CS 1675 Machine Learning
Lecture 15

Multiclass classification (cont) + Decision trees

Milos Hauskrecht
milos@cs.pitt.edu
5329 Sennott Square

Multiclass classification

- **Binary classification:**
  - **Number of classes = 2**
  - **A special case of multiclass classification**

Multiclass classification
  - **Number of classes is > 2**

Discriminative approach

- **Parametric models** of discriminant functions:
  - $g_0(x)$, $g_1(x)$, .. $g_{K-1}(x)$
- Learns the discriminant functions directly

Key issues:

- How to design the discriminant functions?
- How to train them?

Another question:

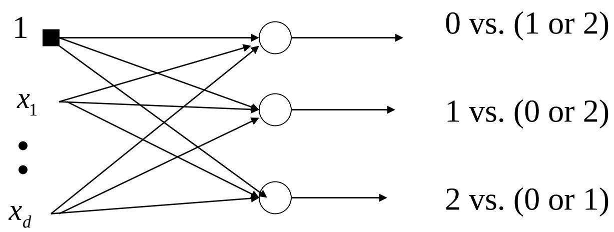
- Can we use binary classifiers and their class outputs to build the multi-class models?

One versus the rest (OVR)

Methods based on binary classification methods

- **Assume:** we have 3 classes labeled 0,1,2
- **Approach 1:**

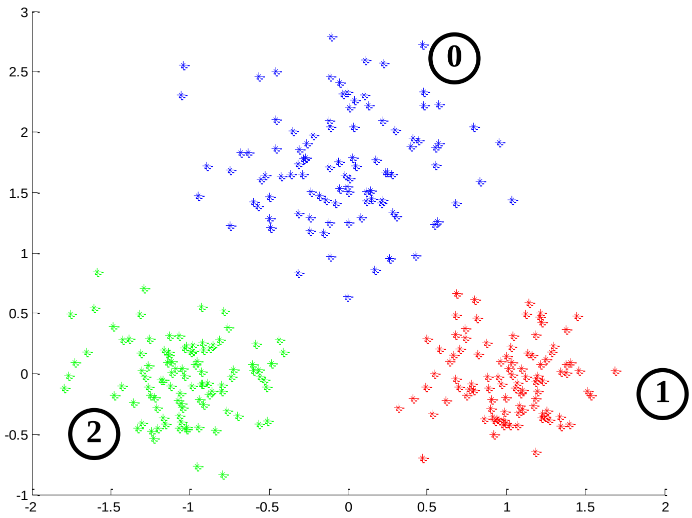
  A binary logistic regression on every class versus the rest (OvR)



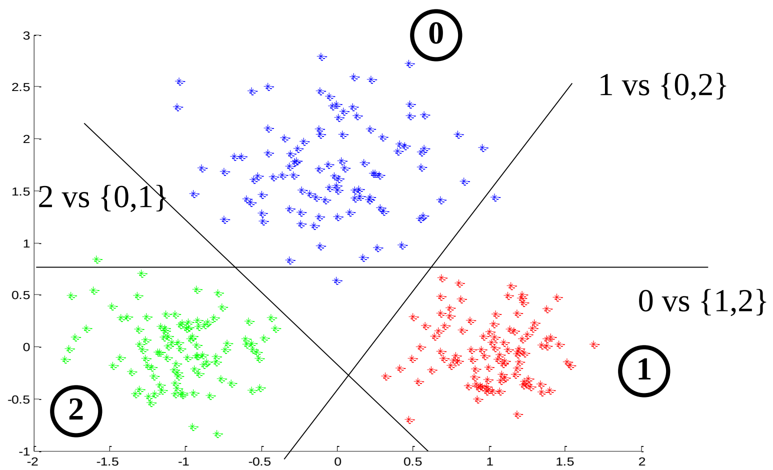
Class decision: class label for a 'singleton' class

– Does not work all the time

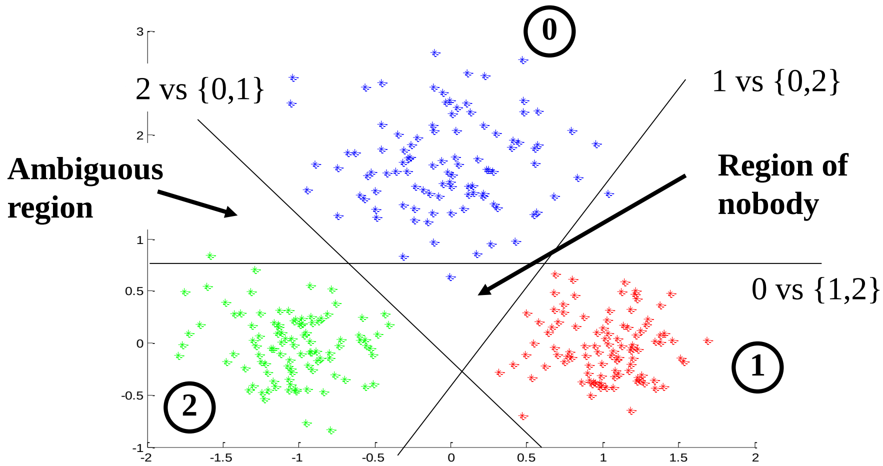
Multiclass classification. Example



Multiclass classification. Approach 1.



