Entity Resolution: Blocking

CompSci 590.03
Instructor: Ashwin Machanavajjhala

Duke UNIVERSITY

Recap: Entity Resolution

Problem of identifying and linking/grouping different manifestations of the same real world object.

Examples of manifestations and objects:

- Different ways of addressing (names, email addresses, FaceBook accounts) the same person in text.
- Web pages with differing descriptions of the same business.
- Different photos of the same object.
- …

Recap: Fellegi & Sunter Model [FS, Science '69]

- *r = (x,y)* is record pair, γ is comparison vector, *M* matches, *U* non-matches

- Decision rule $$R = \frac{P(\gamma \mid r \in M)}{P(\gamma \mid r \in U)}$$

$$R \geq t_l \Rightarrow r \rightarrow \text{Match}$$

$$t_l < R < t_u \Rightarrow r \rightarrow \text{Potential Match}$$

$$R \leq t_u \Rightarrow r \rightarrow \text{Non - Match}$$

- Naïve Bayes Assumption: $P(\gamma \mid r \in M) = \prod_i P(\gamma_i \mid r \in M)$

Duke UNIVERSITY

Outline

- Algorithms for Single Entity ER
  - Computing Pairwise Match scores
  - Blocking: Efficiently Identifying of Near-Duplicates
  - Correlation Clustering: Enforcing Transitivity Constraints
- Algorithms for Relational & Multi-Entity ER

SCALING ENTITY RESOLUTION

Outline

- Definition of Blocking
- Hash-based Blocking
  - Boolean functions over attributes
  - minHash: Locality Sensitive Hashing
- Neighborhood-based Blocking
  - Merge/Purge
  - Canopy Clustering

