Machine Learning in Telecommunications

Paulos Charonyktakis & Maria Plakia

Department of Computer Science, University of Crete
Institute of Computer Science, FORTH

Roadmap

- Motivation
- Supervised Learning
- Algorithms
  - Artificial Neural Networks
  - Naïve Bayes Classifier
  - Decision Trees
- Application on VoIP in Wireless Networks

Landscape in Telecommunications

- **Dramatic growth of mobile data, streaming services, telepresence**
- By 2019 mobile data traffic over 24 exabytes/month worldwide
- Growth in video delivery segment
- Robust growth opportunity for networking, server, and specialized hardware providers, due to:
  - mobile device capacity growth
  - advances in networked home
  - cloud services
  - user-generated content
- Existing & emerging large access markets & services

Motivation

- Need to analyze a large amount of heterogeneous data
- Detect trends and patterns
- Characterize the Quality of Experience of various services
- Analyze the performance of various services, providers, and networks

Objectives: QoE Modeling & Analysis

How does the network performance affect the perceived quality of experience (QoE) of a user ?

- To predict the QoE based on network performance, apply machine learning and data mining algorithms, such as:

  Decision Trees, Support Vector Regression, Artificial Neural Networks, Gaussian Naïve Bayes
- Train the models based on network measurements and opinion scores collected in the context of a service
- Demonstrate this methodology for **VoIP** and **video services**

Machine Learning

- The study of algorithms and systems that improve their performance with experience (Mitchell book)
- Experience = data / measurements / observations

Where to Use Machine Learning

- You have past data, you want to predict the future
- You have data, you want to make sense out of them (find patterns)
- You have a problem which is hard to be modeled
    - Gather input-output pairs to learn the mapping
- Measurements + intelligent behavior usually lead to some form of Machine Learning

Supervised Learning

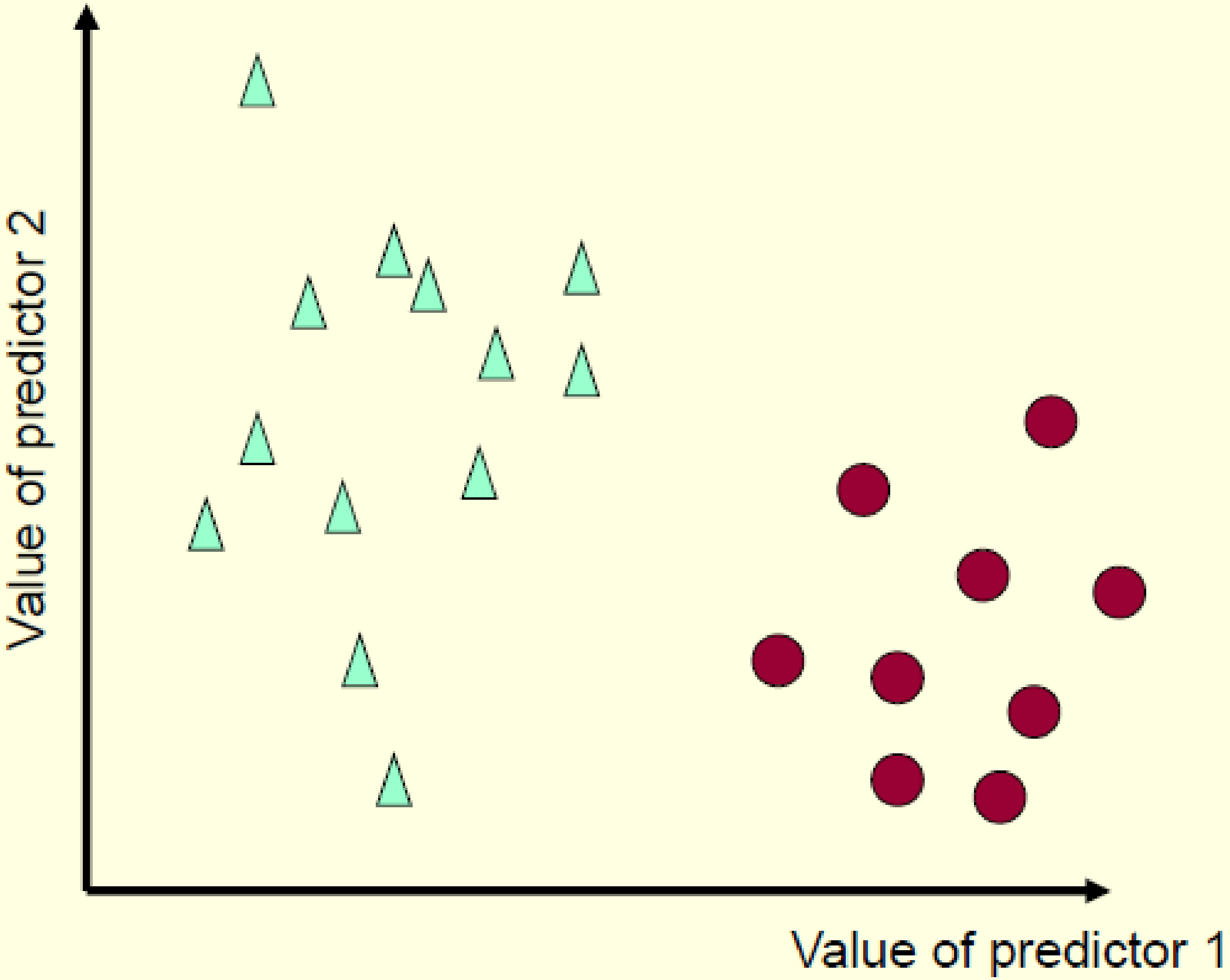
- Learn from examples
- Would like to be able to predict an outcome of interest *y for an object x*
- Learn function y = *f(x)*
- For example, *x is a VoIP call, y is an indicator of QoE*
- Given data $\{<x_i, y_i> : i=1, \ldots, n\}$,
  - x_i *the representation of an* ***object****, i.e., predictors*
  - y_i *the representation of a* ***known outcome****, i.e., class labels*
- Learn the function y = *f(x) that generalizes from the data the "best" (has minimum average error)*

Classification vs. Regression

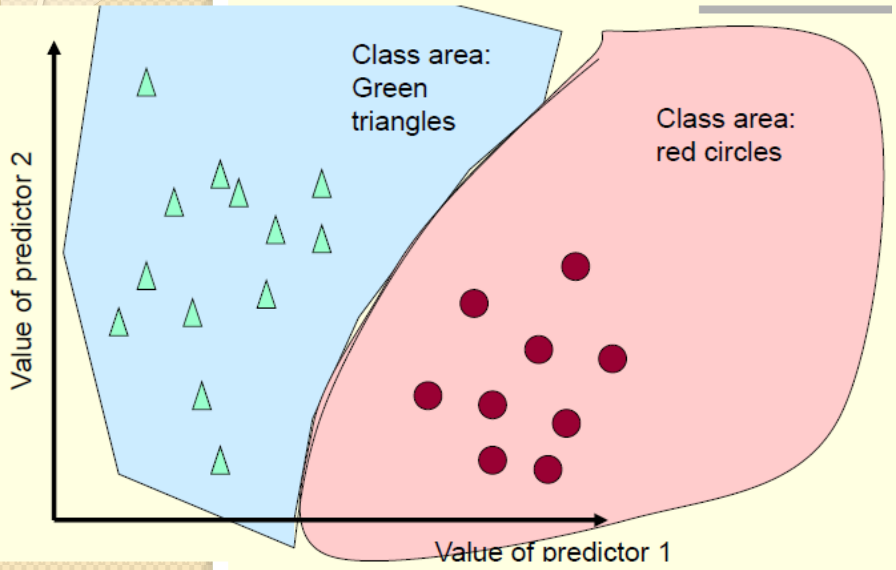
- Classification
  - Constructs decision surfaces
  - Predicts **categorical** class **labels** (discrete or nominal)
  - Classifies (assigns a label) to new data
- Regression
  - Constructs a regression line
  - Predicts **continuous values** along the line

