CPSC 340: Machine Learning and Data Mining

Recommender Systems

Original version of these slides by Mark Schmidt, with modifications by Mike Gelbart.

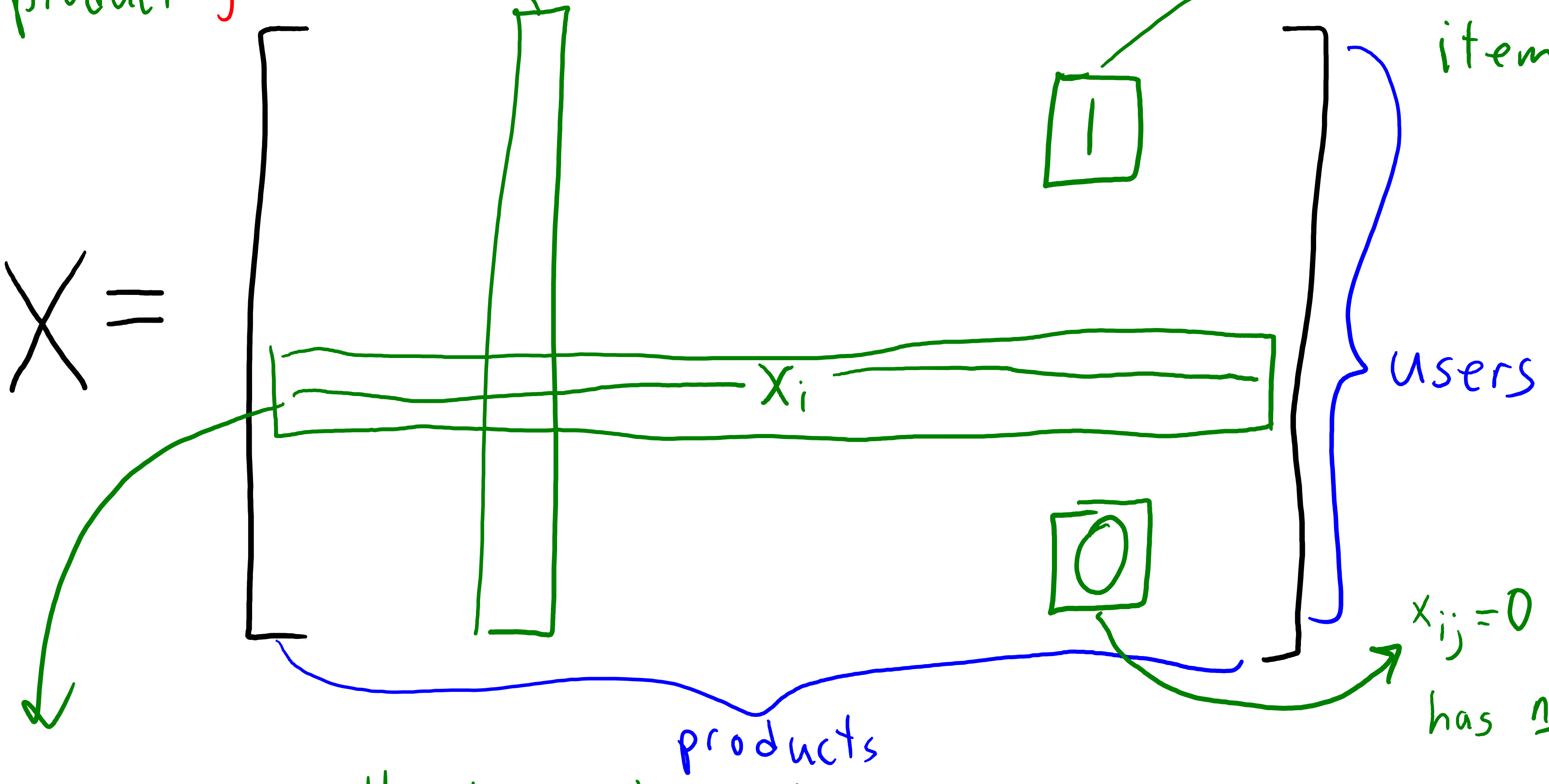
Motivation: Product Recommendation

- 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 x^j gives all users that bought product 'j'

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

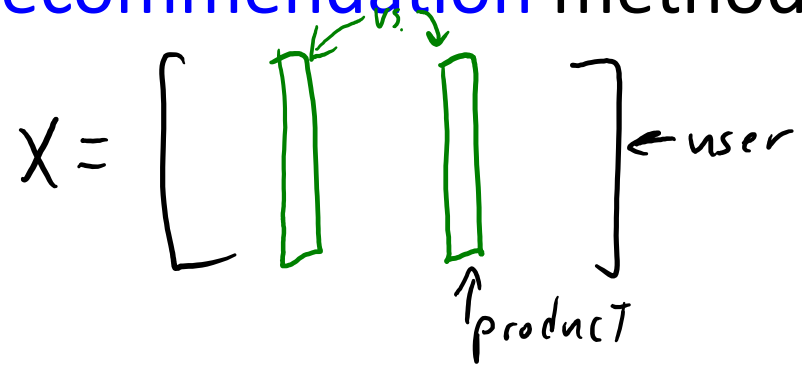


$x_{ij} = 0$ means user 'i' has not buy item 'j'

Row x_i gives all items bought by user 'i'.

Amazon Product Recommendation

- Amazon product recommendation method:



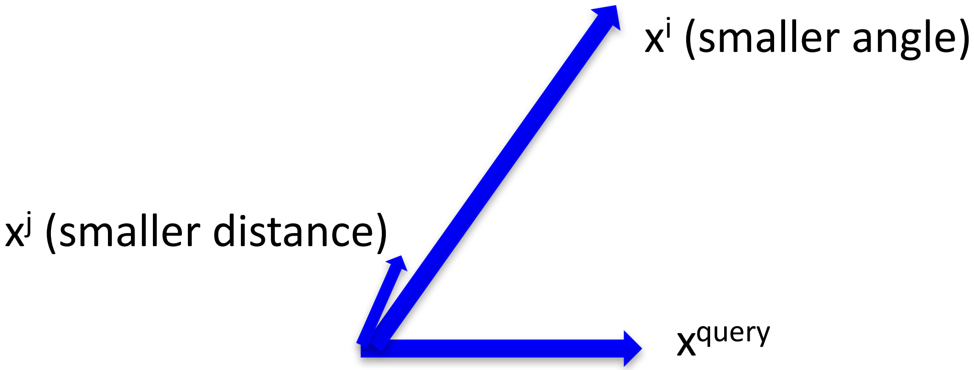
- Find the KNNs across columns.
  - Find indices ‘j’ minimizing a distance measure between x^{query} and x^j.
  - Euclidean distance, normalized Euclidean distance, cosine similarity

Cosine similarity

- The cosine similarity of vectors ‘x’ and ‘y’ is defined as $\frac{x^T y}{||x|| \cdot ||y||}$
  - “Maximize cosine of the angle between x^{query} and x^j”
  - Yields the same ranked KNNs as normalized Euclidean distance
    - Normalized Euclidean distance: first divide each column by its norm, $x^i/||x^i||$.
  - If X is a binary matrix, dot product counts the number of users in common

Cosine similarity

- The cosine similarity of vectors 'x' and 'y' is defined as $$\frac{x^T y}{||x|| \cdot ||y||}$$
- Cosine similarity finds more popular items than Euclidean distance
  - In high dimension, and with sparse vectors, dot products are small
  - Thus most angles are large (there are so many users → directions!)
  - Very small vectors have a Euclidean distance of around $||x^{query}||$
  - This might be "closer" than larger vectors with smaller angles



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
  - But data set is still enormous in the Amazon example.
- We’ve seen a lot of “closest point” problems:
  - KNN classification.
  - K-means clustering.
  - Density-based clustering.
  - Amazon product recommendation.
- Bonus slides: strategies for speeding this up.

(pause)

Recommender Systems

- There are several types recommendation problems.
  - We might want to recommend items given an item.
    - Amazon product recommendation.
  - We might want to recommend items given a user.
    - E.g. Amazon homepage, Netflix homepage.
  - Or a combination (personalized item-based recommendation).

- Recommender systems are now everywhere:
  - Music, news, books, jokes, experts, restaurants, friends, dates, etc.

- Often this problem is framed as predicting missing ratings.

Recommender System Motivation: Netflix Prize

- Netflix Prize:
  - 100M ratings from 0.5M users on 18k movies.
  - Grand prize was $1M for first team to reduce squared error by 10%.
  - Started on October 2nd, 2006.
  - Netflix's system was first beat October 8th.
  - 1% error reduction achieved on October 15th.
  - Steady improvement after that.
    - ML methods soon dominated.
  - One obstacle was 'Napolean Dynamite' problem:
    - Some movie ratings seem very difficult to predict.
    - Should only be recommended to certain groups.

Lessons Learned from Netflix Prize

- Prize awarded in 2009:
  - Ensemble method that averaged 107 models.
  - Increasing diversity of models more important than improving models.
- Winning entry (and most entries) used collaborative filtering:
  - Methods that only looks at ratings, not features of movies/users.
- A simple collaborative filtering method that does really well (7%):
  - "Regularized matrix factorization". Now adopted by many companies.

Collaborative Filtering Problem

- Collaborative filtering is 'filling in' the user-item matrix:

$$Y = \text{user}\left\{\underbrace{\begin{bmatrix} ? & 4 & 3 & 2 & 3 & 3 \\ 2 & 1 & ? & 5 & ? & 5 \\ ? & 1 & ? & 5 & 5 & 5 \\ 2 & 3 & 3 & ? & ? & ? \end{bmatrix}}_{\text{movie}}\right.$$

"Green Lantern"

"Ryan Reynolds"

- We have some ratings available with values {1,2,3,4,5}.
- We want to predict ratings "?" by looking at available ratings.

Collaborative Filtering Problem

- Collaborative filtering is 'filling in' the user-item matrix:

$$Y = \text{user}\left\{ \underbrace{\begin{bmatrix} ? & 4 & 3 & 2 & 3 & 3 \\ 2 & 1 & ? & 5 & ? & 5 \\ ? & 1 & ? & 5 & 5 & 5 \\ 2 & 3 & 3 & ? & ? & ? \end{bmatrix}}_{\text{movie}} \right.$$

"Green Lantern"

"Ryan Reynolds"

Similar user

Similar movie

- What rating would "Ryan Reynolds" give to "Green Lantern"?
  - Why is this not completely crazy? We may have similar users and movies.

Matrix Factorization for Collaborative Filtering

- Our standard latent-factor model for entries in matrix 'Y':

$$y_{ij} \approx (w_j)^T z_i$$

- User 'i' has latent features z_i.
  - Feature 1 could be "likes romantic comedies"
- Movie 'j' has latent features w_j.
  - Feature 1 could be "has elements of a romantic comedy"
- We're automatically learning both the "weights" and the "features"
  - There's a w-z symmetry that's not present in linear regression or even PCA

Matrix Factorization for Collaborative Filtering

- Our standard latent-factor model for entries in matrix 'Y':

$$y_{ij} \approx (w^j)^T z_i$$

- Our loss functions sums over available ratings 'R':

$$f(Z,W) = \sum_{(i,j) \in R} \left((w^j)^T z_i - y_{ij} \right)^2 + \frac{\lambda_1}{2}\|Z\|_F^2 + \frac{\lambda_2}{2}\|W\|_F^2$$

- And we add L2-regularization to both types of features.
  - Basically, this is regularized PCA on the available entries of Y:
  - But with a very different interpretation

$$\underset{n \times d}{Y} \approx \underset{n \times k}{Z}\,\underset{k \times d}{W}$$

- We cannot use SVD because of the missing entries
  - Can use GD, SGD, alternating least squares
  - Weird extra regularization: keep the missing entries but with low weights

Adding Global/User/Movie Biases

- Our standard latent-factor model for entries in matrix 'Y':

$$\hat{y}_{ij} = (w^j)^T z_i$$

- Sometimes we don't assume the y_{ij} have a mean of zero:
  - We could add bias β reflecting average overall rating:

$$\hat{y}_{ij} = \beta + (w^j)^T z_i$$

  - We could also add a user-specific bias β_i and item-specific bias β_j.

$$\hat{y}_{ij} = \beta + \beta_i + \beta_j + (w^j)^T z_i$$

    - Some users rate things higher on average, and movies are rated better on average.
    - These might also be regularized.

Beyond Accuracy in Recommender Systems

- Winning system of Netflix Challenge was never adopted.
- Other issues important in recommender systems:
  - Diversity: how different are the recommendations?
    - If you like 'Battle of Five Armies Extended Edition', recommend Battle of Five Armies?
    - Even if you really really like Star Wars, you might want non-Star-Wars suggestions.
  - Persistence: how long should recommendations last?
    - If you keep not clicking on 'Hunger Games', should it remain a recommendation?
  - Trust: tell user *why* you made a recommendation.
    - Quora gives explanations for recommendations.
  - Social recommendation: what did your friends watch?
  - Freshness: people tend to get more excited about *new/surprising* things.
    - Collaborative filtering does not predict well for new users/movies.
      - New movies don't yet have ratings, and new users haven't rated anything.

Unsupervised vs. Supervised Recommenders

- Main types of approaches:
  1. Collaborative filtering.
     - "Unsupervised" learning (have label matrix 'Y' but no features):
       - We only have labels y_{ij} (rating of user 'i' for movie 'j').
     - Example: Amazon recommendation algorithm.
  2. Content-based filtering.
     - Supervised learning:
       - Extract features x_i of users and items, building model to predict rating y_i given x_i.
       - Apply model to prediction for new users/items.
     - Example: Gmail's "important messages"

Content-Based vs. Collaborative Filtering

- Our latent-factor approach to collaborative filtering (Part 4):

$$\hat{y}_{ij} = \underbrace{(w^j)^T}_{\text{"hidden" features of movie}} \underbrace{z_i}_{\text{"hidden" features of user}}$$

  - Learns about each user/movie, but can't predict on new users/movies.
- A linear model approach to content-based filtering (Part 3):

$$\underbrace{\hat{y}_{ij} = w^T x_{ij}}$$

Our usual $y_i = w^T x_i$ supervised learning setup.

  - Here x_{ij} is a **vector of features** for the movie/user.
    - Usual supervised learning setup: 'y' would contain all the y_{ij}, X would have x_{ij} as rows.
  - Can predict on new users/movies, but can't learn about each user/movie.

Hybrid Approaches

- Hybrid approaches combine content-based/collaborative filtering:
  - SVDfeature (won "KDD Cup" in 2011 and 2012).

Extra factors we learn for specific users and movies.

$$\hat{y}_{ij} = \beta + \beta_i + \beta_j + w^T x_{ij} + (w^j)^T z_i$$

Average rating across all users/movies

Average rating for user 'i'

Average for movie 'j'

Linear model based on user/movie features x_{ij}.

Latent features z_i for user 'i' and latent features w_j for movie 'j'

Standard supervised learning: can predict for new users/movies

  - Note that x_{ij} is a feature *vector*. Also, 'w' and 'w^j' are different parameters.

Social Regularization

- Many recommenders are now connected to social networks.
  - "Login using you Facebook account".

- Often, people like similar movies to their friends.

- Recent recommender systems use social regularization.
  - Add a "regularizer" encouraging friends' weights to be similar:

$$\frac{\lambda}{2} \sum_{(i,j) \in \text{"friends"}} \|z_i - z_j\|^2$$

  - If we get a new user, recommendations are based on friend's preferences.