Multiclass classification. Approach 1.



One versus the rest (OVR)

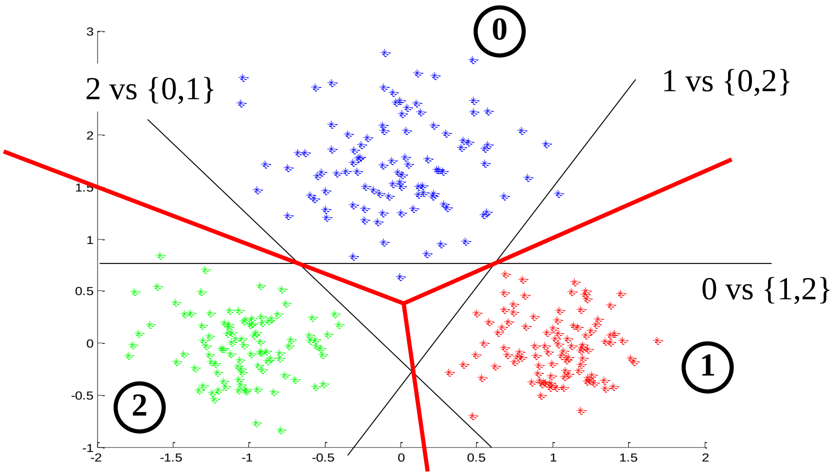
Unclear how to decide on what class to choose in some regions

- **Ambiguous region:**
  - 0 vs. (1 or 2) classifier says 0
  - 1 vs. (0 or 2) classifier says 1
- **Region of nobody:**
  - 0 vs. (1 or 2) classifier says (1 or 2)
  - 1 vs. (0 or 2) classifier says (0 or 2)
  - 2 vs (1 or 2) classifier says (1 or 2)

One solution: Use discriminant functions from binary models

- compare discriminant functions defined on binary classifiers for single option: $g_i(x) = g_{i\,vs\,rest}(x)$
  - discriminant function for *i* trained on *i* vs. *rest*

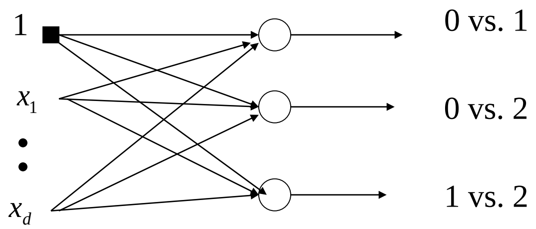
Multiclass classification. Approach 1.



One vs One (OVO)

Methods based on binary classification methods

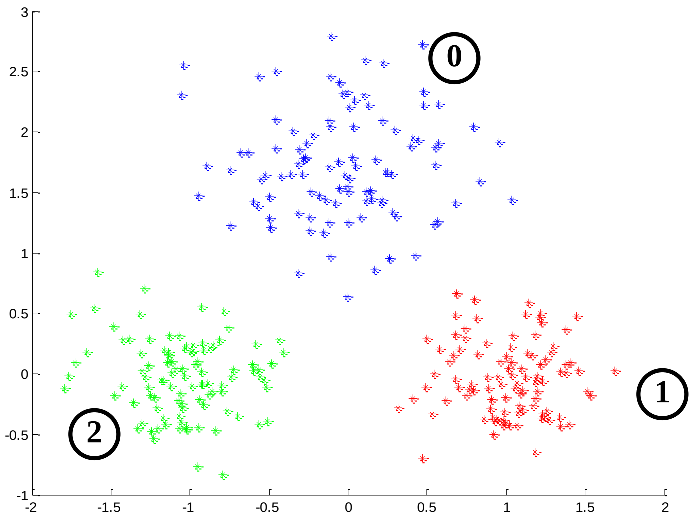
- **Assume:** we have 3 classes labeled 0,1,2
- **Approach 2:**
  - A binary logistic regression on all pairs



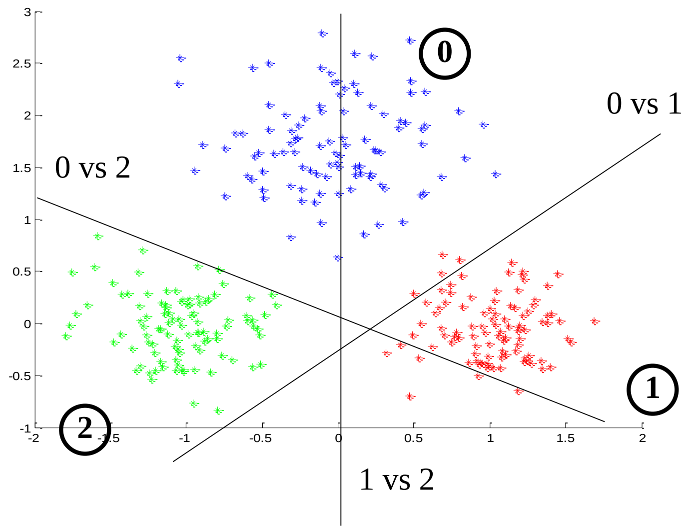
Class decision: class label based on who gets the majority