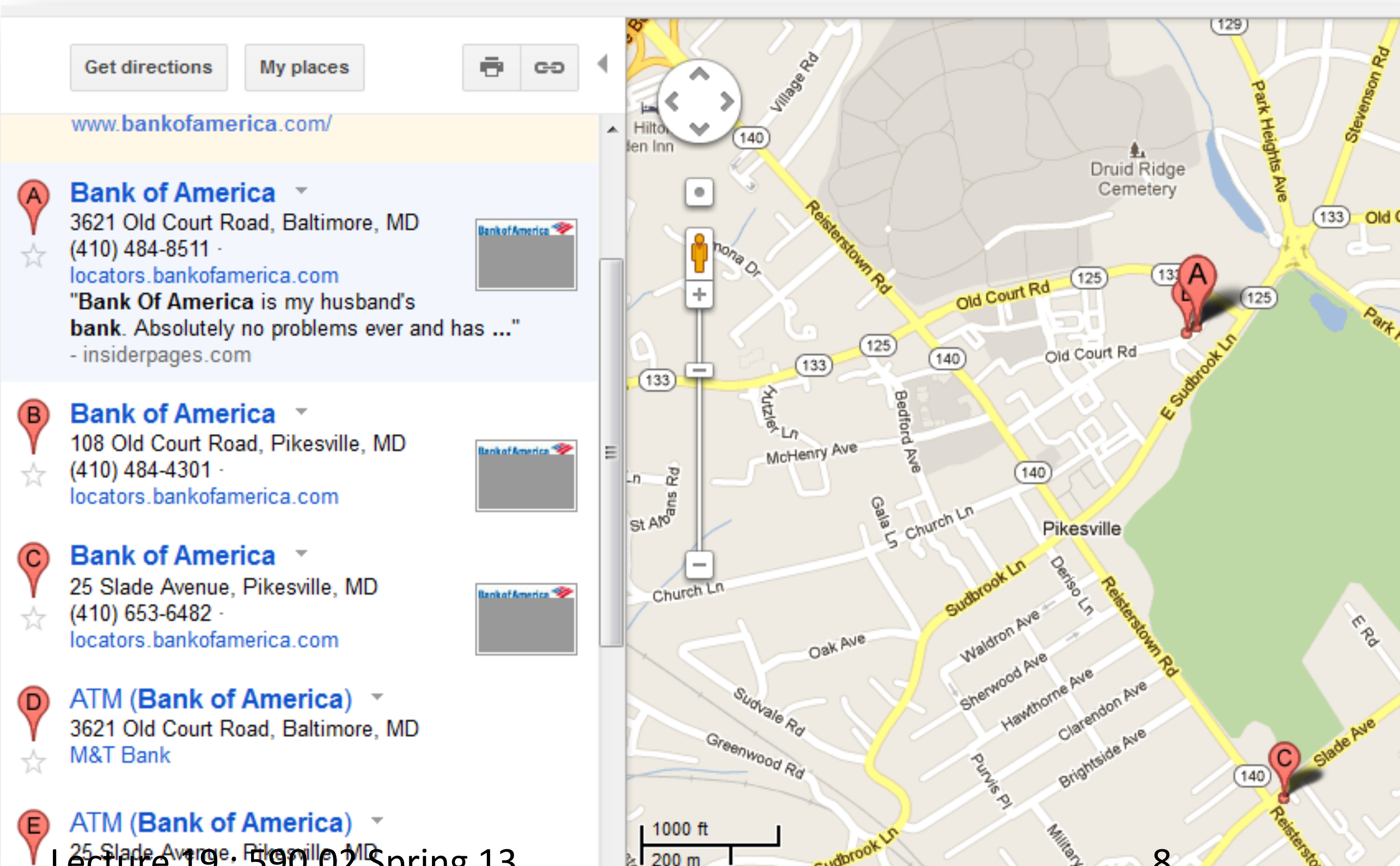
Blocking: Motivation

- Naïve pairwise: $|R|^2$ pairwise comparisons
  - 1000 business listings each from 1,000 different cities across the world
  - 1 trillion comparisons
  - 11.6 days (if each comparison is 1 μs)

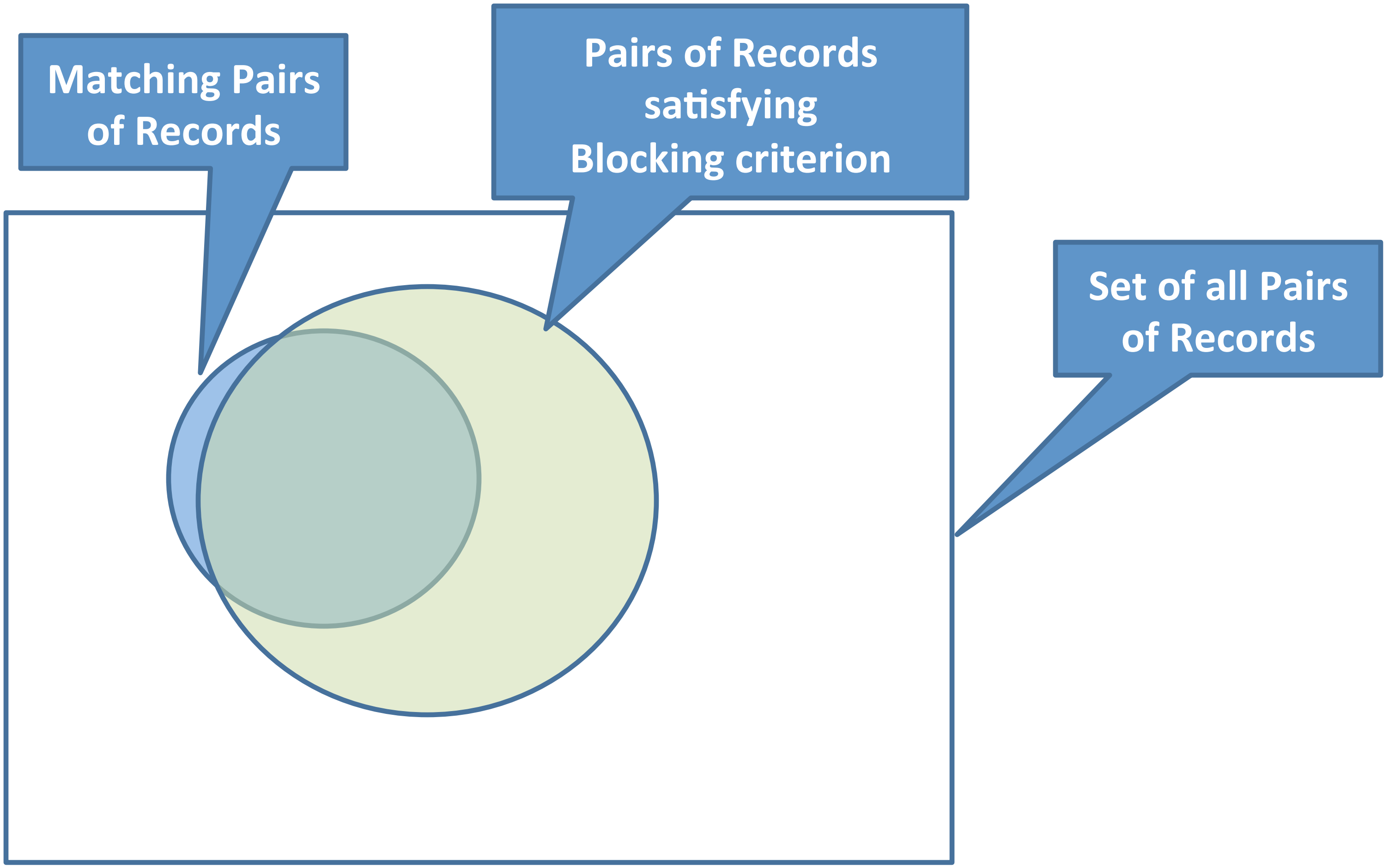
- Mentions from different cities are unlikely to be matches
  - **Blocking Criterion: City**
  - 1 billion comparisons
  - 16 minutes (if each comparison is 1 μs)

