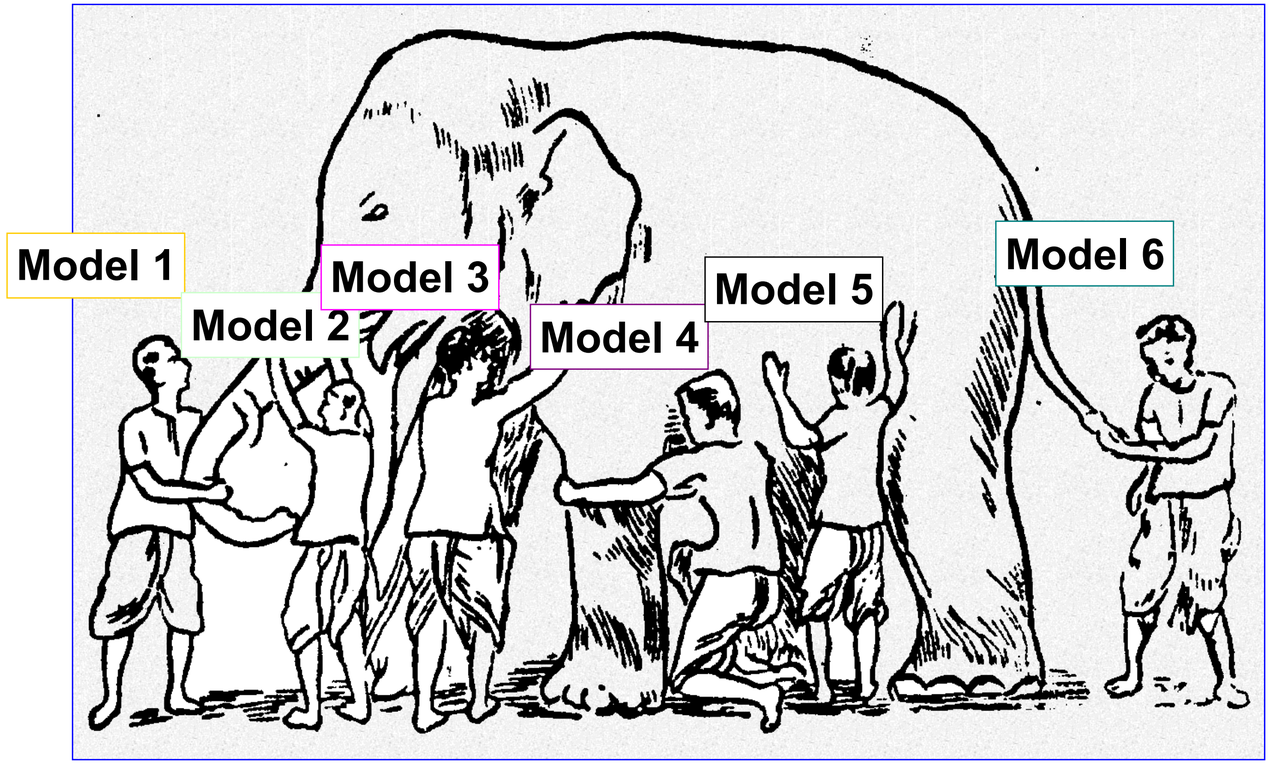
Ensemble Methods
RF

Machine Learning and Deep Learning
Lesson #7

Ensemble Methods

- **Basic idea** is to learn a set of classifiers (experts) and to allow them to vote.
- **Advantage**: improvement in predictive accuracy.
- **Disadvantage**: it is difficult to understand an ensemble of classifiers.



Method for ensembles building

- One way to force a learning algorithm to construct multiple hypotheses:

  run the algorithm several times and provide it with somewhat different data

  This idea is used in the following methods:

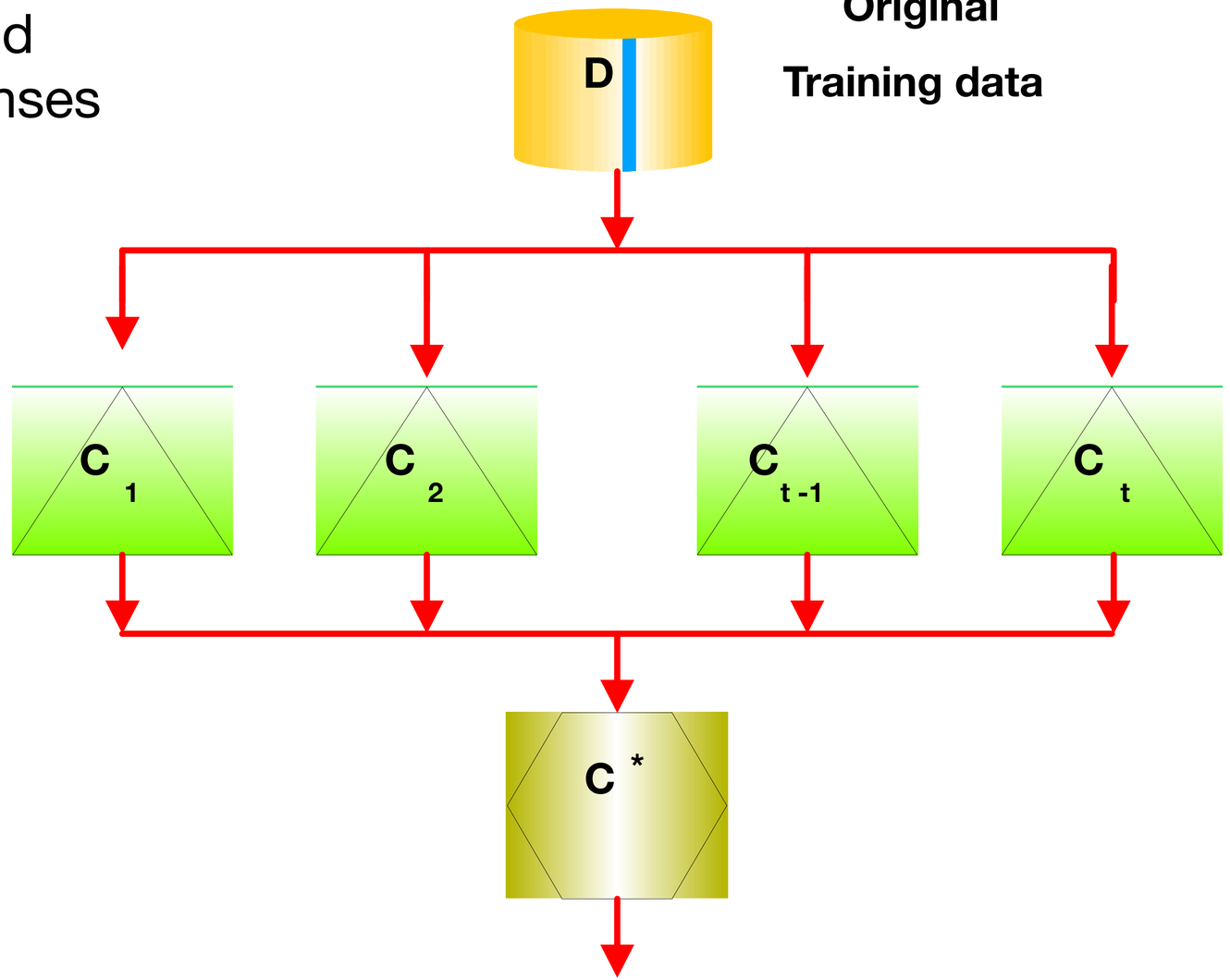
  - *Majority Voting*
  - *Bagging*
  - *Randomness Injection*
  - *Feature-Selection Ensembles:*
    - *Boosting*

Majority Voting

- Build Multiple classifier and take the majority of responses

Step 1:
Build Multiple Classifiers

Step 2:
Combine Classifiers

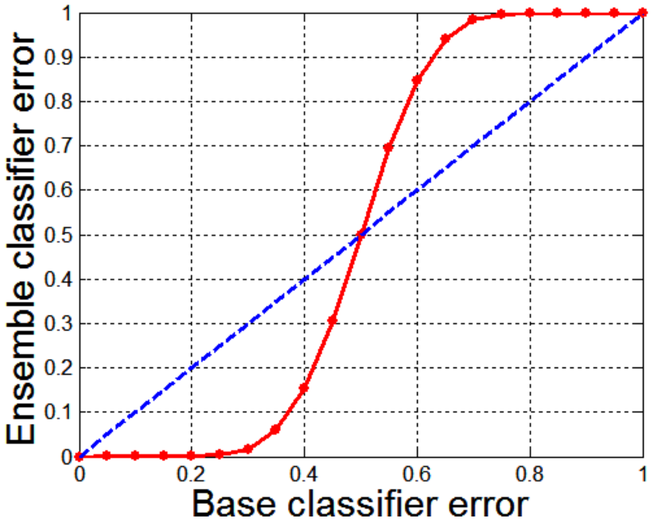


Why does majority work

- Suppose there are 25 base classifiers
- Each classifier has error rate, $\varepsilon = 0.35$
  - Assume errors made by classifiers are uncorrelated
  - Probability that the ensemble classifier makes a wrong prediction follows the Binomial distribution:

$$Pr(X \leq k) = \sum_{i}^{k} \binom{n}{i} p^i (1-p)^{n-i}$$

$$Pr(X \geq 13) = \sum_{13}^{25} \binom{25}{i} \epsilon^i (1-\epsilon)^{25-i} = 0.06$$



Bagging

- **Introduced by Breiman** (1996) “Bagging” stands for “bootstrap aggregating”.

- **Employs simplest way of combining predictions** that belong to the same type.
- Combining can be realized **with voting or averaging**
- Each model receives equal weight

Idealized” version of bagging:

1. **Sample** several training sets of size n (instead of just having one training set of size n)
2. **Build** a classifier for each training set
3. **Combine** the classifier’s predictions
4. This improves performance in almost all cases

Why does it works

- Suppose every classifier learn a function y=h(x)+e where e is the error
- The average classifier error for M classifiers on the training set of D points is

$$E_{AV} = \frac{1}{M} \sum_{i}^{M} E_D[e_i^2]$$

- The commitee/Bagged prediction is the average of predictions

$$y_{COM} = \frac{1}{M} \sum_{i}^{M} h_i(x) + e_i(x)$$

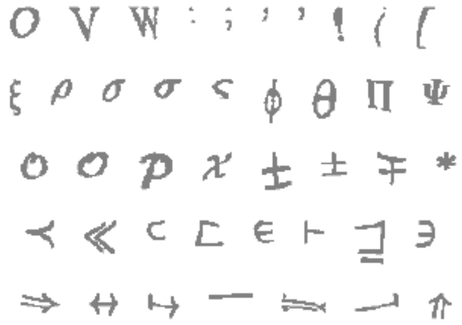
- The average error of the commitee is then

$$E_{COM} = E_D[\frac{1}{M} \sum_{i}^{M} e_i(x)]$$

- that is the average value of the average error

Random Forest

Randomized Forests in Vision

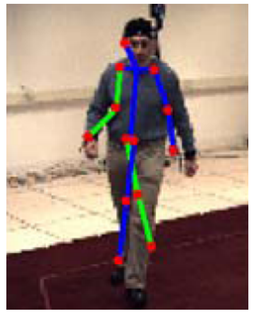


digit recognition

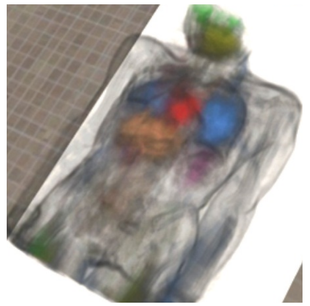
[Lepetit *et al.*, 06]
keypoint recognition

[Moosmann *et al.*, 06]
visual word clustering

[Shotton *et al.*, 08]
object segmentation

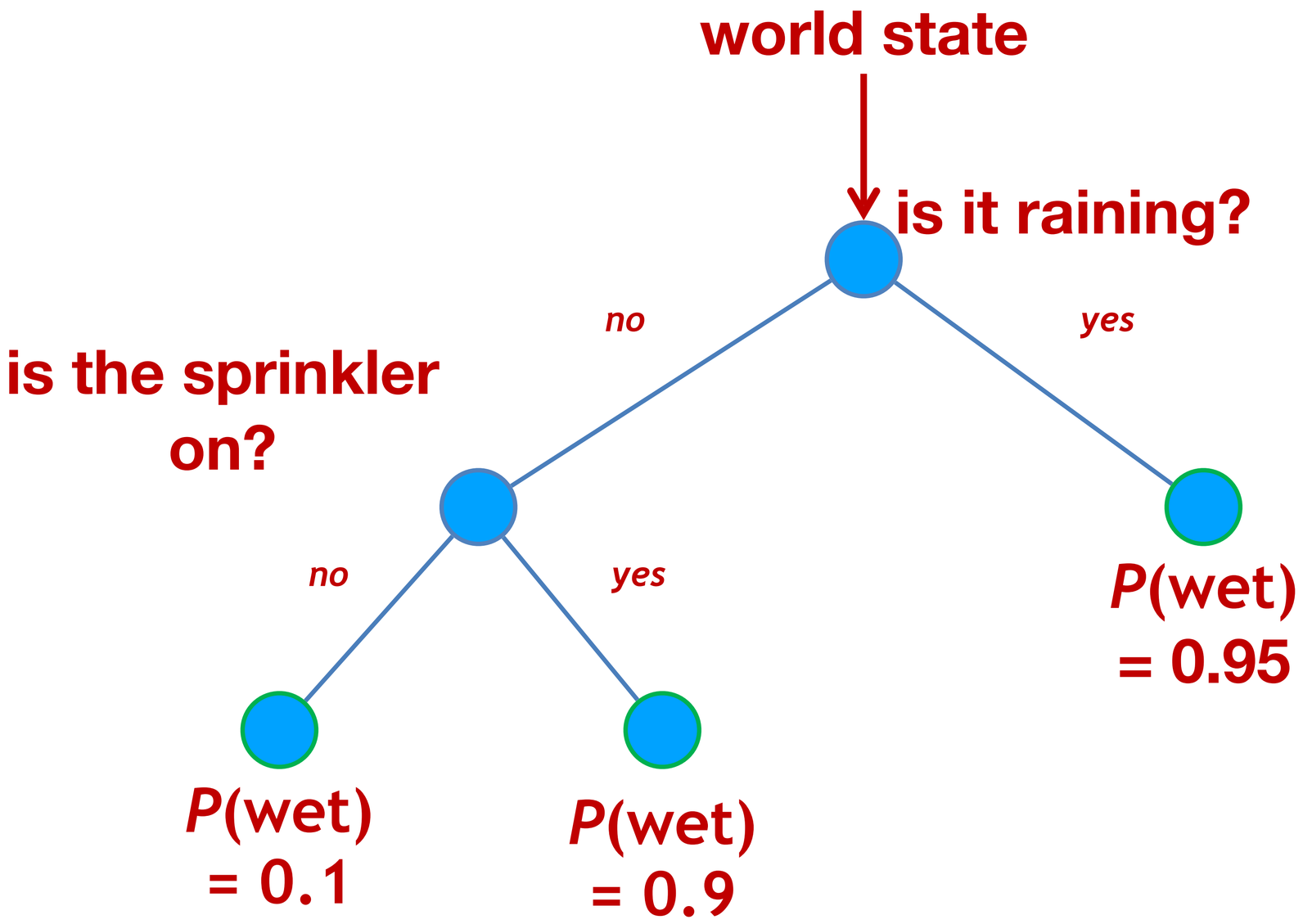


[Rogez *et al.*, 08]
pose estimation

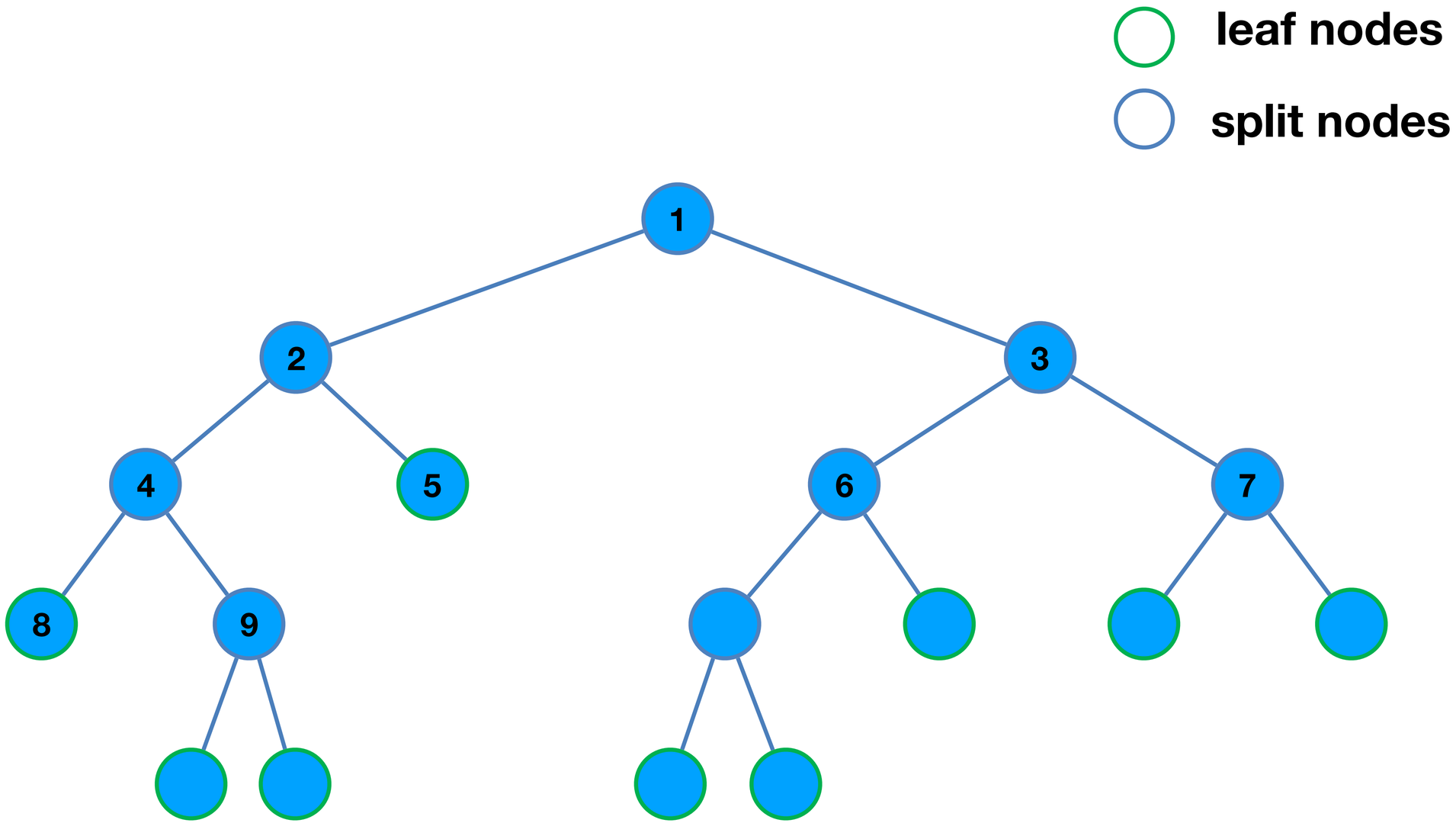


[Criminisi *et al.*, 09]
organ detection

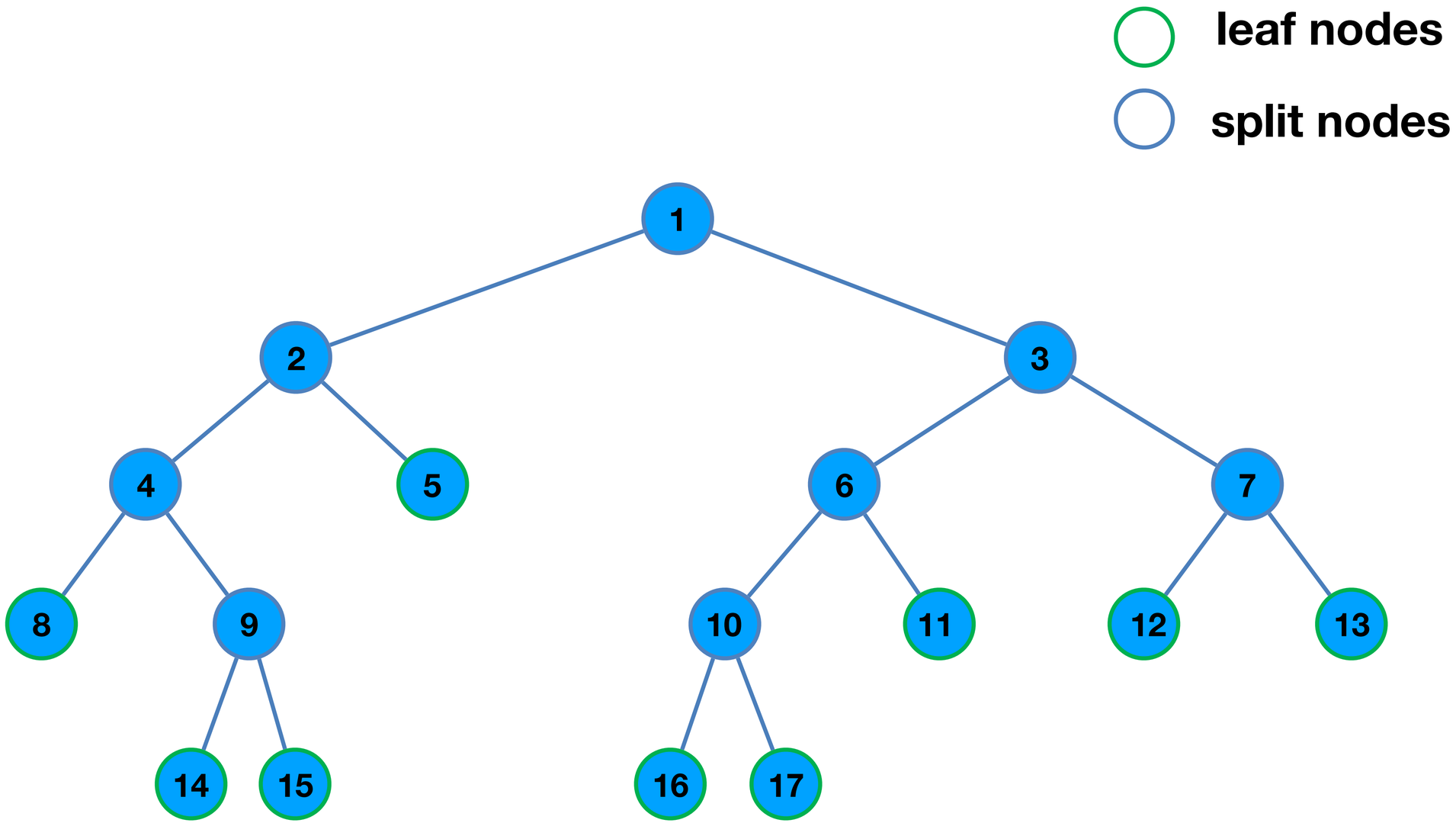
The Basics: Is The Grass Wet?



