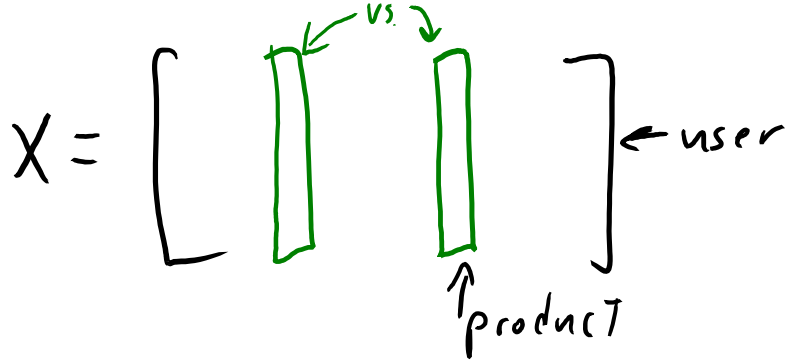
CPSC 340:
Machine Learning and Data Mining

Finding Similar Items

Fall 2019

Last Time: Amazon Product Recommendation

- Amazon product recommendation method:



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

- Method divide each column by its norm, $x^i/||x^i||$.
  - This is called normalization.
  - Normalized KNN is equivalent to maximizing "cosine similarity" (bonus).

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):

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

- 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 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.

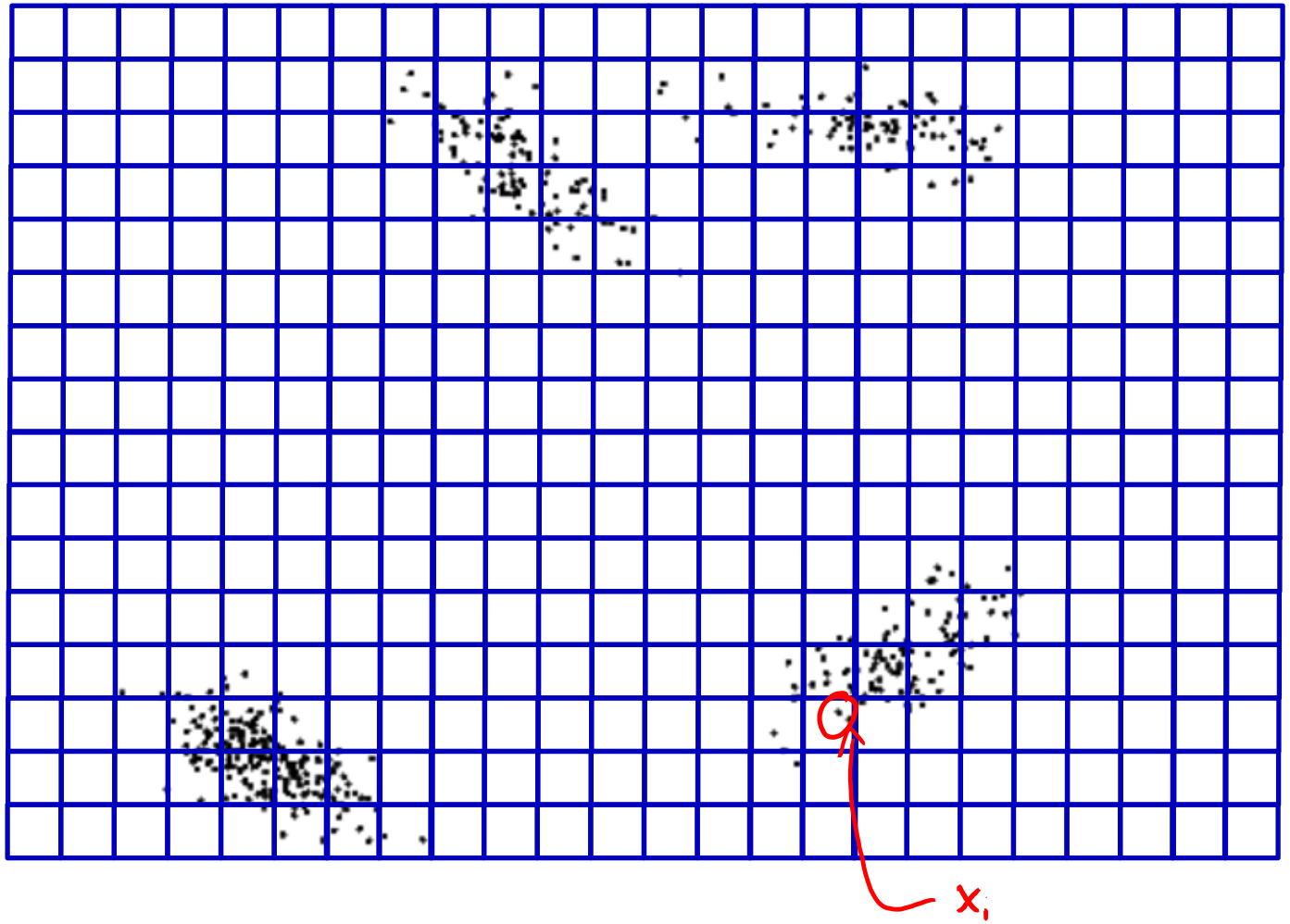
- The answers are called an inverted index, queries now cost O(k).
  - Needs an extra O(nk) storage, which is fine for small ‘k’.

Grid-Based Pruning

- Assume we want to find examples within distance of 'ε' of point x_i.

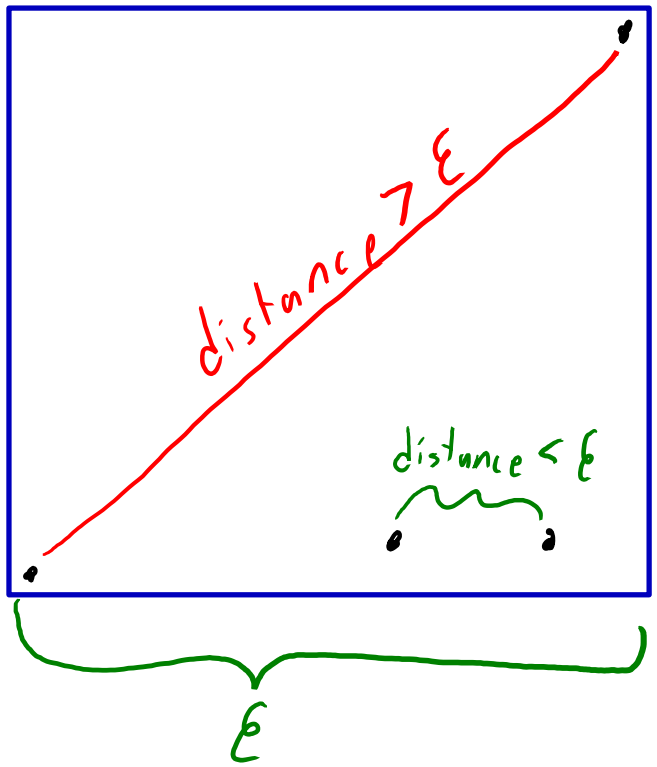
Divide space into squares of length ε.

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?

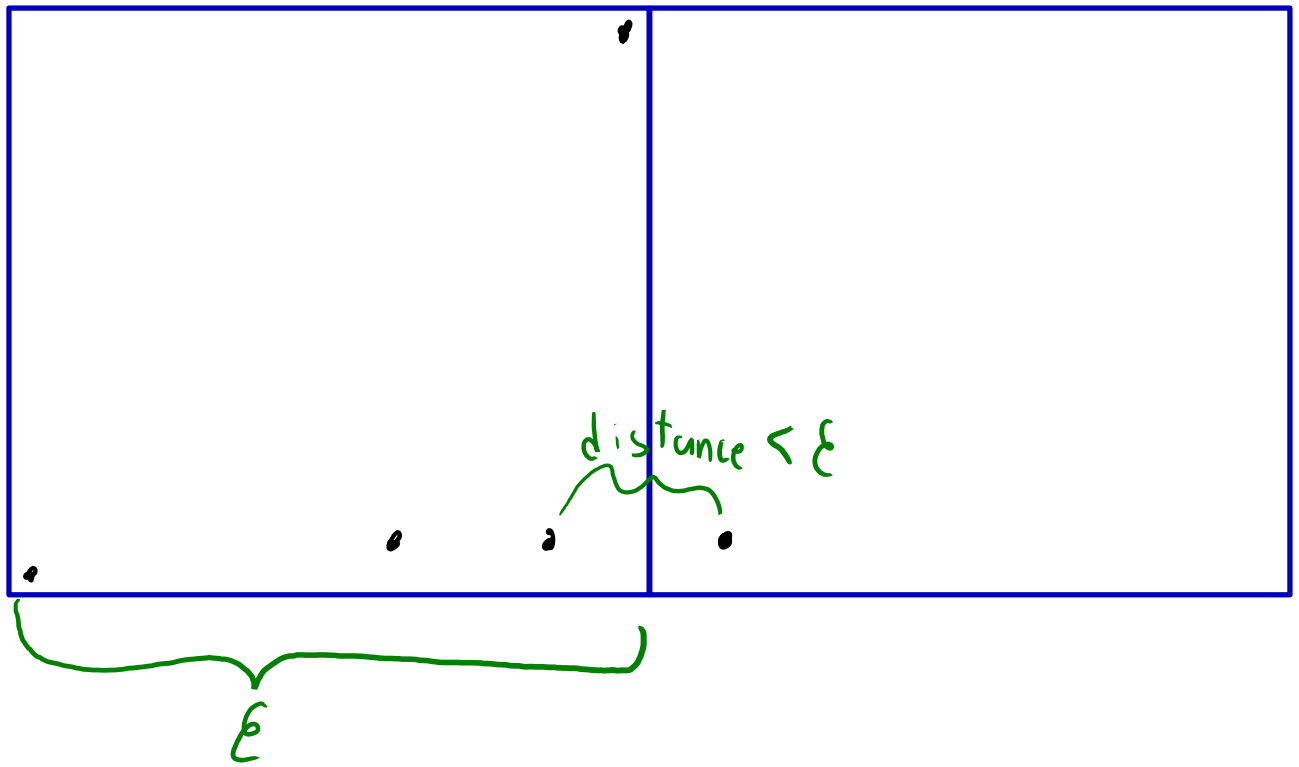


Points in same square can have distance less than ‘ε’.

Grid-Based Pruning

- Which squares do we need to check?

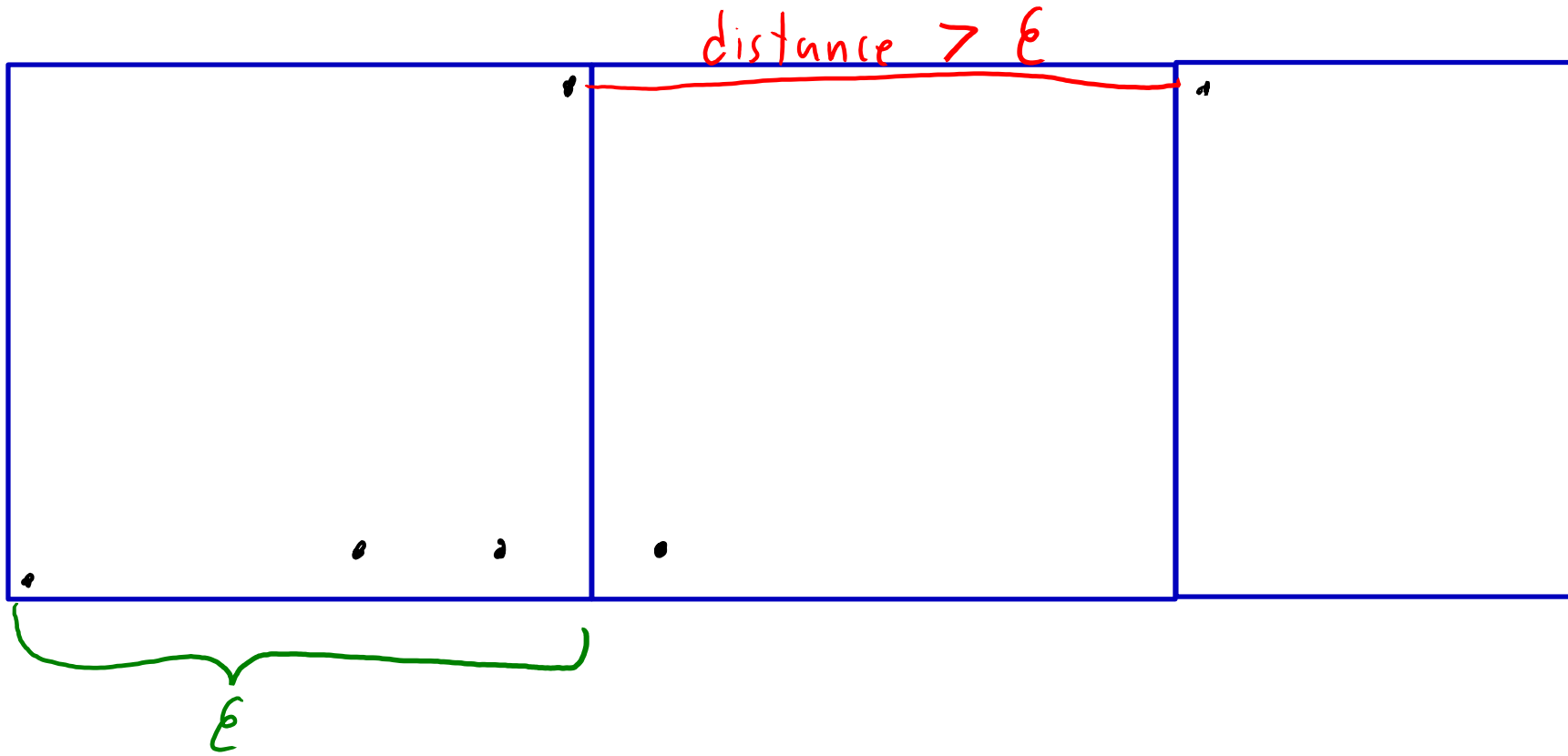
Points in adjacent squares can have distance less than distance ‘ε’.