(pause)

Association Rules

- Consider two sets of items ‘S’ and ‘T’:
  - For example: S = {sunglasses, sandals} and T = {sunscreen}.
- We’re going to 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.

Association Rules vs. Clustering

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

"These rows are in cluster 1"

X=

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

Association Rules vs. Clustering

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

- Association rules:
  - Which features occur together?
  - Relating groups of columns.

$X =$

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

$\underbrace{\text{Sunglasses} \quad \text{Sandals}}_{S} \quad \underbrace{\text{Sunscreen}}_{T}$

If these two columns are "1"

Then this value is probably "1"

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).

$p(S_1 = 1, S_2 = 1, \ldots, S_k = 1)$

- 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).

Finding Sets with High Support

- We can do this with the “a priori algorithm”
- Finding high confidence is easier
- See bonus slide for details

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.
- Other associations you might not want to act on:
  - Beer and diapers

Summary

- Recommender systems try to recommend products.
- Nearest neighbour recommenders
  - Find similar items using nearest neighbour search.
- Collaborative filtering tries to fill in missing values in a matrix.
  - Matrix factorization is a common approach.
  - This can be turned into a recommendation in two ways:
    - Nearest neighbours in latent space
    - Find items with high predicted ratings
- Association Rules: (S => T) means seeing S means T is likely.
- Strategies for fitting linear models with binary/categorical features.
- Global vs. local features allows 'personalized' predictions.

Linear Models with Binary Features

- What is the effect of a binary feature on linear regression?

Year	Gender
1975	1
1975	0
1980	1
1980	0

Height
1.85
2.25
1.95
2.30

- Adding a bias w_0, our linear model is:

$$height = w_0 + w_1 * year + w_2 * gender$$

- The 'gender' variable causes a change in y-intercept:

If $gender == 0$ then $height = w_0 + w_1 * year$

If $gender == 1$ then $height = w_0 + w_1 * year + w_2$

Linear Models with Binary Features

- What if different genders have different slopes?
  - You can use gender-specific feature (as if *d*=4).
  - This is equivalent to separating the data set by gender and training 2 models

Year	Gender
1975	1
1975	0
1980	1
1980	0

$\Rightarrow$

Bias (gender = 1)	Year (gender = 1)	Bias (gender = 0)	Year (gender = 0)
1	1975	0	0
0	0	1	1975
1	1980	0	0
0	0	1	1980

distance = w_0 + w_1 * year (if gender = 1)

distance = w_3 + w_4 * year (if gender = 0)

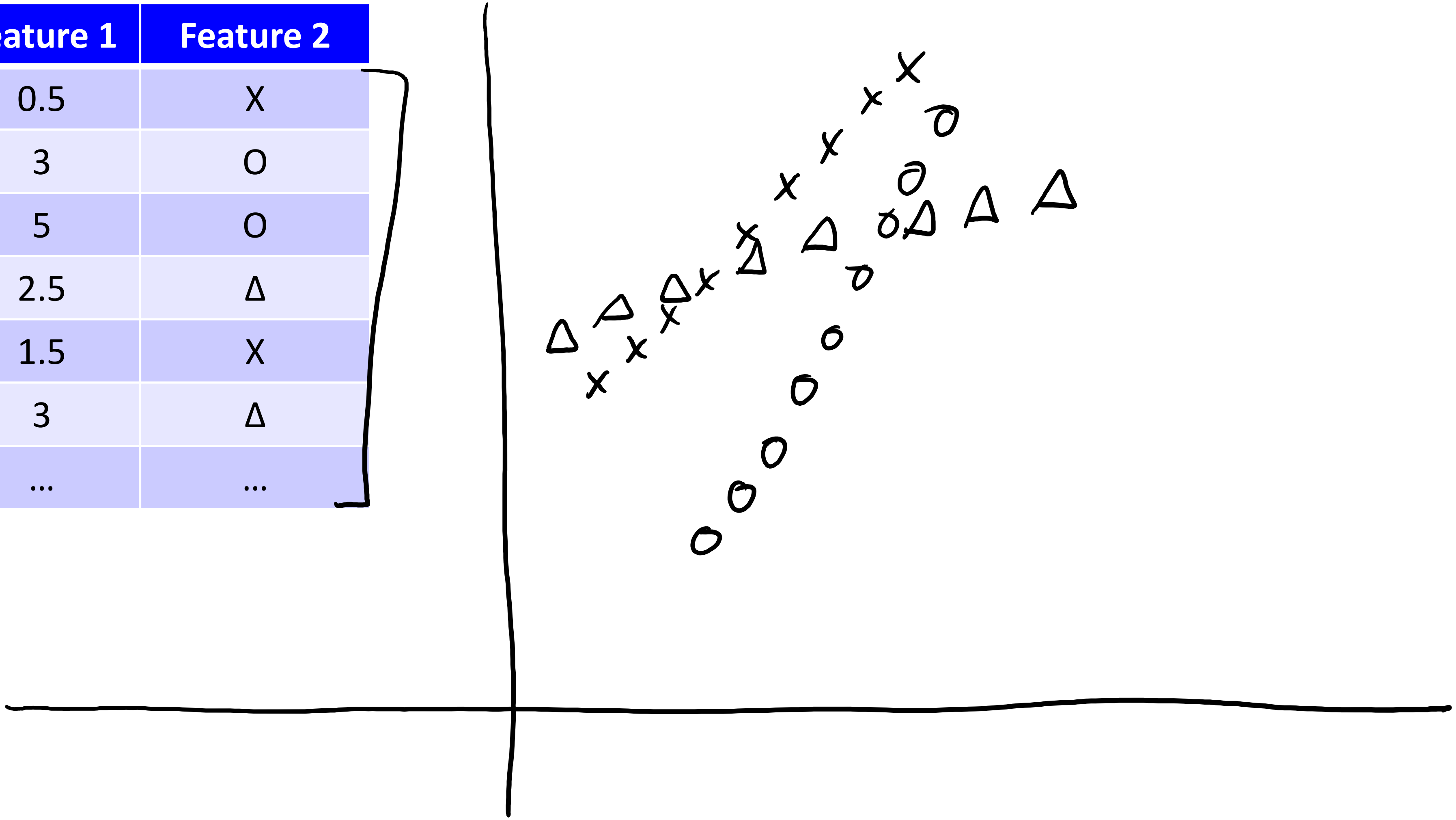
separate bias

separate slope

The same holds for more categories

$X =$

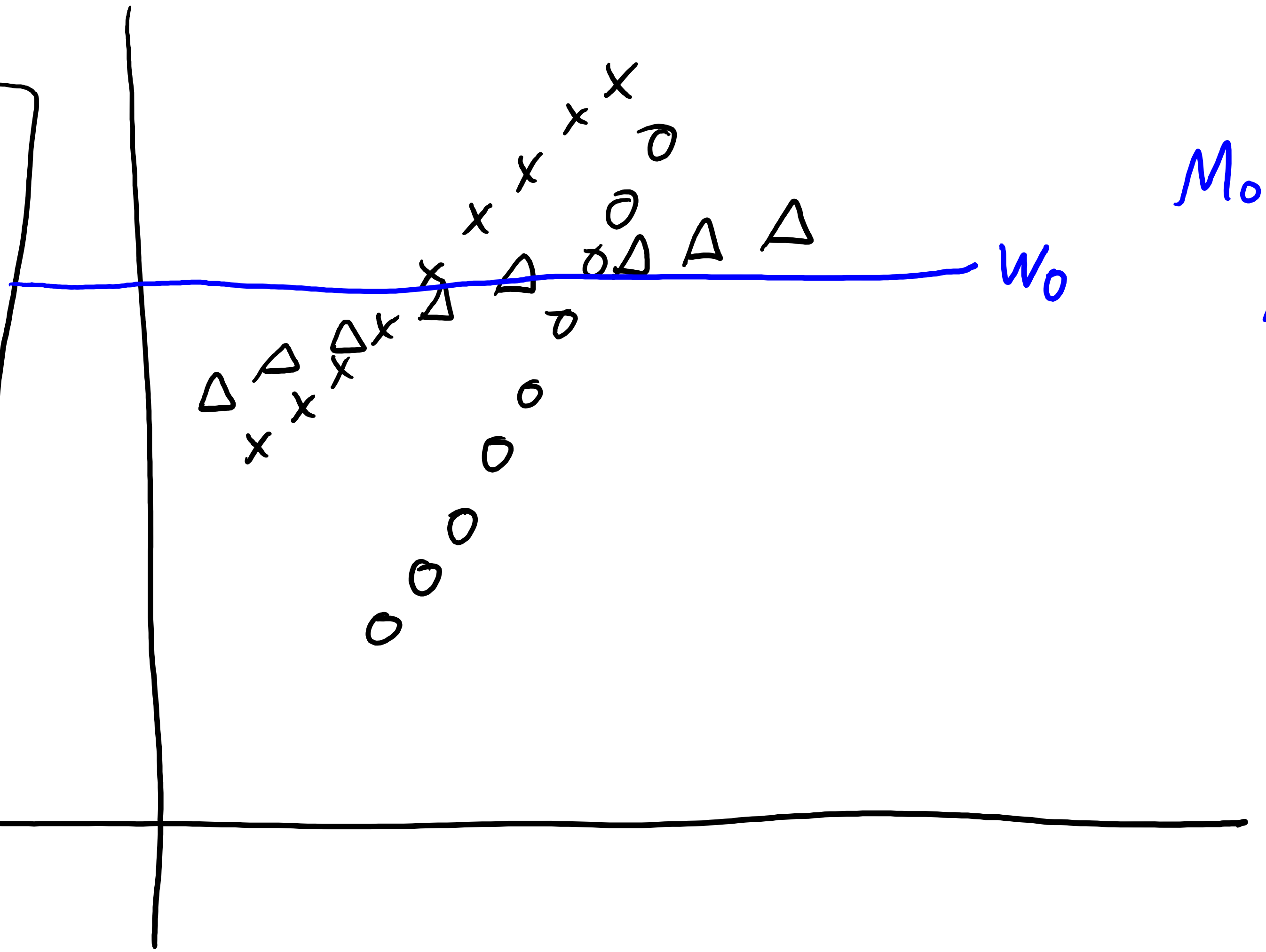
Feature 1	Feature 2
0.5	X
3	O
5	O
2.5	Δ
1.5	X
3	Δ
…	…



