CS 422 Data Mining

Lecture 4

September 13, 2018

Classification Errors

- Training errors (apparent errors)
  - Errors committed on the training set
- Test errors
  - Errors committed on the test set
- Generalization errors
  - Expected error of a model over random selection of records from same distribution

Notes on Overfitting

- Overfitting results in decision trees that are more complex than necessary

- Training error does not provide a good estimate of how well the tree will perform on previously unseen records

- Need ways for estimating generalization errors

Model Selection

- Performed during model building
- Purpose is to ensure that model is not overly complex (to avoid overfitting)
- Need to estimate generalization error
  - Using Validation Set
  - Incorporating Model Complexity
  - Estimating Statistical Bounds

Model Selection:

Using Validation Set

- Divide training data into two parts:
  - Training set:
    - use for model building
  - Validation set:
    - use for estimating generalization error
    - Note: validation set is not the same as test set
- Drawback:
  - Less data available for training

Model Selection:
Incorporating Model Complexity

- Rationale: Occam's Razor
  - Given two models of similar generalization errors, one should prefer the simpler model over the more complex model
  - A complex model has a greater chance of being fitted accidentally by errors in data
  - Therefore, one should include model complexity when evaluating a model

Gen. Error(Model) = Train. Error(Model, Train. Data) + α x Complexity(Model)

Model Selection for Decision Trees

- Pre-Pruning (Early Stopping Rule)
  - Stop the algorithm before it becomes a fully-grown tree
  - Typical stopping conditions for a node:
    - Stop if all instances belong to the same class
    - Stop if all the attribute values are the same
  - More restrictive conditions:
    - Stop if number of instances is less than some user-specified threshold
    - Stop if class distribution of instances are independent of the available features (e.g., using χ^2 test)
    - Stop if expanding the current node does not improve impurity measures (e.g., Gini or information gain).
    - Stop if estimated generalization error falls below certain threshold

Model Selection for Decision Trees

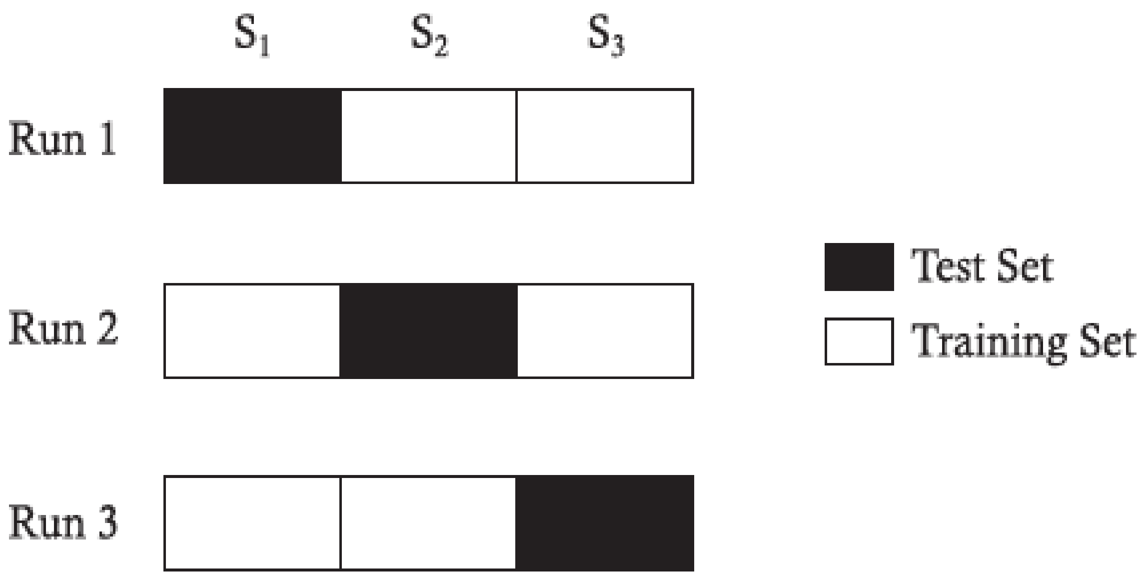
- Post-pruning
  - Grow decision tree to its entirety
  - Subtree replacement
    - Trim the nodes of the decision tree in a bottom-up fashion
    - If generalization error improves after trimming, replace sub-tree by a leaf node
    - Class label of leaf node is determined from majority class of instances in the sub-tree
  - Subtree raising
    - Replace subtree with most frequently used branch

Model Evaluation

- Purpose:
  - To estimate performance of classifier on previously unseen data (test set)
- Holdout
  - Reserve k% for training and (100-k)% for testing
  - Random subsampling: repeated holdout
- Cross validation
  - Partition data into k disjoint subsets
  - k-fold: train on k-1 partitions, test on the remaining one
  - Leave-one-out: $k=n$

Cross-validation Example

- 3-fold cross-validation



History

- Precursors: Expert Based Systems (EBS)

  EBS = Knowledge database + Inference Engine

  - MYCIN: Medical diagnosis system based, 600 rules
  - XCON: System for configuring VAX computers, 2500 rules (1982)
- The rules were created by experts by hand!!
- Knowledge acquisition has to be automatized
  - Substitute the **Expert** by its **archive with solved cases**

Extension to Basic DT

- CHAID (CHi-squared Automatic Interaction Detector) Gordon V. Kass ,1980
- CART (Classification and Regression Trees), Breiman, Friedman, Olsen and Stone, 1984
- ID3 (Iterative Dichotomiser 3), Quinlan, 1986
- C4.5, Quinlan 1993: Based on ID3

General Approach to DT

For decision trees a greedy approach is generally selected:

- Built step by step, instead of building the tree as a whole
- At each step the best split with respect to the train data is selected (following a **split criterion**).
- The tree is grown until a **stopping criterion** is met
- The tree is generally pruned (following a **pruning criterion**) to avoid over-fitting.

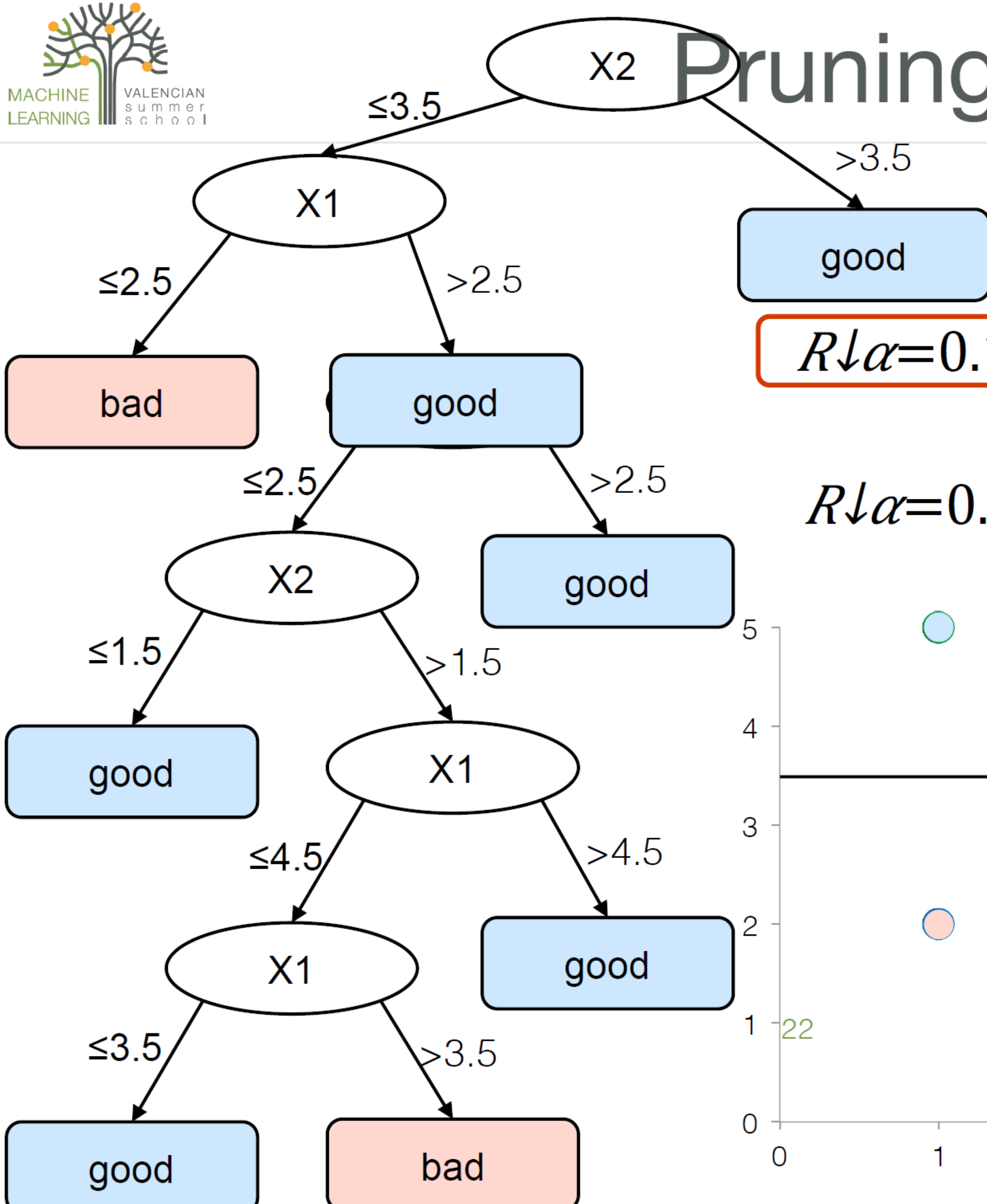
CART

- Cost-complexity based pruning:

$$R_{\alpha}(t) = R(t) + \alpha \cdot C(t)$$

  - R(t) is the error of the decision tree rooted at node t
  - C(t) is the number of leaf nodes from node t
  - Parameter α specifies the relative weight between the accuracy and complexity of the tree

Pruning CART



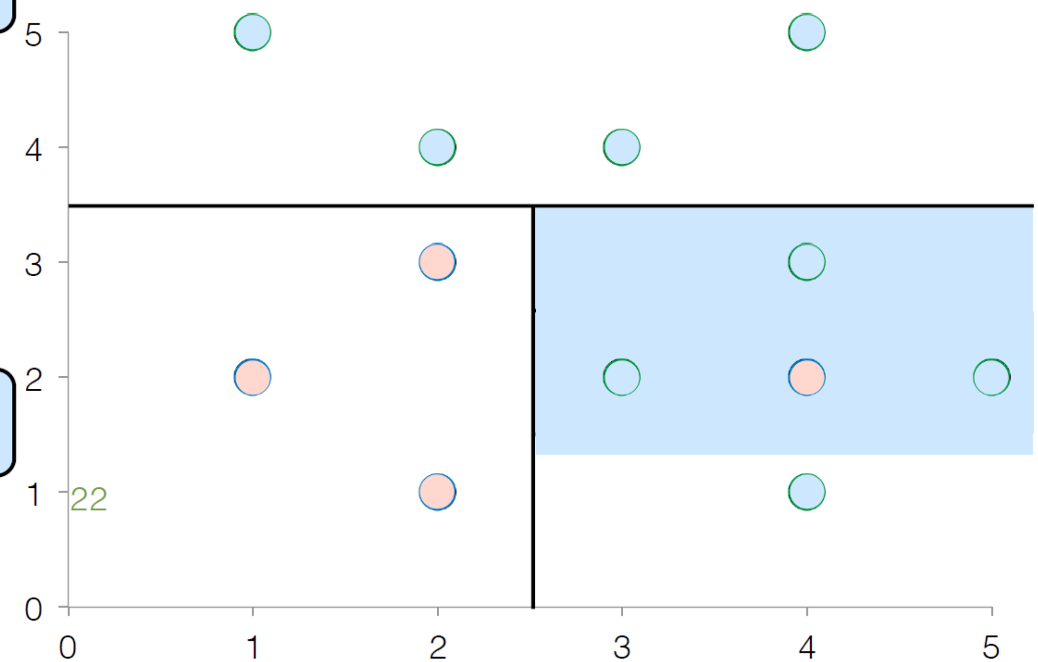
Let's say $\alpha = 0.1$

Pruned:

$$R_{\alpha=0.1}(t) = 1/5 + 0.1 \cdot 1 = 0.3$$

Unpruned

$$R_{\alpha=0.1}(t) = 0 + 0.1 \cdot 5 = 0.5$$



CART

- CART uses 10-fold cross-validation within the training data to estimate alpha. Iteratively nine folds are used for training a tree and one for test.
- A tree is trained on nine folds and it is pruned using all possible alphas (that are finite).
- Then each of those trees is tested on the remaining fold.
- The process is repeated 10 times and the alpha value that gives the best generalization accuracy is kept

Statistics Based Pruning

- C4.5 estimates the accuracy % on the leaf nodes using the upper confidence bound (parameter) of a normal distribution instead of the data.
- Error estimate for subtree is the weighted sum of the error estimates for all its leaves
- This error is higher when few data instances fall on a leaf.
- Hence, leaf nodes with few instances tend to be pruned.

CART vs C4.5

- CART pruning is slower since it has to build 10 extra trees to estimate alpha.
- C4.5 pruning is faster, however the algorithm does not propose a way to compute the confidence threshold
- The statistical grounds for C4.5 pruning are questionable.
- Using cross validation is safer

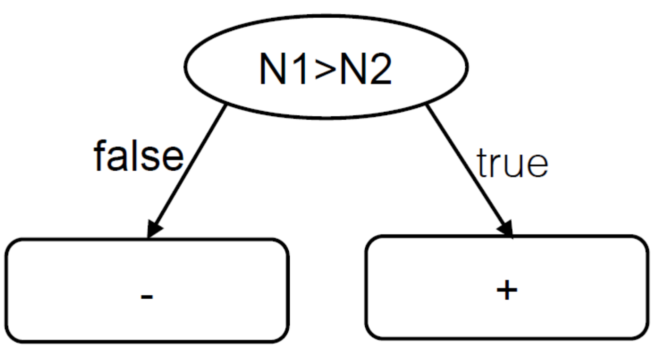
Oblique Splits

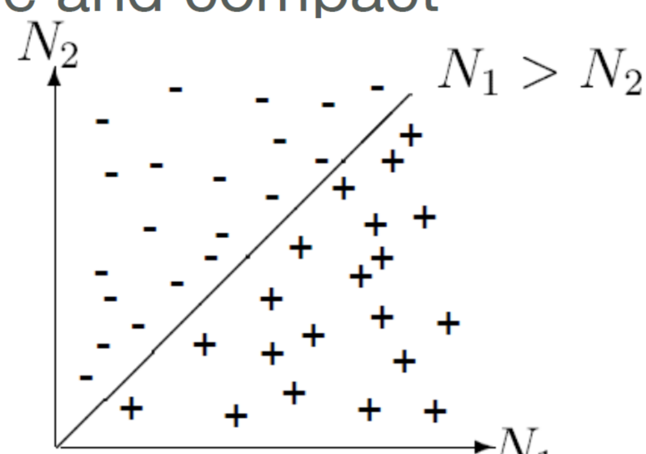
- CART algorithms allows for oblique splits, i.e. splits that are not orthogonal to the attributes axis

- The algorithm searches for planes with good impurity reduction

- The growing tree process becomes slower

- But trees become more expressive and compact





Comparison

	Splitting criterion	Pruning criterion	Other features
CART	• Gini • Twoing	Cross-validation post-pruning	• Regression/Classif. • Nominal/numeric attributes • Missing values • Oblique splits • Nominal splits grouping
ID3	Information Gain (IG)	Pre-pruning.	• Classification • Nominal attributes
C4.5	• Information Gain (IG) • Information Gain Ratio (IGR)	Statistical based post-pruning	• Classification • Nominal/numeric attributes • Missing values • Rule generator • Multiple nodes split

Continuous Attributes: Computing Gini Index

- Use Binary Decisions based on one value
- Several Choices for the splitting value
  - Number of possible splitting values
    = Number of distinct values
- Each splitting value has a count matrix associated with it
  - Class counts in each of the partitions, A < v and A ≥ v
- Simple method to choose best v
  - For each v, scan the database to gather count matrix and compute its Gini index
  - Computationally Inefficient! Repetition of work.

