CS 2750 Machine Learning
Lecture 11

Multi-way classification.
Decision trees.

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

Multi-way classification

- **Binary classification** $Y = \{0,1\}$
- **Multi-way classification**
  - **K classes** $Y = \{0,1,\ldots,K-1\}$
  - **Goal:** learn to classify correctly K classes
  - Or **learn** $f : X \rightarrow \{0,1,\ldots,K-1\}$
- **Errors:**
  - **Zero-one (misclassification) error for an example:**

$$Error_1(\mathbf{x}_i, y_i) = \begin{cases} 1 & f(\mathbf{x}_i, \mathbf{w}) \neq y_i \\ 0 & f(\mathbf{x}_i, \mathbf{w}) = y_i \end{cases}$$

  - **Mean misclassification error (for a dataset):**

$$\frac{1}{n}\sum_{i=1}^{n} Error_1(\mathbf{x}_i, y_i)$$

Multi-way classification

Approaches:

- **Generative model approach**
  - Generative model of the distribution p(**x**,y)
  - Learns the parameters of the model through density estimation techniques
  - Discriminant functions are based on the model
    - "Indirect" learning of a classifier
- **Discriminative approach**
  - Parametric discriminant functions
  - Learns discriminant functions **directly**
    - A logistic regression model.

Generative model approach

Indirect:

1. **Represent and learn the distribution** $p(\mathbf{x}, y)$
2. **Define and use probabilistic discriminant functions**

$$g_i(\mathbf{x}) = \log\, p(y = i \mid \mathbf{x})$$

Model $p(\mathbf{x}, y) = p(\mathbf{x} \mid y)\, p(y)$

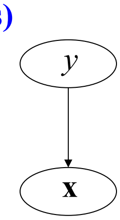
- $p(\mathbf{x} \mid y)$ = **Class-conditional distributions (densities)**

  k class-conditional distributions

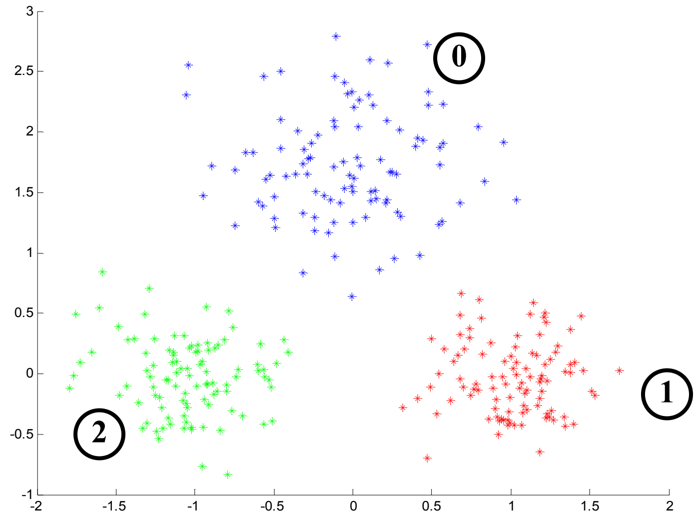
  $$p(\mathbf{x} \mid y = i) \qquad \forall i \quad 0 \le i \le K-1$$

- $p(y)$ = **Priors on classes**
- **-** probability of class y

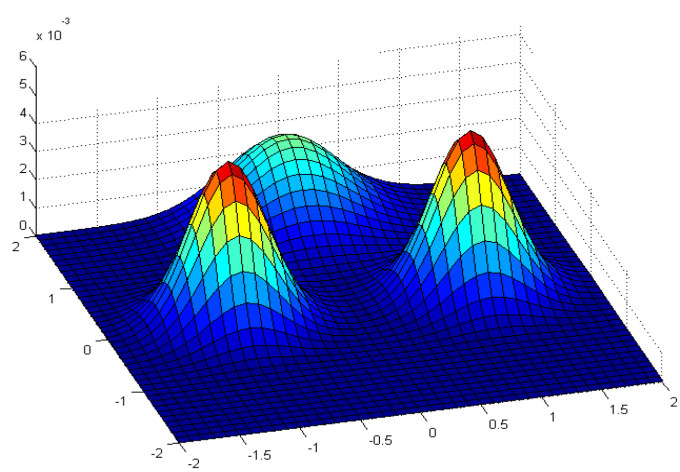
$$\sum_{i=1}^{K-1} p(y = i) = 1$$



Multi-way classification. Example



Multi-way classification



Making class decision

Discriminant functions can be based on:

- **Likelihood of data** – choose the class (Gaussian) that explains the input data (**x**) better (likelihood of the data)

Choice: $$i = \arg\max_{i=0,\dots k-1} p(\mathbf{x} \mid \boldsymbol{\theta}_i)$$

$$p(\mathbf{x} \mid \boldsymbol{\theta}_i) \approx p(\mathbf{x} \mid \mu_i, \boldsymbol{\Sigma}_i) \quad \text{For Gaussians}$$

- **Posterior of a class** – choose the class with higher posterior probability

Choice: $$i = \arg\max_{i=0,\dots k-1} p(y = i \mid \mathbf{x}, \boldsymbol{\theta}_i)$$

$$p(y = i \mid \mathbf{x}) = \frac{p(\mathbf{x} \mid \Theta_i) p(y = i)}{\sum_{j=0}^{k-1} p(\mathbf{x} \mid \Theta_j) p(y = j)}$$

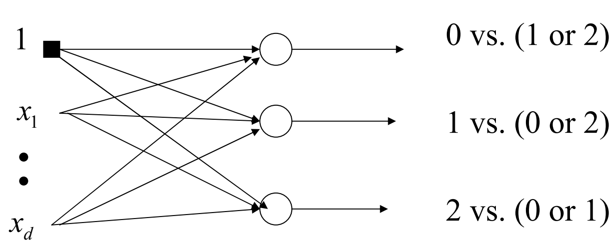
Discriminative approach

- **Parametric model** of discriminant functions
- Learns the discriminant functions directly

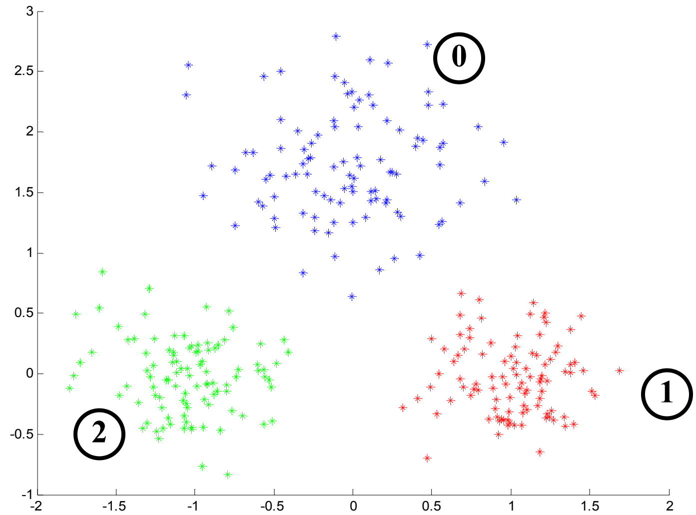
How to learn to classify multiple classes, say 0,1,2?

Approach 1:

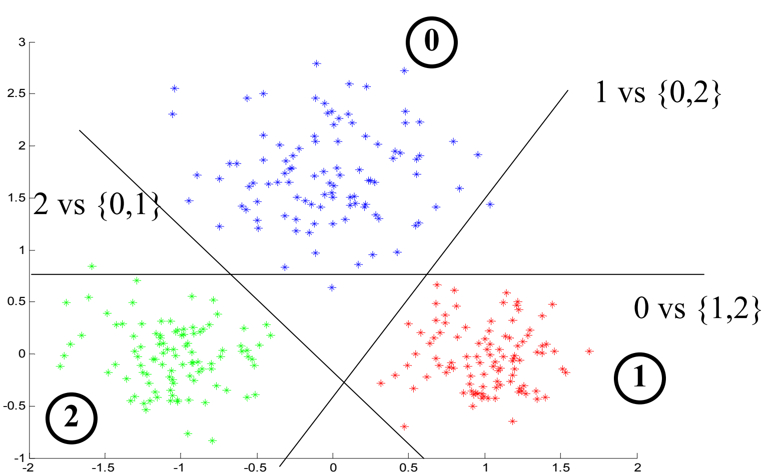
– A binary logistic regression on every class versus the rest