Linear Models with Categorical Features

$X =$

Feature 1	Feature 2
0.5	X
3	O
5	O
2.5	Δ
1.5	X
3	Δ
…	…



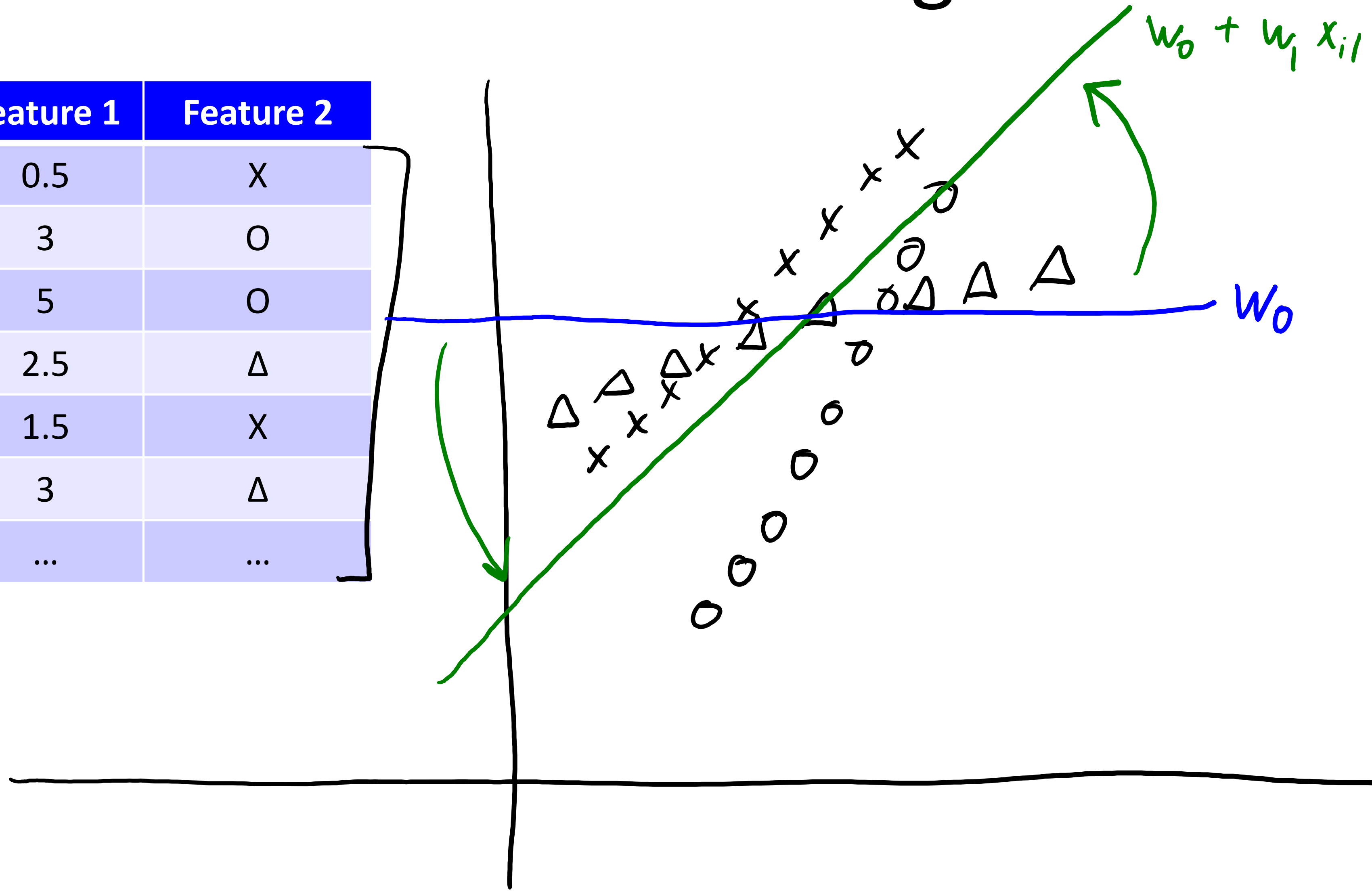
Model 1: only <u>bias</u>

$y_i = w_0$

Linear Models with Categorical Features

$X =$

Feature 1	Feature 2
0.5	X
3	O
5	O
2.5	Δ
1.5	X
3	Δ
…	…



Model 1: only bias

$y_i = w_0$

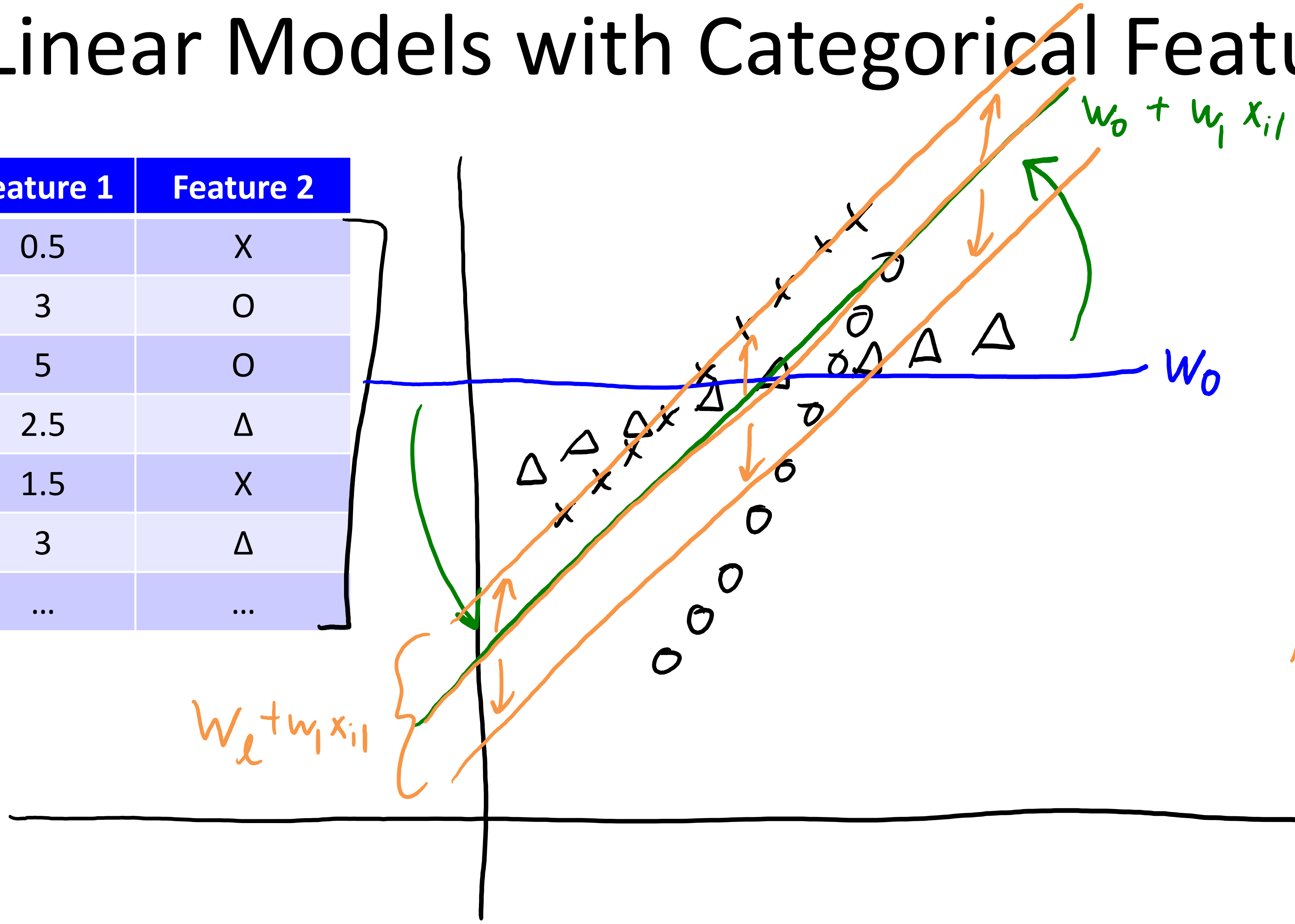
Model 2: bias + feature 1

$y_i = w_0 + w_1 x_{i1}$

Linear Models with Categorical Features

$X =$

Feature 1	Feature 2
0.5	X
3	O
5	O
2.5	Δ
1.5	X
3	Δ
…	…



Model 1: only bias

$y_i = w_0$

Model 2: bias + feature 1

$y_i = w_0 + w_1 x_{i1}$

Model 3: "local" bias + feature 1

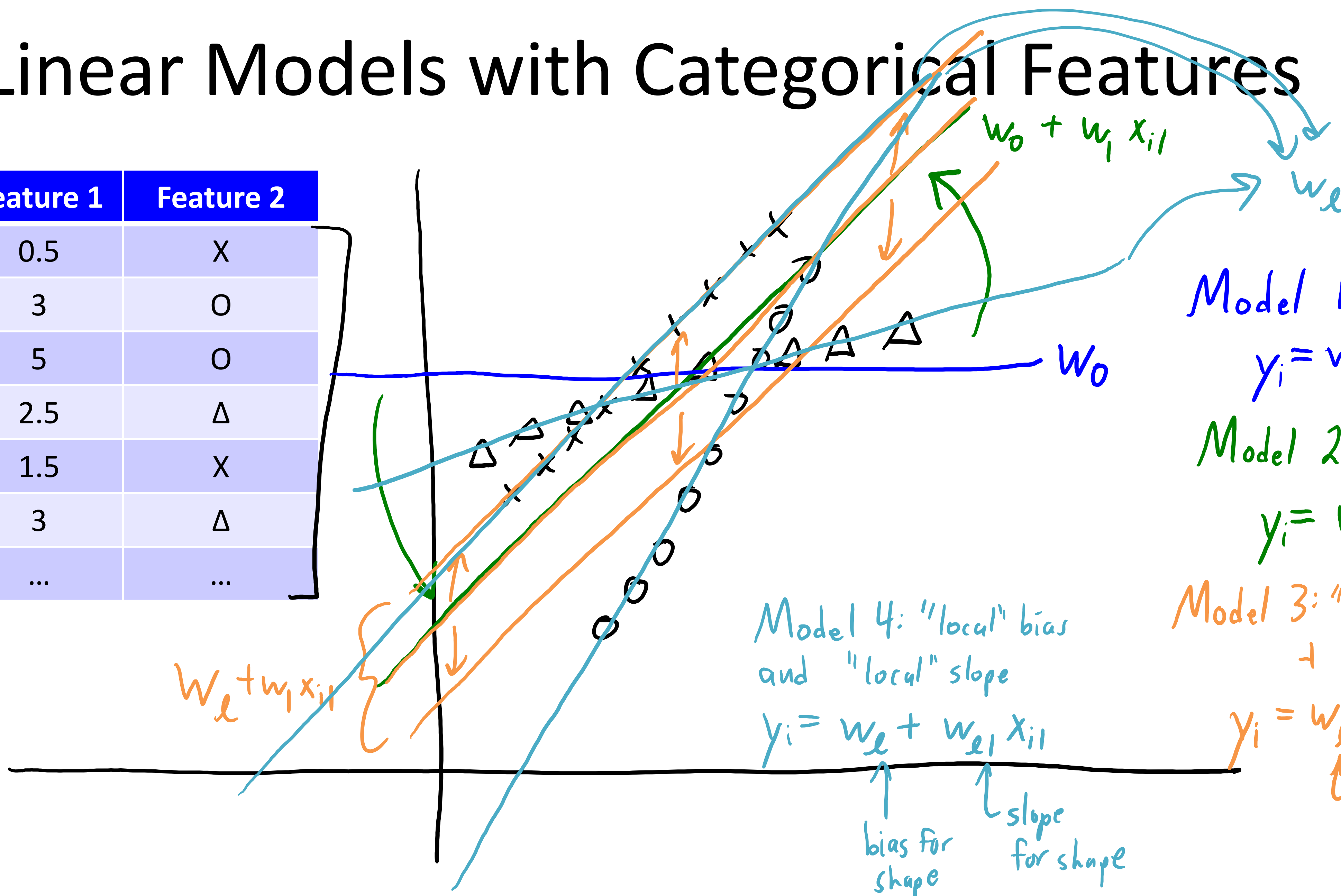
$y_i = w_\ell + w_1 x_{i1}$

shape

Linear Models with Categorical Features

$X =$

Feature 1	Feature 2
0.5	X
3	O
5	O
2.5	Δ
1.5	X
3	Δ
…	…



Model 1: only <u>bias</u>

$y_i = w_0$

Model 2: bias + feature 1

$y_i = w_0 + w_1 x_{i1}$

Model 3: "local" bias + feature 1

$y_i = w_\ell + w_1 x_{i1}$

shape

Model 4: "local" bias and "local" slope

$y_i = w_\ell + w_{\ell 1} x_{i1}$

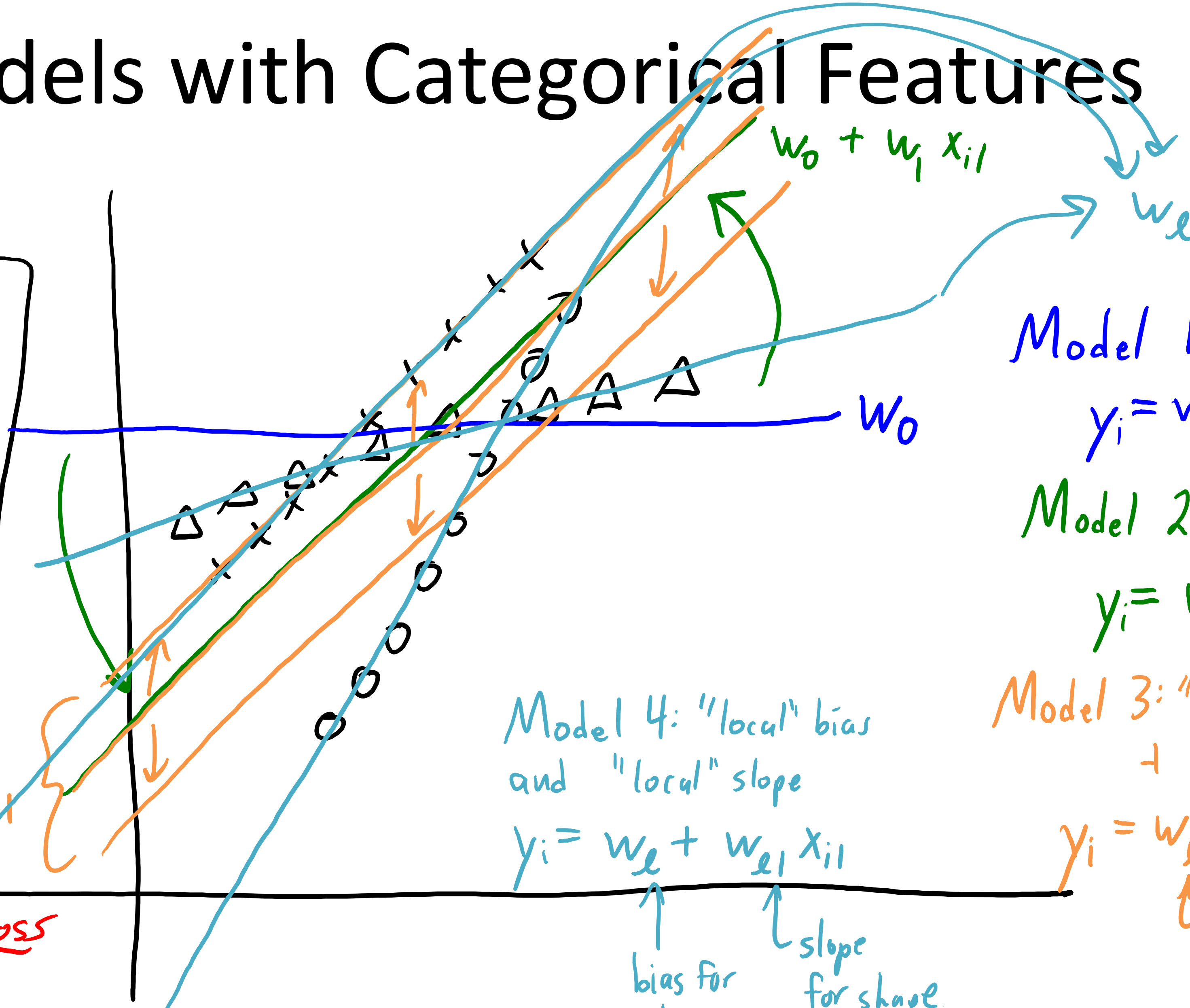
bias for shape

slope for shape

Linear Models with Categorical Features

$X =$

Feature 1	Feature 2
0.5	X
3	O
5	O
2.5	Δ
1.5	X
3	Δ
…	…



Model 1: only bias

$y_i = w_0$

Model 2: bias + feature 1

$y_i = w_0 + w_1 x_{i1}$

Model 3: "local" bias + feature 1

$y_i = w_\ell + w_1 x_{i1}$

shape

Model 4: "local" bias and "local" slope

$y_i = w_\ell + w_{\ell 1} x_{i1}$

bias for shape

slope for shape

Could also share information across categories with global bias slope:

$y_i = w_0 + w_1 x_{i1} + w_\ell + w_{\ell 1} x_{i\ell}$

Sharing information with global parameters

- But with 'local' model for each gender we don't share information.
- To share information across genders, include a 'global' version.

Year	Gender
1975	1
1975	0
1980	1
1980	0

$\Longrightarrow$

Year	Year (if gender = 1)	Year (if gender = 0)
1975	1975	0
1975	0	1975
1980	1980	0
1980	0	1980

- 'Global' year feature: influence of time on both genders.
  - E.g., improvements in technique.
- 'Local' year feature: gender-specific deviation from global trend.
  - E.g., different effects of performance-enhancing drugs.

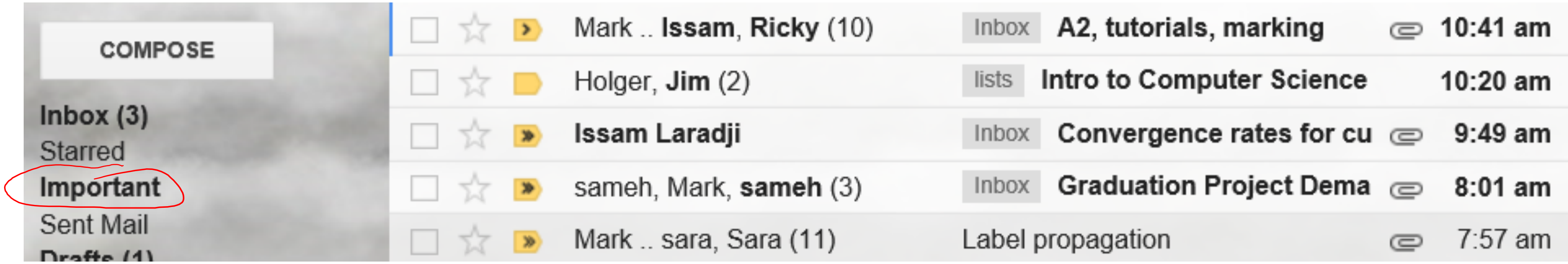
"global" across genders

"local" to gender

$y_i = w_0 + w_1 * year + w_3 * year$

Motivation: Identifying Important E-mails

- How can we automatically identify ‘important’ e-mails?



- We have a big collection of e-mails:
  – Mark as ‘important’ if user takes some action based on them.
- There might be some “universally” important messages:
  – “This is your mother, something terrible happened, give me a call ASAP.”
- But your “important” message may be unimportant to others.
  – Similar for spam: “spam” for one user could be “not spam” for another.

The Big Global/Local Feature Table

X =

y = "important"
"not important"
⋮

"global" features: shared by all users

"local" features for user "1": set to 0 for all other users.

"local" features for user "2"

Predicting Importance of E-mail For New User

- Consider a new user:
  - Start out with no information about them.
  - Use global features to predict what is important to generic user.

$$y_i = \text{sign}(\underbrace{w_g^T x_g}_{\text{features/weights shared across users.}})$$

- With more data, update global features and user's local features:
  - Local features make prediction *personalized*.

$$y_i = \text{sign}(w_g^T x_g + \underbrace{w_u^T x_u}_{\text{features/weights specific to user.}})$$

  - What is important to *this* user?
- Gmail's system: classification with logistic regression.

Amazon Product Recommendation

- Consider this user-item matrix:

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

- Using Euclidean distance:
  - 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):

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

- 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.

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.