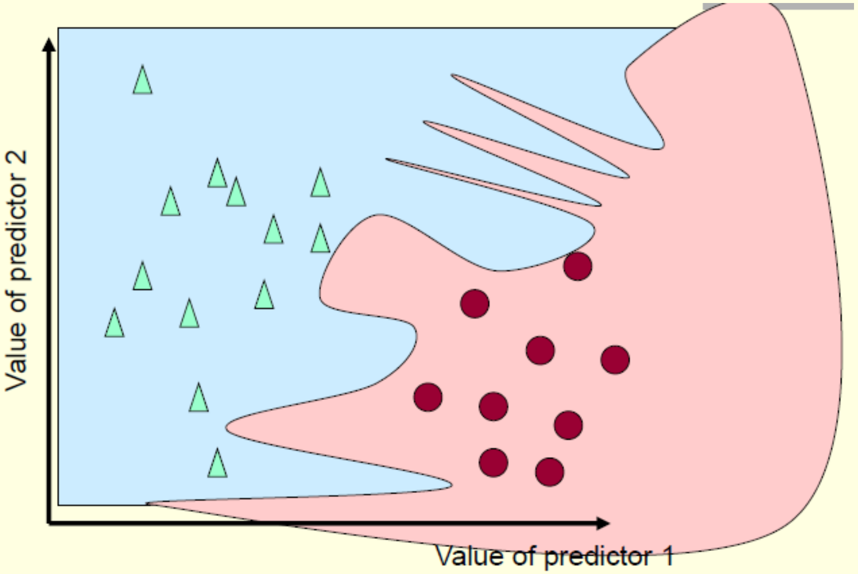
Algorithms:
Artificial Neural Networks

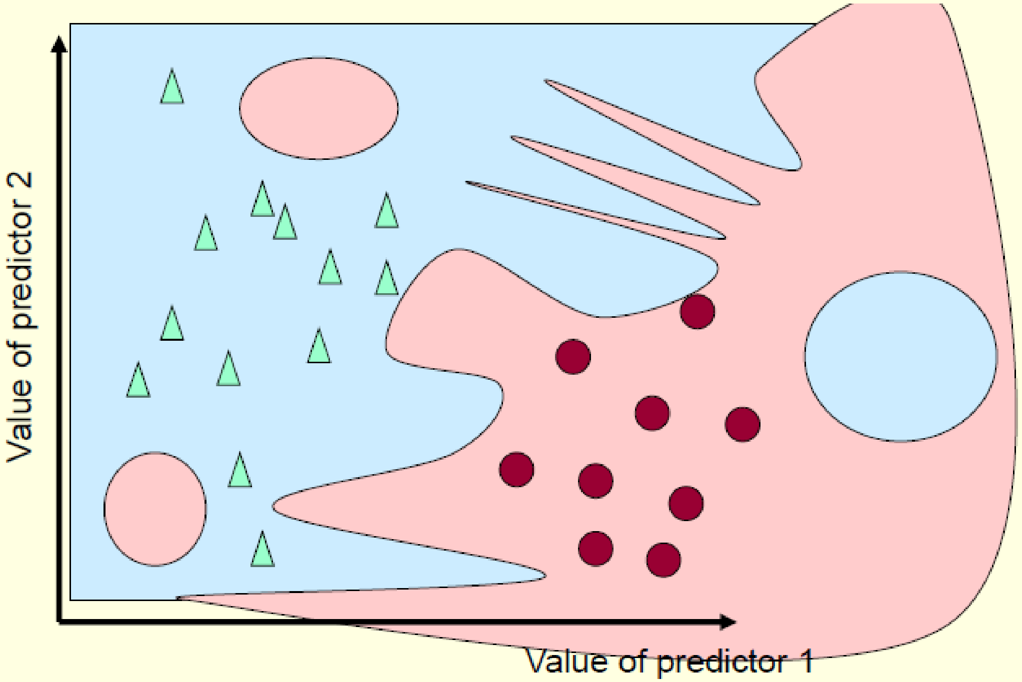
Binary Classification Example



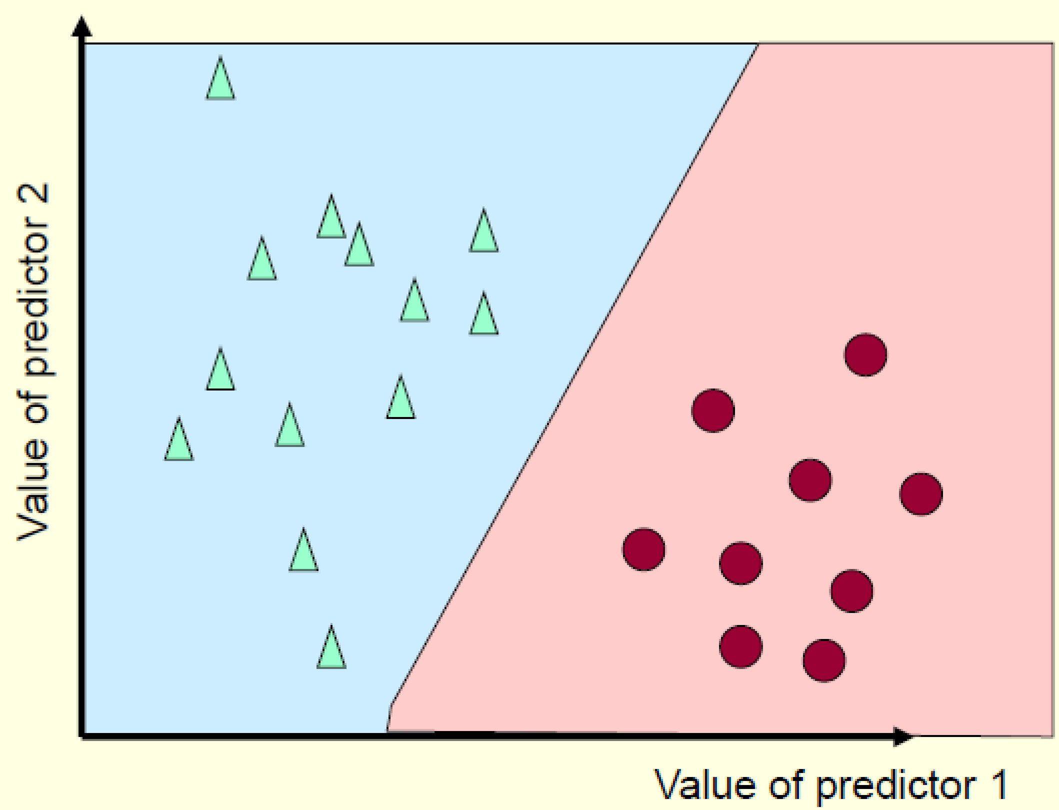
Possible Decision Areas





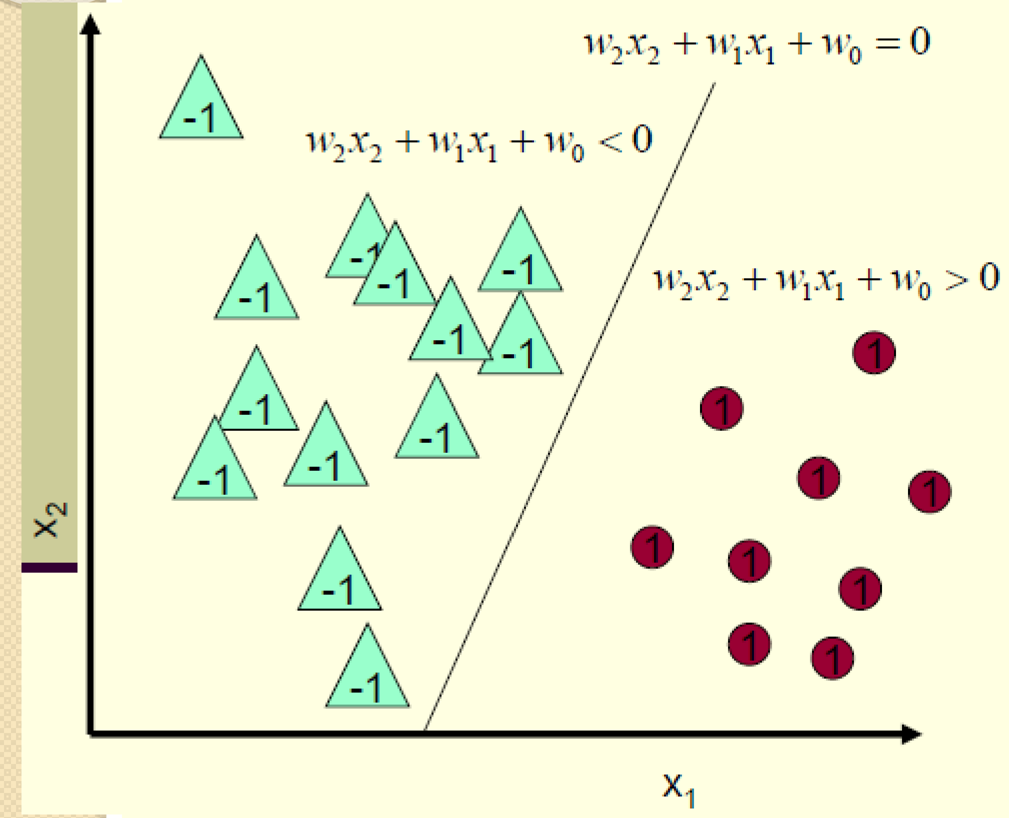


Binary Classification Example



- The simplest non-trivial decision function is the straight line
- One decision surface
- Decision surface partitions space into two subspaces
- In the case of high dimensional space, a **hyperplane** is the decision function

Specifying a Line (1)



Line equation:

$$w_2x_2 + w_1x_1 + w_0 = 0$$

Classifier model:

If $w_2x_2 + w_1x_1 + w_0 \geq 0$

- Output 1

Else

- Output -1

Specifying a Line (2)

- Classifier becomes

$$\mathrm{sgn}(w_2x_2 + w_1x_1 + w_0) =$$

$$\mathrm{sgn}(w_2x_2 + w_1x_1 + w_0x_0),$$

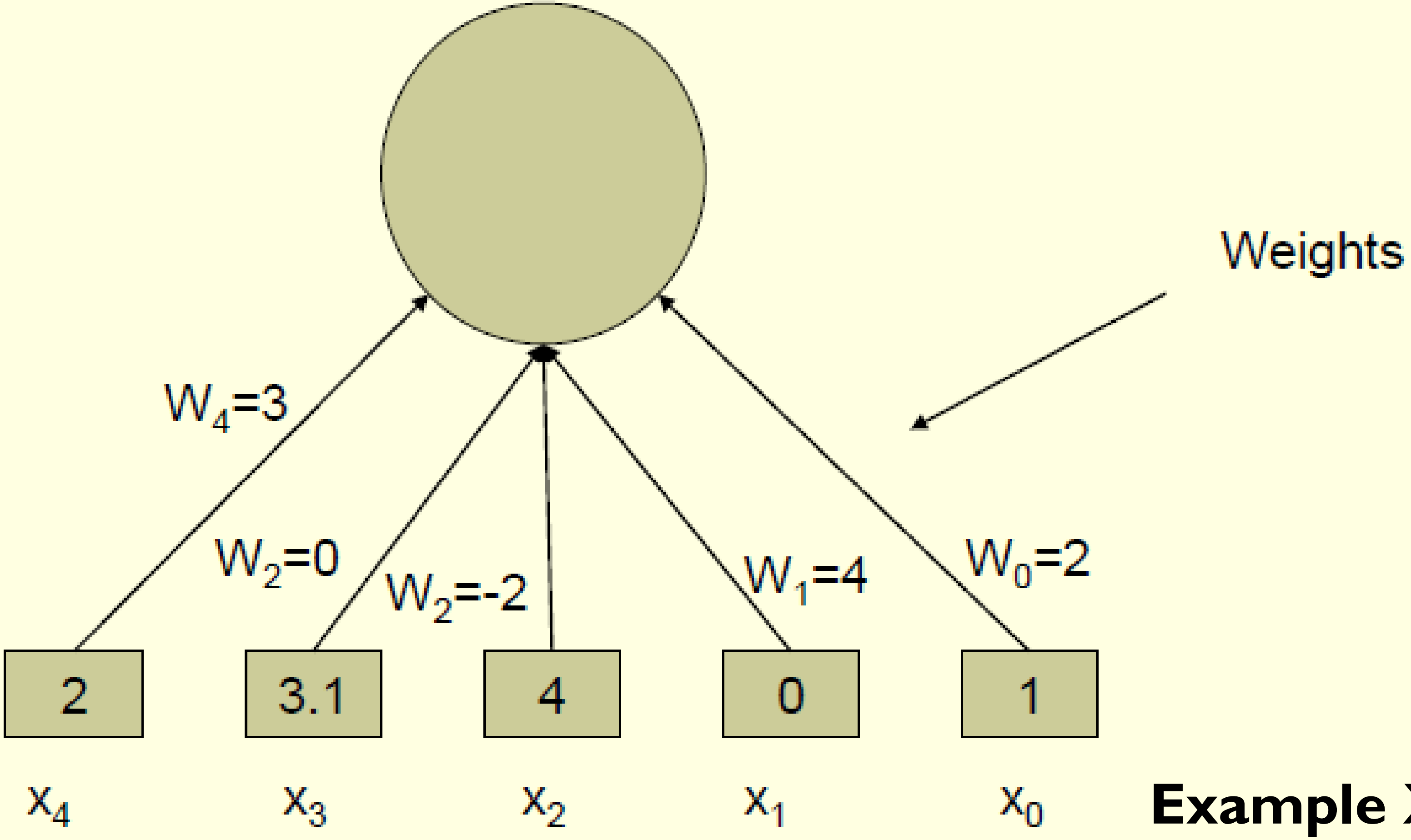
set $x_0 = 1$ always

Let n be the number of predictors

$$\mathrm{sgn}(\textstyle\sum_{i=0}^{n} w_ix_i), \text{ or}$$

$$\mathrm{sgn}(\vec{w} \cdot \vec{x})$$

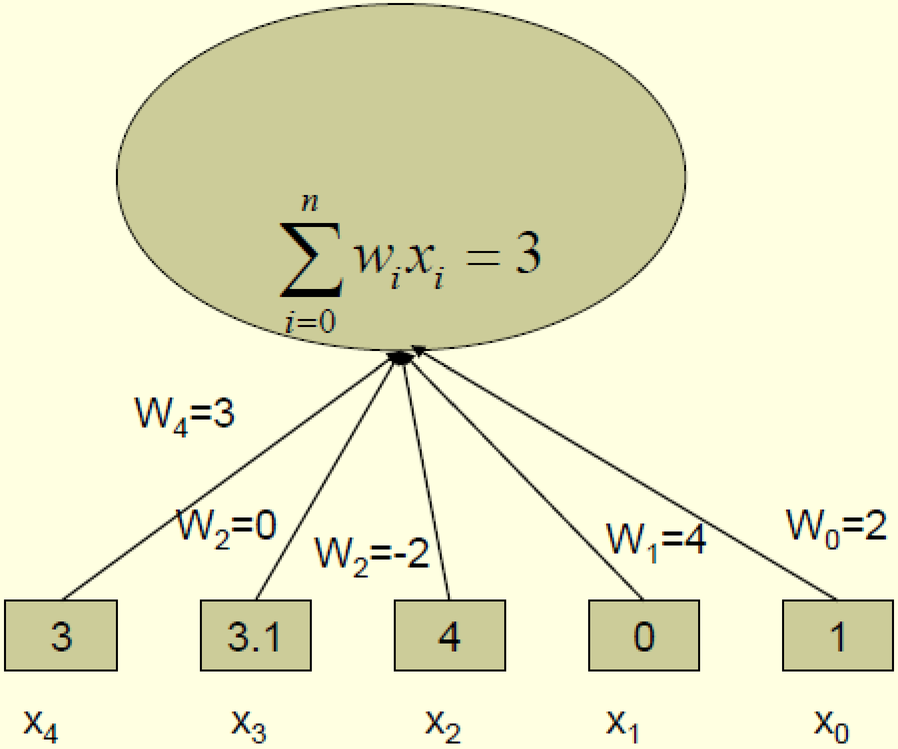
The simplest neural network: the Perceptron