Grid-Based Pruning

- Which squares do we need to check?

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



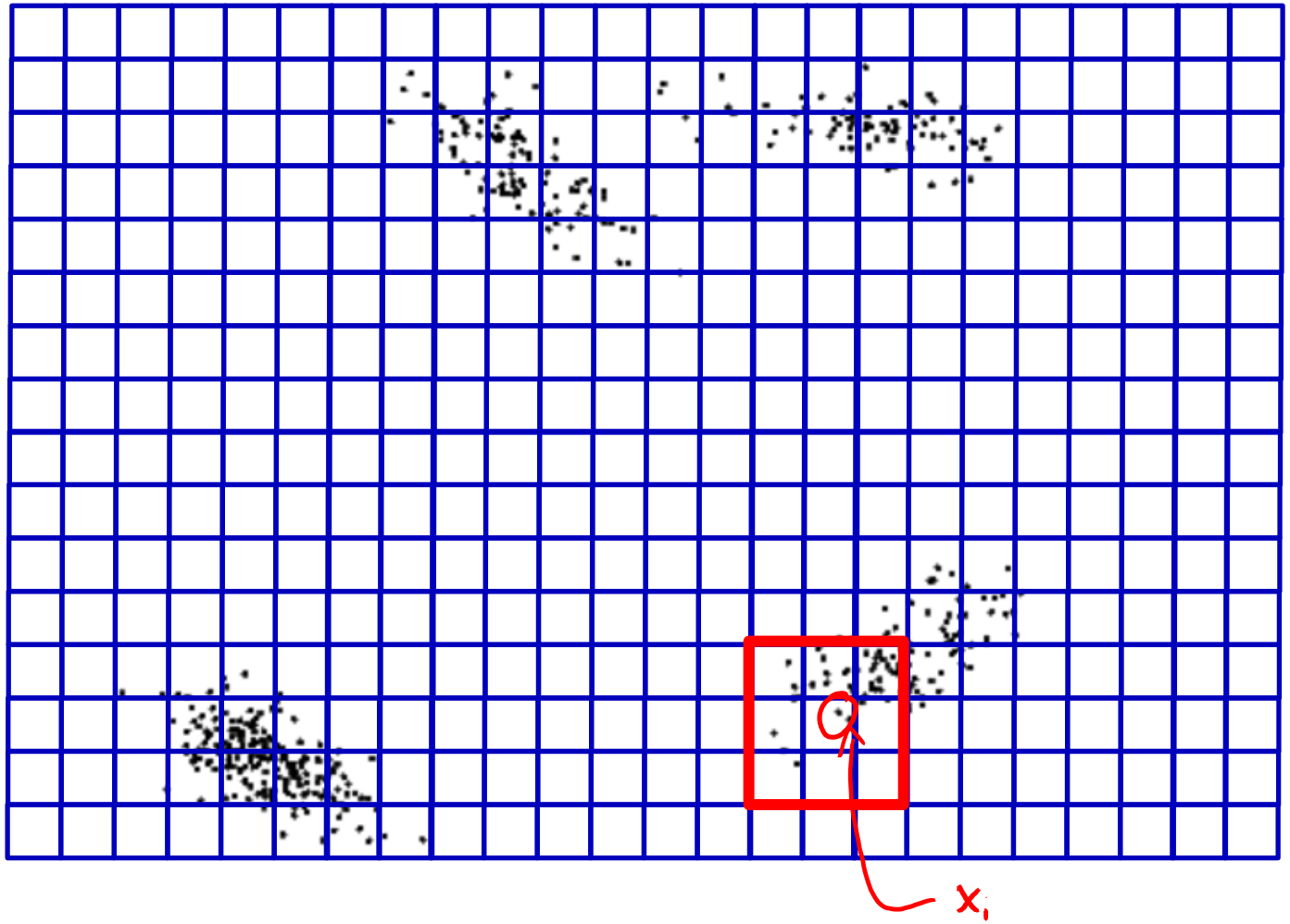
Grid-Based Pruning

- Assume we want to find examples within distance of 'ε' of point x_i.

Divide space into squares of length ε.

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

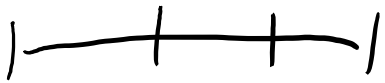
Only need to check points in same and adjacent squares.

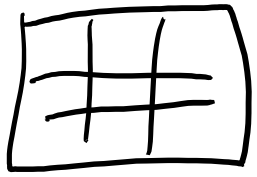


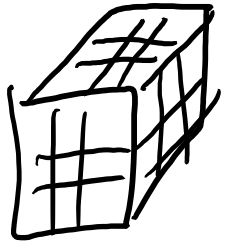
Grid-Based Pruning Discussion

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

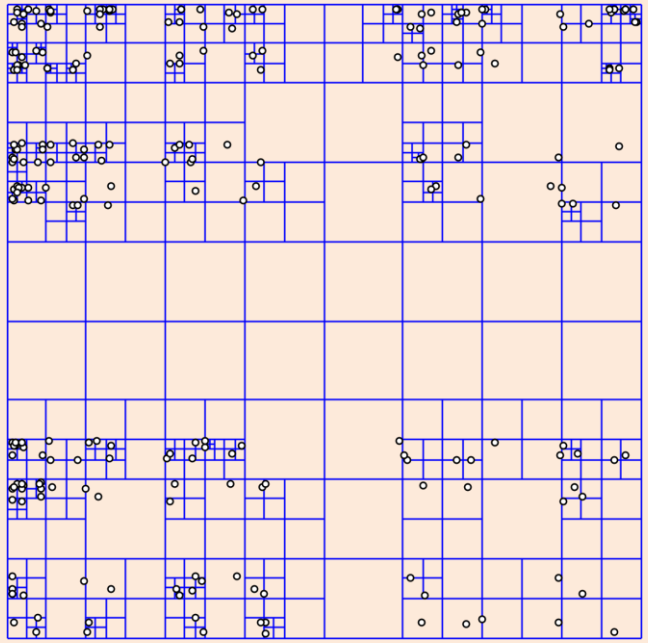


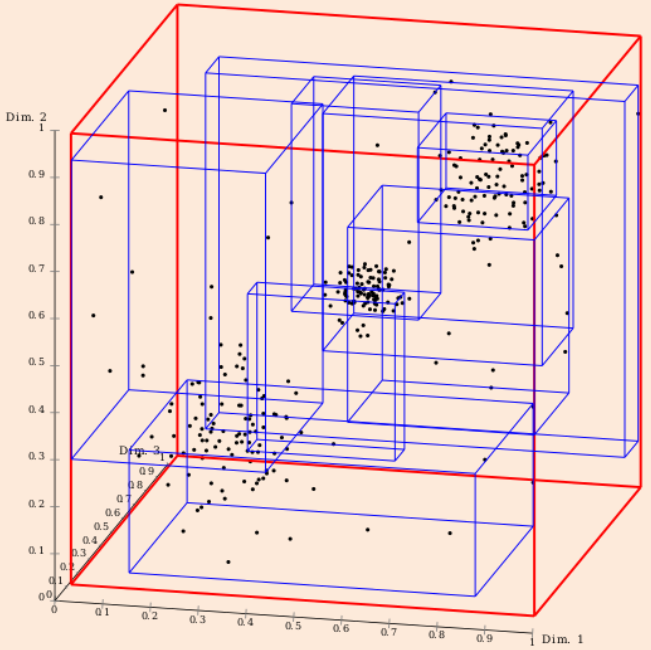


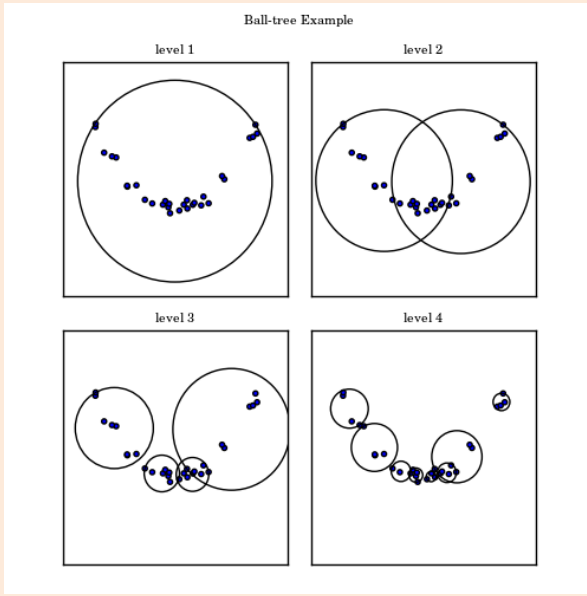


Grid-Based Pruning Discussion

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







- Work 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

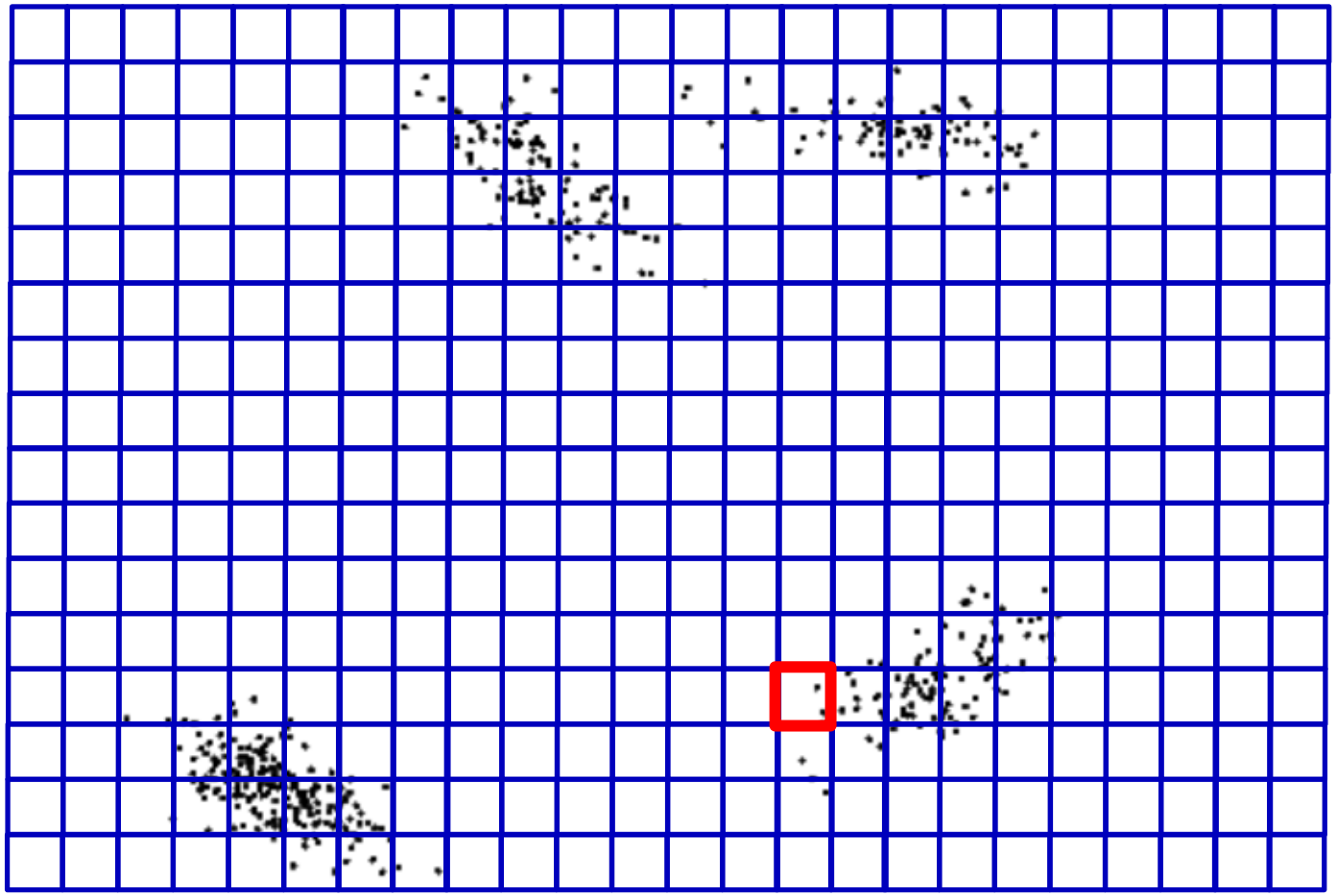
- *Approximate* nearest neighbours:
  - We can allow errors in the nearest neighbour calculation to gain speed.

- 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.