The Basics: Binary Decision Trees

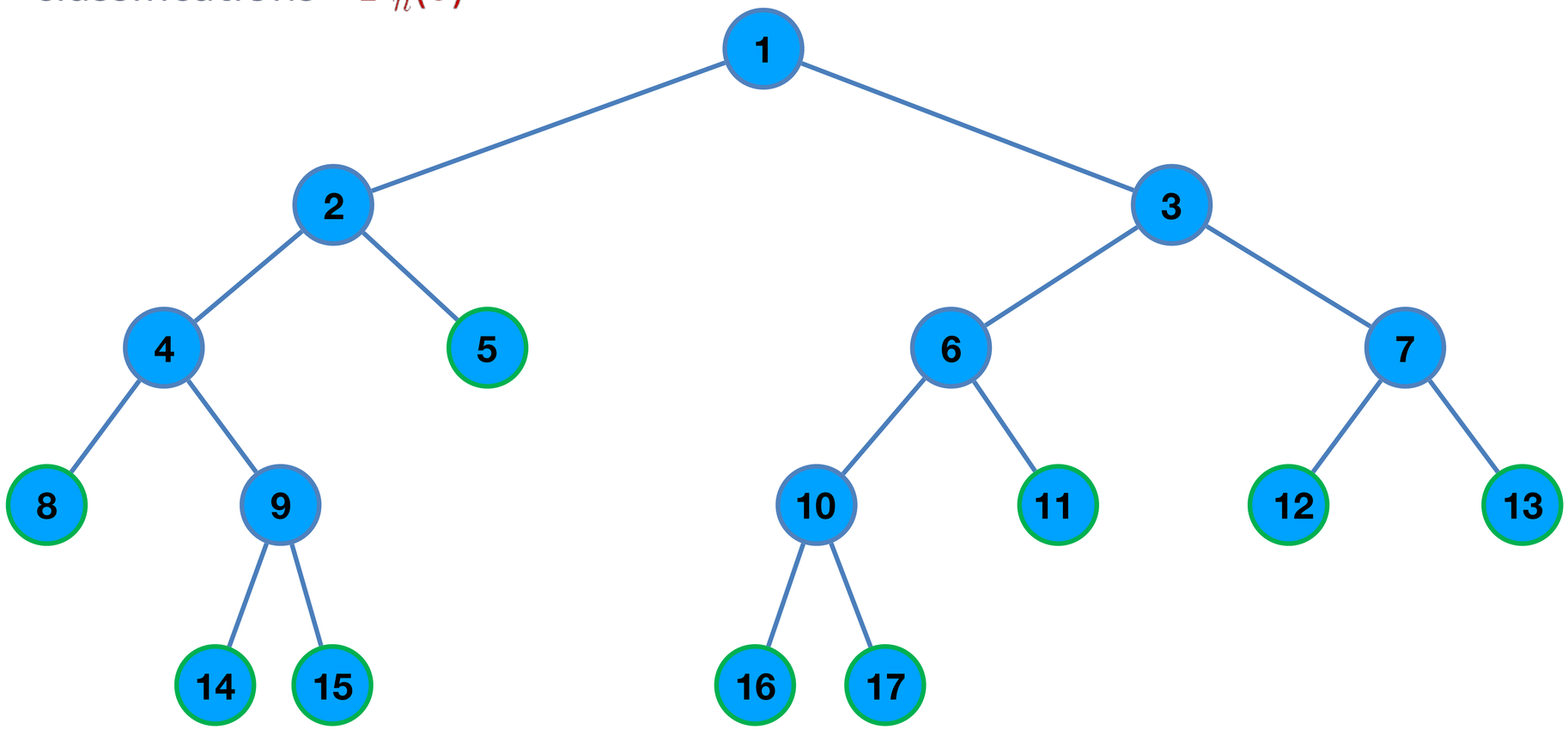


The Basics: Binary Decision Trees



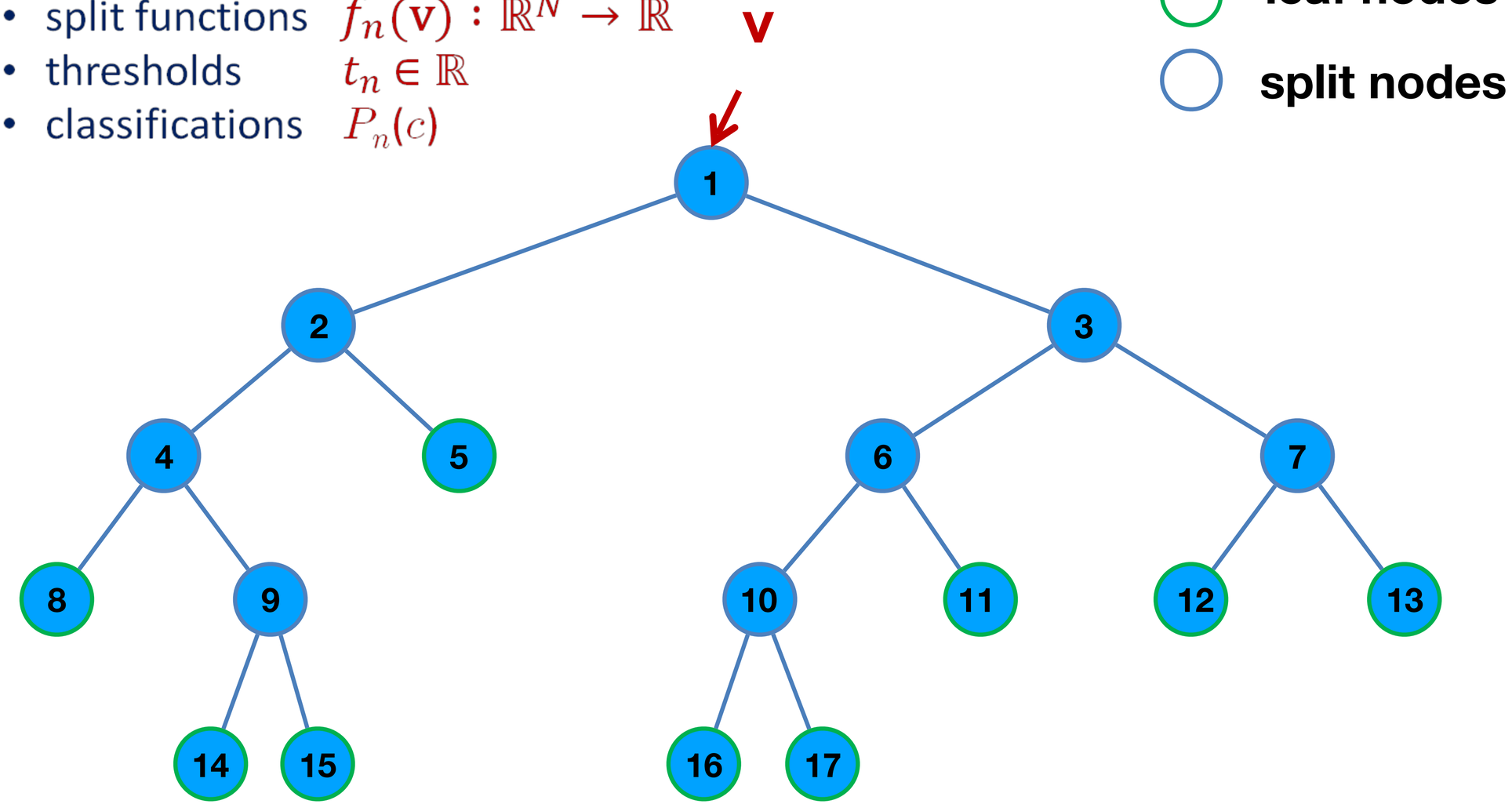
The Basics: Binary Decision Trees

- feature vector $\mathbf{v} \in \mathbb{R}^N$
- split functions $f_n(\mathbf{v}) : \mathbb{R}^N \rightarrow \mathbb{R}$
- thresholds $t_n \in \mathbb{R}$
- classifications $P_n(c)$



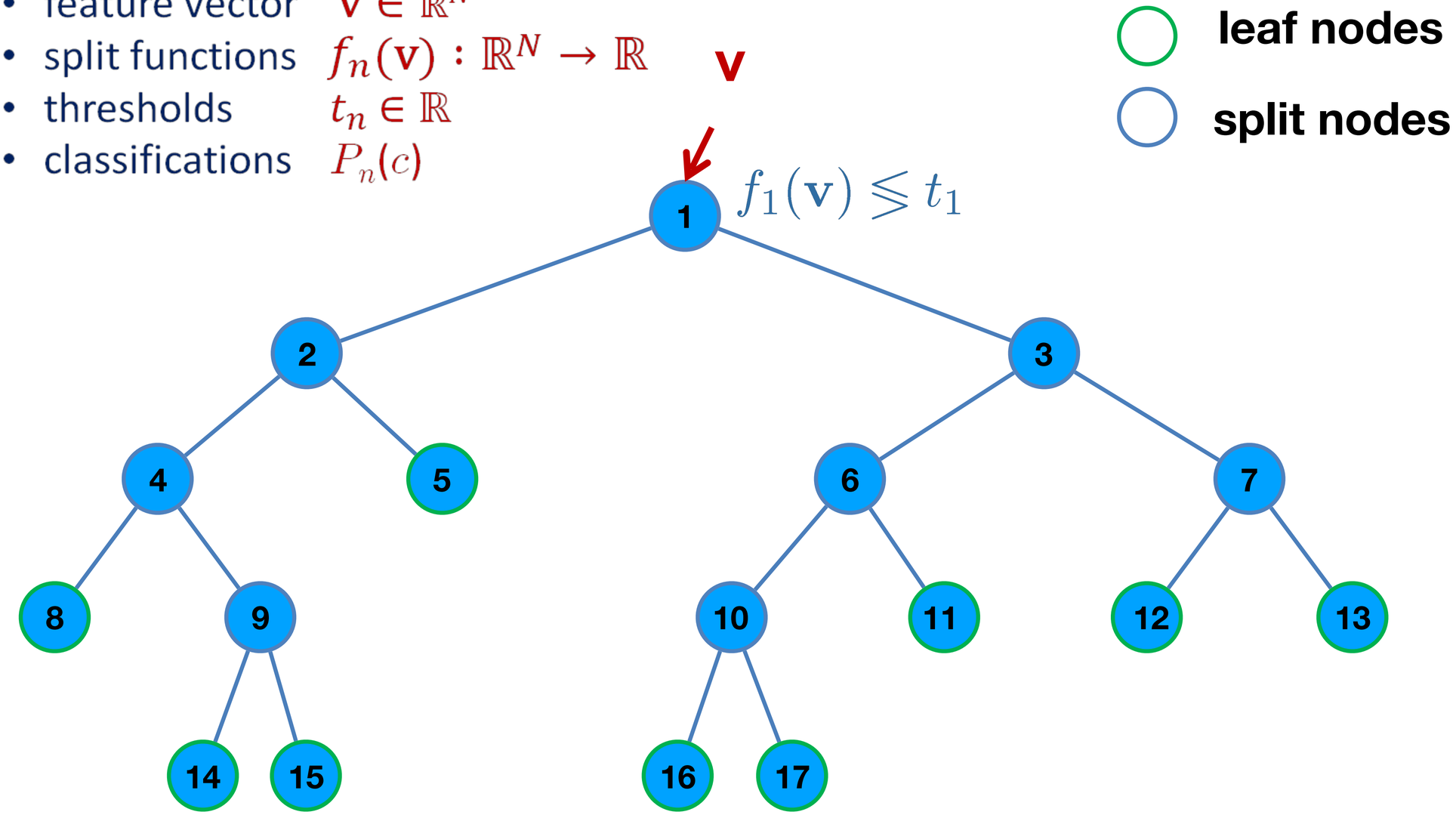
The Basics: Binary Decision Trees

- feature vector $\mathbf{v} \in \mathbb{R}^N$
- split functions $f_n(\mathbf{v}) : \mathbb{R}^N \rightarrow \mathbb{R}$
- thresholds $t_n \in \mathbb{R}$
- classifications $P_n(c)$



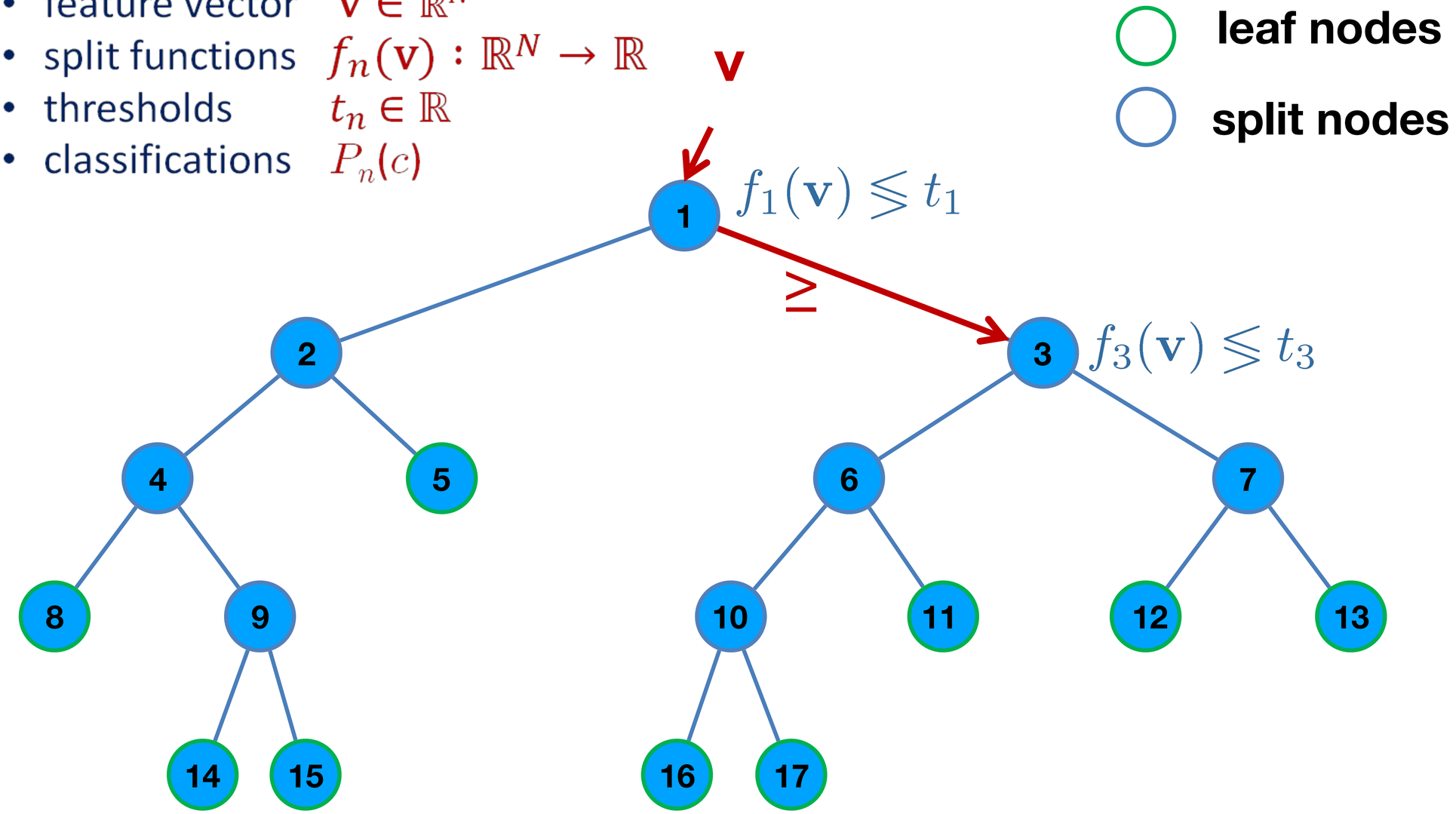
The Basics: Binary Decision Trees

- feature vector $\mathbf{v} \in \mathbb{R}^N$
- split functions $f_n(\mathbf{v}) : \mathbb{R}^N \rightarrow \mathbb{R}$
- thresholds $t_n \in \mathbb{R}$
- classifications $P_n(c)$



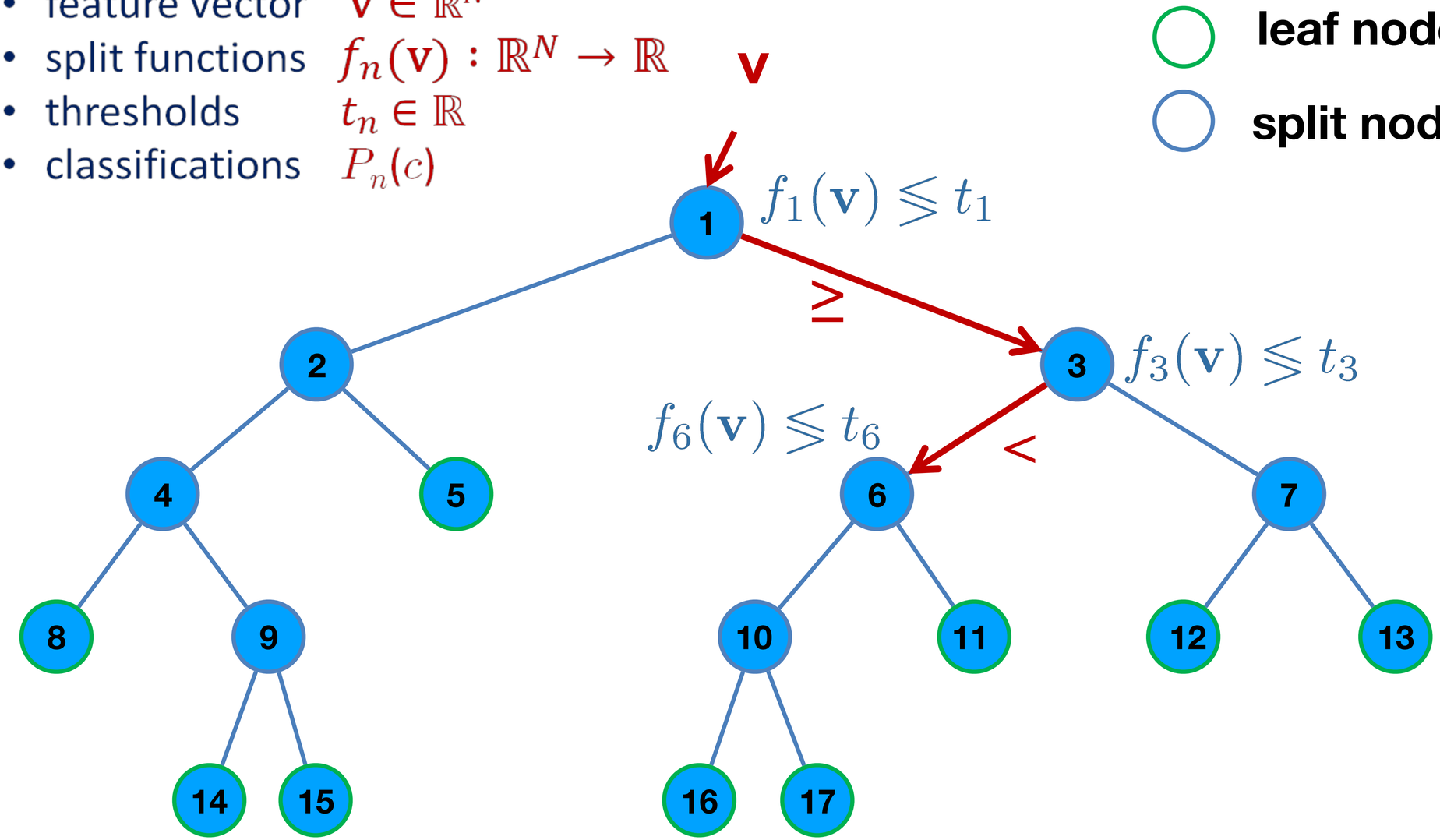
The Basics: Binary Decision Trees

- feature vector $\mathbf{v} \in \mathbb{R}^N$
- split functions $f_n(\mathbf{v}) : \mathbb{R}^N \rightarrow \mathbb{R}$
- thresholds $t_n \in \mathbb{R}$
- classifications $P_n(c)$



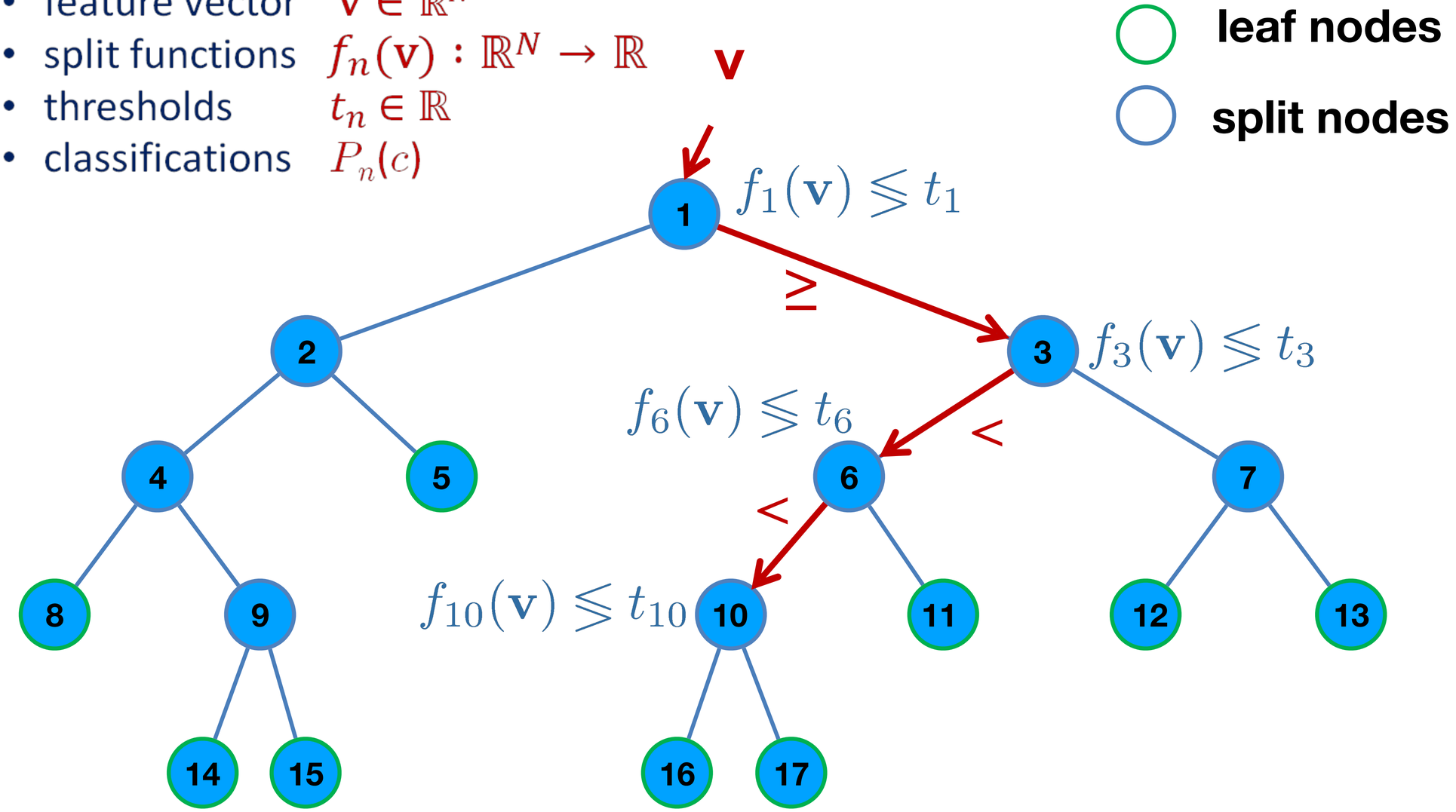
The Basics: Binary Decision Trees

- feature vector $\mathbf{v} \in \mathbb{R}^N$
- split functions $f_n(\mathbf{v}) : \mathbb{R}^N \rightarrow \mathbb{R}$
- thresholds $t_n \in \mathbb{R}$
- classifications $P_n(c)$