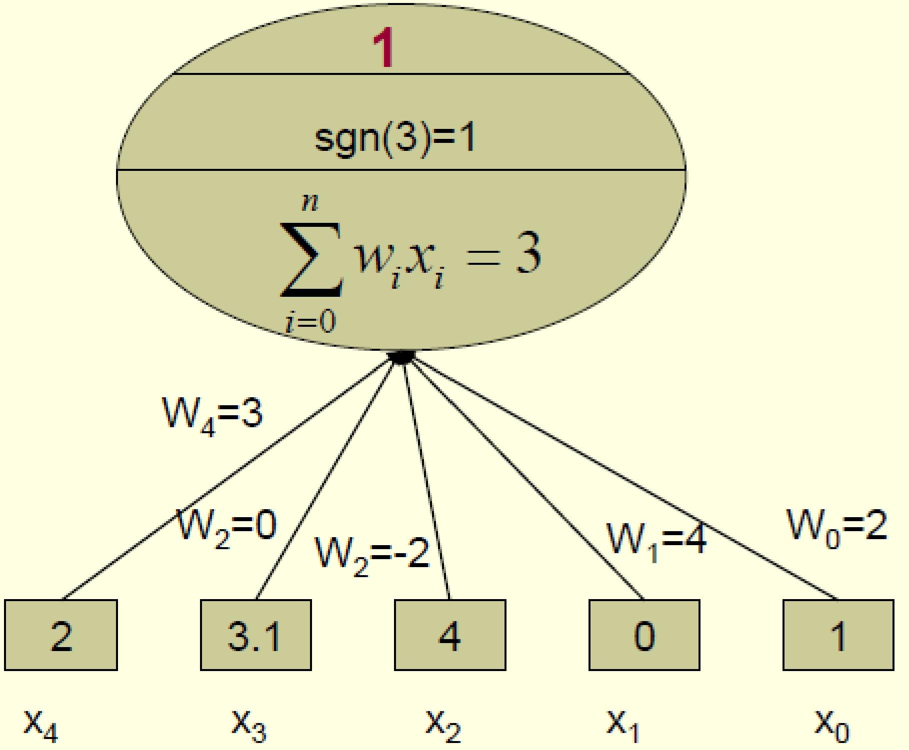
Example X

The Perceptron is a 2-layer neural network

The simpler Neural: The Perceptron



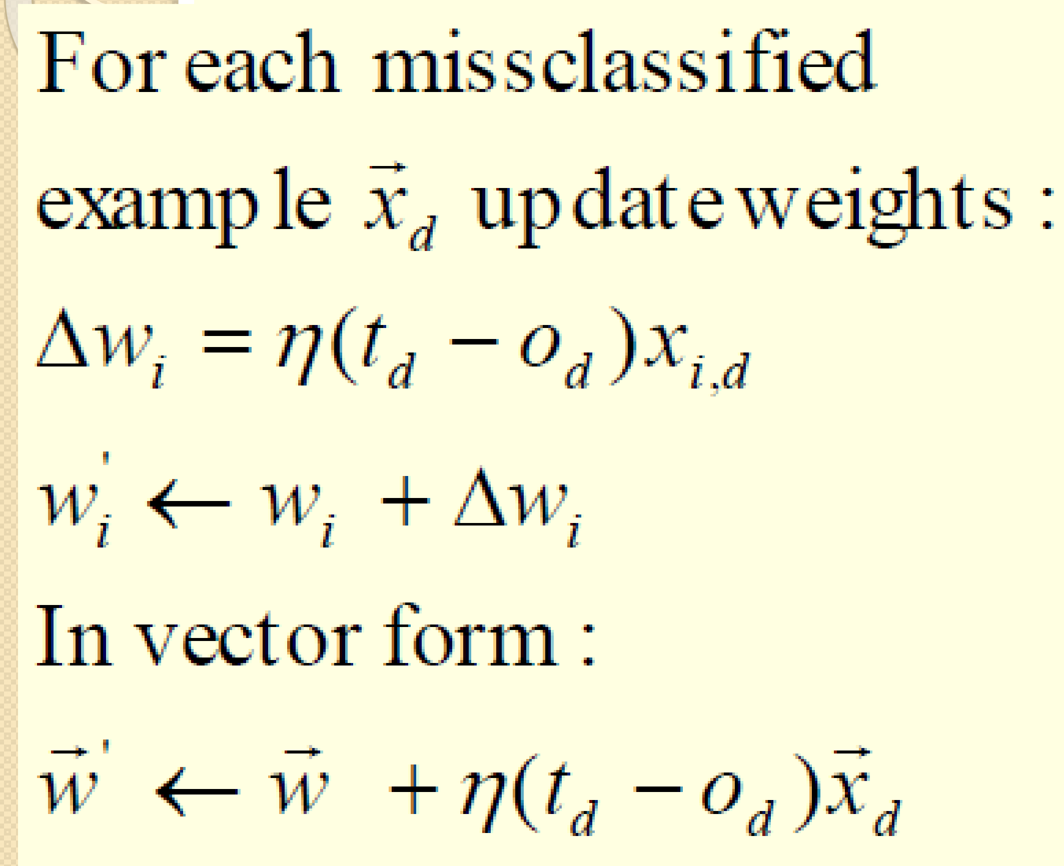
The simpler Neural: The Perceptron



Training Perceptrons

- Start with random weights
- Update in an intelligent way to improve them using the data
- Intuitively:
  - Decrease the weights that increase the sum
  - Increase the weights that decrease the sum
- Repeat for all training instances until convergence

Perceptron Training Rule



For each missclassified example $\vec{x}_d$ update weights :

$$\Delta w_i = \eta(t_d - o_d)x_{i,d}$$

$$w_i^{'} \leftarrow w_i + \Delta w_i$$

In vector form :

$$\vec{w}^{'} \leftarrow \vec{w} + \eta(t_d - o_d)\vec{x}_d$$

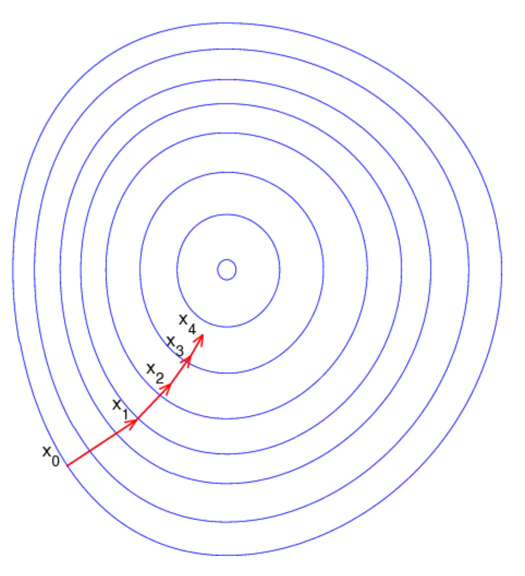
- *η: arbitrary learning rate (e.g. 0.5)*
- *t_d : (true) label of the d-th example*
- *o_d: output of the perceptron on the dth example*
- *$x_{i,d}$: value of predictor variable i of example d*
- *$t_d = o_d$: No change (for correctly classified examples)*

Analysis of the Perceptron Training Rule

- Algorithm will always converge within finite number of iterations if the data are **linearly separable**
- Otherwise, it may oscillate (no convergence)

Gradient Descent

- A first-order optimization algorithm
- Finds a local minimum
- Steps proportional to the *negative* of the gradient of the function at the current point



Training by Gradient Descent

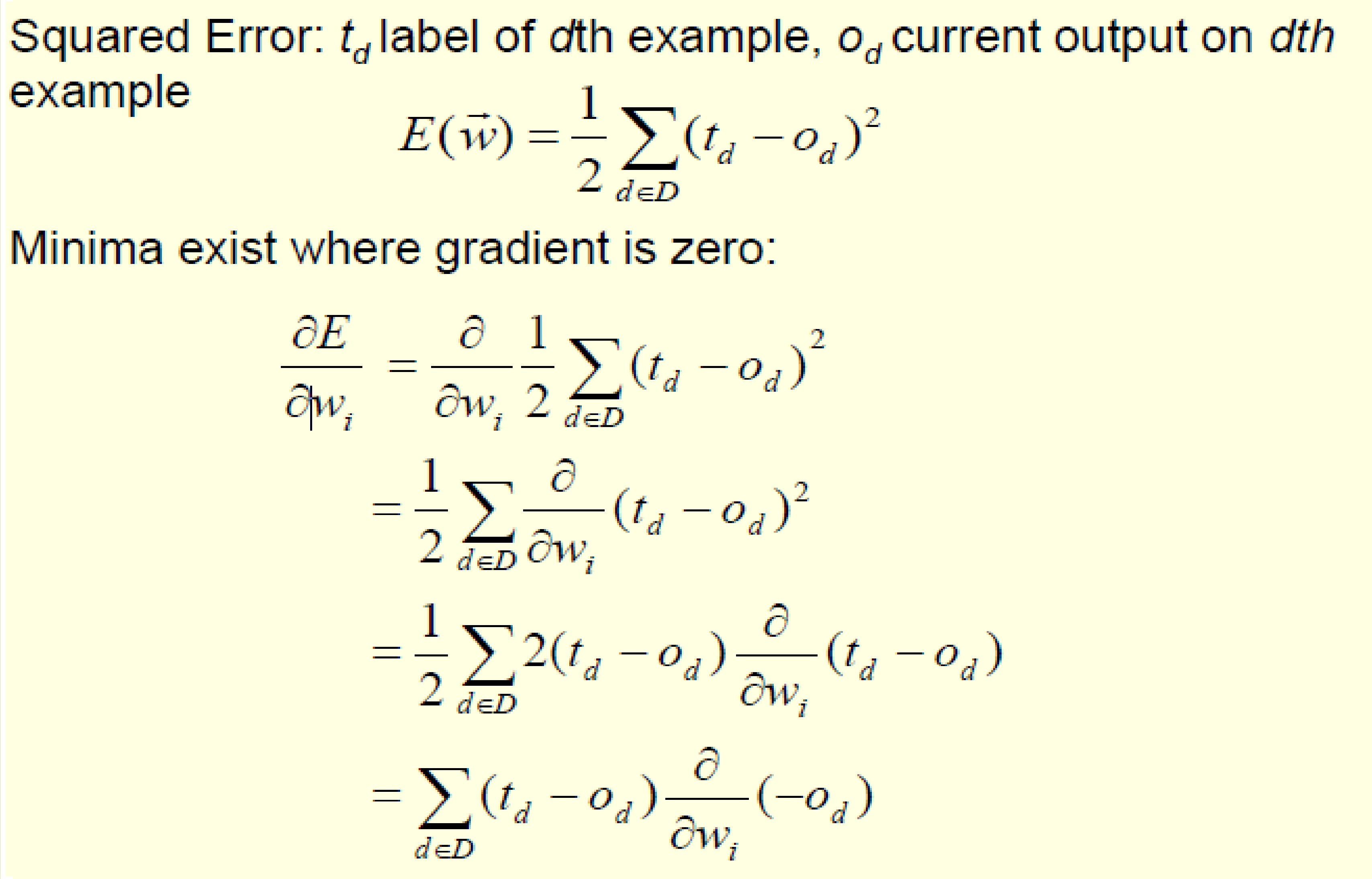
Idea:

- Define an error function
- Search for weights that minimize the error, i.e., find weights that zero the error gradient

Similar with the Perceptron training rule, but it the gradient descent:

- Always converges
- Generalizes to training networks of perceptrons (neural networks) and training networks for multicategory classification or regression

