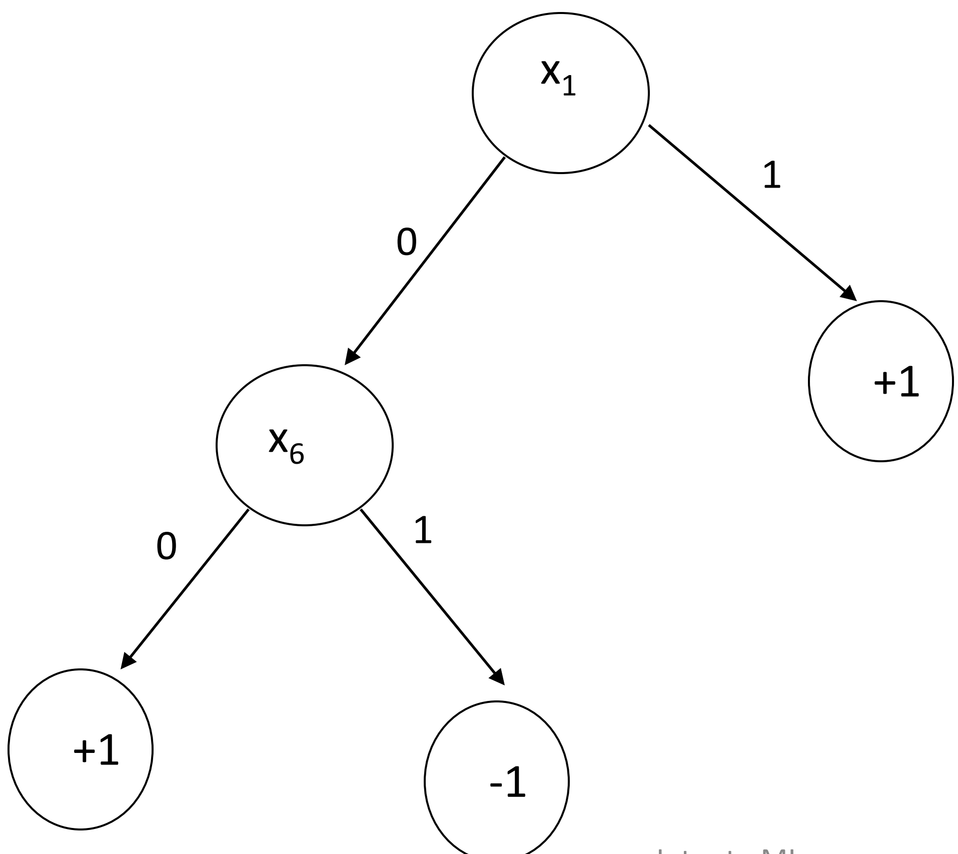
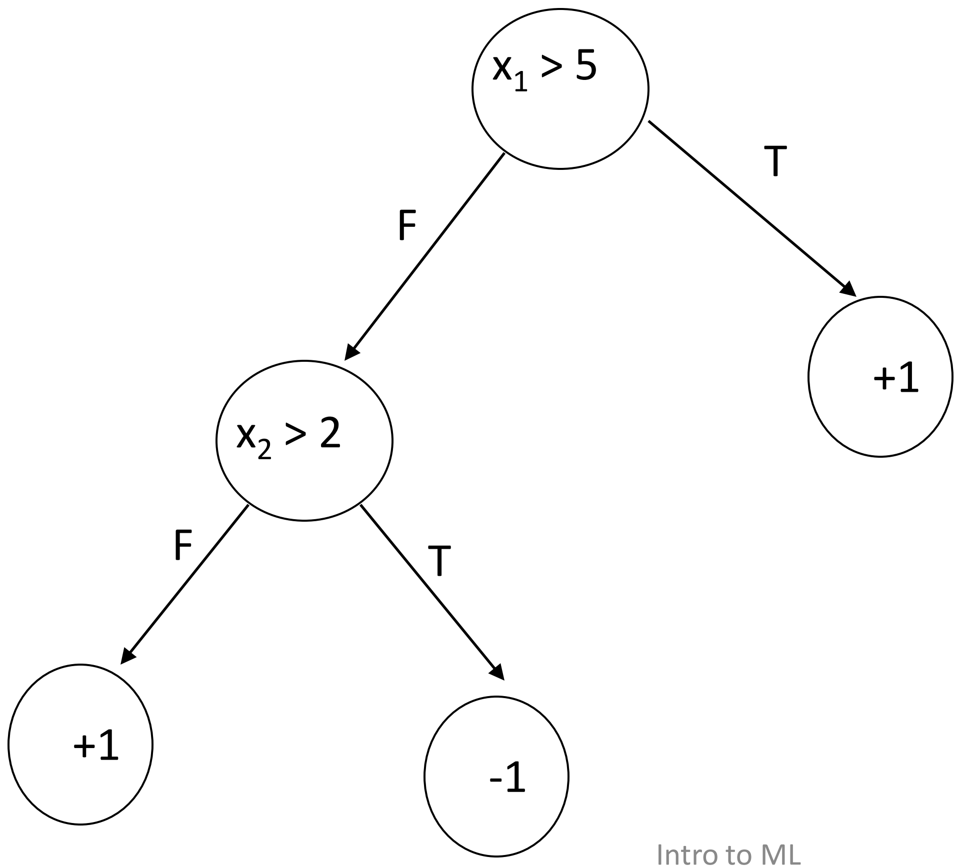
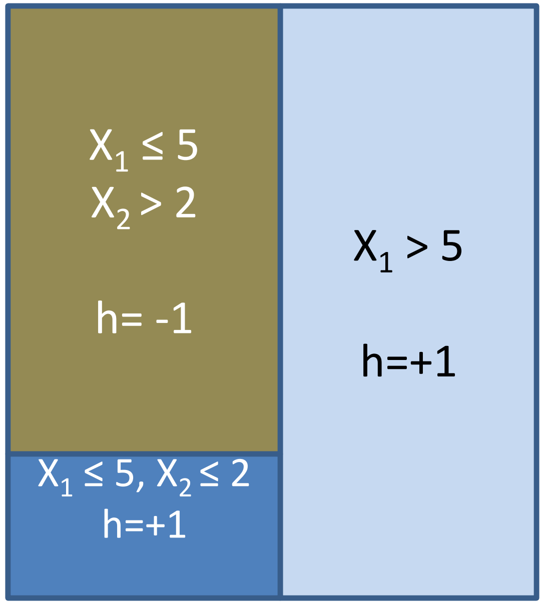
Decision Trees and Ensemble Methods