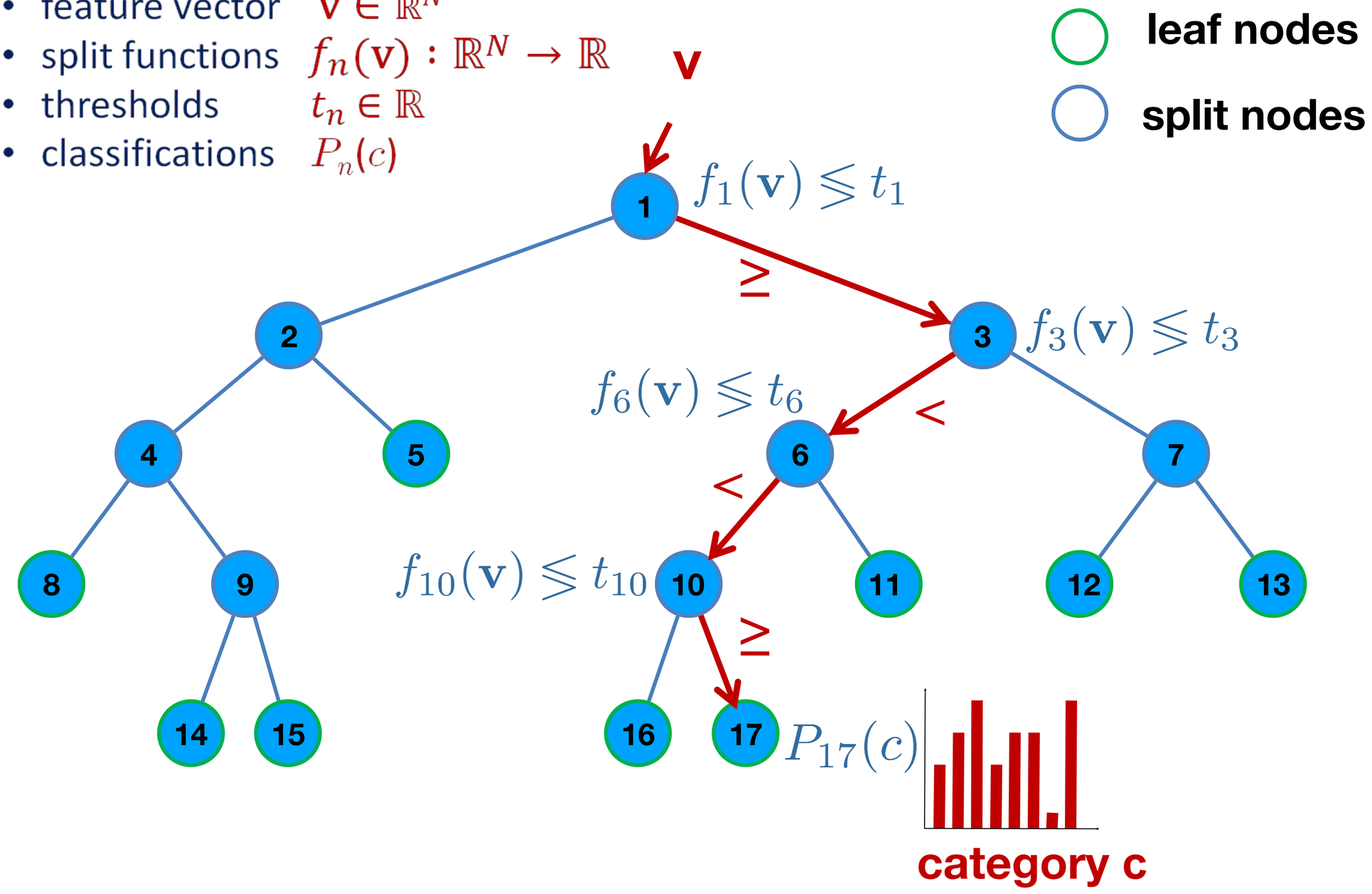
The Basics: Binary Decision Trees

- feature vector $\mathbf{v} \in \mathbb{R}^N$
- split functions $f_n(\mathbf{v}) : \mathbb{R}^N \rightarrow \mathbb{R}$
- thresholds $t_n \in \mathbb{R}$
- classifications $P_n(c)$



The Basics: Binary Decision Trees

- feature vector $\mathbf{v} \in \mathbb{R}^N$
- split functions $f_n(\mathbf{v}) : \mathbb{R}^N \rightarrow \mathbb{R}$
- thresholds $t_n \in \mathbb{R}$
- classifications $P_n(c)$

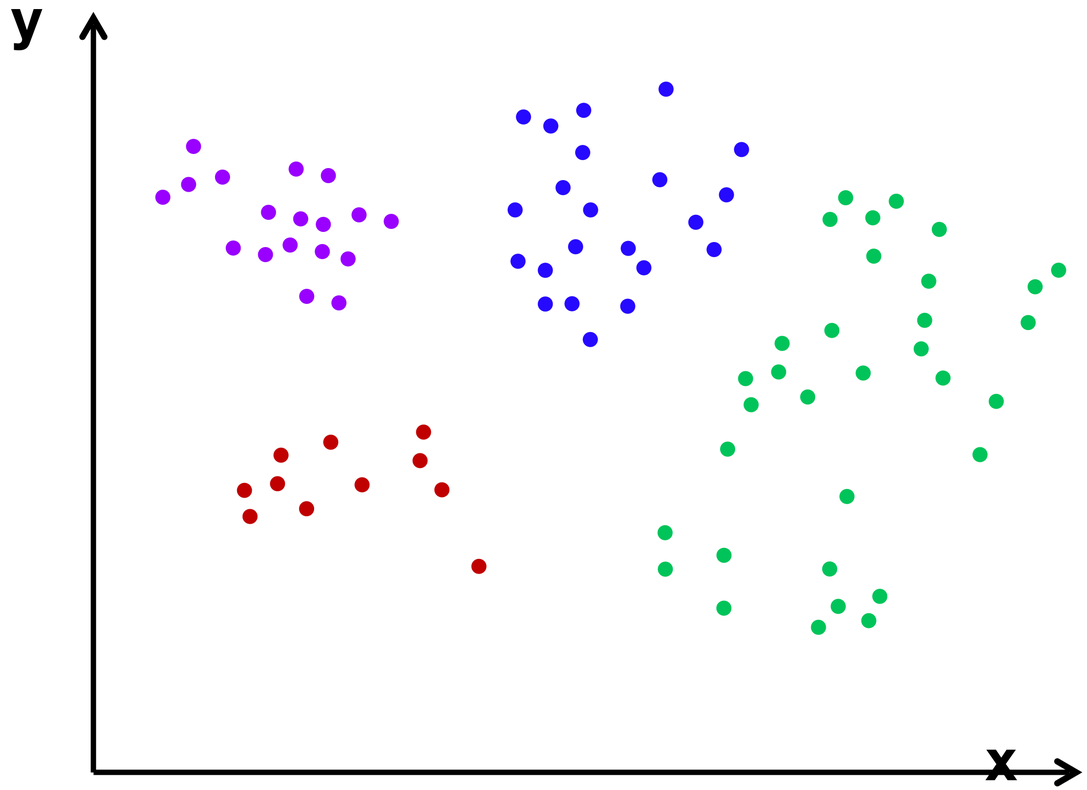


Decision Tree Pseudo-Code

```
double[] ClassifyDT(node, v)
    if node.IsSplitNode then
        if node.f(v) >= node.t then
            return
        ClassifyDT(node.right, v)
        else
            return
        ClassifyDT(node.left, v)
        end
    else
    return node.P
    end
end
```

Toy Learning Example

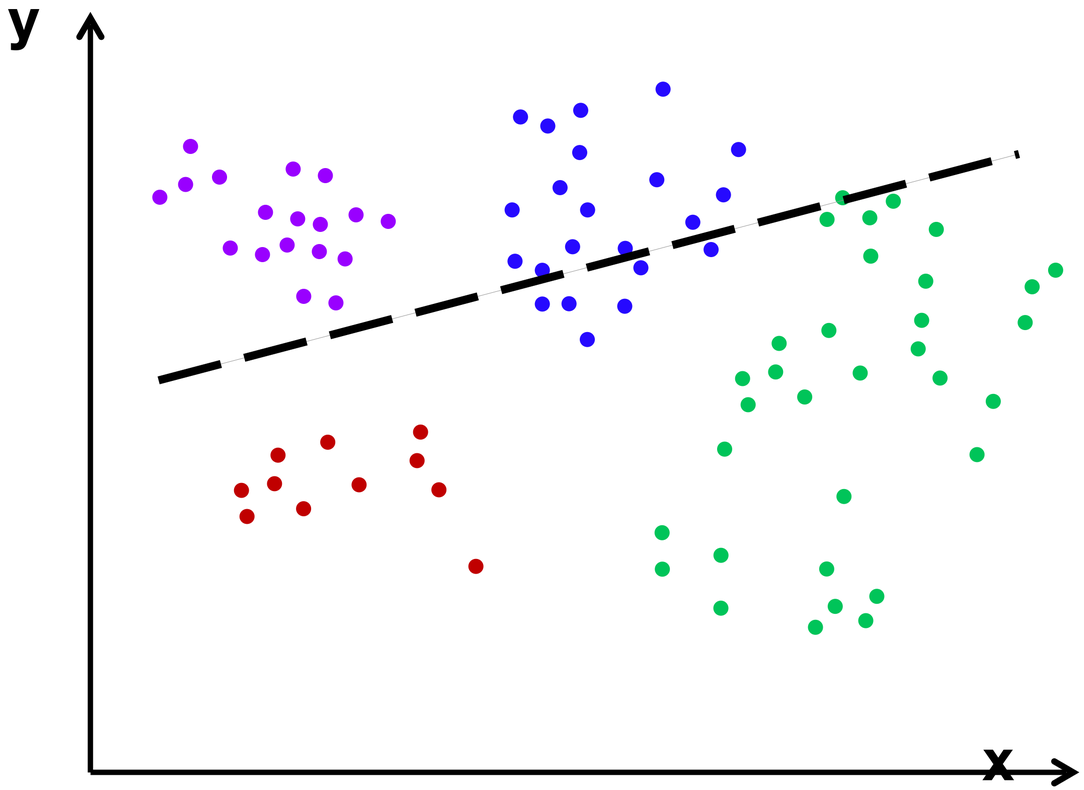
- **Try** several lines, chosen at random
- **Keep** line that best separates data
  - **information gain**
- **Recurse**



- **feature vectors are x, y coordinates:** $v = [x, y]^T$
- **split functions are lines with parameters a, b:** $f_n(v) = ax + by$
- **threshold determines intercepts:** t_n
- **four classes: purple, blue, red, green**

Toy Learning Example

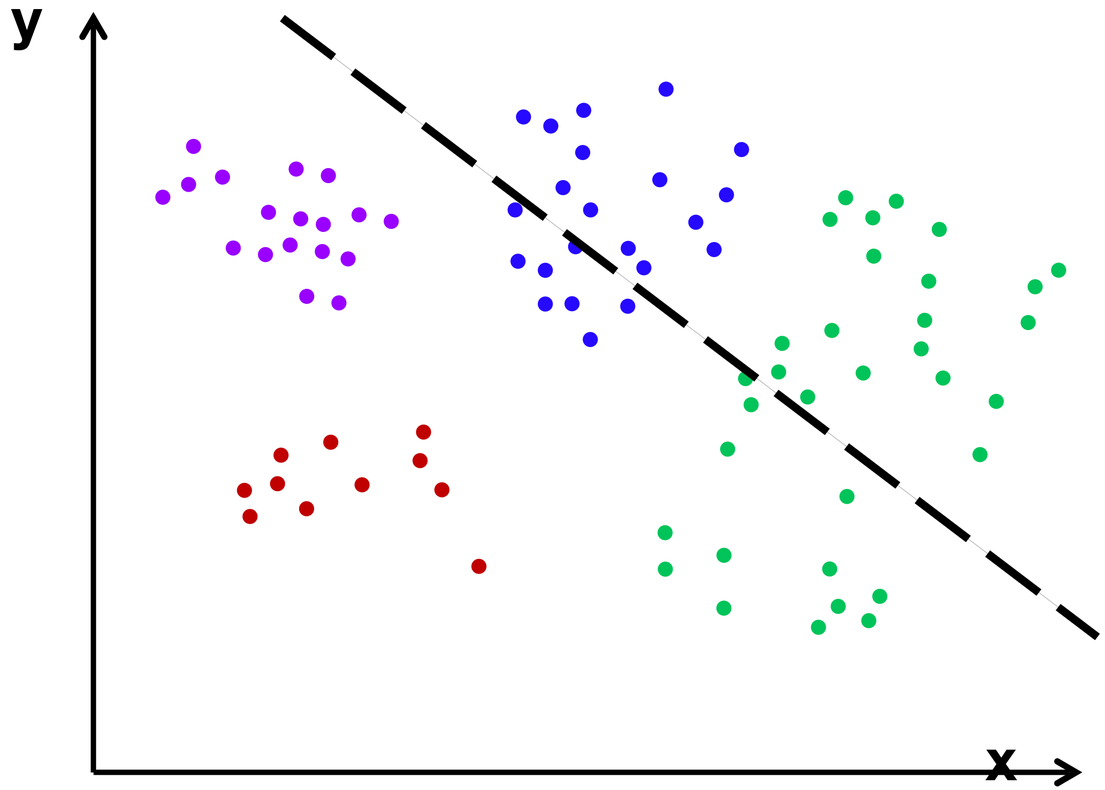
- **Try** several lines, chosen at random
- **Keep** line that best separates data
  - **information gain**
- **Recurse**



- **feature vectors are x, y coordinates:** $v = [x, y]^T$
- **split functions are lines with parameters a, b:** $f_n(v) = ax + by$
- **threshold determines intercepts:** t_n
- **four classes: purple, blue, red, green**

Toy Learning Example

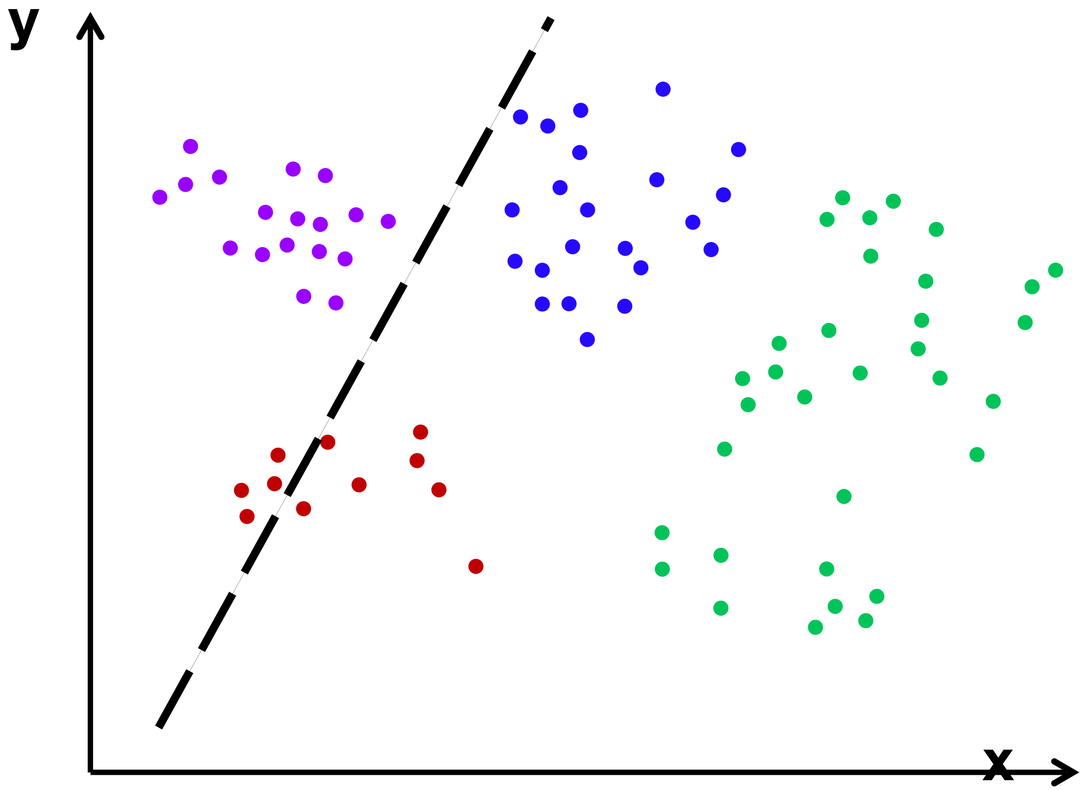
- **Try** several lines, chosen at random
- **Keep** line that best separates data
  - **information gain**
- **Recurse**



- **feature vectors are x, y coordinates:** $v = [x, y]^T$
- **split functions are lines with parameters a, b:** $f_n(v) = ax + by$
- **threshold determines intercepts:** t_n
- **four classes: purple, blue, red, green**

Toy Learning Example

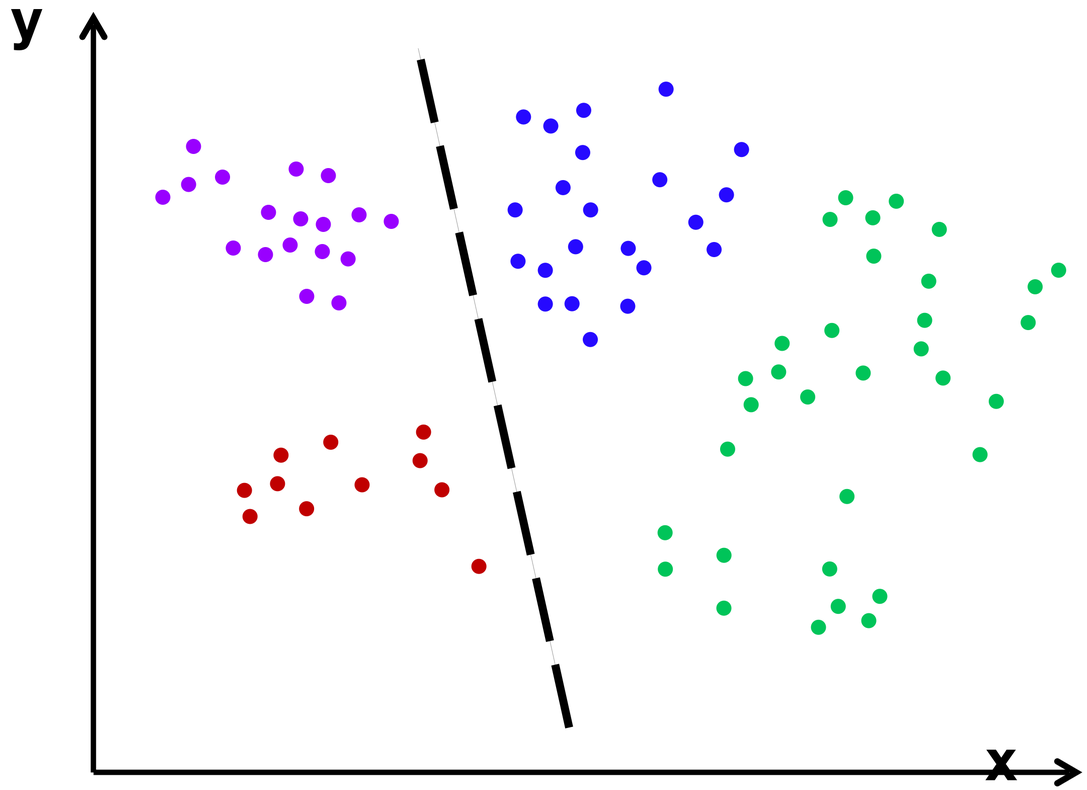
- **Try** several lines, chosen at random
- **Keep** line that best separates data
  - **information gain**
- **Recurse**



- **feature vectors are x, y coordinates:** $v = [x, y]^T$
- **split functions are lines with parameters a, b:** $f_n(v) = ax + by$
- **threshold determines intercepts:** t_n
- **four classes: purple, blue, red, green**

Toy Learning Example

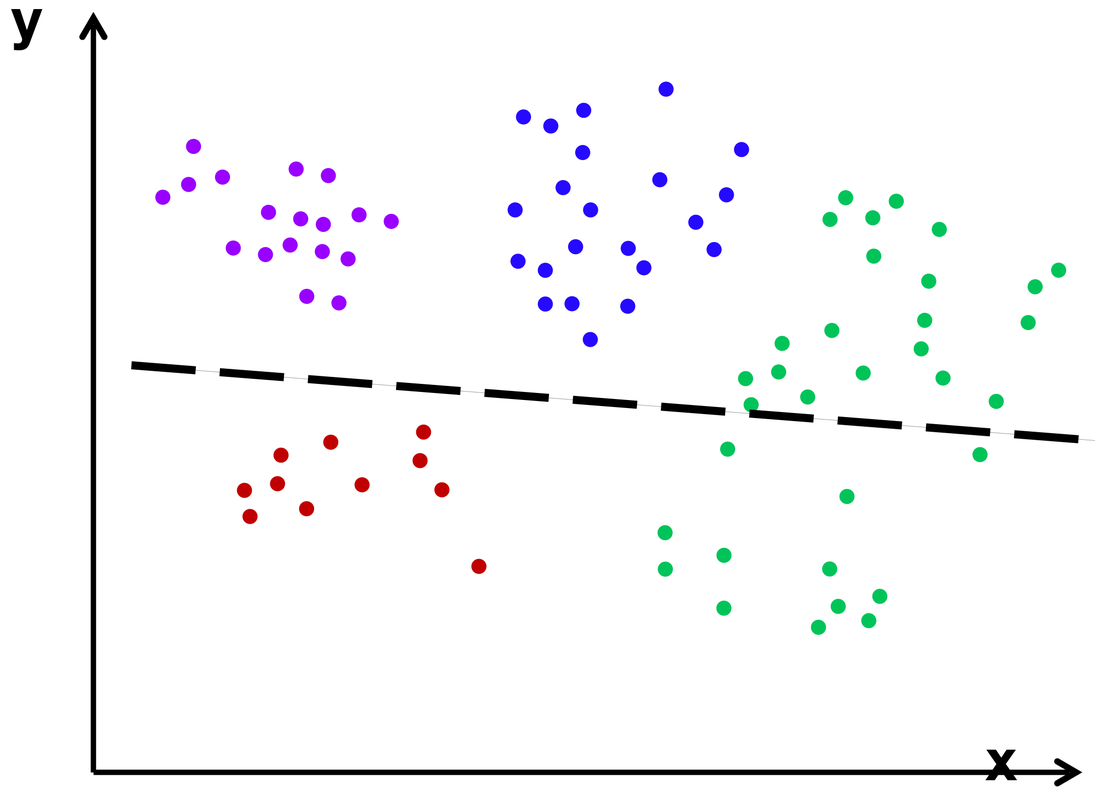
- **Try** several lines, chosen at random
- **Keep** line that best separates data
  - **information gain**
- **Recurse**



- **feature vectors are x, y coordinates:** $v = [x, y]^T$
- **split functions are lines with parameters a, b:** $f_n(v) = ax + by$
- **threshold determines intercepts:** t_n
- **four classes: purple, blue, red, green**

Toy Learning Example

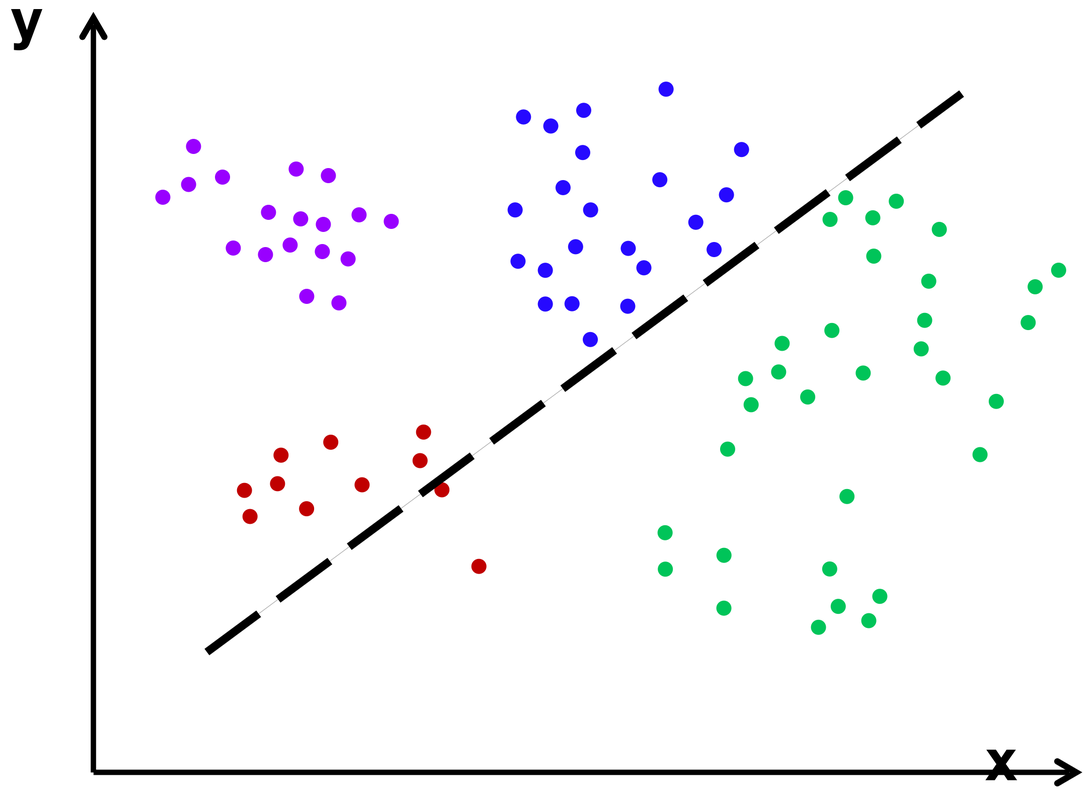
- **Try** several lines, chosen at random
- **Keep** line that best separates data
  - **information gain**
- **Recurse**



