CPSC 340: Machine Learning and Data Mining

Finding Similar Items

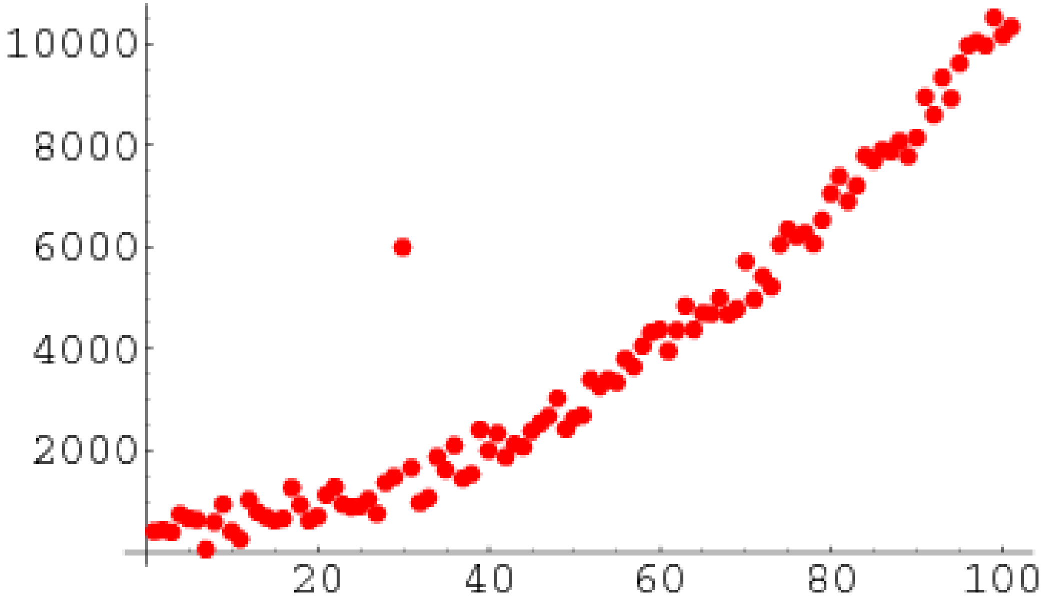
Fall 2017

Admin

- Assignment 1 is due tonight.
  - 1 late day to hand in Monday, 2 late days for Wednesday.
- Assignment 2 will be up soon.
  - Start early.
- We'll start using gradients and linear algebra next week:
  - Many people get lost when we get to this material.
  - If you aren't comfortable with these, start reviewing/practicing!

Last Time: Outlier Detection

- We discussed outlier detection:
  - Identifying “unusually” different objects.
  - Hard to precisely define.



- We discussed 3 common approaches:
  - Fit a model, see if points fit the model.
  - Plot the data, and look for weird points.
  - Cluster the data, and see if points don’t cluster.

Distance-Based Outlier Detection

- Most outlier detection approaches are based on distances.
- Can we skip model/plot/clustering and just measure distances?
  - How many points lie in a radius ‘r’?
  - What is distance to k^{th} nearest neighbour?

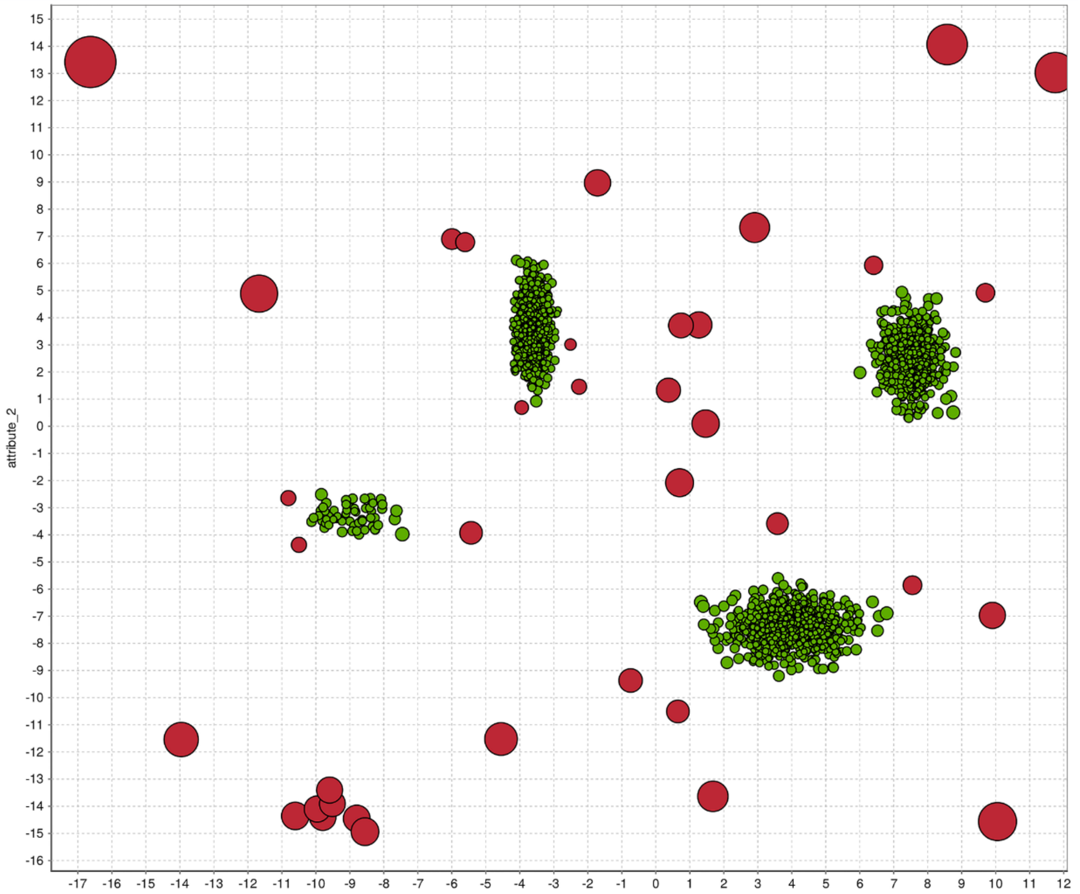
- UBC connection (first paper on this topic):

Algorithms for Mining Distance-Based Outliers in Large Datasets

Edwin M. Knorr and Raymond T. Ng
Department of Computer Science
University of British Columbia

Global Distance-Based Outlier Detection: KNN

- KNN outlier detection:
  - For each point, compute the average distance to its KNN.
  - Sort the set of ‘n’ average distances.
  - Choose the biggest values as outliers.
    - Filter out points that are far from their KNNs.

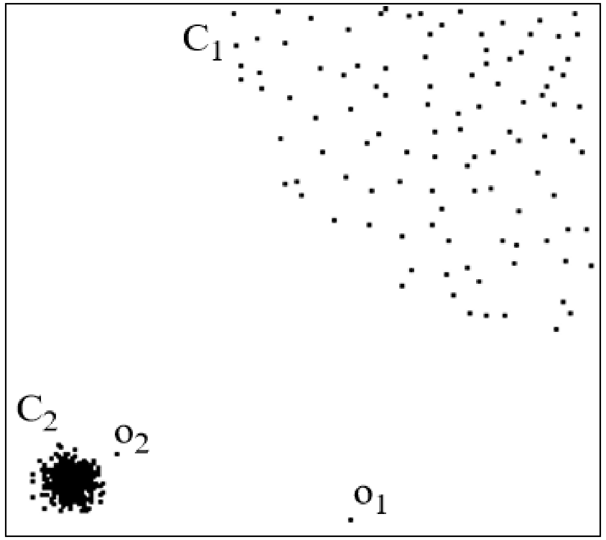


- Goldstein and Uchida [2016]:
  - Compared 19 methods on 10 datasets.
  - KNN best for finding “global” outliers.
  - “Local” outliers best found with local distance-based methods…

http://journals.plos.org/plosone/article?id=10.1371%2Fjournal.pone.0152173

Local Distance-Based Outlier Detection

- As with density-based clustering, problem with differing densities:



- Outlier o_2 has similar density as elements of cluster C_1.
- Basic idea behind local distance-based methods:
  - Outlier o_2 is "relatively" far compared to its neighbours.