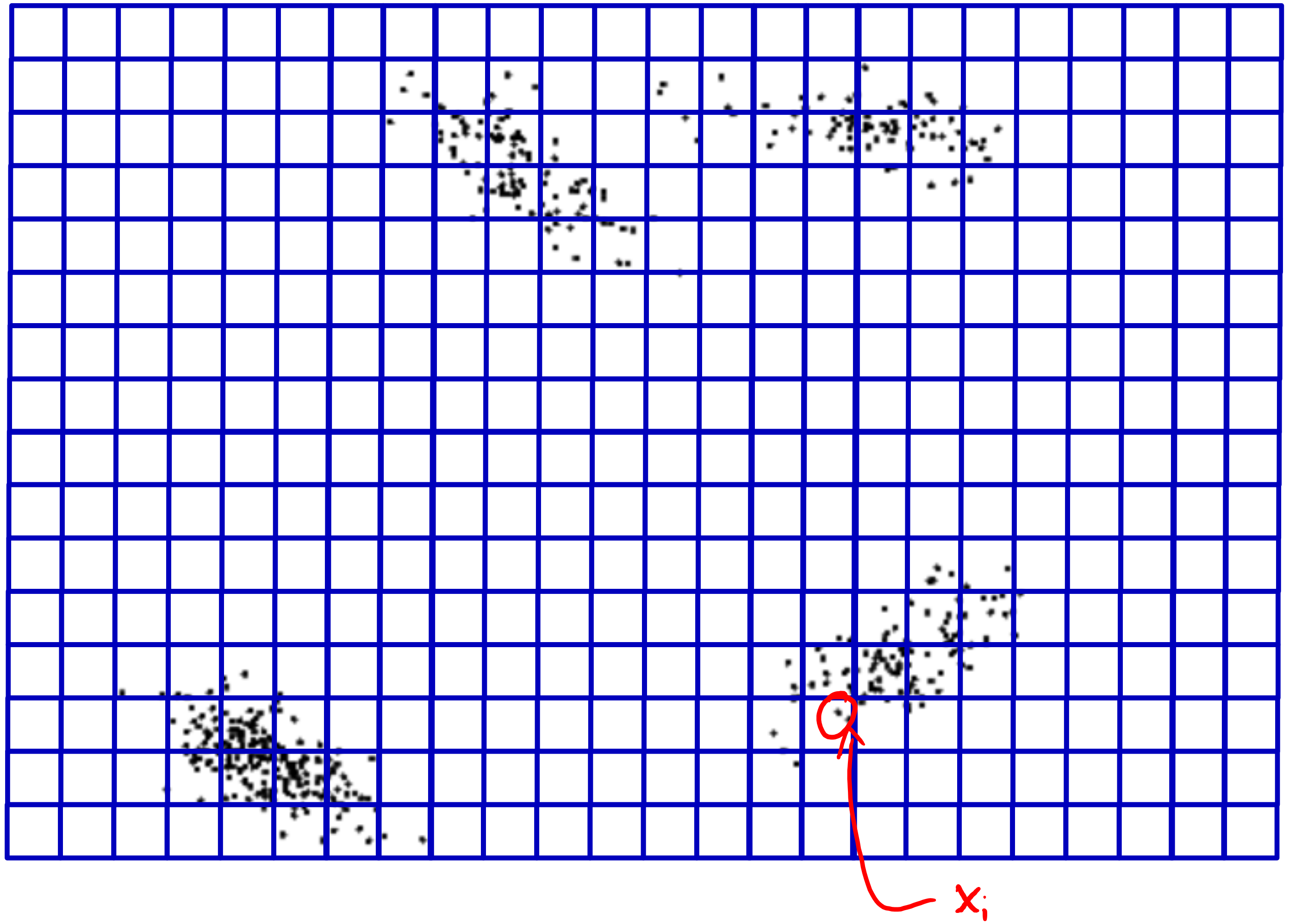
- The answers are called an inverted index, queries now cost O(k).
  - Needs an extra O(nk) storage.

Grid-Based Pruning

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

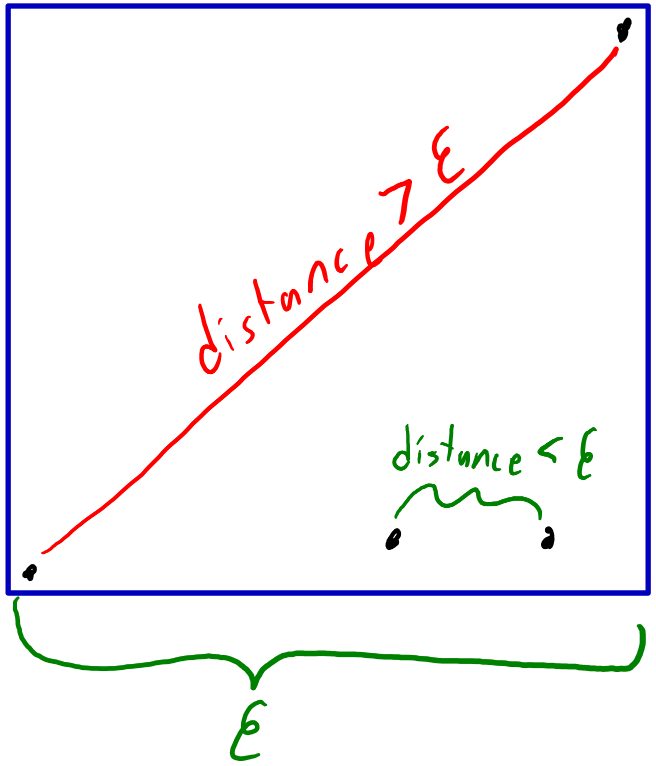
Divide space into squares of length ε.

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?

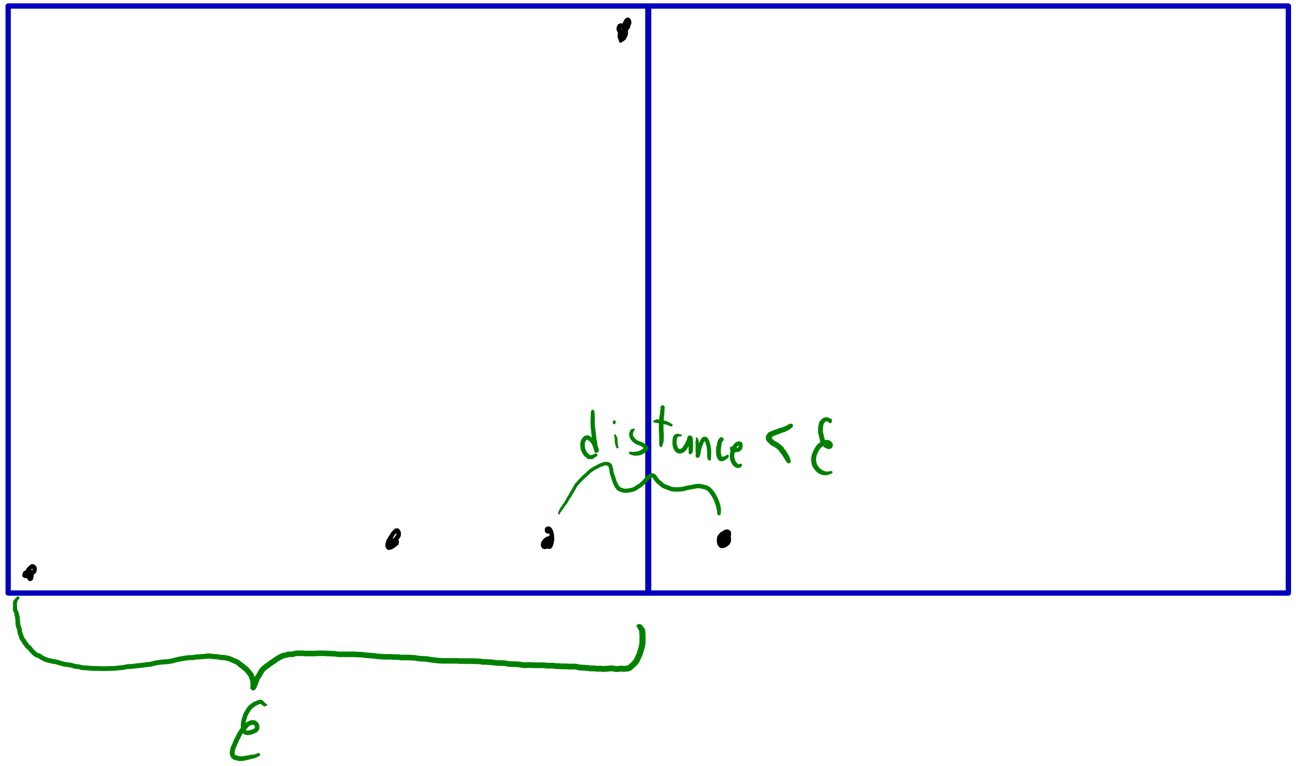


Points in same square can have distance less than ‘ε’.

Grid-Based Pruning

- Which squares do we need to check?

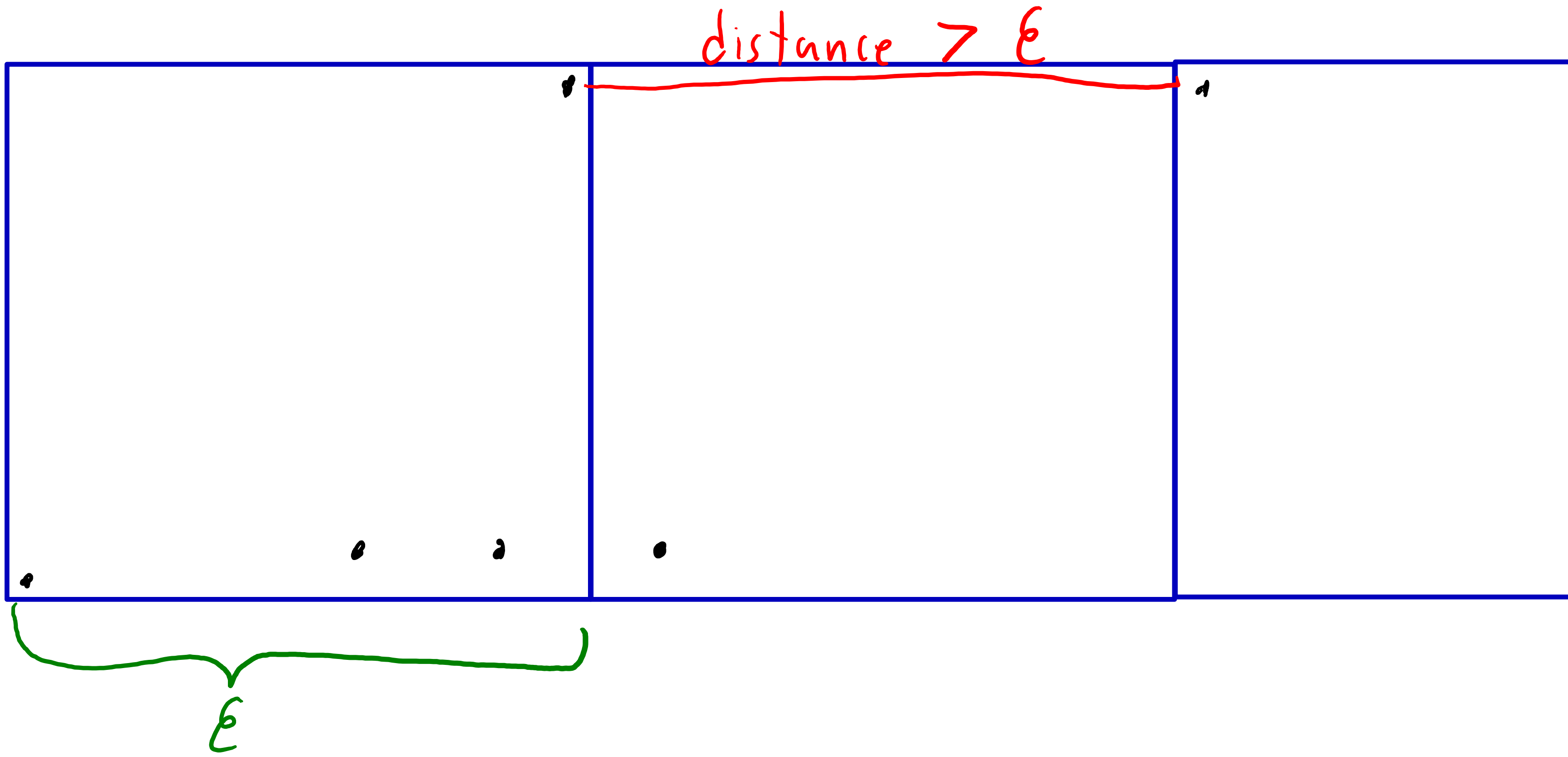
Points in adjacent squares can have distance less than distance ‘ε’.



Grid-Based Pruning

- Which squares do we need to check?

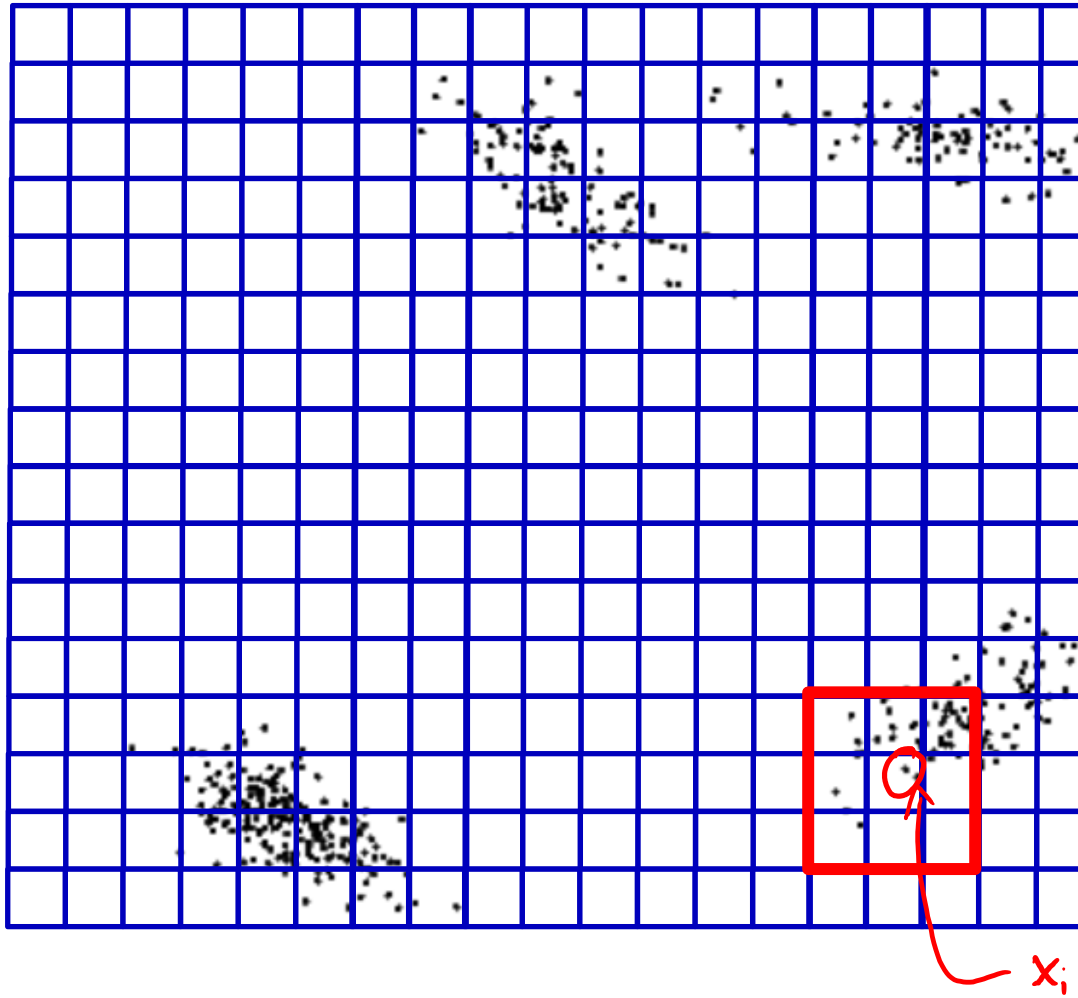
Points in non-adjacent squares must have distance more than ‘ε’.