Blocking: Motivation

- Mentions from different cities are unlikely to be matches
  - May miss potential matches



Blocking: Motivation



Duke UNIVERSITY

Blocking: Problem Statement

Input: Set of records *R*

Output: Set of *blocks/canopies*

$$\{C_1, C_2, \ldots, C_k\}, where\ \forall_i C_i \subset R\ and \bigcup_i C_i = R$$

Intuition:

- *Only compare pairs of records that appear within each block / canopy*
- *Use a simple function (approximate distance) to generate canopies.*

Blocking: Problem Statement

$$\{C_1, C_2, \ldots, C_k\}, where\ \forall_i C_i \subset R\ and \bigcup_i C_i = R$$

Metrics:

- Efficiency (or reduction ratio) : $$\frac{number\ of\ pairs\ compared}{total\ number\ of\ pairs\ in\ R \times R}$$
$$= \frac{|\{(x,y) \mid \exists i\ C_i, s.t.\ \ x,y \in C_i\}|}{r(r-1)/2}$$

- Recall* (or pairs completeness) : $$\frac{number\ of\ true\ matches\ compared}{number\ of\ true\ matches\ in\ R \times R}$$

**Need to know ground truth in order to compute this metric*

Blocking: Problem Statement

Metrics:

- Efficiency (or reduction ratio) : $$\frac{number\ of\ pairs\ compared}{total\ number\ of\ pairs\ in\ R \times R}$$
- Recall* (or pairs completeness) : $$\frac{number\ of\ true\ matches\ compared}{number\ of\ true\ matches\ in\ R \times R}$$
- Precision* (or pairs quality) : $$\frac{number\ of\ true\ matches\ compared}{number\ of\ matches\ compared}$$
- Max Canopy Size: $$max_i\ |C_i|$$

**Need to know ground truth in order to compute this metric*

Blocking: Problem Statement

Input: Set of records *R*

Output: Set of *blocks/canopies*

$$\{C_1, C_2, \dots, C_k\}, where\ \forall_i C_i \subset R\ and \bigcup_i C_i = R$$

Variants:

- *Disjoint Blocking*: Each record appears in one block.

$$\forall_{i,j} C_i \cap C_j = \emptyset$$

- *Non-disjoint Blocking*: Records can appear in more than one block.

- Tradeoff recall for computation.

