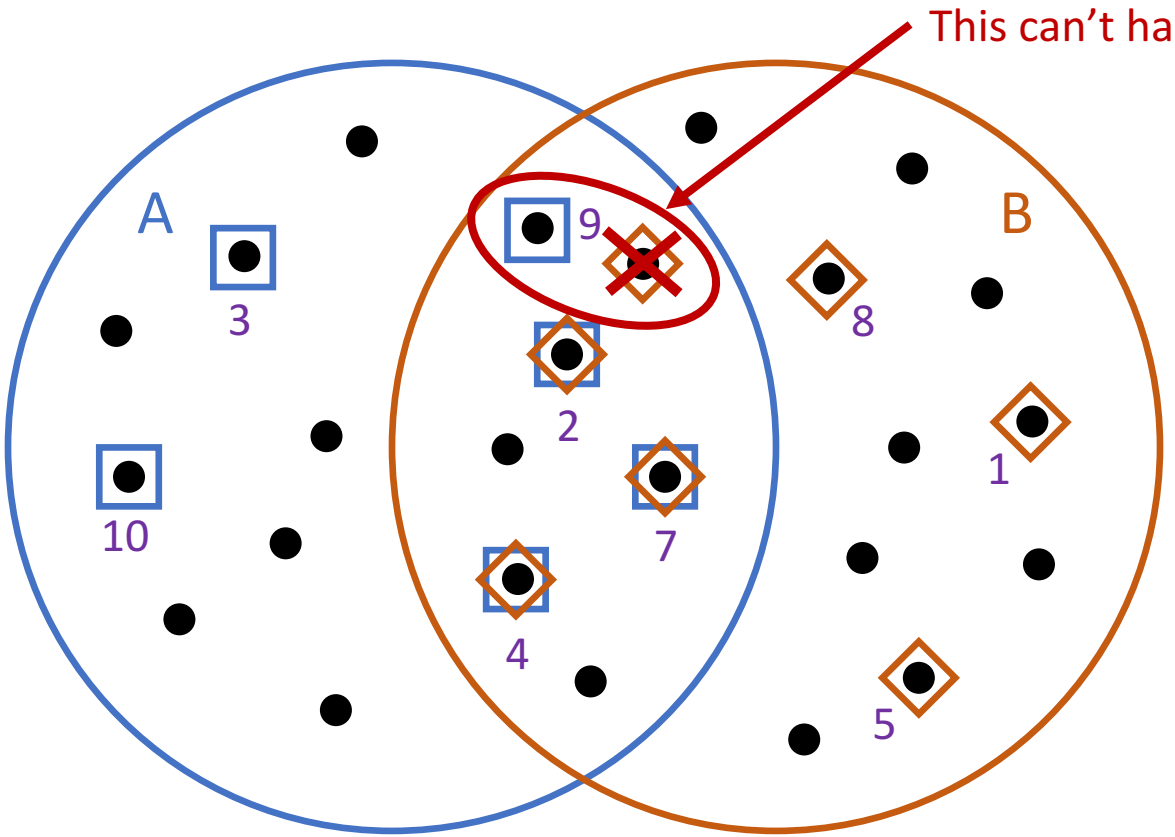
Minhashing/Bottom-d Sketch

- Sample smallest $d = 6$ hashes of k-mers in each set



Minhashing/Bottom-d Sketch

- Sample smallest $d = 6$ hashes of k-mers in each set



Comparing sketches

- Jaccard estimate j
  - $\frac{|A \cap B|}{|A \cup B|} \approx \frac{|S(A \cup B) \cap S(A) \cap S(B)|}{|S(A \cup B)|}$
- Get $S(A \cup B)$ by merge sort operation in $O(d)$ time
  - Merge until d unique hashes seen
  - Count number of matches $c = 3$
  - $j = \frac{c}{d}$
- Error of estimate is $\epsilon = \frac{1}{\sqrt{d}}$

$S(A)$	$S(A \cup B)$	$S(B)$
2	1	1
3	2	2
4	3	4
7	4	5
9	5	7
10	7	8

Building Bottom-d Sketch

- Takes $O(n \log d)$ time
  - Traverse string, hashing k-mers
  - Keep sorted list of smallest d
  - Check each new hash against max in list
  - $O(\log d)$ time to insert if necessary
- Actually expected time $O(n + d \log d \log n)$
  - Because Pr[ith hash gets inserted in list] $= \frac{d}{i}$
  - So effectively linear

Minhash parameters

- Probability some k-mer x appears in a random genome of length n
  - $\Pr[x \in A] \approx 1 - \left(1 - |\Sigma|^{-k}\right)^n$
    - Alphabet size $|\Sigma| = 4$
- For $k = 16, n = 3$Gbp:
  - Probability of a given 16-mer in a genome is ≈ 0.5
  - $\approx 25\%$ of 16-mers expected to be shared between two random 3 Gbp genomes
  - Too short k-mers can overestimate Jaccard, especially for distant genomes
  - Very long could underestimate, but less of an issue

Minhash parameters

- Value of k to achieve a desired probability q of seeing a given k-mer in sequence length n
  - $k \approx \left\lceil \log_{|\Sigma|} \left(\frac{n(1-q)}{q} \right) \right\rceil$
- 5 Mbp genome, $q = 0.01$, $k \approx 14$
- 3 Gbp genome, $q = 0.01$, $k \approx 19$
- Mash default: k = 21 and s = 1000
  - 8 kB per sketch

Mash distance

- Mash distance based on Jaccard estimate j
  - $-\frac{1}{k}\ln\frac{2j}{1+j}$
- Based on Poisson error model
- Implicitly uses average size of the two sets, penalizing sets of different size

Some related works

- Assembly overlaps
  - Assembling large genomes with single-molecule sequencing and locality-sensitive hashing (Berlin et al., 2015)
- Containment for different size sets
  - Improving Min Hash Via the Containment Index with Applications to Metagenomic Analysis (Koslicki and Zabeti, 2017)

Implementation

- MurmurHash3
- Open Bloom Filter Library
- Mash

Other Random Stuff

Other Random Stuff

Other Random Stuff

Fruit Fly Brains

- Locality Sensitive Hashing (LSH)
  - A neural algorithm for a fundamental computing problem (Dasgupta, Stevens, and Navlakha, 2017)
- Bloom filters
  - (Dasgupta, Sheehan, Stevens, and Navlakha, upcoming)
  - Have 3 special properties
    - Continuous-valued novelty
    - Distance sensitivity
    - Time sensitivity

Thanks!