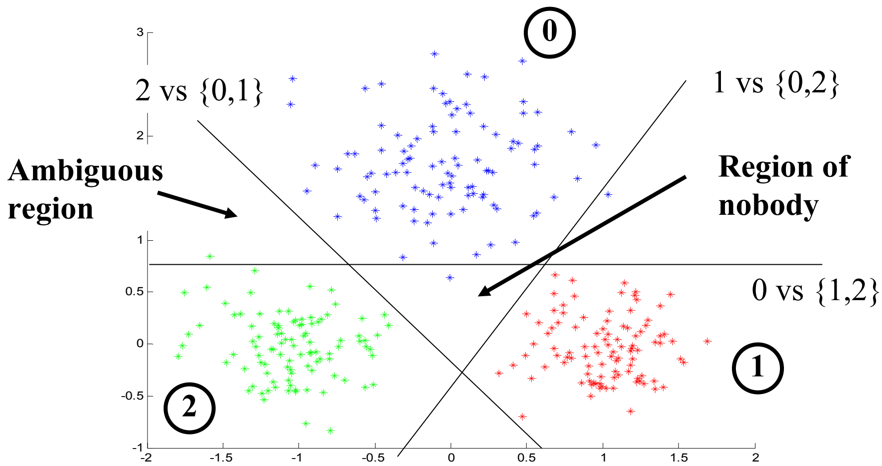
Multi-way classification. Example



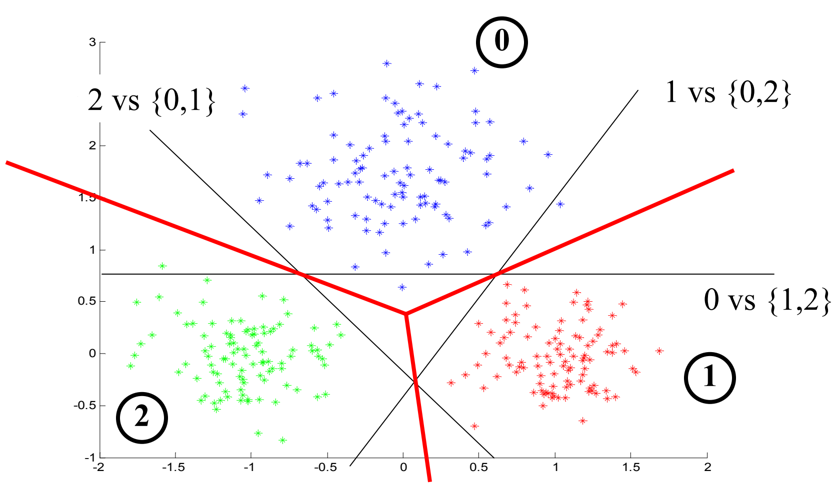
Multi-way classification. Approach 1.



Multi-way classification. Approach 1.



Multi-way classification. Approach 1.

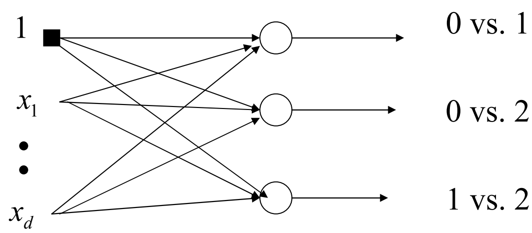


Discriminative approach.

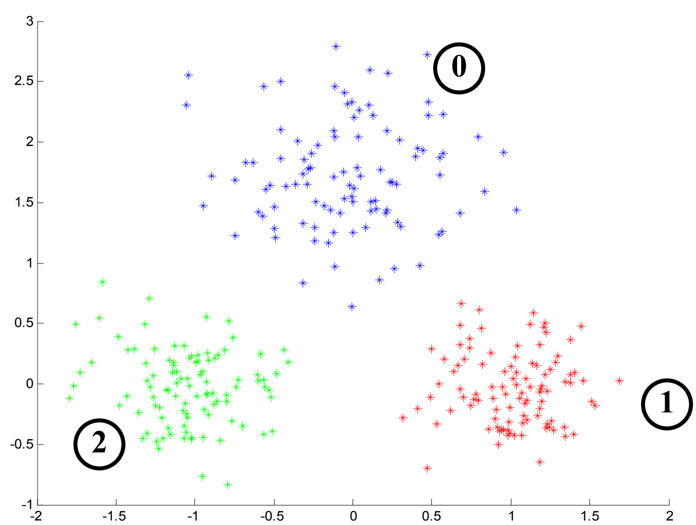
How to learn to classify multiple classes, say 0,1,2 ?