- **feature vectors are x, y coordinates:** $v = [x, y]^T$
- **split functions are lines with parameters a, b:** $f_n(v) = ax + by$
- **threshold determines intercepts:** t_n
- **four classes: purple, blue, red, green**

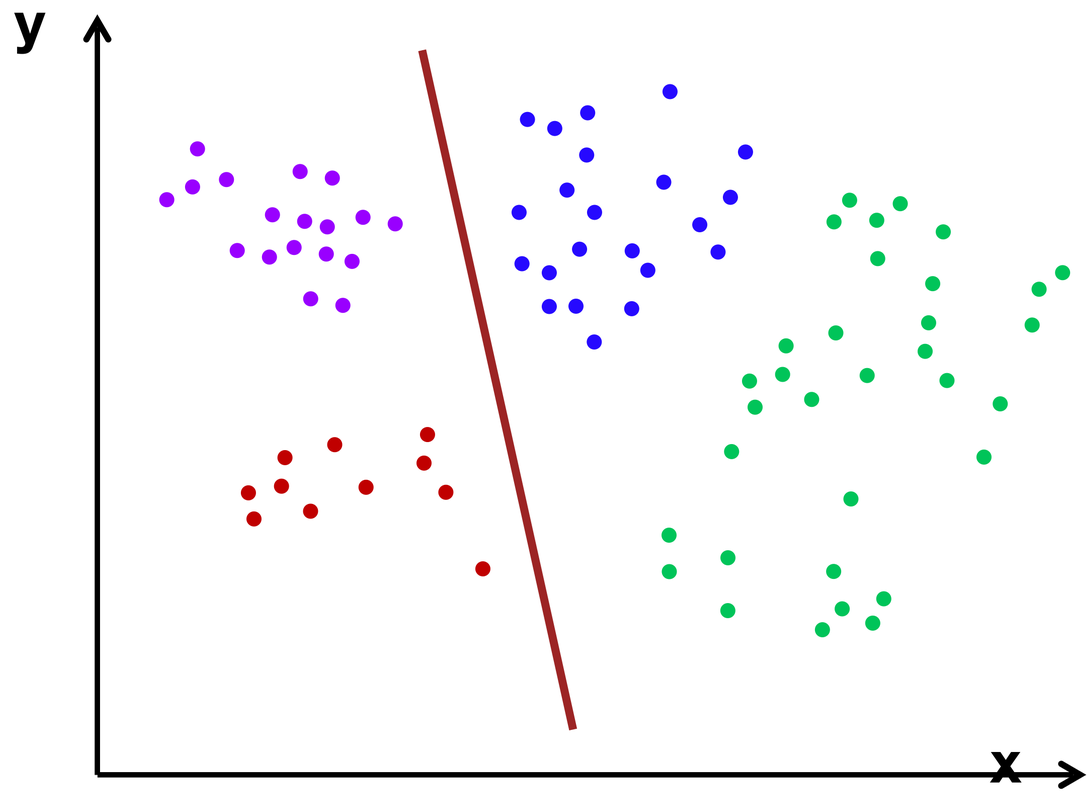
Toy Learning Example

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



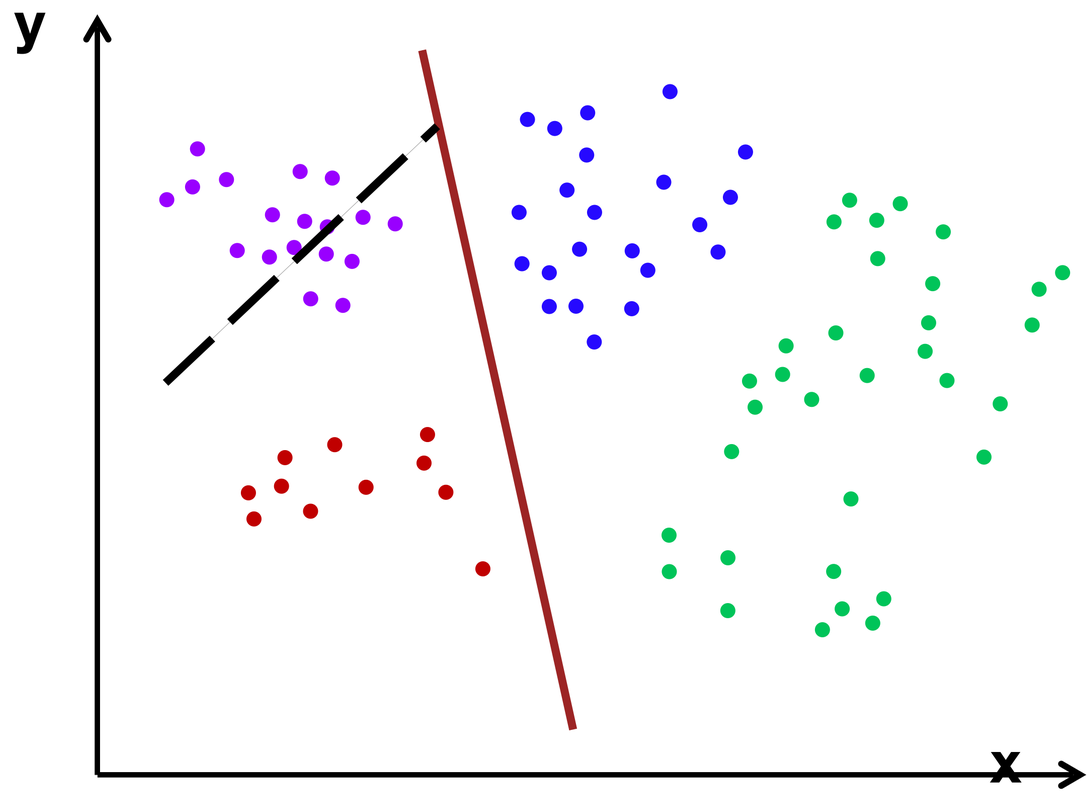
- **feature vectors are x, y coordinates:** $v = [x, y]^T$
- **split functions are lines with parameters a, b:** $f_n(v) = ax + by$
- **threshold determines intercepts:** t_n
- **four classes: purple, blue, red, green**

Toy Learning Example



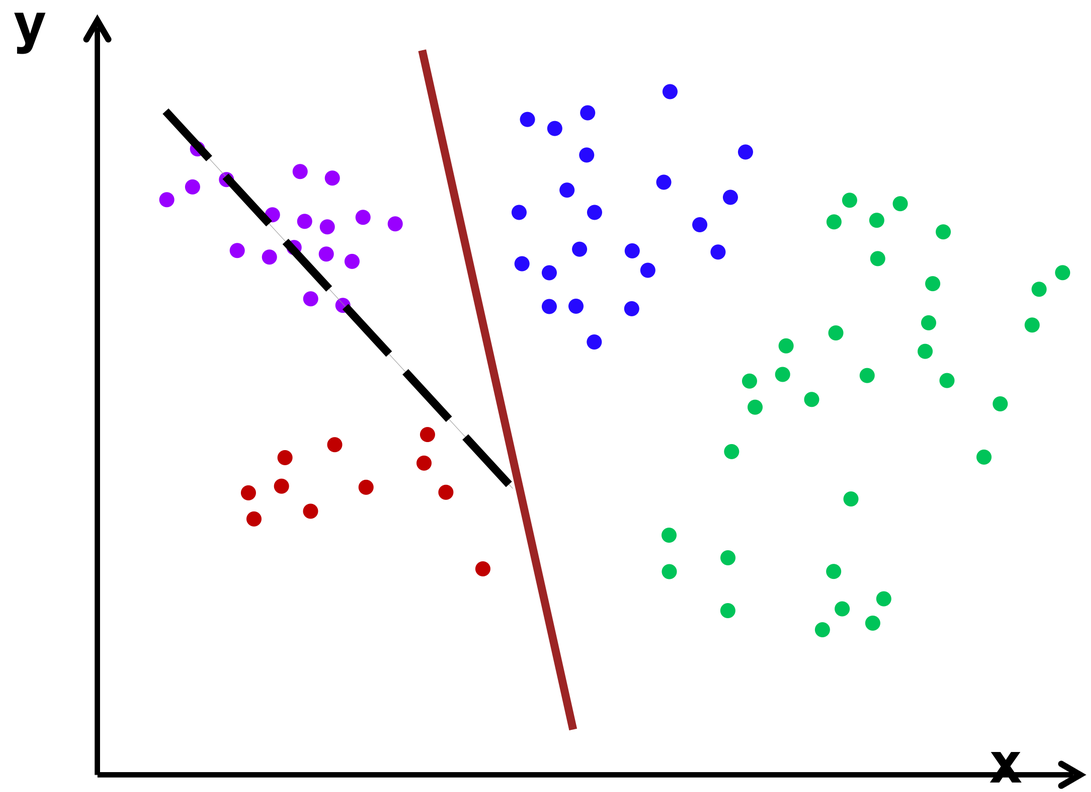
- feature vectors are x, y coordinates: $v = [x, y]^T$
- split functions are lines with parameters a, b: $f_n(v) = ax + by$
- threshold determines intercepts: t_n
- four classes: purple, blue, red, green

Toy Learning Example



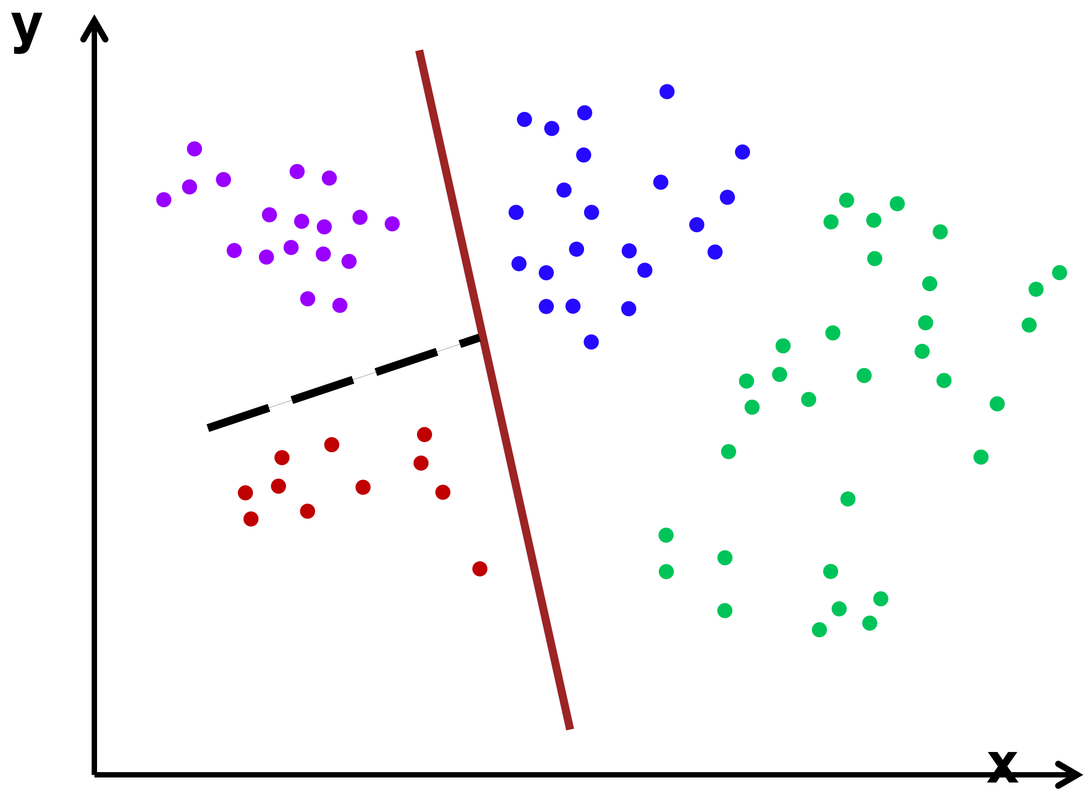
- feature vectors are x, y coordinates: $v = [x, y]^T$
- split functions are lines with parameters a, b: $f_n(v) = ax + by$
- threshold determines intercepts: t_n
- four classes: purple, blue, red, green

Toy Learning Example



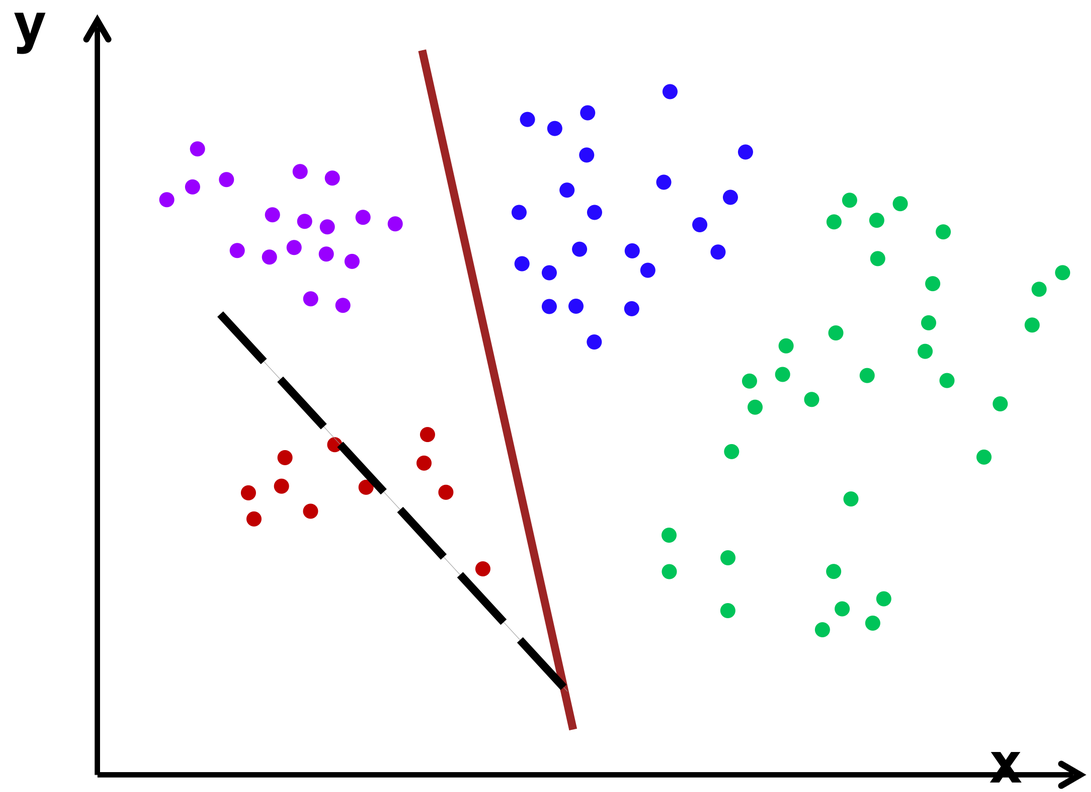
- feature vectors are x, y coordinates: $v = [x, y]^T$
- split functions are lines with parameters a, b: $f_n(v) = ax + by$
- threshold determines intercepts: t_n
- four classes: purple, blue, red, green

Toy Learning Example



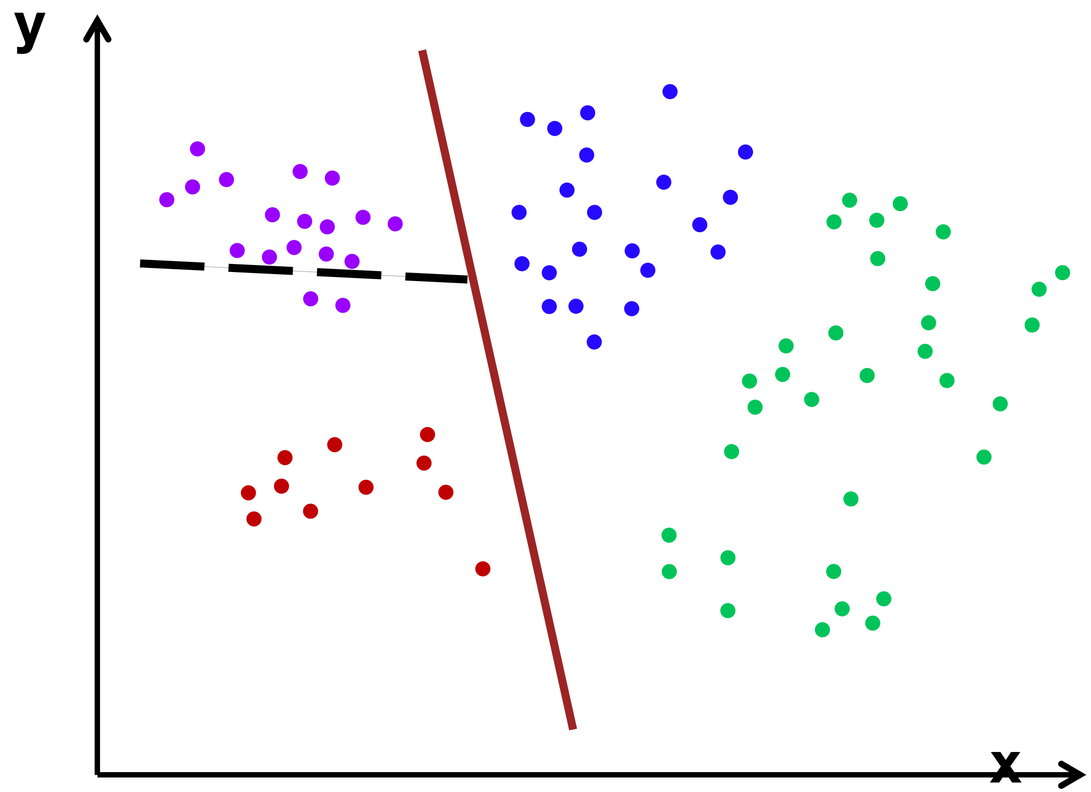
- feature vectors are x, y coordinates: $v = [x, y]^T$
- split functions are lines with parameters a, b: $f_n(v) = ax + by$
- threshold determines intercepts: t_n
- four classes: purple, blue, red, green

Toy Learning Example



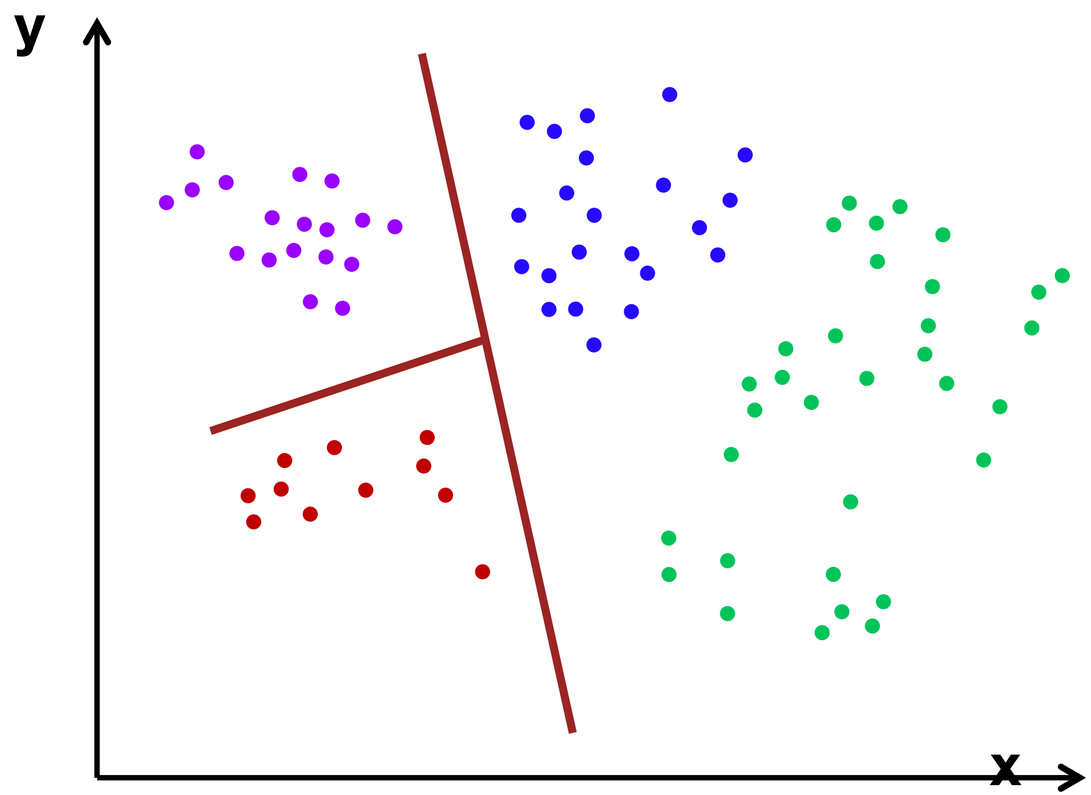
- feature vectors are x, y coordinates: $v = [x, y]^T$
- split functions are lines with parameters a, b: $f_n(v) = ax + by$
- threshold determines intercepts: t_n
- four classes: purple, blue, red, green

Toy Learning Example



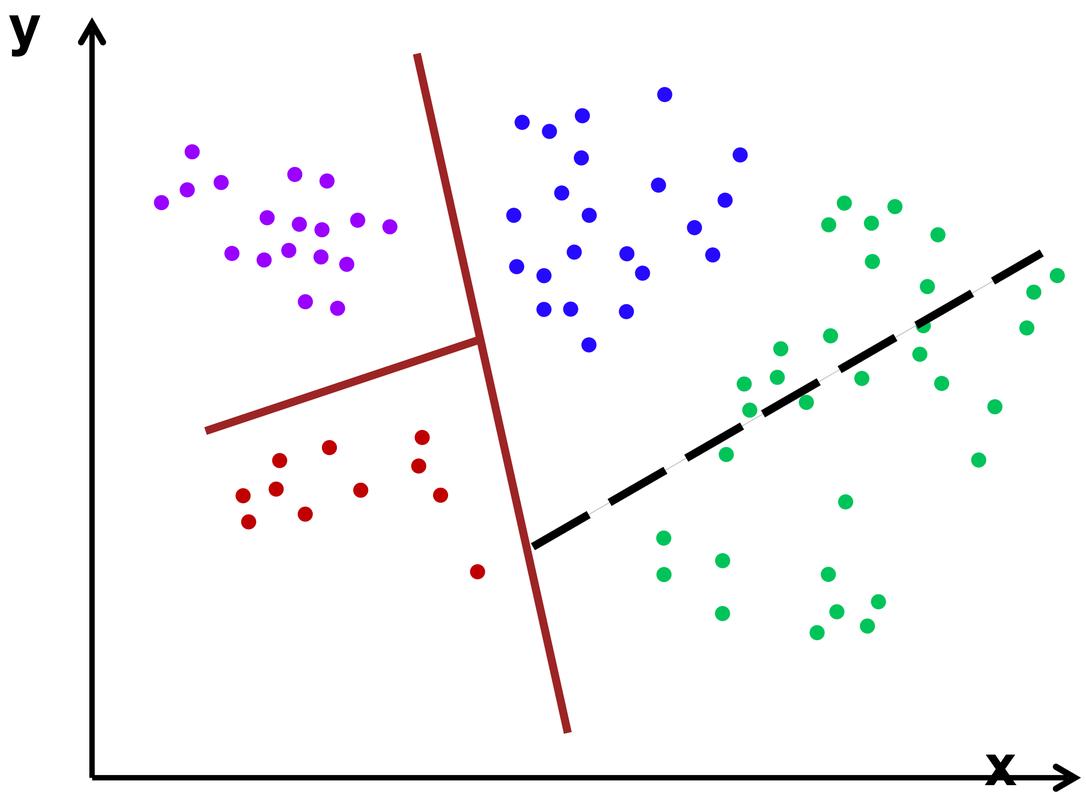
- feature vectors are x, y coordinates: $v = [x, y]^T$
- split functions are lines with parameters a, b: $f_n(v) = ax + by$
- threshold determines intercepts: t_n
- four classes: purple, blue, red, green