Setting Up the Gradient Descent



Squared Error: t_d label of dth example, o_d current output on dth example

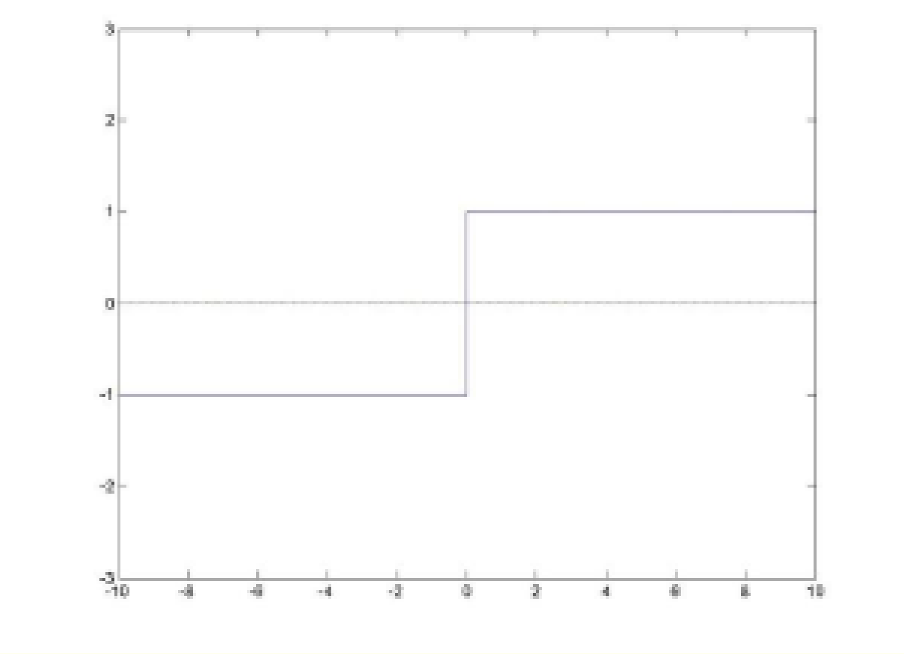
$$E(\vec{w}) = \frac{1}{2}\sum_{d \in D}(t_d - o_d)^2$$

Minima exist where gradient is zero:

$$\begin{aligned}
\frac{\partial E}{\partial w_i} &= \frac{\partial}{\partial w_i}\frac{1}{2}\sum_{d \in D}(t_d - o_d)^2 \\
&= \frac{1}{2}\sum_{d \in D}\frac{\partial}{\partial w_i}(t_d - o_d)^2 \\
&= \frac{1}{2}\sum_{d \in D}2(t_d - o_d)\frac{\partial}{\partial w_i}(t_d - o_d) \\
&= \sum_{d \in D}(t_d - o_d)\frac{\partial}{\partial w_i}(-o_d)
\end{aligned}$$

The Sign Function is not Differentiable

$$\frac{\partial}{\partial w_i}(-o_d) = -\frac{\partial o_d}{\partial w_i} = 0, \text{ everywhere except } o_d = 0$$



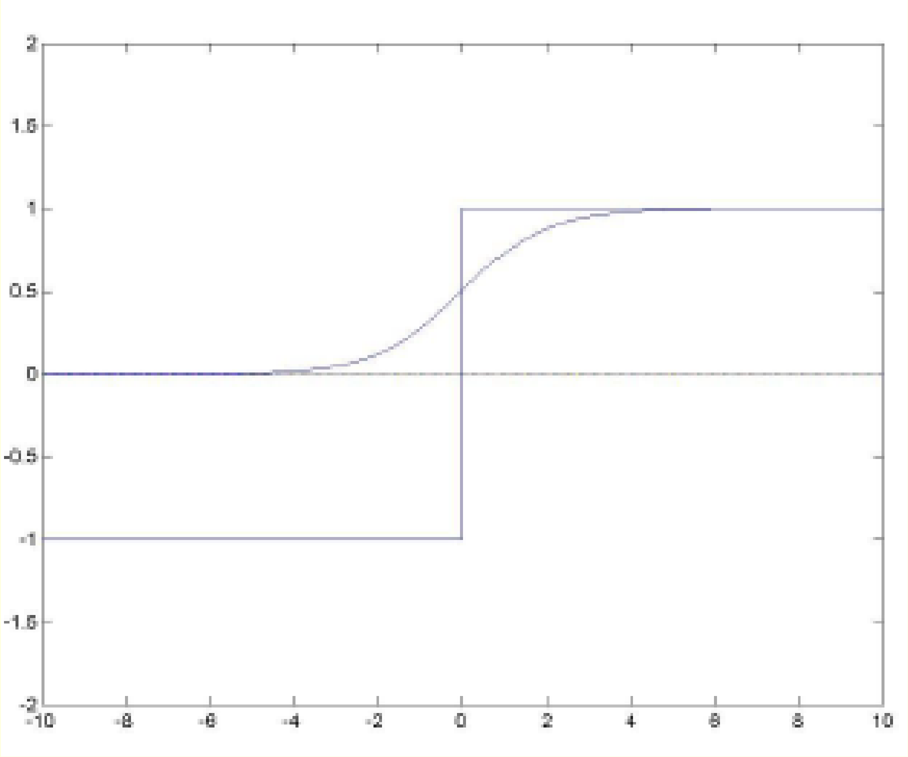
Use Differentiable Transfer Functions

- Replace with the sigmoid

$$sig(\vec{w} \cdot \vec{x}_d)$$

$$sig(y) = \frac{1}{1 + e^{-y}}$$

$$\frac{dsig(y)}{dy} = sig(y)(1 - sig(y))$$



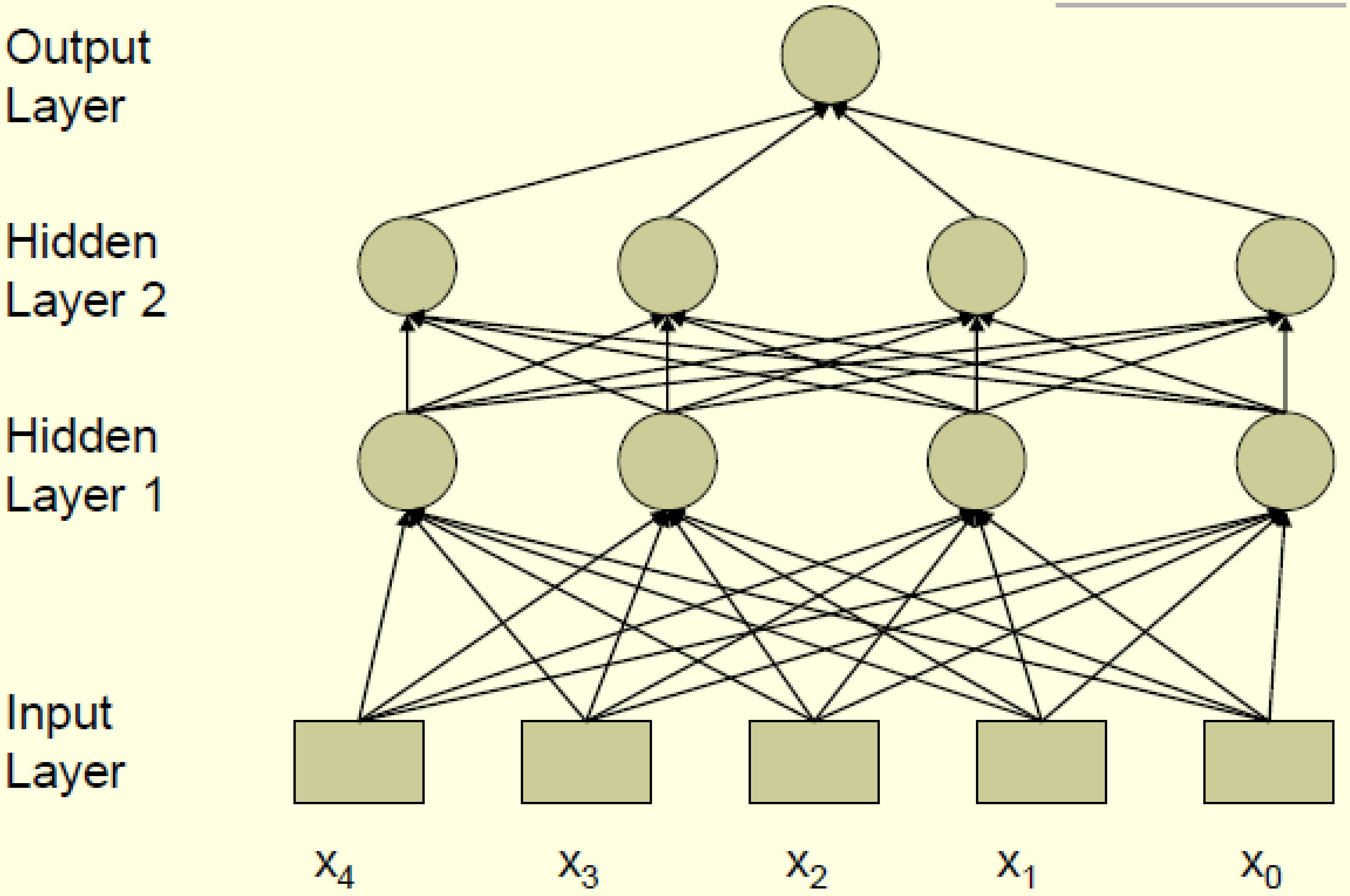
Updating the Weights with Gradient Descent

$$\vec{w} \leftarrow \vec{w} - \eta \nabla E(\vec{w})$$

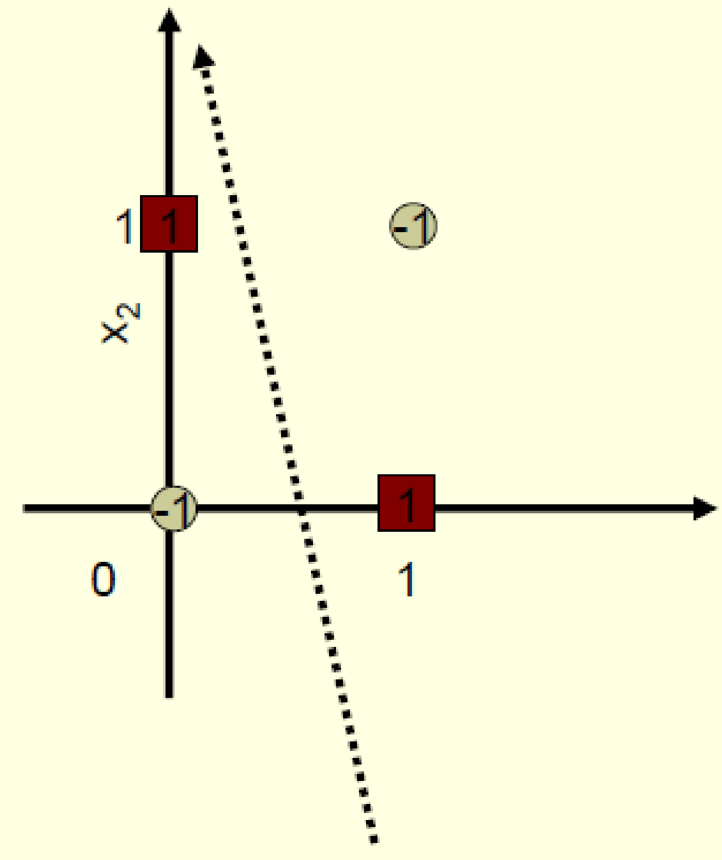
$$\vec{w} \leftarrow \vec{w} + \eta \sum_{d \in D} (t_d - o_d) sig(\vec{w} \cdot \vec{x}_d)(1 - sig(\vec{w} \cdot \vec{x}_d)) \cdot \vec{x}_d$$

- Each weight update goes through all training instances
- Each weight update more expensive but more accurate
- Always converges to a local minimum regardless of the data
- When using the sigmoid: output is a real number between 0 & 1
- Thus, labels (desired outputs) have to be represented with numbers from 0 to 1