Duke UNIVERSITY

Outline

- Definition of Blocking
- Hash-based Blocking
  - Boolean functions over attributes
  - minHash: Locality Sensitive Hashing
- Neighborhood-based Blocking
  - Merge/Purge
  - Canopy Clustering

Blocking Algorithms 1

- Hash based blocking
  - Each block C_i is associated with a hash key h_i.
  - Record x is hashed to C_i if $hash(x) = h_i$.
  - Each hash function results in disjoint blocks.
  - *Easy parallel (MapReduce) implementation.*

Duke UNIVERSITY

Hash-based Blocking

- What is a *hash* function?
  - Deterministic function of attribute values
  - Boolean Functions over attribute values [Bilenko et al ICDM'06, Michelson et al AAAI'06, Das Sarma et al CIKM '12]
  - **minHash** (min-wise independent permutations) [Broder et al STOC'98]

Duke UNIVERSITY

Blocking Algorithms 1

- Hash based blocking
  - Each block C_i is associated with a hash key h_i.
  - Record x is hashed to C_i if $hash(x) = h_i$.
  - Each hash function results in disjoint blocks.
  - *Easy parallel (MapReduce) implementation.*

- Non-disjoint variant:
  - Each block is associated with a set of K hash keys.
  - Each record x is hashed using N hash functions.
  - Two records are in the same block if they share K out of N hash keys.
  - *MapReduce implementation?*

Simple Blocking: Inverted Index on a Key

Examples of blocking keys:

- First three characters of last name
- City + State + Zip
- Character or Token n-grams
- Minimum infrequent n-grams

Duke UNIVERSITY

Learning Optimal Blocking Functions

- Using one or more blocking keys may be insufficient
  - 2,376,206 American's shared the surname Smith in the 2000 US
  - NULL values may create large blocks.

- Solution: Construct blocking functions by combining simple functions

Complex Blocking Functions

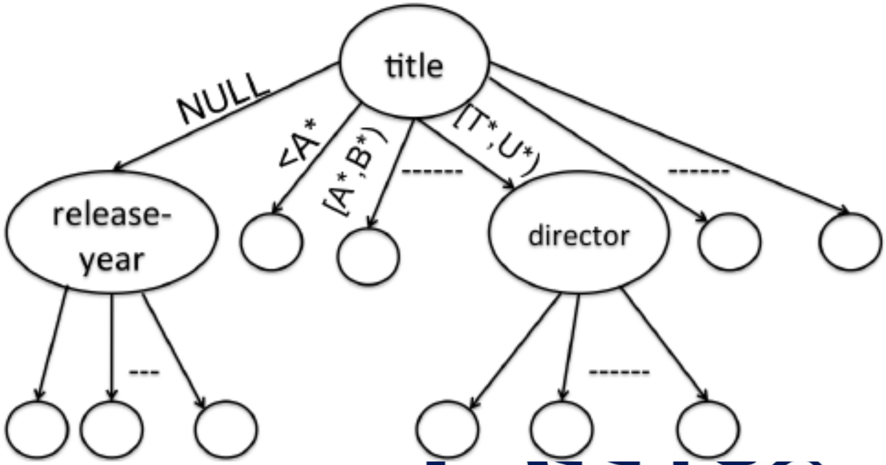
- Conjunction of functions
  - {City} AND {last four digits of phone}

- Chain-trees

  If ({City} = NULL or LA) **then** {last four digits of phone} AND {area code}

  else {last four digits of phone} AND {City}

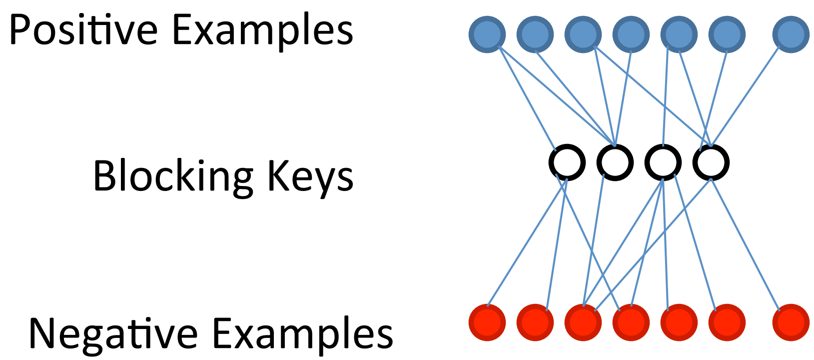
- BlkTrees



Learning an Optimal function [Bilenko et al ICDM '06]

- Find k blocking functions that eliminate the most non-matches, while retaining almost all matches.
  - Need a training set of positive and negative pairs