Toy Learning Example



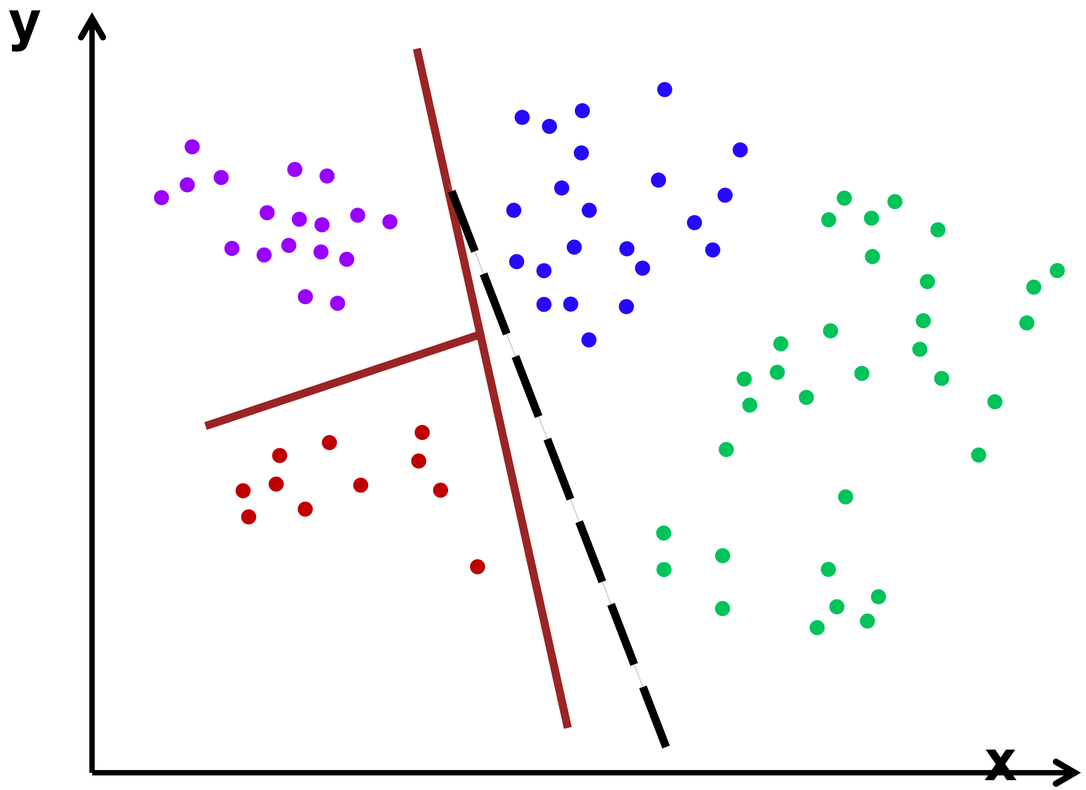
- feature vectors are x, y coordinates: $v = [x, y]^T$
- split functions are lines with parameters a, b: $f_n(v) = ax + by$
- threshold determines intercepts: t_n
- four classes: purple, blue, red, green

Toy Learning Example



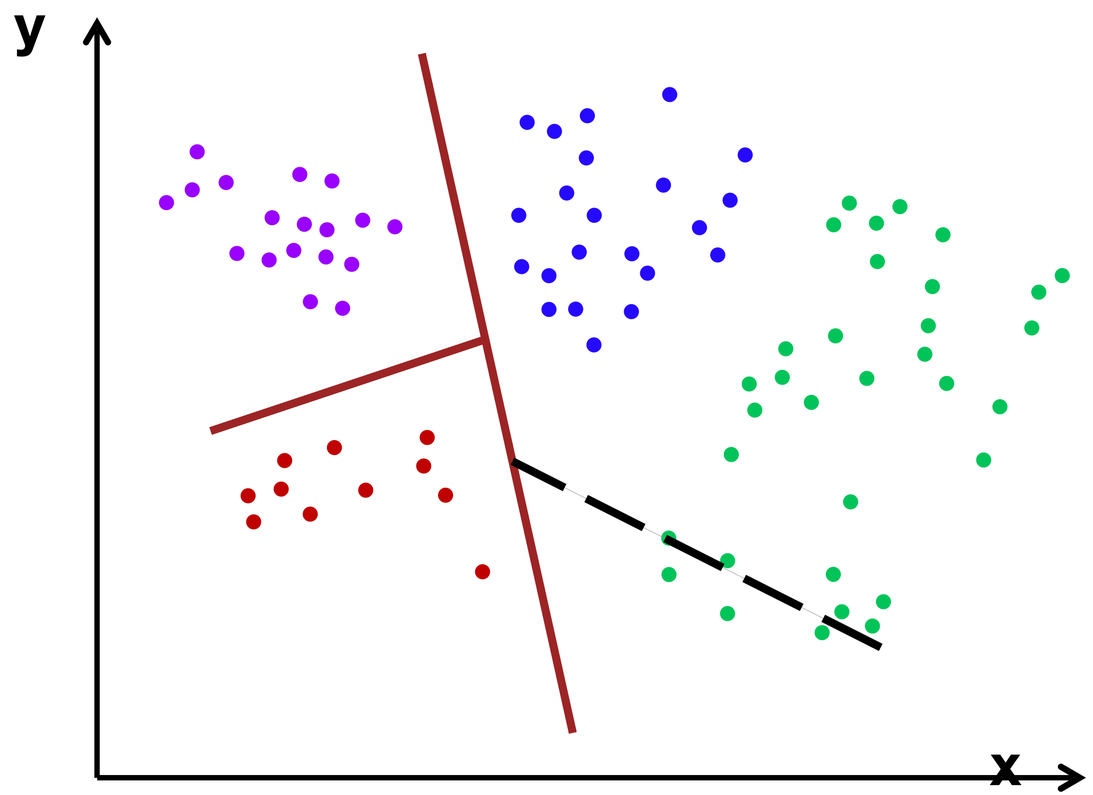
- feature vectors are x, y coordinates: $v = [x, y]^T$
- split functions are lines with parameters a, b: $f_n(v) = ax + by$
- threshold determines intercepts: t_n
- four classes: purple, blue, red, green

Toy Learning Example



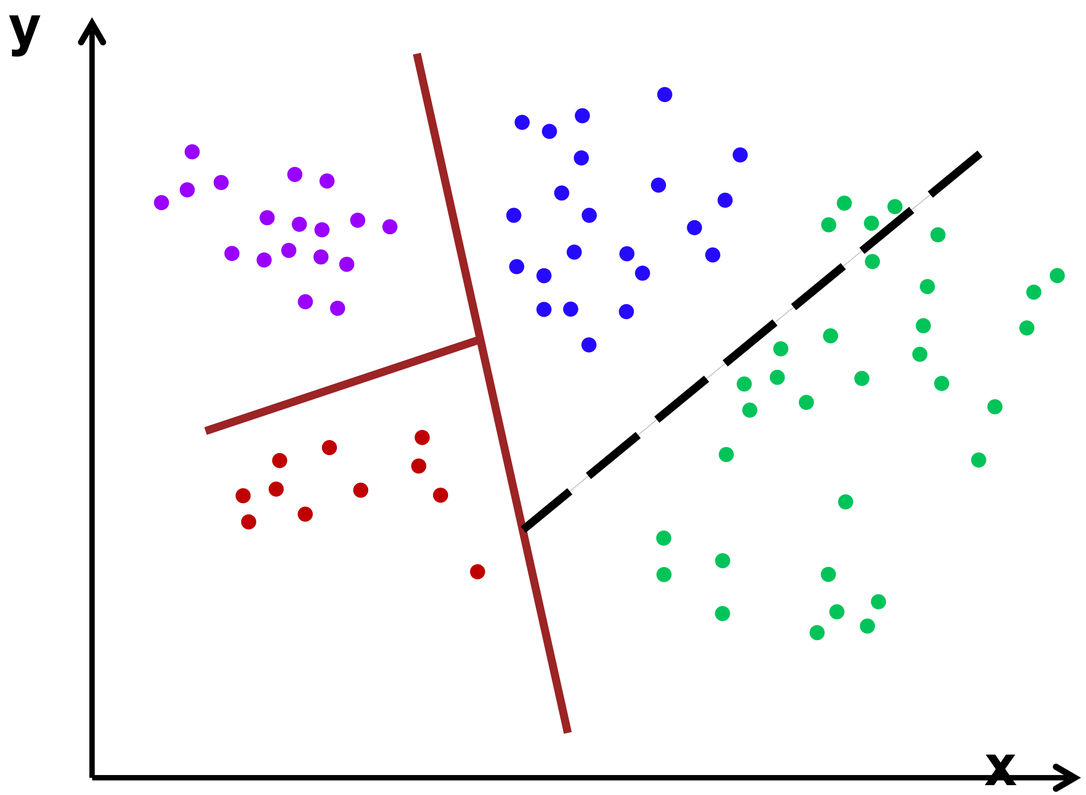
- feature vectors are x, y coordinates: $v = [x, y]^T$
- split functions are lines with parameters a, b: $f_n(v) = ax + by$
- threshold determines intercepts: t_n
- four classes: purple, blue, red, green

Toy Learning Example



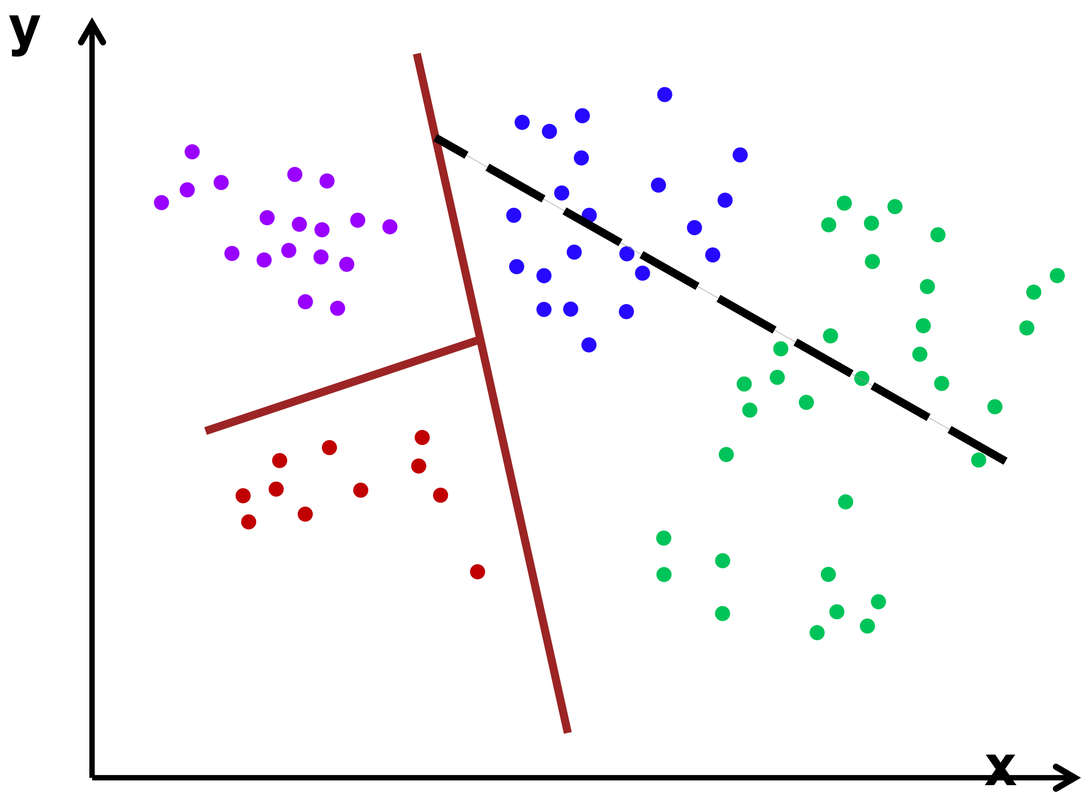
- feature vectors are x, y coordinates: $v = [x, y]^T$
- split functions are lines with parameters a, b: $f_n(v) = ax + by$
- threshold determines intercepts: t_n
- four classes: purple, blue, red, green

Toy Learning Example



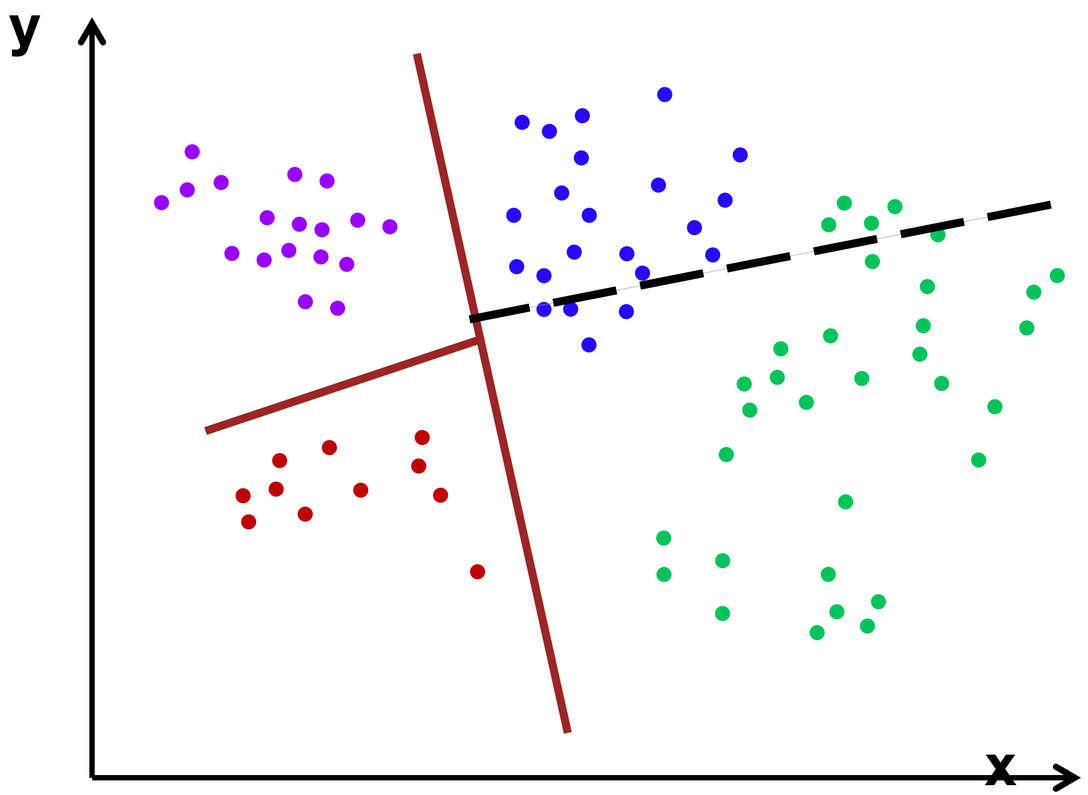
- feature vectors are x, y coordinates: $v = [x, y]^T$
- split functions are lines with parameters a, b: $f_n(v) = ax + by$
- threshold determines intercepts: t_n
- four classes: purple, blue, red, green

Toy Learning Example



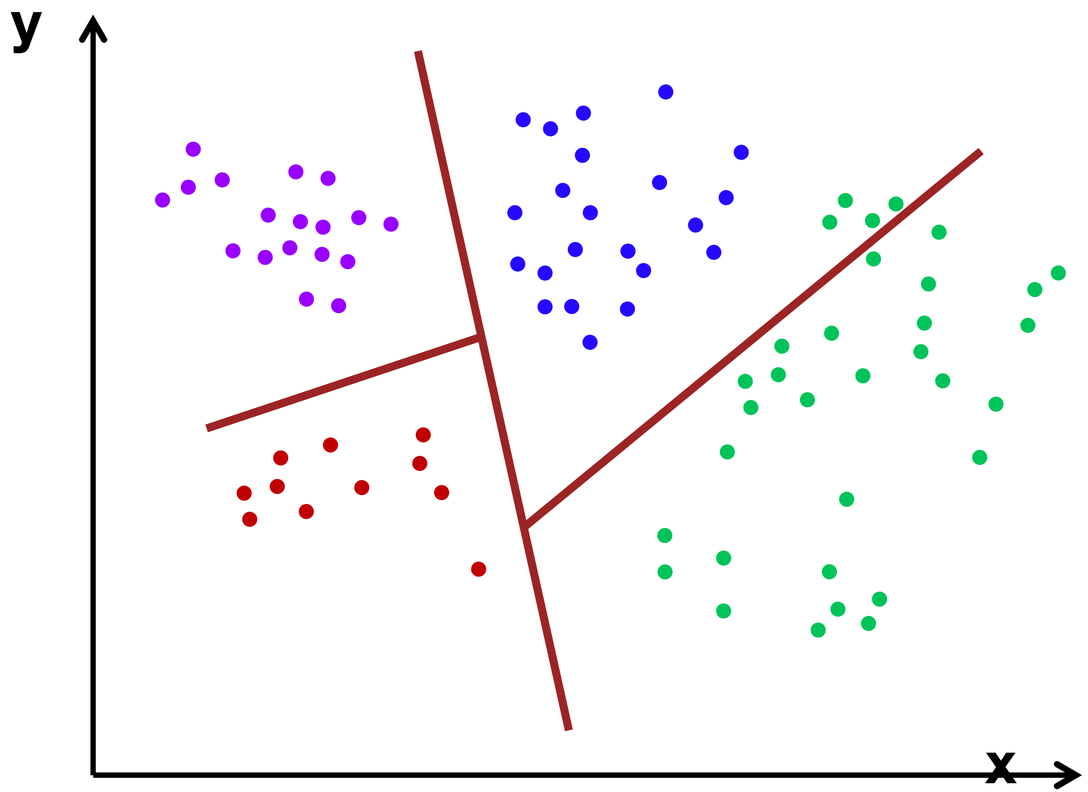
- feature vectors are x, y coordinates: $v = [x, y]^T$
- split functions are lines with parameters a, b: $f_n(v) = ax + by$
- threshold determines intercepts: t_n
- four classes: purple, blue, red, green

Toy Learning Example

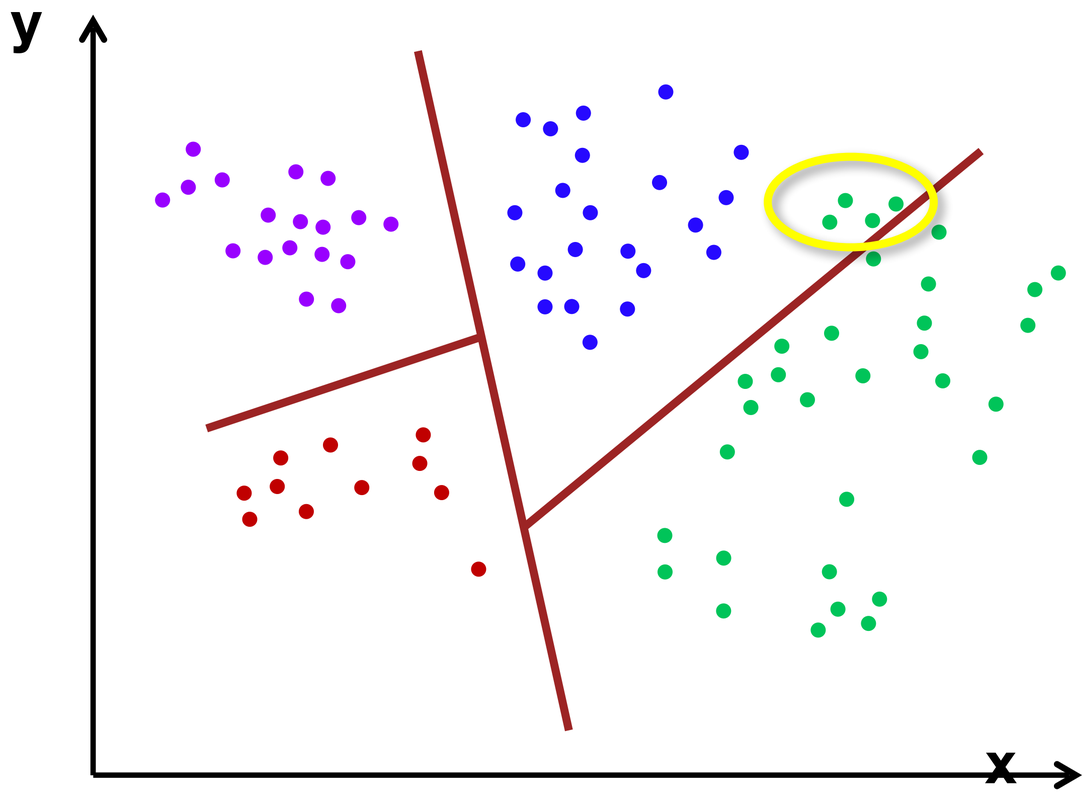


- feature vectors are x, y coordinates: $v = [x, y]^T$
- split functions are lines with parameters a, b: $f_n(v) = ax + by$
- threshold determines intercepts: t_n
- four classes: purple, blue, red, green

Toy Learning Example



Toy Learning Example



Randomized Learning

- Recursively split examples at node n
  - set I_n indexes labeled training examples ($\mathbf{v}_i$, l_i):

left split

$$I_\mathrm{l} = \{i \in I_n \mid f(\mathbf{v}_i) < t\}$$

right split

$$I_\mathrm{r} = I_n \setminus I_\mathrm{l}$$

threshold

function of example i's feature vector

- At node n, $P_n(c)$ is histogram of example labels l_i

More Randomized Learning

- Features f(v) chosen at random from feature pool f in F features set

$$\begin{aligned} \textbf{left split } I_{\mathrm{l}} &= \{i \in I_n \mid f(\mathbf{v}_i) < t\} \\ \textbf{right split } I_{\mathrm{r}} &= I_n \setminus I_{\mathrm{l}} \end{aligned}$$

- Thresholds t chosen in range

$$t \in (\min_i f(\mathbf{v}_i), \max_i f(\mathbf{v}_i))$$

- Choose f and t to maximize gain in information

$$\Delta E = -\frac{|I_{\mathrm{l}}|}{|I_n|} E(I_{\mathrm{l}}) - \frac{|I_{\mathrm{r}}|}{|I_n|} E(I_{\mathrm{r}})$$

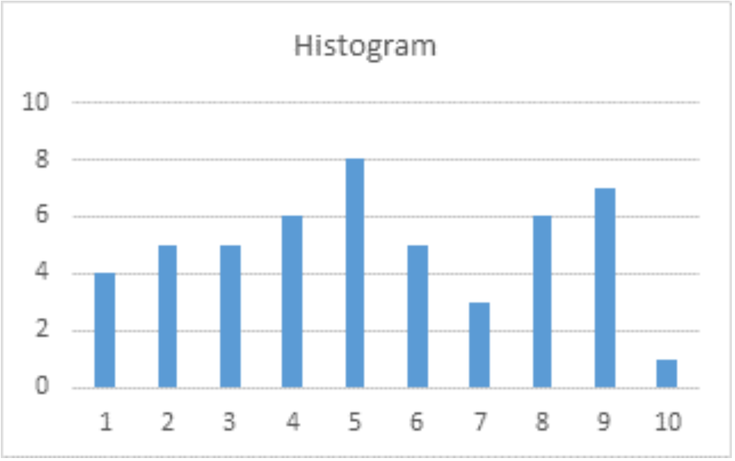
Entropy E calculated from histogram of labels in I