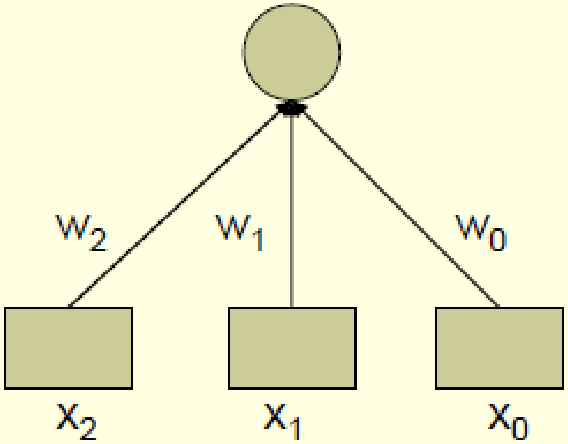
Feed-Forward Neural Networks



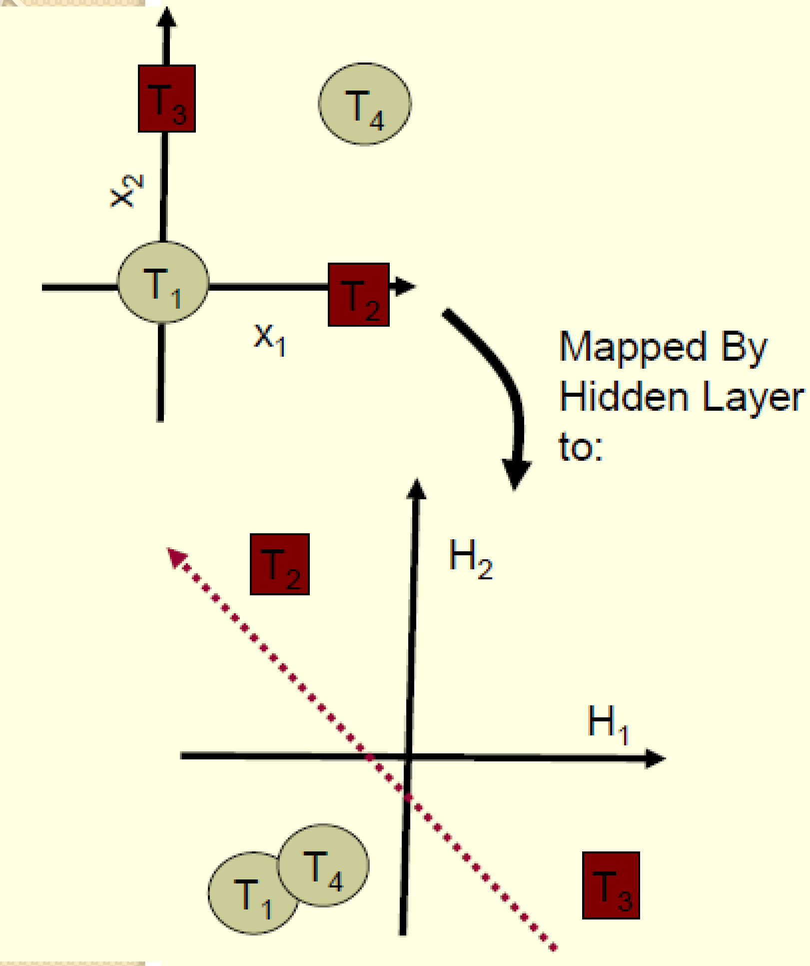
Increased Expressiveness Example: Exclusive OR



No line (no set of three weights) can separate the training examples (learn the true function).



From the Viewpoint of the Output Layer

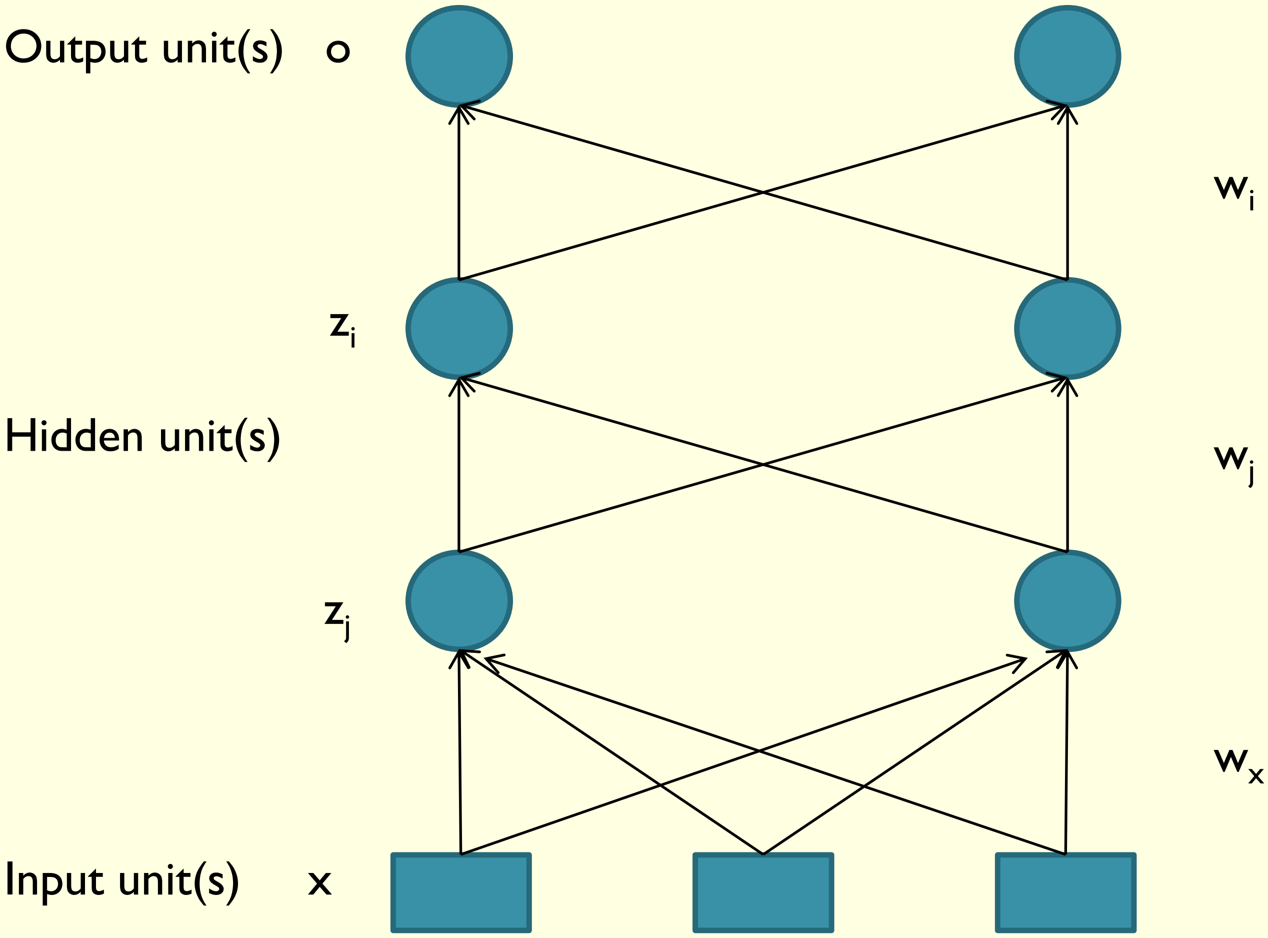


- Each hidden layer maps to a **new feature space**
- Each hidden node is a new constructed feature
- Original Problem may become separable (or easier)

How to Train Multi-Layered Networks

- Select a network structure (number of hidden layers, hidden nodes, and connectivity)
- Select transfer functions that are differentiable
- Define a (differentiable) error function
- Search for weights that minimize the error function, using gradient descent or other optimization method
- Backpropagation

Back-Propagating the Error



Output unit(s) o

w_i

z_i

Hidden unit(s)

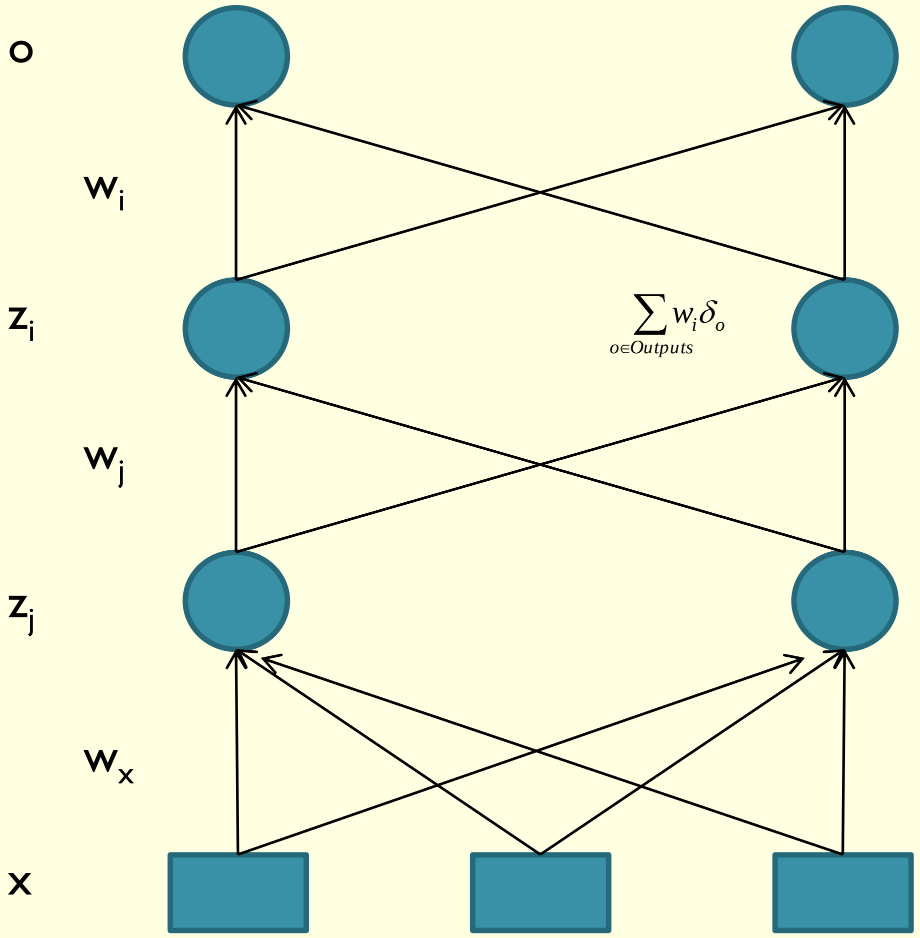
w_j

z_j

w_x

Input unit(s) x

Back-Propagating the Error



$$\delta_o = o\,(1-o)\,(t-o)$$

$$\delta_i = z_i\,(1-z_i) \sum_{o \in Outputs} w_i \delta_o$$

$$\delta_j = z_j\,(1-z_j) \sum_{i \in NextLayer} w_j \delta_i$$

Back-Propagation

For a given input vector $\vec{x}$

Notation :

o_u output of every unit u in network

t desired output

$w_{i\to j}$ weight from unit i to unit j

$x_{i\to j}$ input from unit i going to unit j

Update weights rule :

$$w'_{i\to j} = w_{i\to j} + \eta\delta_j x_{i\to j}$$

Define :

$\delta_k = o_k(1-o_k)(t - o_k)$, when k is the output unit

$$\delta_k = o_k(1-o_k) \sum_{u\in \text{Outputs of unit } k} w_{k\to u}\delta_u$$

Back-Propagation Algorithm

- Propagate the input forward through the network
- Calculate the outputs of all nodes (hidden and output)
- Propagate the error backward
- Update the weights:

$$w_{r\to t} \leftarrow w_{r\to t} - \eta \frac{\partial E(w_{r\to t})}{\partial w_{r\to t}}$$

$$w_{r\to t} \leftarrow w_{r\to t} + \eta \cdot \delta_t \cdot x_r$$

Training with Back-Propagation

- Go once through all training examples & update the weights (1epoch)
- Iterate until a stopping criterion is satisfied
- The hidden layers learn new features and map to new spaces
- Training reaches a local minimum of the error surface

Overfitting with Neural Networks

- If number of hidden units (and weights) is large, it is easy to "memorize" the training set (or parts of it) and not generalize
- Typically, the optimal number of hidden units is much smaller than the input units
- Each hidden layer maps to a space of smaller dimension

Representational Power

- Perceptron: Can learn only linearly separable functions
- Functions learnable by a neural network
  - Boolean Functions: one hidden layer
  - Continuous Functions: one hidden layer and sigmoid units
  - Arbitrary Functions: two hidden layers and sigmoid units
- Number of hidden units in all cases unknown

ANN in Matlab

- Create an ANN

  net =feedforwardnet(hiddenSizes)
- [net] = train(net,X,T) takes a **network** net, **input data** X and **target data** T and returns the **network** after training it.
- sim(net, X) takes a **network** net and **inputs** X and returns the estimated **outputs** Y generated by the **network**.