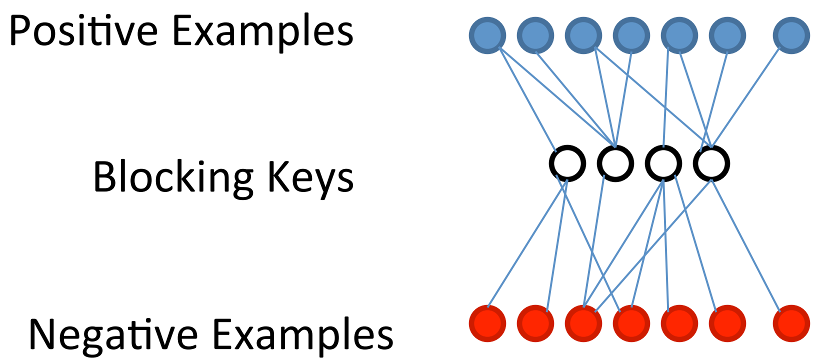
- Algorithm Idea: Red-Blue Set Cover



Pick k Blocking keys such that
(a) At most ε blue nodes are not covered
(b) Number of red nodes covered is minimized

Learning an Optimal function [Bilenko et al ICDM '06]

- Algorithm Idea: Red-Blue Set Cover



Pick k Blocking keys such that
(a) At most ε blue nodes are not covered
(b) Number of red nodes covered is minimized

- Greedy Algorithm:
  - Construct "good" conjunctions of blocking keys $\{p_1, p_2, \ldots\}$.
  - Pick k conjunctions $\{p_{i1}, p_{i2}, \ldots, p_{ik}\}$, such that the following is minimized

$$\frac{\text{number of new blue nodes covered by } p_{i_j}}{\text{number of red nodes covered by } p_{i_j}}$$

minHash (Minwise Independent Permutations)

- Let F_x be a set of features for mention *x*
  - (functions of) attribute values
  - character ngrams
  - optimal blocking functions …
- Let π be a random permutation of features in F_x
  - E.g., order imposed by a random hash function

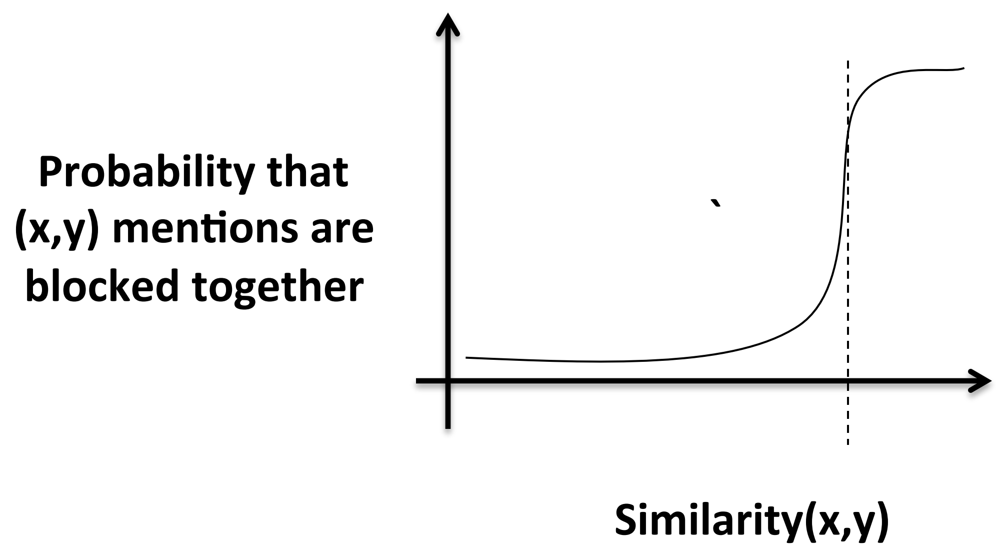
- *minHash(x)* = minimum element in F_x according to π

Why minHash works?