Decision Trees - Boolean



Decision Trees Continuous





Decision Tree Pruning

Problem Statement

- We like to output small decision tree
  - Model Selection
- The building is done until zero training error
- Option I : Stop Early
  - Small decrease in index function
  - Cons: may miss structure
- Option 2: Prune after building.

Pruning

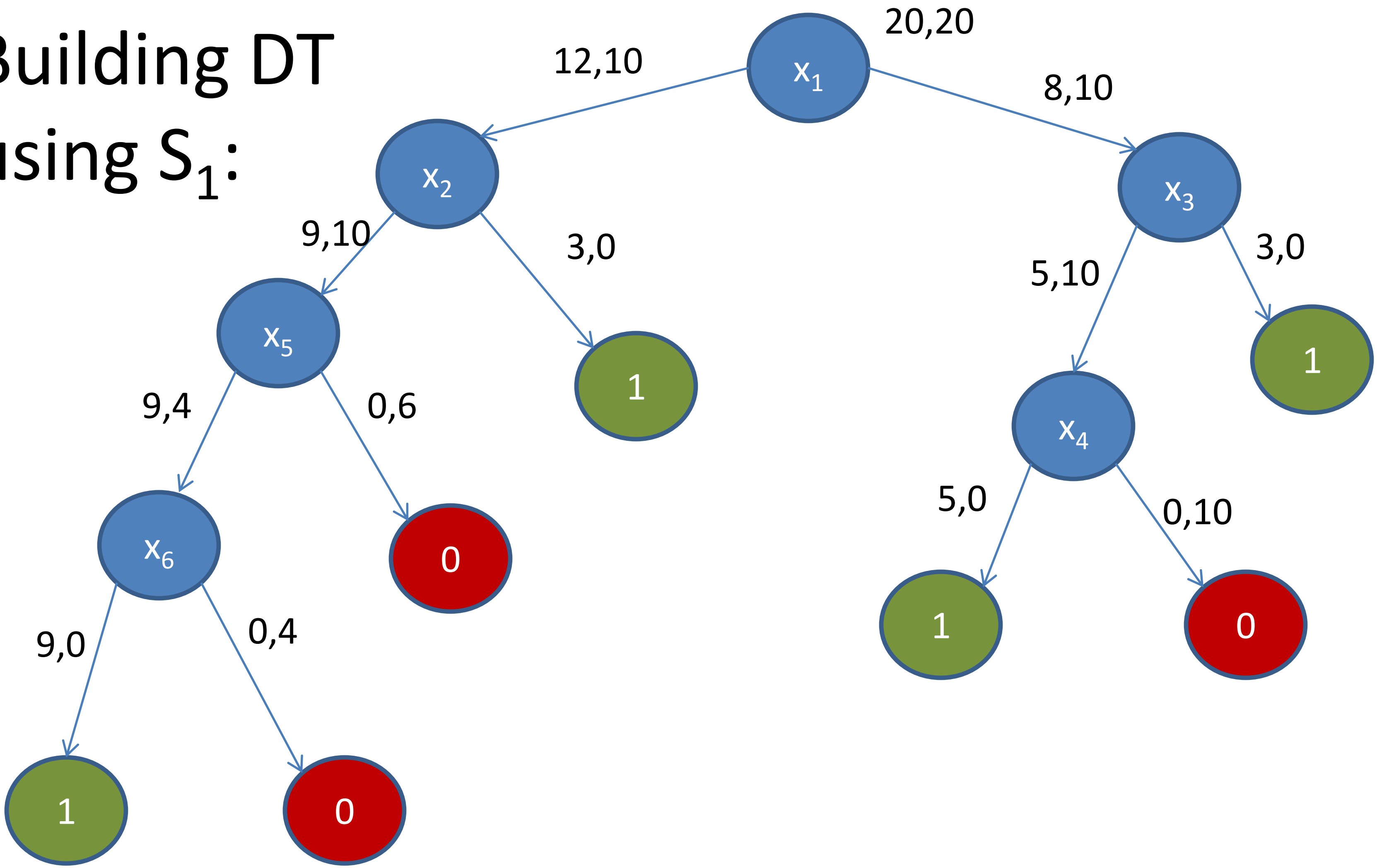
- Input: tree T
- Sample: S
- Output: Tree T'
- Basic Pruning: T' is a sub-tree of T
  - Can only replace inner nodes by leaves
- More advanced:
  - Replace an inner node by one of its children