- Example

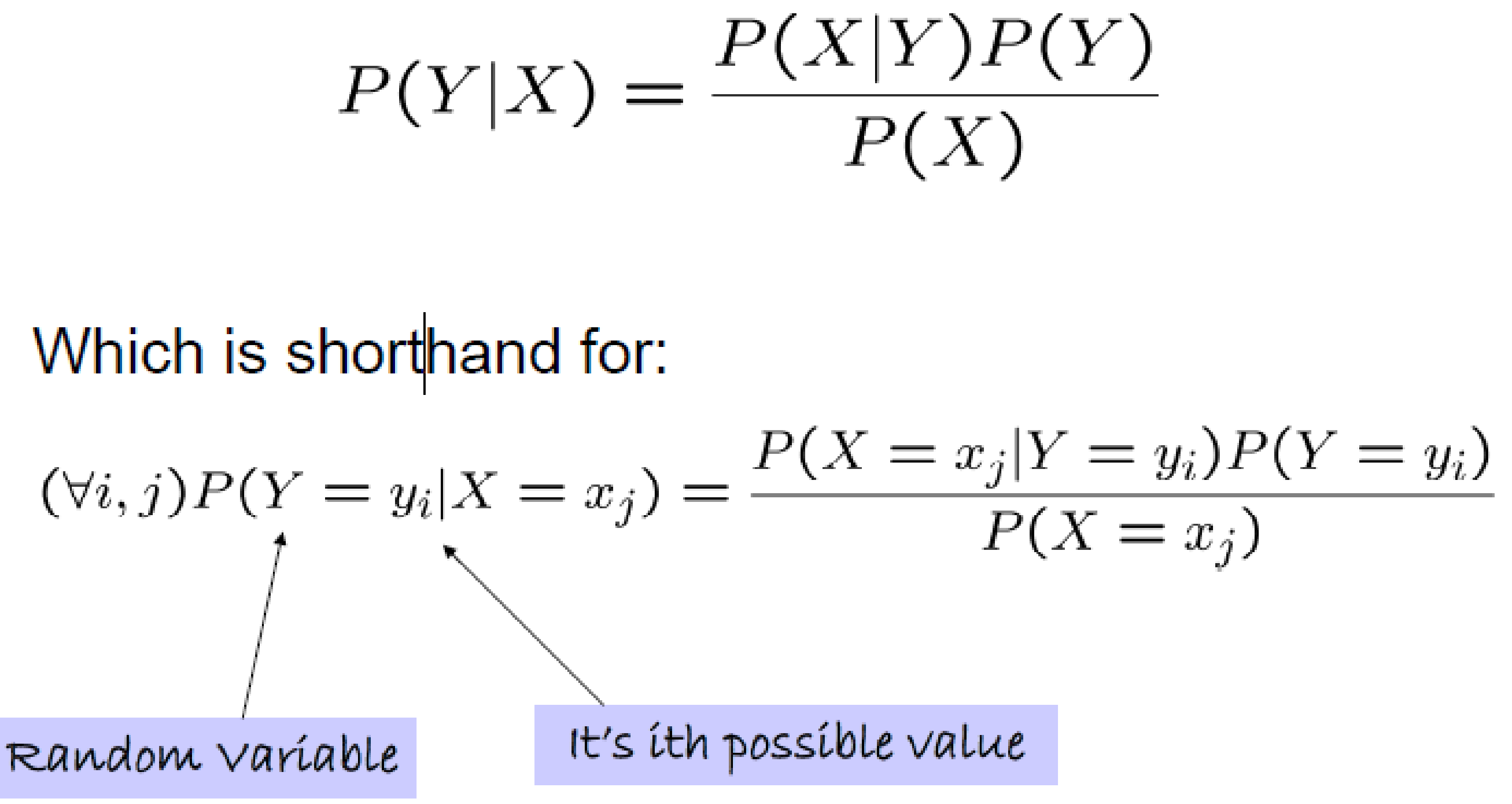
```
layers = [2 4];    % 2 hidden layers with size 2 and 4, respectively
net = feedforwardnet([2 4]);
net = init(net);          % initialize
% traindata is a struct that contains the training set
net = train(net,  traindata.examples,  traindata.labels);

% testdata is a struct that contains the testing set
predictions = sim(net,  testdata.examples);
```

Conclusions

- Can deal with both real and discrete domains
- Can also perform density or probability estimation
- Very fast classification time
- Relatively slow training time (does not easily scale to thousands of inputs)
- One of the most successful classifiers yet
- Successful design choices still a black art
- Easy to overfit or underfit if care is not applied

Algorithms:
Naïve Bayes Classifier

Bayes Rule



$$P(Y|X) = \frac{P(X|Y)P(Y)}{P(X)}$$

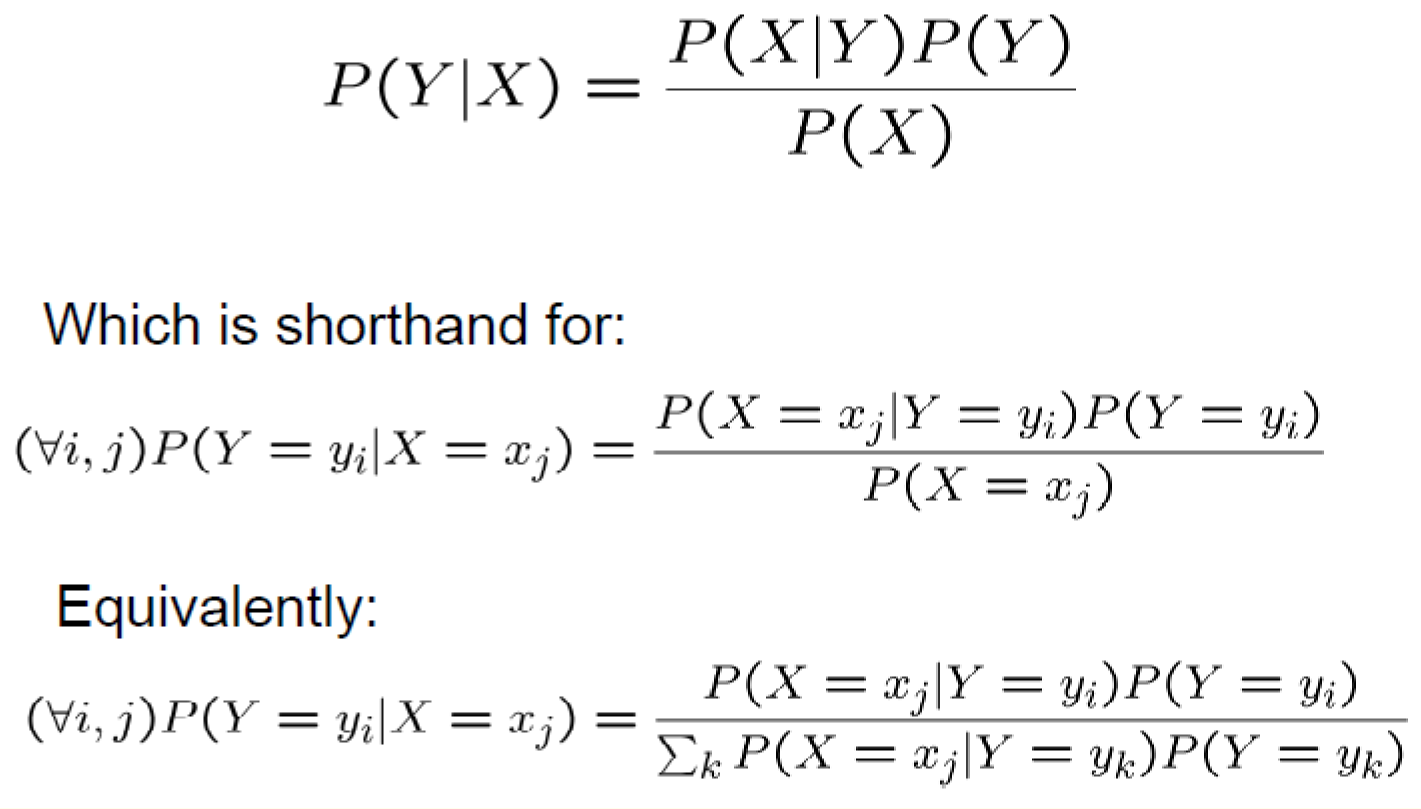
Which is shorthand for:

$$(\forall i, j) P(Y = y_i | X = x_j) = \frac{P(X = x_j | Y = y_i) P(Y = y_i)}{P(X = x_j)}$$

Random variable

It's ith possible value

Bayes Rule



$$P(Y|X) = \frac{P(X|Y)P(Y)}{P(X)}$$

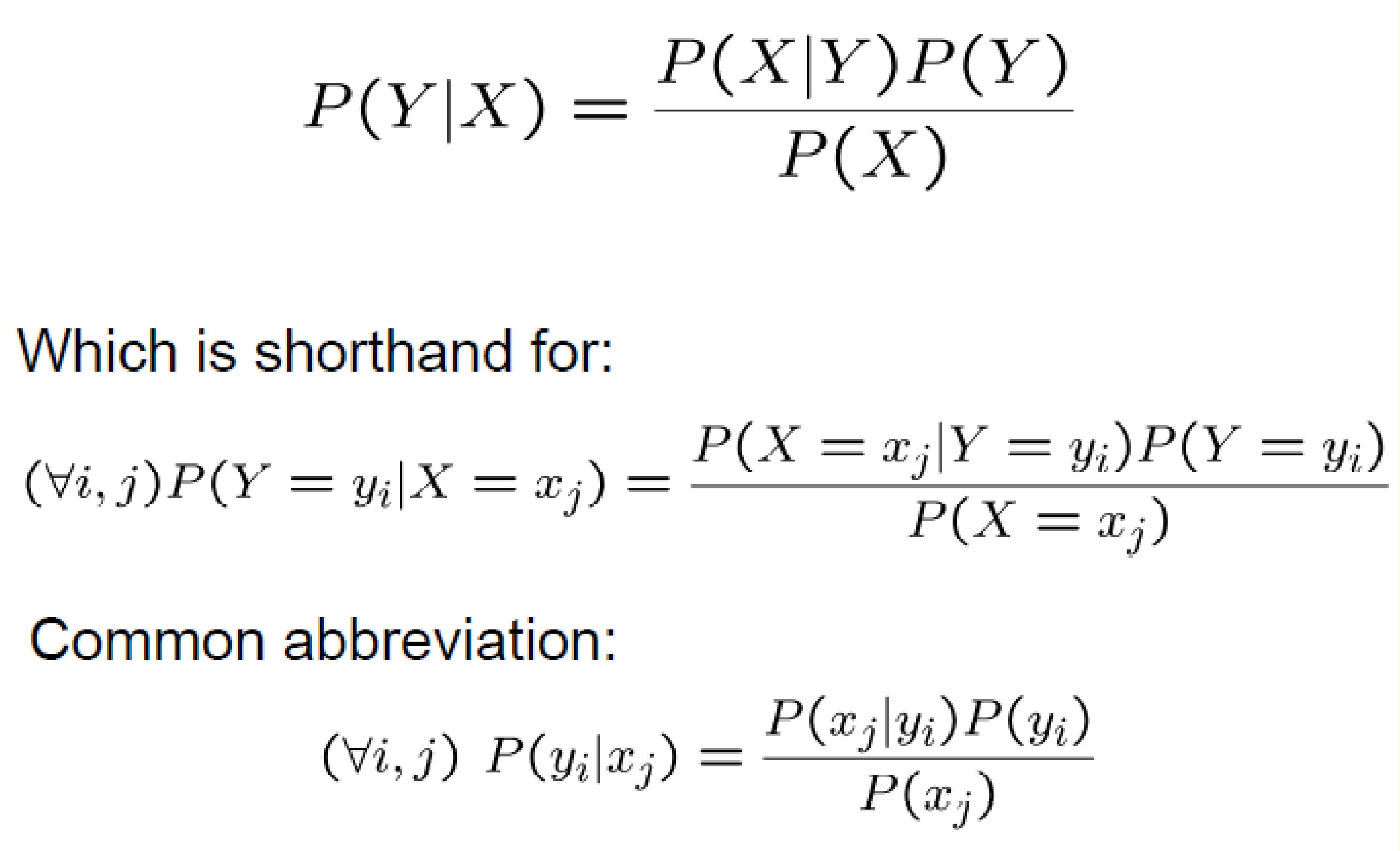
Which is shorthand for:

$$(\forall i, j) P(Y = y_i | X = x_j) = \frac{P(X = x_j | Y = y_i) P(Y = y_i)}{P(X = x_j)}$$