- Does not work all the time

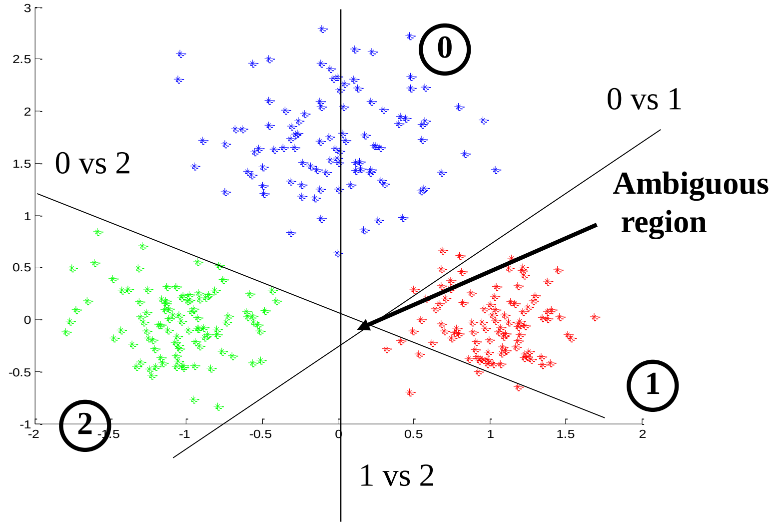
Multiclass classification. Example



Multiclass classification (OVO)



Multiclass classification OVO



One vs one (OVO) model

Unclear how to decide on what class to choose in some regions

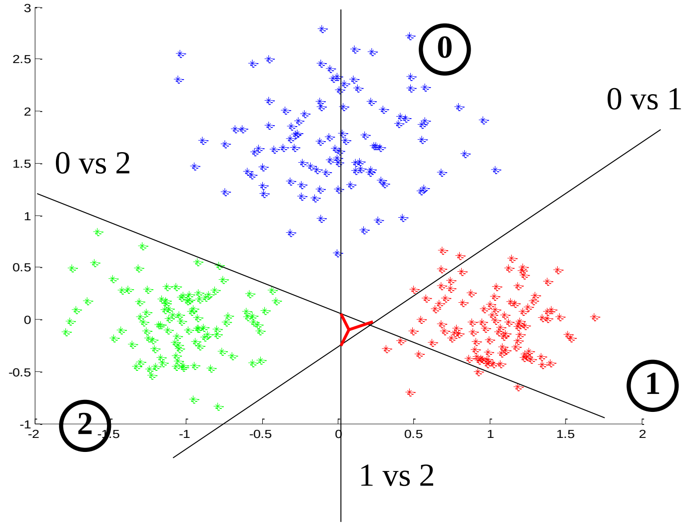
- **Ambiguous region:**
  - 0 vs. 1 classifier says 0
  - 1 vs. 2 classifier says 1
  - 2 vs. 0 classifier says 2

One possible solution:

- Use discriminant functions from binary models
- Define a new discriminant function by adding the discriminant functions for pairwise classifiers

$$g_i(x) = \sum_{j \neq i} g_{i\,vs\,j}(x)$$

Multiclass classification.



Multiclass classification

OVR and OVO:

- define multiclass classifier using output classes of binary classifiers

Problems: ambiguous regions, regions of nobody

Solution: define discriminant functions for the multiclass case using the discriminant functions from binary classification problems

A Concern:

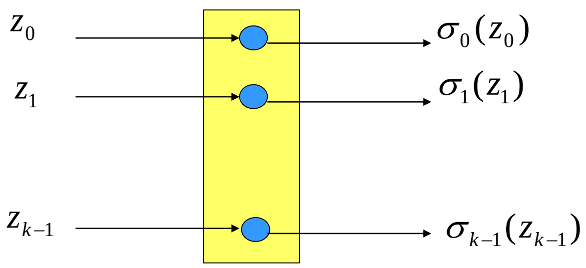
- Calibration of the discriminant functions
  - Discriminant functions from independently trained binary classification models may not be directly comparable

Solution:

- joint learning of discriminant functions

Softmax function

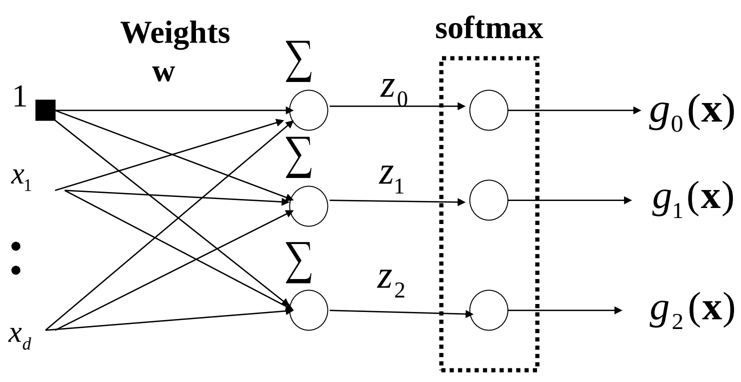
- **Multiple inputs → outputs probabilities**



$$\sigma_i(z_i) = \frac{\exp(z_i)}{\sum_{j=0}^{k-1} \exp(z_j)} \qquad \sum_{i=0}^{k-1} \sigma_i(z_i) = 1$$

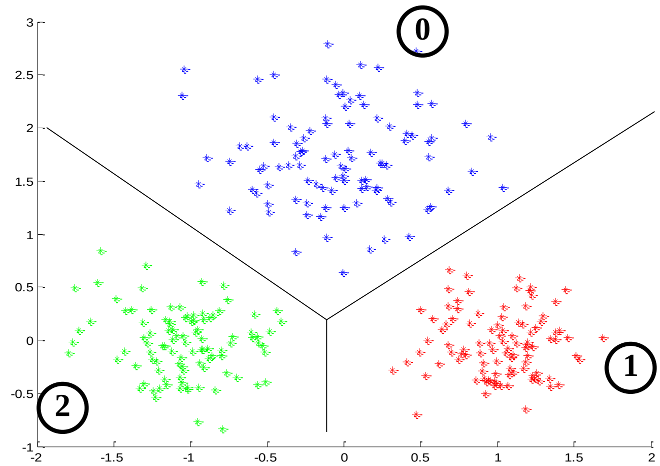
Multiclass classification with softmax

- **Multiclass discriminant functions (they are related via softmax)**



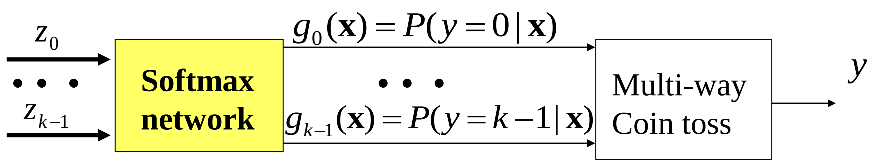
$$g_i(\mathbf{x}) = p(y = i \mid \mathbf{x}) = \frac{\exp(\mathbf{w}_i^T \mathbf{x})}{\sum_{j=0}^{k-1} \exp(\mathbf{w}_j^T \mathbf{x})} \qquad \sum_i g_i(\mathbf{x}) = 1$$

Multiclass classification with softmax



Learning of the softmax model

- Learning of parameters **w**: statistical view



Assume outputs y are transformed to one hot vectors:

$$y \in \{0 \quad 1 \quad .. \quad k-1\} \quad \Rightarrow \quad y \in \left\{ \begin{pmatrix} 1 \\ 0 \\ .. \\ 0 \end{pmatrix} \begin{pmatrix} 0 \\ 1 \\ .. \\ 0 \end{pmatrix} \; .. \; \begin{pmatrix} 0 \\ 0 \\ .. \\ 1 \end{pmatrix} \right\}$$

Learning of the softmax model

- Learning of the parameters **w**: statistical view
- **Likelihood of outputs**

$$L(D,\mathbf{w}) = p(\mathbf{Y} \mid \mathbf{X},\mathbf{w}) = \prod_{i=1,..n} p(y_i \mid \mathbf{x}_i,\mathbf{w})$$

- We want parameters **w** that maximize the likelihood
- **Log-likelihood trick**
  – Optimize log-likelihood of outputs instead:

$$l(D,\mathbf{w}) = \log \prod_{i=1,..n} p(y_i \mid \mathbf{x},\mathbf{w}) = \sum_{i=1,..n} \log p(y_i \mid \mathbf{x},\mathbf{w})$$

$$= \sum_{i=1,..n} \sum_{j=0}^{k-1} \log g_j(\mathbf{x}_i)^{y_{i,j}} = \sum_{i=1,..n} \sum_{j=0}^{k-1} y_{i,j} \log g_j(\mathbf{x}_i)$$

- **Corresponding error (negative log likelihood)**

$$J(D,\mathbf{w}) = -\sum_{i=1}^{n} \sum_{j=0}^{k-1} y_{i,j} \log g_j(\mathbf{x}_i)$$

Learning of the softmax model

- **Error to optimize:**

$$J(D,\mathbf{w}) = -\sum_{i=1}^{n} \sum_{j=0}^{k-1} y_{i,j} \log g_j(\mathbf{x}_i)$$