Grid-Based Pruning

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

Divide space into squares of length ε.



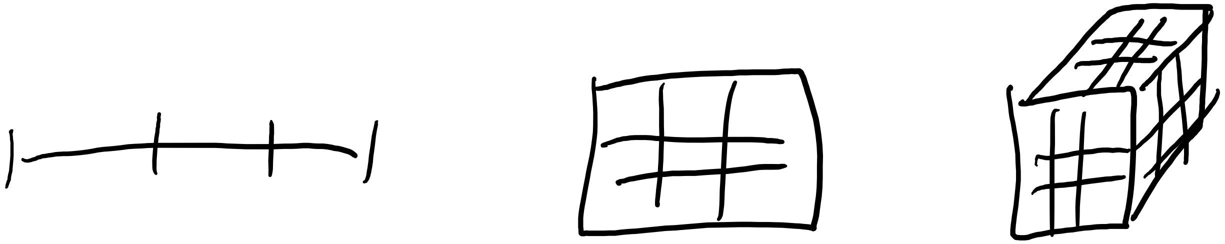
Hash examples based on squares:
Hash[“64,76”] = $\{x_3, x_{70}\}$
(Dict in Python/Julia)

Only need to check points in same and adjacent squares.

Grid-Based Pruning Discussion

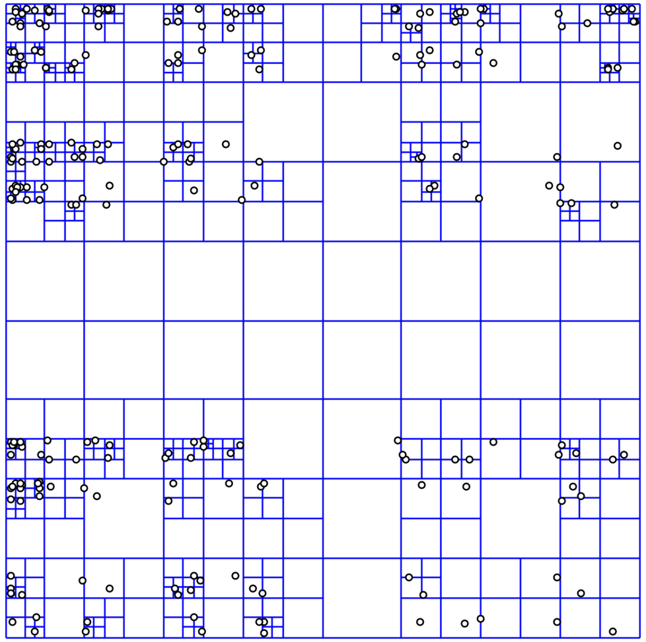
- Similar ideas can be used for other "closest point" calculations.
  - Can be used with any norm.
  - If you want KNN, can use need grids of multiple sizes.

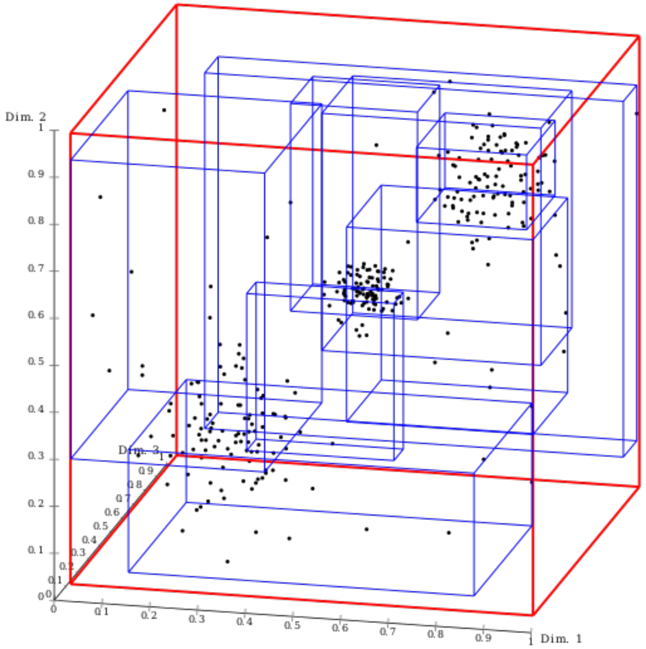
- But we have the "curse of dimensionality":
  - Number of adjacent regions increases exponentially:
    - 2 with d=1, 8 with d=2, 26 with d=3, 80 with d=4, 252 with d=5, 3^d-1 in d-dimension.

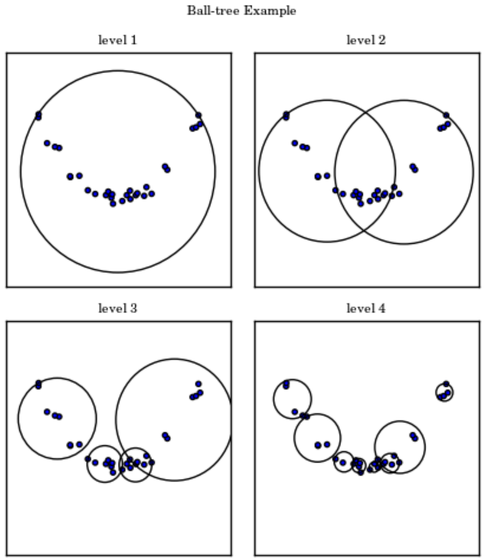


Grid-Based Pruning Discussion

- 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

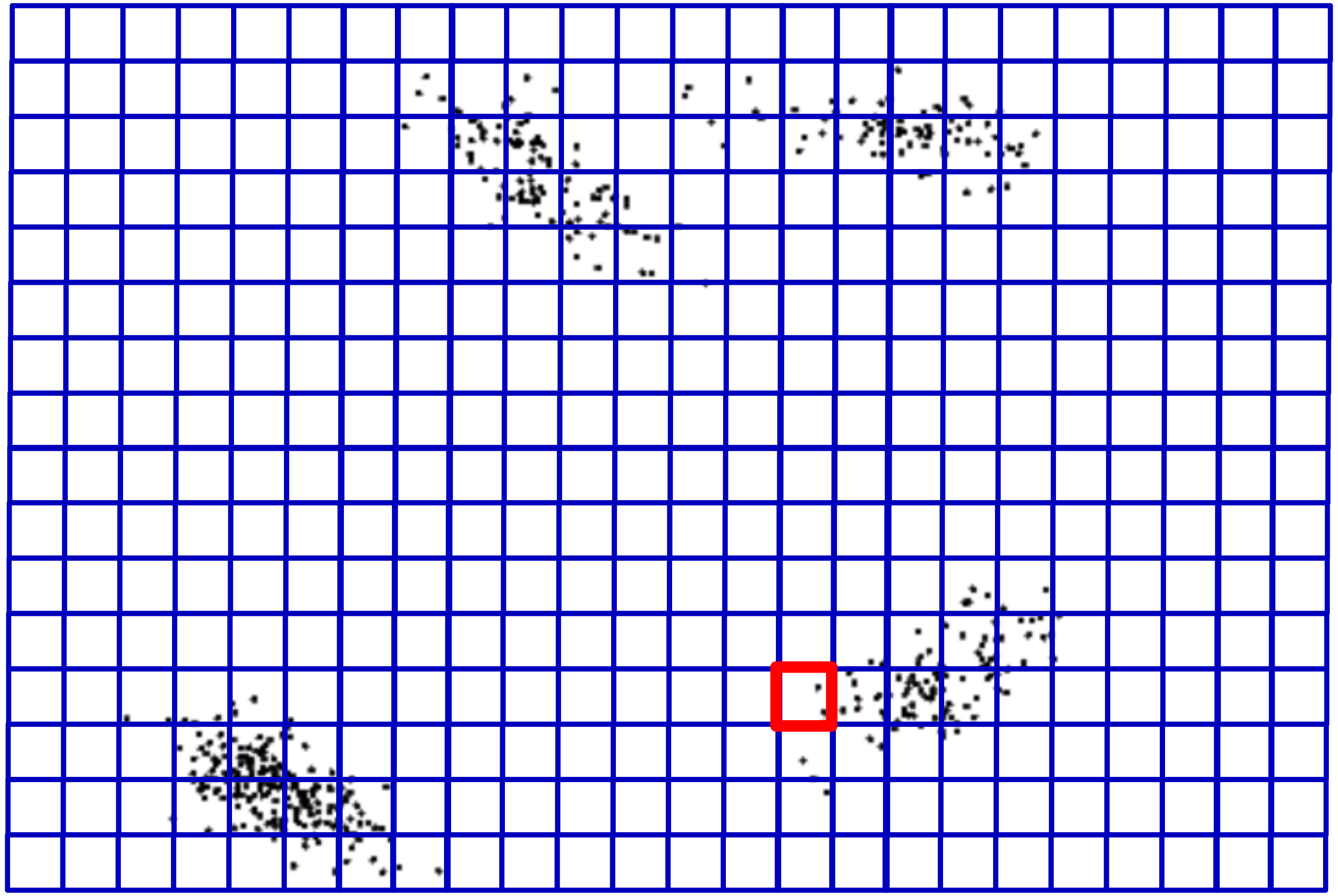
- *Approximate* nearest neighbours:
  - We allow errors in the nearest neighbour calculation to gain speed.

- 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

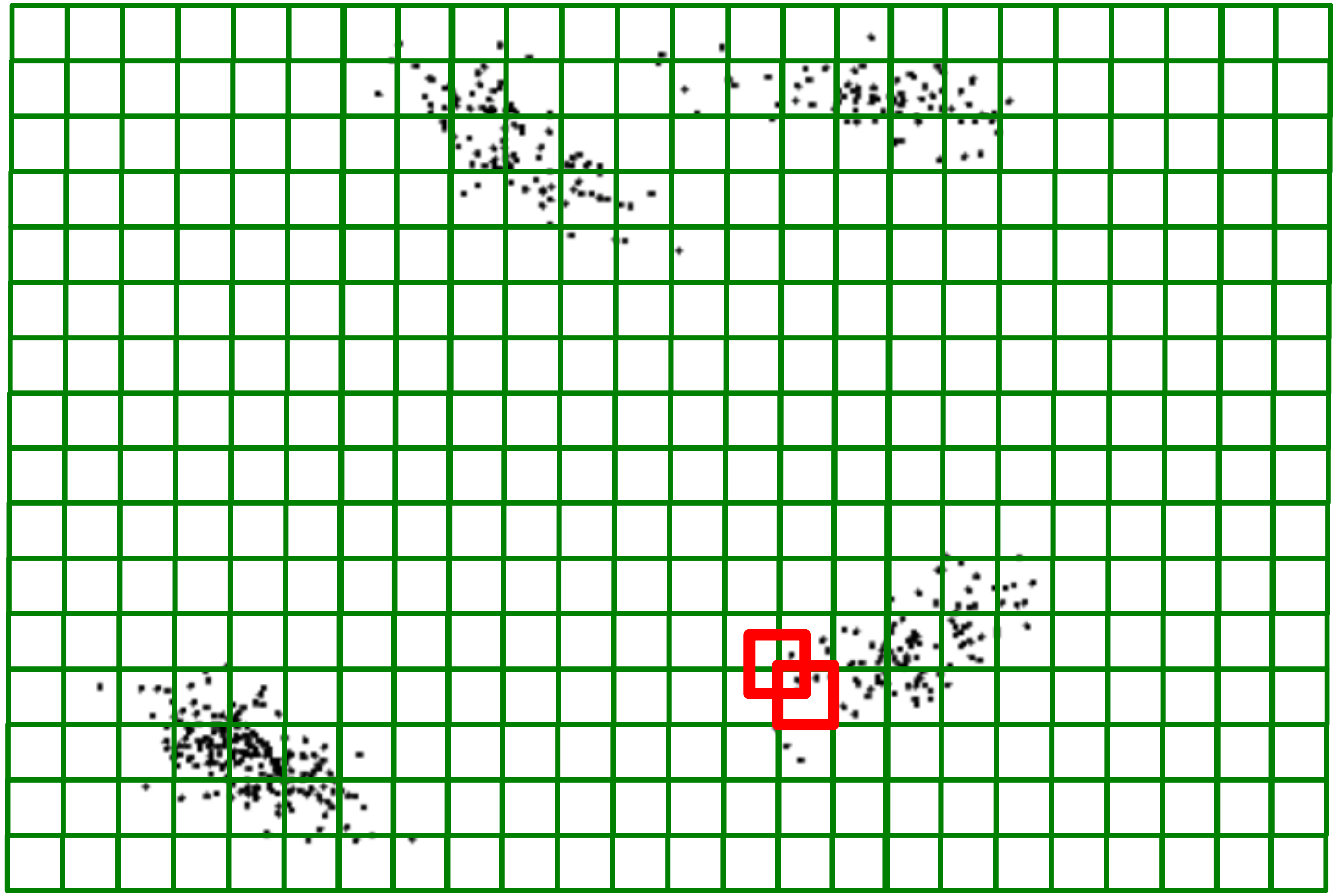
Grid 1:



Approximate Nearest Neighbours

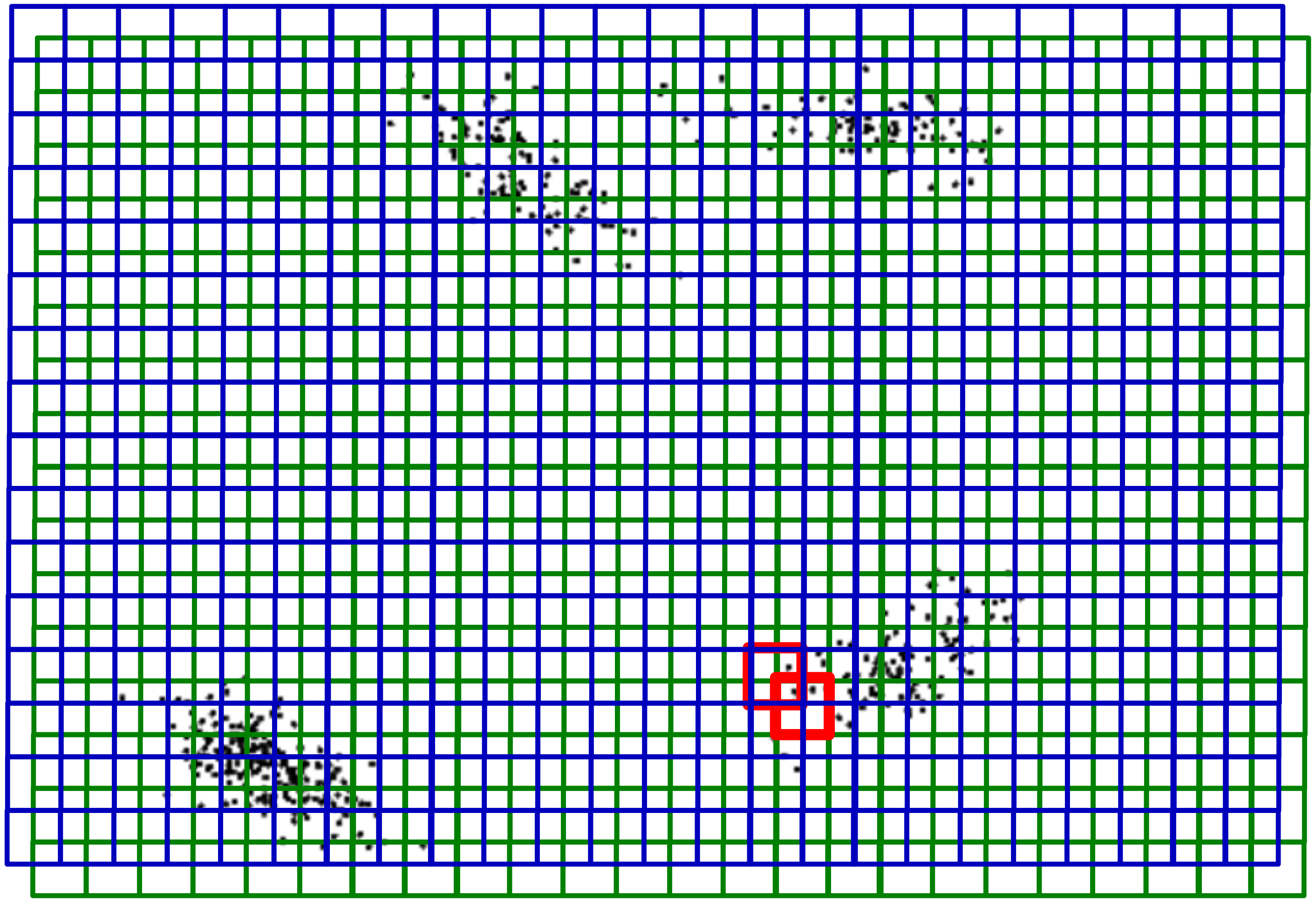
- Using multiple sets of regions improves accuracy.