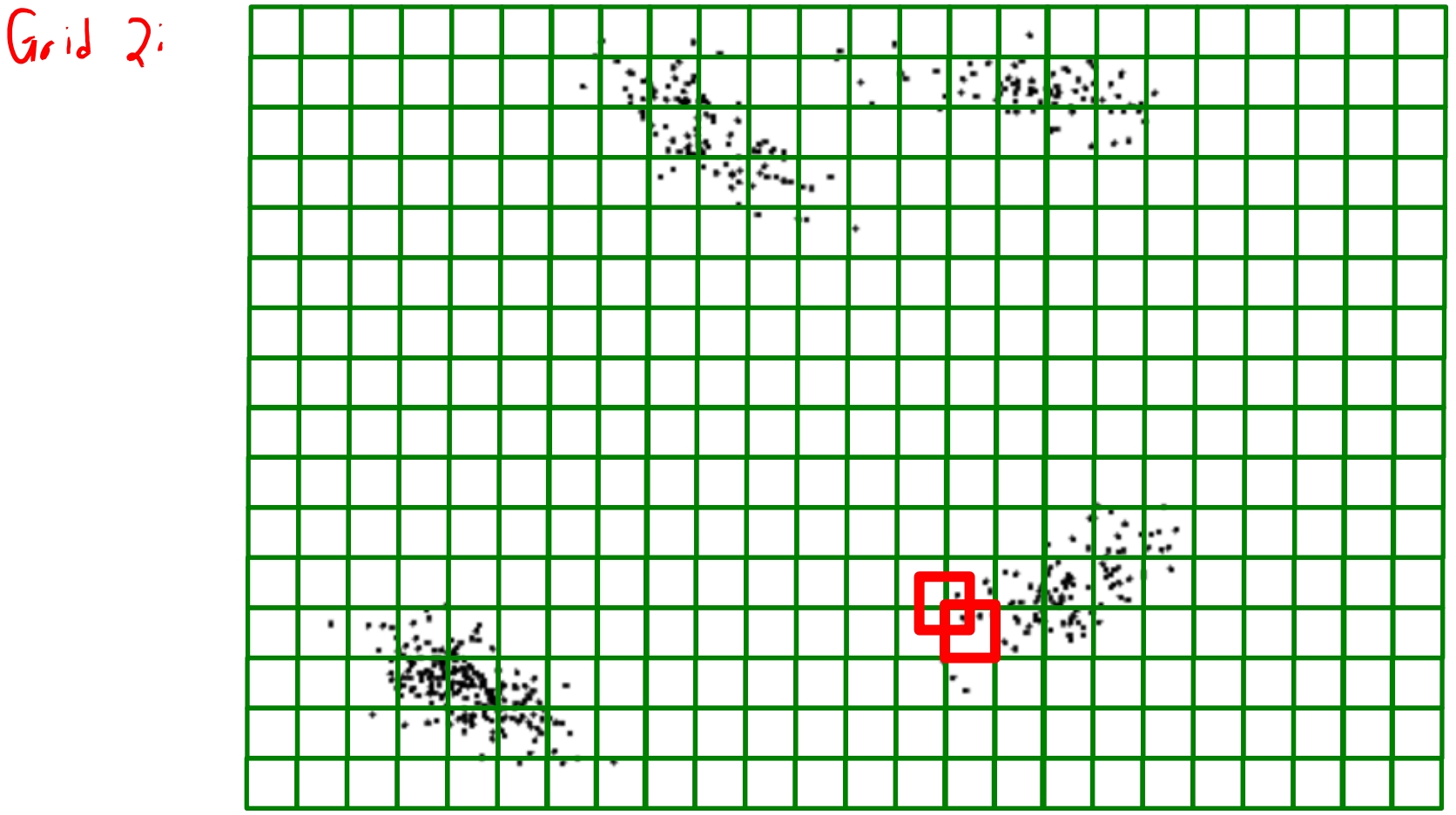
Approximate Nearest Neighbours

Grid 1:



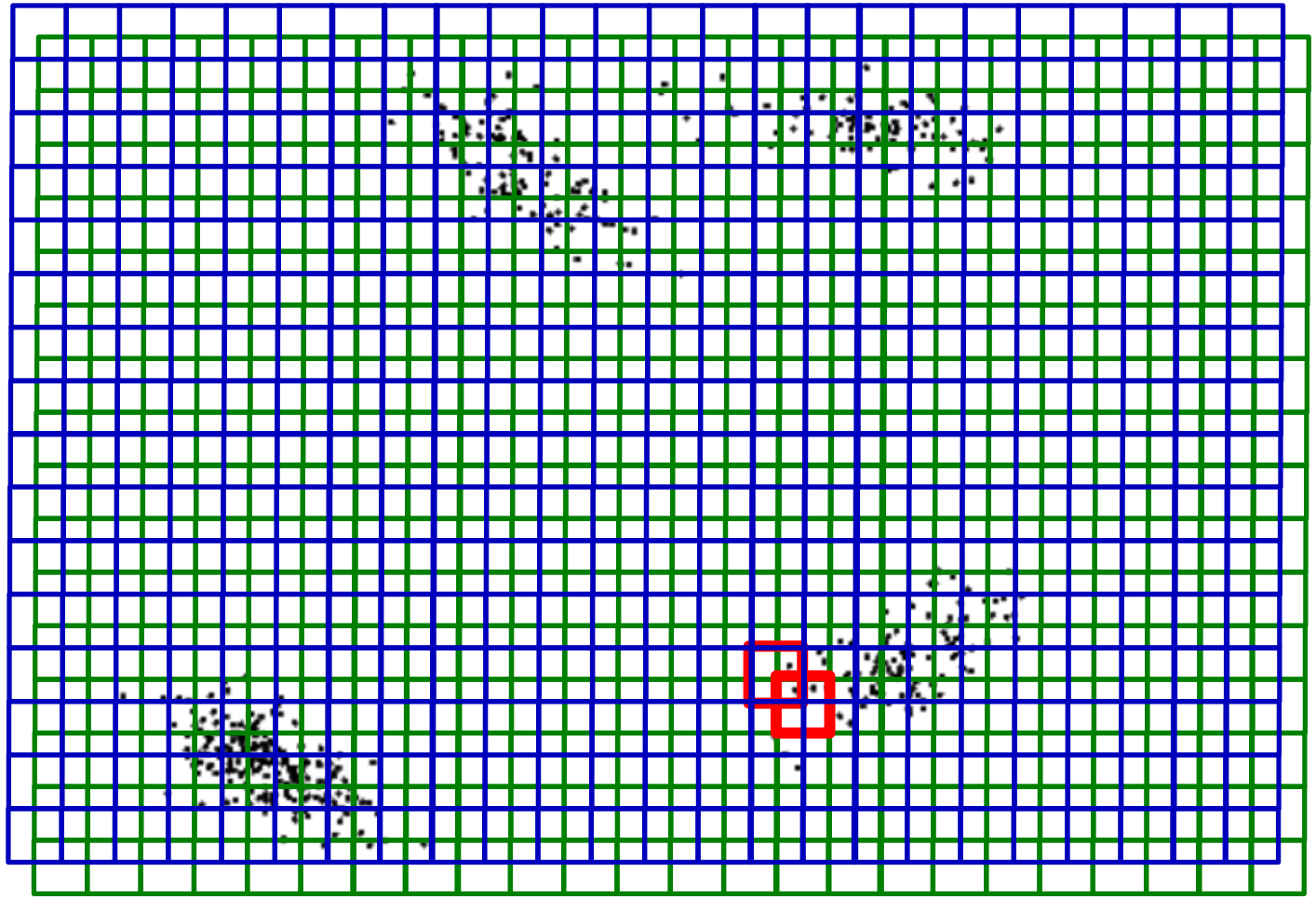
Approximate Nearest Neighbours

- Using multiple sets of regions improves accuracy.



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 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.

(pause)

Shingling: Decomposing Examples into Parts

- Detecting plagiarism (copying) is another "finding similar items" task.
  - However, it's unlikely that an entire document is plagiarized.
    - So something like "Euclidean distance between documents" doesn't seem right.
  - Instead, you want to find if two documents have similar "parts".
    - Sequences of words that are copied.

- This idea of finding similar "parts" is used in various places.
  - We previously saw "bag of words" to divide text into parts/words.

- Common: divide examples into "parts", measure similarity of "parts".
  - "Shingling" is a word meaning "divide objects into parts".

- Given "shingles", can search for similar parts instead of whole examples.

Shingling and Hashing

- As an example, n-grams are one way to shingle text data.
  - Example input: “there are lots of applications of nearest neighbours”.
  - Example trigram output (set of each three consecutive words):
    - {“there are lots”, “are lots of”, “lots of applications”, “of applications of”, “applications of nearest”, “of nearest neighbours”}.

- **Exact** matching of individual shingles is fast using hashing:
  - Hash key would be the shingle.
  - Hash value would be the training examples that include the shingle.

hash{"applications of nearest"} = [36, 801, 968, 2046, ...]

Index of an example containing the trigram.

Shingling Example

- Train example: “there are lots of applications of nearest neighbours methods”.
  - Trigram shingles: {“there are lots”, “are lots of”, “lots of applications”, “of applications of”, “applications of nearest”, “of nearest neighbours”}.

- Test example: “nearest neighbours methods are found in lots of applications.
  - Trigram shingles: {“nearest neighbours methods”, “neighbours methods are”, “methods are found”, “are found in”, “in lots of”, “lots of applications”.}
  - These two trigrams would refer to the training example above in the hash table.

- To detect plagiarism, you would shingle an entire document.
  - And probably use longer n-grams.

Shingling Applications

- You could alternately measure similarity between sets of shingles.
  - Say that objects are “similar” if they share a lot of shingles.
  - Bonus: “minhash” randomized method for approximate Jaccard similarity.
    - Without storing all the shingles (because there may too many to store).

- Example applications where finding similar shingles is useful:
  - Detecting plagiarism (shared n-grams indicates copying).
  - Entity resolution (finding whether two citations refer to the same document).
  - BLAST gene search tool (shingle parts of a biological sequence for fast retrieval).
  - Anti-virus software (virus “signature” is a byte sequence known to be malicious).
  - Intrusion detection systems (often also based on “signatures”).
  - Fingerprint recognition (shingles are “minutiae” in different regions of image).

(pause)

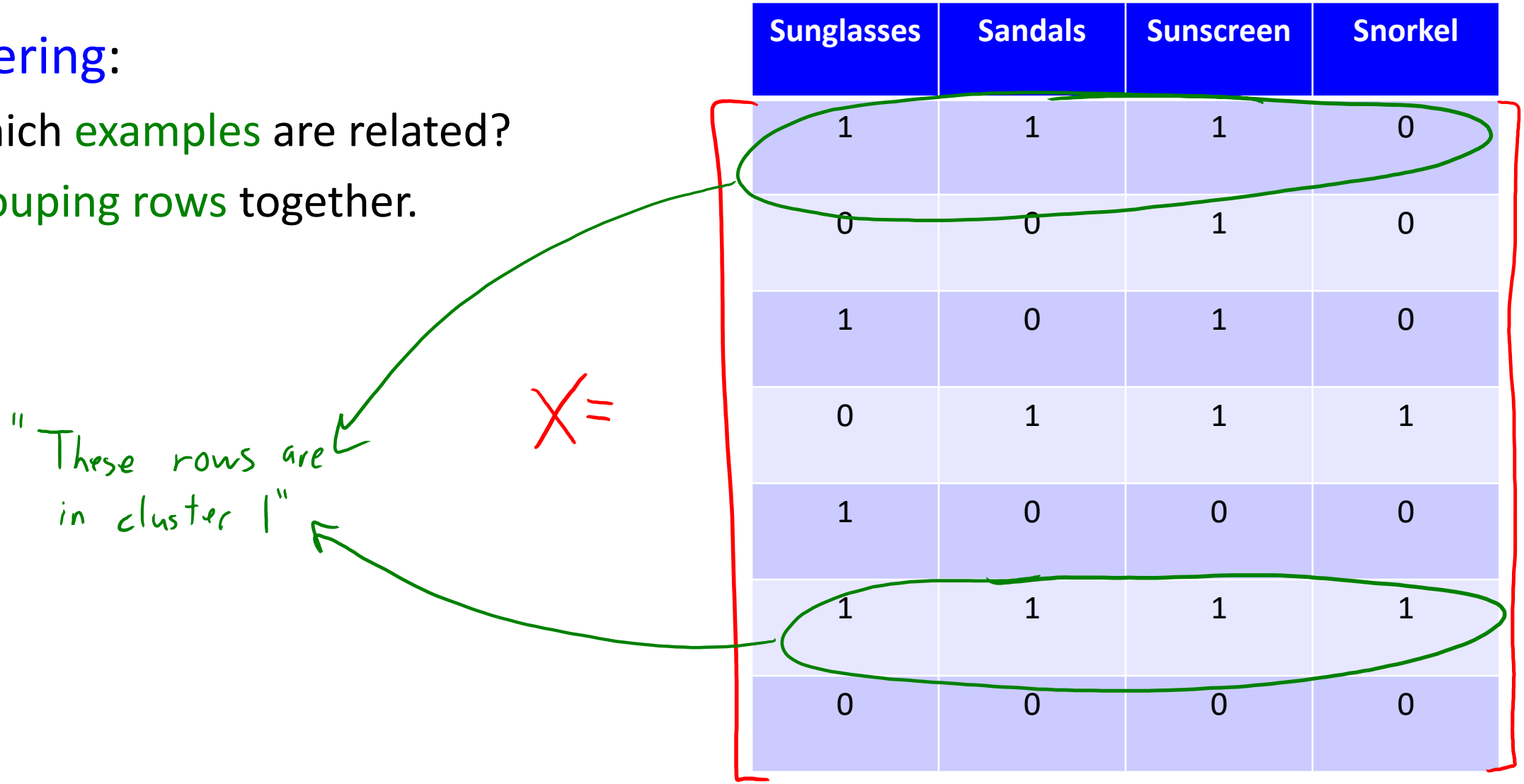
Motivation: Product Recommendation

- "Frequent itemsets": sets of items frequently 'bought' together.

- With this information, you could:
  - Put them close to each other in the store.
  - Make suggestions/bundles on a website.