- **Gradient**

$$\frac{\partial}{\partial w_{ju}} J(D,\mathbf{w}) = \sum_{i=1}^{n} -x_{i,u}(y_{i,j} - g_j(\mathbf{x}_i))$$

- The same very easy **gradient update** as used for the binary logistic regression

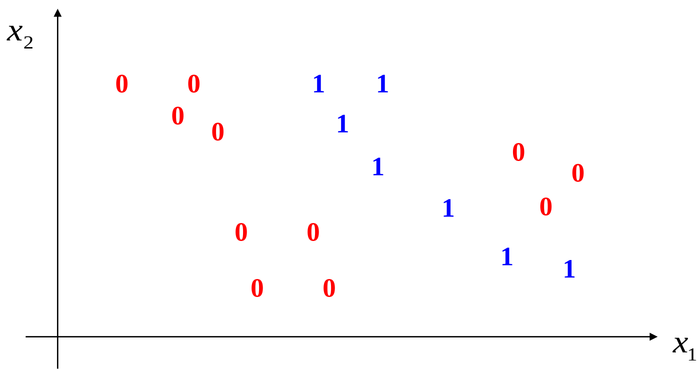
$$\mathbf{w}_j \leftarrow \mathbf{w}_j + \alpha \sum_{i=1}^{n} (y_{i,j} - g_j(\mathbf{x}_i))\mathbf{x}_i$$

- We have to update the weights of k networks

Decision trees

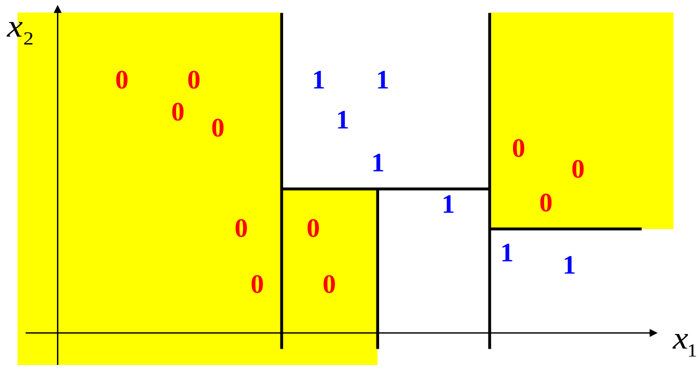
Decision tree classification

- An alternative approach to classification:
  - **Partition the input space to regions**
  - **Regress or classify independently in every region**



Decision tree classification

- An alternative approach to classification:
  - **Partition the input space to regions**
  - **Regress or classify independently in every region**



Decision tree classification

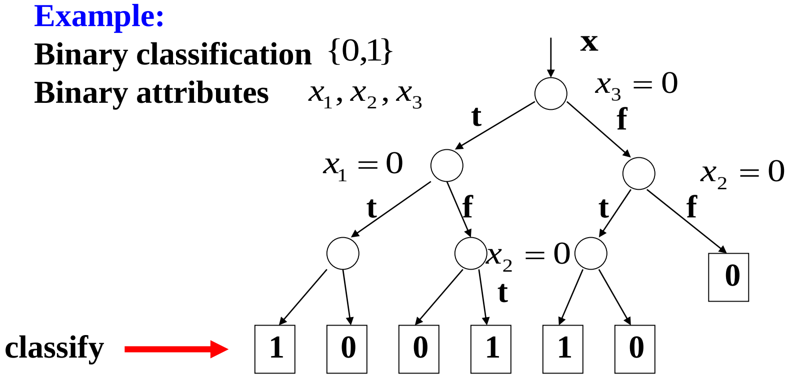
Decision tree model:

- formed by simple conditions (tests) on individual dimensions x_i that recursively split the input space **x**
- Classify at the bottom of the tree

Example:

Binary classification $\{0,1\}$

Binary attributes x_1, x_2, x_3



Decision trees

Decision tree model:

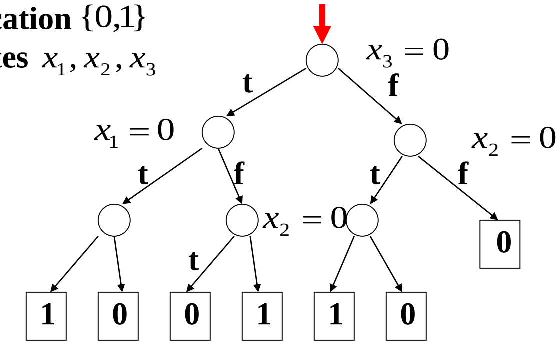
- formed by simple conditions (tests) on individual dimensions x_i that recursively split the input space **x**
- Classify at the bottom of the tree

Example:

Binary classification $\{0,1\}$

Binary attributes x_1, x_2, x_3

$\mathbf{x} = (x_1, x_2, x_3) = (1,0,0)$



classify

Decision trees

Decision tree model:

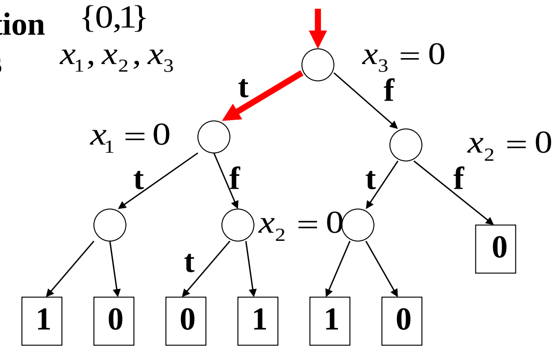
- formed by simple conditions (tests) on individual dimensions x_i that recursively split the input space **x**
- Classify at the bottom of the tree

Example:

Binary classification $\{0,1\}$

Binary attributes x_1, x_2, x_3

$\mathbf{x} = (x_1, x_2, x_3) = (1,0,0)$



classify

Decision trees

Decision tree model:

- formed by simple conditions (tests) on individual dimensions x_i that recursively split the input space $\mathbf{x}$
- Classify at the bottom of the tree

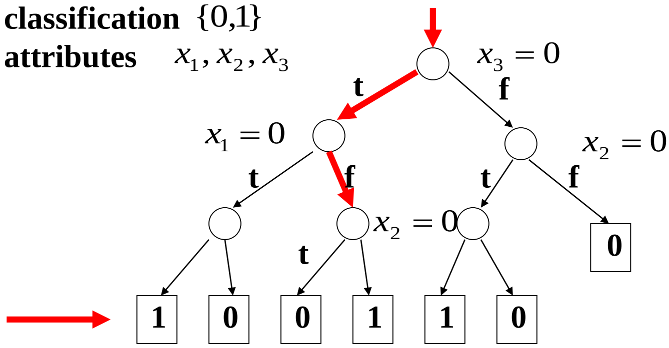
Example:

$\mathbf{x} = (x_1, x_2, x_3) = (1,0,0)$

Binary classification $\{0,1\}$

Binary attributes x_1, x_2, x_3

classify



Decision trees

Decision tree model:

- formed by simple conditions (tests) on individual dimensions x_i that recursively split the input space $\mathbf{x}$
- Classify at the bottom of the tree

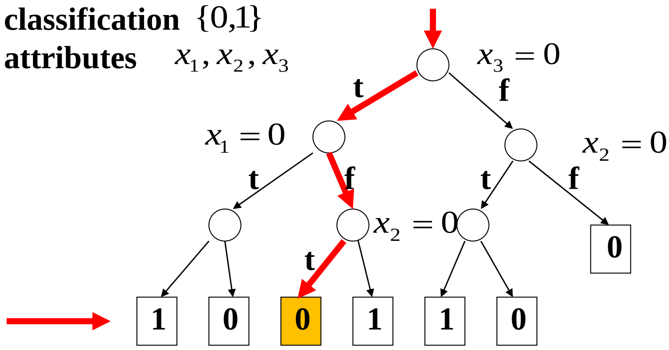
Example:

$\mathbf{x} = (x_1, x_2, x_3) = (1,0,0)$

Binary classification $\{0,1\}$

Binary attributes x_1, x_2, x_3

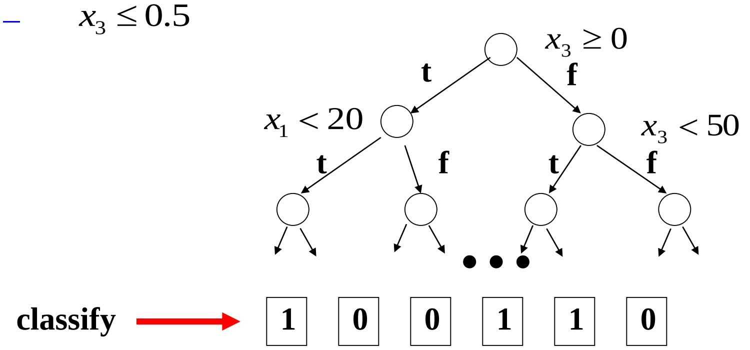
classify



Decision tree splits

Splits/conditions:

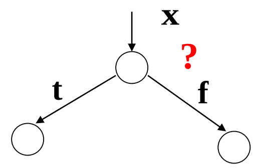
- **Equalities on categorical, binary values**
  - $x_3 = 0$ **or** $x_2 = Blue$
- **Inequalities for real values**
  - $x_3 \leq 0.5$



Tree construction

How to construct /learn the decision tree?

Top-down algorithm:



- Finds the best split (condition) that can improve the classification performance after that split
- Stops when no improvement possible

Question: How to measure the improvement?