Equivalently:

$$(\forall i, j) P(Y = y_i | X = x_j) = \frac{P(X = x_j | Y = y_i) P(Y = y_i)}{\sum_k P(X = x_j | Y = y_k) P(Y = y_k)}$$

Bayes Rule



$$P(Y|X) = \frac{P(X|Y)P(Y)}{P(X)}$$

Which is shorthand for:

$$(\forall i, j) P(Y = y_i | X = x_j) = \frac{P(X = x_j | Y = y_i) P(Y = y_i)}{P(X = x_j)}$$

Common abbreviation:

$$(\forall i, j) \; P(y_i | x_j) = \frac{P(x_j | y_i) P(y_i)}{P(x_j)}$$

Bayes Classifier

- Training data:

X

Y

Sky	**Temp**	**Humid**	**Wind**	**Water**	**Enjoy**
Sunny	Warm	Normal	Strong	Warm	Yes
Sunny	Warm	High	Strong	Warm	Yes
Rain	Cold	High	Strong	Warm	No
Sunny	Warm	High	Strong	Cool	Yes

- Learning = estimating $P(X|Y), P(Y)$
- Classification = using Bayes rule to calculate $P(Y \mid X^{new})$
- X^{new} is a new example

Naïve Bayes Assumption

$X = <X_1, \ldots, X_n>$, n: dimensions of X

Y discrete-valued

- X_i and X_j are conditionally independent given Y, for all i≠j

$$P(X_1 \ldots X_n|Y) = \prod_i P(X_i|Y)$$

Naïve Bayes classification

- Bayes rule:

$$P(Y = y_k | X_1 \ldots X_n) = \frac{P(Y = y_k) P(X_1 \ldots X_n | Y = y_k)}{\sum_j P(Y = y_j) P(X_1 \ldots X_n | Y = y_j)}$$

- Assuming conditional independence:

$$P(Y = y_k | X_1 \ldots X_n) = \frac{P(Y = y_k) \prod_i P(X_i | Y = y_k)}{\sum_j P(Y = y_j) \prod_i P(X_i | Y = y_j)}$$

- So, the classification rule for a new example $X^{new} = <X_i, \ldots, X_n>$

$$Y^{new} \leftarrow \arg\max_{y_k} \; P(Y = y_k) \prod_i P(X_i | Y = y_k)$$

Naïve Bayes Algorithm

- Train Naïve Bayes (examples)

```
for each label $y_k$
    estimate $P(Y = y_k)$
    for each value $x_{ij}$ of each predictor $X_i$
        estimate $P(X_i = x_{ij} | Y = y_k)$
    end
end
```

$$Y^{new} \leftarrow \arg\max_{y_k} \quad P(Y = y_k) \prod_i P(X_i | Y = y_k)$$

- Classify a new example X^{new}

Estimating Parameters: *Y*, *X*$_i$ *discrete-valued*

- Parameter estimation:

$$\hat{\pi}_k = \hat{P}(Y = y_k) = \frac{\#D\{Y = y_k\}}{|D|}$$

$$\hat{\theta}_{ijk} = \hat{P}(X_i = x_{ij} | Y = y_k) = \frac{\#D\{X_i = x_{ij} \wedge Y = y_k\}}{\#D\{Y = y_k\}}$$

What if we have continuous *Xi* ?

- Gaussian Naïve Bayes (GNB) assume

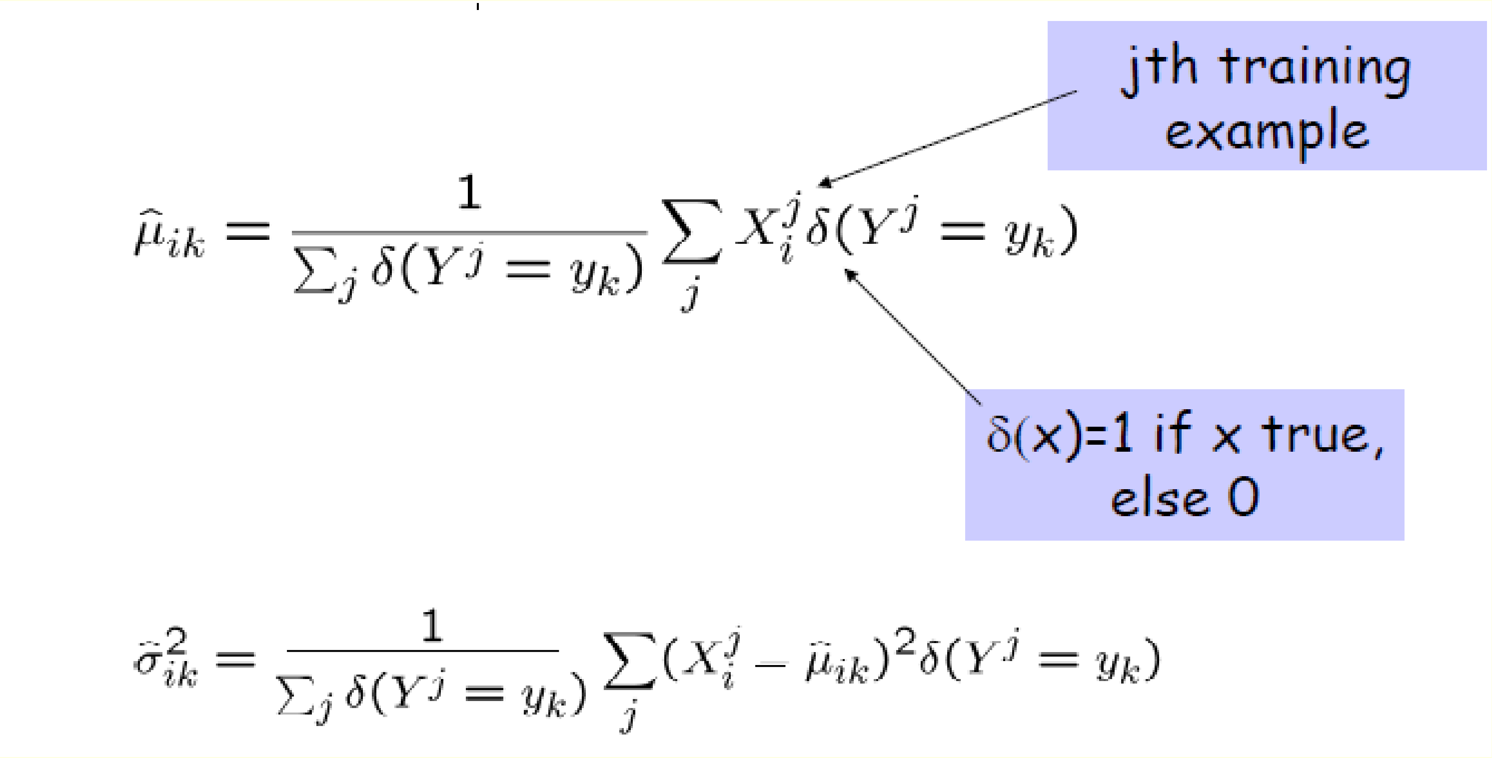
$$P(X_i = x \mid Y = y_k) = \frac{1}{\sigma_{ik}\sqrt{2\pi}} e^{\frac{-(x-\mu_{ik})^2}{2\sigma_{ik}^2}}$$

Sometimes assume variance

- is independent of Y (i.e., σi),
- or independent of Xi (i.e., σk)
- or both (i.e., σ)

Estimating Parameters: *Y discrete*, X_i *continuous*

- Maximum likelihood estimates:



jth training example

$$\hat{\mu}_{ik} = \frac{1}{\sum_j \delta(Y^j = y_k)} \sum_j X_i^j \delta(Y^j = y_k)$$

δ(x)=1 if x true, else 0

$$\hat{\sigma}_{ik}^2 = \frac{1}{\sum_j \delta(Y^j = y_k)} \sum_j (X_i^j - \hat{\mu}_{ik})^2 \delta(Y^j = y_k)$$

Naïve Bayes in Matlab

- Create a new Naïve object:

  nb = NaiveBayes.fit(X,Y), X is a matrix of predictor values, Y is a vector of n class labels

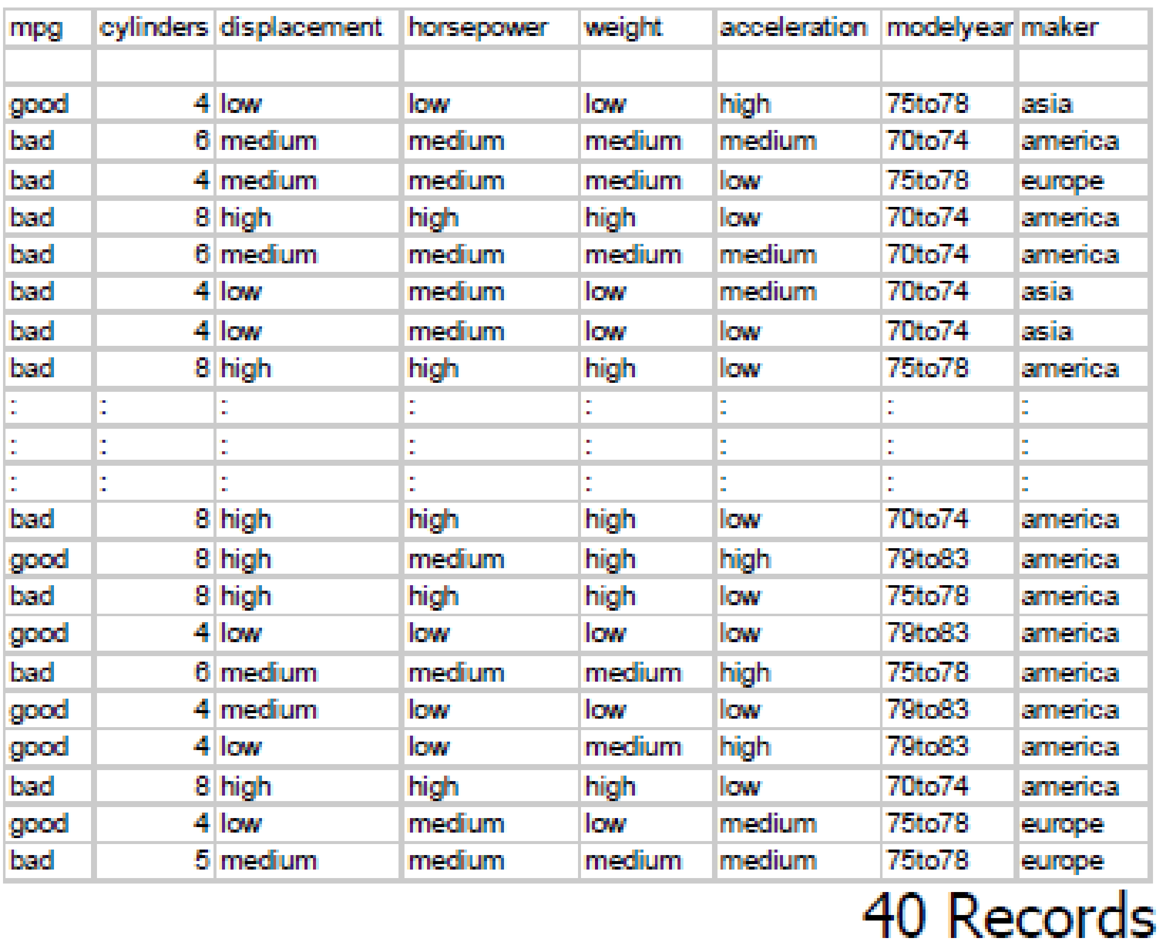
- post = posterior(nb, test) returns the posterior probability of the observations in test
- Predict a value

  predictedValue = predict(nb, test)