Clustering vs. Frequent Itemsets

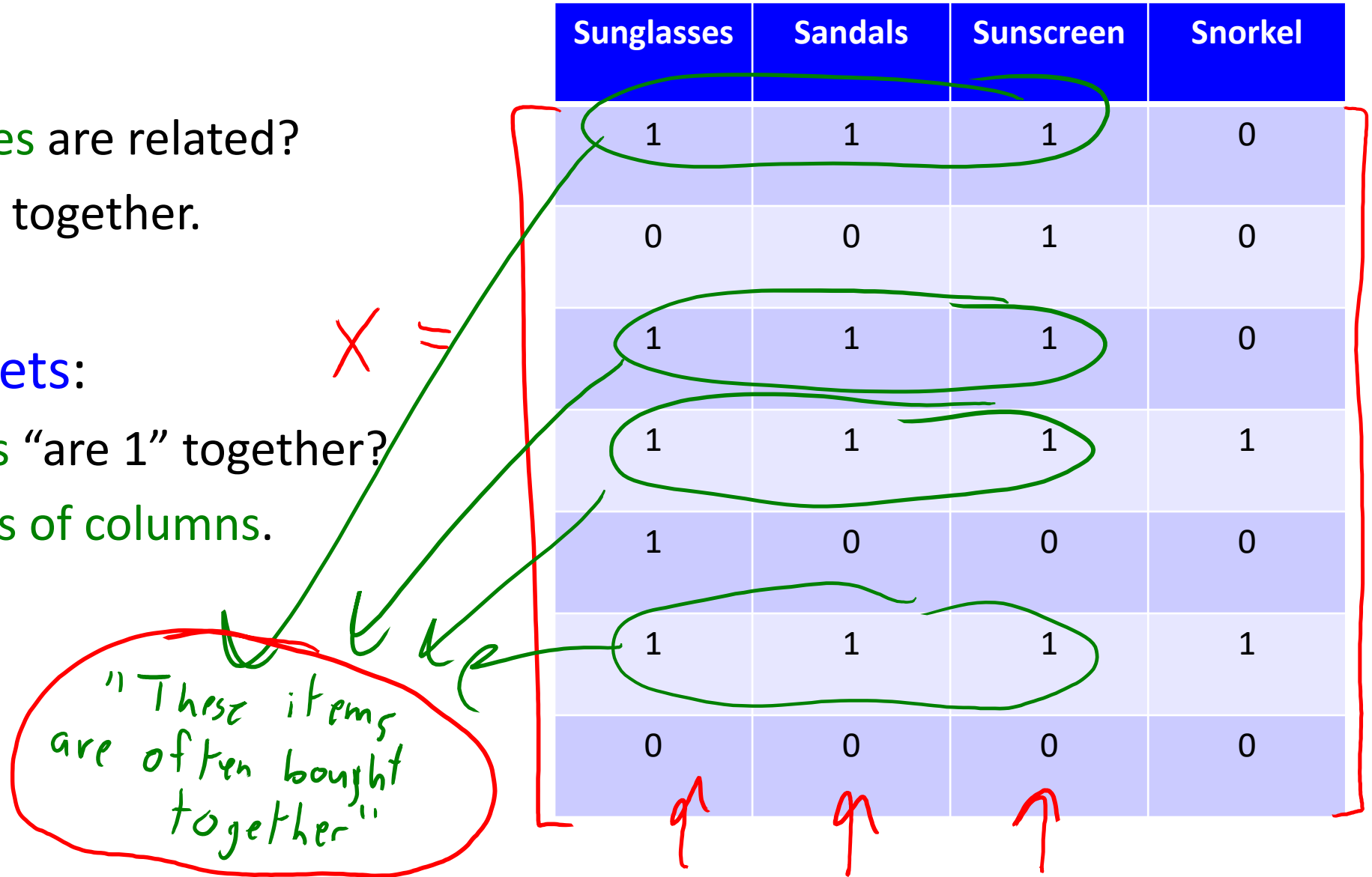
- Clustering:
  - Which examples are related?
  - Grouping rows together.



Sunglasses	Sandals	Sunscreen	Snorkel
1	1	1	0
0	0	1	0
1	0	1	0
0	1	1	1
1	0	0	0
1	1	1	1
0	0	0	0

Clustering vs. Frequent Itemsets

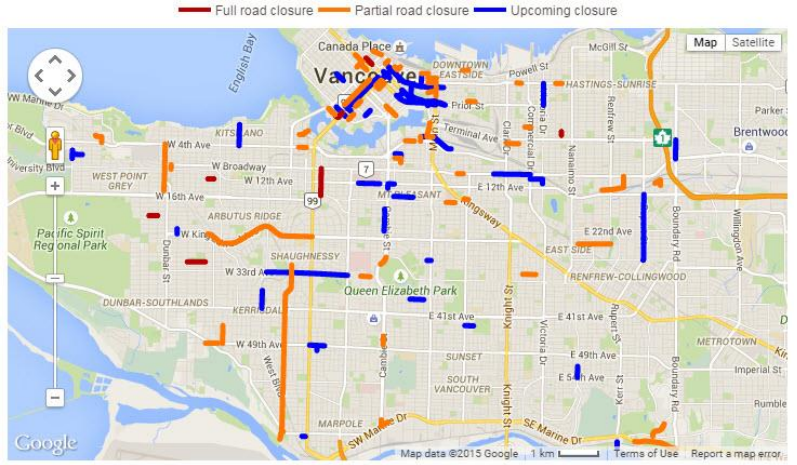
- Clustering:
  - Which examples are related?
  - Grouping rows together.
- Frequent Itemsets:
  - Which features "are 1" together?
  - Relating groups of columns.

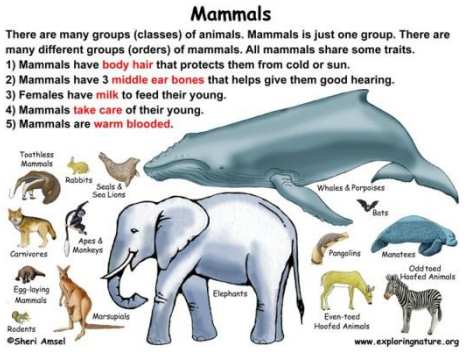


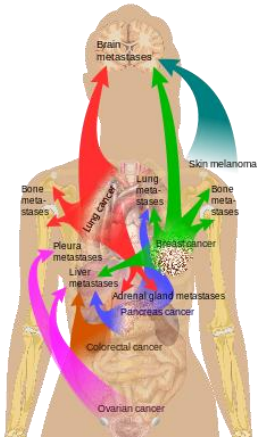
Sunglasses	Sandals	Sunscreen	Snorkel
1	1	1	0
0	0	1	0
1	1	1	0
1	1	1	1
1	0	0	0
1	1	1	1
0	0	0	0

Applications of Association Rules

- Which foods are frequently eaten together?
- Which genes are turned on at the same time?
- Which traits occur together in animals?
- Where do secondary cancers develop?
- Which traffic intersections are busy/closed at the same time?
- Which players outscore opponents together?





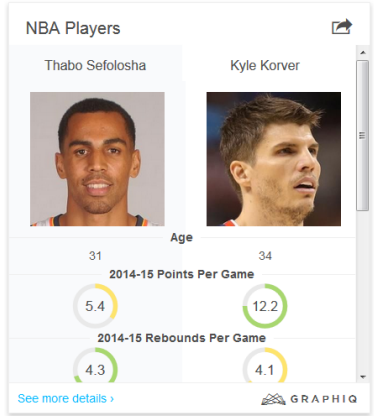


Atlanta Hawks #1

Minutes played together: 398
Combined net rating (per 48 minutes): 23.8
Overall rank among two-man lineups: 1st

Reaction: Against all odds, the most efficient tandem in the NBA is a pair of thirty-something wings. Kyle Korver and Thabo Sefolosha complement each other perfectly, with Korver providing the scoring punch and Sefolosha taking on the toughest defensive assignment for the Hawks.

With Sefolosha still getting back up to speed after a calf injury sidelined him for two months, the Hawks should probably just attach him to Korver until the two can get their chemistry back to how it was. Because any combination of players that can help a team outscore its opponents by 23.8 points per game is probably one worth exploring further.



http://www.exploringnature.org/db/view/624
https://en.wikipedia.org/wiki/Metastasis
http://basketball-players.pointafter.com/stories/3791/most-valuable-nba-duos#30-atlanta-hawks
http://modo.coop/blog/tips-from-our-pros-avoiding-late-charges-during-summer

"Support" of an Itemset

- "Support" p(S = 1) is the proportion of examples with all 'S' items.
- How do we compute p(S = 1)?
  - If S = {bread, milk}, we count proportion of times they are both "1".

Bread	Eggs	Milk	Oranges
1	1	1	0
0	0	1	0
1	0	1	0
0	1	0	1
…	…	…	…

→ yes
→ no
→ yes
→ no
⋮

$p(S=1) =$ # times all elements of 'S' are '1' / n

Challenge in Learning Association Rule

- Frequent itemset goal (given a threshold ‘s’):
  - Find all sets ‘S’ with $p(S = 1) \geq s$.

- Challenge: with ‘d’ features there are 2^d-1 possible sets.

For d=4 we have {1}, {2}, {3}, {4}, {1,2}, {1,3}, {1,4}, {2,3}, {2,4}, {3,4},
{1,2,3}, {1,2,4}, {1,3,4}, {2,3,4}, {1,2,3,4}

- It takes too long to even write all sets unless ‘d’ is tiny.
- Can we avoid testing all sets?
  - Yes, using a basic property of probabilities…
    (“downward-closure/anti-monotonicity”)

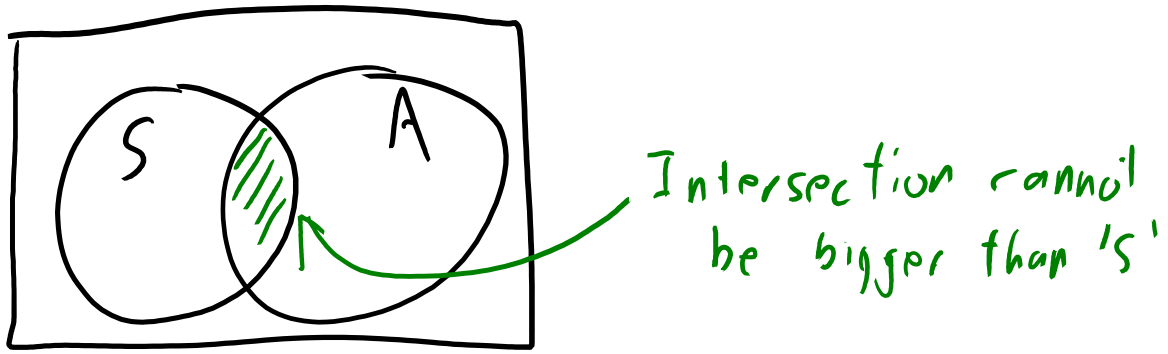
Upper Bound on Joint Probabilities

- Suppose we know that $p(S = 1) \geq s$.
- Can we say anything about $p(S = 1, A = 1)$?
  - Probability of buying all items in 'S', plus another item 'A'.
- Yes, $p(S = 1, A = 1)$ cannot be bigger than $p(S = 1)$.

By the product rule we have $p(S=1, A=1) = p(A=1 \mid S=1)\, p(S=1)$

$p(A=1 \mid S=1)$: a number ≤ 1

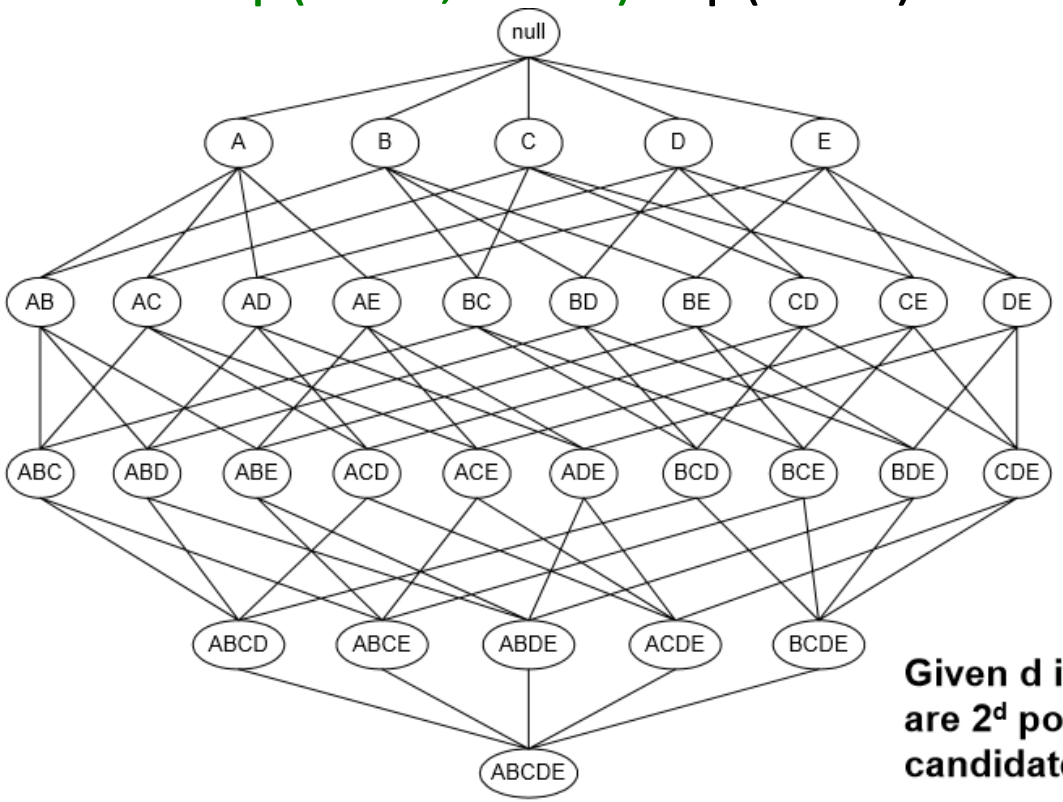
$\leq p(S=1)$



- E.g., probability of rolling 2 sixes on 2 dice (1/36) is less than 1 six on one di (1/6).