Approach 2:

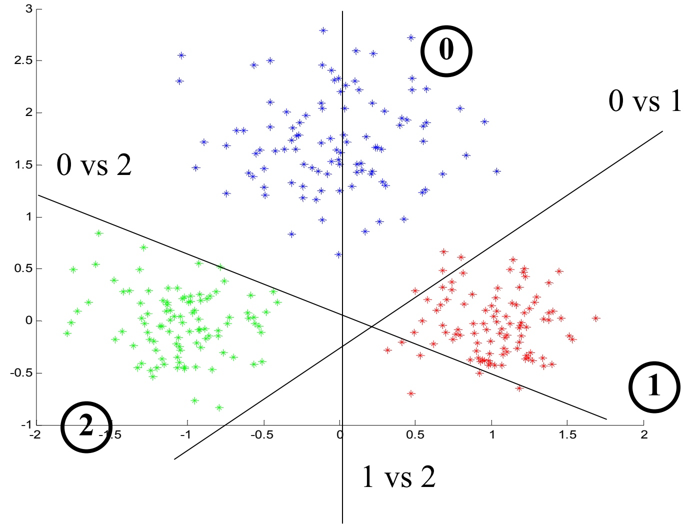
– A binary logistic regression on all pairs



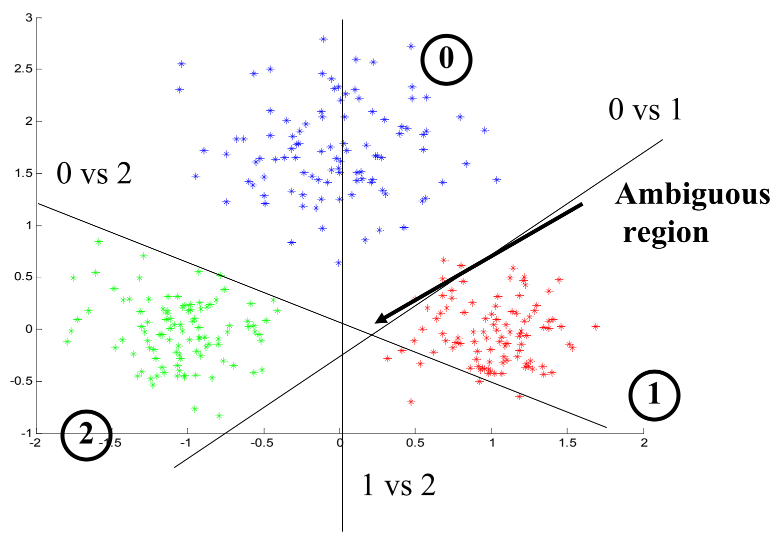
Multi-way classification. Example



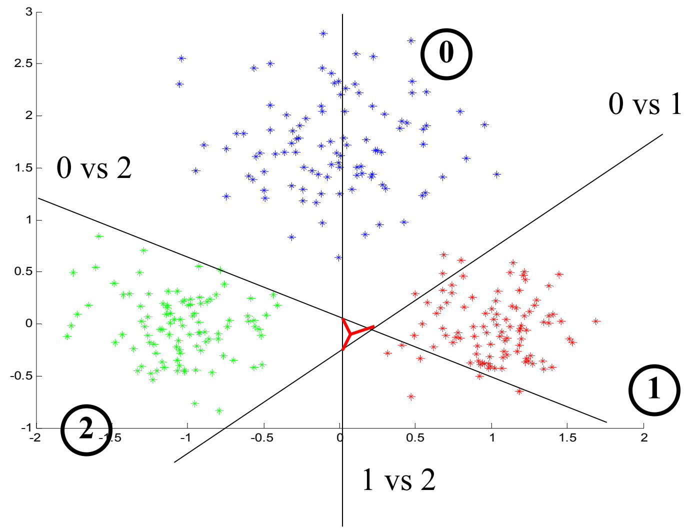
Multi-way classification. Approach 2



Multi-way classification. Approach 2

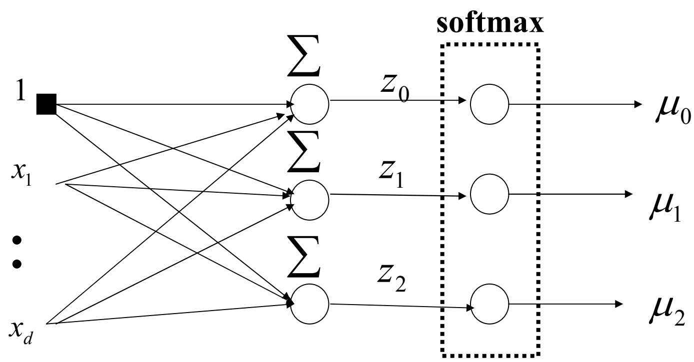


Multi-way classification. Approach 2



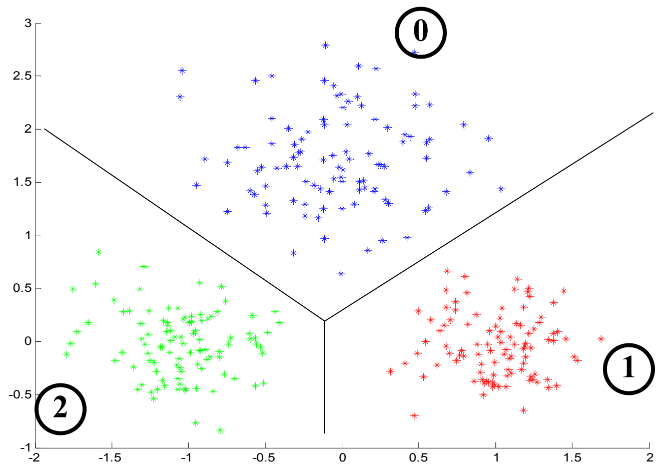
Multi-way classification with softmax

- A solution to the problem of having an ambiguous region



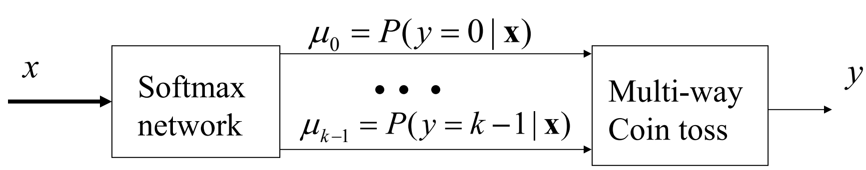
$$p(y=i\,|\,\mathbf{x}) = \mu_i = \frac{\exp(\mathbf{w}_i^T\mathbf{x})}{\sum_j \exp(\mathbf{w}_j^T\mathbf{x})} \qquad \sum_i \mu_i = 1$$

Multi-way classification with softmax



Learning of the softmax model

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



Assume outputs y are transformed as follows

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