Entropy of a split

- The label histogram is computed for every split I_l and I_r
- Entropy is computed using the histogram on the labels confusion

$$E = -\sum_{i}^{nlabel} h_i log_2(h_i)$$

- where hi is the histogram value



- If all classes have the same probability **Entropy is 1-> Maximum disorder**
- If a class is alone **Entropy is 0 because h_i =1**

- **Maximize the info Gain equal to minimize the entropy of a split**

$$\Delta E = -\frac{|I_\mathrm{l}|}{|I_n|}E(I_\mathrm{l}) - \frac{|I_\mathrm{r}|}{|I_n|}E(I_\mathrm{r})$$

- Mix as few labels as possible

Implementation Details

- How many features and thresholds to try?
  - just one = “extremely randomized” [Geurts *et al.* 06]
  - few -> fast training, may under-fit, maybe too deep
  - many -> slower training, may over-fit

- When to stop growing the tree?
  - maximum depth
  - minimum entropy gain
  - delta class distribution
  - pruning

Randomized Learning Pseudo Code

```
TreeNode LearnDT(I)

    repeat featureTests times
        let f = RndFeature()
        let r = EvaluateFeatureResponses(I, f)

        repeat threshTests times
            let t = RndThreshold(r)
            let (I_l, I_r) = Split(I, r, t)
            let gain = InfoGain(I_l, I_r)
            if gain is best then remember f, t, I_l, I_r
        end
    end

    if best gain is sufficient
        return SplitNode(f, t, LearnDT(I_l), LearnDT(I_r))
    else
        return LeafNode(HistogramExamples(I))
    end
end
```

Binary Decision Trees Summary

- Fast **greedy training algorithms**
  - can search infinite pool of features
  - heterogeneous pool of features

- **Fast testing** algorithm

- Needs **careful choice of hyper-parameters**
  - maximum depth
  - number of features and thresholds to try

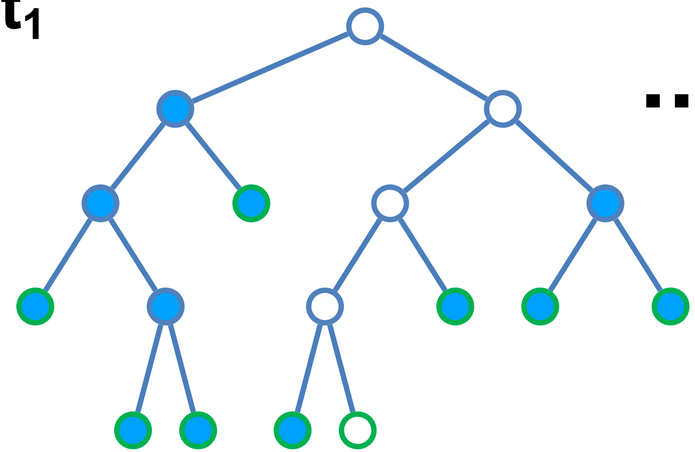
- Prone to over-fitting

A Forest of Trees

leaf nodes

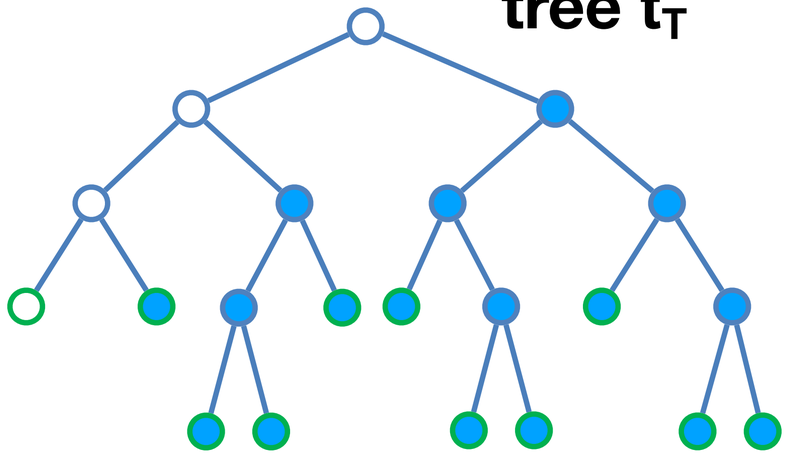
split nodes

tree t_1



.....

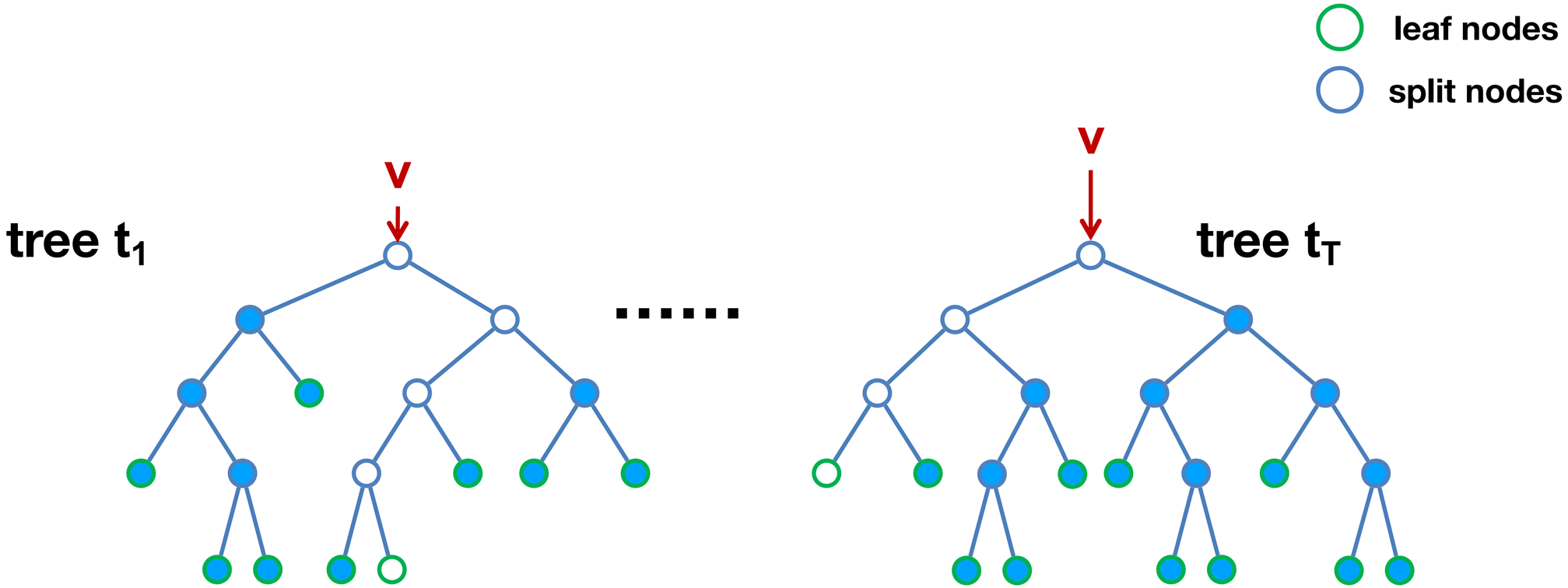
tree t_T



$$\frac{1}{T}$$

[Amit & Geman 97]
[Breiman 01]
[Lepetit *et al.* 06]

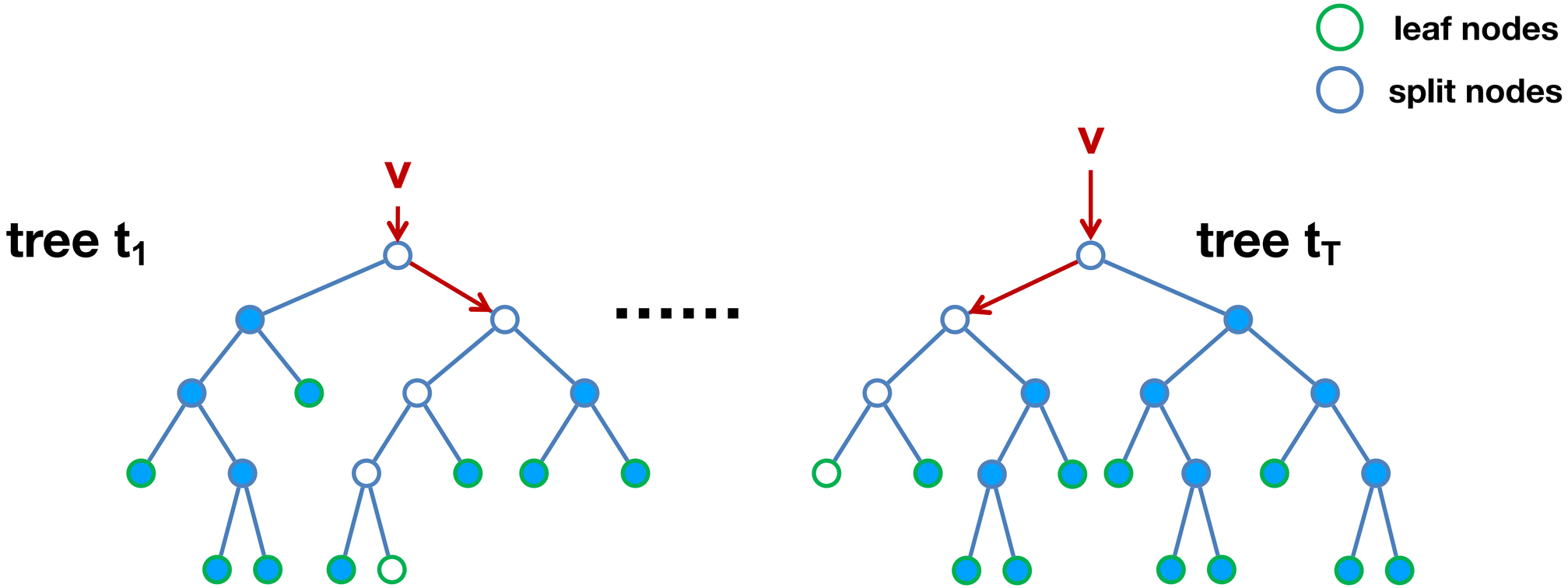
A Forest of Trees



$$\frac{1}{T}$$

[Amit & Geman 97]
[Breiman 01]
[Lepetit *et al.* 06]

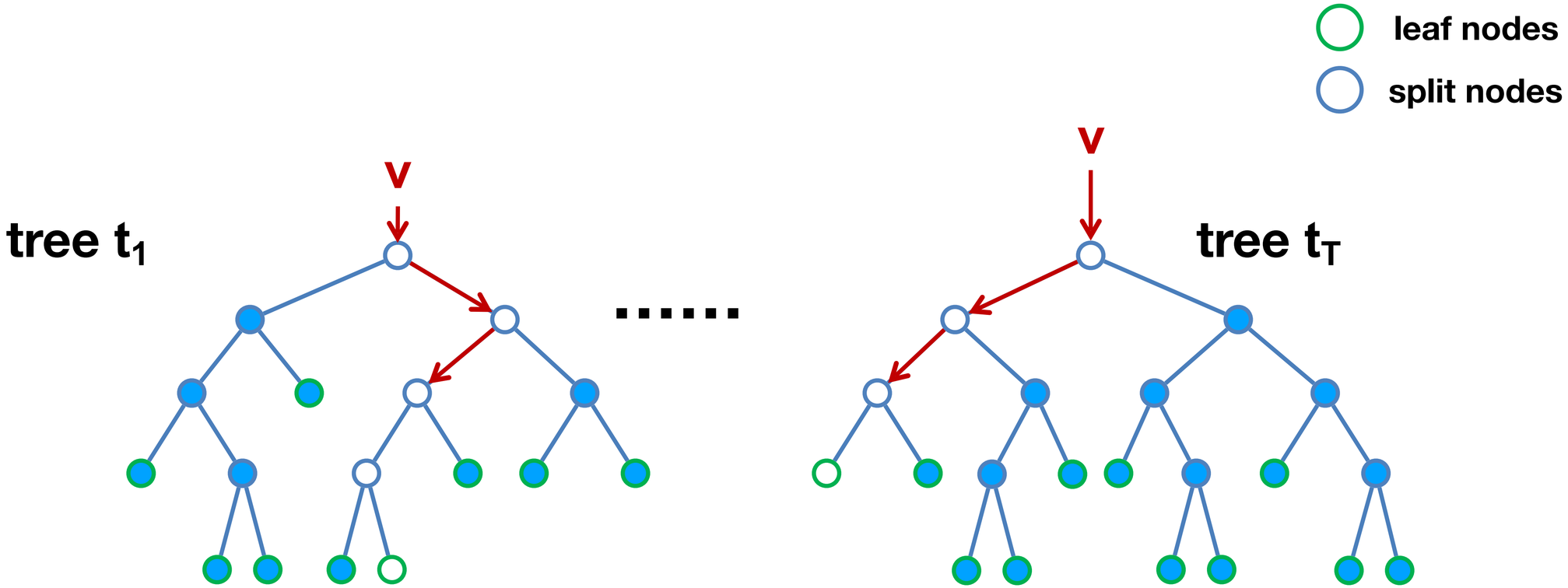
A Forest of Trees



$$\frac{1}{T}$$

[Amit & Geman 97]
[Breiman 01]
[Lepetit *et al.* 06]

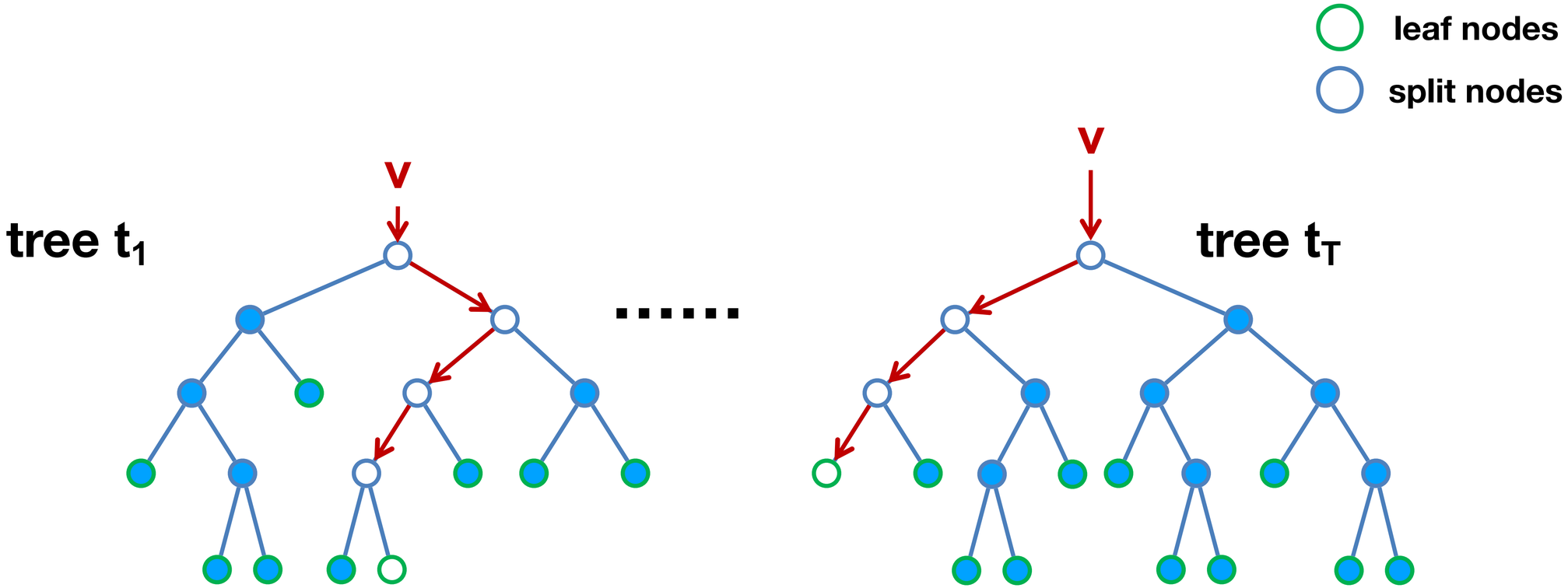
A Forest of Trees



$$\frac{1}{T}$$

[Amit & Geman 97]
[Breiman 01]
[Lepetit *et al*. 06]

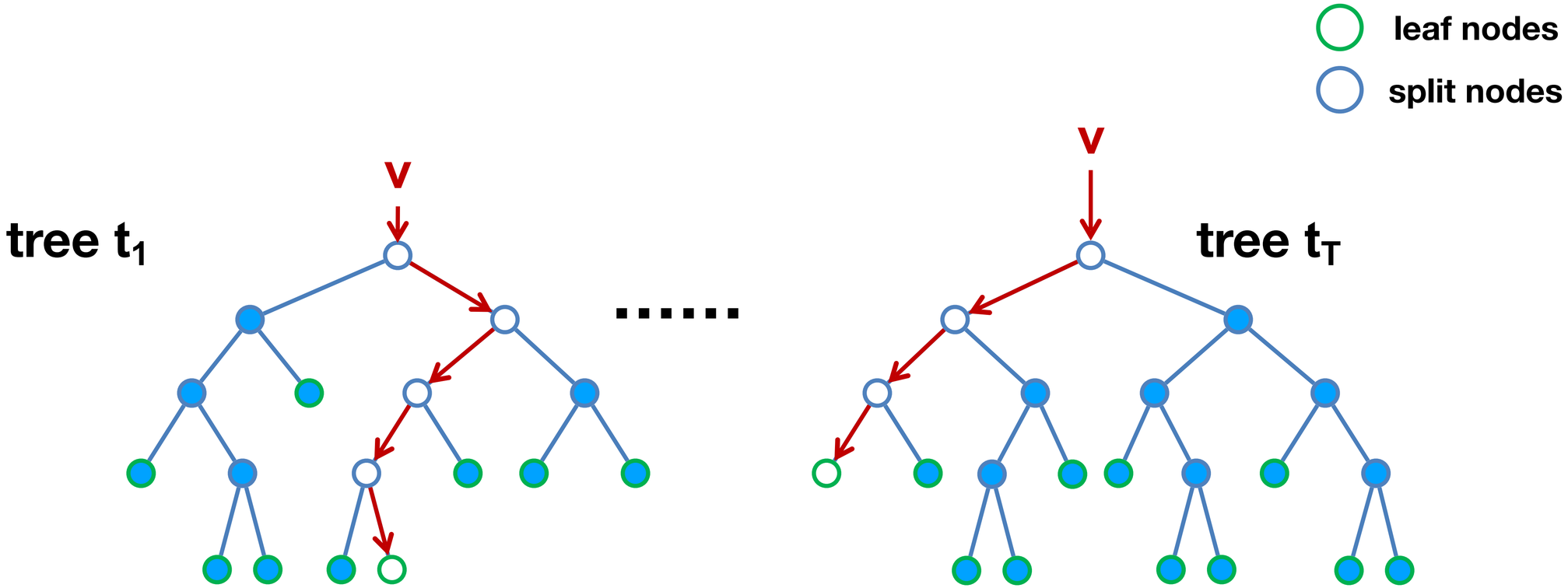
A Forest of Trees



$$\frac{1}{T}$$

[Amit & Geman 97]
[Breiman 01]
[Lepetit *et al*. 06]

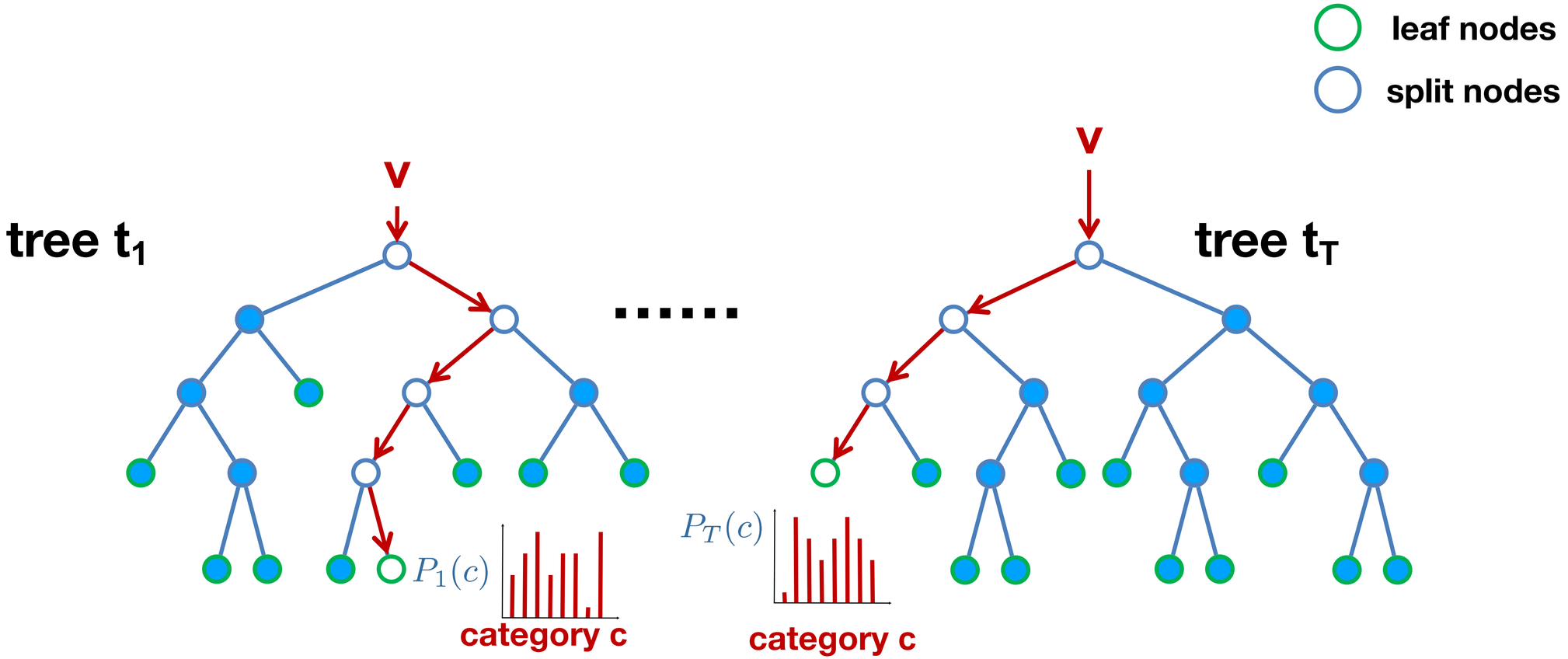
A Forest of Trees



$$\frac{1}{T}$$

[Amit & Geman 97]
[Breiman 01]
[Lepetit *et al*. 06]