ID	Home Owner	Marital Status	Annual Income	Defaulted
1	Yes	Single	125K	No
2	No	Married	100K	No
3	No	Single	70K	No
4	Yes	Married	120K	No
5	No	Divorced	95K	Yes
6	No	Married	60K	No
7	Yes	Divorced	220K	No
8	No	Single	85K	Yes
9	No	Married	75K	No
10	No	Single	90K	Yes

Annual Income ?

	≤ 80	> 80
Defaulted Yes	0	3
Defaulted No	3	4

Continuous Attributes: Computing Gini Index...

- For efficient computation: for each attribute,
  - Sort the attribute on values
  - Linearly scan these values, each time updating the count matrix and computing gini index
  - Choose the split position that has the least gini index

Cheat	No	No	No	Yes	Yes	Yes	No	No	No	No
	Annual Income									
Sorted Values →	60	70	75	85	90	95	100	120	125	220

Continuous Attributes: Computing Gini Index...

- For efficient computation: for each attribute,
  - Sort the attribute on values
  - Linearly scan these values, each time updating the count matrix and computing gini index
  - Choose the split position that has the least gini index

	Cheat	No	No	No	Yes	Yes	Yes	No	No	No	No
		Annual Income									
Sorted Values →		60	70	75	85	90	95	100	120	125	220

Split Positions →	55		65		72		80		87		92		97		110		122		172		230	
	<=	>	<=	>	<=	>	<=	>	<=	>	<=	>	<=	>	<=	>	<=	>	<=	>	<=	>

Continuous Attributes: Computing Gini Index...

- For efficient computation: for each attribute,
  - Sort the attribute on values
  - Linearly scan these values, each time updating the count matrix and computing gini index
  - Choose the split position that has the least gini index

Cheat	No	No	No	Yes	Yes	Yes	No	No	No	No
Annual Income										
Sorted Values	60	70	75	85	90	95	100	120	125	220

Split Positions	55		65		72		80		87		92		97		110		122		172		230	
	<=	>	<=	>	<=	>	<=	>	<=	>	<=	>	<=	>	<=	>	<=	>	<=	>	<=	>
Yes							0	3														
No							3	4														
Gini							0.343															

Continuous Attributes: Computing Gini Index...

- For efficient computation: for each attribute,
  – Sort the attribute on values
  – Linearly scan these values, each time updating the count matrix and computing gini index
  – Choose the split position that has the least gini index

Cheat	No	No	No	Yes	Yes	Yes	No	No	No	No
	Annual Income									
Sorted Values	60	70	75	85	90	95	100	120	125	220

Split Positions	55		65		72		80		87		92		97		110		122		172		230	
	<=	>	<=	>	<=	>	<=	>	<=	>	<=	>	<=	>	<=	>	<=	>	<=	>	<=	>
Yes							0	3	1	2												
No							3	4	3	4												
Gini							0.343		0.417													

Continuous Attributes: Computing Gini Index...

- For efficient computation: for each attribute,
  - Sort the attribute on values
  - Linearly scan these values, each time updating the count matrix and computing gini index
  - Choose the split position that has the least gini index

Cheat	No	No	No	Yes	Yes	Yes	No	No	No	No
	Annual Income									
Sorted Values	60	70	75	85	90	95	100	120	125	220

Split Positions	55		65		72		80		87		92		97		110		122		172		230	
	<=	>	<=	>	<=	>	<=	>	<=	>	<=	>	<=	>	<=	>	<=	>	<=	>	<=	>
Yes	0	3	0	3	0	3	0	3	1	2	2	1	3	0	3	0	3	0	3	0	3	0
No	0	7	1	6	2	5	3	4	3	4	3	4	3	4	4	3	5	2	6	1	7	0
Gini	0.420		0.400		0.375		0.343		0.417		0.400		***0.300***		0.343		0.375		0.400		0.420	

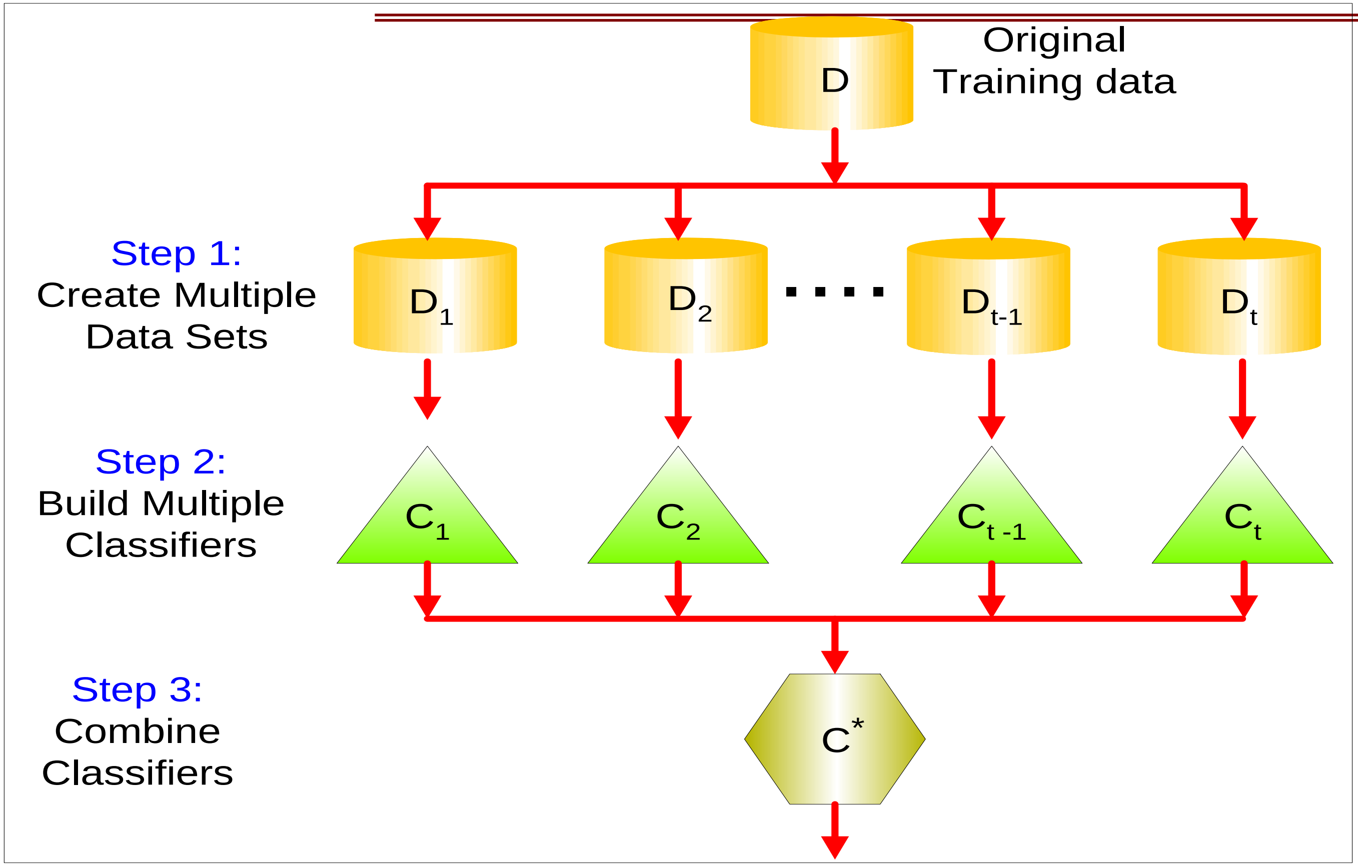
- Random forest classifier

Ensemble Methods

- Construct a set of classifiers from the training data

- Predict class label of previously unseen records by aggregating predictions made by multiple classifiers

General Idea



Why does it work?

- Suppose there are 25 base classifiers
  - Each classifier has error rate, ε = 0.35
  - Assume classifiers are independent
  - Probability that the ensemble classifier makes a wrong prediction:

$$\sum_{i=13}^{25} \binom{25}{i} \varepsilon^i (1-\varepsilon)^{25-i} = 0.06$$

Examples of Ensemble Methods

- How to generate an ensemble of classifiers?
  - Bagging
  - Boosting

Bagging

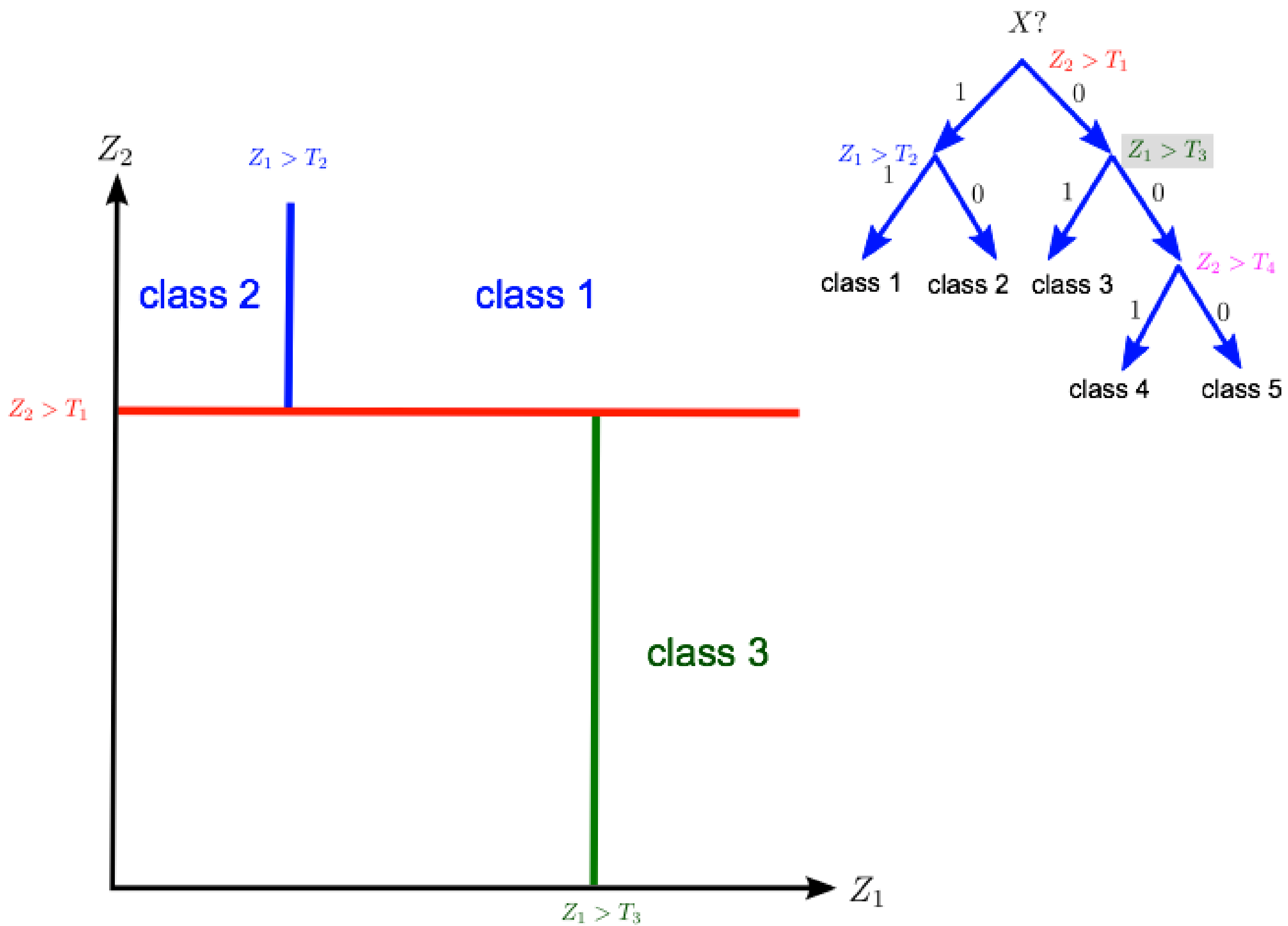
- Sampling with replacement

Original Data	1	2	3	4	5	6	7	8	9	10
Bagging (Round 1)	7	8	10	8	2	5	10	10	5	9
Bagging (Round 2)	1	4	9	1	2	3	2	7	3	2
Bagging (Round 3)	1	8	5	10	5	5	9	6	3	7

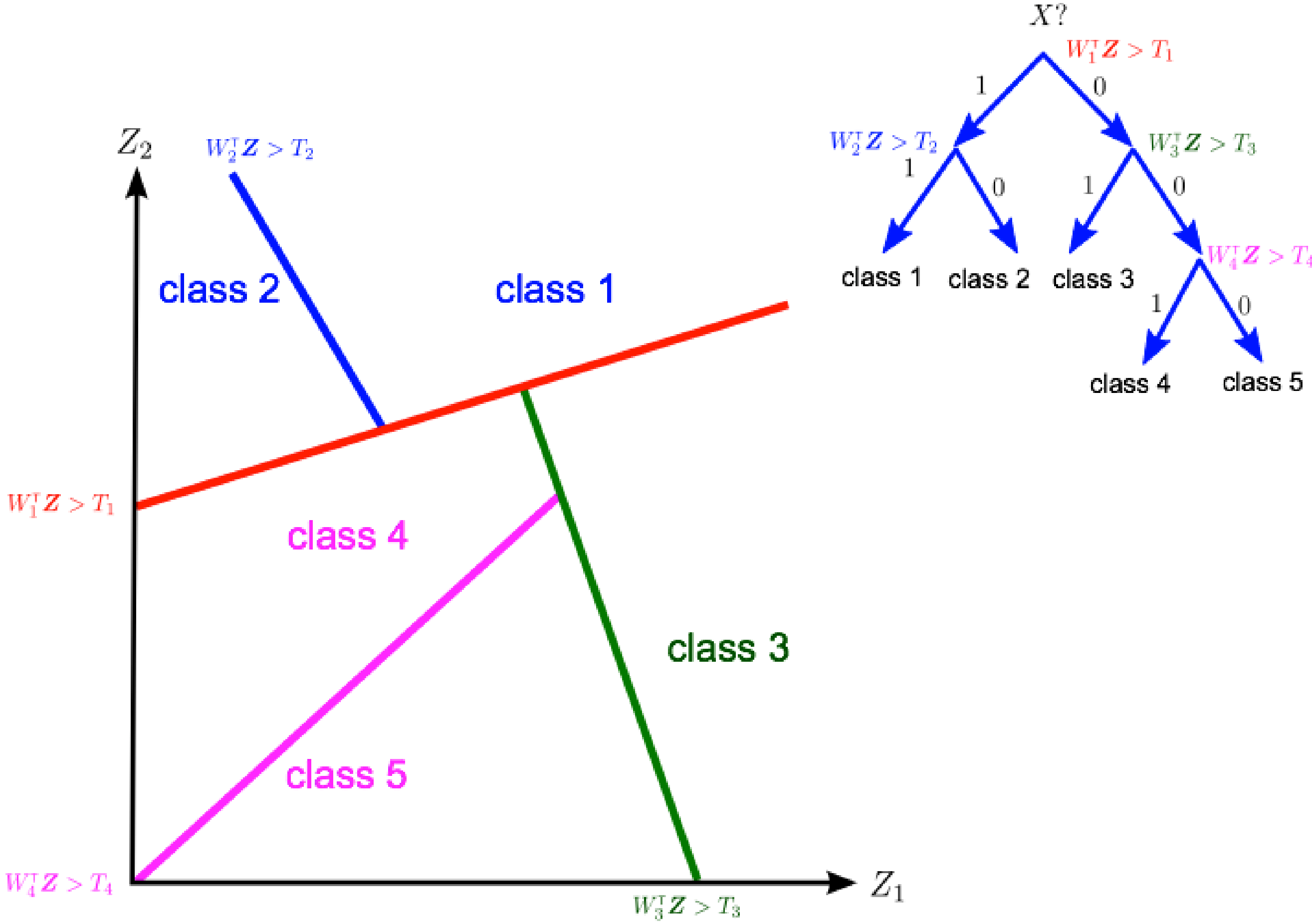
- Build classifier on each bootstrap sample