Reduced Error Pruning

- Split the sample to two part S_1 and S_2
- Use S_1 to build a tree.
- Use S_2 to decide when to prune.
- Process every inner node *v*
  - After all its children have been processed
  - Compute the observed error of T_v and possible *leaf(v)*
  - If *leaf(v)* has less errors replace T_v by *leaf(v)*
- *Alternative: require the difference to be statistically significant*
  - *Can be theoretically analyzed*

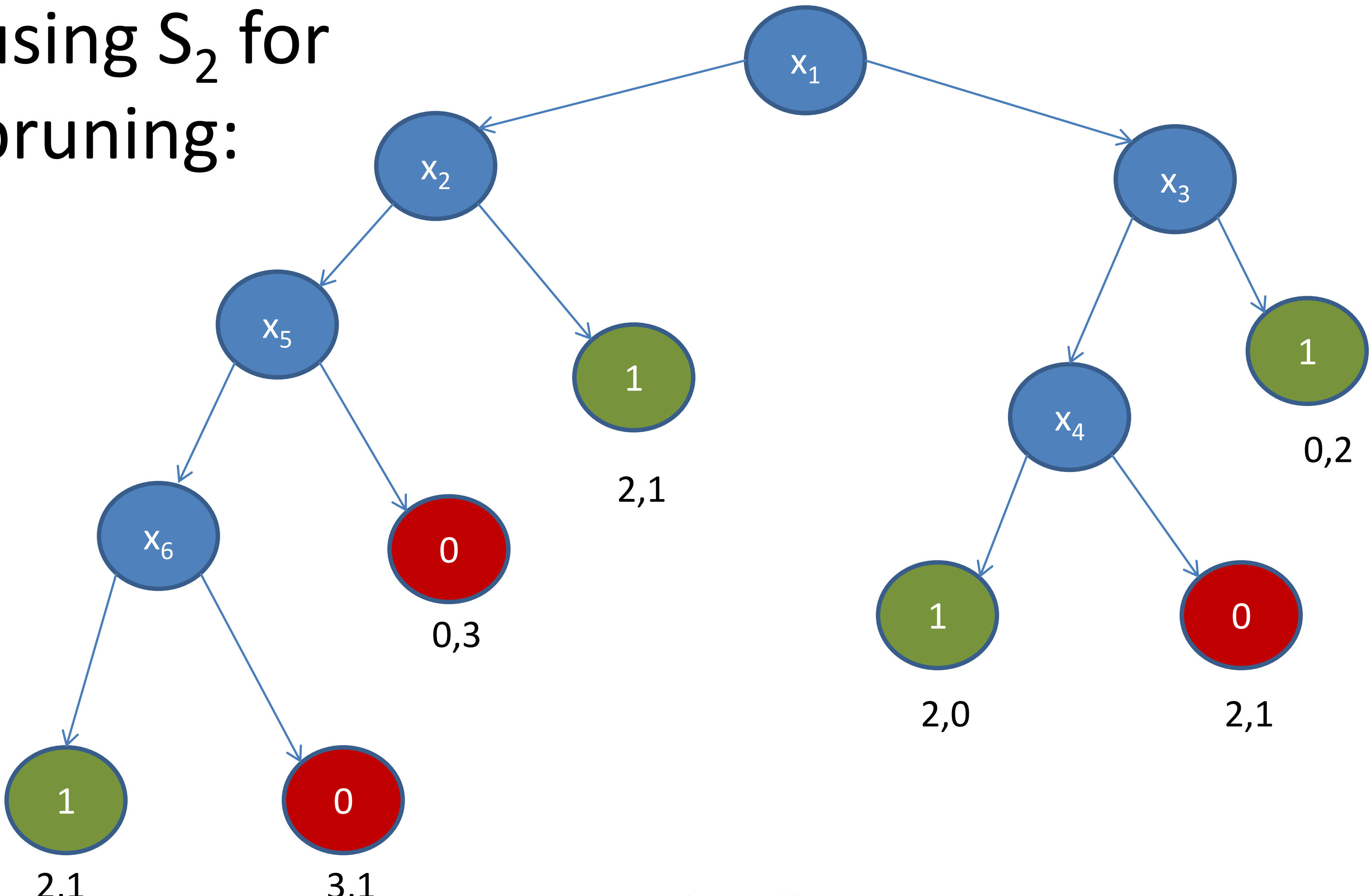
Reduced Error Pruning: Example

Building DT using S_1:

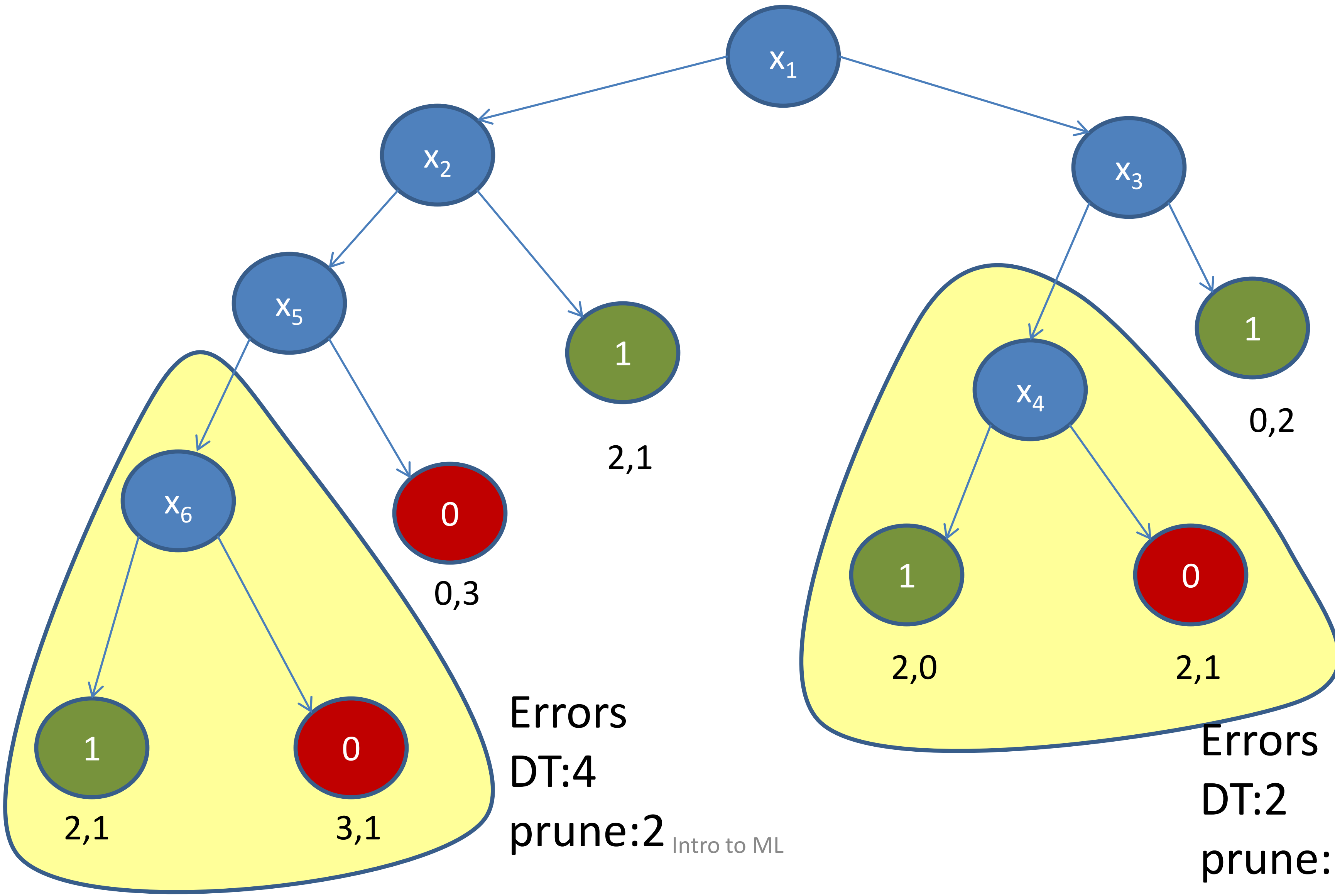


Reduced Error Pruning: Example