We measure it with the help of :

- **Impurity measure:** measures the degree of mixing of the two classes in the subset of the training data D
  - Worst (maximum impurity) when # of 0s and 1s is the same

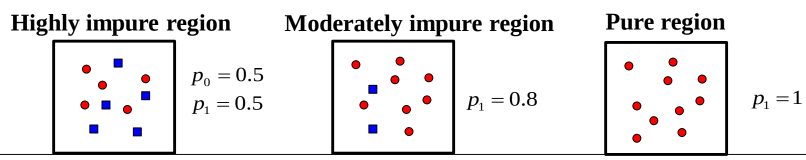
Impurity measure

Let D be a collection of data instances:

- Let D_0 and D_1 be subsets of D corresponding to class 0 and class 1
- Proportion if class 0 and class 1 instances $p_0 = \frac{|D_0|}{|D|}$ $p_1 = \frac{|D_1|}{|D|}$

Impurity measure *I*(D)

- Measures the degree of mixing of the two classes in D
- The impurity measure should be:
  - Largest when data are split evenly among the classes
  - Should be 0 when all data belong to the same class

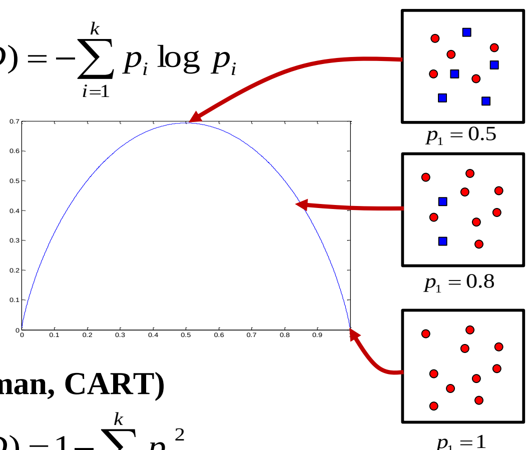


Impurity measures

- There are various impurity measures used in the literature
  - **Entropy based measure (Quinlan, C4.5)**

$$I(D) = Entropy(D) = -\sum_{i=1}^{k} p_i \log p_i$$

Example for k=2



  - **Gini measure (Breiman, CART)**

$$I(D) = Gini(D) = 1 - \sum_{i=1}^{k} {p_i}^2$$

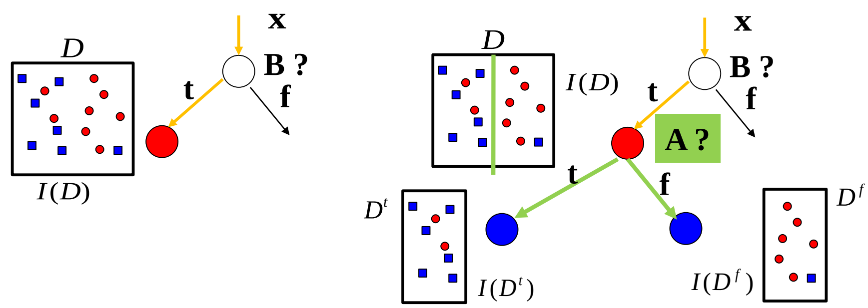
Classification improvement

Idea: **add the split that reduces the impurity the most**

Gain due to split – expected reduction in the impurity

Split condition

$$Gain(D, A) = I(D) - \left[\frac{|D^t|}{|D|} I(D^t) + \frac{|D^f|}{|D|} I(D^f) \right]$$



Decision tree learning

Greedy learning algorithm:

- Builds the tree in the top-down fashion
- Gradually expands the leaves of the partially built tree

Algorithm sketch:

Repeat until no or small improvement in the impurity

- Find the attribute and the condition with the highest gain
- Add the condition to the tree and split the set accordingly

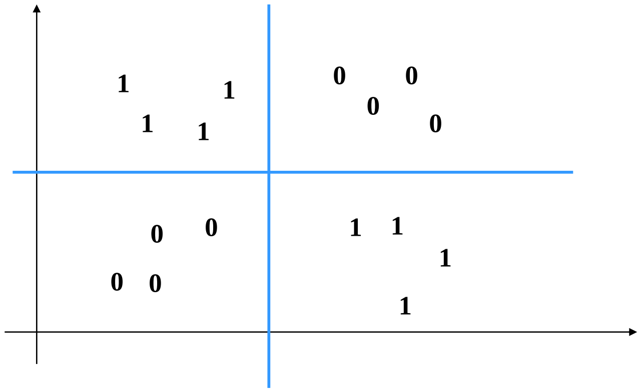
Limitations: greedy approach:

- It looks at a single attribute condition and gain in each step
- May fail when the combination of attributes is needed to improve the purity

Decision tree learning

- **Limitations of greedy methods**

  Cases in which only a combination of two or more attributes improves the impurity