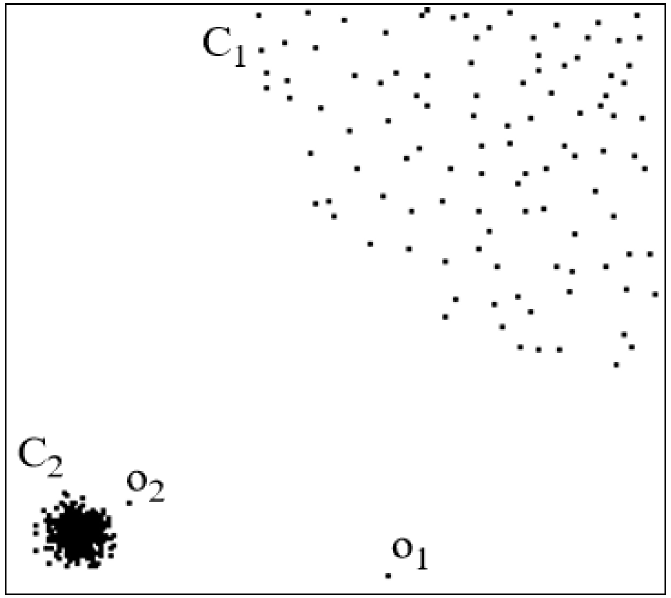
http://www.dbs.ifi.lmu.de/Publikationen/Papers/LOF.pdf

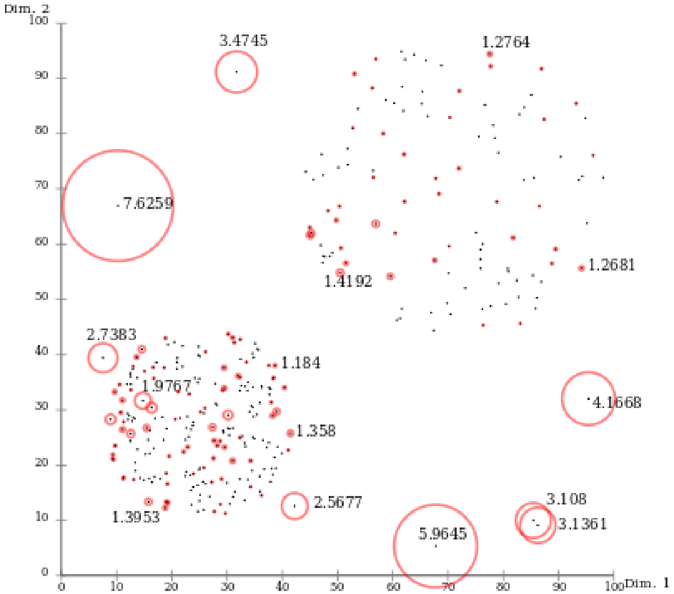
Local Distance-Based Outlier Detection

- "Outlierness" ratio of example 'i':

$$\frac{\text{average distance of 'i' to its KNNs}}{\text{average distance of \underline{neighbours} of 'i' to their KNNs}}$$

- If outlierness > 1, x_i is further away from neighbours than expected.

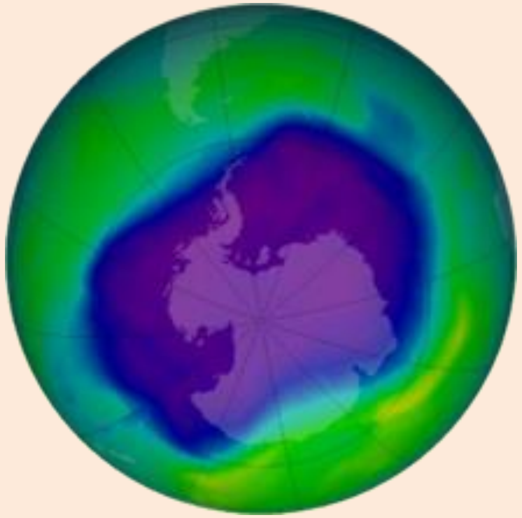




http://www.dbs.ifi.lmu.de/Publikationen/Papers/LOF.pdf
https://en.wikipedia.org/wiki/Local_outlier_factor

Problem with Unsupervised Outlier Detection

- Why wasn't the hole in the ozone layer discovered for 9 years?



- Can be hard to decide when to report an outler:
  - If you report too many non-outliers, users will turn you off.
  - Most antivirus programs do not use ML methods (see ["base-rate fallacy“]())

https://en.wikipedia.org/wiki/Ozone_depletion

Supervised Outlier Detection

- Final approach to outlier detection is to use supervised learning:
    - $y_i = 1$ if x_i is an outlier.
    - $y_i = 0$ if x_i is a regular point.

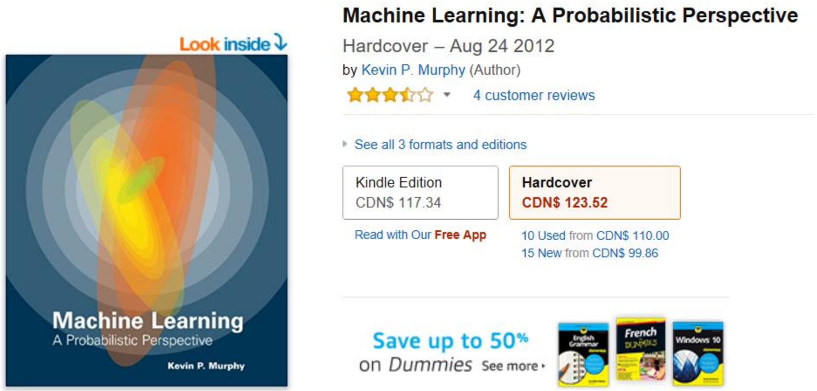
- We can use our methods for supervised learning:
  - We can find very complicated outlier patterns.
  - Classic credit card fraud detection methods used decision trees.

- But it needs supervision:
  - We need to know what outliers look like.
  - We may not detect new “types” of outliers.

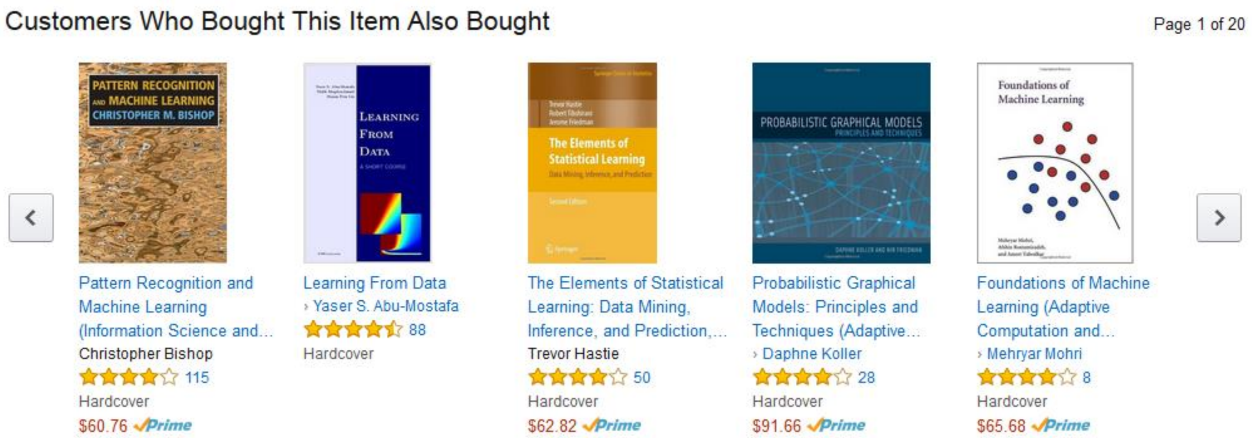
(pause)

Motivation: Product Recommendation

- A customer comes to your website looking to buy at item:



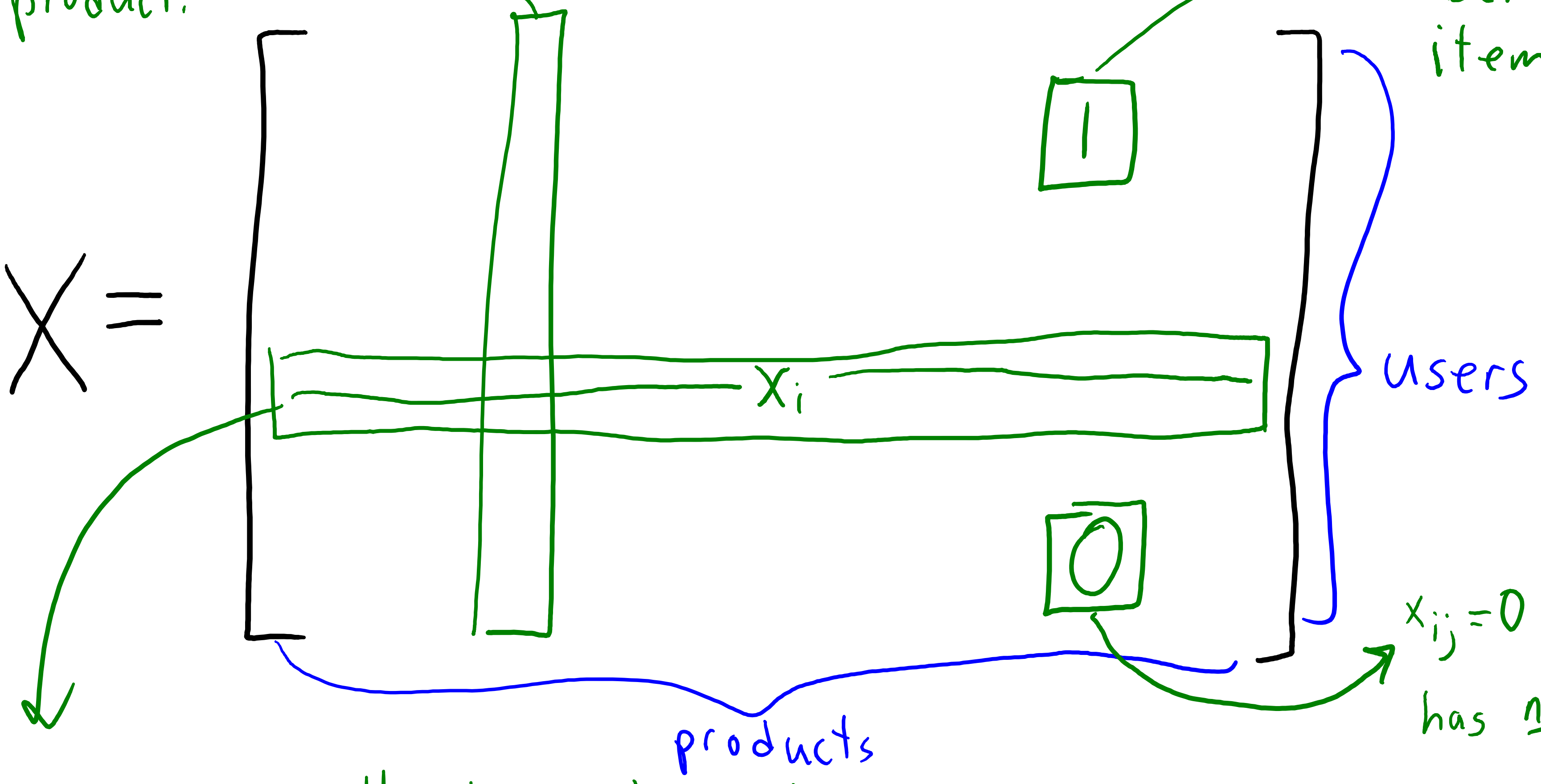
- You want to find similar items that they might also buy:



User-Product Matrix

Column gives all users that bought product.

$X_{ij} = 1$ means user 'i' bought item 'j'!

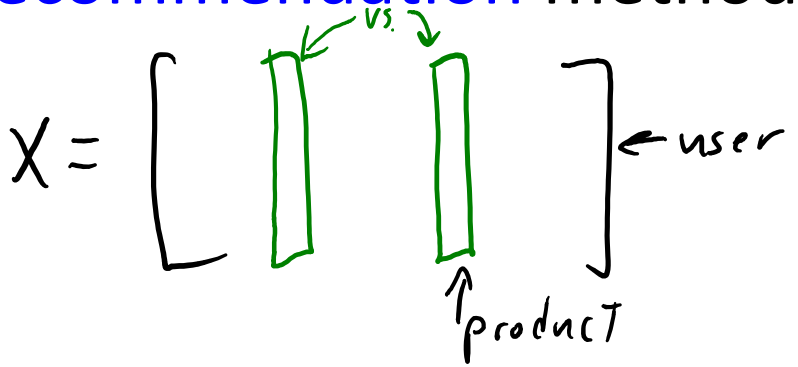


$x_{ij} = 0$ means user 'i' has not buy item 'j'

Row x_i gives all items bought by user 'i'. By convention, x_i is a $d \times 1$ column vector.

Amazon Product Recommendation

- Amazon product recommendation method:



- Return the KNNs across columns.
  - Find 'j' values minimizing $||x^i - x^j||$.
  - Products that were bought by similar users.

- But first divide each column by its norm, $x^i/||x^i||$.
  - This is called normalization.
  - Reflects whether product is bought by many people or few people.

Amazon Product Recommendation

- Consider this user-item matrix:

$$X = \begin{array}{c} \\ \text{John} \\ \text{Paul} \\ \text{George} \\ \text{Ringo} \\ \text{Yoko} \end{array} \begin{array}{c} \begin{array}{cccccc} \text{Product 1} & \text{Product 2} & \text{Product 3} & \text{Product 4} & \text{Product 5} & \text{Product 6} \end{array} \\ \begin{bmatrix} 1 & 1 & 1 & 1 & 0 & 1 \\ 1 & 0 & 1 & 0 & 1 & 0 \\ 1 & 0 & 1 & 0 & 1 & 1 \\ 1 & 0 & 1 & 0 & 1 & 1 \\ 1 & 1 & 0 & 1 & 0 & 0 \end{bmatrix} \end{array}$$

- Product 1 is most similar to Product 3 (bought by lots of people).
- Product 2 is most similar to Product 4 (also bought by John and Yoko).
- Product 3 is equally similar to Products 1, 5, and 6.
  - Does not take into account that Product 1 is more popular than 5 and 6.

Amazon Product Recommendation

- Consider this user-item matrix (normalized):

$$X = \begin{array}{c} \\ \text{John} \\ \text{Paul} \\ \text{George} \\ \text{Ringo} \\ \text{Yoko} \end{array} \begin{array}{cccccc} \text{Product 1} & \text{Product 2} & \text{Product 3} & \text{Product 4} & \text{Product 5} & \text{Product 6} \end{array}$$

$$\begin{bmatrix} 1/\sqrt{5} & 1/\sqrt{2} & 1/\sqrt{4} & 1/\sqrt{2} & 0 & 1/\sqrt{3} \\ 1/\sqrt{5} & 0 & 1/\sqrt{4} & 0 & 1/\sqrt{3} & 0 \\ 1/\sqrt{5} & 0 & 1/\sqrt{4} & 0 & 1/\sqrt{3} & 1/\sqrt{3} \\ 1/\sqrt{5} & 0 & 1/\sqrt{4} & 0 & 1/\sqrt{3} & 1/\sqrt{3} \\ 1/\sqrt{5} & 1/\sqrt{2} & 0 & 1/\sqrt{2} & 0 & 0 \end{bmatrix}$$

- Product 1 is most similar to Product 3 (bought by lots of people).
- Product 2 is most similar to Product 4 (also bought by John and Yoko).
- Product 3 is most similar to Product 1.
  - Normalization means it prefers the popular items.

Cost of Finding Nearest Neighbours

- With ‘n’ users and ‘d’ products, finding KNNs costs O(nd).
  - Not feasible if ‘n’ and ‘d’ are in the millions.

- It’s faster if the user-product matrix is sparse: O(z) for z non-zeroes.
  - But ‘z’ is still enormous in the Amazon example.

Closest-Point Problems

- We've seen a lot of "closest point" problems:
  - K-nearest neighbours classification.
  - K-means clustering.
  - Density-based clustering.
  - Hierarchical clustering.
  - KNN-based outlier detection.
  - Outlierness ratio.
  - Amazon product recommendation.

- How can we possibly apply these to Amazon-sized datasets?

But first the easy case: “Memorize the Answers”

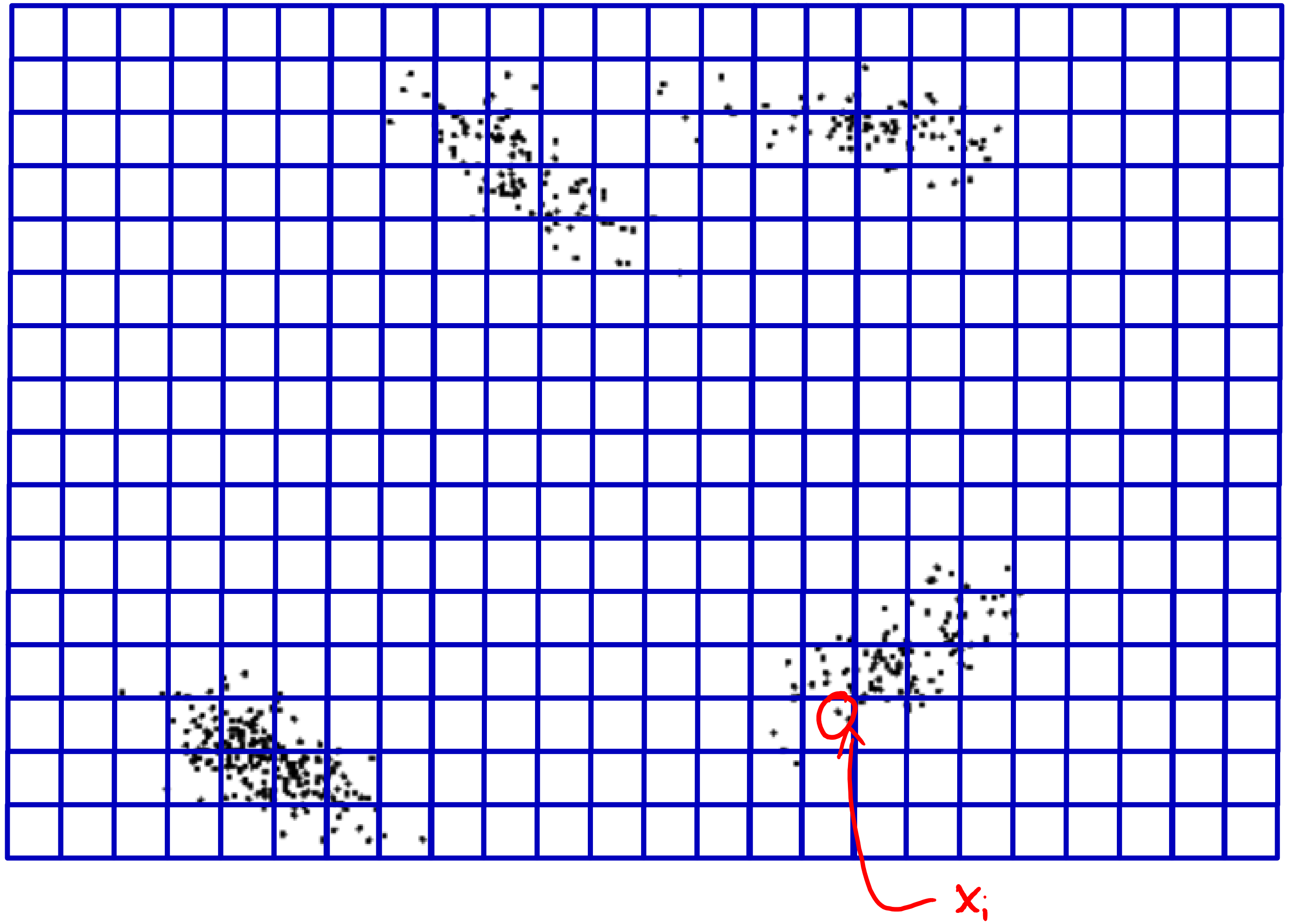
- Easy case: you have a limited number of possible test examples.
  - E.g., you will always choose an existing product (not arbitrary features).
- In this case, just memorize the answers:
  - For each test example, compute all KNNs and store pointers to answers.
  - At test time, just return a set of pointers to the answers.
- The answers are called an inverted index, queries now cost O(k).
  - Needs an extra O(nk) storage.