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.

Warm-Starting

- We've used data {X,y} to fit a model.
- We now have new training data and want to update model.

- Do we need to re-fit from scratch?

- This is the warm starting problem.
  - It's easier to warm start some models than others.

Easy Case: K-Nearest Neighbours and Counting

- K-nearest neighbours:
  - KNN just stores the training data, so just store the new data.

- Counting-based models:
  - Models that base predictions on frequencies of events.
  - E.g., naïve Bayes.

  - Just update the counts:

$$p(\text{"vicodin"} \mid \text{"spam"}) = \frac{\text{count of \{vicodin, spam\} in }\underline{\text{new and old data}}}{\text{count of "spam" in }\underline{\text{new and old data}}}$$

Medium Case: L2-Regularized Least Squares

- L2-regularized least squares is obtained from linear algebra:

$$w = (X^TX + \lambda I)^{-1}(X^Ty)$$

  – Cost is $O(nd^2 + d^3)$.

- Given one new point, we need to compute:
  – X^Ty with one row added, which costs $O(d)$.
  – Old X^TX plus $x_ix_i^T$, which costs $O(d^2)$.
  – Solution of linear system, which costs $O(d^3)$.
  – So cost of adding 't' data point is $O(td^3)$.

- With "matrix factorization updates", can reduce this to $O(td^2)$.

Medium Case: Logistic Regression

- We fit logistic regression by gradient descent on a convex function.
- With new data, convex function f(w) changes to new function g(w).
- If we don't have much more data, 'f' and 'g' will be "close".
  - Start gradient descent on 'g' with minimizer of 'f'.
  - You can show that it requires fewer iterations.

Hard Cases: Non-Convex/Greedy Models

- For decision trees we could also "restart" the algorithm:
  - With new data, consider splitting nodes that we didn't split before.
- However, this won't in general give same result as re-fitting.

- Similar heuristics/conclusions for other non-convex/greedy models:
  - K-means clustering.
  - Collaborative filtering.

- On the other hand, you can add new examples and features and continue PCA algorithms ("non-convex but harmless").

Tensor Factorization

- Tensors are higher-order generalizations of matrices:

Scalar $\alpha = \underbrace{[\;]}_{1\times 1}$ Vector $a = \underbrace{\begin{bmatrix} \\ \\ \end{bmatrix}}_{d\times 1}$ Matrix $A = \underbrace{\begin{bmatrix} & & \\ & & \\ & & \end{bmatrix}}_{d\times d}$ Tensor $A =$

$d \times d \times d$

- Generalization of matrix factorization is tensor factorization:

$$y_{ijm} \approx \sum_{c=1}^{k} w_{jc} z_{ic} v_{mc}$$

- Useful if there are other relevant variables:
  - Instead of ratings based on {user,movie}, ratings based {user,movie,age}.
  - Useful if ratings change over time.

Motivation for Topic Models

- Want a model of the "factors" making up documents.
  - Instead of latent-factor models, they're called topic models.
  - The canonical topic model is latent Dirichlet allocation (LDA).

Suppose you have the following set of sentences:

- I like to eat broccoli and bananas.
- I ate a banana and spinach smoothie for breakfast.
- Chinchillas and kittens are cute.
- My sister adopted a kitten yesterday.
- Look at this cute hamster munching on a piece of broccoli.

What is latent Dirichlet allocation? It's a way of automatically discovering **topics** that these sentences contain. For example, given these sentences and asked for 2 topics, LDA might produce something like

- **Sentences 1 and 2**: 100% Topic A
- **Sentences 3 and 4**: 100% Topic B
- **Sentence 5**: 60% Topic A, 40% Topic B
- **Topic A**: 30% broccoli, 15% bananas, 10% breakfast, 10% munching, … (at which point, you could interpret topic A to be about food)
- **Topic B**: 20% chinchillas, 20% kittens, 20% cute, 15% hamster, … (at which point, you could interpret topic B to be about cute animals)

  - "Topics" could be useful for things like searching for relevant documents.

http://blog.echen.me/2011/08/22/introduction-to-latent-dirichlet-allocation/

Term Frequency – Inverse Document Frequency

- In information retrieval, classic word importance measure is TF-IDF.

- First part is the term frequency tf(t,d) of term ‘t’ for document ‘d’.
  - Number of times “word” ‘t’ occurs in document ‘d’, divided by total words.
  - E.g., 7% of words in document ‘d’ are “the” and 2% of the words are “Lebron”.

- Second part is document frequency df(t,D).
  - Compute number of documents that have ‘t’ at least once.
  - E.g., 100% of documents contain “the” and 0.01% have “LeBron”.

- TF-IDF is tf(t,d)*log(1/df(t,D)).

Term Frequency – Inverse Document Frequency

- The TF-IDF statistic is tf(t,d)*log(1/df(t,D)).
  - It's high if word 't' happens often in document 'd', but isn't common.
  - E.g., seeing "LeBron" a lot it tells you something about "topic" of article.
  - E.g., seeing "the" a lot tells you nothing.

- There are *many* variations on this statistic.
  - E.g., avoiding dividing by zero and all types of "frequencies".

- Summarizing 'n' documents into a matrix X:
  - Each row corresponds to a document.
  - Each column gives the TF-IDF value of a particular word in the document.

Latent Semantic Indexing

- TF-IDF features are very redundant.
  - Consider TF-IDFs of “LeBron”, “Durant”, “Harden”, and “Kobe”.
  - High values of these typically just indicate topic of “basketball”.

- We can probably compress this information quite a bit.

- Latent Semantic Indexing/Analysis:
  - Run latent-factor model (like PCA or NMF) on TF-IDF matrix X.
  - Treat the principal components as the “topics”.
  - Latent Dirichlet allocation is a variant that avoids weird df(t,D) heuristic.

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'.

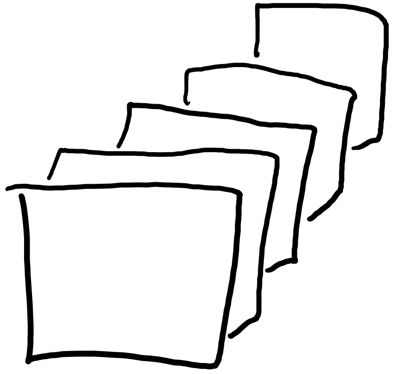
- Association rule learning problem:
  - Given support 's' and confidence 'c'.
  - Output all rules with support at least 's' and confidence at least 'c'.

- 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.

Bonus Slide: Tensor Factorization

- Tensors are higher-order generalizations of matrices:

Scalar $\alpha = [\ ]_{1\times 1}$ Vector $a = [\ ]_{d\times 1}$ Matrix $A = [\ ]_{d\times d}$ Tensor $A =$



$d \times d \times d$

- Generalization of matrix factorization is tensor factorization:

$$y_{ijm} \approx \sum_{c=1}^{k} w_{jc} z_{ic} v_{mc}$$

- Useful if there are other relevant variables:
    - Instead of ratings based on {user,movie}, ratings based {user,movie,age}.
    - Useful if ratings change over time.

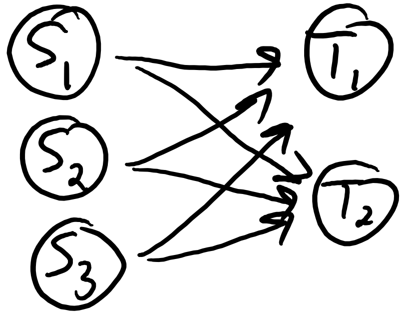
Bonus Slide: Sequential Pattern Analysis

- 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.

Association Rules

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

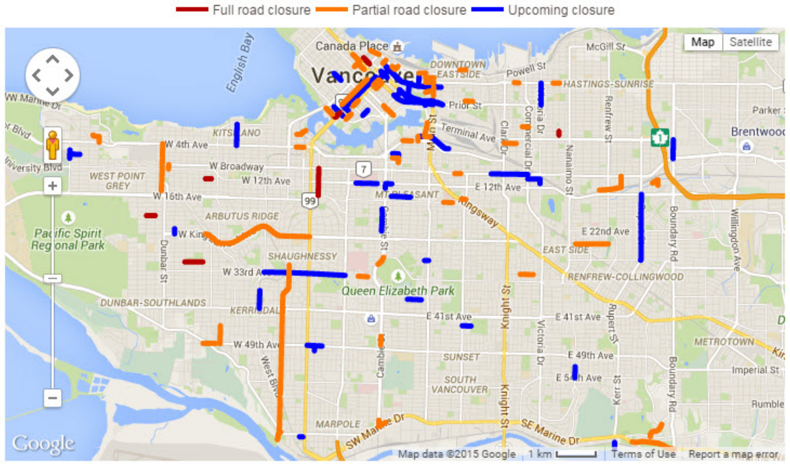
    I'm using p(T = 1 | S = 1) for $p(T_1 = 1, T_2 = 1, \ldots, T_k = 1 \mid S_1 = 1, S_2 = 1, \ldots, S_c = 1)$.
- Association rules are directed but not necessarily causal:

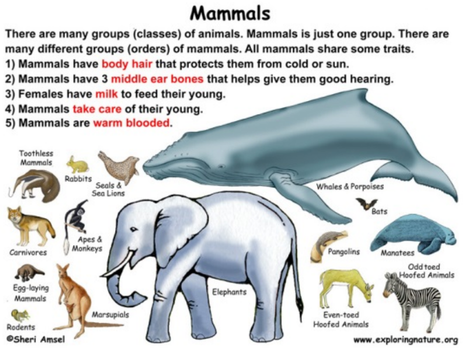


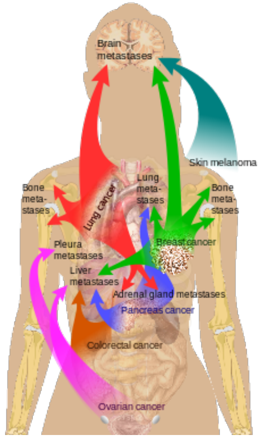
  - $p(T \mid S) \neq p(S \mid T)$.
    - E.g., buying sunscreen doesn't necessarily imply buying sunglasses/sandals:
  - The correlation could be backwards or due to a common cause.
    - E.g., the common cause is that you are going to the beach.

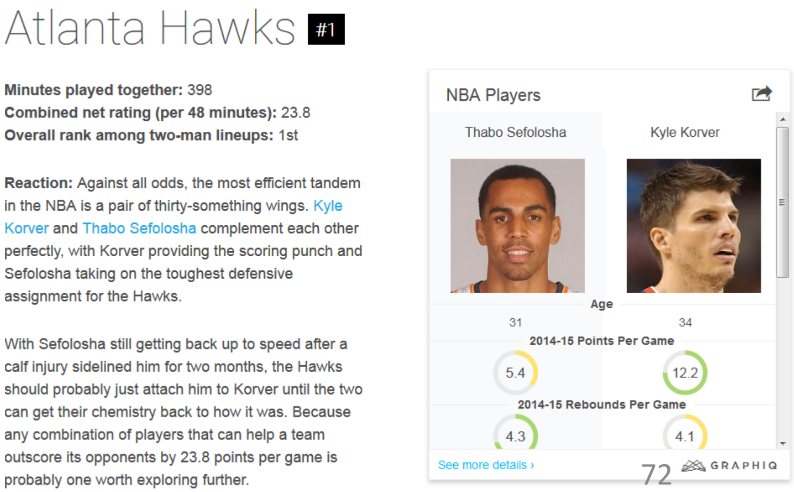
Applications of Association Rules

- 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?









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

Woes with notation/definitions

- In some books/sources, support is defined as on the previous slide
  - For example Wikipedia or *Mining of Massive Datasets*
- In other sources, support is defined as
  - How often does $S \cap T$ (S and T) happen?
    - How often were sunglasses, sandals and sunscreen bought together?
  - Joint probability: $p(S = 1, T=1)$.
  - For example in *Database Management Systems* by Ramakrishnan & Gehrke
- Furthermore, in some texts, for a rule S=>T, T must be a single item. In other cases it can itself be a set.

Finding Sets with High Support

- First let's focus on finding sets 'S' with high support ("frequent itemsets")
- 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
⋮

$$p(S=1) = \frac{\#\text{ times } \underline{\text{all}} \text{ elements of 'S' are '1'}}{n}$$

Challenge in Learning Association Rule

- Consider the problem of finding all sets 'S' with $p(S = 1) \geq s$.
  - 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…

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 $p(S = 1)$.