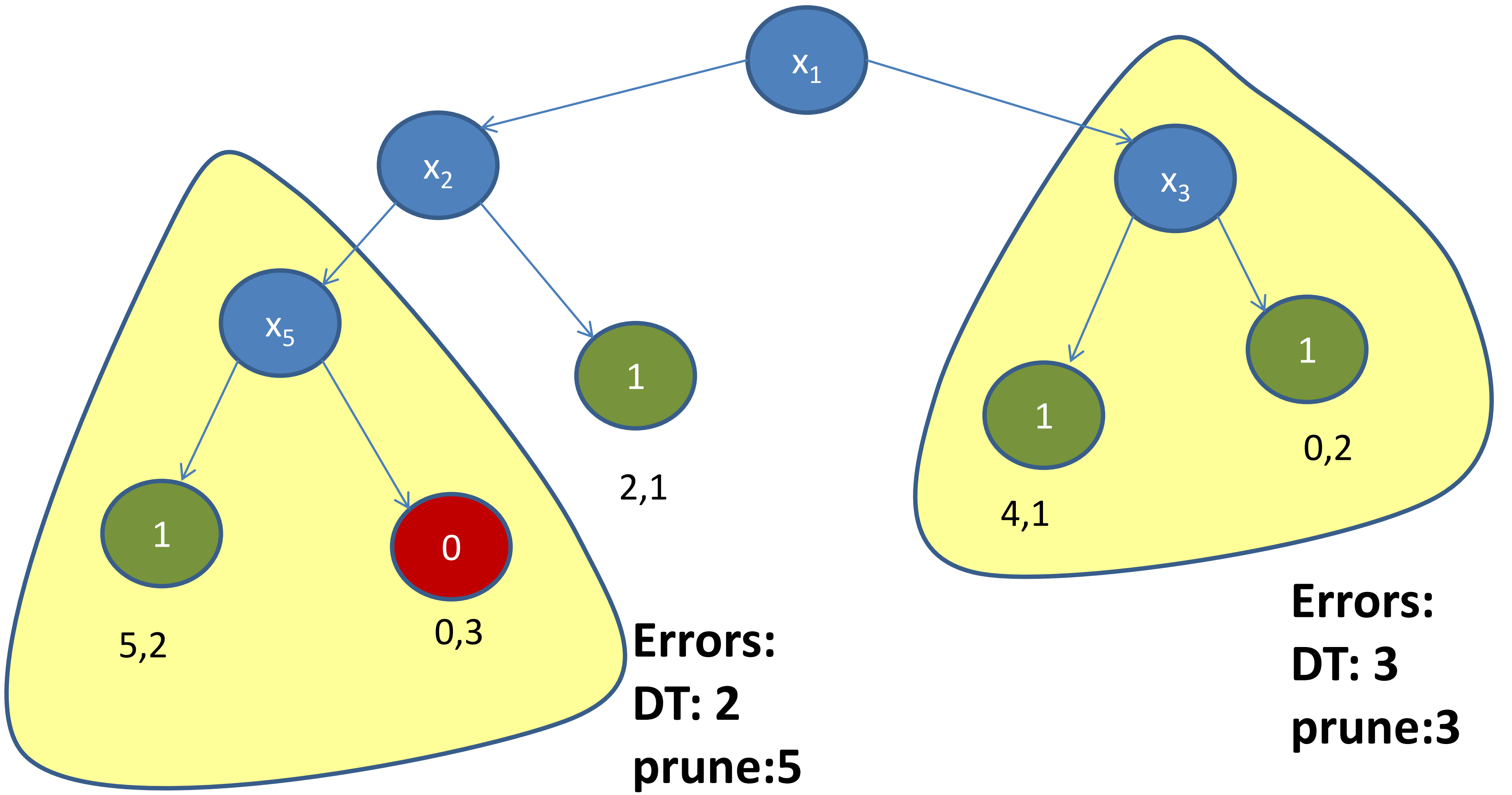
using S_2 for pruning:



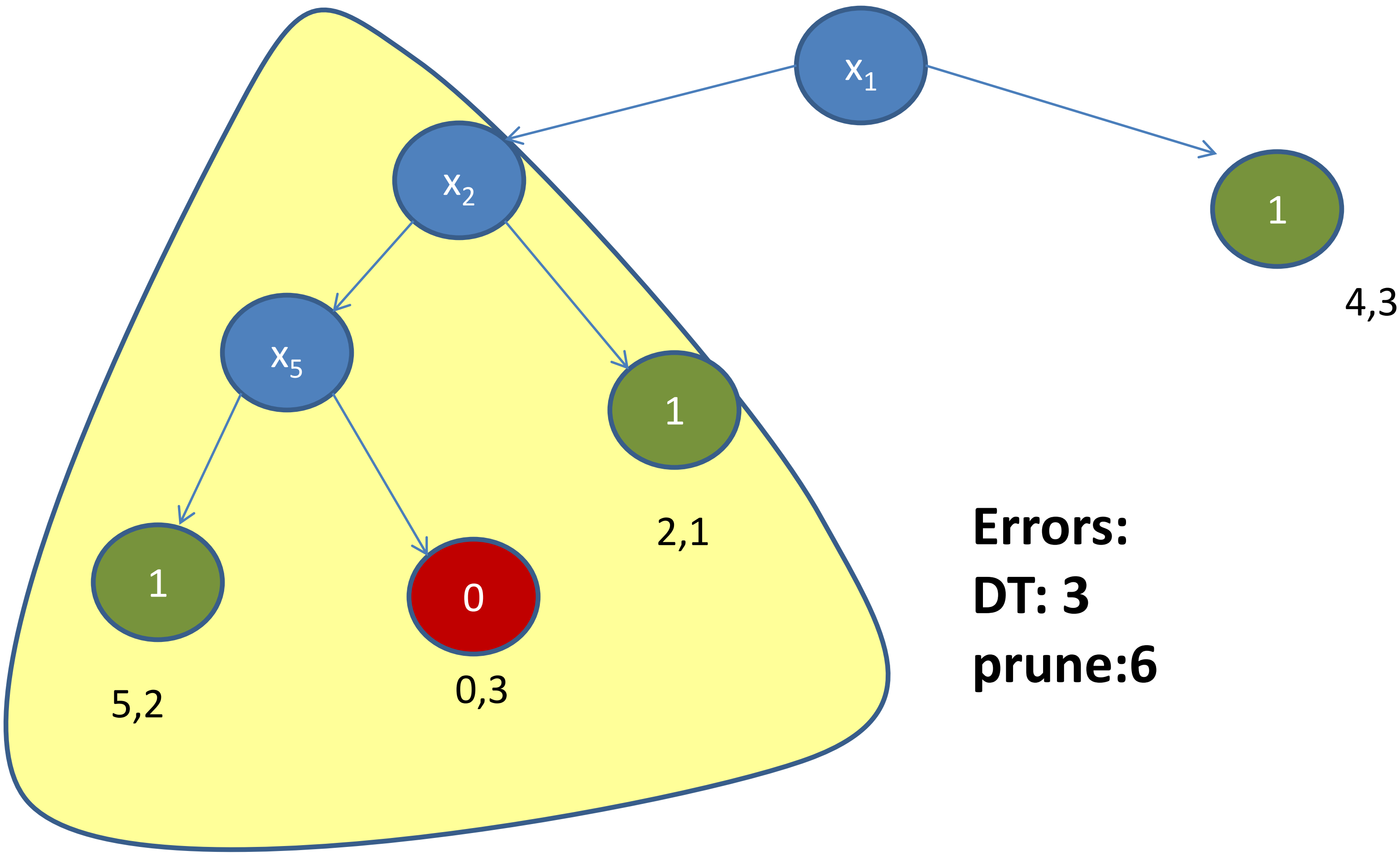
Reduced Error Pruning: Example



Reduced Error Pruning: Example

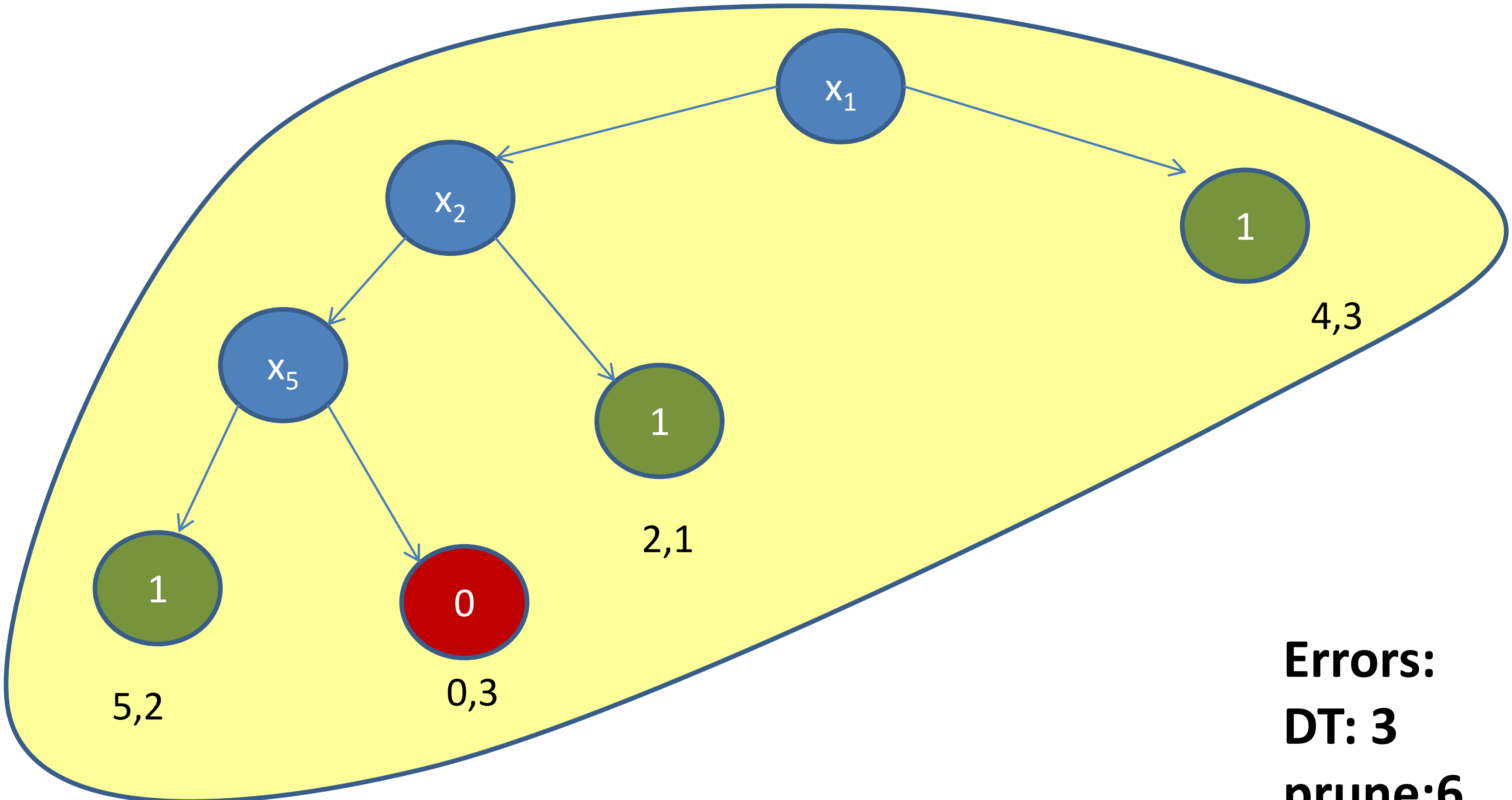


Reduced Error Pruning: Example



Errors:
DT: 3
prune:6

Reduced Error Pruning: Example



Errors:
DT: 3
prune:6

Pruning: Model Selection

- Generate DT for each pruning size
  - compute the minimal error pruning
  - At most *m* different decision-trees
- Select between the prunings
  - Cross Validation
  - Structural Risk Minimization
    - Any other index method

Finding the minimum pruning

- Procedure Compute
- Inputs:
  - k : number of errors
  - T : tree
  - S : sample
- Output:
  - P* : pruned tree
  - Size* : size of P

- Compute(k,T,S,P*,size*)
- IF *IsLeaf(T)= TRUE*
  - IF $Errors(T) \leq k$
    - THEN *size*=1*
    - ELSE *size* =* ∞
  - *P*=T*; return;
- IF $Errors(root(T)) \leq k$
  - *Size*=1*; *P*=root(T)*; return;

Procedure compute

- For i = 0 to k DO
  - Call Compute(i, T[0], S_0, $P_{i,0}$, $size_{i,0}$)
  - Call Compute(k-i, T[1], S_1, $P_{i,1}$, $size_{i,1}$)
- Size* = minimum {$size_{i,0}$ + $size_{i,1}$ +1}
- i* = arg min {$size_{i,0}$ + $size_{i,1}$ +1}
- P* = MakeTree(root(T),$P_{i*,0}$, $P_{i*,1}$}
- Return
- What is the time complexity?

Cross Validation

- Split the sample S_1 and S_2
- Build a tree using S_1
- Compute the candidate prunings
  - $P_1, \ldots, P_m$
- Select using S_2
  - $T^* = \mathrm{Arg\ min\ error}(P_i, S_2)$
- Output the tree T*
  - Has the smallest error on S_2

SRM

- Build a Tree *T* using *S*
- Compute the candidate prunings
  - $P_1, \ldots, P_m$
  - k_d the size of the pruning with *d* errors
- Select using the **SRM** formula

$$\min_d \{error(S, T_d) + \sqrt{\frac{k_d}{m}}\}$$

Drawbacks

- Running time
  - Since |T| = O(m)
  - Running time $O(m^2)$
  - Many passes over the data
- Significant drawback for large data sets