Learning of the softmax model

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

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

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

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

$$= \sum_{i=1,..n} \sum_{q=0}^{k-1} \log \mu_i^{y_{i,q}} = \sum_{i=1,..n} \sum_{q=0}^{k-1} y_{i,q} \log \mu_{i,q}$$

- **Objective to optimize** $J(D_i,\mathbf{w}) = -\sum_{i=1}^{n} \sum_{q=0}^{k-1} y_{i,q} \log \mu_{i,q}$

Learning of the softmax model

- **Error to optimize:**

$$J(D_i,\mathbf{w}) = -\sum_{i=1}^{n} \sum_{q=0}^{k-1} y_{i,q} \log \mu_{i,q}$$

- **Gradient**

$$\frac{\partial}{\partial w_{jq}} J(D_i,\mathbf{w}) = \sum_{i=1}^{n} -x_{i,j}(y_{i,q} - \mu_{i,q})$$

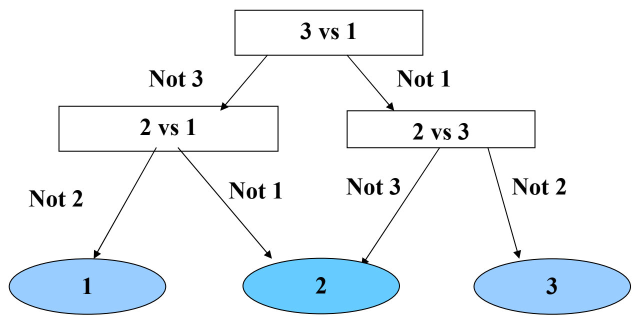
- The same very easy **gradient update** as used for the binary logistic regression

$$\mathbf{w}_q \leftarrow \mathbf{w}_q + \alpha \sum_{i=1}^{n} (y_{i,q} - \mu_{i,q})\mathbf{x}_i$$