A Forest of Trees

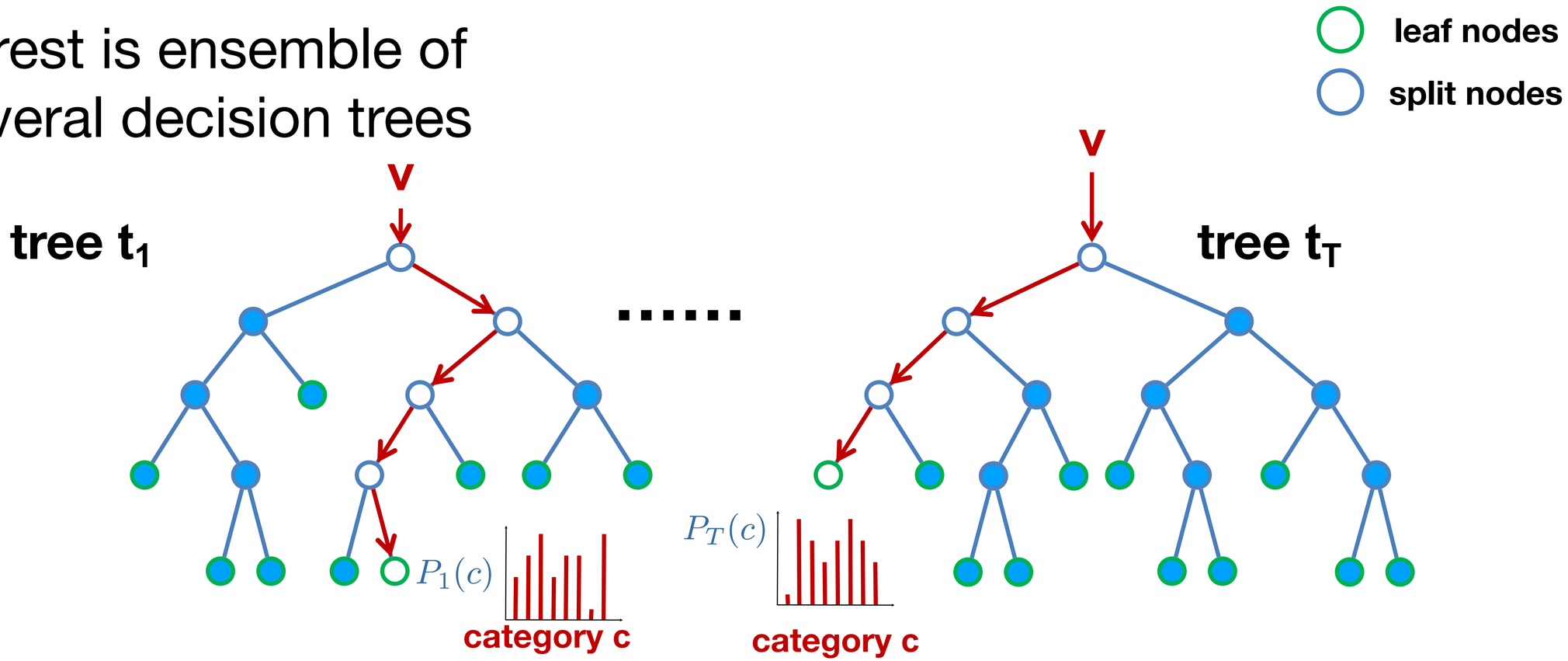


$$\frac{1}{T}$$

[Amit & Geman 97]
[Breiman 01]
[Lepetit *et al*. 06]

A Forest of Trees

- Forest is ensemble of several decision trees



- classification is

$$P(c|\mathbf{v}) = \frac{1}{T}\sum_{t=1}^{T} P_t(c|\mathbf{v})$$

[Amit & Geman 97]
[Breiman 01]
[Lepetit *et al*. 06]

Decision Forests Pseudo-Code

```
double[] ClassifyDF(forest, v)
        // allocate memory
        let P = double[forest.CountClasses]

        // loop over trees in forest
        for t = 1 to forest.CountTrees
              let P' = ClassifyDT(forest.Tree[t], v)
              P = P + P' // sum distributions
        end

        // normalise
        P = P / forest.CountTrees
end
```

Learning a Forest

- Divide training examples into T subsets I_t t=1…T
  - improves generalization
  - reduces memory requirements & training time

- Train each decision tree t on subset I_t
  - same decision tree learning as before

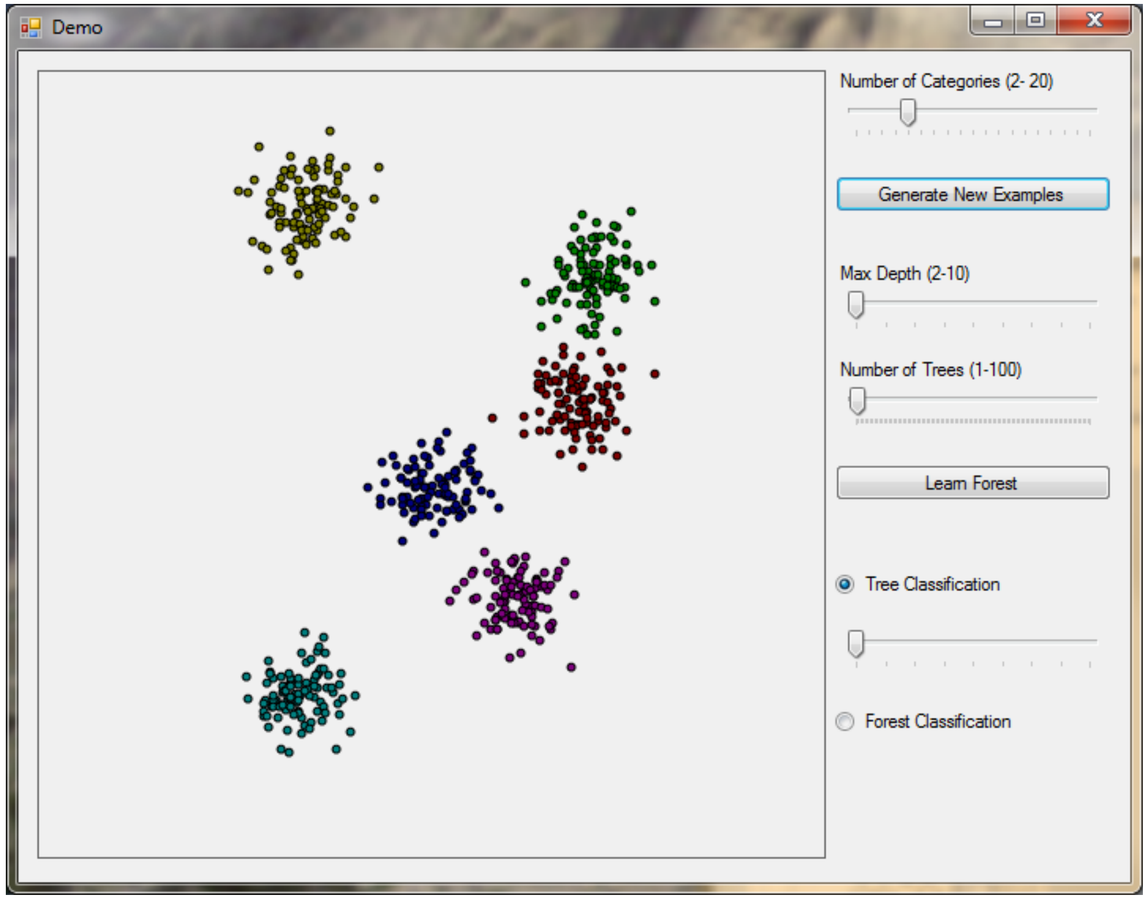
  - **Subsets can be chosen at random or hand-picked**
  - **Subsets can have overlap (and usually do)**
  - **Can enforce subsets of *images* (not just examples)**
  - **Could also divide the feature pool into subsets**

- **Multi-core friendly**

Learning a Forest Pseudo Code

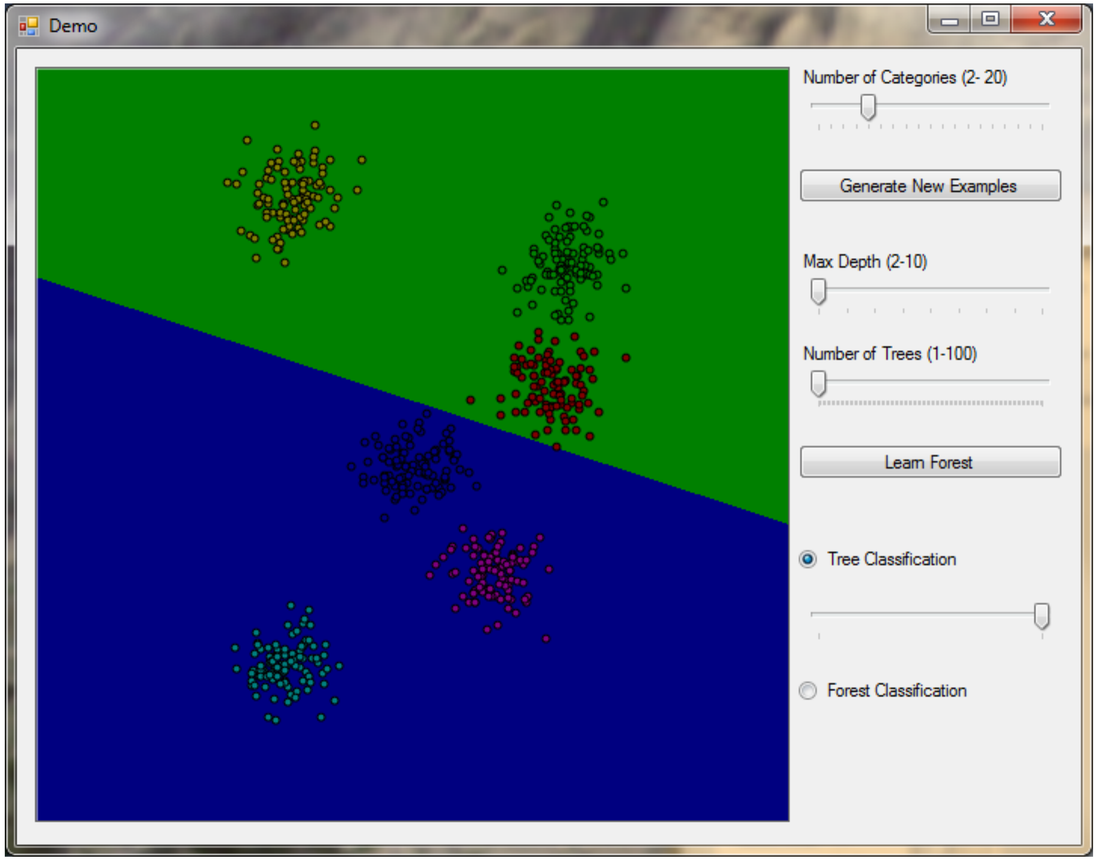
```
Forest LearnDF(countTrees, I)
        // allocate memory
        let forest = Forest(countTrees)

        // loop over trees in forest
        for t = 1 to countTrees
              let I_t = RandomSplit(I)
              forest[t] = LearnDT(I_t)
        end

        // return forest object
        return forest
end
```

Toy Forest Classification Demo



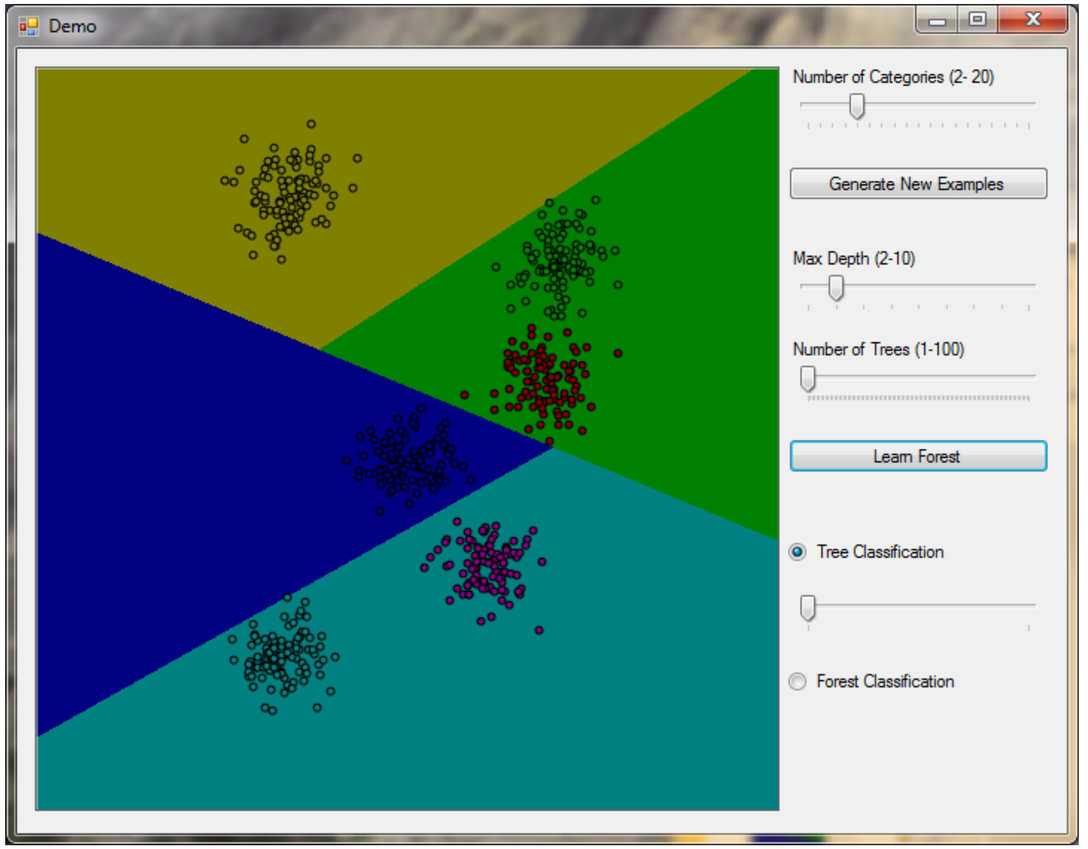
6 classes in a 2 dimensional feature space.
Split functions are lines in this space.

Toy Forest Classification Demo



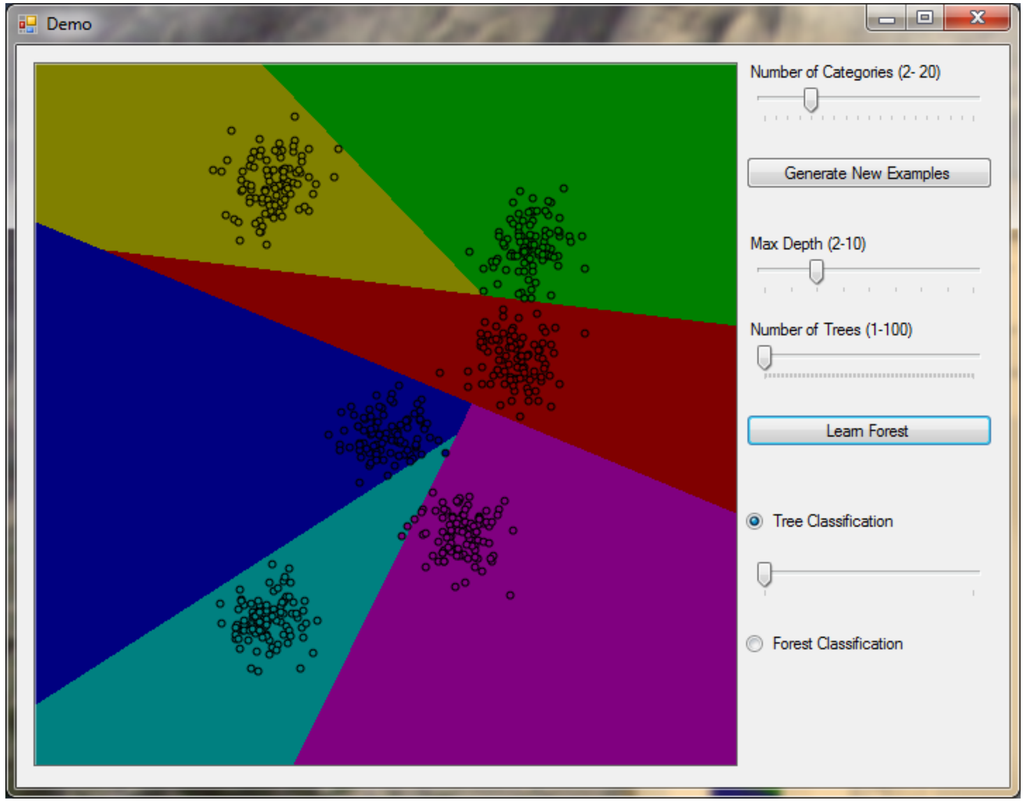
With a depth 2 tree, you cannot separate all six classes.

Toy Forest Classification Demo



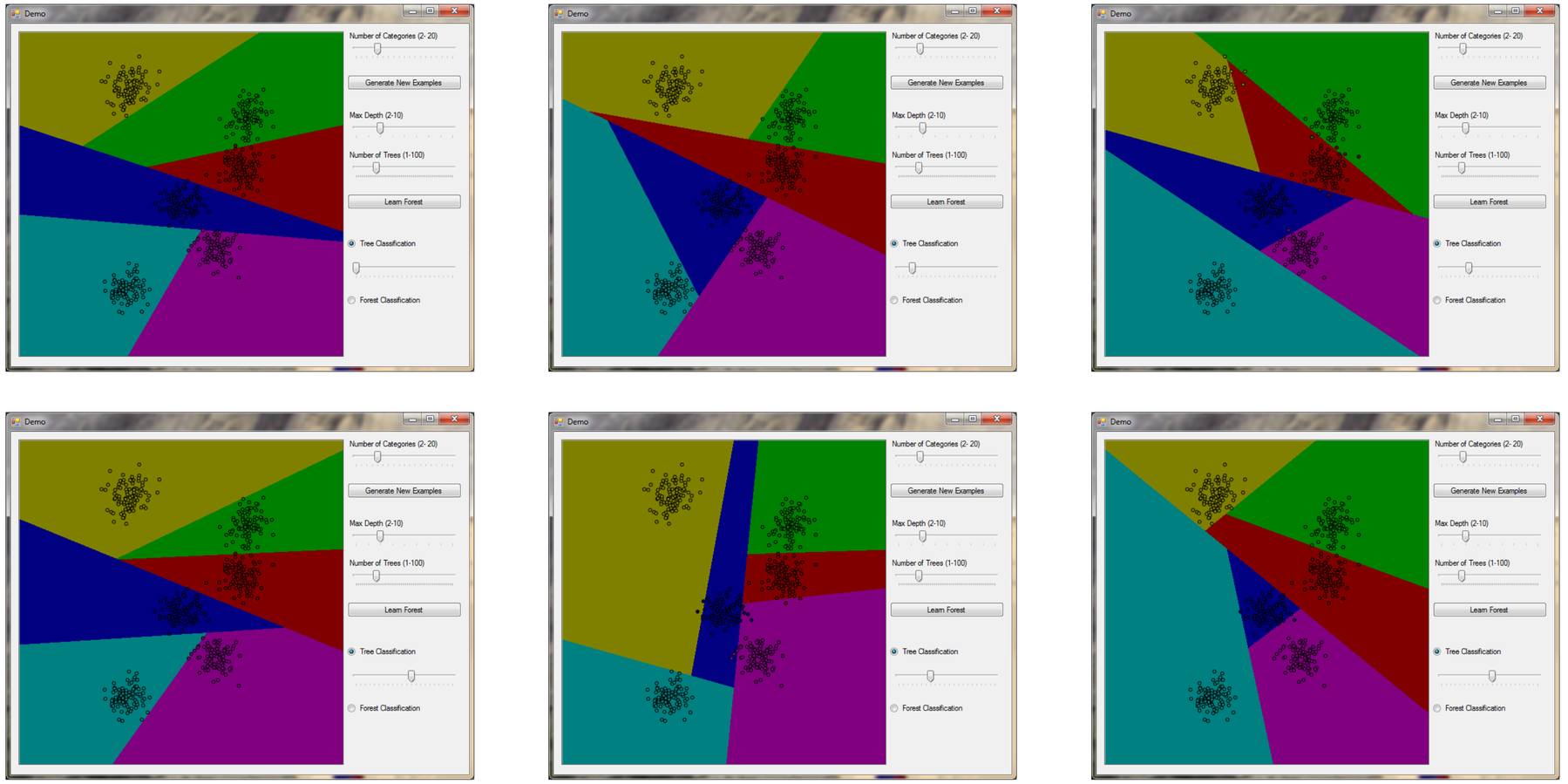
With a depth 3 tree, you are doing better, but still cannot separate all six classes.

Toy Forest Classification Demo



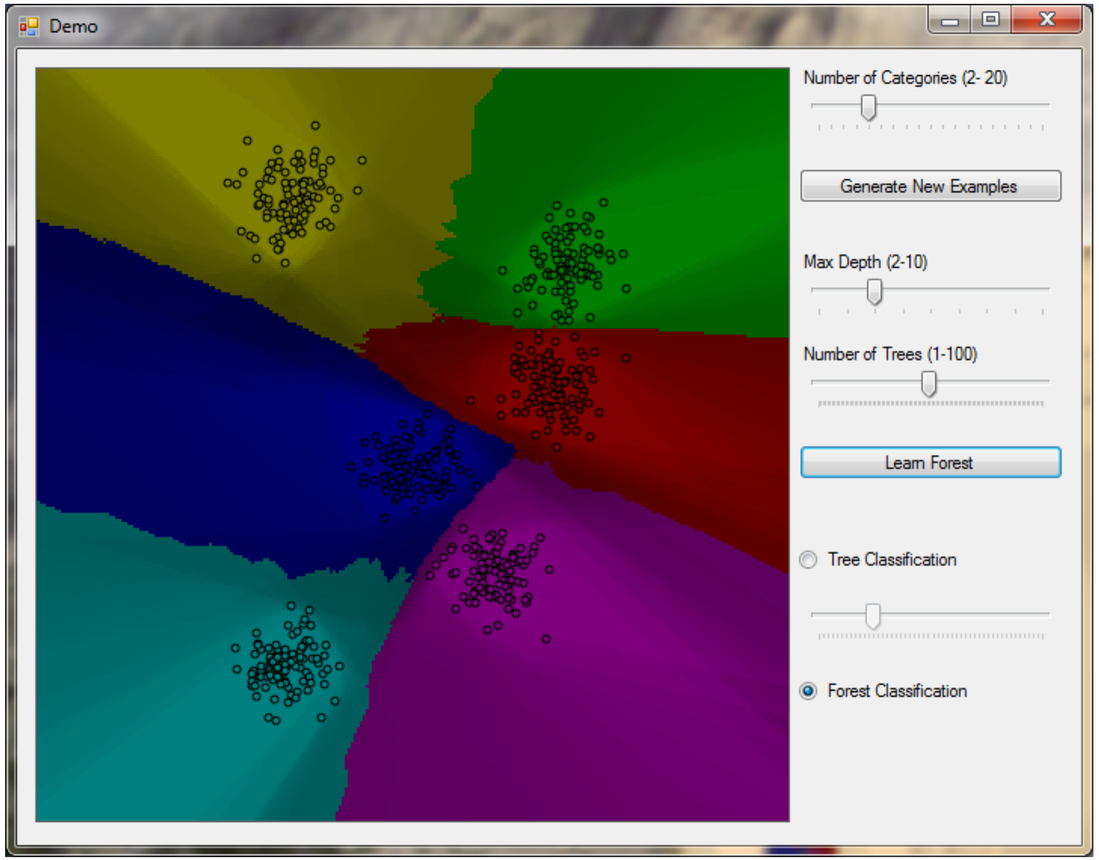
With a depth 4 tree, you now have at least as many leaf nodes as classes, and so are able to classify most examples correctly.

Toy Forest Classification Demo



Different trees within a forest can give rise to very different decision boundaries, none of which is particularly good on its own.

Toy Forest Classification Demo

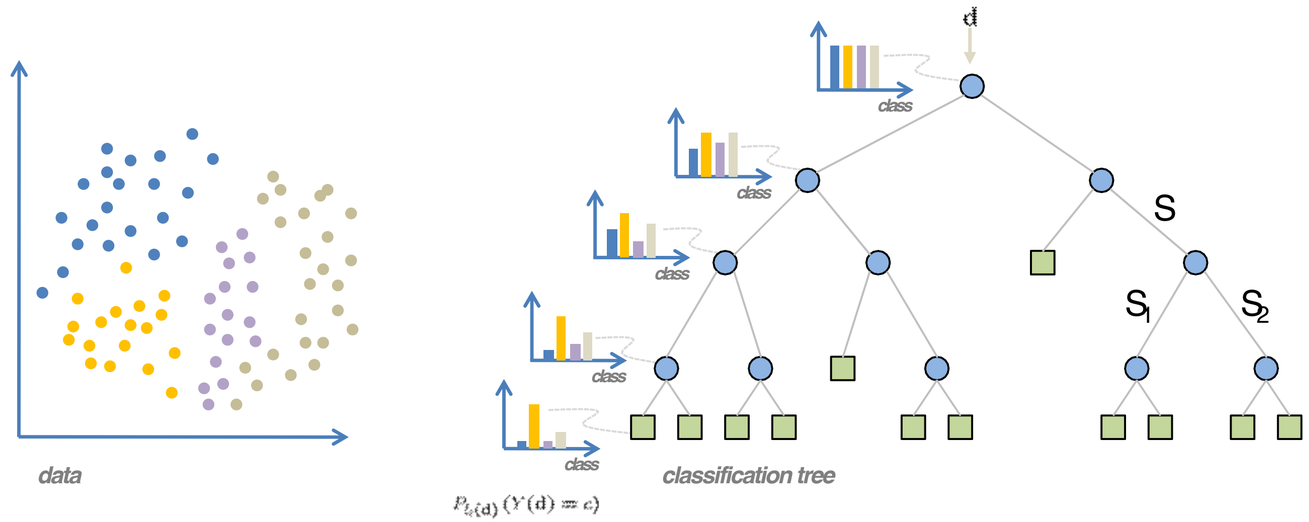


But averaging together many trees in a forest can result in decision boundaries that look very sensible, and are even quite close to the max margin classifier. (Shading represents entropy – darker is higher entropy).