Support Set Pruning

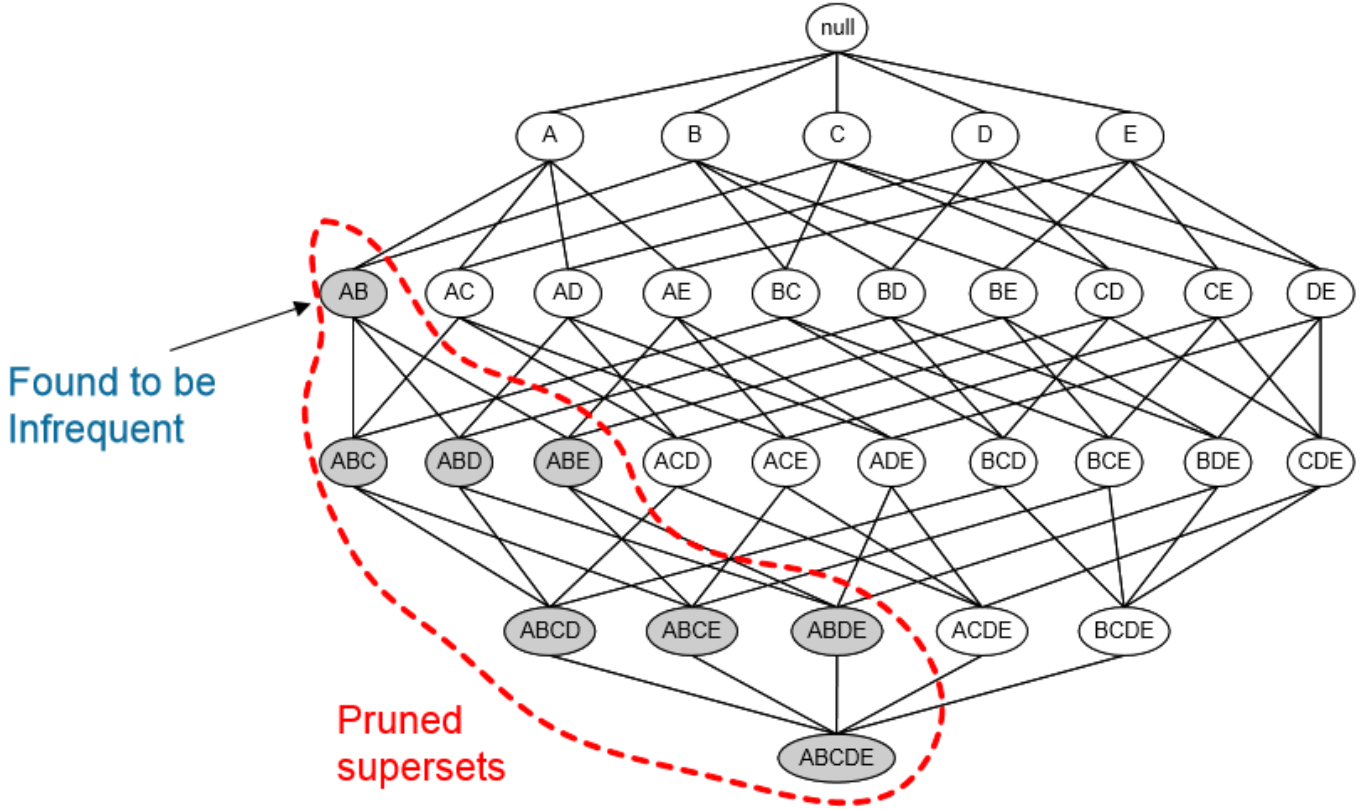
- This property means that p(S = 1) < s implies p(S = 1, A = 1) < s.
  - If p(sunglasses=1) < 0.1, then p(sunglasses=1, sandals=1) is less than 0.1.
  - We never consider p(S = 1, A = 1) if p(S = 1) has low support.



Given d items, there are 2^d possible candidate itemsets

Support Set Pruning

- This property means that p(S = 1) < s implies p(S = 1, A = 1) < s.
  - If p(sunglasses=1) < 0.1, then p(sunglasses=1, sandals=1) is less than 0.1.
  - We never consider p(S = 1, A = 1) if p(S = 1) has low support.



p(E) must be greater than

p(D, E) which must be greater than

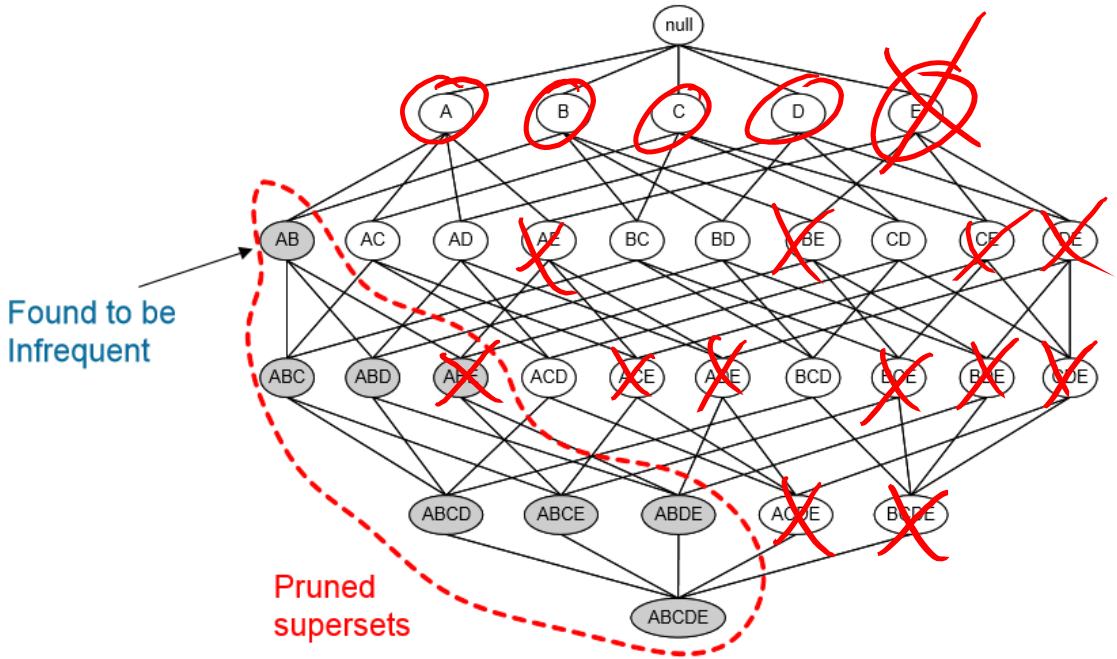
p(C, D, E) which must be greater than

p(B, C, D, E) which must be greater than

p(A, B, C, D, E)

A Priori Algorithm

- A priori algorithm for finding all subsets with p(S = 1) ≥ s.
  1. Generate list of all sets ‘S’ that have a size of 1.
  2. Set k = 1.
  3. Prune candidates ‘S’ of size ‘k’ where p(S = 1) < s.
  4. Add all sets of size (k+1) that have all subsets of size k in current list.
  5. Set k = k + 1 and go to 3.



A Priori Algorithm

Bread	Coke	Milk	Beer	Diaper	Eggs
1	0	1	0	1	0
0	1	0	1	1	1
1	0	1	0	1	1
⋮	⋮	⋮	⋮	⋮	⋮

Let's take minimum support as s = 0.30.

First compute probabilities for sets of size k = 1:

Item S	p(S = 1)
Bread	0.4
Coke	0.2
Milk	0.4
Beer	0.3
Diaper	0.4
Eggs	0.1

Bread, milk, diaper, beer have support at least 's'.

A Priori Algorithm

Bread	Coke	Milk	Beer	Diaper	Eggs
1	0	1	0	1	0
0	1	0	1	1	1
1	0	1	0	1	1
⋮	⋮	⋮	⋮	⋮	⋮

Let's take minimum support as s = 0.30.

First compute probabilities for sets of size k = 1:

Item S	p(S=1)
Bread	0.4
Coke	0.2
Milk	0.4
Beer	0.3
Diaper	0.4
Eggs	0.1

Bread, milk, diaper, beer have support at least 's'.

Combine sets of size k=1 with support 's' to make sets of size k = 2:

Itemset S	p(S=1)
{Bread, Milk}	0.3
{Bread, Beer}	0.2
{Bread, Diaper}	0.3
{Milk, Beer}	0.2
{Milk, Diaper}	0.3
{Beer, Diaper}	0.3

We don't check rules with coke or eggs.

{Bread, Milk}, {Bread, Diaper}, {Milk, Diaper}, {Beer, Diaper} have support at least 's'.

A Priori Algorithm

Bread	Coke	Milk	Beer	Diaper	Eggs
1	0	1	0	1	0
0	1	0	1	1	1
1	0	1	0	1	1
⋮	⋮	⋮	⋮	⋮	⋮

Let's take minimum support as s = 0.30.

First compute probabilities for sets of size k = 1:

Item S	p(S=1)
Bread	0.4
Coke	0.2
Milk	0.4
Beer	0.3
Diaper	0.4
Eggs	0.1

Bread, milk, diaper, beer have support at least 's'.

Combine sets of size k=1 with support 's' to make sets of size k = 2:

Itemset S	p(S=1)
{Bread, Milk}	0.3
{Bread, Beer}	0.2
{Bread, Diaper}	0.3
{Milk, Beer}	0.2
{Milk, Diaper}	0.3
{Beer, Diaper}	0.3

We don't check rules with coke or eggs.

{Bread, Milk}, {Bread, Diaper}, {Milk, Diaper}, {Beer, Diaper} have support at least 's'.

Check sets of size k = 3 where all subsets of size k = 2 have high support:

Itemset	p(S=1)
{Bread, Milk, Diaper}	0.3

(All other 3-item and higher-item counts are < 0.3)

(We only considered 13 out 63 possible rules.)

A Priori Algorithm Discussion

- Some implementations prune the output:
  - 'Maximal frequent subsets':
    - Only return sets S with $p(S = 1) \geq s$ where no superset S' has $p(S' = 1) \geq s$.
    - E.g., don't return {break,milk} if {bread, milk, diapers} also has high support.

- Number of rules we need to test is hard to quantify:
  - Need to test more rules for small 's'.
  - Need to test more rules as average #items per example increase.

- Computing $p(S = 1)$ if S has 'k' elements costs O(nk).
  - But there is some redundancy:
    - Computing p({1,2,3}) and p({1,2,4}) can re-use some computation.
  - Hashing can be used to speed up various computations.

Spurious Associations

- For large ‘d’, high probability of returning spurious associations:
  - With random data, one of the 2^d rules is likely to look strong.

- Classical story:
  - "In 1992, Thomas Blischok, manager of a retail consulting group at Teradata, and his staff prepared an analysis of 1.2 million market baskets from about 25 Osco Drug stores. Database queries were developed to identify affinities. The analysis "did discover that between 5:00 and 7:00 p.m. that consumers bought beer and diapers". Osco managers did NOT exploit the beer and diapers relationship by moving the products closer together on the shelves.“

- Fun with spurious with spurious correlations here.
  - Check whether rules make sense, and chance of finding spurious associations.

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.
    - Surveyed common approaches (and said that problem is ill-defined).
  - Finding similar items.
    - Amazon product recommendation.
    - Region-based pruning for fast “closest point” calculations.
    - Shingling divides objects into parts, matches individual parts of measures part set distance.
    - Frequent itemsets: finding items often bought together (a prior is an efficient method).

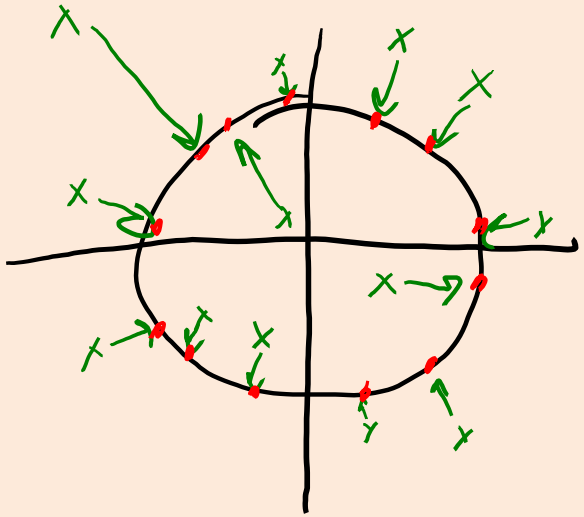
Summary

- Fast nearest neighbour methods drastically reduce search time.
  - Inverted indices, distance-based pruning.
- Shingling: dividing objects into parts.
  - Could just try to match individual parts.
  - Could measure Jaccard score between sets of parts.
- Support: measure of how often we see S.
- Frequent itemsets: sets of items with sufficient support.
- A priori algorithm: uses inequalities to prune search for sets.

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

Cosine Similarity vs. Normalized Nearest Neighbours

- The Amazon paper says they “maximize cosine similarity”.
- But this is equivalent to normalized nearest neighbours.
- Intuition for this equivalence:
  - When you normalize the features, they are all on the unit ball.
  - Nearest neighbours on the unit ball maximize inner product (cosine sim.):



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:

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

$$\equiv \operatorname*{argmin}_j \frac{1}{2} \frac{x_i^T x_i}{\|x_i\|^2} - \frac{2\, x_i^T x_j}{\|x_i\| \cdot \|x_j\|} + \frac{1}{2} \frac{x_j^T x_j}{\|x_j\|^2}$$

$$\equiv \operatorname*{argmin}_j - \frac{x_i^T x_j}{\|x_i\| \cdot \|x_j\|}$$

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

Locality-Sensitive Hashing

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

- Johnson-Lindenstrauss lemma (paraphrased):
  - Define element 'c' of the k-dimensional 'z_i' by:

$$z_{ic} = w_{c1} x_{i1} + w_{c2} x_{i2} + \cdots + w_{cd} x_{id}$$

  - Where the scalars 'w_{cj}' are samples from a standard normal distribution.
    - We can collect them into a 'k' by 'd' 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 $k = \Omega(\log(d))$.