More on Pruning

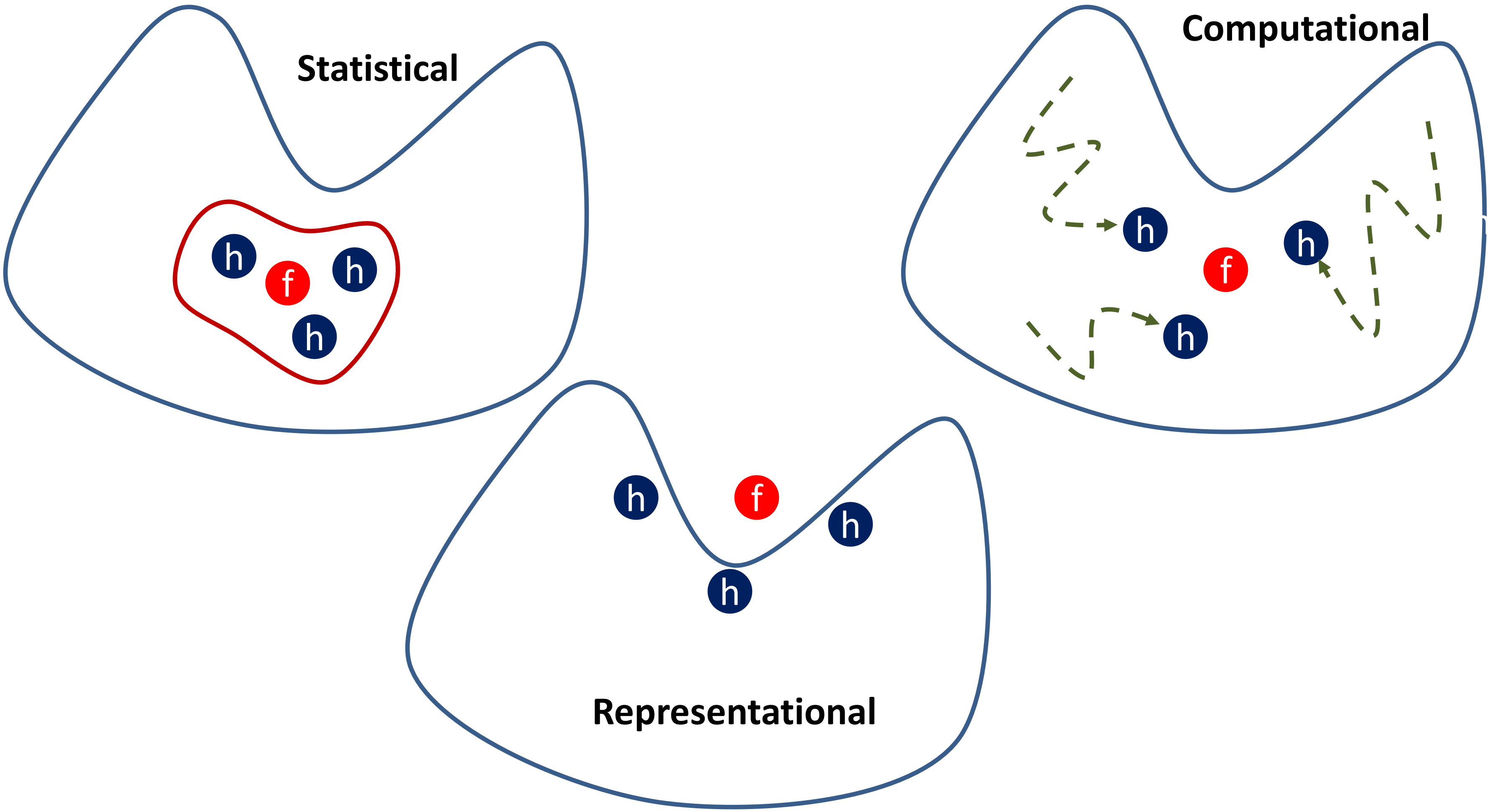
- Considered only leaf replacement
  - Substitute a sub-tree by a leaf
- Other popular alternatives
  - Replace a node by one of its children.
- Reduce error pruning
  - Conceptually similar
- Model selection

Ensemble Methods

Ensemble Methods

- High level idea
  - Generate multiple hypotheses
  - Combine them to a single classifier
- Two important questions
  - How do we generate multiple hypotheses
    - we have only one sample
  - How do we combine the multiple hypotheses
    - Majority, AdaBoost, …

Rational for Ensemble Methods



Boosting

- Boosting is actually an ensemble method
- Generating different hypotheses:
  - By changing the sample distribution
- Combining hypotheses
  - weighted linear predictor
    - Weights determine when hypo. is selected.

Bagging

- Input: a single learning algorithm $\mathcal{A}$
- How do we generate different Hypotheses
  - sampling
    - with replacement (maintains the statistics)
  - Formally, given a sample S
  - Sub sample $S_1, \dots, S_k$
  - Run $\mathcal{A}$ on S_i to generate h_i
- Combining: Simple majority

Bagging rational: Bias versus Var

- Why is one hypothesis worse than many ?!
- Expected error of h_i
  - identical to all h_i
  - worse than training on all sample
    - smaller sample
  - BIAS

- Variance of the error
  - single hypothesis - fluctuates considerably
  - majority of many - much more stable
  - More stable → better generalization
    - the training error better reflects the true error

Stacking

- Input:
  - Sample S
  - k algorithms A_i
  - combing algo C
- Run A_i on S generate h_i
- Given h_1 , … , h_k
  - generate new sample
  - $(x,y) \rightarrow (h_1(x), \ldots, h_k(x), y)$
  - Run C to generate H
- Output H

- What can be A_i?
- What can be C?

- Bagging:
  - A_i sub-samples
  - C is a majority

- AdaBoost
  - A_i weak hypo time i
  - C weighted majority

Random Forest: motivation

- Decision Trees Bias
  - Decision tree creation is very noisy
  - Depends on particular sample
- Lowering Variance:
  - Averaging over decision trees
- How can we generate different decision trees?
  - Sub-sample the sample
  - Force certain attributes

Random Forest:

- Create K different decision trees:
- Sample:
  - Select a random sub-sample
  - Practice: 66%
- GOAL:
  - Generate a variety of DT
  - Well correlated with y
- Combining: Majority

- Attributes:
  - In each node select subset F of attributes
    - |F|=M
    - Weak learners
  - Select the best attr. in F
- Values of M:
  - M=1: random
  - M=N all attributes
    - regular DT
  - $1 \ll M \ll N$
    - Subset of attributes

Random Forest: Conclusion

- Benefits:
  - Fast to run
  - Fairly stable outcome
  - Competitive performance
  - Handles missing/partial data
- Weaknesses:
  - Losses the interpretability
    - of DT
  - Many parameters around
  - Feature selection could be also a weakness