- Each sample has probability $(1 – 1/n)^n$ of being selected

Feature Space

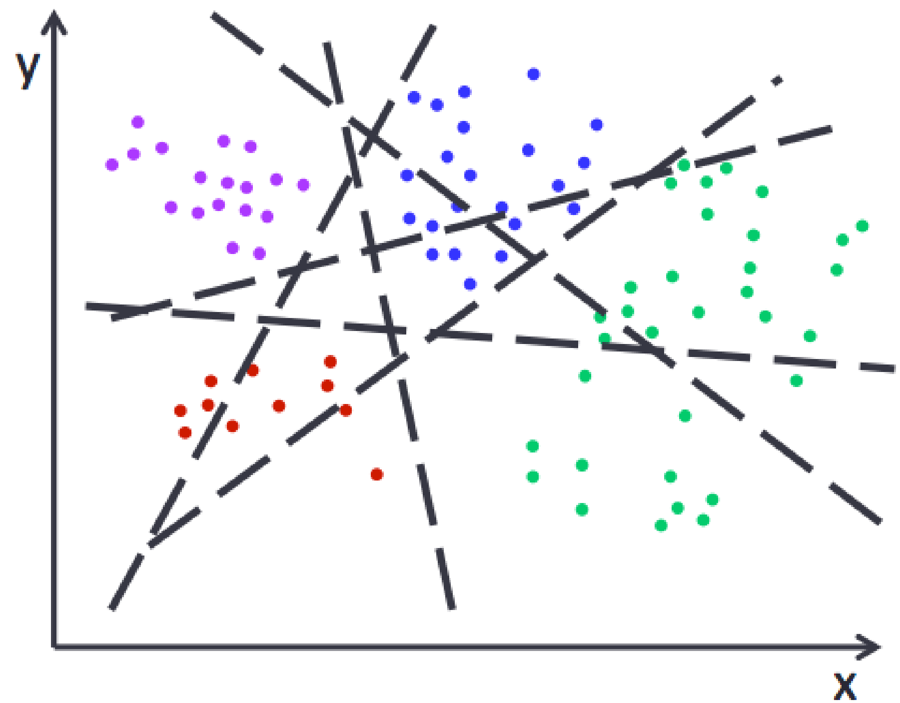


Feature Space



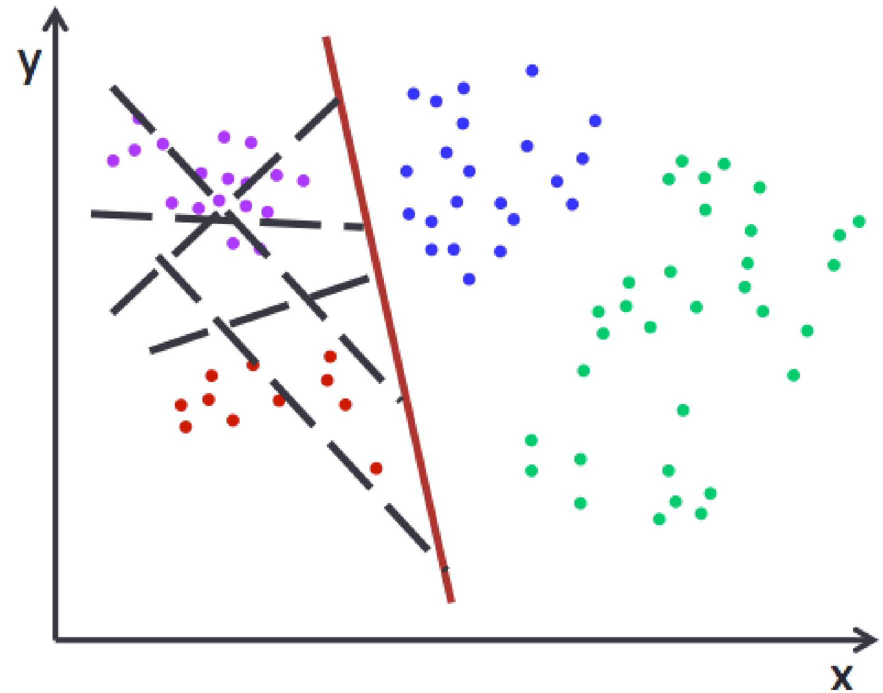
Example

- Try several lines, chosen at random
- Keep line that best separates data
  - Information gain
- Recurse



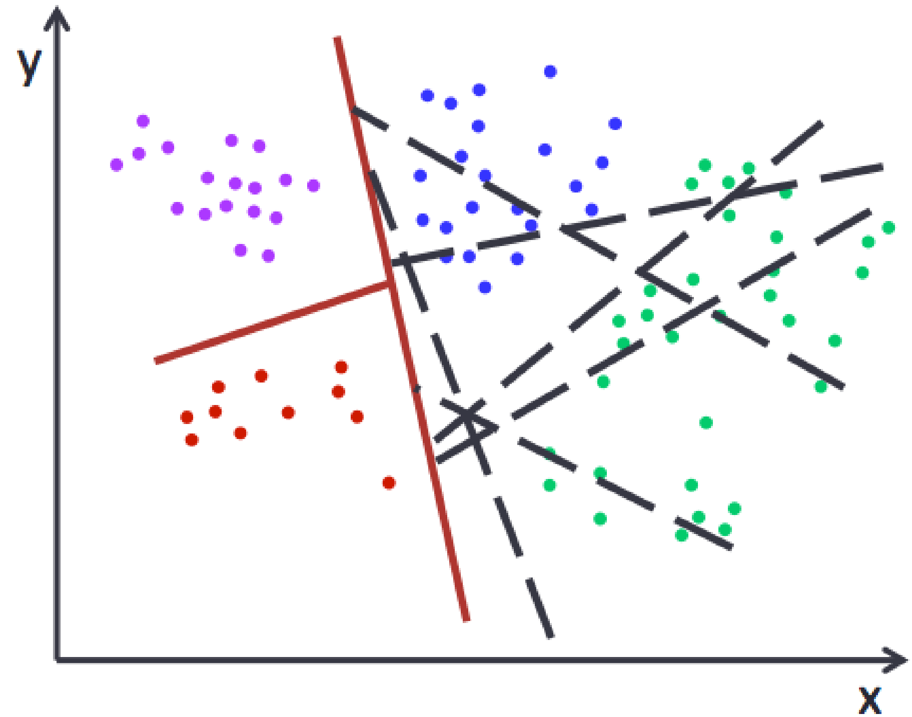
Example

- Try several lines, chosen at random
- Keep line that best separates data
  - Information gain
- Recurse



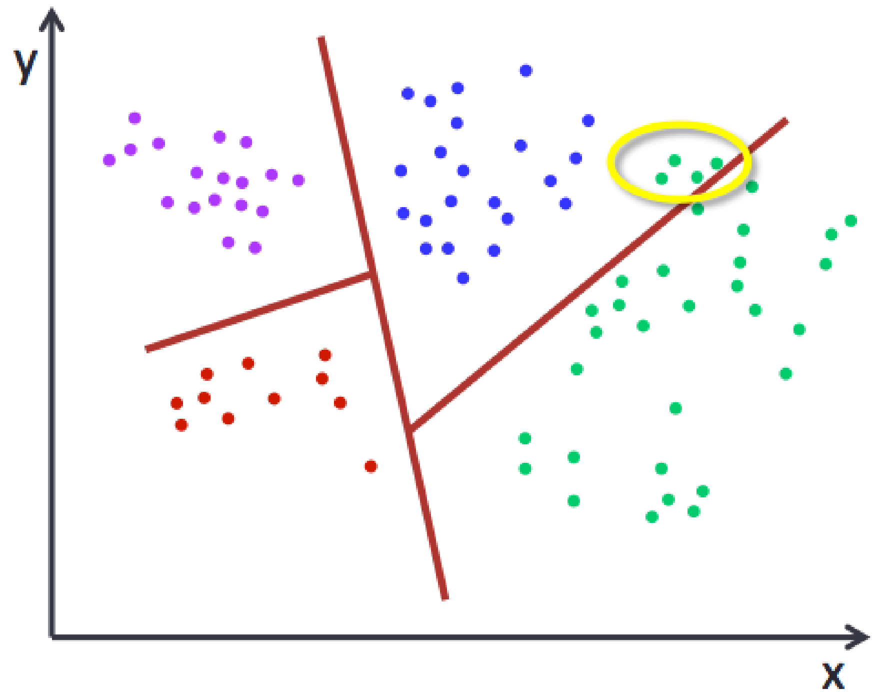
Example

- Try several lines, chosen at random
- Keep line that best separates data
  - Information gain
- Recurse



Example

- Try several lines, chosen at random
- Keep line that best separates data
  - Information gain
- Recurse

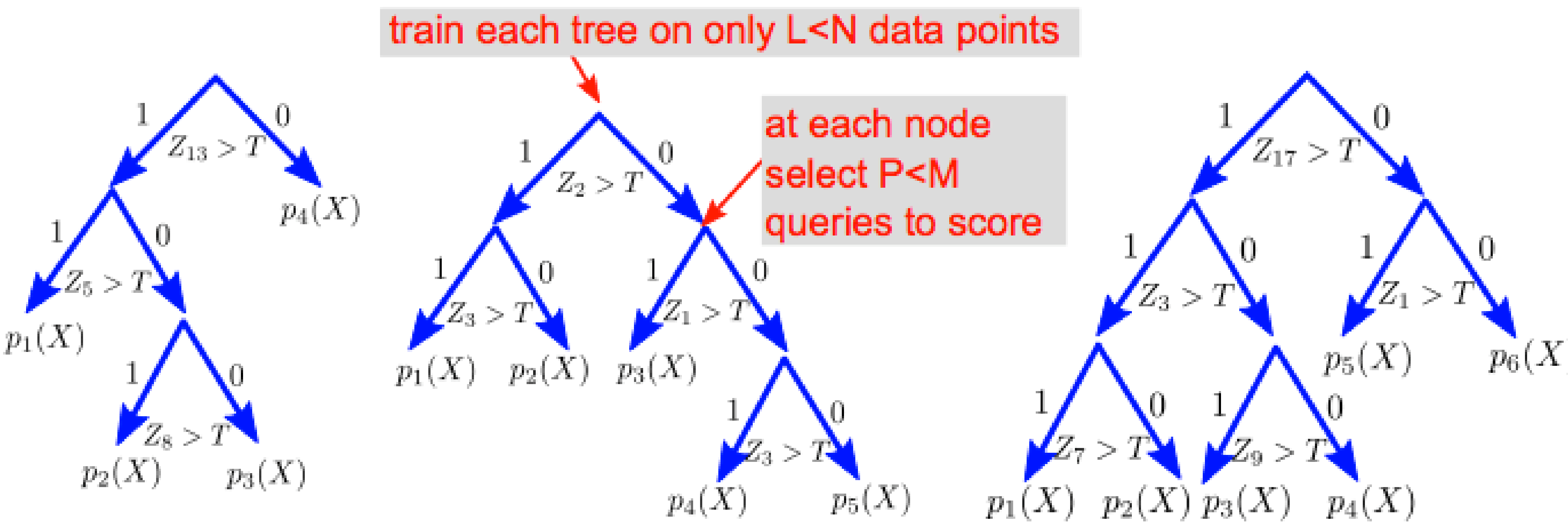


- Random forest classifier

Random Forest

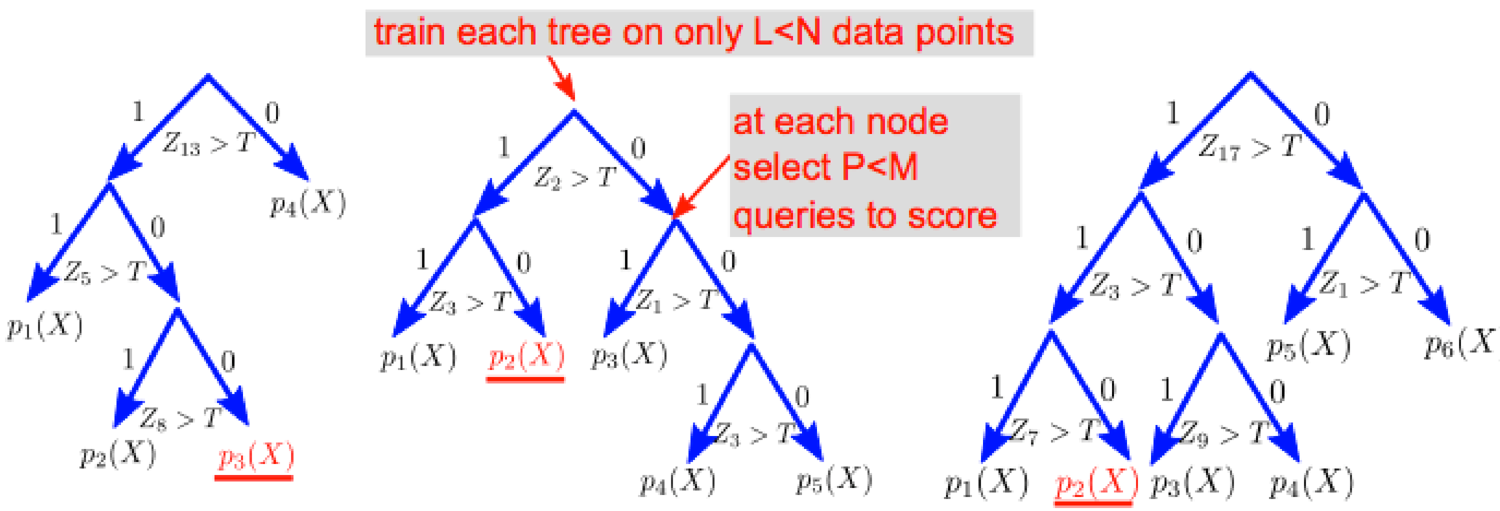
- Train a collection of trees
- Ensemble method
- Averages over (diverse) classification trees (a forest)
- For each tree draw L samples of the original data
- At each node randomly sample P queries and choose the best among them

Random Forest

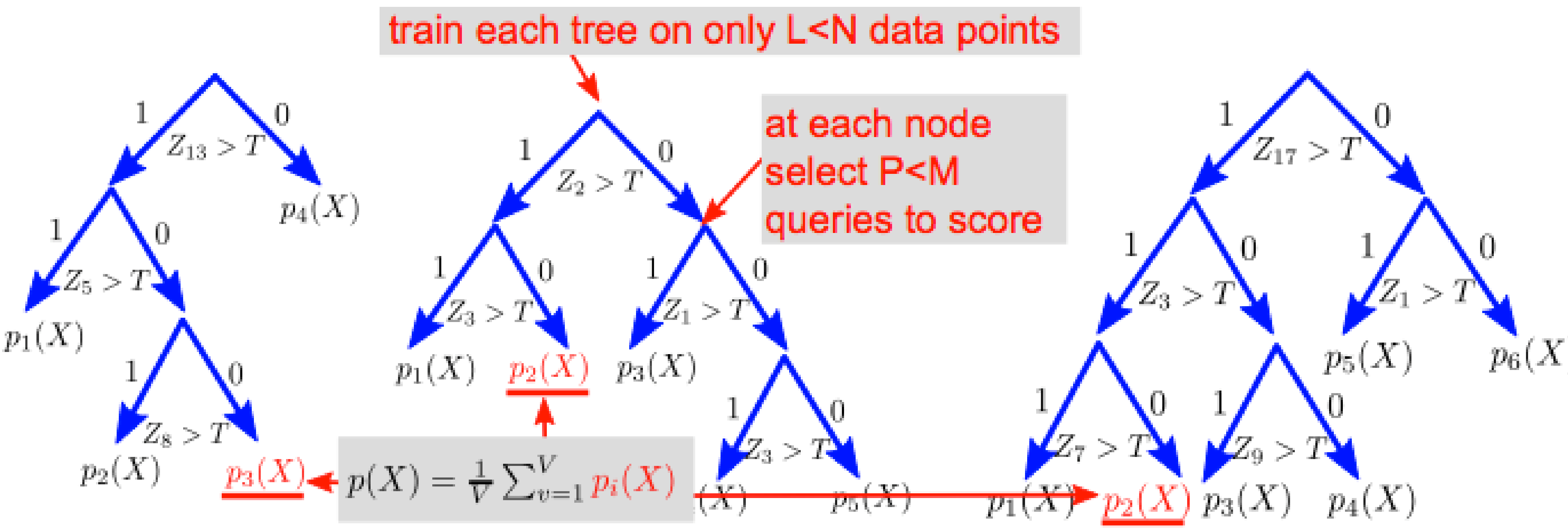


Random Forest

- Aggregate across trees (majority vote or average ⇒ mixture model)
- Avoids over-fitting and computationally efficient