Because probabilities are non-negative $p(S=1, A=1) \leq p(S=1, A=1) + p(S=1, A=0)$

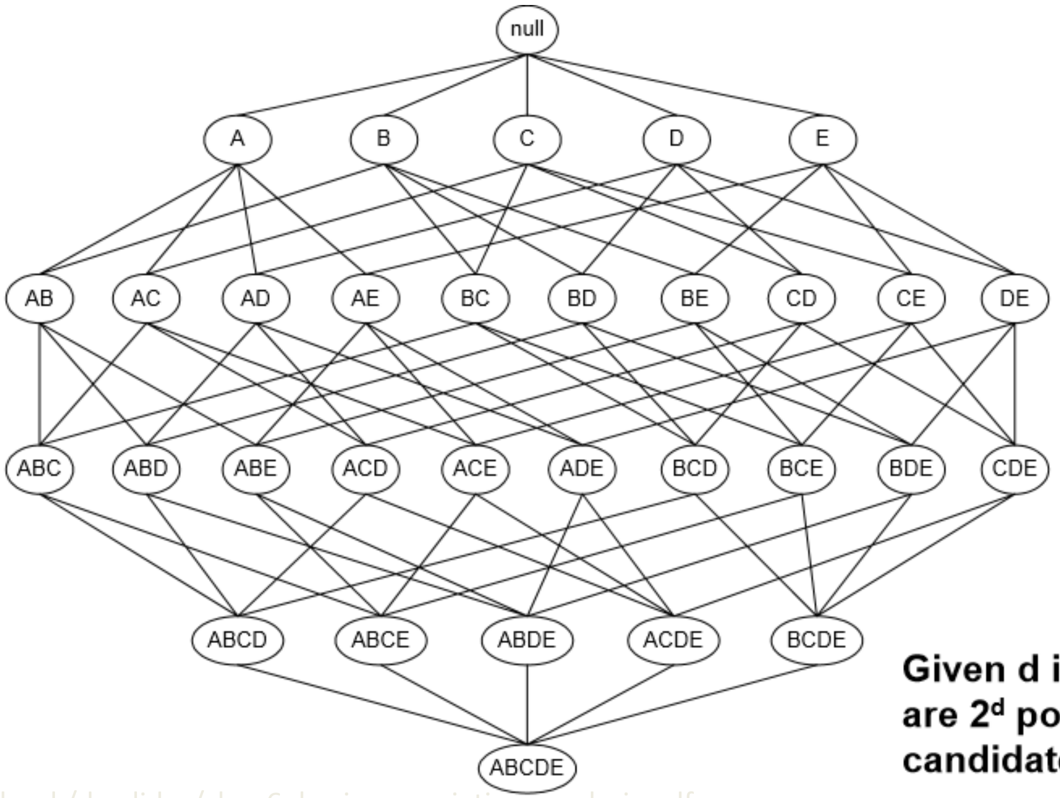
By the marginalization rule $p(S=1) = p(S=1, A=1) + p(S=1, A=0)$

Putting these together gives $p(S=1, A=1) \leq p(S=1)$

- E.g., probability of rolling {4,5} on 2 dice (1/36) is less than rolling 4 on one die (1/6).

Support Set Pruning

- This property means that $p(S_1 = 1) < s$ implies $p(S_1 = 1, S_2 = 1) < s$.
  - If p(sunglasses=1) < 0.1, then p(sunglasses=1,sandals=1) is less than 0.1.
  - We never consider $p(S_1 = 1, S_2 = 1)$ if $p(S_1 = 1)$ has low support.

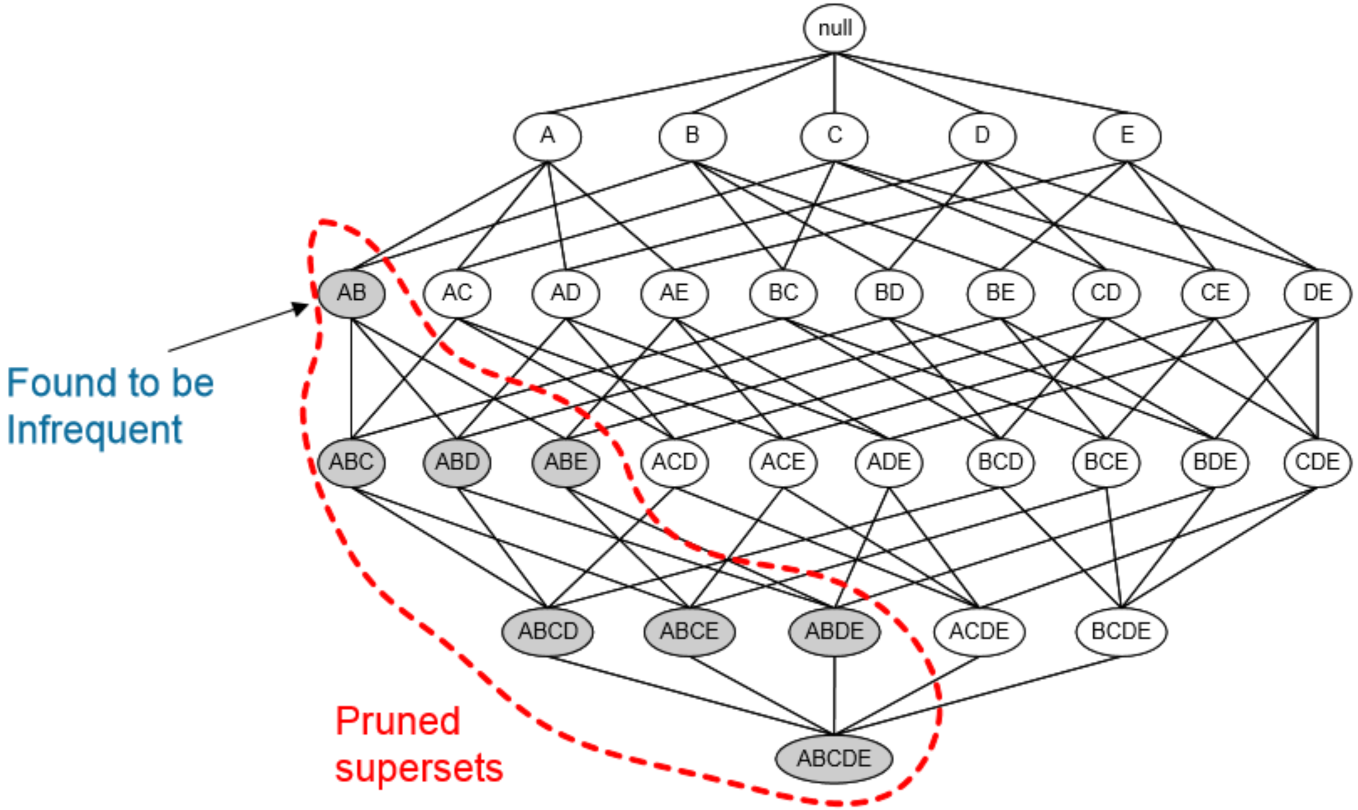


Given d items, there are 2^d possible candidate itemsets

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

Support Set Pruning

- This property means that $p(S_1 = 1) < s$ implies $p(S_1 = 1, S_2 = 1) < s$.
  - If p(sunglasses=1) < 0.1, then p(sunglasses=1,sandals=1) is less than 0.1.
  - We never consider $p(S_1 = 1, S_2 = 1)$ if $p(S_1 = 1)$ has low support.



p(E) must be greater than

p(D, E) which must be greater than

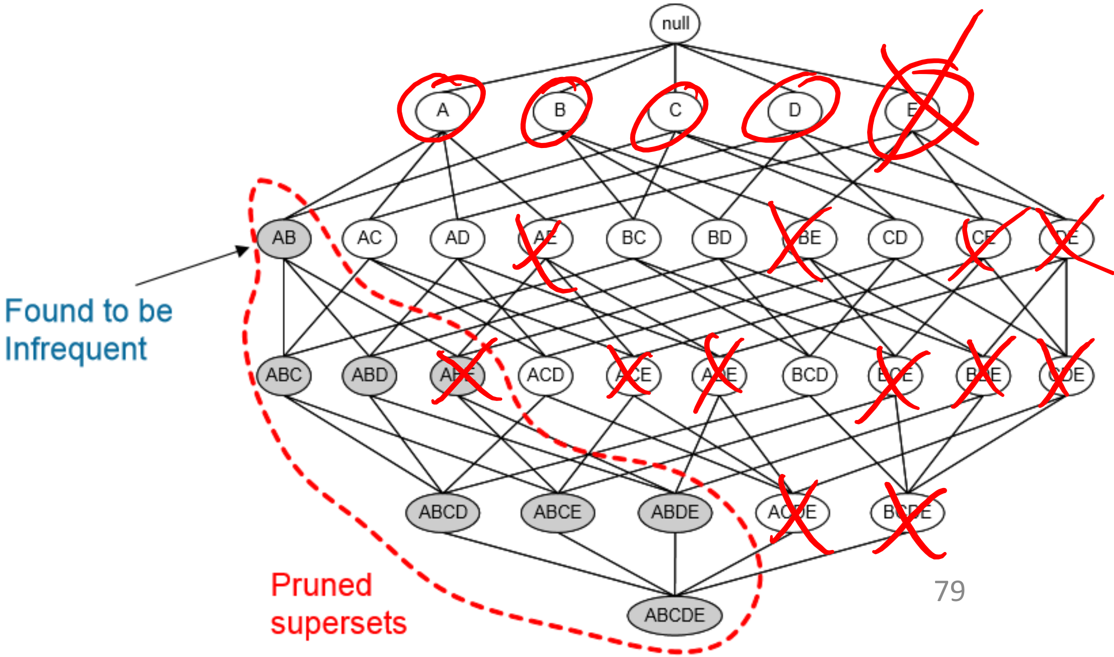
p(C, D, E) which must be greater than

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:

We don't check rules with coke or eggs.

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

{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.

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.)

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:

We don't check rules with coke or eggs.

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

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

A Priori Algorithm Discussion

- Some implementations only return ‘Maximal frequent subsets’:
  - Only return sets S with p(S = 1) ≥ s where no superset S’ has p(S’ = 1) ≥ 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 counts 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.
  - Hash trees can be used to speed up various computations.

Generating Rules

- A priori algorithm gives all 'S' with p(S = 1) ≥ s.
- To generate the rules, we consider subsets of frequent itemsets
  - If {1,2,3} is a frequent itemset, candidate rules involving these items 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:

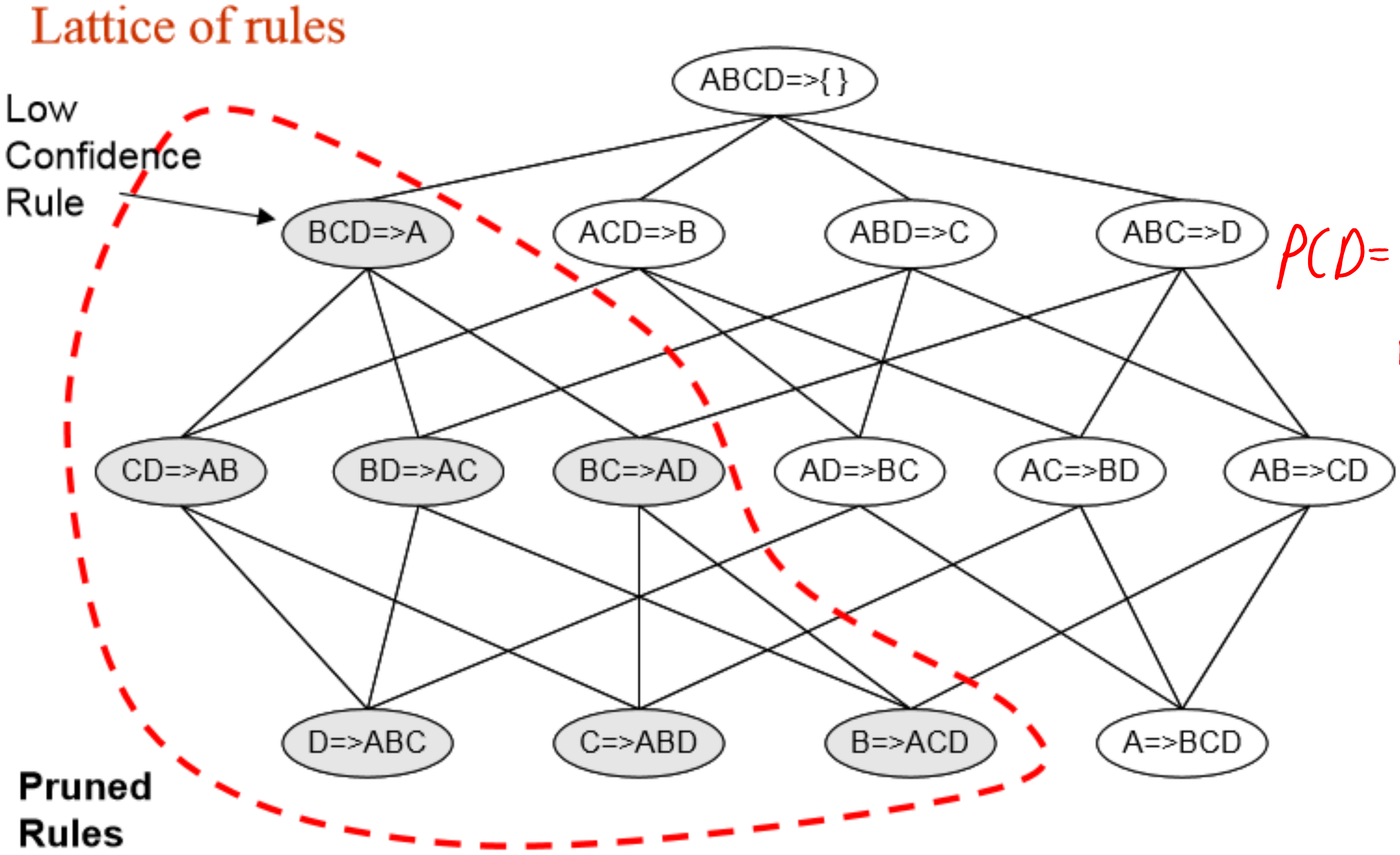
By definition of conditional probability we have $p(T=1 \mid S=1) = \frac{p(S=1, T=1)}{p(S=1)}$

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 die is a 6.