- But now we have to update the weights of k networks

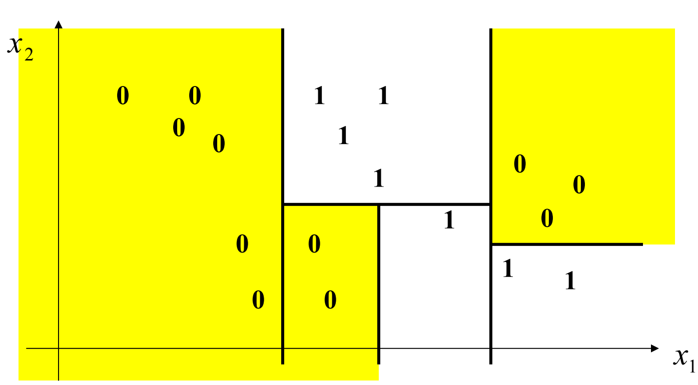
Multi-way classification

- **Yet another approach 3**



Decision trees

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



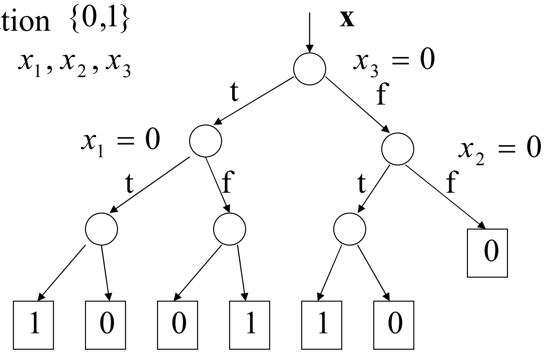
Decision trees

- The partitioning idea is used in the **decision tree model**:
  - Split the space recursively according to inputs in **x**
  - Regress or classify at the bottom of the tree

Example:

Binary classification $\{0,1\}$

Binary attributes x_1, x_2, x_3

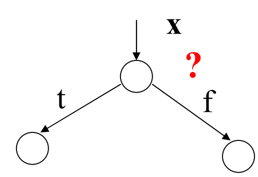


classify →

Decision trees

How to construct the decision tree?

- **Top-bottom algorithm**:
  - Find the best split condition (quantified based on the impurity measure)
  - Stops when no improvement possible



- **Impurity measure**:
  - Measures how well are the two classes separated
  - Ideally we would like to separate all 0s and 1
- Splits of **finite vs. continuous value attributes**

  Continuous value attributes conditions: $x_3 \leq 0.5$

Impurity measure

Let $|D|$ - Total number of data entries

$|D_i|$ - Number of data entries classified as i

$$p_i = \frac{|D_i|}{|D|}$$ - ratio of instances classified as i

- **Impurity measure** defines how well the classes are separated
- In general the impurity measure should satisfy:
  - Largest when data are split evenly for attribute values

$$p_i = \frac{1}{\text{number of classes}}$$

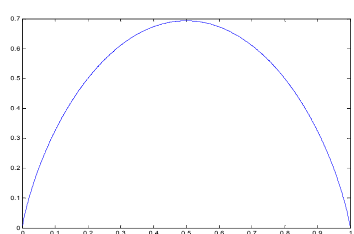
  - Should be 0 when all data belong to the same class

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)**

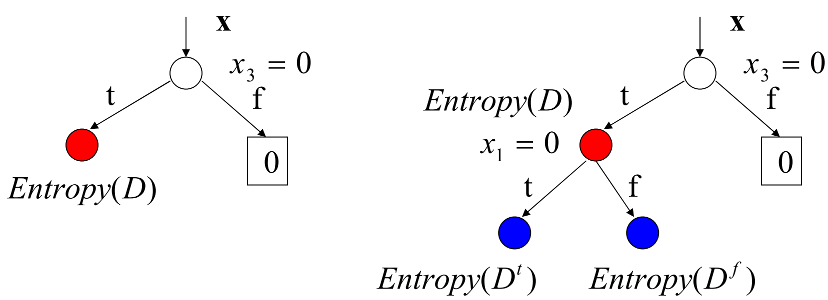
$$I(D) = Gini(D) = 1 - \sum_{i=1}^{k} p_i^2$$