Random Forest

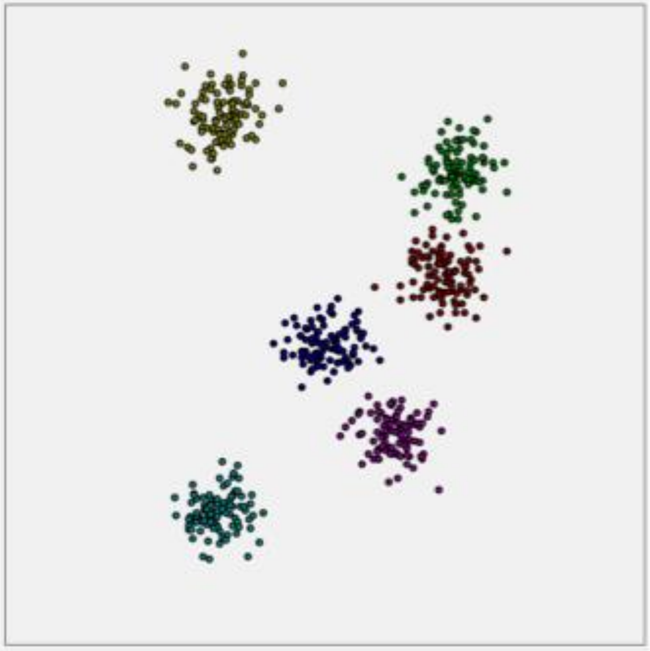


Random Forests

- Random forests are a very popular tool for classification, e.g. in computer vision
- Based on decision trees: classifiers constructed greedily using the conditional entropy
- The extension hinges on two ideas:
  - building an ensemble of trees by training on subsets of data
  - considering a reduced number of possible queries (attributes) at each node

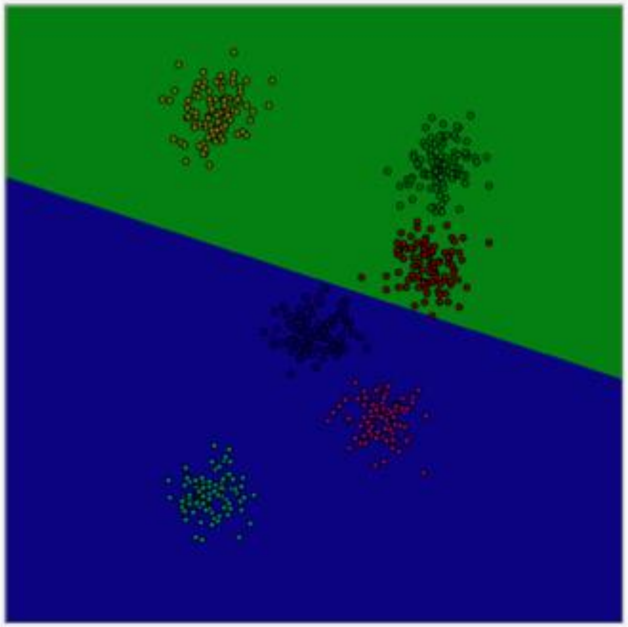
Random Forests

- 6 classes in a 2 dimensional feature space.
- Split functions are lines in this space.



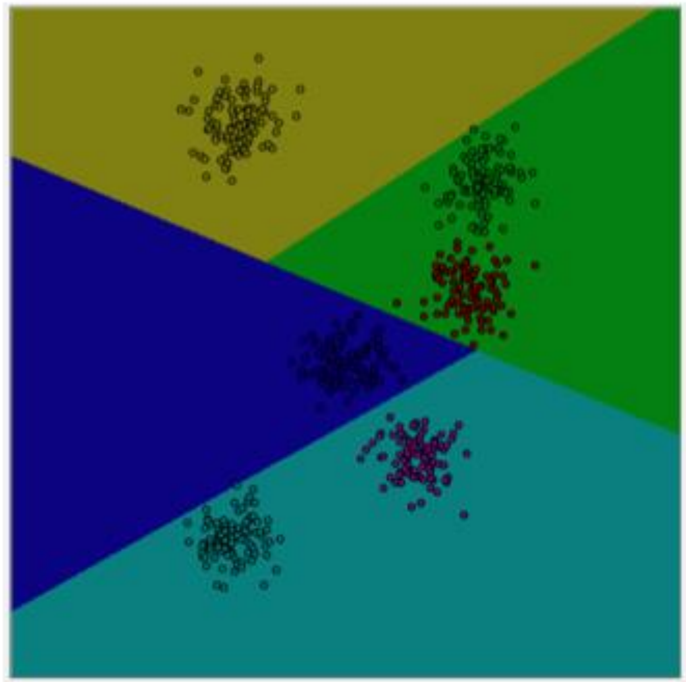
Random Forests

- With a depth 2 tree, we cannot separate all six classes.



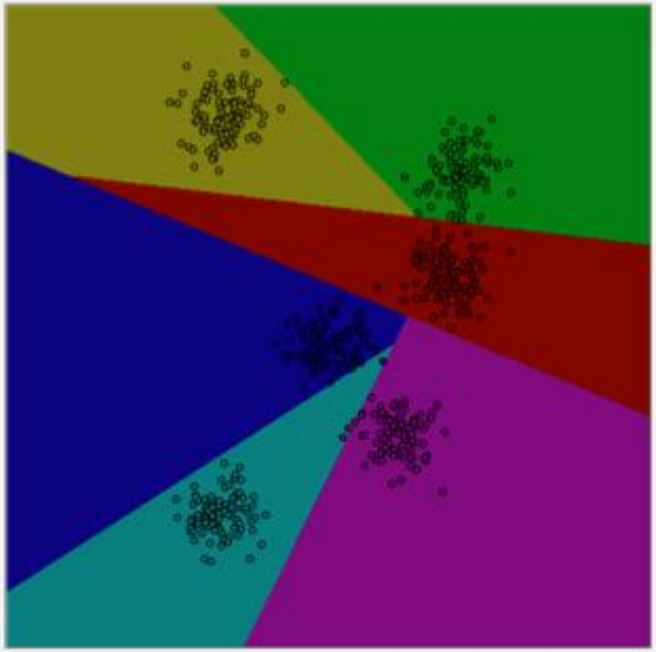
Random Forests

- With a depth 3 tree, we can do better, but still cannot separate all six classes.

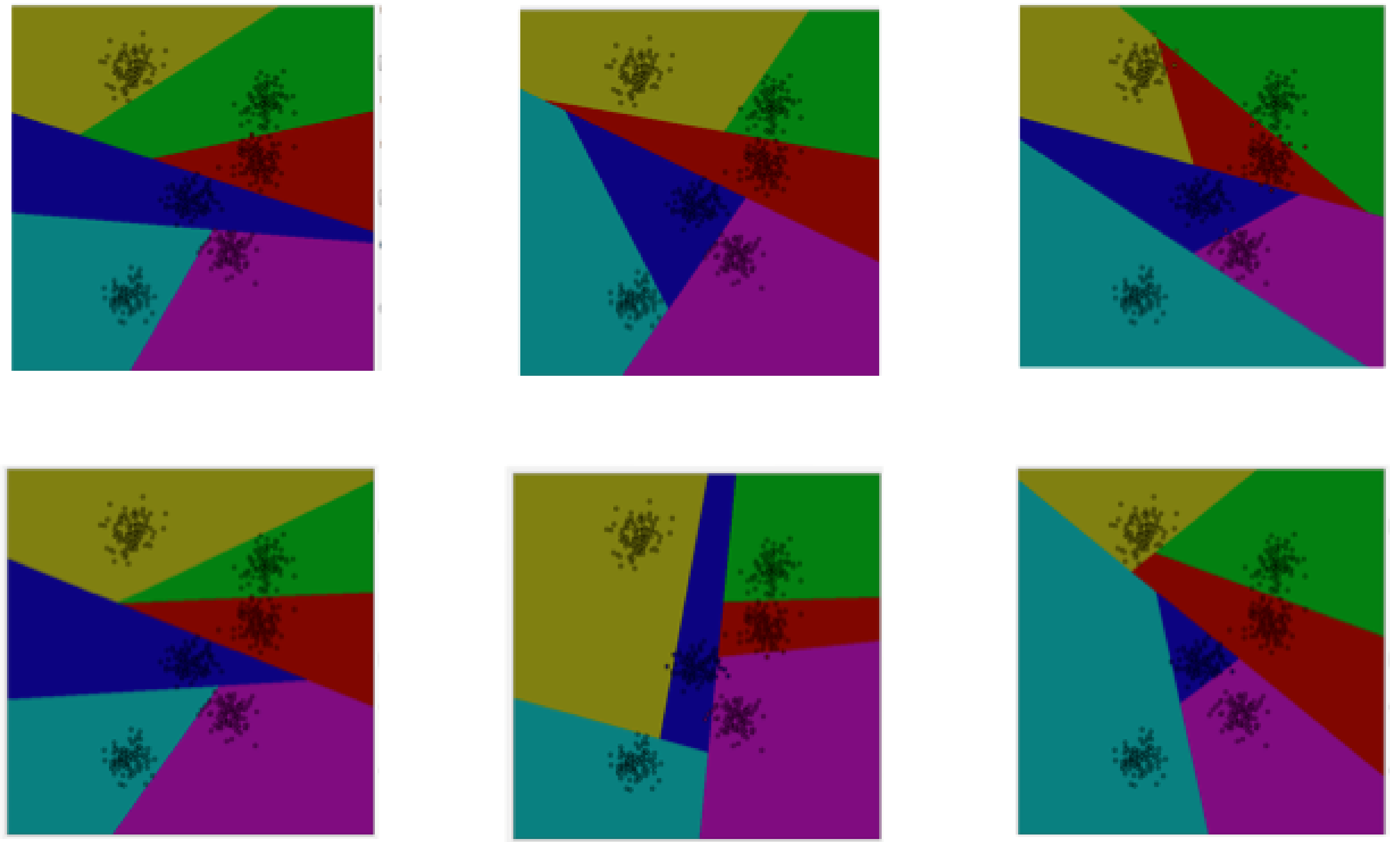


Random Forests

- With a depth 4 tree, we now have at least as many leaf nodes as classes,
- and so are able to classify most examples correctly.



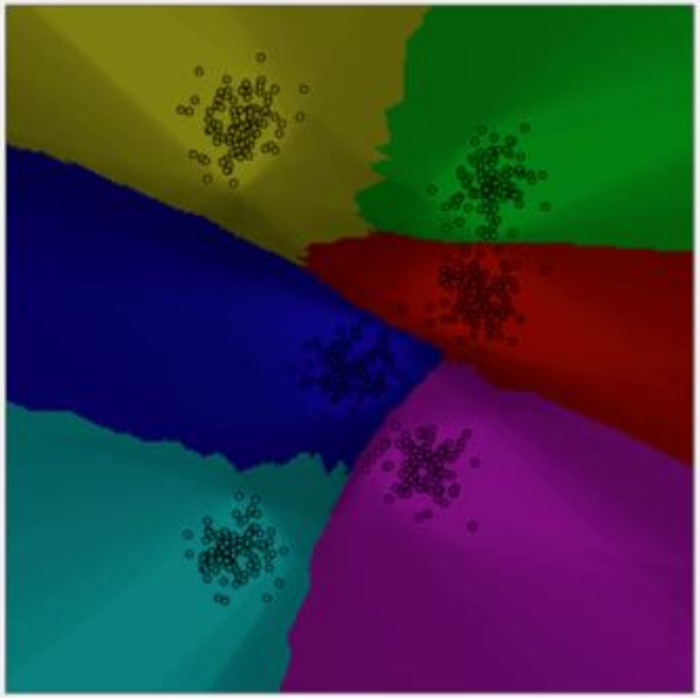
Random Forests



Randomly trained decision trees can give rise to very different decision boundaries, none of which is particularly good on its own.

Random Forests

- Bagging (averaging together) many trees
  - decision boundaries look very sensible
  - even quite close to the max margin classifier (Shading represents entropy – darker is higher entropy).



- Association Rule Mining

Association Analysis

- Large Data
  - Transactions
  - Market basket transactions

Association Analysis

- Large Data
  - Transactions
  - Market basket transactions

- Association Analysis
  - Discovering of interesting relationships in large data sets

Market Basket Analysis

TID	Items
1	Bread, Milk
2	Bread, Diaper, Beer, Eggs
3	Milk, Diaper, Beer, Coke
4	Bread, Milk, Diaper, Beer
5	Bread, Milk, Diaper, Coke

Market Basket Analysis

TID	Items
1	Bread, Milk
2	Bread, Diaper, Beer, Eggs
3	Milk, Diaper, Beer, Coke
4	Bread, Milk, Diaper, Beer
5	Bread, Milk, Diaper, Coke

Market Basket Analysis

- Diapers + Beer!

Market Basket Analysis

- Diapers + Beer!
- Diapers ->
  - baby ->
  - don't go out to a bar ->
  - buy more beer for home

Market Basket Analysis

- Hot dog and mustard

+ **Mustard**

Market Basket Analysis

- Hot dog and mustard

Association Rules: General Idea

- Given a set of baskets
  - Want to discover association rules
  - People who bought {x,y,z} tend to buy {v,w}
    - Amazon!
- 2 step approach:
  - Find frequent itemsets
  - Generate association rules

Applications

- Items = products, Baskets = sets of products someone bought in one trip to the store
  - Real market baskets: Chain stores keep TBs of data about what customers buy together
  - Tells how typical customers navigate stores, lets them position tempting items
  - Suggests tie-in “tricks”, e.g., run sale on diapers and raise the price of beer
  - Amazon’s people who bought X also bought Y

Applications

- Plagiarism
  - Baskets = sentences, Items = documents containing those sentences
  - Items that appear together too often could represent plagiarism
  - Notice items are “in” baskets, not “part of”

Applications

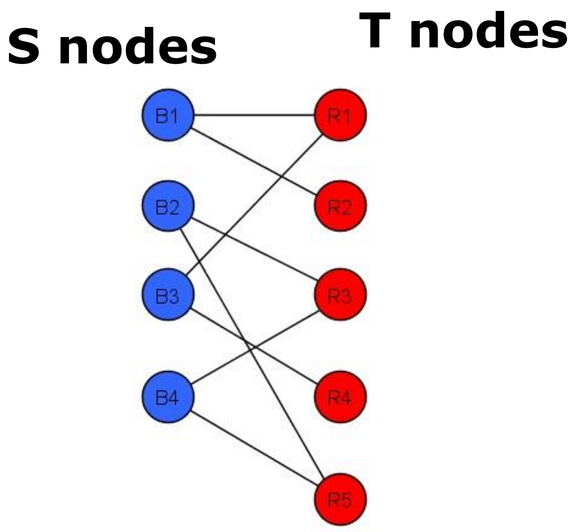
- Medical domain
  - Baskets = patients, Items = drugs & side-effects
  - Has been used to detect combinations of drugs that result in particular side-effects
  - But requires extension: Absence of an item needs to be observed as well as presence

Applications

- Biomarkers
  - Baskets are sets of data about the patient: genome, blood-chemistry analysis, medical history. Items are biomarkers s.a. genes, blood protein or diseases
  - Frequent item: one disease + biomarkers

Applications

- Finding communities in graphs (e.g., web)
  - Baskets = nodes; Items = outgoing neighbors
  - Searching for complete bipartite subgraphs Ks,t of a big graph



Association Analysis

- A large set of items
  - e.g., things sold in a supermarket
- A large set of baskets each is a small subset of items
  - e.g., the things one customer buys on one day
- A general many-many mapping (association) between two kinds of things
- But we ask about connections among “items”, not “baskets”

- Association Analysis
  - Discovering of interesting relationships in large data sets

TID	*Items*
1	Bread, Milk
2	Bread, Diaper, Beer, Eggs
3	Milk, Diaper, Beer, Coke
4	Bread, Milk, Diaper, Beer
5	Bread, Milk, Diaper, Coke

Market basket transactions

Association Analysis

- Association Analysis
  - Discovering of interesting relationships in large data sets

TID	*Items*
1	Bread, Milk
2	Bread, Diaper, Beer, Eggs
3	Milk, Diaper, Beer, Coke
4	Bread, Milk, Diaper, Beer
5	Bread, Milk, Diaper, Coke

Market basket transactions

{Diapers → Beer}
{Beer, Bread} → {Milk}

Association Analysis

- Association Analysis
  - Discovering of interesting relationships in large data sets