Decision tree learning

By reducing the impurity measure we can grow **very large trees**

Problem: Overfitting

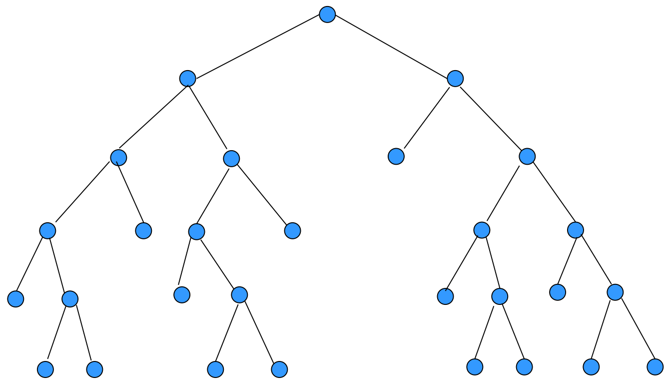
- We may split and classify very well the training set, but we may do worse in terms of the generalization error

Solutions to the overfitting problem:

- **Solution 1. Build the tree then prune the branches**
  - Build the tree, then eliminate leaves that overfit
  - Use **validation set** to test for the overfit
- **Solution 2. Prune while building the tree**
  - Test for the overfit in the tree building phase
  - Stop building the tree when performance on the **validation set** deteriorates

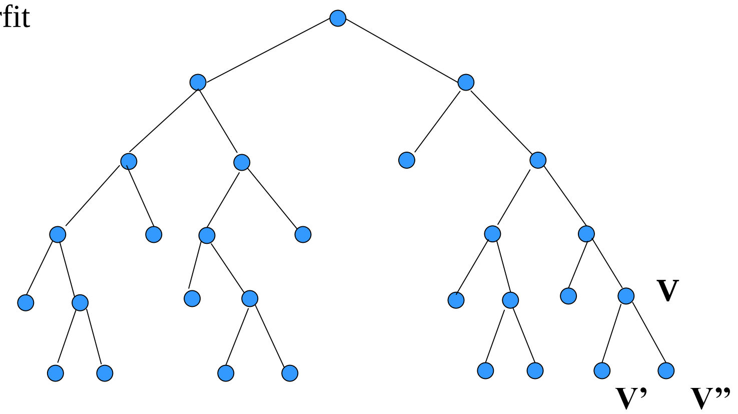
Decision tree learning

Backpruning: Prune branches of the tree built in the first phase in the botton-up fashion by using the validation set to test for the overfit



Decision tree learning

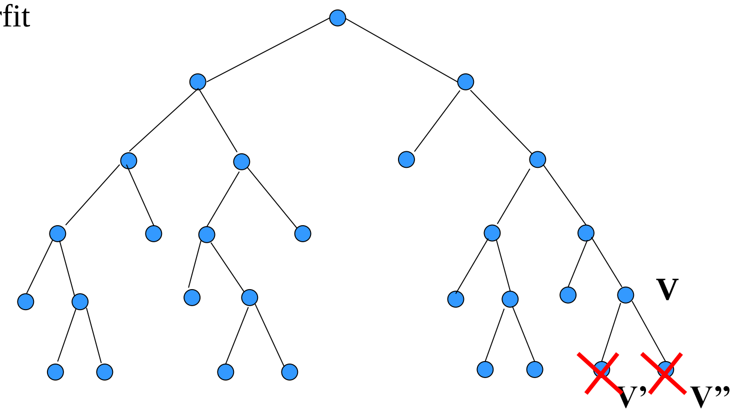
Backpruning: Prune branches of the tree built in the first phase in the botton-up fashion by using the validation set to test for the overfit



Compare: #Errors (V) vs #Error (V') + # Errors(V'')

Decision tree learning

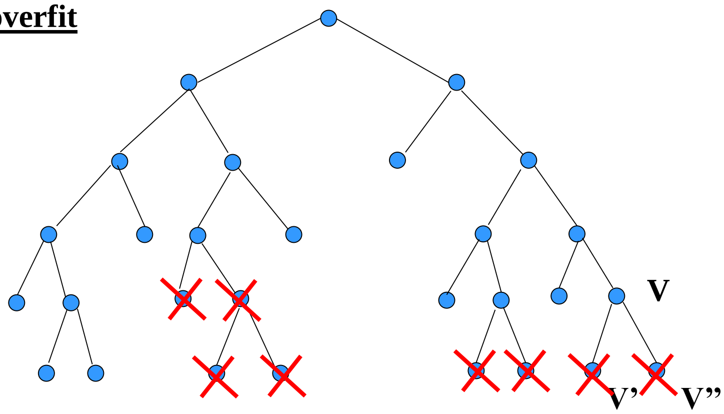
Backpruning: Prune branches of the tree built in the first phase in the botton-up fashion by using the validation set to test for the overfit



Compare: #Errors (V) < #Error (V') + # Errors(V'')

Decision tree learning

Backpruning: Prune branches of the tree built in the first phase in the botton-up fashion by using **the validation set to test for the overfit**



Compare: #Errors (V) < #Error (V') + # Errors(V'')