Surprising property: For a random permutation π,

$$P(minHash(x) = minhash(y)) = \frac{F_x \cap F_y}{F_x \cup F_y}$$

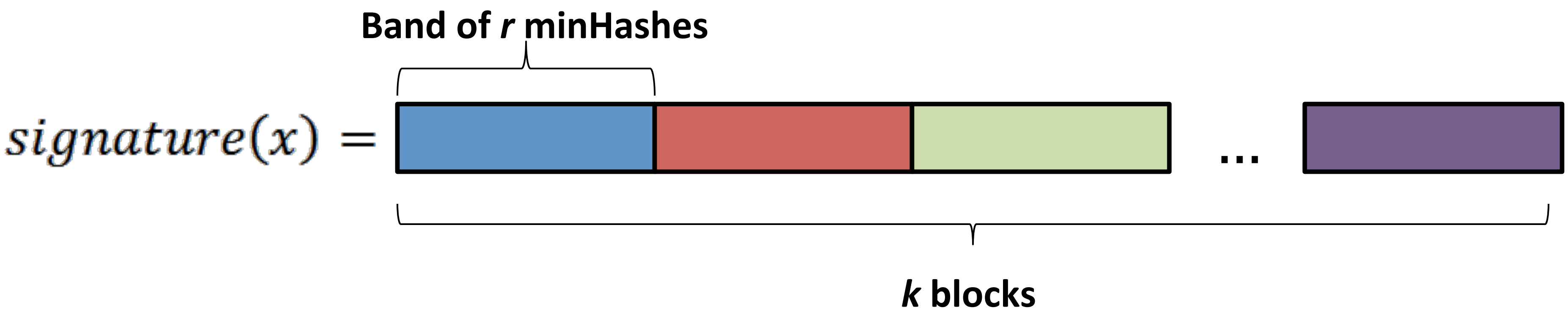
How to build a blocking scheme such that only pairs with Jacquard similarity > s fall in the same block (with high prob)?



Duke UNIVERSITY

Blocking using minHashes

- Compute minHashes using *r * k* permutations (hash functions)



- Signature's that match on ***1 out of k*** bands, go to the same block.

minHash Analysis

False Negatives: (missing matches)

P(pair x,y not in the same block with Jacquard sim = s) $= (1 - s^r)^k$

should be very low for high similarity pairs

False Positives: (blocking non-matches)

P(pair x,y in the same block with Jacquard sim = s) $= k \times s^r$

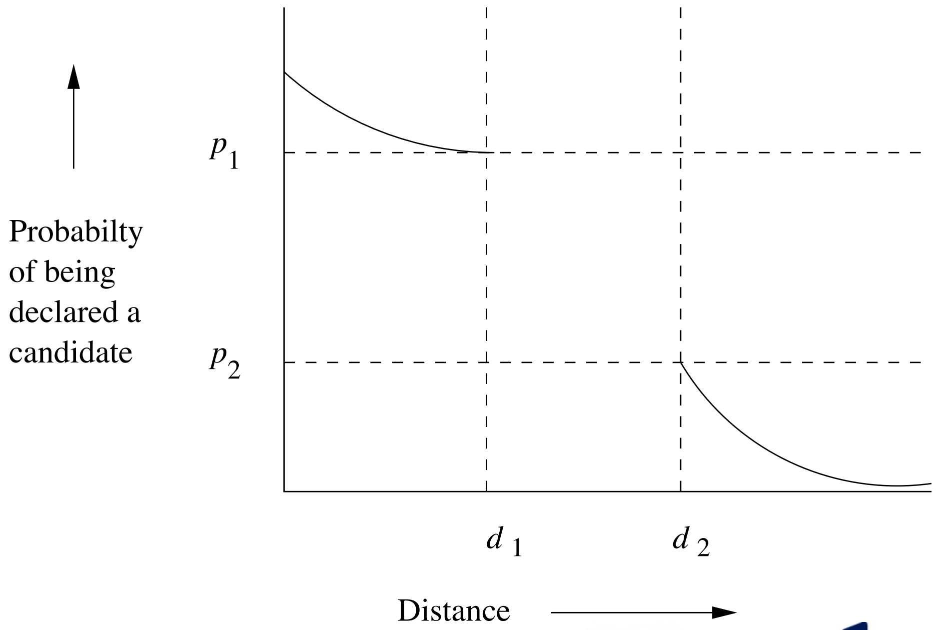
$r = 5, k = 20$

Sim(s)	P(not same block)
0.9	10^{-8}
0.8	0.00035
0.7	0.025
0.6	0.2
0.5	0.52
0.4	0.81
0.3	0.95
0.2	0.994
0.1	0.9998

Locality Sensitive Hashing Functions

Let d1 and d2 be two distances. A family of functions **F** is said to be *(d1, d2, p1, p2)-sensitive* if for all f in **F**,

- If d(x,y) < d1,
  then P[f(x) = f(y)] > p1
- If d(x,y) > d2,
  then P[f(x) = f(y)] < p2



Locality sensitive family for Jaccard distance

- minHash is one example of locality sensitive family that can strongly distinguish pairs that are close from pairs that are far.