Grid-Based Pruning

- Assume we want to find objects within a distance of ‘r’ of point x_i.

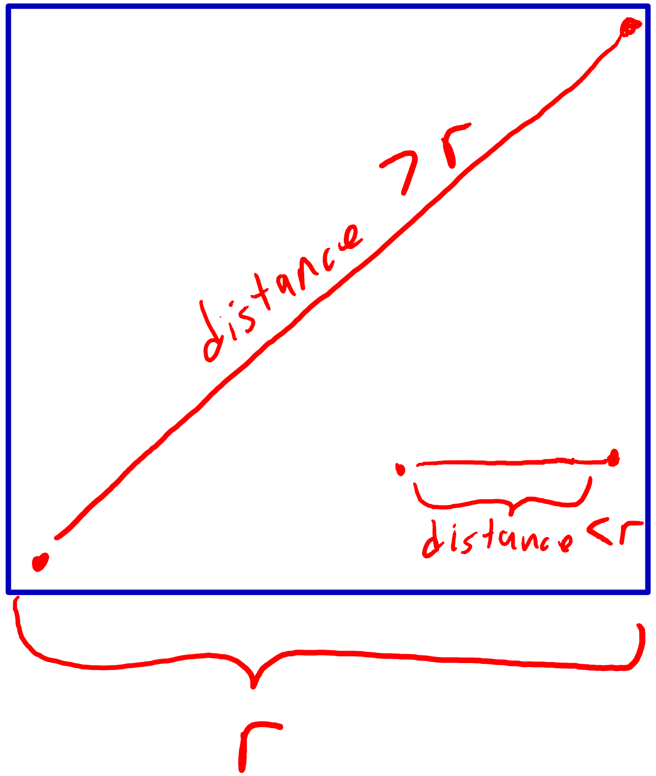
Divide space into squares of length r.

Hash examples based on squares:
Hash[“64,76”] = $\{x_3, x_{70}\}$
(Dict in Python/Julia)



Grid-Based Pruning

- Which squares do we need to check?

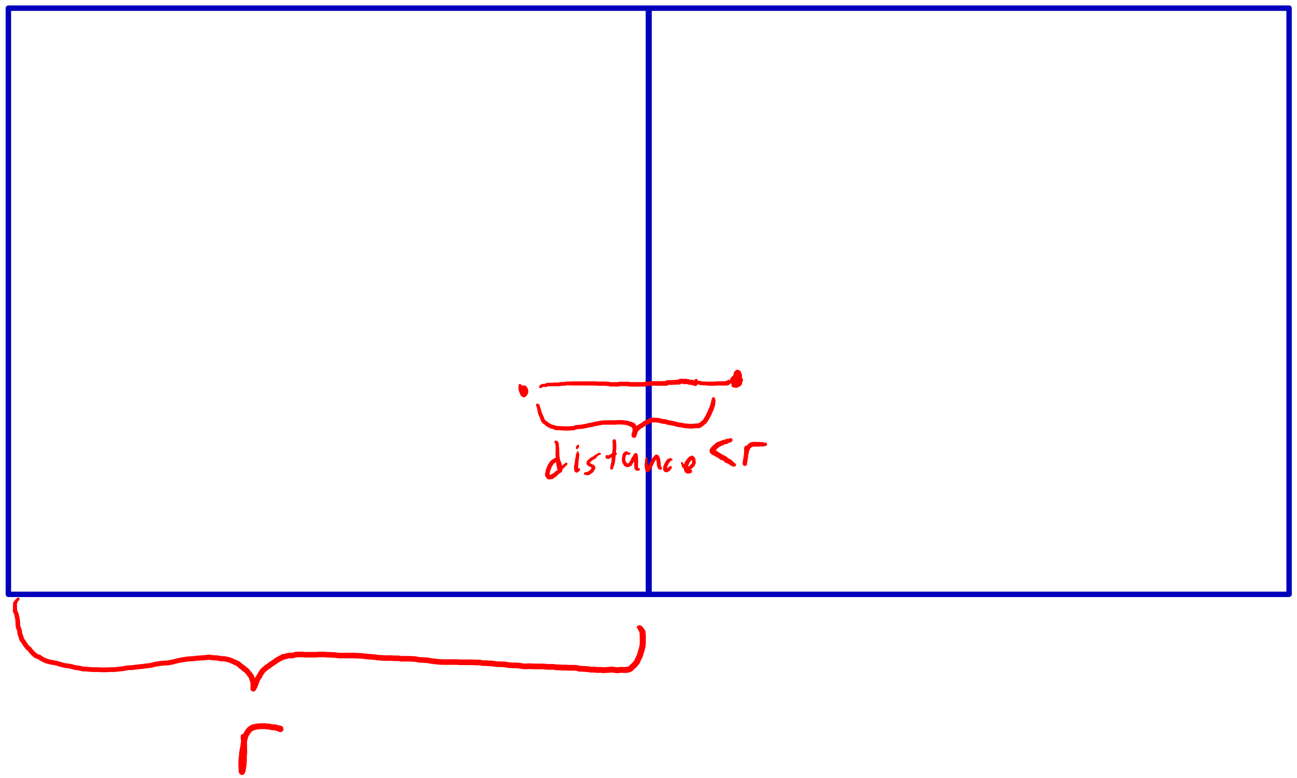


Points in same square can have distance less than ‘r’.

Grid-Based Pruning

- Which squares do we need to check?

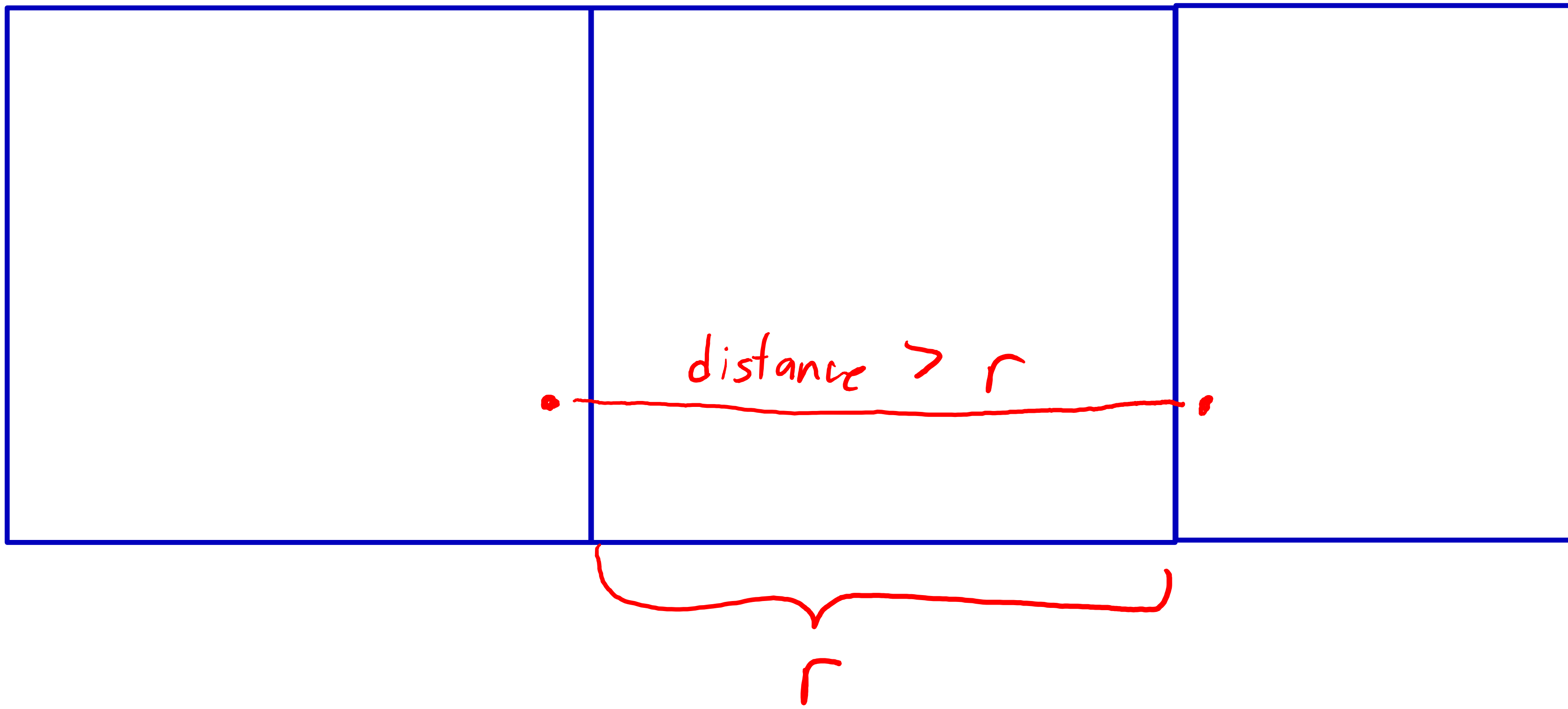
Points in adjacent squares can have distance less than distance 'r'.