TID	*Items*
1	Bread, Milk
2	Bread, Diaper, Beer, Eggs
3	Milk, Diaper, Beer, Coke
4	Bread, Milk, Diaper, Beer
5	Bread, Milk, Diaper, Coke

Market basket transactions

Association rule:

{Diapers → Beer}
{Beer, Bread} → {Milk}

Association Analysis

- **Association Analysis**
  - Discovering of interesting relationships in large data sets

TID	*Items*
1	Bread, Milk
2	Bread, Diaper, Beer, Eggs
3	Milk, Diaper, Beer, Coke
4	Bread, Milk, Diaper, Beer
5	Bread, Milk, Diaper, Coke

Market basket transactions

Association rule:

{Diapers → Beer}
{Beer, Bread} → {Milk}

Co-occurrence, not causality

Problem Definition

- Set of items I={i1,i2,…,id}
- Set of transaction T={t1,t2,…tN}

TID	Items
1	Bread, Milk
2	Bread, Diaper, Beer, Eggs
3	Milk, Diaper, Beer, Coke
4	Bread, Milk, Diaper, Beer
5	Bread, Milk, Diaper, Coke

Market basket transactions

I={Bread, Milk, Diaper, Eggs, Beer, Coke}

T={t1={Bread, Milk},
t2={Bread,Diaper, Beer, Eggs},
…}

Problem Definition

- Itemset $X = \{i | i \subseteq I\}$
  - {Bread, Milk}
  - *k*-itemset has k items
  - {Bread, Milk} is a *2*-itemset
- Transaction t_i contains an itemset X

TID	*Items*
1	Bread, Milk
2	Bread, Diaper, Beer, Eggs
3	Milk, Diaper, Beer, Coke
4	Bread, Milk, Diaper, Beer
5	Bread, Milk, Diaper, Coke

Market basket transactions

$t_i \subseteq T$

t1={Bread, Milk}

t_i contains X

$X \subseteq t_i$

$X = \{i_k, i_m, ..\}$,

where $i_k \subseteq I$

X={Bread, Milk}

Problem Definition

- Itemset $X = \{i | i \subseteq I\}$
  - {Bread, Milk}
  - *k*-itemset has k items
  - {Bread, Milk} is a *2*-itemset
- Transaction t_i contains an itemset X

TID	*Items*
1	Bread, Milk
2	Bread, Diaper, Beer, Eggs
3	Milk, Diaper, Beer, Coke
4	Bread, Milk, Diaper, Beer
5	Bread, Milk, Diaper, Coke

Market basket transactions

$t_i \subseteq T$

t1={Bread, Milk}

t_i contains X, Y

$X \subseteq t_i$, $Y \subseteq t_i$

X={Bread, Milk}
Y={Bread}

Problem Definition

- Set of items $I=\{i_1,i_2,\ldots,i_d\}$
- Set of transaction $T=\{t_1,t_2,\ldots t_N\}$
- Itemset $X = \{i|i \subseteq I\}$
  - *k*-itemset has k items
- Transaction t_i contains an itemset X

Support Count

- Support Count of an itemset X: $\sigma(X)$
  - $\sigma(X)$ = Number of transactions that contain X

Support Count

- Support Count of an itemset X
  - Number of transactions that contain X
  - Number of transations that support {Bread, Milk}?

TID	*Items*
1	Bread, Milk
2	Bread, Diaper, Beer, Eggs
3	Milk, Diaper, Beer, Coke
4	Bread, Milk, Diaper, Beer
5	Bread, Milk, Diaper, Coke

Market basket transactions

Support Count

- Support Count of an itemset X
  - Number of transactions that contain X
  - Number of transations that support {Bread, Milk}?

TID	*Items*
1	Bread, Milk
2	Bread, Diaper, Beer, Eggs
3	Milk, Diaper, Beer, Coke
4	Bread, Milk, Diaper, Beer
5	Bread, Milk, Diaper, Coke

Market basket transactions

- $\sigma(\{\text{Bread, Milk}\}) = 3$

Support Count

- Support Count of an itemset X
  - Number of transactions that contain X
  - Number of transations that support {Bread}?

TID	*Items*
1	Bread, Milk
2	Bread, Diaper, Beer, Eggs
3	Milk, Diaper, Beer, Coke
4	Bread, Milk, Diaper, Beer
5	Bread, Milk, Diaper, Coke

Market basket transactions

Support Count

- Support Count of an itemset X
  - Number of transactions that contain X
  - Number of transations that support {Bread}?

TID	*Items*
1	Bread, Milk
2	Bread, Diaper, Beer, Eggs
3	Milk, Diaper, Beer, Coke
4	Bread, Milk, Diaper, Beer
5	Bread, Milk, Diaper, Coke

Market basket transactions

- $\sigma(\{\text{Bread}\}) = 4$

Support Count

- Support Count of an itemset X
  - Number of transactions that contain X
  - Number of transations that support {Bread, Milk, Diaper, Coke}?

TID	*Items*
1	Bread, Milk
2	Bread, Diaper, Beer, Eggs
3	Milk, Diaper, Beer, Coke
4	Bread, Milk, Diaper, Beer
5	Bread, Milk, Diaper, Coke

Market basket transactions

Support Count

- Support Count of an itemset X
  - Number of transactions that contain X
  - Number of transations that support {Bread, Milk, Diaper, Coke}?

TID	Items
1	Bread, Milk
2	Bread, Diaper, Beer, Eggs
3	Milk, Diaper, Beer, Coke
4	Bread, Milk, Diaper, Beer
5	Bread, Milk, Diaper, Coke

Market basket transactions

- **σ({Bread, Milk Diaper, Coke}) = 1**

Association Rule

- Association rule is an implication expression
  - X -> Y where X and Y are disjoint itemsets

Association Rule

- Association rule is an implication expression
  - X -> Y where X and Y are disjoint itemsets

TID	*Items*
1	Bread, Milk
2	Bread, Diaper, Beer, Eggs
3	Milk, Diaper, Beer, Coke
4	Bread, Milk, Diaper, Beer
5	Bread, Milk, Diaper, Coke

Market basket transactions

Association rules:

{Diapers $\rightarrow$ Beer}
{Beer, Bread} $\rightarrow$ {Milk}

Association Rule

- Association rule:
  - Support
  - Confidence

Association Rule

- Association rule:
  - Support X->Y
    - Number of transactions containing $X \cup Y$
    - $S(X\text{->}Y) = \sigma(X \cup Y)/N$

TID	*Items*
1	Bread, Milk
2	Bread, Diaper, Beer, Eggs
3	Milk, Diaper, Beer, Coke
4	Bread, Milk, Diaper, Beer
5	Bread, Milk, Diaper, Coke

Market basket transactions

Association rules:

$\{\text{Diapers} \rightarrow \text{Beer}\}$
$\{\text{Beer, Bread}\} \rightarrow \{\text{Milk}\}$

$S(\text{Diapers} \cup \text{Beer}) =$

?

$S(\text{Beer, Bread} \cup \text{Milk}) =$

?

Association Rule

- **Association rule:**
  - Support X->Y
    - Number of transactions containing $X \cup Y$
    - $S(X\text{->}Y) = \sigma(X \cup Y)/N$

TID	*Items*
1	Bread, Milk
2	Bread, Diaper, Beer, Eggs
3	Milk, Diaper, Beer, Coke
4	Bread, Milk, Diaper, Beer
5	Bread, Milk, Diaper, Coke

Market basket transactions

Association rules:

$\{\text{Diapers} \rightarrow \text{Beer}\}$
$\{\text{Beer, Bread}\} \rightarrow \{\text{Milk}\}$

$S(\text{Diapers} \cup \text{Beer}) =$ **3/5**

$S(\text{Beer, Bread} \cup \text{Milk}) =$ **1/5**

Association Rule

- Association rule:
  - Support
  - Confidence

Association Rule

- **Association rule:**
  - Support
  - Confidence X->Y
    - How often transactions that contain X also contain Y
    - $c(X \rightarrow Y) = \sigma(X \cup Y) / \sigma(X)$

Association Rule

- Association rule:
  - Support
  - Confidence X->Y
    - How often transactions that contain X also contain Y
    - $c(X \rightarrow Y) = \sigma(X \cup Y) / \sigma(X)$

TID	Items
1	Bread, Milk
2	Bread, Diaper, Beer, Eggs
3	Milk, Diaper, Beer, Coke
4	Bread, Milk, Diaper, Beer
5	Bread, Milk, Diaper, Coke

Market basket transactions

Association rules:

{Diapers → Beer}
{Beer, Bread} → {Milk}

$C(\textbf{Diapers} \rightarrow \textbf{Beer}) =$

?

$C(\textbf{Beer, Bread} \rightarrow \textbf{Milk}) =$

?

Association Rule

- Association rule:
  - Support
  - Confidence X->Y
    - How often transactions that contain X also contain Y
    - $c(X\text{->}Y) = \sigma(X \cup Y)/ \sigma(X)$

TID	Items
1	Bread, Milk
2	Bread, Diaper, Beer, Eggs
3	Milk, Diaper, Beer, Coke
4	Bread, Milk, Diaper, Beer
5	Bread, Milk, Diaper, Coke

Market basket transactions

Association rules:

{Diapers → Beer}
{Beer, Bread} → {Milk}

$C(\textbf{Diapers} \rightarrow \textbf{Beer}) =$

3/4

$C(\textbf{Beer, Bread} \rightarrow \textbf{Milk}) =$

1/2

Use of Support and Confidence

- Support
  - Rule with a low support can occur by chance
  - Low support rules are not interesting from the business perspective
  - Eliminate uninteresting rules
- Confidence
  - Reliability of the implication from an association rule X->Y
  - Conditional probability P(Y|X)

Association Rule Mining Problem

- Given a set of transactions T, the goal of association rule mining is to find all rules having
  - support ≥ *minsupport* threshold
  - confidence ≥ *minconfidence* threshold

Association Rule Mining Problem

- Given a set of transactions T, the goal of association rule mining is to find all rules having
  - support ≥ *minsupport* threshold
  - confidence ≥ *minconfidence* threshold

Minsup=0.4
Minconf=0.6

{Milk,Diaper} → {Beer} (s=0.4, c=0.67)
{Milk,Beer} → {Diaper} (s=0.4, c=1.0)
{Diaper,Beer} → {Milk} (s=0.4, c=0.67)
{Beer} → {Milk,Diaper} (s=0.4, c=0.67)
{Diaper} → {Milk,Beer} (s=0.4, c=0.5)
{Milk} → {Diaper,Beer} (s=0.4, c=0.5)

Association Rule Mining Problem

- Given a set of transactions T, the goal of association rule mining is to find all rules having
  - support ≥ *minsupport* threshold
  - confidence ≥ *minconfidence* threshold

{Milk,Diaper} → {Beer} (s=0.4, c=0.67)
{Milk,Beer} → {Diaper} (s=0.4, c=1.0)
{Diaper,Beer} → {Milk} (s=0.4, c=0.67)
{Beer} → {Milk,Diaper} (s=0.4, c=0.67)
{Diaper} → {Milk,Beer} (s=0.4, c=0.5)
{Milk} → {Diaper,Beer} (s=0.4, c=0.5)

Minsup=0.4
Minconf=0.6

{Milk,Diaper} → {Beer}
{Diaper,Beer} → {Milk}
{Beer} → {Milk,Diaper}

Computational Challenge

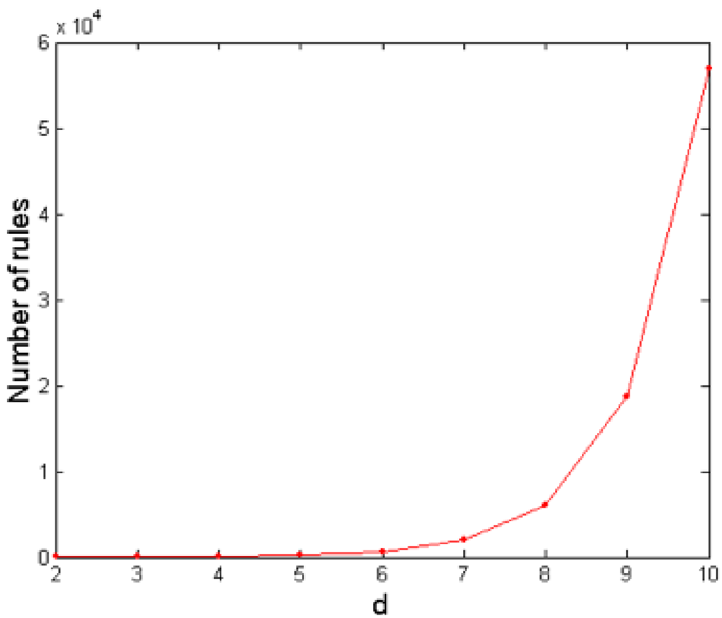
- Brute-force approach
  - Compute support and confidence for every possilbe rule

{Milk,Diaper} → {Beer} (s=0.4, c=0.67)
{Milk,Beer} → {Diaper} (s=0.4, c=1.0)
{Diaper,Beer} → {Milk} (s=0.4, c=0.67)
{Beer} → {Milk,Diaper} (s=0.4, c=0.67)
{Diaper} → {Milk,Beer} (s=0.4, c=0.5)
{Milk} → {Diaper,Beer} (s=0.4, c=0.5)

  - In our example d=6, there are 602 rules
    - If minsup=20%
    - If minconf=50%, then
    - 80% of rules are discarded

Computational Challenge

- The number of possible rules that contains d items
  - $R = 3^d - 2^{(d+1)} + 1$



$$R = \sum_{k=1}^{d-1}\left[\binom{d}{k} \times \sum_{j=1}^{d-j}\binom{d-k}{j}\right]$$
$$= 3^d - 2^{d+1} + 1$$

Computational Challenge

TID	Items
1	Bread, Milk
2	Bread, Diaper, Beer, Eggs
3	Milk, Diaper, Beer, Coke
4	Bread, Milk, Diaper, Beer
5	Bread, Milk, Diaper, Coke

Example of Rules:

{Milk,Diaper} $\rightarrow$ {Beer} (s=0.4, c=0.67)
{Milk,Beer} $\rightarrow$ {Diaper} (s=0.4, c=1.0)
{Diaper,Beer} $\rightarrow$ {Milk} (s=0.4, c=0.67)
{Beer} $\rightarrow$ {Milk,Diaper} (s=0.4, c=0.67)
{Diaper} $\rightarrow$ {Milk,Beer} (s=0.4, c=0.5)
{Milk} $\rightarrow$ {Diaper,Beer} (s=0.4, c=0.5)

- All the above rules are binary partitions of the same itemset: {Milk, Diaper, Beer}
- Rules originating from the same itemset have identical support but can have different confidence
- Thus, we may decouple the support and confidence requirements