- The family of minHash functions is a (d1, d2, 1-d1, 1-d2)-sensitive family for any d1, d2.

Amplifying a Locality-sensitive family

- AND construction:
  - Construct a new family F' consisting of r members of F
  - f in F' = {f1, f2, …, fr}
  - f(x) = f(y) iff for all i, fi(x) = fi(y)
  - If F is (d1, d2, p1, p2)-sensitive, then F' is (d1, d2, $p1^r$, $p2^r$)-sensitive

- OR construction:
  - Construct a new family F' consisting of b members of F
  - f in F' = {f1, f2, …, fb}
  - f(x) = f(y) iff there exists i, fi(x) = fi(y)
  - If F is (d1, d2, p1, p2)-sensitive,
    then F' is (d1, d2, $1-(1-p1)^b$, $1-(1-p2)^b$)-sensitive

Duke UNIVERSITY

Example

- Suppose F is (0.2, 0.6, 0.8, 0.4)-sensitive.
- We use AND-construction with r= 4 to create F1
- We use OR-construction with b=4 to create F2

- F2 is (0.2, 0.6, $1-(1-0.8^4)^4$, $1-(1-0.4^4)^4$)
  = (0.2, 0.6, 0.875, 0.0985)-sensitive

Duke UNIVERSITY

LSH for Hamming distance

- Given two vectors x, y
- Hamming distance h(x,y) = number of positions where x and y are different

- minHash: (d1, d2, 1-d1/d, 1-d2/d)-sensitive

LSH for Cosine Distance

- Cosine Distance: angle between two vectors

- Locality sensitive function **F**:
  Pick a random vector vf.
  f(x) = f(y) is x.vf and y.vf have the same sign.

- **F** is (d1, d2, (180-d1)/180, d2/180)-sensitive

- Another method:
  Generate v in $\{-1, +1\}^d$ (d is the dimensionality of x)
  f(x) = f(y) is x.vf and y.vf have the same sign.

Summary of Hash-based Blocking

- Complex boolean functions can be built to optimize recall using a training set of matches and non-matches

- Locality sensitive hashing functions can strongly distinguish pairs that are close from pairs that are far.

- AND and OR construction help amplify the distinguishing capability of locality sensitive functions.

Outline

- Definition of Blocking
- Hash-based Blocking
  - Boolean functions over attributes
  - minHash: Locality Sensitive Hashing
- Neighborhood-based Blocking
  - Merge/Purge
  - Canopy Clustering

Duke UNIVERSITY

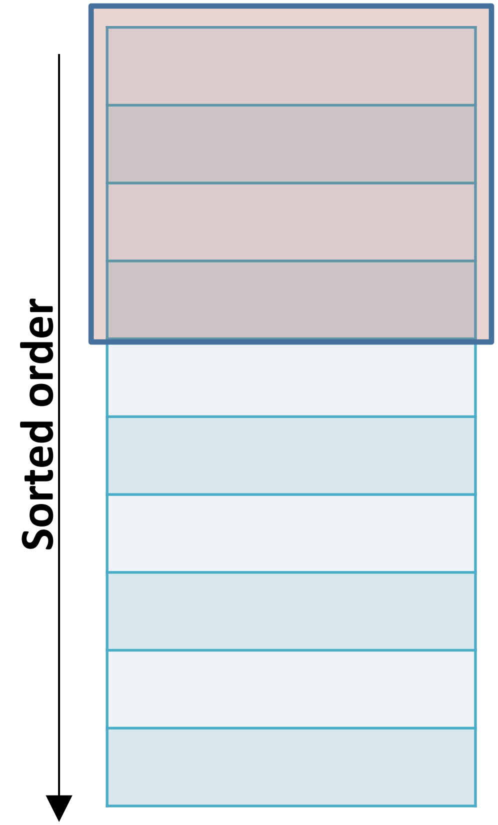
Blocking Algorithms 2

- Pairwise Similarity/Neighborhood based blocking
  - Nearby nodes according to a similarity metric are clustered together
  - Results in non-disjoint canopies.