Impurity measures

- **Gain due to split** – expected reduction in the impurity measure (entropy example)

$$Gain(D, A) = Entropy(D) - \sum_{v \in Values(A)} \frac{|D^v|}{|D|} Entropy(D^v)$$

$|D^v|$ - a partition of D with the value of attribute A = v



Decision tree learning

- **Greedy learning algorithm**:

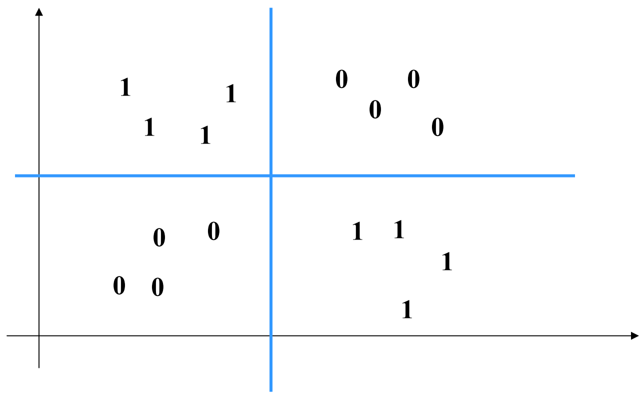
  Repeat until no or small improvement in the purity
  - Find the attribute with the highest gain
  - Add the attribute to the tree and split the set accordingly

- Builds the tree in the top-down fashion
  - Gradually expands the leaves of the partially built tree
- The method is greedy
  - It looks at a single attribute and gain in each step
  - May fail when the combination of attributes is needed to improve the purity (parity functions)

Decision tree learning

- **Limitations of greedy methods**

  Cases in which 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.**
  - Prune branches of the tree built in the first phase
  - Use validation set to test for the overfit
- **Solution 2**.
  - Test for the overfit in the tree building phase
  - Stop building the tree when performance on the validation set deteriorates

K-Nearest-Neighbours for Classification

- Given a data set with N_k data points from class C_k and $\sum_k N_k = N$, we have

$$p(\mathbf{x}) = \frac{K}{NV}$$

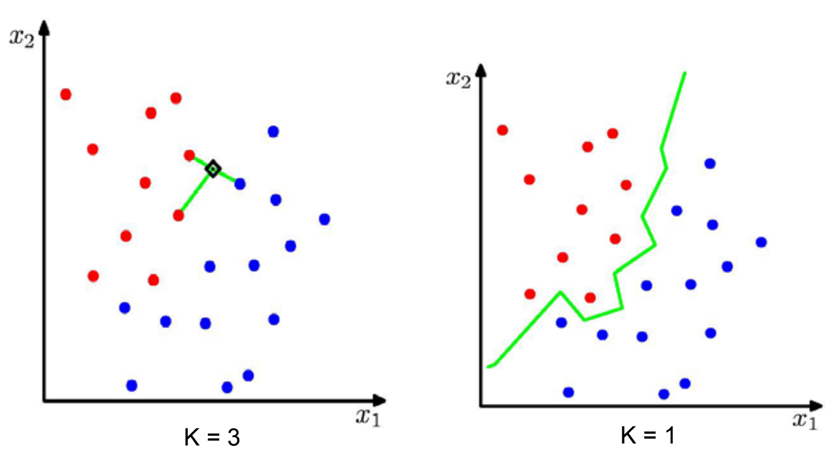
- and correspondingly

$$p(\mathbf{x}|\mathcal{C}_k) = \frac{K_k}{N_k V}.$$

- Since $p(\mathcal{C}_k) = N_k/N$ Bayes' theorem gives

$$p(\mathcal{C}_k|\mathbf{x}) = \frac{p(\mathbf{x}|\mathcal{C}_k)p(\mathcal{C}_k)}{p(\mathbf{x})} = \frac{K_k}{K}.$$

K-Nearest-Neighbours for Classification



Nonparametric kernel-based classification

- **Kernel function: k(x,x')**
  - Models similarity between x, x'
  - **Example:** Gaussian kernel we used in kernel density estimation

$$k(x,x') = \frac{1}{(2\pi h^2)^{D/2}} \exp\left(- \frac{(x-x')^2}{2h^2} \right)$$

$$p(x) = \frac{1}{N} \sum_{i=1}^{N} k(x, x_i)$$

- **Kernel for classification**

$$p(y = C_k \mid x) = \frac{\sum_{x':y'=C_k} k(x,x')}{\sum_{x'} k(x,x')}$$