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

Non-Binary Frequent Itemsets

- We considered measuring things like p(sunglasses=1, sunscreen=1).
- You could consider more general probabilities,
  like (milk > 0.5, egg > 1, lactase ≤ 0).
  - Just as easy to count from the data.
  - The a priori algorithm can be modified to handle this.
    - Though it's more expensive.

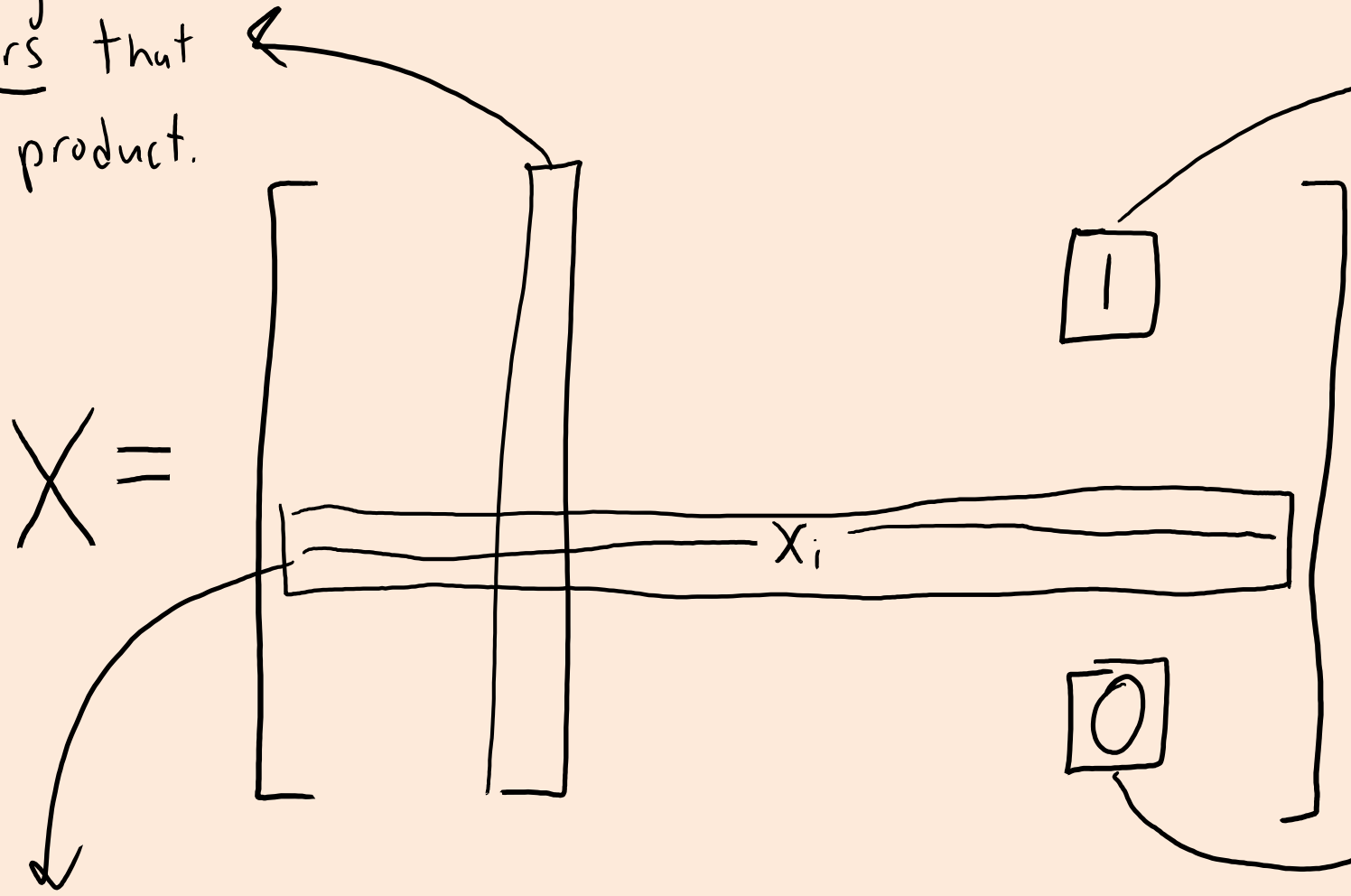
- An application is "deny constraints" for outlier detection:
  - Find the rules that have a really high probability (like 0.99 or 1).
  - Mark the examples not satisfying these rules as outliers.

Association Rules

- Consider two sets of items ‘S’ and ‘T’:
  - For example: S = {sunglasses, sandals} and T = {sunscreen}.
- We can also consider association rules (S => T):
  - If you buy all items ‘S’, you are likely to also buy all items ‘T’.
  - E.g., if you buy sunglasses and sandals, you are likely to buy sunscreen.

User-Product Matrix

Column gives all users that bought product.



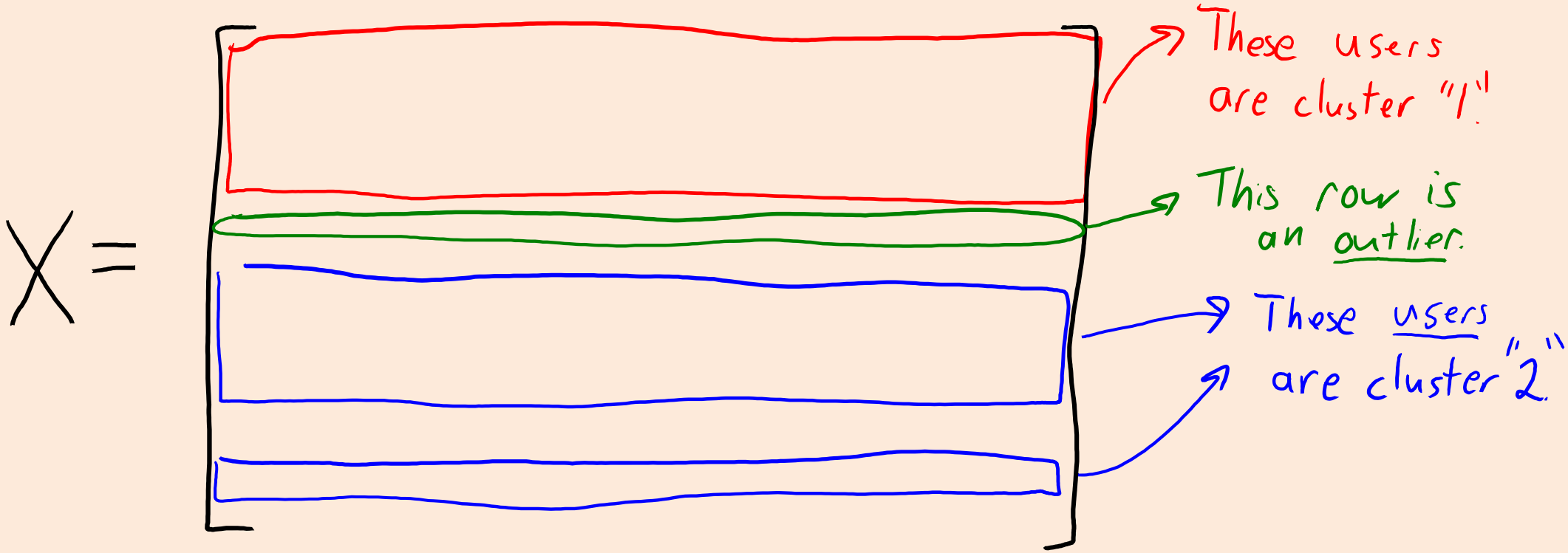
$X_{ij} = 1$ means user 'i' bought item 'j'

$X_{ij} = 0$ means user 'i' did 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.

Clustering User-Product Matrix

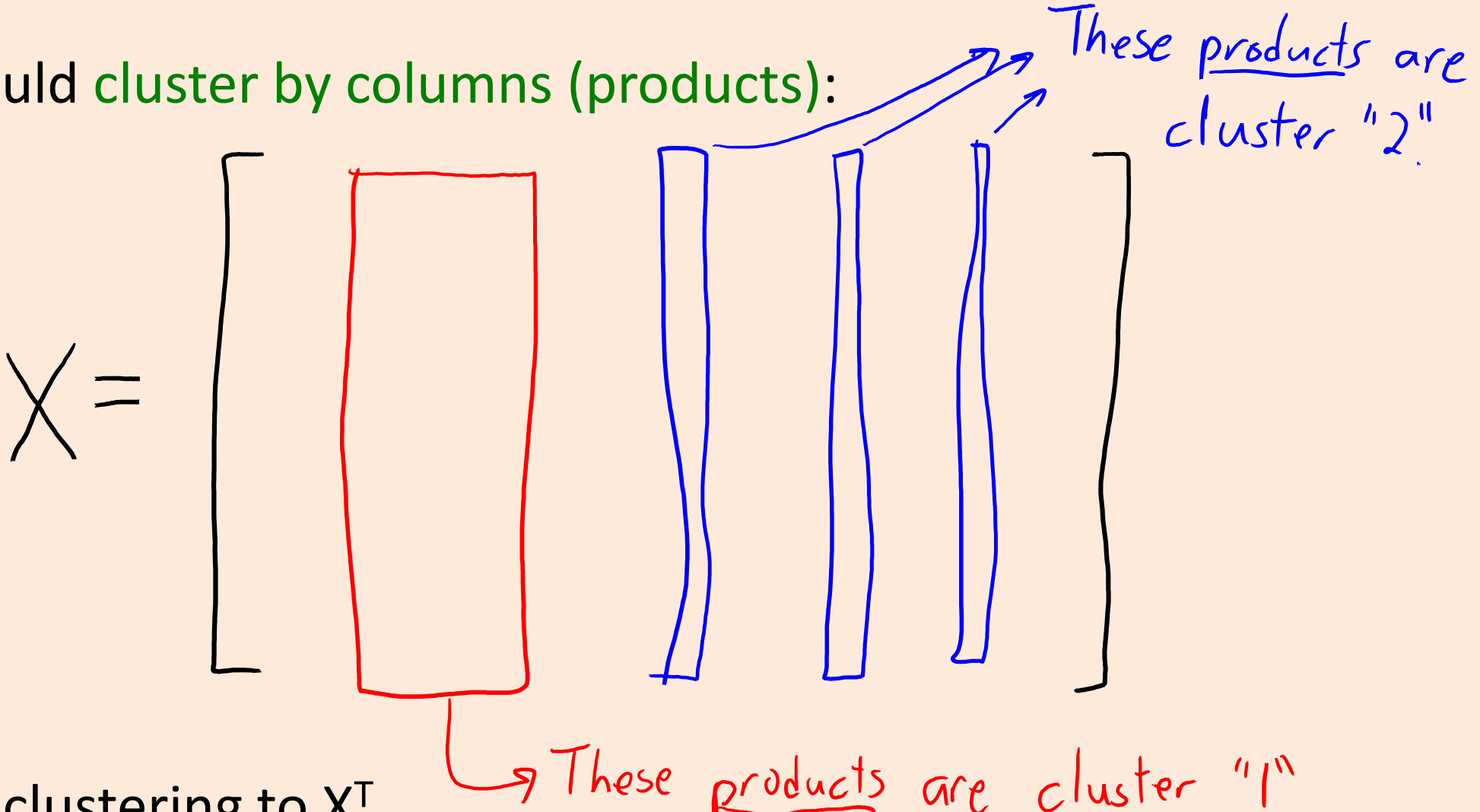
- Normally think of clustering by rows (users):



- We also find outliers by rows.

Clustering User-Product Matrix

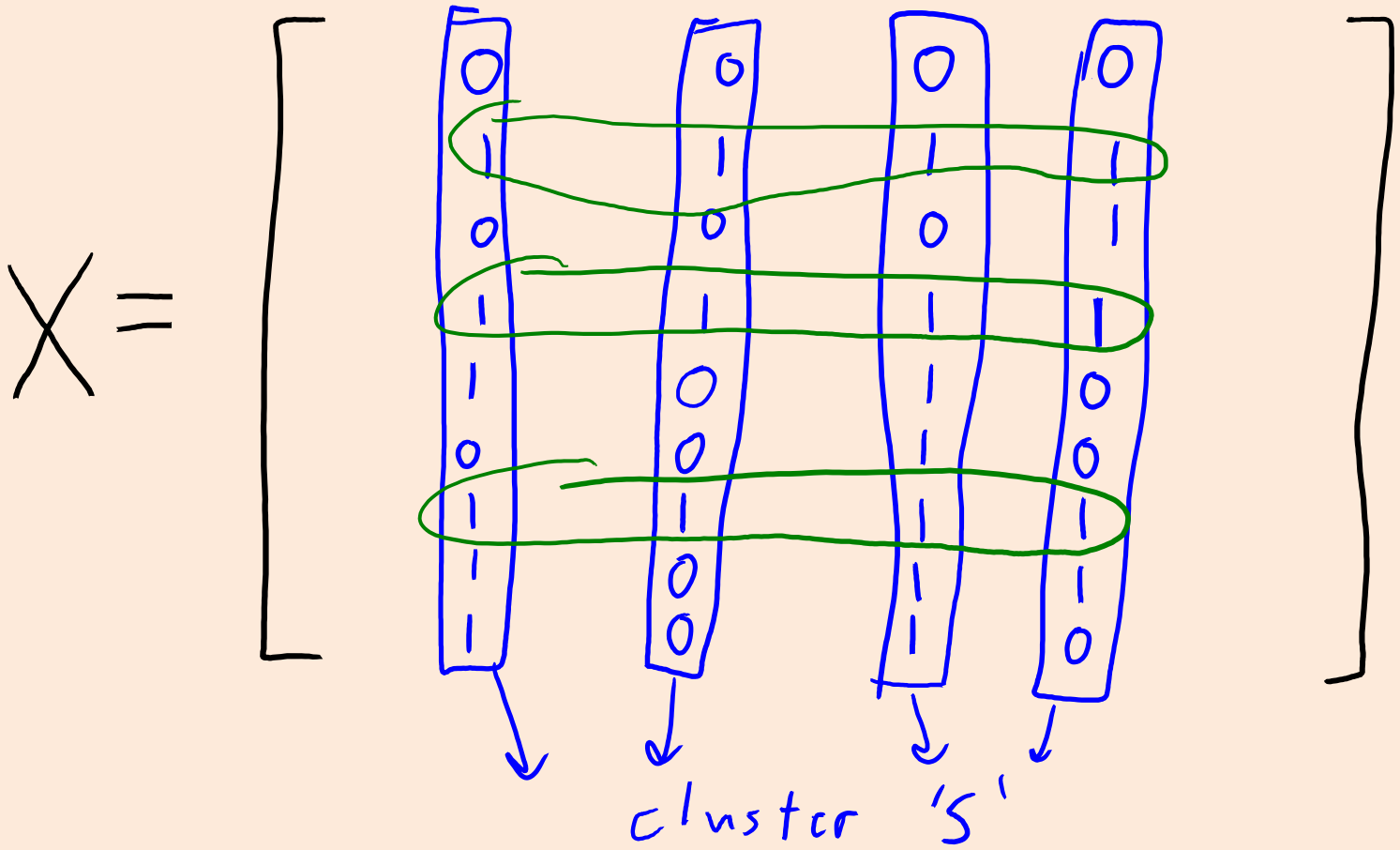
- We could cluster by columns (products):



- Apply clustering to X^T.