Grid-Based Pruning

- Which squares do we need to check?

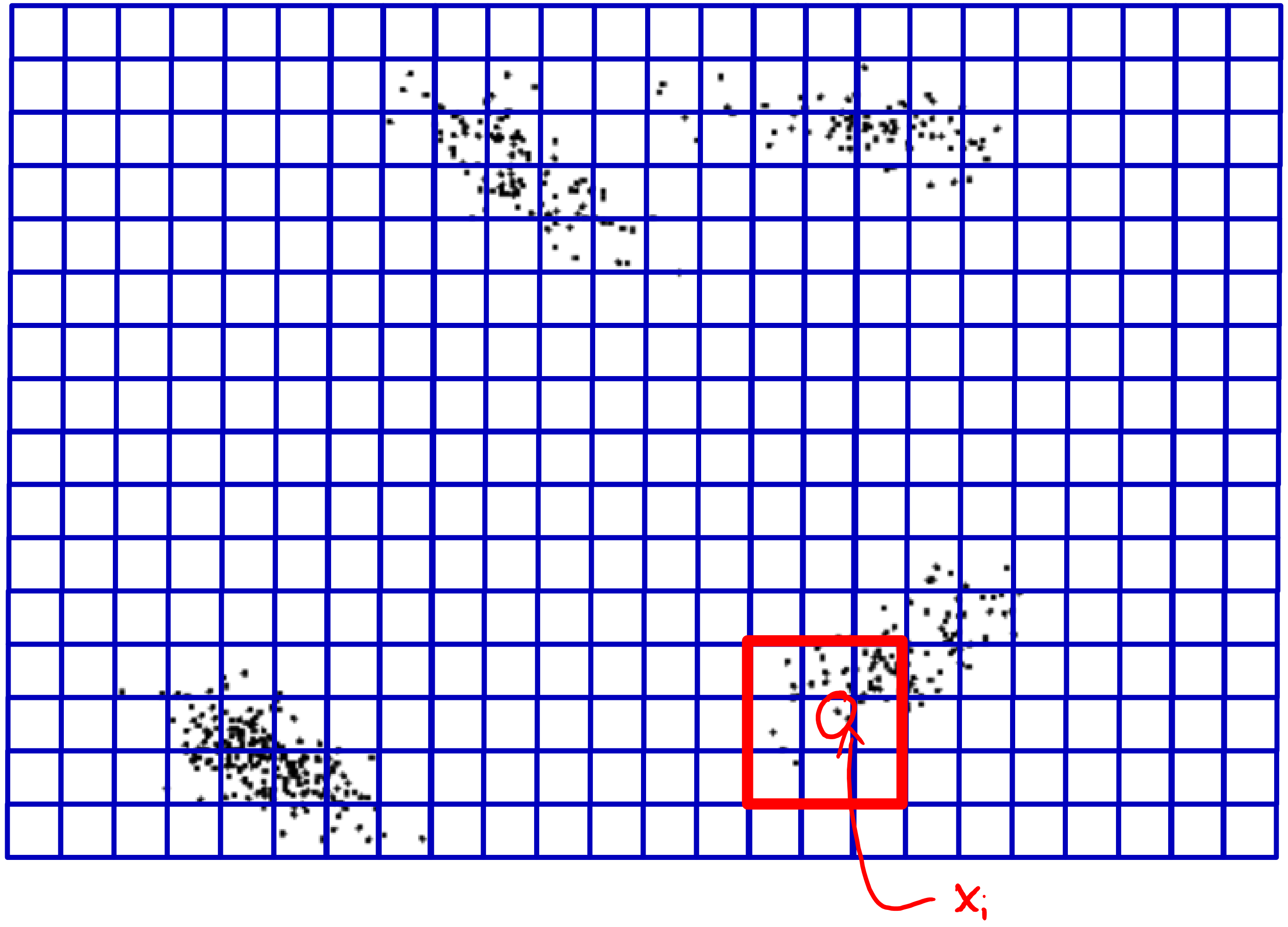
Points in non-adjacent squares must have distance more than ‘r’.



Grid-Based Pruning

- Assume we want to find objects within a distance of 'r' of point x_i.

Divide space into squares of length r.



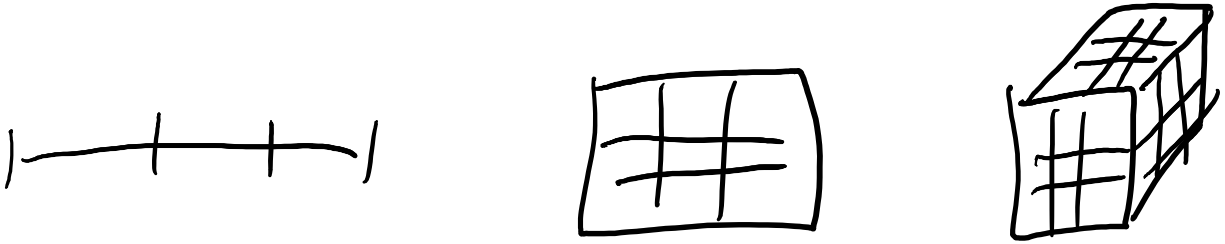
Hash examples based on squares:
Hash["64,76"] = $\{x_3, x_{70}\}$
(Dict in Python/Julia)

Only need to check points in same and adjacent squares.

Grid-Based Pruning Discussion

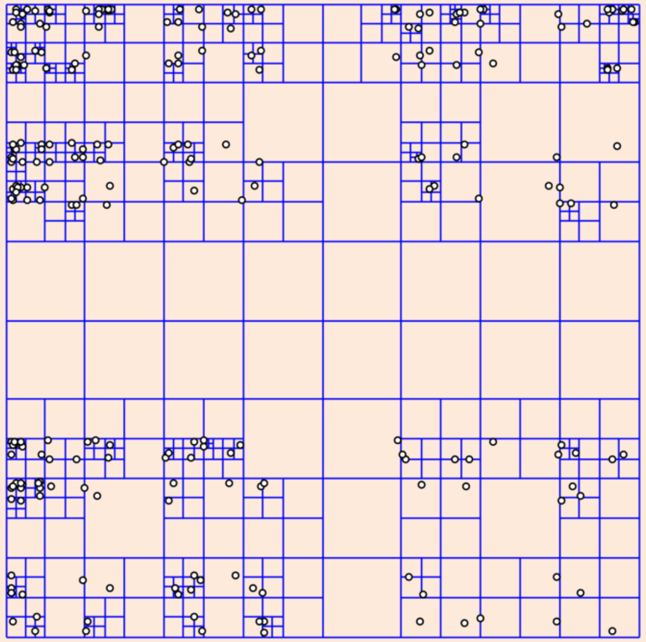
- Similar ideas can be used for other "closest point" calculations.
  - Can be used with any norm.
  - If you want KNN, can use need grids of multiple sizes.