Computational Challenge

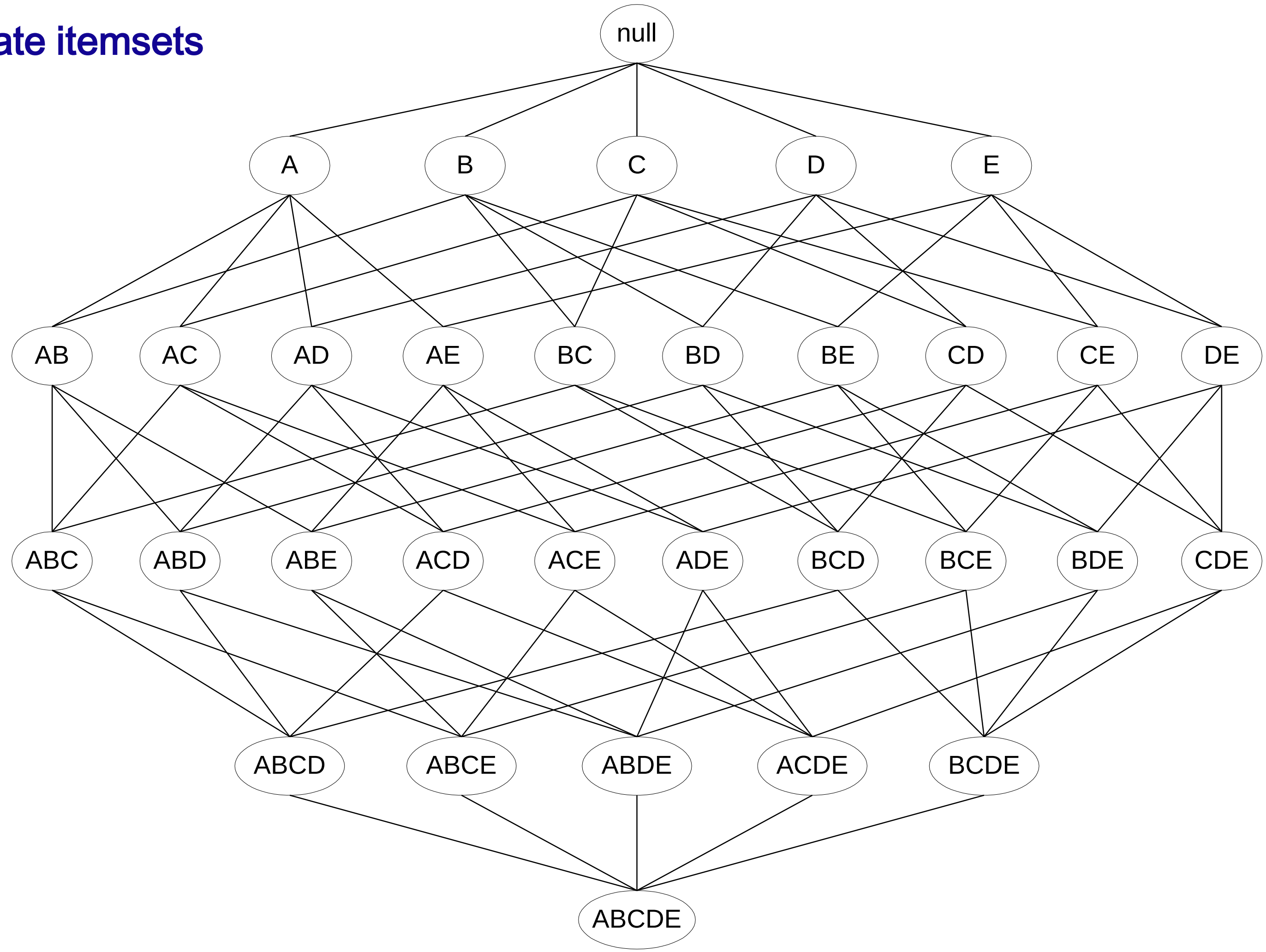
- Two steps
  - Frequent Itemset Generation
    - Generate all itemsets with support $\geq$ *minsup*
  - Rule Generation
    - Generate high confidence rules from each frequent itemset

Frequent itemset

- Brute-force approach
  - Support count for every itemset
  - Use lattice structure

Frequent itemset

- Use lattice structure
- Enumerate itemsets



Frequent itemset

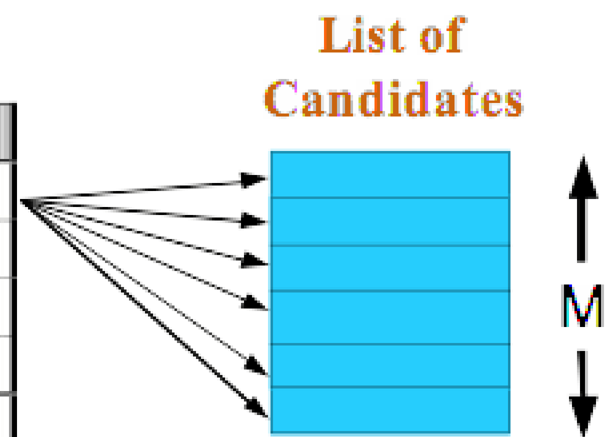
- Each itemset in the lattice is a candidate frequent itemset
- Count the support of each candidate by scanning the database
  - Match each transaction against every candidate

Transactions

TID	Items
1	Bread, Milk
2	Bread, Diaper, Beer, Eggs
3	Milk, Diaper, Beer, Coke
4	Bread, Milk, Diaper, Beer
5	Bread, Milk, Diaper, Coke

N

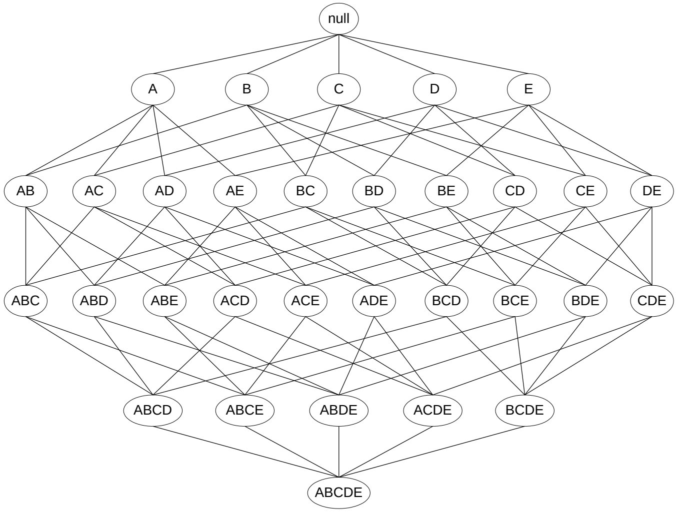
w



Frequent itemset

- Use lattice structure

I={A,B,C,D,E}



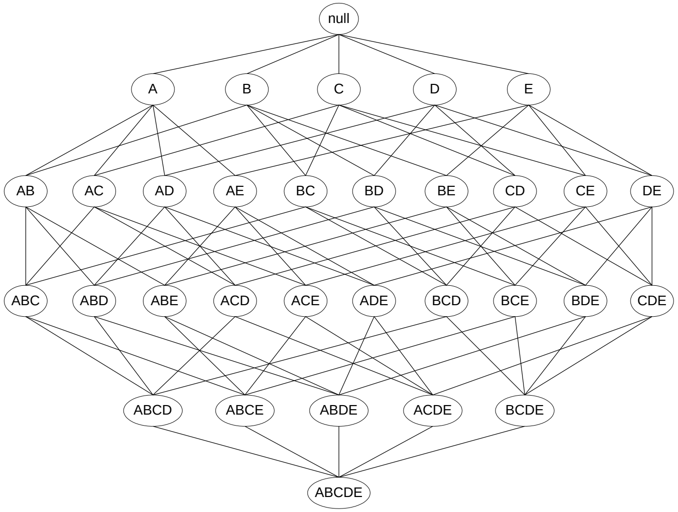
Frequent itemset

- Use lattice structure

I={A,B,C,D,E}

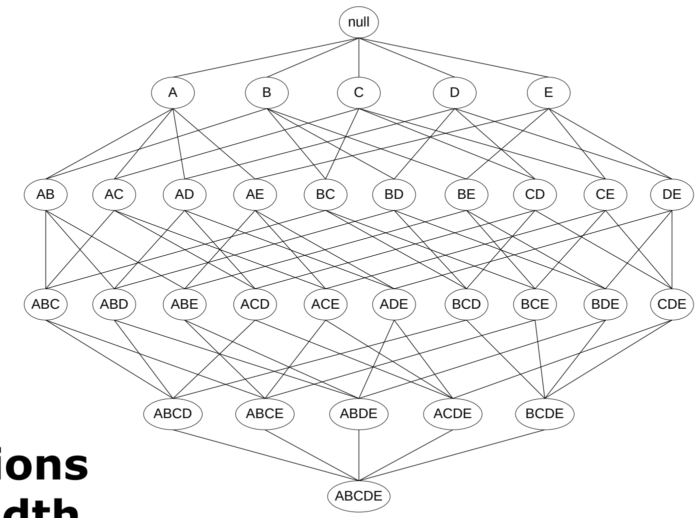
K items
|I|=k

M = 2^k-1 itemsets



Frequent itemset

- Use lattice structure



I={A,B,C,D,E}

K items
|I|=k

M = 2^k-1 itemsets
N it the number of transactions
w is the max transaction width
(max number of items per transaction)

O(NMw) computations

Reduce Complexity

- Reduce the number of candidate itemsets M
- Reduce the number of transactions
- Reduce the number of comparisons

- Apriori Principle

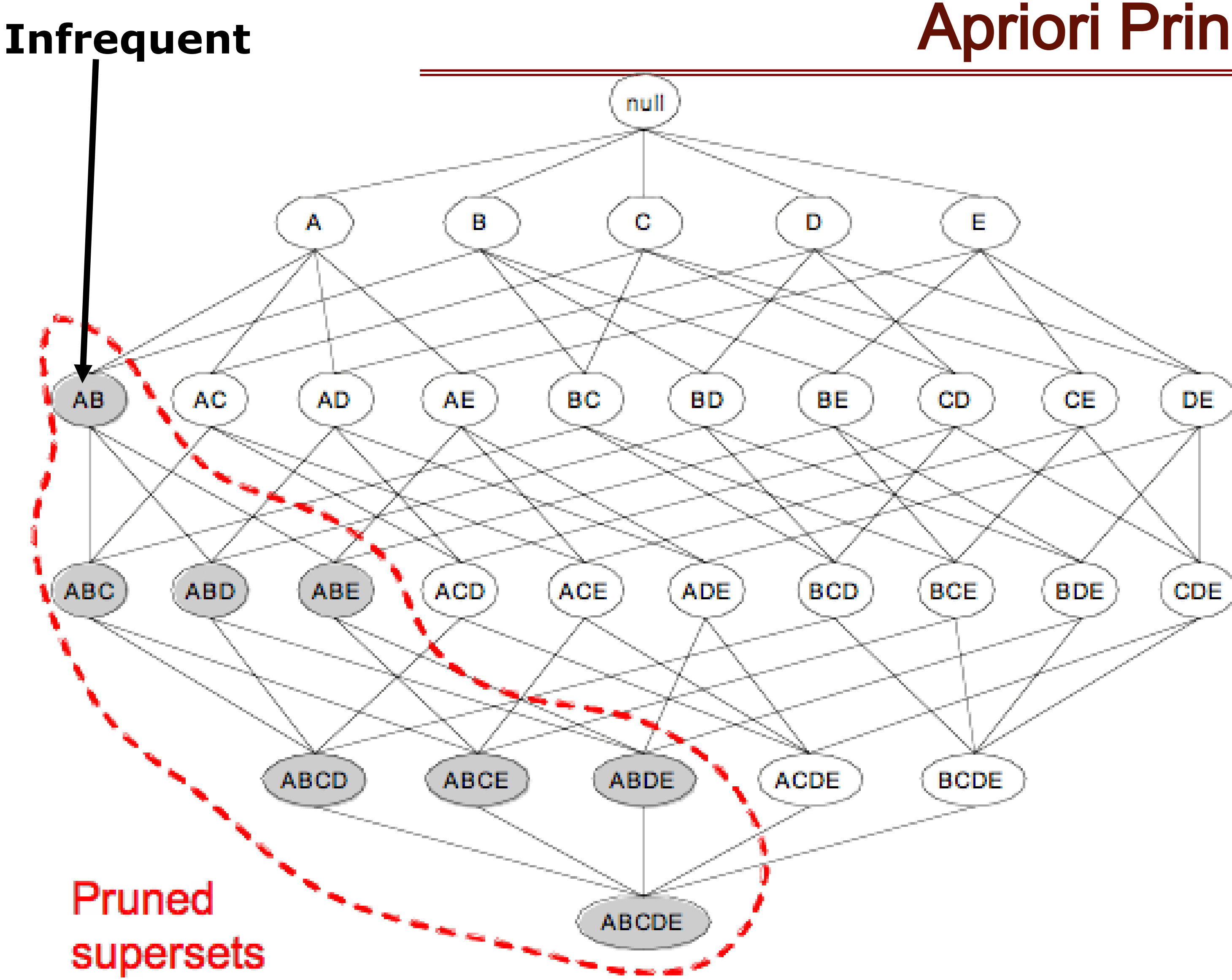
Apriori Principle

- Apriori principle:
  - If an itemset is frequent, then all of its subsets must also be frequent
- Apriori principle holds due to the following property of the support measure:

$$\forall X, Y : (X \subseteq Y) \Rightarrow s(X) \geq s(Y)$$

  - Support of an itemset never exceeds the support of its subsets
  - This is known as the anti-monotone property of support

Apriori Principle



Apriori Algorithm

- **Method:**
  - Let k=1
  - Generate frequent itemsets of length 1
  - Repeat until no new frequent itemsets are identified
    - Generate length (k+1) candidate itemsets from length k frequent itemsets
    - Prune candidate itemsets containing subsets of length k that are infrequent
    - Count the support of each candidate by scanning the DB
    - Eliminate candidates that are infrequent, leaving only those that are frequent

Minimum Support = 3

Apriori Algorith

Items (1-itemsets)

Item	Count
Bread	4
Coke	2
Milk	4
Beer	3
Diaper	4
Eggs	1

Pairs (2-itemsets)

(No need to generate candidates involving Coke or Eggs)

Itemset	Count
{Bread,Milk}	3
{Bread,Beer}	2
{Bread,Diaper}	3
{Milk,Beer}	2
{Milk,Diaper}	3
{Beer,Diaper}	3

Minimum Support = 3

Apriori Algorithm

Items (1-itemsets)

Item	Count
Bread	4
Coke	2
Milk	4
Beer	3
Diaper	4
Eggs	1

Pairs (2-itemsets)

(No need to generate candidates involving Coke or Eggs)

Itemset	Count
{Bread,Milk}	3
{Bread,Beer}	2
{Bread,Diaper}	3
{Milk,Beer}	2
{Milk,Diaper}	3
{Beer,Diaper}	3

Triplets (3-itemsets)

Itemset	Count
{Bread,Milk,Diaper}	3

If every subset is considered,

$$^6C_1 + {}^6C_2 + {}^6C_3 = 41$$

With support-based pruning,

$$6 + 6 + 1 = 13$$

Apriori Algorithm

- **Method:**
  - Let k=1
  - Generate frequent itemsets of length 1
  - Repeat until no new frequent itemsets are identified
    - Generate length (k+1) candidate itemsets from length k frequent itemsets
    - Prune candidate itemsets containing subsets of length k that are infrequent
    - Count the support of each candidate by scanning the DB
    - Eliminate candidates that are infrequent, leaving only those that are frequent

K+1 Set Generation - I

- Generate length (k+1) candidate itemsets from length k frequent itemsets
  - Fk X F1

Item	Count
Bread	4
Coke	2
Milk	4
Beer	3
Diaper	4
Eggs	1

{Bread}
{Beer}
{Diaper}
{Milk}
{Coke}
{Eggs}
...

{Bread, Milk}
{Bread, Beer}
{Bread, Diaper}
{Milk, Bread}
{Milk, Beer}
{Milk, Diaper}
...

?

?

K+1 Set Generation - I

- Generate length (k+1) candidate itemsets from length k frequent itemsets
  - Fk X F1

Item	Count
Bread	4
Coke	2
Milk	4
Beer	3
Diaper	4
Eggs	1

Itemset	Count
{Bread,Milk}	3
{Bread,Beer}	2
{Bread,Diaper}	3
{Milk,Beer}	2
{Milk,Diaper}	3
{Beer,Diaper}	3

{Bread}
{Beer}
{Diaper}
{Milk}
{Coke}
{Eggs}
...

{Bread, Milk}
{Bread, Beer}
{Bread, Diaper}
{Milk, Bread}
{Milk, Beer}
{Milk, Diaper}
...

K+1 Set Generation - I

- Generate length (k+1) candidate itemsets from length k frequent itemsets
  - Fk X F1

Item	Count
Bread	4
Coke	2
Milk	4
Beer	3
Diaper	4
Eggs	1

Itemset	Count
{Bread,Milk}	3
{Bread,Beer}	2
{Bread,Diaper}	3
{Milk,Beer}	2
{Milk,Diaper}	3
{Beer,Diaper}	3

{Bread}
{Beer}
{Diaper}
{Milk}
{Coke}
{Eggs}
...

{Bread, Milk}
~~{Bread, Beer}~~
{Bread, Diaper}
{Milk, Bread}
~~{Milk, Beer}~~
{Milk, Diaper}
...