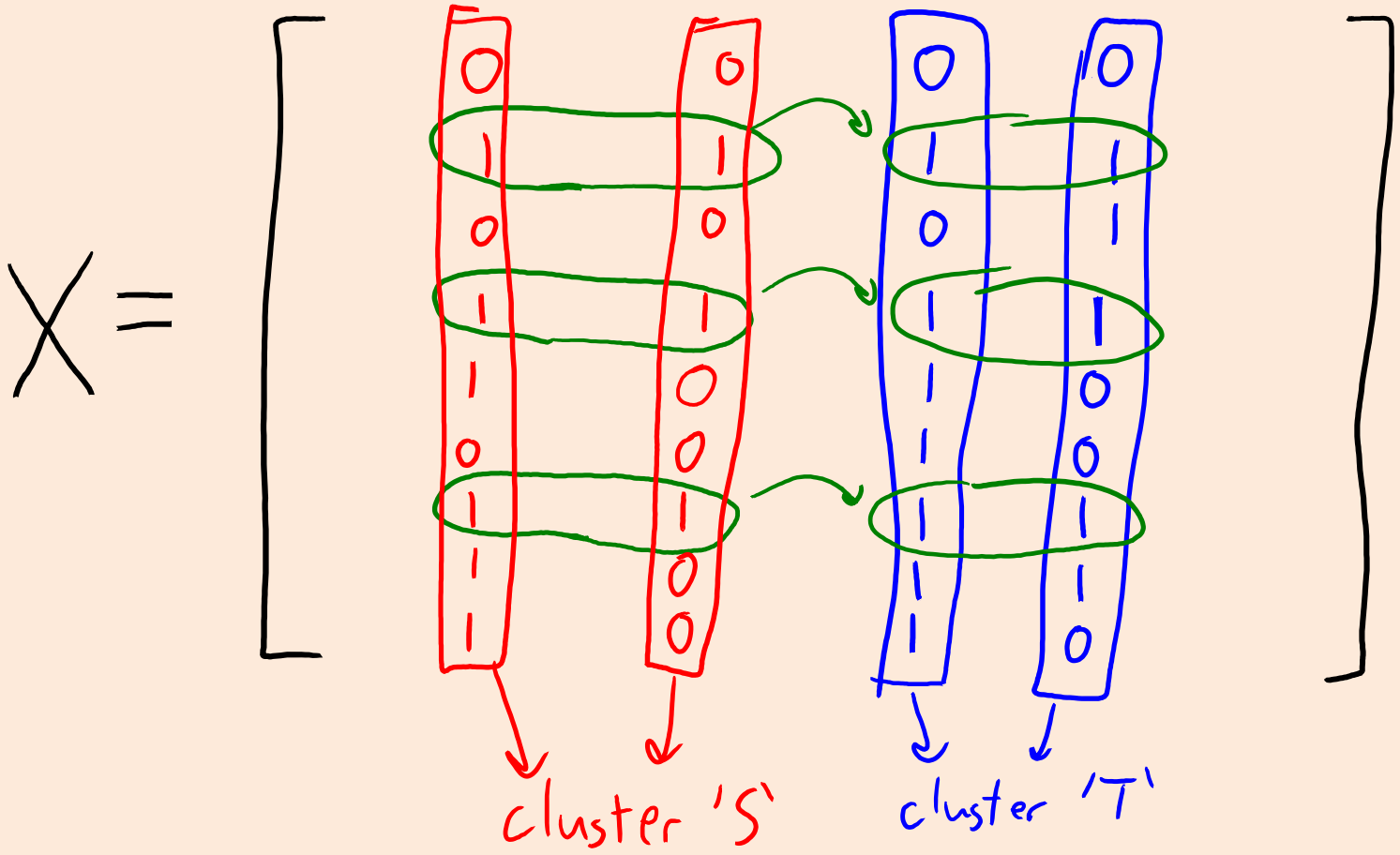
Frequent Itemsets

- Frequent itemsets: we frequently have all ‘1’ values in cluster S.



Association Rules

- Association rules (S => T): all ‘1’ in cluster S => all ‘1’ in cluster T.

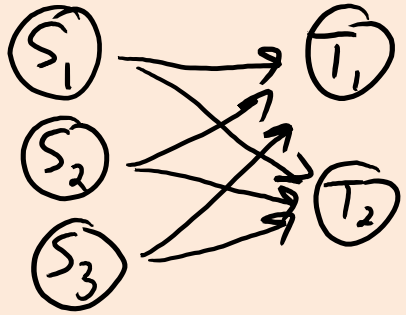


Association Rules

- Interpretation in terms of conditional probability:
  - The rule (S => T) means that p(T = 1 | S = 1) is 'high'.

    I'm using p(T = 1 | S = 1) for $p(T_1 = 1, T_2 = 1, \ldots, T_k = 1 \mid S_1 = 1, S_2 = 1, \ldots, S_c = 1)$.

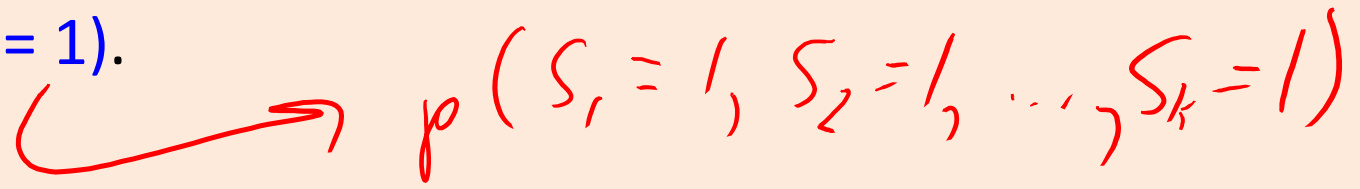
- Association rules are directed but not necessarily causal:
  - p(T | S) ≠ p(S | T).



    - E.g., buying sunscreen doesn't necessarily imply buying sunglasses/sandals:
  - The correlation could be backwards or due to a common cause.
    - E.g., the common cause is that you are going to the beach.

Support and Confidence

- We “score” rule (S => T) by “support” and “confidence”.
  - Running example: {sunglasses,sandals} => suncreen.
- Support:
  - How often does ‘S’ happen?
  - How often were sunglasses and sandals bought together?
  - Marginal probability: p(S = 1).
- Confidence:
  - When ‘S’ happens, how often does ‘T’ happen?
  - When sunglasses+sandals were bought, how often was sunscreen bought?
  - Conditional probability: p(T = 1| S = 1).



Support and Confidence

- We're going to look for rules that:
  1. Happen often (high support), $p(S = 1) \geq$'s'.
  2. Are reliable (high confidence), $p(T = 1 | S = 1) \geq$ 'c'.

- Association rule learning problem:
  - Given support 's' and confidence 'c'.
  - Output all rules with support at least 's' and confidence at least 'c'.

- A common variation is to restrict size of sets:
  - Returns all rules with $|S| \leq k$ and/or $|T| \leq k$.
  - Often for computational reasons.

Generating Rules

- A priori algorithm gives all 'S' with $p(S = 1) \geq s$.
- To generate the rules, we consider subsets of each high-support 'S':
  - If S = {1,2,3}, candidate rules are:
    - {1} => {2,3}, {2} => {1,3}, {3} => {1,2}, {1,2} => {3}, {1,3} => {2}, {2,3} => {1}.
  - There is an exponential number of subsets.
- But we can again prune using rules of probability:

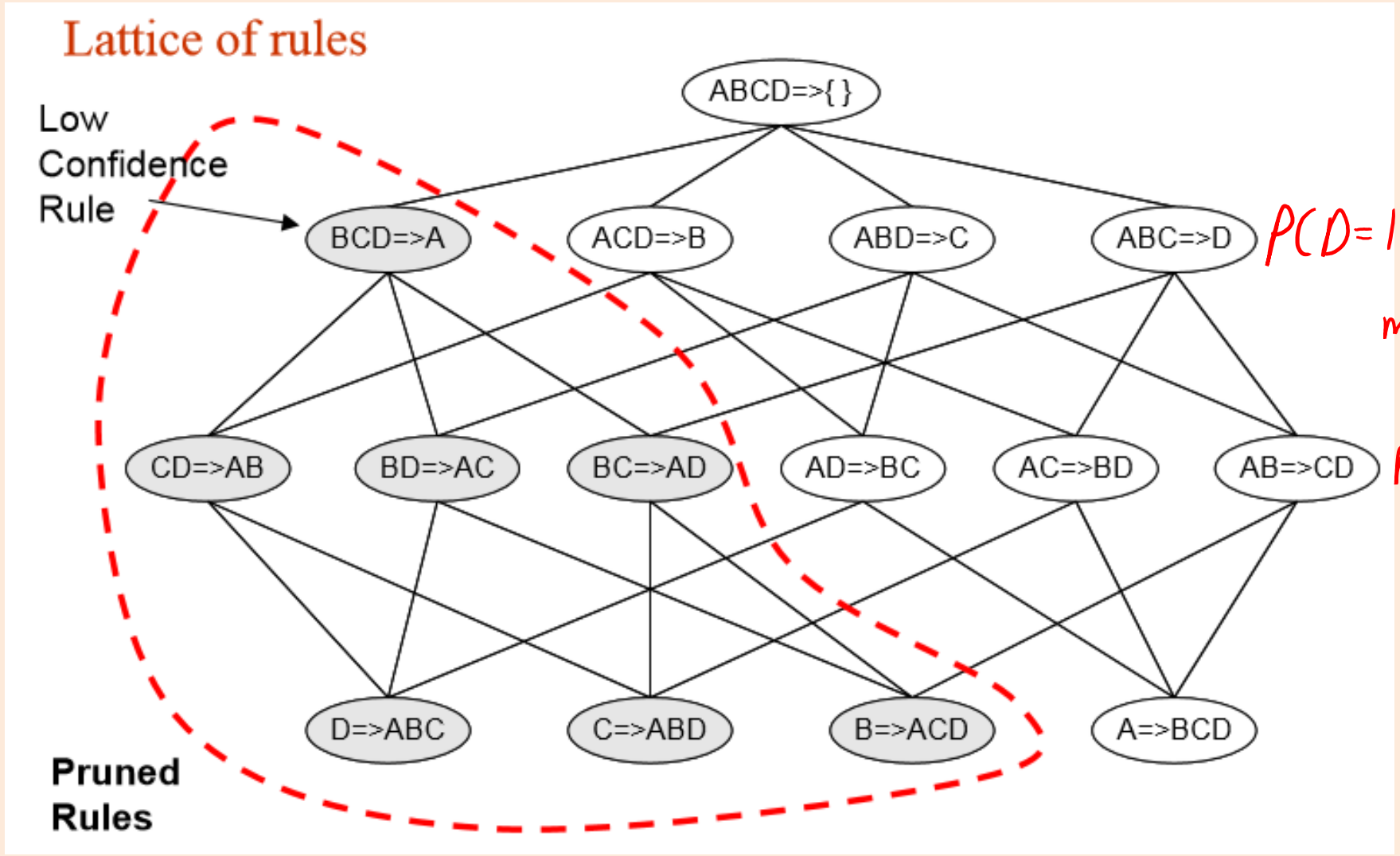
By definition of conditional probability we have $p(T=1 \mid S=1) = \frac{p(S=1, T=1)}{p(S=1)}$

And since $p(S=1) \leq 1$ we have $p(T=1 \mid S=1) \geq p(S=1, T=1)$

By the same logic we have $p(T=1, R=1 \mid S=1, Q=1) \geq p(T=1, R=1, Q=1 \mid S=1)$

- E.g., probability of rolling 2 sixes is higher if you know one di is a 6.