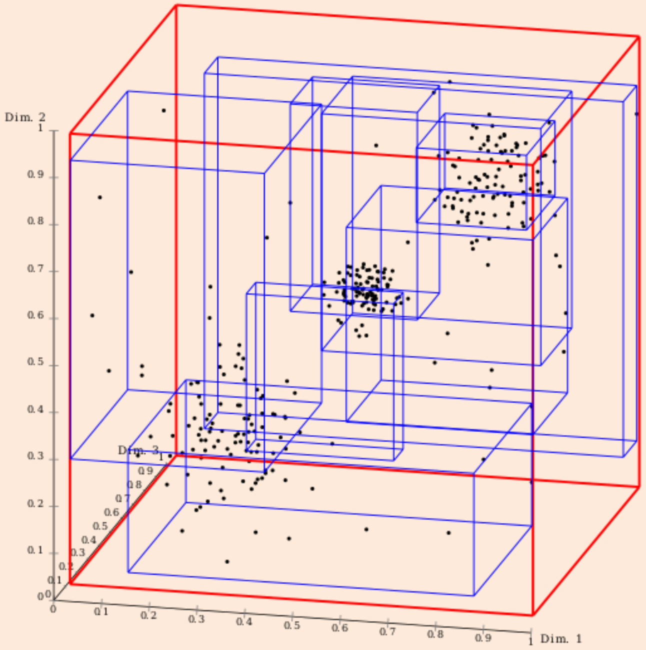
- But we have the "curse of dimensionality":
  - Number of adjacent regions increases exponentially:
    - 2 with d=1, 8 with d=2, 26 with d=3, 80 with d=4, 252 with d=5, 3^d-1 in d-dimension.

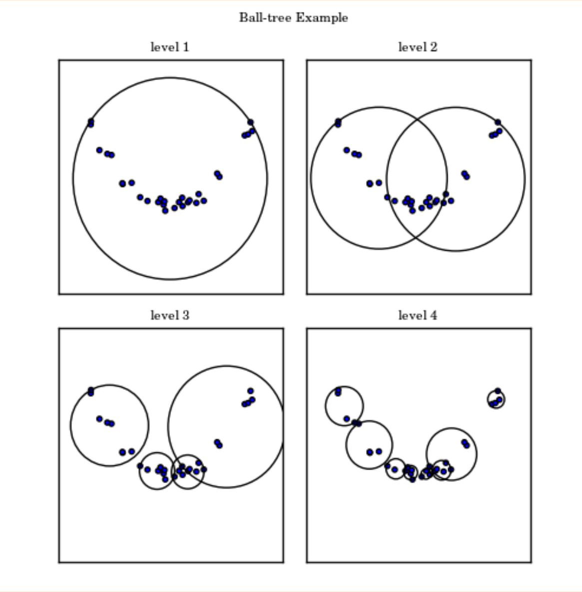


Grid-Based Pruning Discussion

- Better choices of regions:
  - Quad-trees.
  - Kd-trees.
  - R-trees.
  - Ball-trees.







- Works better than squares, but worst case is still exponential.

https://en.wikipedia.org/wiki/Quadtree
https://en.wikipedia.org/wiki/R-tree
http://www.astroml.org/book_figures/chapter2/fig_balltree_example.html

Approximate Nearest Neighbours

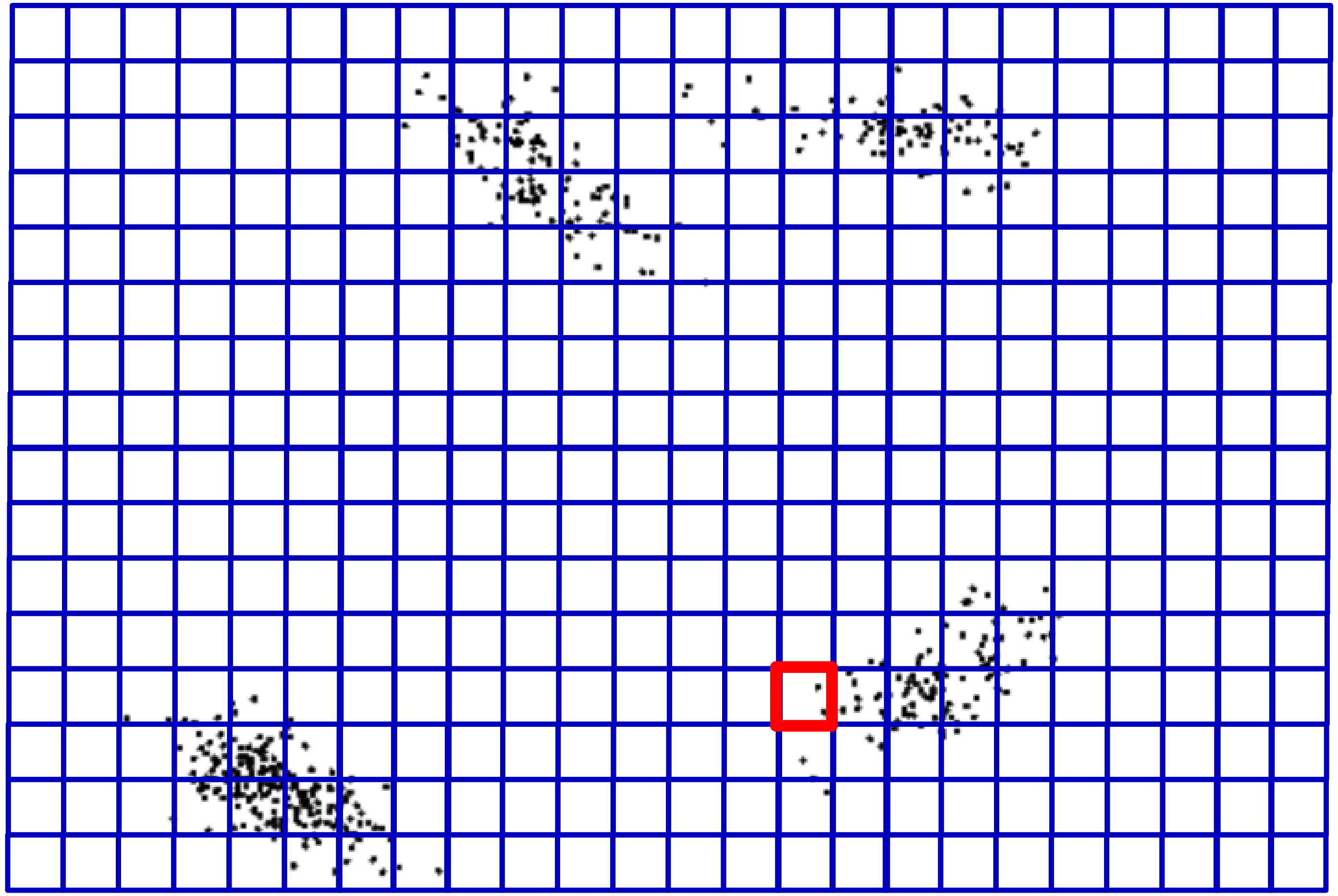
- *Approximate* nearest neighbours:
  - We allow errors in the nearest neighbour calculation to gain speed.

- A simple and very-fast approximate nearest neighbour method:
  - Only check points within the same square.
  - Works if neighbours are in the same square.
  - But misses neighbours in adjacent squares.

- A simple trick to improve the approximation quality:
  - Use more than one grid.
  - So "close" points have more "chances" to be in the same square.

Approximate Nearest Neighbours

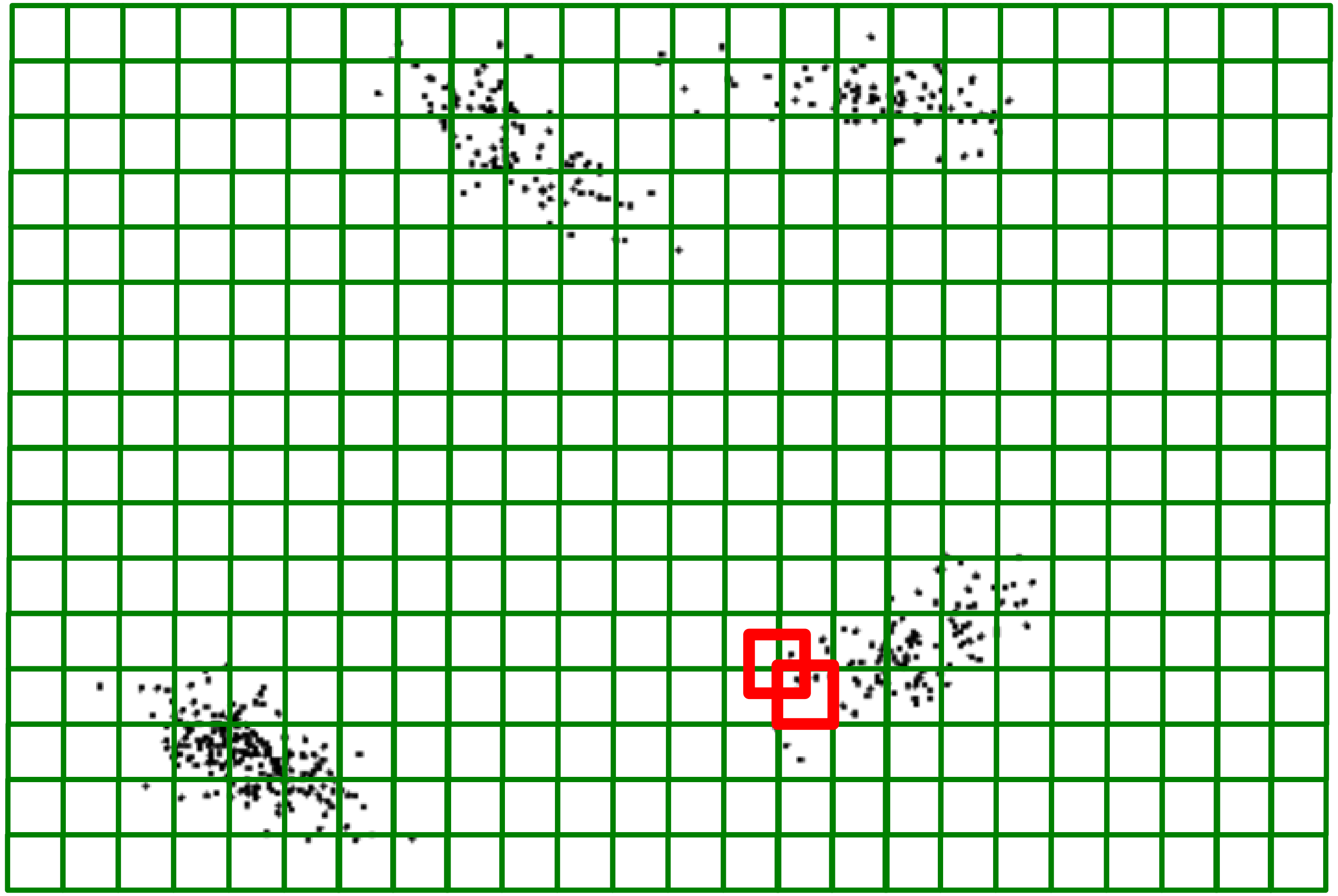
Grid 1:



Approximate Nearest Neighbours

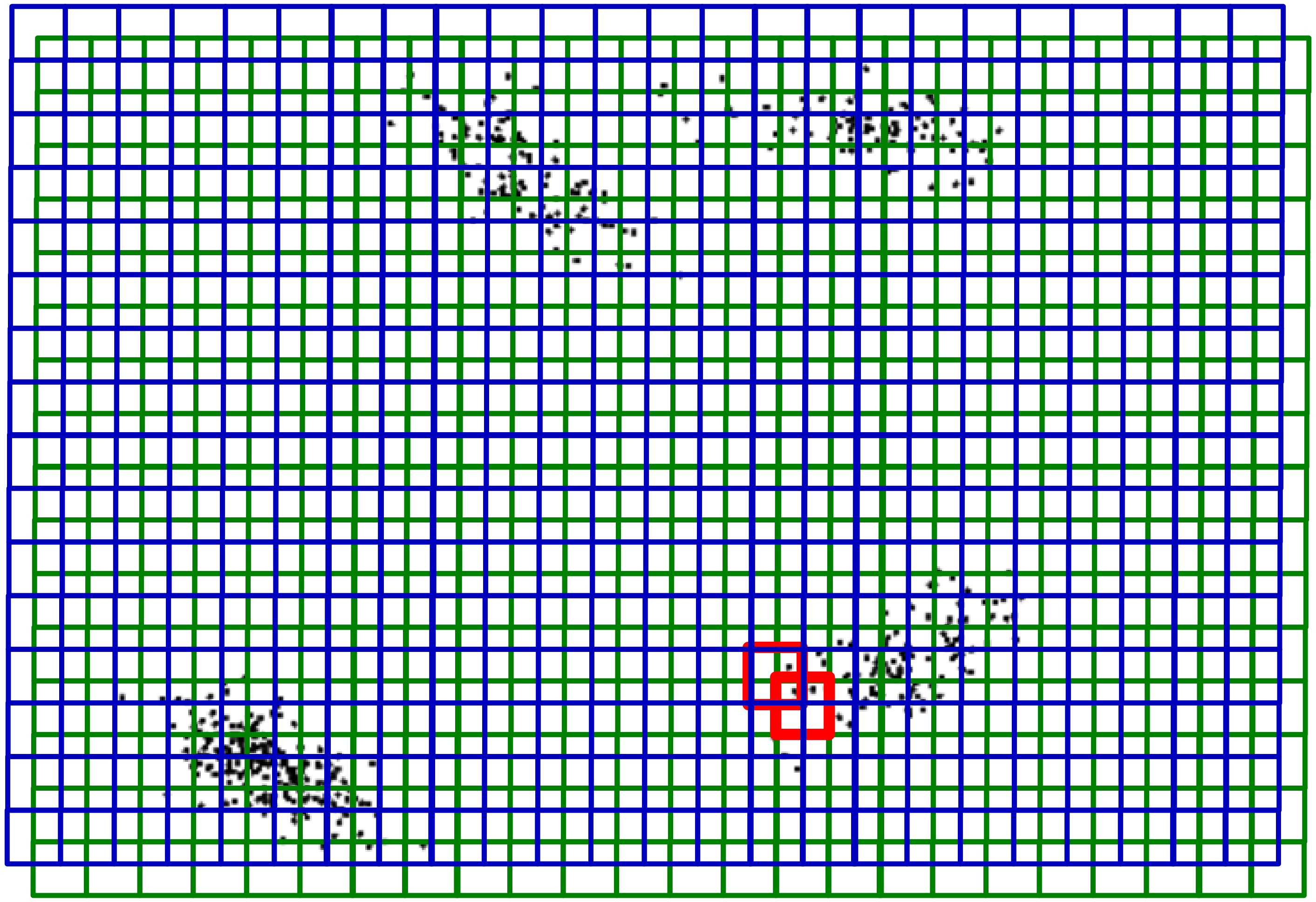
- Using multiple sets of regions improves accuracy.

Grid 2:



Approximate Nearest Neighbours

- Using multiple sets of regions improves accuracy.



Locality-Sensitive Hashing

- Even with multiple regions, approximation can be poor for large ‘d’.

- Common Solution (locality-sensitive hashing):
  - Replace features x_i with lower-dimensional features z_i.
    - E.g., turns each a 1000000-dimensional x_i into a 10-dimensional z_i.
  - Choose random z_i to preserve high-dimensional distances (bonus slides).

$$||z_i - z_j|| \approx ||x_i - x_j||$$

  - Find points hashed to the same square in lower-dimensional ‘z_i’ space.
  - Repeat with different random z_i values to increase chances of success.

End of Part 2: Key Concepts

- We focused on 3 unsupervised learning tasks:
  - Clustering.
    - Partitioning (k-means) vs. density-based.
    - “Flat” vs. hierarachial (agglomerative).
    - Vector quantization.
    - Label switching.
  - Outlier Detection.
    - Ambiguous objective.
    - Common approaches (model-based, graphical, clustering, distance-based, supervised).
  - Finding similar items.
    - Amazon product recommendation.
    - Region-based pruning for fast “closest point” calculations.

- If previous years we also covered “association rules”:
  - http://www.cs.ubc.ca/~schmidtm/Courses/340-F16/L12.pdf

Summary

- Distance-based outlier detection:
  - Based on measuring (relative) distance to neighbours.
- Supervised-learning for outlier detection:
  - Can detect complex outliers given a training set.
- Amazon product recommendation:
  - Find similar items using nearest neighbour search.
- Fast nearest neighbour methods drastically reduce search time.
  - Inverted indices, distance-based pruning.

- Next week: how do we do supervised learning with a *continuous* y_i?

Locality-Sensitive Hashing

- How do we make distance-preserving low-dimensional features?

- Johnson-Lindenstrauss lemma (paraphrased):
  - Define element 'j' of 'z_i' by:

$$z_{ij} = w_{j1} x_{i1} + w_{j2} x_{i2} + \cdots + w_{jd} x_{id}$$

  - Where the scalars 'w_{jc}' are samples from a standard normal distribution.
    - We can collect them into a matrix 'W', which is the same for all 'i'.

  - If the dimension 'k' of the 'z_i' is large enough, then: $\|z_i - z_j\| \approx \|x_i - x_j\|$
    - Specifically, we'll require k = Ω(log(d)).

Locality-Sensitive Hashing

- Locality-sensitive hashing:
  1. Multiply X by a random Gaussian matrix ‘W’ to reduce dimensionality.
  2. Hash dimension-reduced points into regions.
  3. Test points in the same region as potential nearest neighbours.

- Now repeat with a different random matrix.
  – To increase the chances that the closest points are hashed together.