K+1 Set Generation - I

- Generate length (k+1) candidate itemsets from length k frequent itemsets

Fk	X	F1	=	Fk+1
{Bread, Milk}		{Bread}		{Bread, Milk, Beer}
{Bread, Diaper}		{Beer}		{Bread, Milk , Diaper}
{Milk, Bread}		{Diaper}		{Bread, Diaper, Beer}
{Milk, Diaper}		{Milk}		…
…		…		

Item	Count
Bread	4
Coke	2
Milk	4
Beer	3
Diaper	4
Eggs	1

Itemset	Count
{Bread,Milk}	3
{Bread,Beer}	2
{Bread,Diaper}	3
{Milk,Beer}	2
{Milk,Diaper}	3
{Beer,Diaper}	3

Min Support = 3

K+1 Set Generation - I

- Generate length (k+1) candidate itemsets from length k frequent itemsets

Fk X F1 = Fk+1

Fk	F1	Fk+1
{Bread, Milk}	{Bread}	~~{Bread, Milk, Beer}~~
{Bread, Diaper}	{Beer}	{Bread, Milk , Diaper}
{Milk, Bread}	{Diaper}	~~{Bread, Diaper, Beer}~~
{Milk, Diaper}	{Milk}	…
…	…	

Itemset	Count
{Bread,Milk}	3
{Bread,Beer}	2
{Bread,Diaper}	3
{Milk,Beer}	2
{Milk,Diaper}	3
{Beer,Diaper}	3

Min Support = 3

K+1 Set Generation - II

- Generate length (k+1) candidate itemsets from length k frequent itemsets
  - Fk X Fk
  - Merge a pair of k-itemsets if the first k-1 items are identical

{Bread, Milk}
{Bread, Beer}
{Diaper, Bread}
{Diaper, Milk}
...

{*Bread, Milk,* Beer}

{*Diaper, Milk*, Bread}
...

Itemset	Count
{Bread,Milk}	3
{Bread,Beer}	2
{Bread,Diaper}	3
{Milk,Beer}	2
{Milk,Diaper}	3
{Beer,Diaper}	3

Min Support = 3

K+1 Set Generation - II

- Generate length (k+1) candidate itemsets from length k frequent itemsets
  - Fk X Fk
  - Merge a pair of k-itemsets if the first k-1 items are identical

{Bread, Milk}
{Bread, Beer}
{Diaper, Bread}
{Diaper, Milk}
...

~~**{*Bread, Milk,* Beer}**~~

{*Diaper, Milk*, Bread}
...

Itemset	Count
{Bread,Milk}	3
{Bread,Beer}	2
{Bread,Diaper}	3
{Milk,Beer}	2
{Milk,Diaper}	3
{Beer,Diaper}	3

Min Support = 3

Apriori Algorithm

- Level-wise algorithm
- Generate and test strategy
- Number of iterations $k_{max}+1$
- K_{max} is the max size of the frequent itemset

Apriori Algorithm

- **Method:**
  - Let k=1
  - Generate frequent itemsets of length 1
  - Repeat until no new frequent itemsets are identified
    - Generate length (k+1) candidate itemsets from length k frequent itemsets
    - Prune candidate itemsets containing subsets of length k that are infrequent
    - Count the support of each candidate by scanning the DB
    - Eliminate candidates that are infrequent, leaving only those that are frequent

Support Counting

- Frequency of each candidate itemset
- Compare each transaction against each candidate, update the counts

Support Counting

K-1 Iteration's itemsets

Itemset	Count
{Bread,Milk}	3
{Bread,Beer}	2
{Bread,Diaper}	3
{Milk,Beer}	2
{Milk,Diaper}	3
{Beer,Diaper}	3

K Iteration's candidate itemsets

{Bread, Milk}
{Bread, Beer}
{Diaper, Bread}
{Diaper, Milk}
...

{Bread, Milk, Beer}

{Diaper, Bread, Milk}
...

Support Counting

K-1 Iteration's itemsets

K Iteration's candidate itemsets

Itemset	Count
{Bread,Milk}	3
{Bread,Beer}	2
{Bread,Diaper}	3
{Milk,Beer}	2
{Milk,Diaper}	3
{Beer,Diaper}	3

{Bread, Milk}
{Bread, Beer}
{Diaper, Bread}
{Diaper, Milk}
...

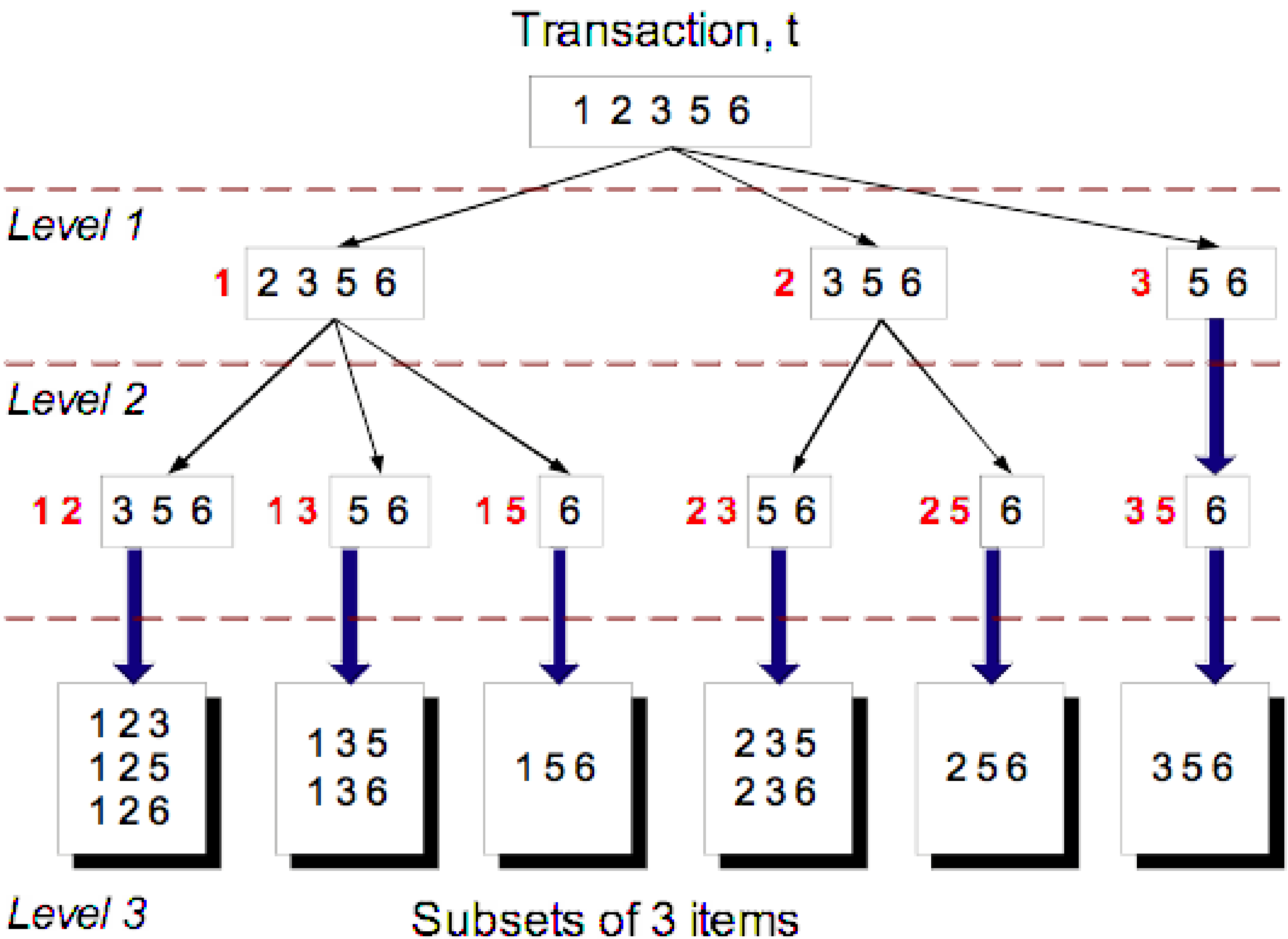
{Bread, Milk, Beer}

{Diaper, Bread, Milk}
...

Transactions

TID	Items
1	Bread, Milk
2	Bread, Diaper, Beer, Eggs
3	Milk, Diaper, Beer, Coke
4	Bread, Milk, Diaper, Beer
5	Bread, Milk, Diaper, Coke

Given a transaction t, what are the possible subsets of size 3?

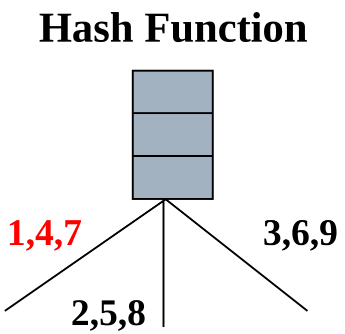


Reducing Number of Comparisons

- Candidate counting:
  - Scan the database of transactions to determine the support of each candidate itemset
  - To reduce the number of comparisons, store the candidates in a hash structure
  - Store transactions in the hash as well
  - Instead of matching each transaction against every candidate, match it against candidates contained in the hashed buckets

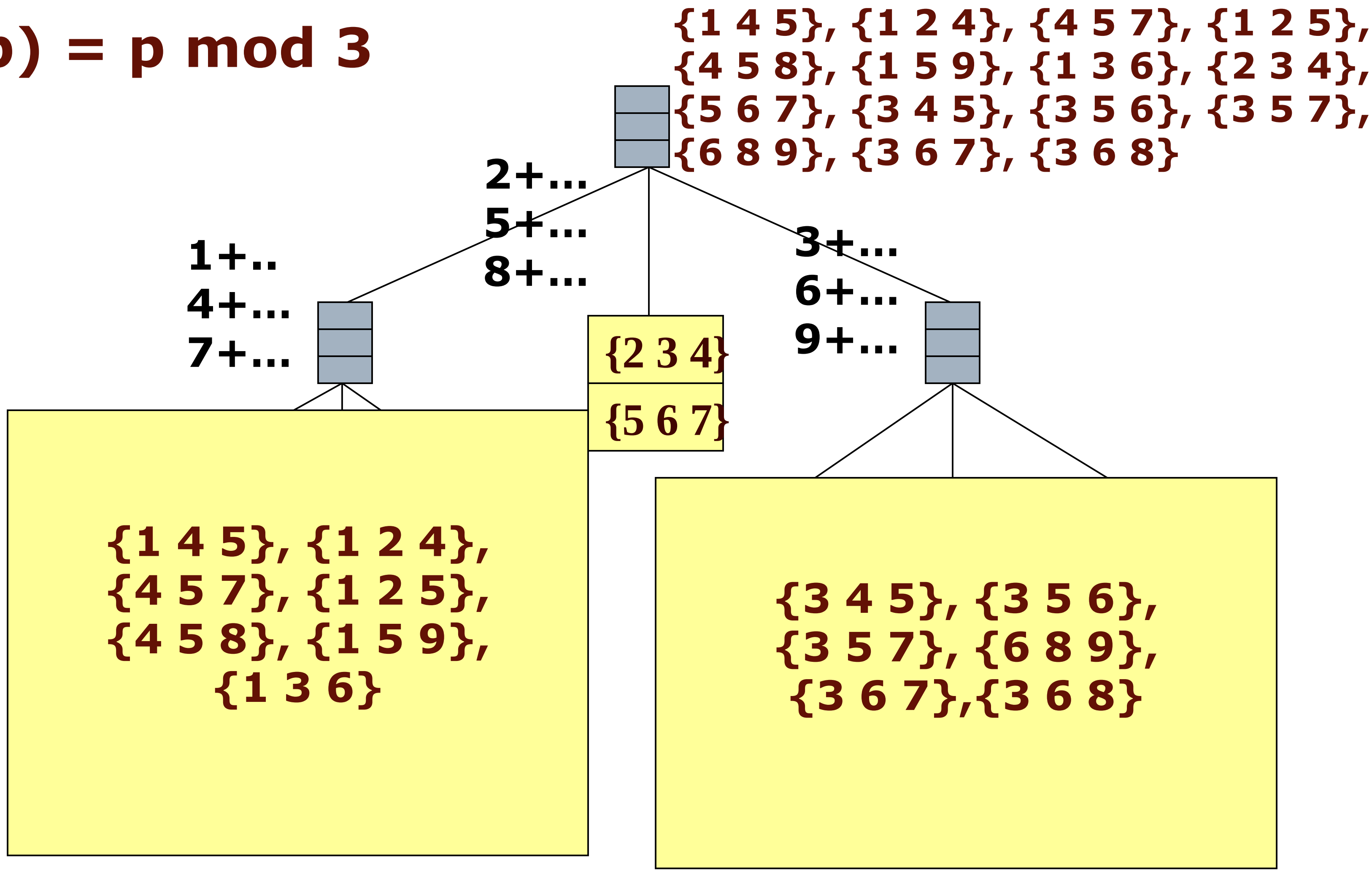
Candidate Itemsets Hash Tree

- Suppose you have 9 items, 15 candidate itemsets of length 3:
  - {1 4 5}, {1 2 4}, {4 5 7}, {1 2 5}, {4 5 8}, {1 5 9}, {1 3 6}, {2 3 4}, {5 6 7}, {3 4 5}, {3 5 6}, {3 5 7}, {6 8 9}, {3 6 7}, {3 6 8}
- Hash function
  - Max leaf size: max number of itemsets stored in a leaf node (if number of candidate itemsets exceeds max leaf size, split the node)
  - H(p) = p mod 3
  - Sort items in the itemsets