- Techniques
  - Sorted Neighborhood Approach [Hernandez et al SIGMOD'95]
  - Canopy Clustering [McCallum et al KDD'00]

Sorted Neighborhood [Hernandez et al SIGMOD'95]

- Compute a **Key** for each record.
- **Sort** the records based on the key.
- **Merge**: Check whether a record matches with *(w-1)* previous records.
  - Implementation?
- Perform multiple passes with different keys



Canopy Clustering [McCallum et al KDD'00]

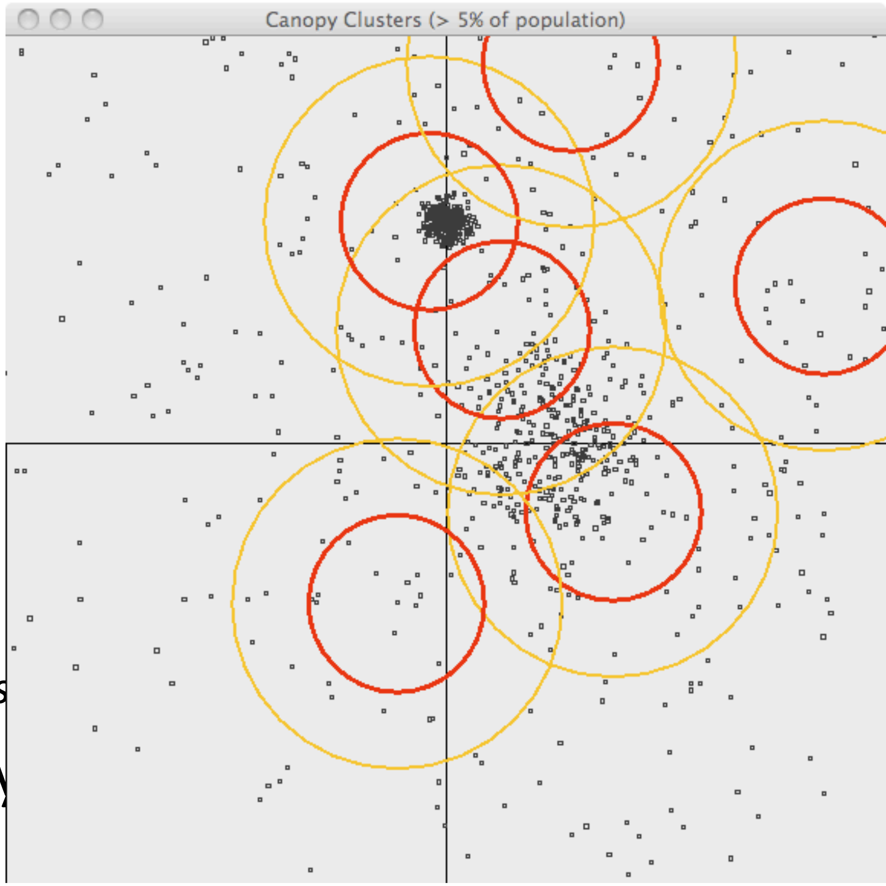
Input: Mentions M,

$d(x,y)$, a distance metric,

thresholds $T_1 > T_2$

Algorithm:

1. Pick a random element x from M
2. Create new canopy C_x using mentions y s.t. $d(x,y) < T_1$
3. Delete all mentions y from M s.t. $d(x,y) < T_2$ *(from consideration in this*
4. Return to Step 1 if M is not empty



Summary of Blocking

- $O(|R|^2)$ pairwise computations can be prohibitive.
  - Blocking eliminates comparisons on a large fraction of non-matches.
- Hash-based Blocking:
  - Construct (one or more) hash keys from features
  - Records not matching on any key are not compared.
- Neighbohood based Blocking:
  - Form overlapping canopies of records based on similarity.
  - Only compare records within a cluster.

Duke UNIVERSITY