- An accessible overview is here:
  – http://www.slaney.org/malcolm/yahoo/Slaney2008-LSHTutorial.pdf

Cosine Similarity vs. Normalized Nearest Neighbours

- The Amazon paper says they "maximize cosine similarity".
- But this is equivalent to normalized nearest neighbours.
- Proof for k=1:

$$\underbrace{\underset{j}{\operatorname{argmin}} \left\| \frac{x_i}{\|x_i\|} - \frac{x_j}{\|x_j\|} \right\|}_{\text{normalized nearest neighbour}} \equiv \underset{j}{\operatorname{argmin}} \frac{1}{2} \left\| \frac{x_i}{\|x_i\|} - \frac{x_j}{\|x_j\|} \right\|^2$$

$$\equiv \underset{j}{\operatorname{argmin}} \frac{1}{2} \frac{x_i^T x_i}{\|x_i\|^2} - \frac{2 x_i^T x_j}{\|x_i\| \cdot \|x_j\|} + \frac{1}{2} \frac{x_j^T x_j}{\|x_j\|^2}$$

$$\equiv \underset{j}{\operatorname{argmin}} - \frac{x_i^T x_j}{\|x_i\| \cdot \|x_j\|}$$

$$\equiv \operatorname{argmax} \frac{x_i^T x_j}{\|x_i\| \cdot \|x_j\|} \quad \Big\} \rightarrow \text{maximum cosine similarity}$$

Outlierness (Symbol Definition)

- Let $N_k(x_i)$ be the k-nearest neighbours of x_i.
- Let $D_k(x_i)$ be the average distance to k-nearest neighbours:

$$D_k(x_i) = \frac{1}{k} \sum_{j \in N_k(x_i)} \|x_i - x_j\|$$

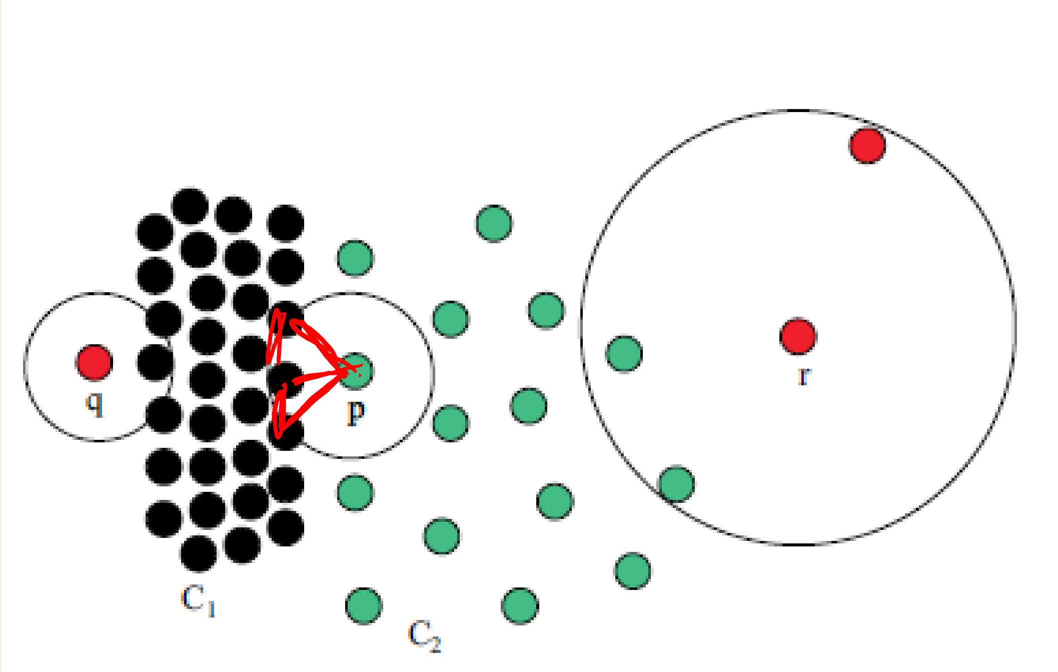
- Outlierness is ratio of $D_k(x_i)$ to average $D_k(x_j)$ for its neighbours 'j':

$$O_k(x_i) = \frac{D_k(x_i)}{\frac{1}{k} \sum_{j \in N_k(x_i)} D_k(x_j)}$$

- If outlierness > 1, x_i is further away from neighbours than expected.

Outlierness with Close Clusters

- If clusters are close, outlierness gives unintuitive results:

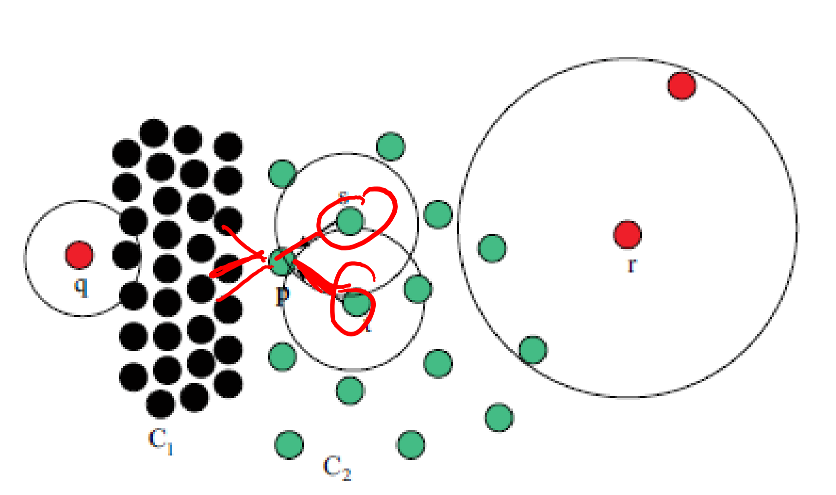


- In this example, ‘p’ has higher outlierness than ‘q’ and ‘r’:
  - The green points are not part of the KNN list of ‘p’ for small ‘k’.

http://www.comp.nus.edu.sg/~atung/publication/pakdd06_outlier.pdf

Outlierness with Close Clusters

- ‘Influenced outlierness’ (INFLO) ratio:
  - Include in denominator the ‘reverse’ k-nearest neighbours:
    - Points that have ‘p’ in KNN list.
  - Adds ‘s’ and ‘t’ from bigger cluster that includes ‘p’:



- But still has problems:
  - Dealing with hierarchical clusters.
  - Yields many false positives if you have “global” outliers.
  - Goldstein and Uchida [2016] recommend just using KNN.

http://www.comp.nus.edu.sg/~atung/publication/pakdd06_outlier.pdf

Malware and Intrusion Detection Systems

- In antivirus software and software for network intrusion detection systems, another method of outlier detection is common:
  - “Signature-based” methods: keep a list of byte sequences that are known to be malicious. Raise an alarm if you detect one.
  - Typically looks for **exact** matches, so can be implemented very quickly.
    - E.g., using data structures like “suffix trees”.
  - Can’t detect new types of outliers, but if you are good at keeping your list of possible malicious sequences up to date then this is very effective.
  - Here is an article discussing why ML is *not* common in these settings:
    - http://www.icir.org/robin/papers/oakland10-ml.pdf

Shingling: Decomposing Objects into Pars

- We say that a program is a virus if it has a malicious byte sequence.
  - We don't try to compute similarity of the whole program.
- This idea of finding similar "parts" is used in various places.
- A key tool to be help us do this is "shingling":
  - Dividing an object into consecutive "parts".
  - For example, we previously saw "bag of words".
- Given the shingles, we can search for similar parts rather than whole objects.

Shingling Applications

- For example, n-grams are one way to shingle text data.
  - If we use tri-grams, the sentence “there are lots of applications of nearest neighbours” would have these shingles:
    - {“there are lots”, “are lots of”, “lots of applications”, “of applications of”, “applications of nearest”, “of nearest neighbours”}.
  - We can find similar items using similarity/distance between sets.
    - For example, using the Jaccard similarity.

- Applications where finding similar shingles is useful:
  - Detecting plagiarism (shared n-grams indicates copying).
  - BLAST gene search tool (shingle parts of a biological sequence).
  - Entity resolution (finding whether two citations refer to the same thing).
  - Fingerprint recognition (shingles are “minutiae” in different image grid cells).

Shingling Practical Issues

- In practice, you can save memory by not storing the full shingles.
- Instead, define a hash function mapping from shingles to bit-vectors, and just store the bit-vectors.

- However, for some applications even storing the bit-vectors is too costly:
  - This leads to randomized algorithms for computing Jaccard score between huge sets even if you don't store all the shingles.

- Conceptually, it's still useful to think of the "bag of shingles" matrix:
  - X_{ij} is '1' if object 'i' has shingle 'j'.

Minhash and Jaccard Similarity

- Let $h(x_i)$ be the smallest index 'j' where x_{ij} is non-zero ("minhash").

- Consider a random permutation of the possible shingles 'j':
  - In Julia: randperm(d).
  - The value $h(x_i)$ will be different based on the permutation.

- Neat fact:
  - Probability that $h(x_i) = h(x_j)$ is the Jaccard similarity between x_i and x_j.
- Proof idea:
  - Probability that you stop with $h(x_i) = h(x_j)$ is given by probability that $x_{ik}=x_{jk}=1$ for a random 'k', divided by probability that at least one of $x_{ik}=1$ or $x_{jk}=1$ is true for a random 'k'.

Low-Memory Randomized Jaccard Approximation

- The "neat fact" lets us approximate Jaccard similarity without storing the shingles.

- First we generate a bunch of random permutations.
  - In practice, use a random hash function to randomly map 1:d to 1:d.
- For each example, go through its shingles to compute $h(x_i)$ for each permutation.
  - No need to store the shingles.
- Approximate Jaccard(x_i, x_j) as the fraction of permutations where $h(x_i)=h(x_j)$.