Confident Set Pruning



$P(D=1 \mid A=1, B=1, C=1)$

must be greater than

$P(C=1, D=1 \mid A=1, B=1)$

http://www-users.cs.umn.edu/~kumar/dmbook/dmslides/chap6_basic_association_analysis.pdf

Association Rule Mining Issues

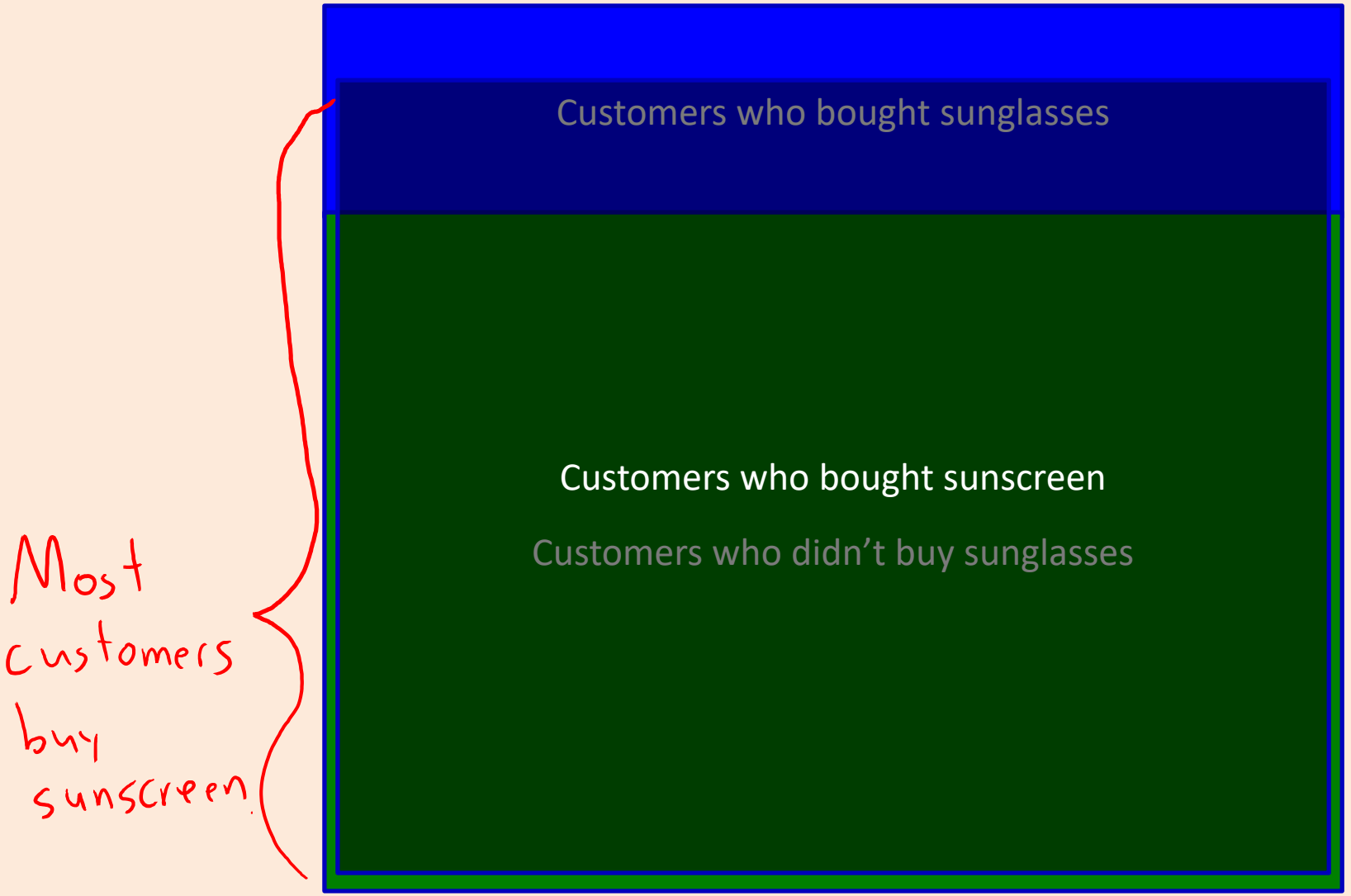
- Spurious associations:
  - Can it return rules by chance?
- Alternative scores:
  - Support score seems reasonable.
  - Is confidence score the right score?
- Faster algorithms than a priori:
  - ECLAT/FP-Growth algorithms.
  - Generate rules based on subsets of the data.
  - Cluster features and only consider rules within clusters.

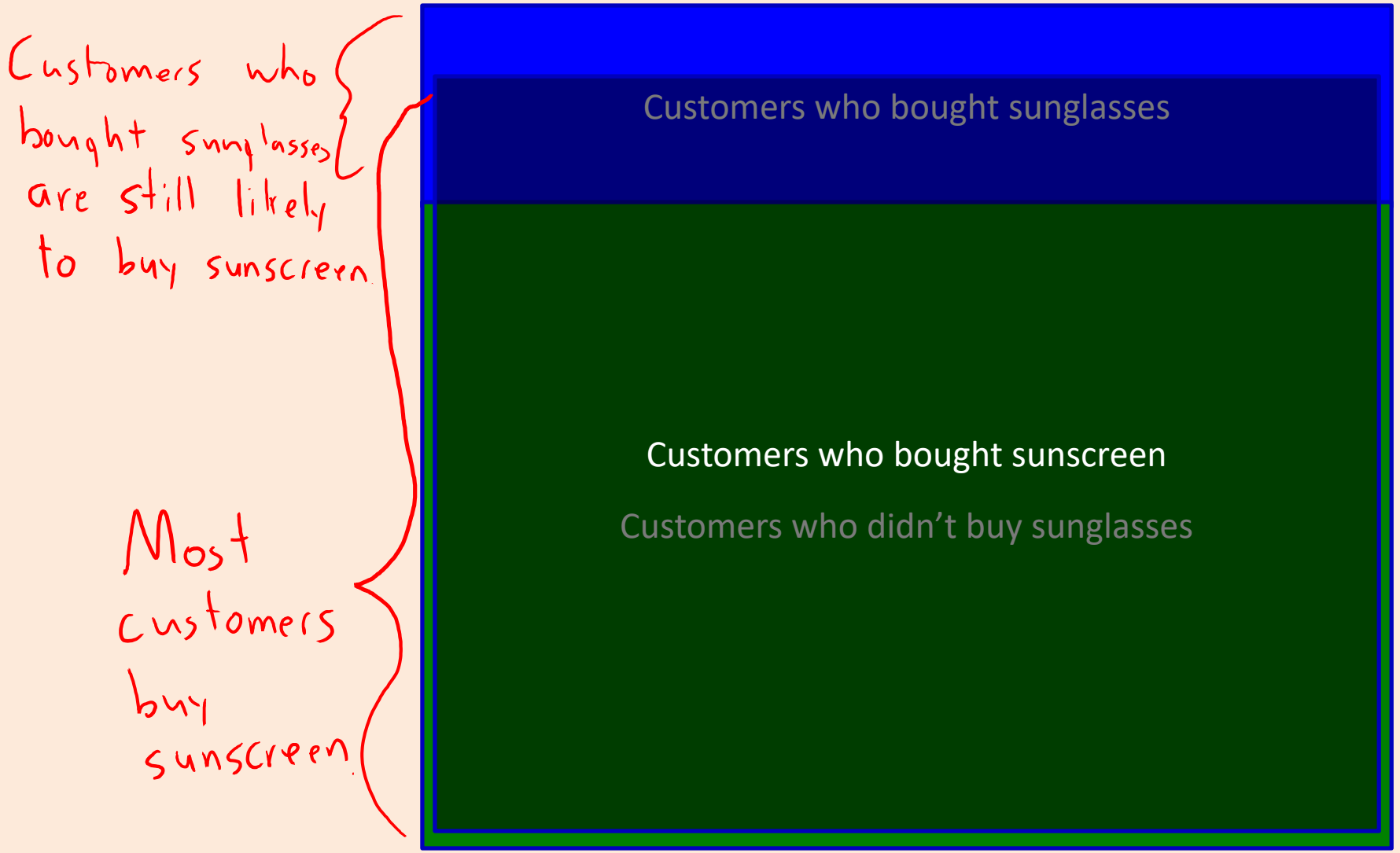
Problem with Confidence

- Consider the “Sunscreen Store”:
  - Most customers go there to buy sunscreen.

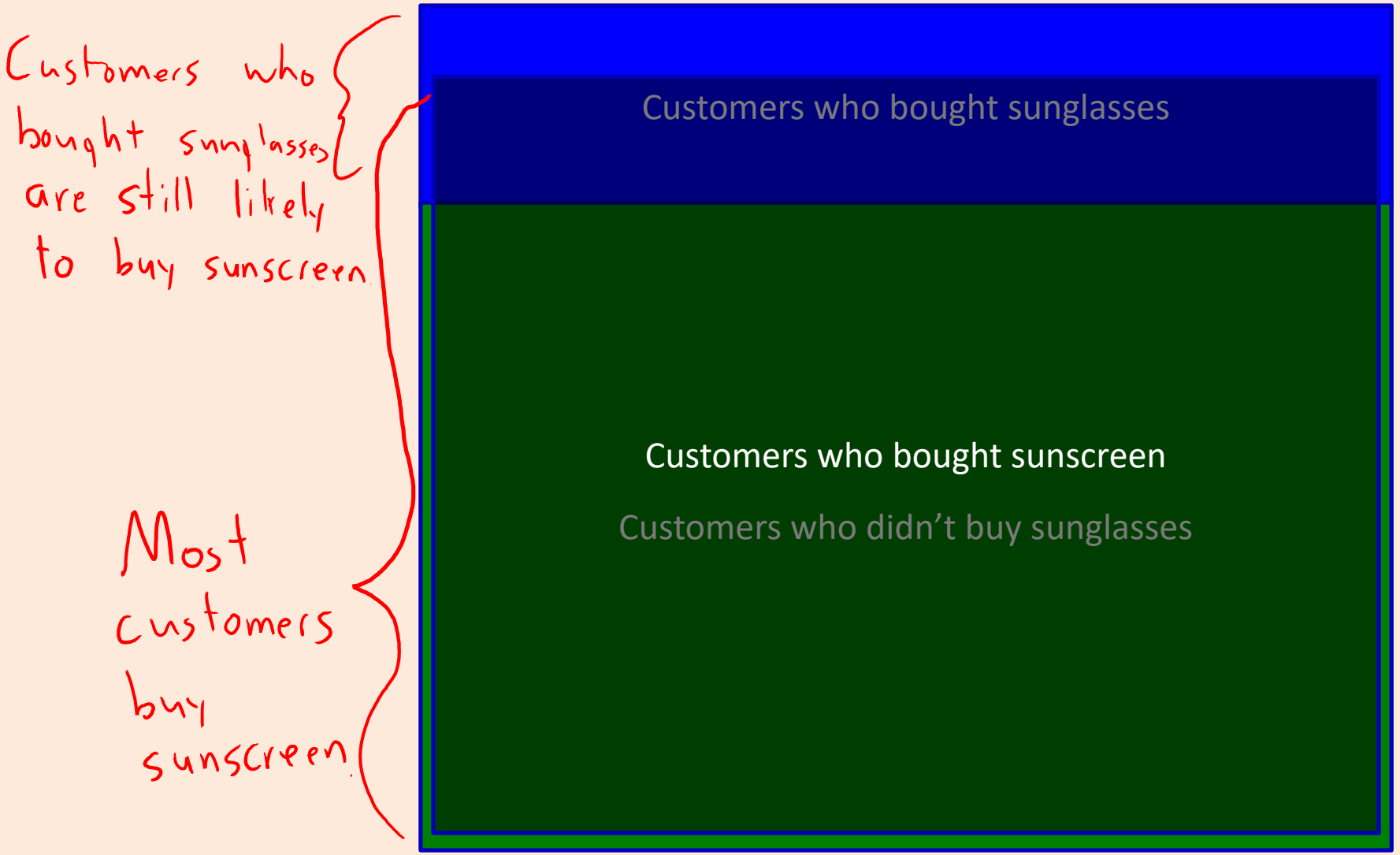
- Now consider rule (sunglasses => sunscreen).
  - If you buy sunglasses, it could mean you weren’t there for sunscreen:
    - p(sunscreen = 1| sunglasses = 1) < p(sunscreen = 1).
  - So (sunglasses => sunscreen) could be a misleading rule:
    - You are less likely to buy sunscreen if you buy sunglasses.
  - But the rule could have high confidence.


Customers who bought sunglasses
Customers who didn’t buy sunglasses


Customers who bought sunglasses
Customers who bought sunscreen
Customers who didn't buy sunglasses
Most customers buy sunscreen.


Customers who bought sunglasses are still likely to buy sunscreen
Most customers buy sunscreen
Customers who bought sunglasses
Customers who bought sunscreen
Customers who didn't buy sunglasses


Customers who bought sunglasses are still likely to buy sunscreen
Most customers buy sunscreen
Customers who bought sunglasses
Customers who bought sunscreen
Customers who didn't buy sunglasses
But knowing that they bought sunglasses make it less likely they bought sunscreen



Customers who bought sunglasses are still likely to buy sunscreen

But knowing that they bought sunglasses make it less likely they bought sunscreen

Most customers buy sunscreen

- One alternative to confidence is "lift":
  - How much more likely does 'S' make us to buy 'T'?

Normalize by probability of buying if you don't know 'S'

Confidence

$$Lift(S \Rightarrow T) = \frac{p(T=1 \mid S=1)}{p(T=1)}$$

Sequential Pattern Analysis

- Finding patterns in data organized according to a sequence:
  - Customer purchases:
    - 'Star Wars' followed by 'Empire Strikes Back' followed by 'Return of the Jedi'.
  - Stocks/bonds/markets:
    - Stocks going up followed by bonds going down.
- In data mining, called sequential pattern analysis:
  - If you buy product A, are you likely to buy product B at a later time?
- Similar to association rules, but now order matters.
  - Many issues stay the same.
- Exist sequential versions of many association rule methods:
  - Generalized sequential pattern (GSP) algorithm is like a priori algorithm.

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.
  - 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
    - But this is now changing and ML is starting to appear in anti-virus software:
      - http://icitech.org/wp-content/uploads/2017/02/ICIT-Analysis-Signature-Based-Malware-Detection-is-Dead.pdf

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.
  - For sequences may also use “suffix trees” to speed up finding hash keys.

- 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 example ‘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 $h(x_i)=h(x_j)$.