Confident Set Pruning



$P(D=1 | A=1, B=1, C=1)$

must be greater than

$P(C=1, D=1 | A=1, B=1)$

- Or… computation is very fast if T can only be a single item

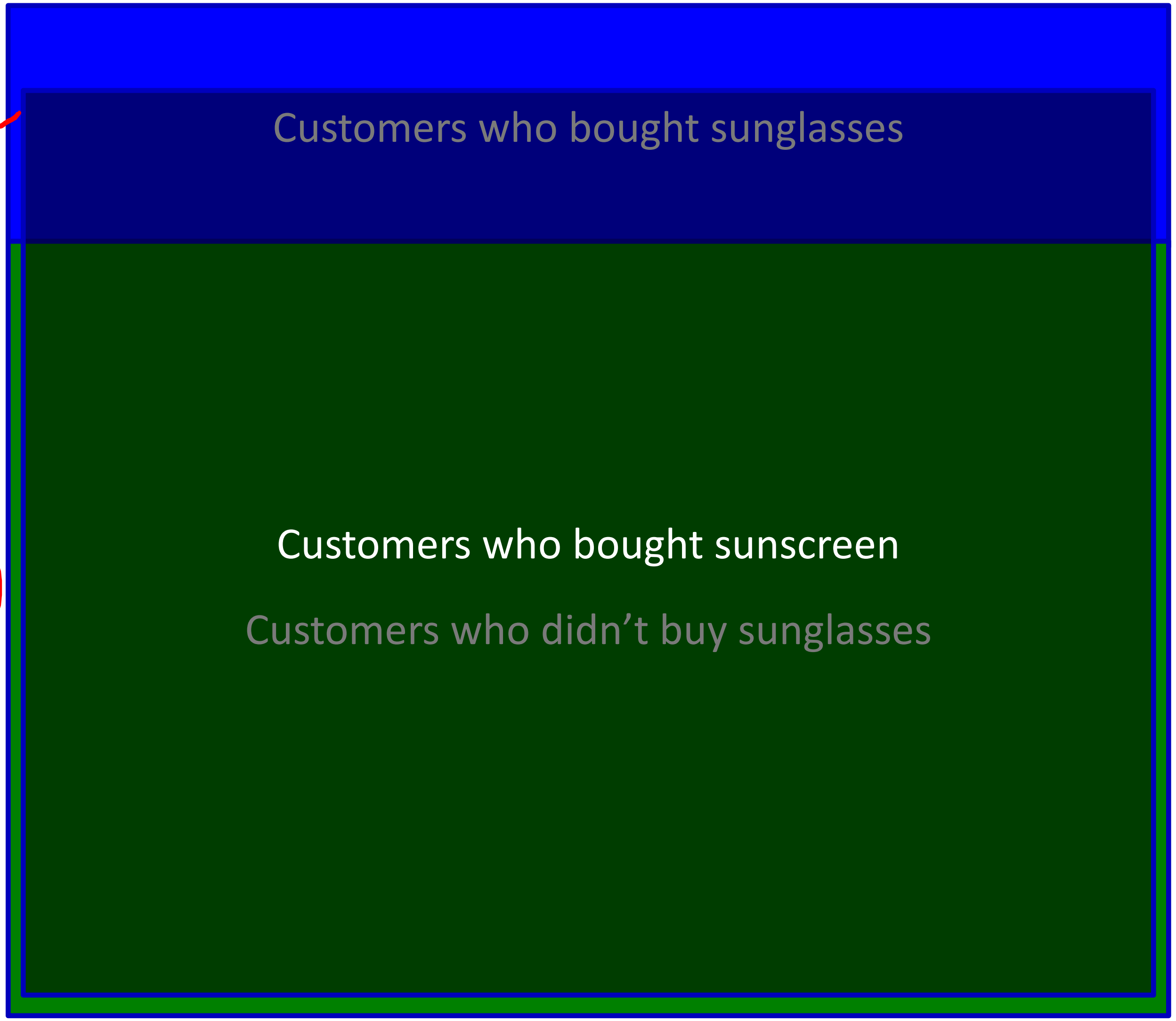
Association Rule Mining Issues

- 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.
  - Amazon’s recommendation system.

Problem with Confidence

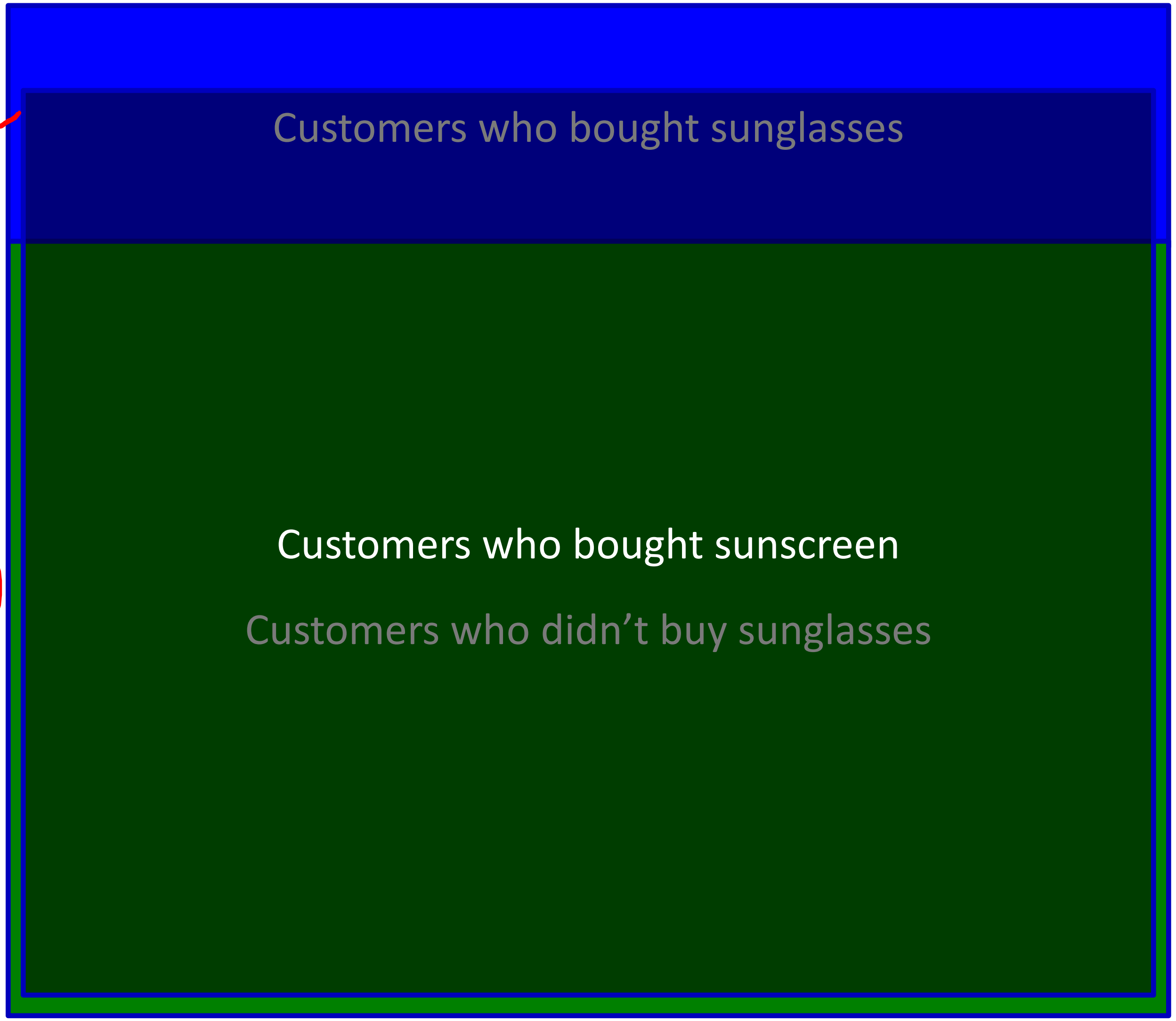
- Consider the “Sunscreen Store”:
  - Most customers go there to buy sunscreen.
- Now consider rule (sunglasses => sunscreen).
  - If you buy sunglasses, it could mean you weren’t there for sunscreen:
    - p(sunscreen = 1| sunglasses = 1) < p(sunscreen = 1).
  - So (sunglasses => sunscreen) could be a misleading rule:
    - You are less likely to buy sunscreen if you buy sunglasses.
  - But the rule could have high confidence.
- Example:
  - p(sunscreen) = 0.9 (marginal probability)
  - p(sunglasses) = 0.2 (marginal probability)
  - p(sunscreen and sunglasses) = 0.1 (joint probability)
  - This means p(sunscreen | sunglasses) = 0.1/0.2 = 0.5 (conditional probability)


Customers who bought sunglasses
Customers who didn’t buy sunglasses



Most customers buy sunscreen

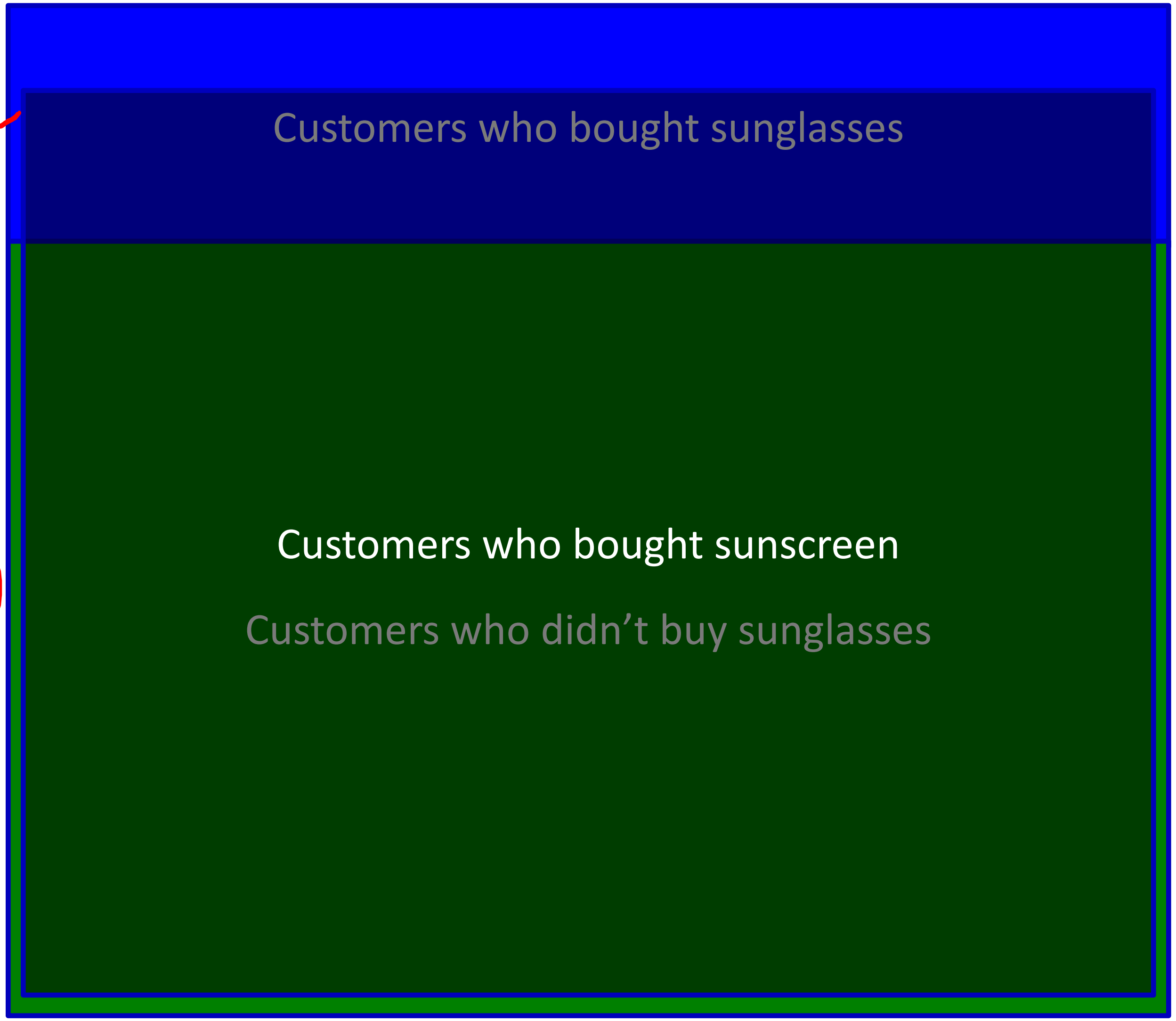
Customers who bought sunglasses are still likely to buy sunscreen.



Most customers buy sunscreen.

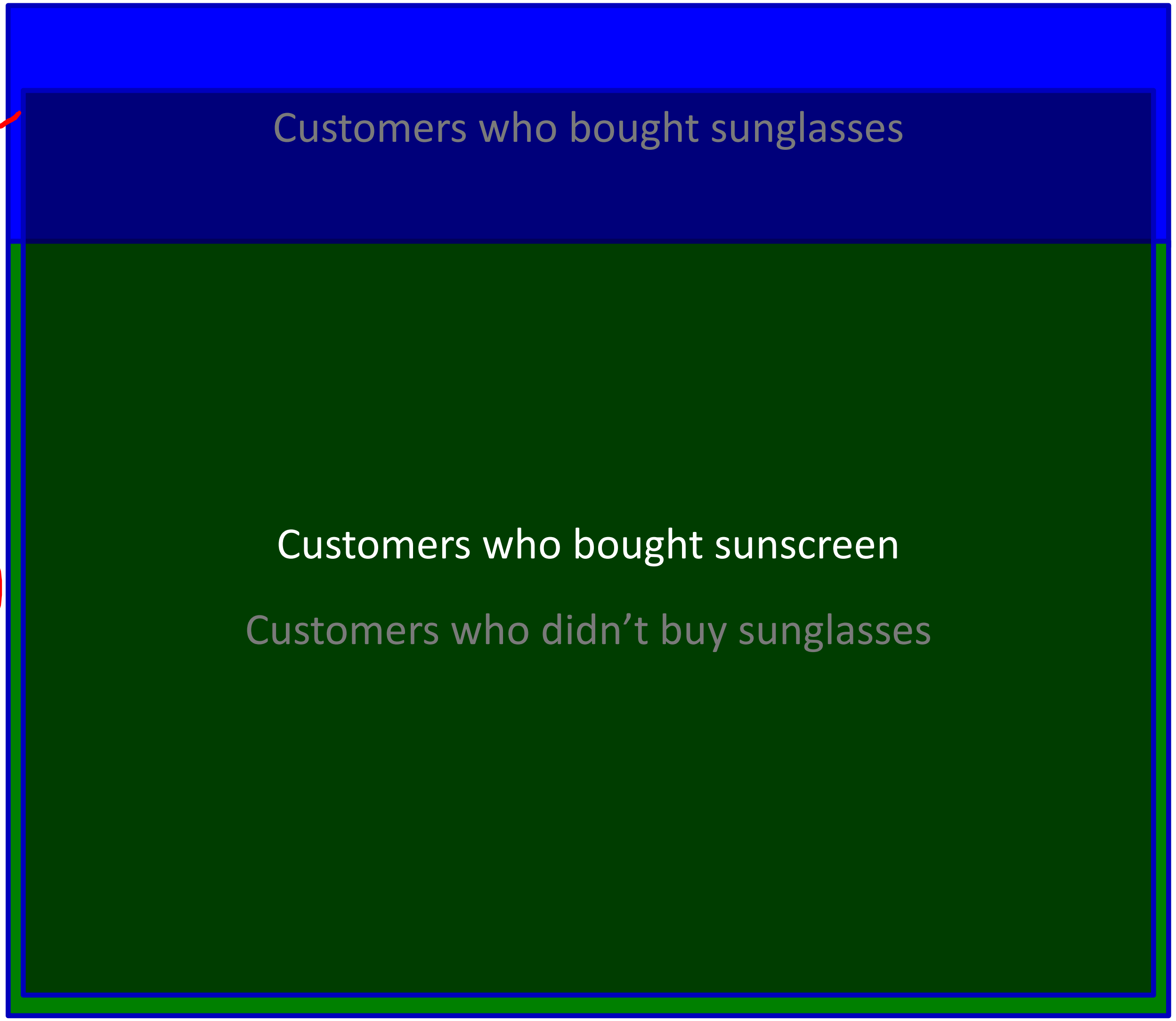
Customers who bought sunglasses are still likely to buy sunscreen.

Most customers buy sunscreen



But knowing that they bought sunglasses make it <u>less</u> 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

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

- One alternative to confidence is “lift”:
  - How much more likely does ‘S’ make us to buy ‘T’?

Confidence

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

Amazon Recommendation Algorithm

- Recommend nearest neighbours with cosine similarity:

$$\cos(x_i, x_j) = \frac{\sum_{k=1}^{d} x_{ik} x_{jk}}{\|x_i\| \, \|x_j\|}$$

  - If $\cos(x_i,x_j) = 1$, products were bought by exact same users.
  - This is just the dot product, but normalized
  - This is pretty similar to Euclidean distance but normalizes for magnitude
    - If two products were bought rarely but by different people, don't consider similar

$$X = \underbrace{\begin{bmatrix} \quad & \quad \\ \quad & \quad \end{bmatrix}}_{\text{user}} \left.\vphantom{\begin{bmatrix} \quad \\ \quad \end{bmatrix}}\right\} \text{product}$$