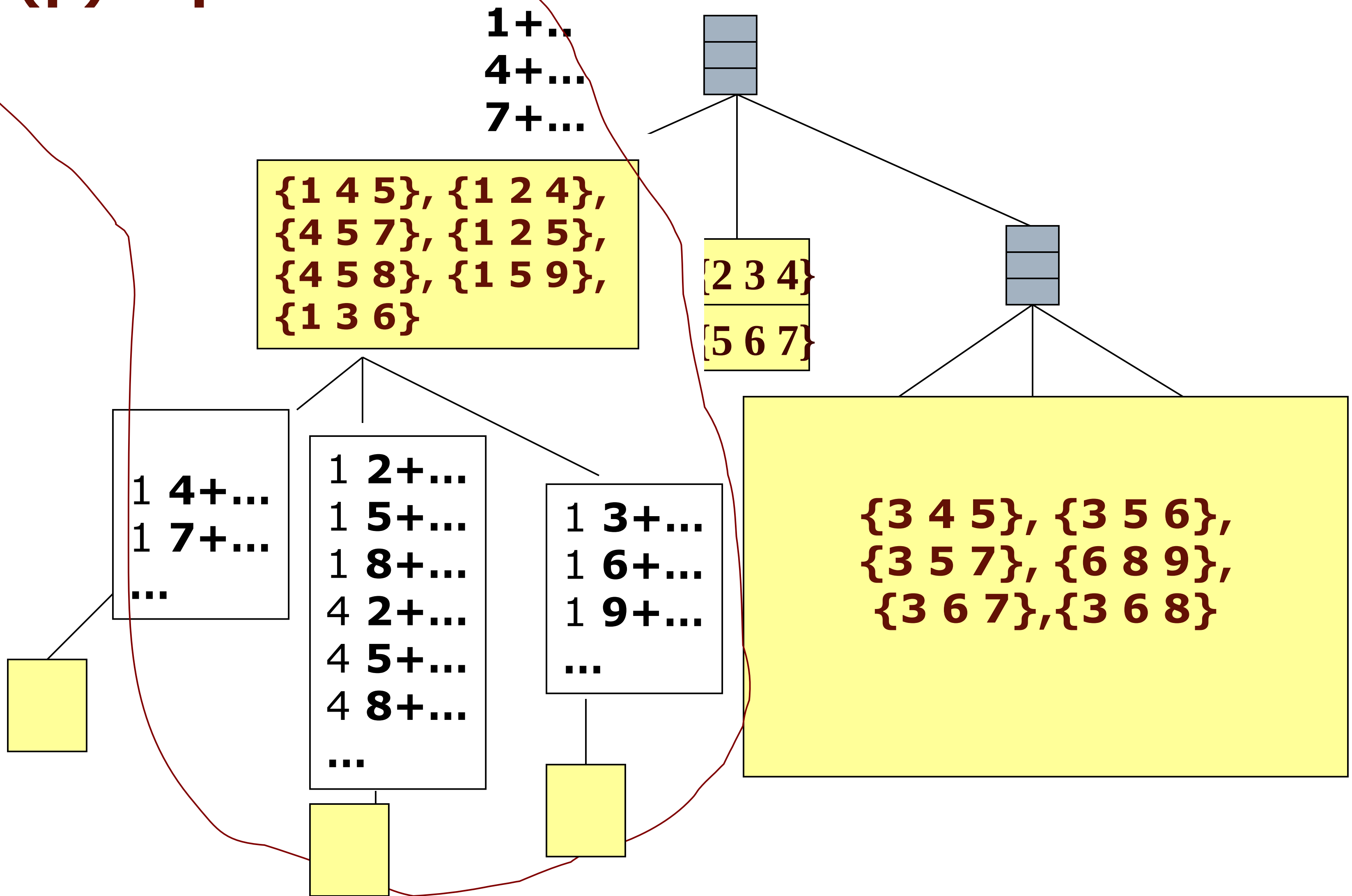
Candidate Itemsets Hash Tree

H(p) = p mod 3



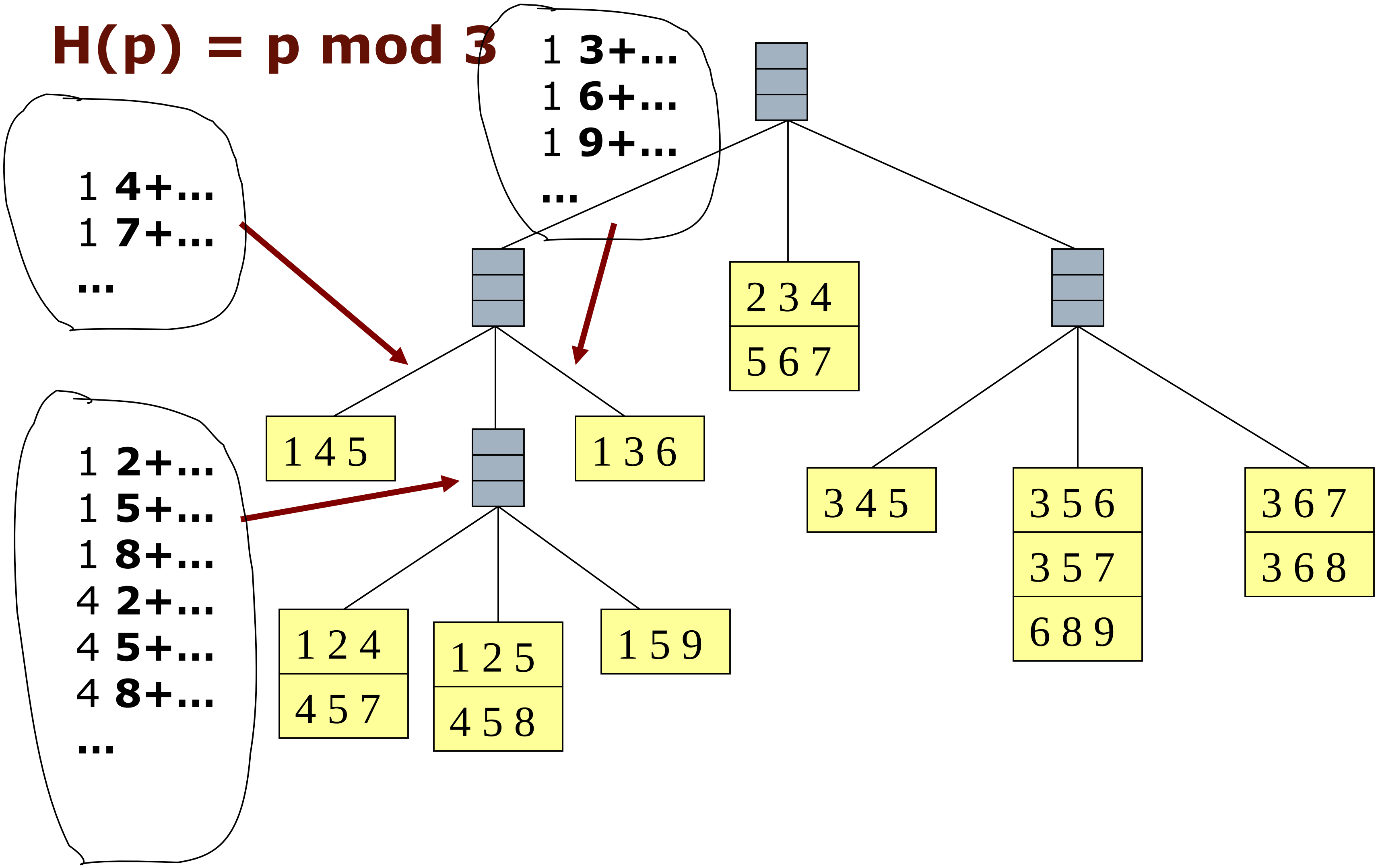
Candidate Itemsets Hash Tree

H(p) = p mod 3

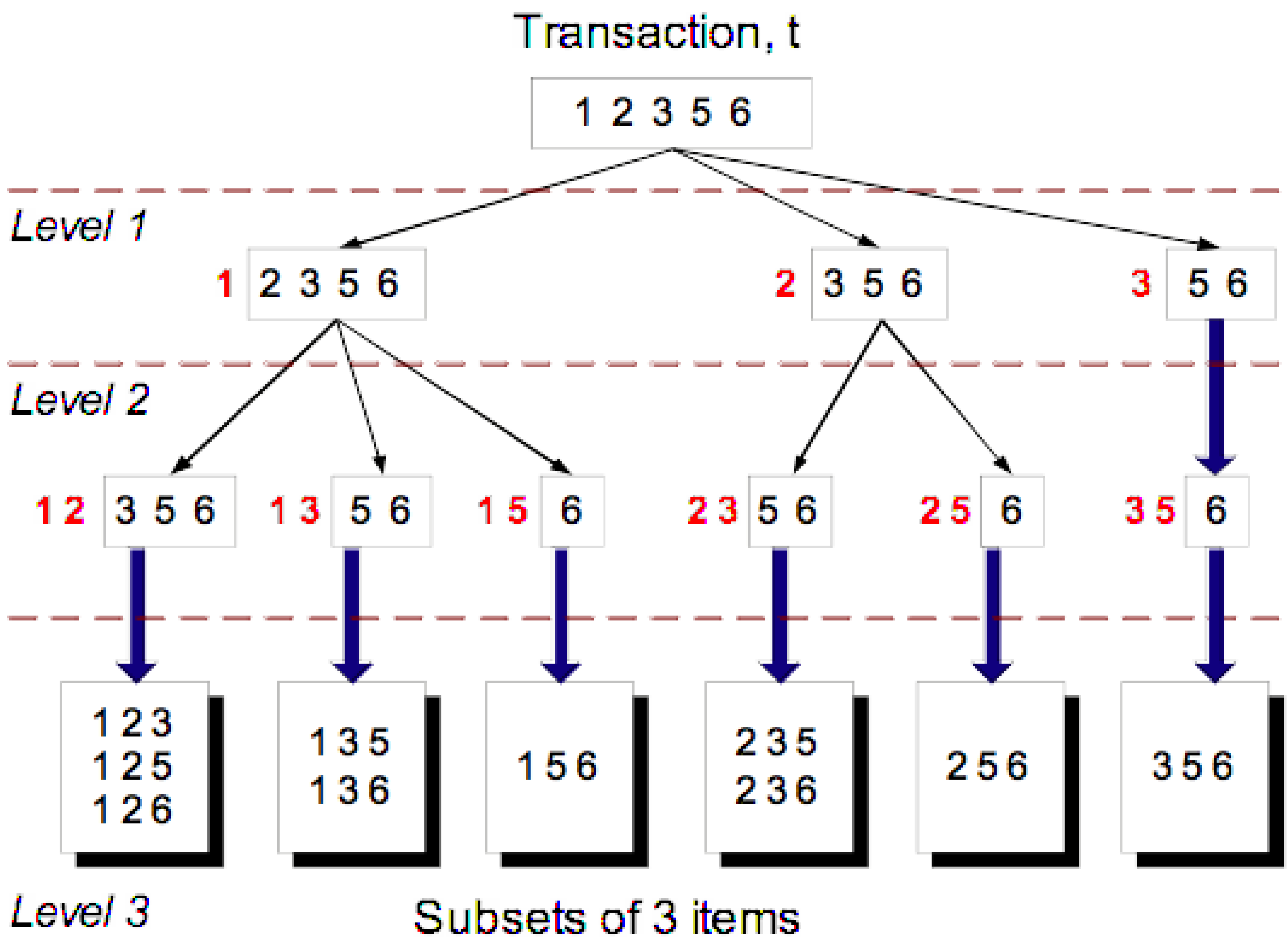


Candidate Itemsets Hash Tree

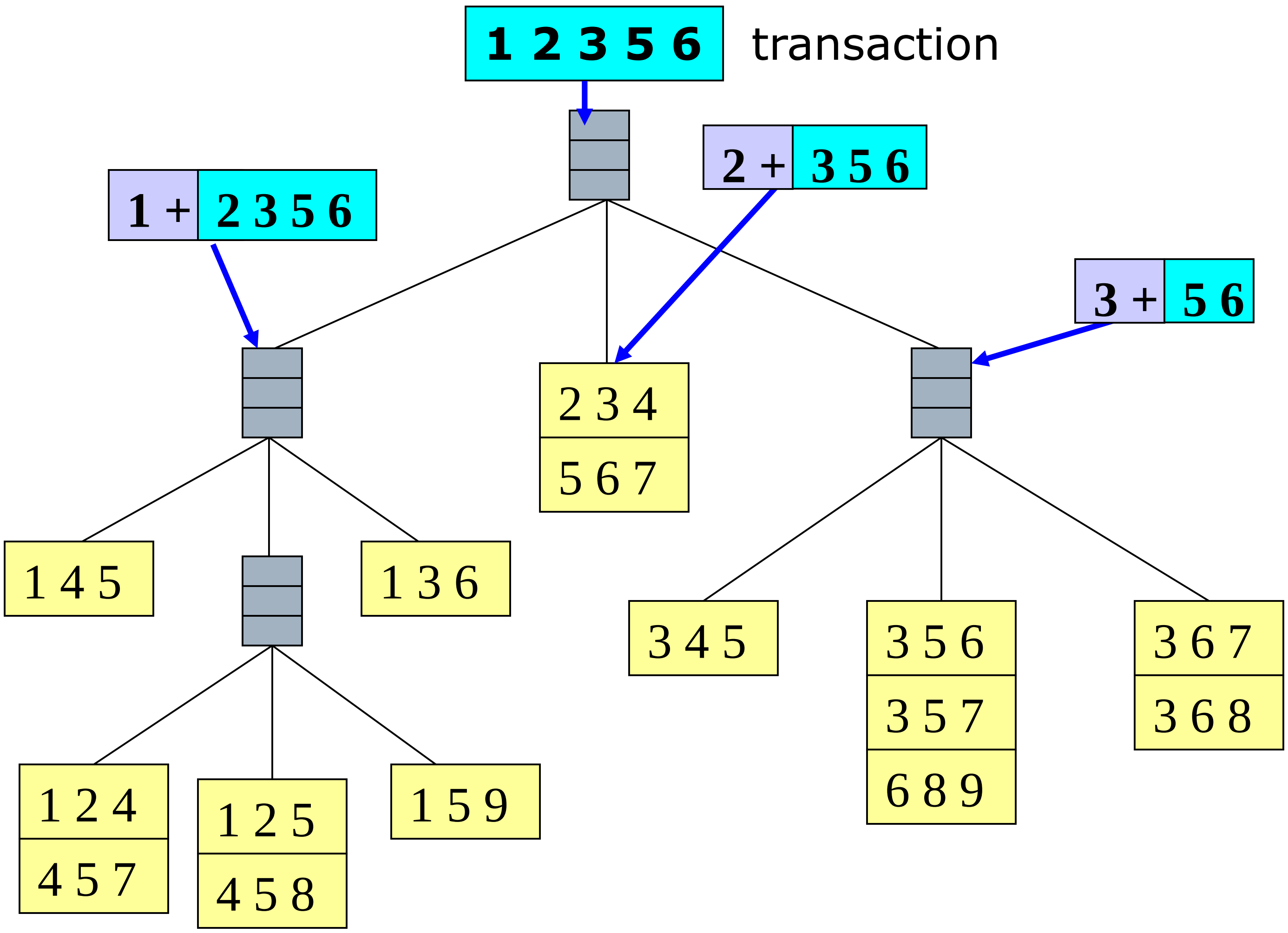
$H(p) = p \bmod 3$



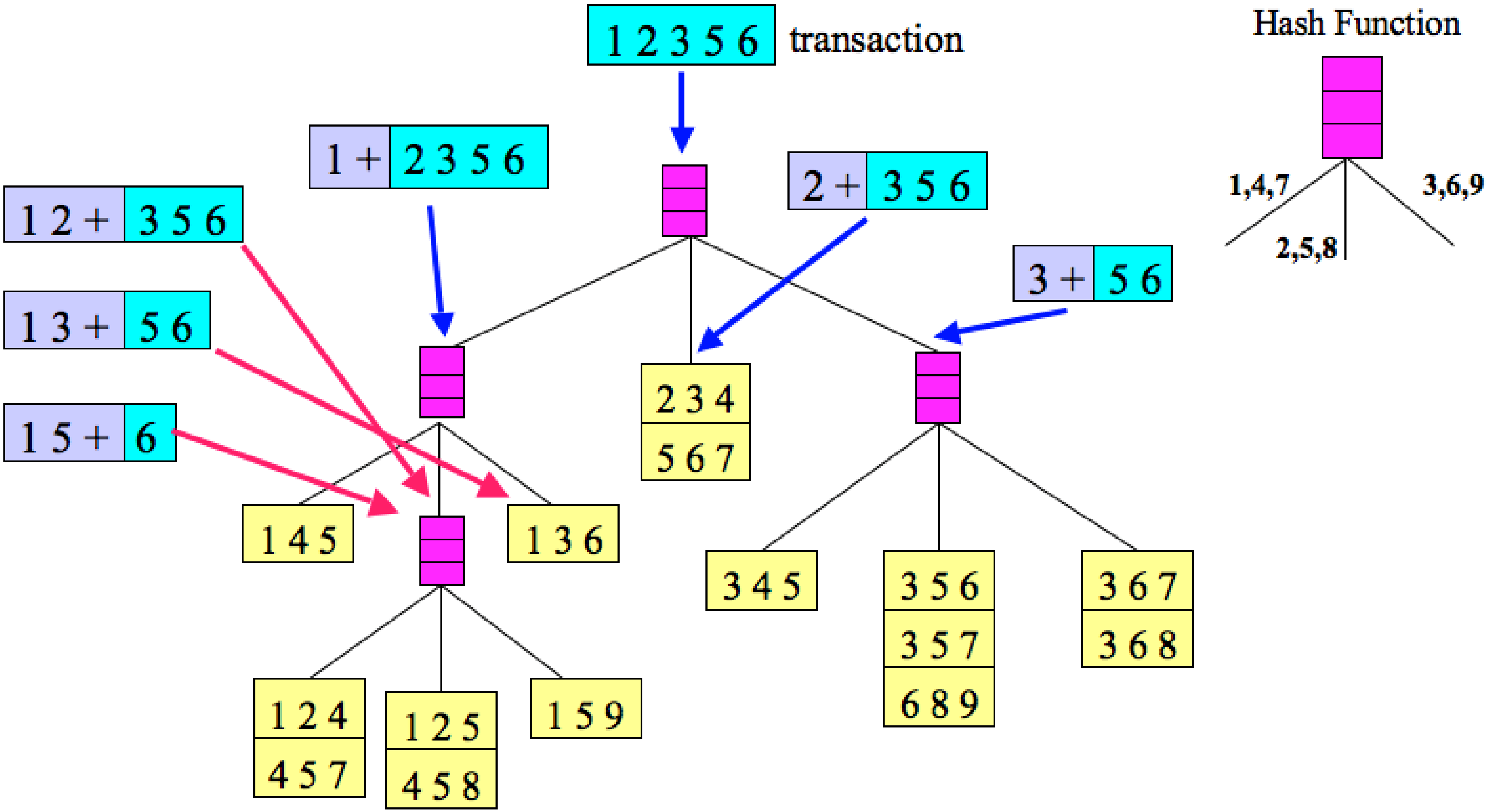
Enumerating Itemsets in Transaction



Itemsets from Transaction in Candidate Hash Tree



Itemsets from Transaction in Candidate Hash Tree



Count Update

Increment counts for matching candidate Itemsets: {1,3,6}, {1,2,5} {3,5,6}