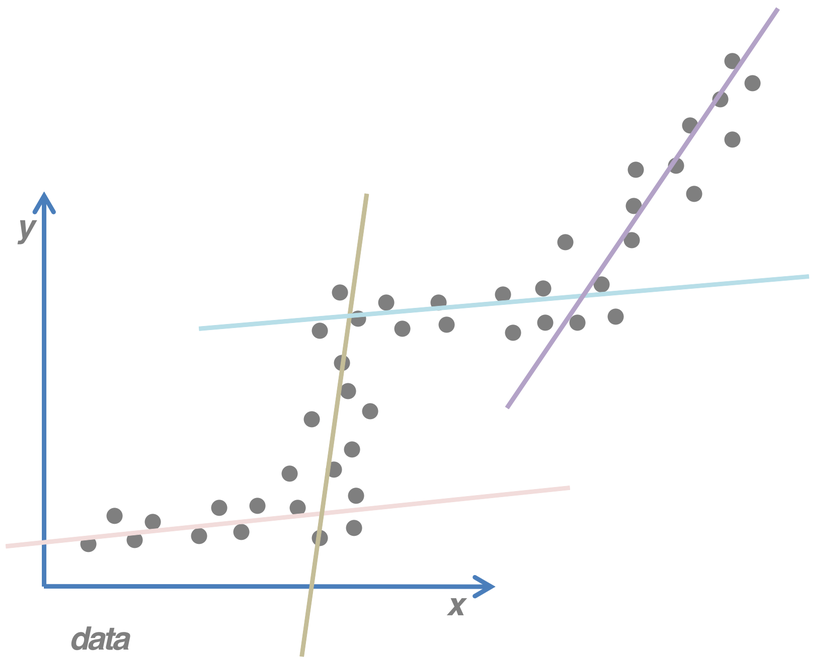
Tree outputs and objective functions

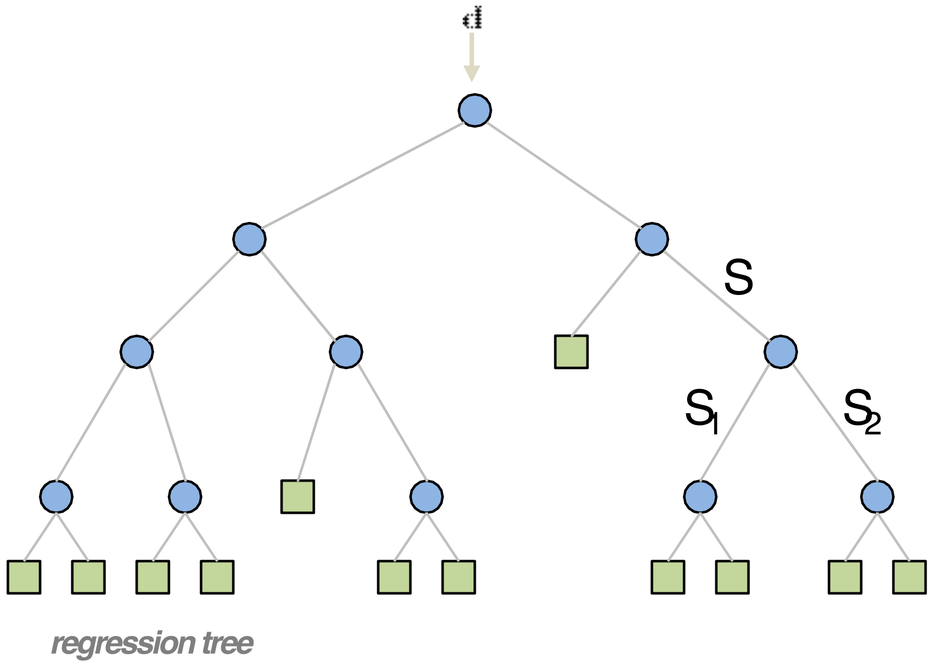
- Trees can be trained for
  - classification, regression, or clustering

- Change the object function
  - information gain for classification: $I = H(S) - \sum_{i=1}^{2} \frac{|S_i|}{|S|} H(S_i)$ measure of distribution purity



Regression trees





regression tree

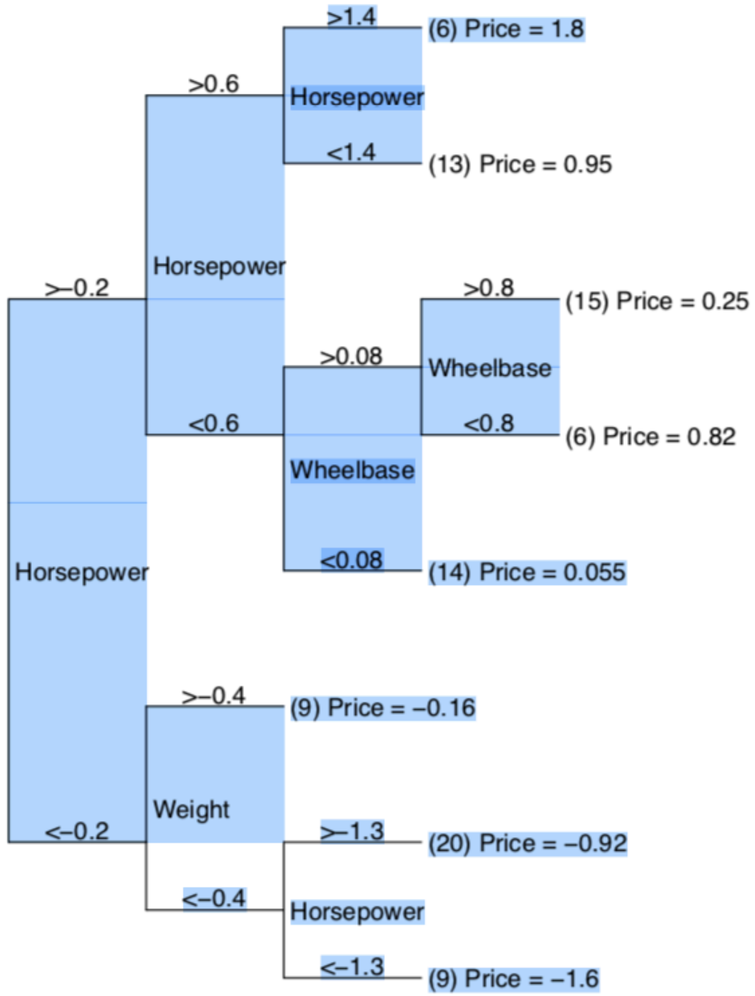
- **Real-valued output *y***
- **Object function: maximize** measure of fit of model

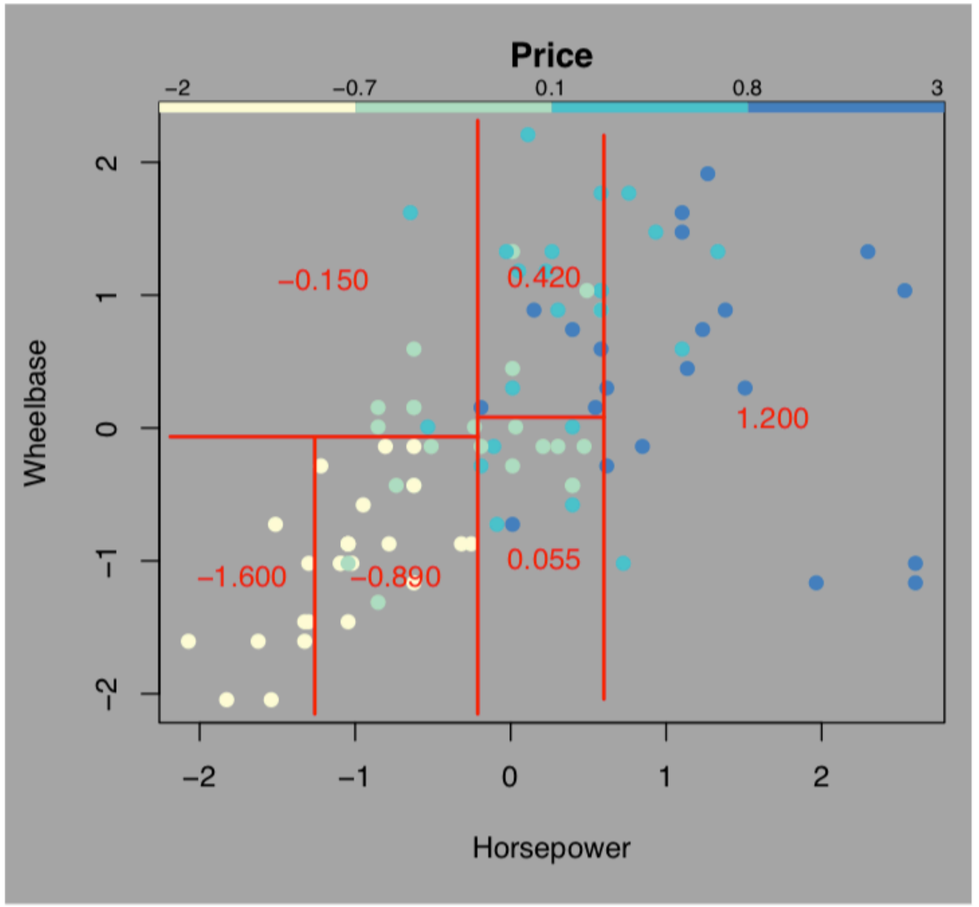
$$Err(S) - \sum_{i=1}^{2} \frac{|S_i|}{|S|} Err(S_i)$$

$$Err(S) = \sum_{j \in S} \left(y_j - y(x_j)\right)^2$$

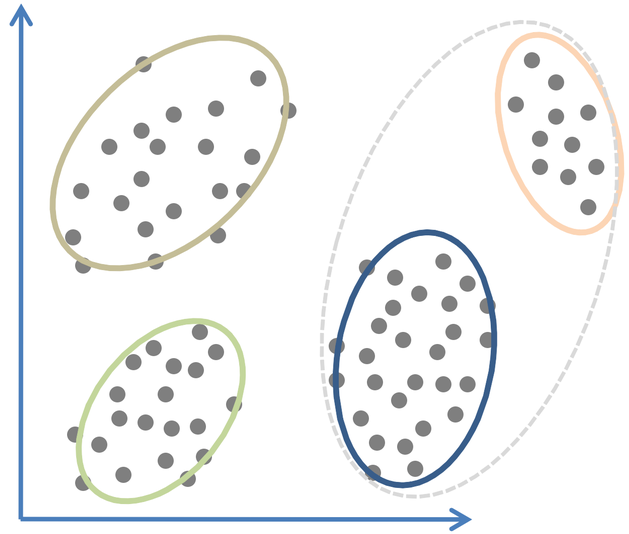
e.g. linear model y = ax+b,
Or just constant model

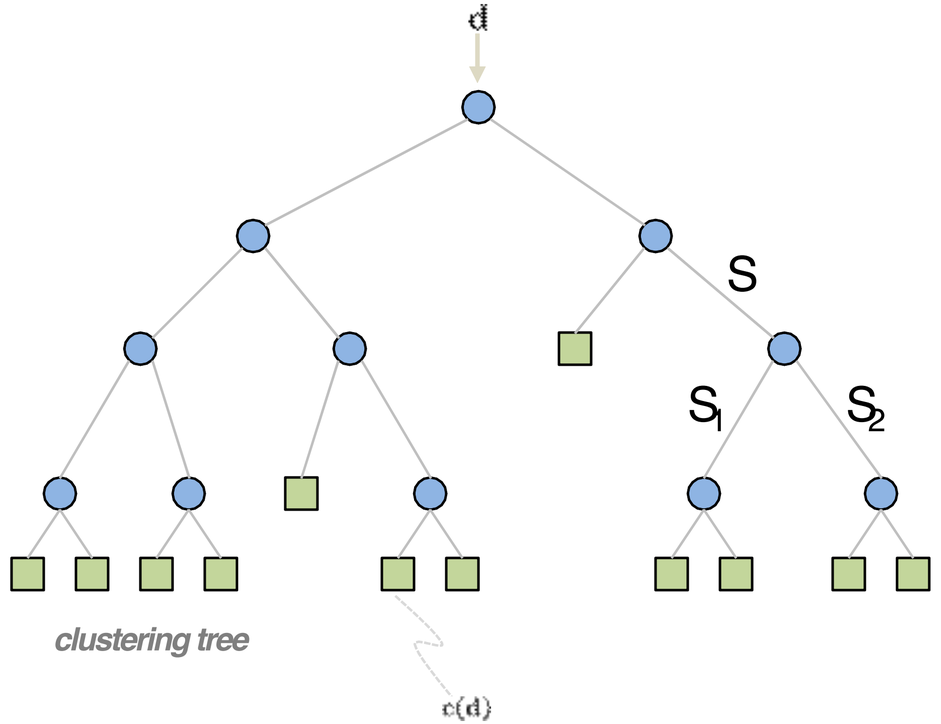
Regression trees





Clustering trees





clustering tree

- **Output is cluster membership**
- **Option 1 – minimize imbalance:** $B = |\log|S_1| - \log|S_2||$
- **Option 2 – maximize Gaussian likelihood:** $T = |\Lambda_S| - \sum_{i=1}^{2} \frac{|S_i|}{|S|} |\Lambda_{S_i}|$

measure of cluster tightness
(maximizing a function of info gain
for Gaussian distributions)

[Moosmann *et al.* 06]

Clustering example

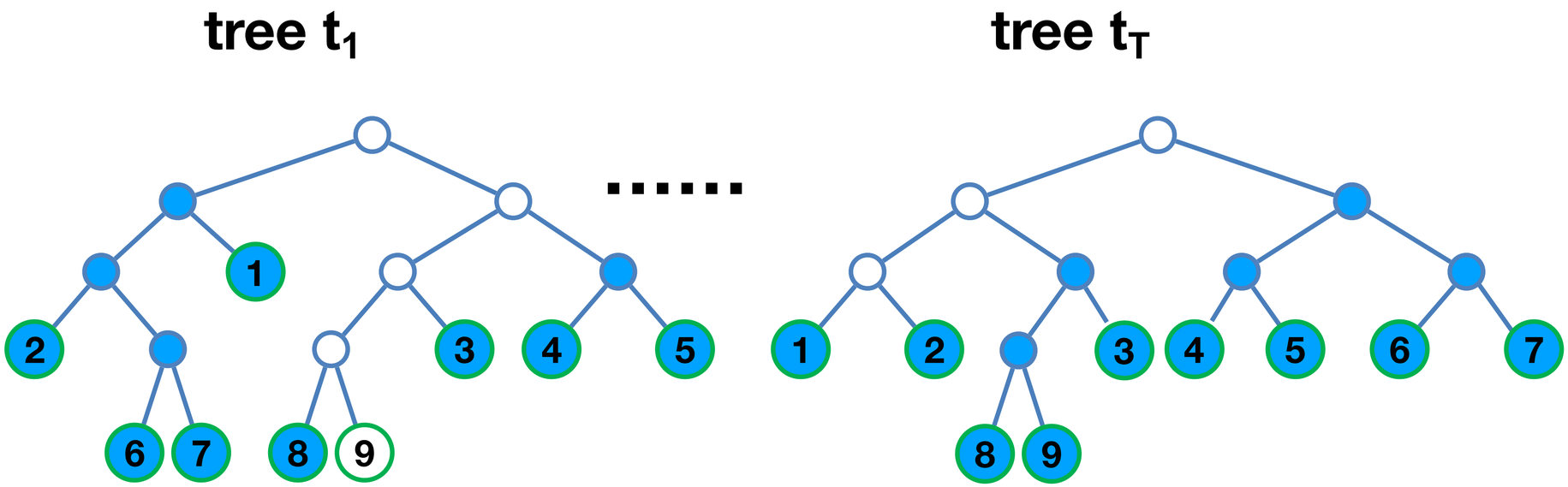
[Moosmann *et al.* 06]

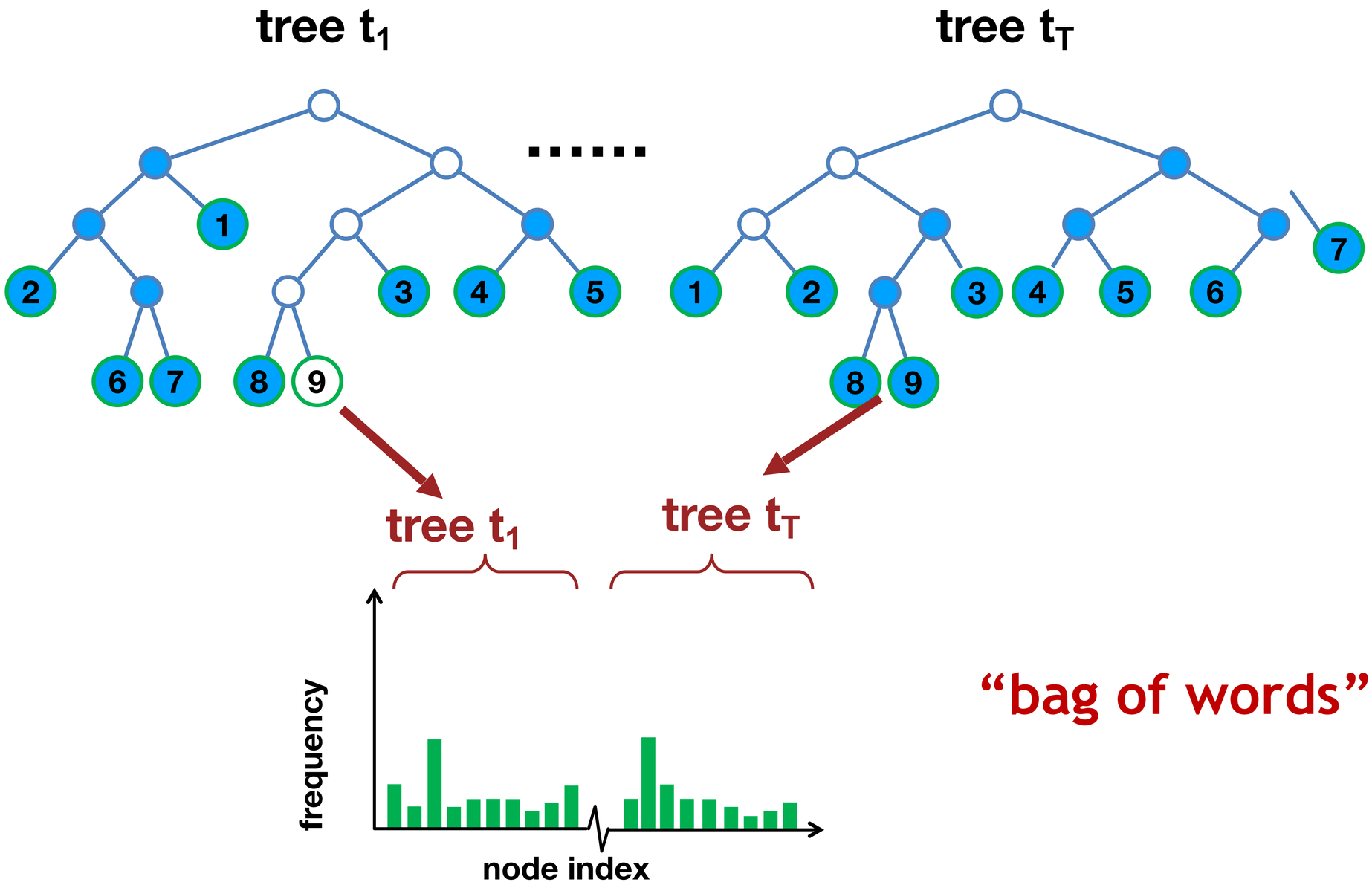
[Sivic *et al*. 03]
[Csurka *et al*. 04]

- Visual words good for e.g. matching, recognition

  but *k*-means clustering very slow

- Randomized forests for clustering descriptors
  - e.g. SIFT, texton filter-banks, etc.
- **Leaf nodes in forest are clusters**
  - concatenate histograms from trees in forest




tree t_1
tree t_T
.....
1
2
3
4
5
6
7
8
9
1
2
3
4
5
6
7
8
9
tree t_1
tree t_T
frequency
node index

"bag of words"

Real-Time Object Segmentation [Shotton *et al.* 2008]

Real-Time Object Segmentation [Shotton *et al.* 2008]

Take Home Message

- Randomized decision forests
  - very fast
  - accuracy comparable with other classifiers
  - simple to implement
  - extremely flexible tools for computer vision

Web Resources on Random Forests

- Tutorial Webpage
  - http://mi.eng.cam.ac.uk/~tkk22/iccv09_tutorial

- Leo Breiman's Webpage
  - http://www.stat.berkeley.edu/~breiman/RandomForests

- Regression Trees
  - http://www.stat.cmu.edu/~cshalizi/350-2006/lecture-10.pdf

References (red = most relevant)

- Geurts *et al.*
  - Extremely Randomized Trees.
  - Machine Learning 2006.

- Grauman & Darrel
  - The Pyramid Match Kernel: Discriminative Classification with Sets of Image Features.
  - ICCV 2005.

- Hua *et al.*
  - Discriminant Embedding for Local Image Descriptors.
  - ICCV 2007.

- Jurie & Triggs
  - Creating Efficient Codebooks for Visual Recognition.
  - ICCV 2005.

- Lazebnik *et al.*
  - Beyond Bags of Features: Spatial Pyramid Matching for Recognizing Natural Scene Categories.
  - CVPR 2006.

- Lepetit *et al.*
  - Keypoint Recognition using Randomized Trees.
  - PAMI 2006.

- Özuysal *et al.*
  - Fast Keypoint Recognition in Ten Lines of Code.
  - CVPR 2007.
- Rogez *et al.*
  - Randomized Trees for Human Pose Detection.
  - CVPR 2008.
- Sharp
  - Implementing Decision Trees and Forests on a GPU.
  - ECCV 2008.
- Shotton *et al.*
  - Semantic Texton Forests for Image Categorization and Segmentation.
  - CVPR 2008.
- Shotton *et al.*
  - TextonBoost for Image Understanding: Multi-Class Object Recognition and Segmentation by Jointly Modeling Texture, Layout, and Context.
  - IJCV 2007.
- Sivic & Zisserman
  - Video Google: A Text Retrieval Approach to Object Matching in Videos.
  - ICCV 2003.
- Tibshirani & Hastie
  - Margin trees for high-dimensional classification.
  - JMLR 2007.
- Torralba *et al.*
  - Sharing visual features for multiclass and multiview object detection.
  - PAMI 2007.
- Tu
  - Probabilistic Boosting-Tree: Learning Discriminative Models for Classification, Recognition, and Clustering.
  - ICCV 2005.
- Tu
  - Auto-context and Its application to High-level Vision Tasks.
  - CVPR 2008.
- Tuytelaars & Schmid
  - Vector Quantizing Feature Space with a Regular Lattice.
  - ICCV 2007.
- Varma & Zisserman
  - A statistical approach to texture classification from single images.
  - IJCV 2005.
- Verbeek & Triggs
  - Region Classification with Markov Field Aspect Models.
  - CVPR 2007.
- Viola & Jones
  - Robust Real-time Object Detection.
  - IJCV 2004.
- Winn *et al.*
  - Object Categorization by Learned Universal Visual Dictionary.
  - ICCV 2005.
- Wu *et al.*
  - Enlarging the Margins in Perceptron Decision Trees.
  - Machine Learning 2000.
- Yeh *et al.*
  - Adaptive Vocabulary Forests for Dynamic Indexing and Categry Learning.