Algorithms: Decision Trees

A small dataset: Miles Per Gallon

- Suppose we want to predict MPG

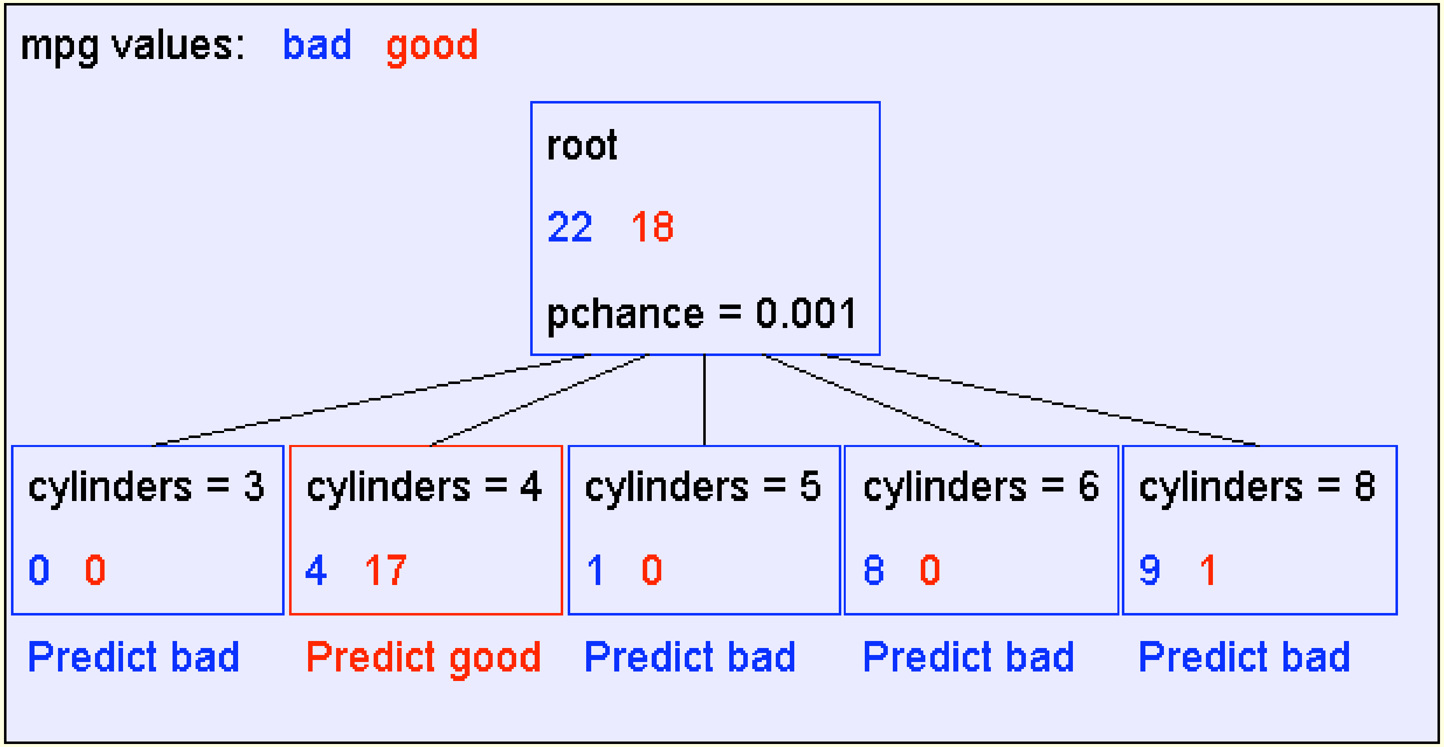


mpg	cylinders	displacement	horsepower	weight	acceleration	modelyear	maker
good	4	low	low	low	high	75to78	asia
bad	6	medium	medium	medium	medium	70to74	america
bad	4	medium	medium	medium	low	75to78	europe
bad	8	high	high	high	low	70to74	america
bad	6	medium	medium	medium	medium	70to74	america
bad	4	low	medium	low	medium	70to74	asia
bad	4	low	medium	low	low	70to74	asia
bad	8	high	high	high	low	75to78	america
:	:	:	:	:	:	:	:
:	:	:	:	:	:	:	:
:	:	:	:	:	:	:	:
bad	8	high	high	high	low	70to74	america
good	8	high	medium	high	high	79to83	america
bad	8	high	high	high	low	75to78	america
good	4	low	low	low	low	79to83	america
bad	6	medium	medium	medium	high	75to78	america
good	4	medium	low	low	low	79to83	america
good	4	low	low	medium	high	79to83	america
bad	8	high	high	high	low	70to74	america
good	4	low	medium	low	medium	75to78	europe
bad	5	medium	medium	medium	medium	75to78	europe

40 Records

- From the UCI repository

A Decision Stump



mpg values: bad good

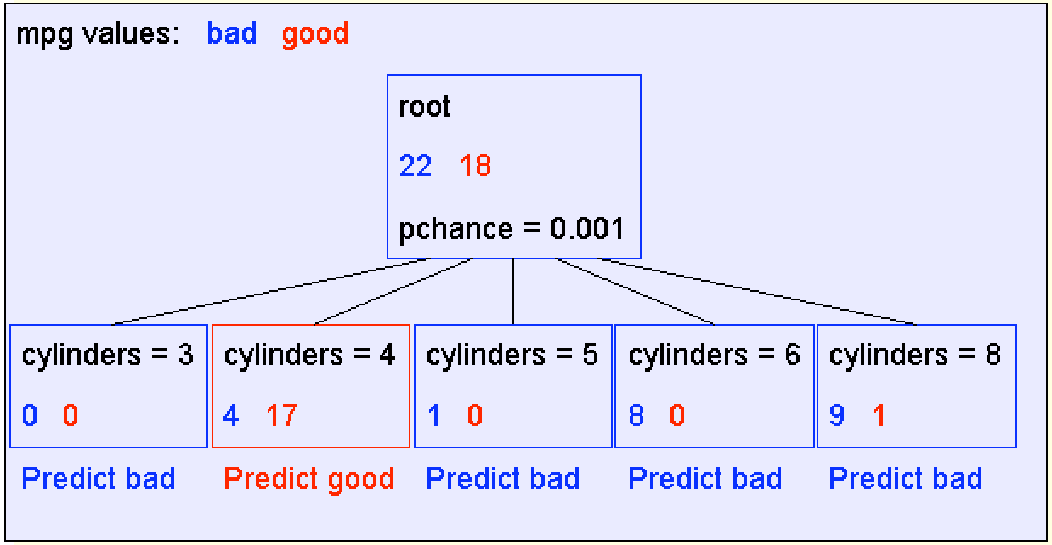
root

22 18

pchance = 0.001

cylinders = 3	cylinders = 4	cylinders = 5	cylinders = 6	cylinders = 8
0 0	4 17	1 0	8 0	9 1
Predict bad	Predict good	Predict bad	Predict bad	Predict bad

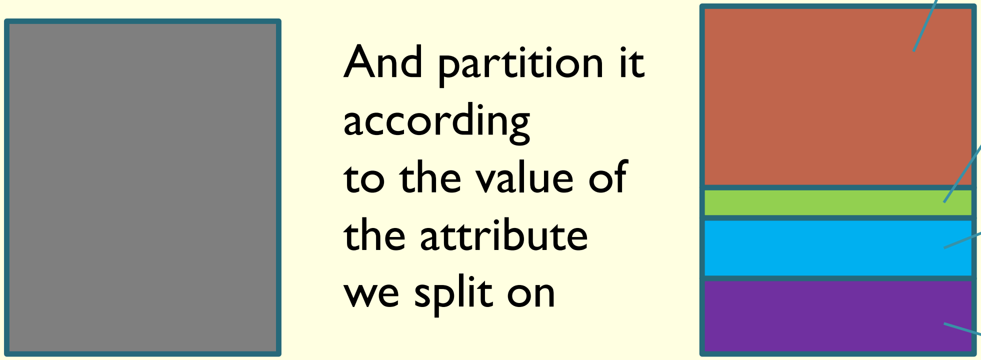
Recursion Step



mpg values: bad good

root
22 18
pchance = 0.001

cylinders = 3	cylinders = 4	cylinders = 5	cylinders = 6	cylinders = 8
0 0	4 17	1 0	8 0	9 1
Predict bad	Predict good	Predict bad	Predict bad	Predict bad



Take the Original Dataset..

And partition it according to the value of the attribute we split on

Records in which cylinders = 4

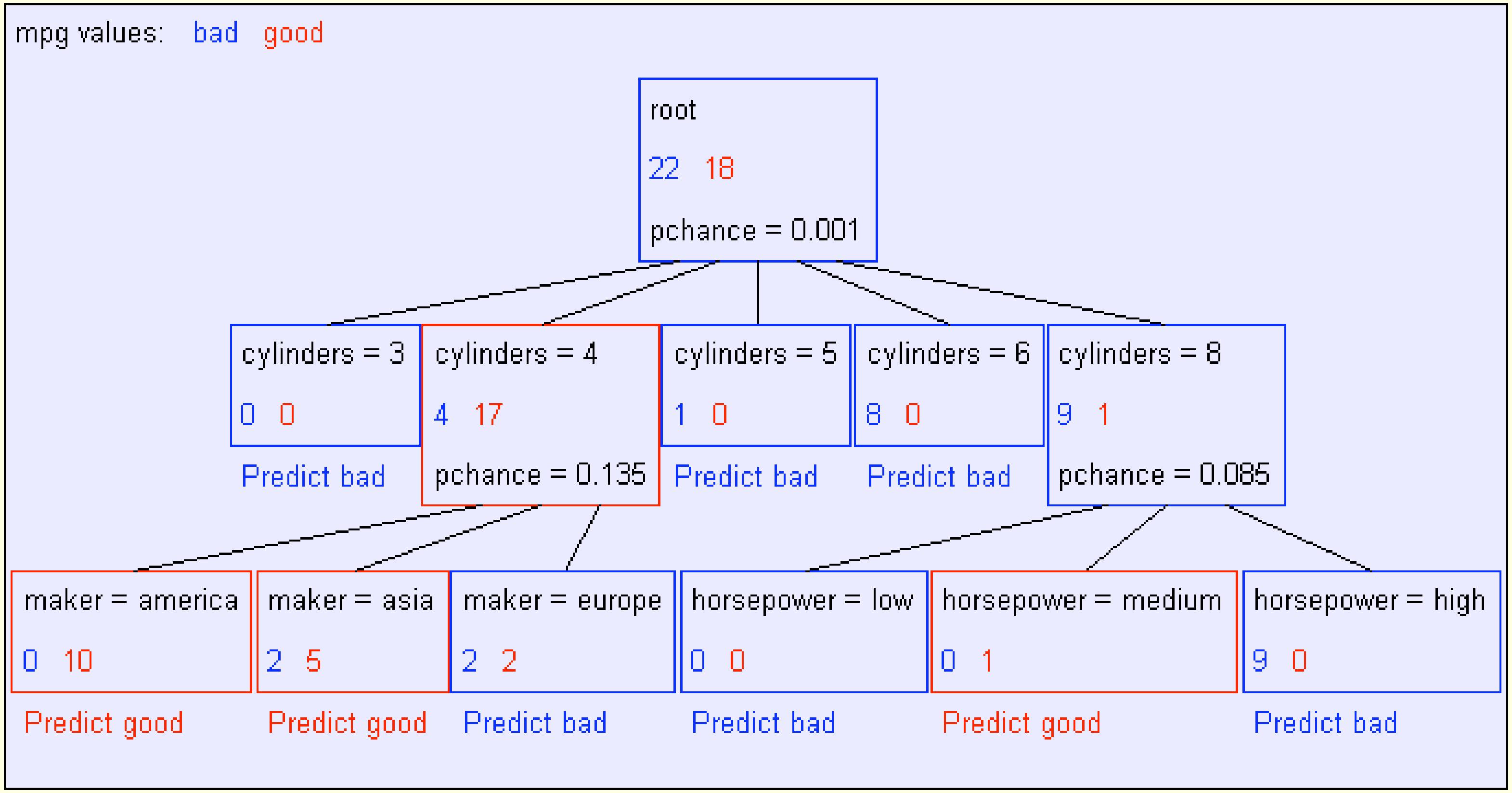
Records in which cylinders = 5

Records in which cylinders = 6

Records in which cylinders = 8

Build Tree from these Records

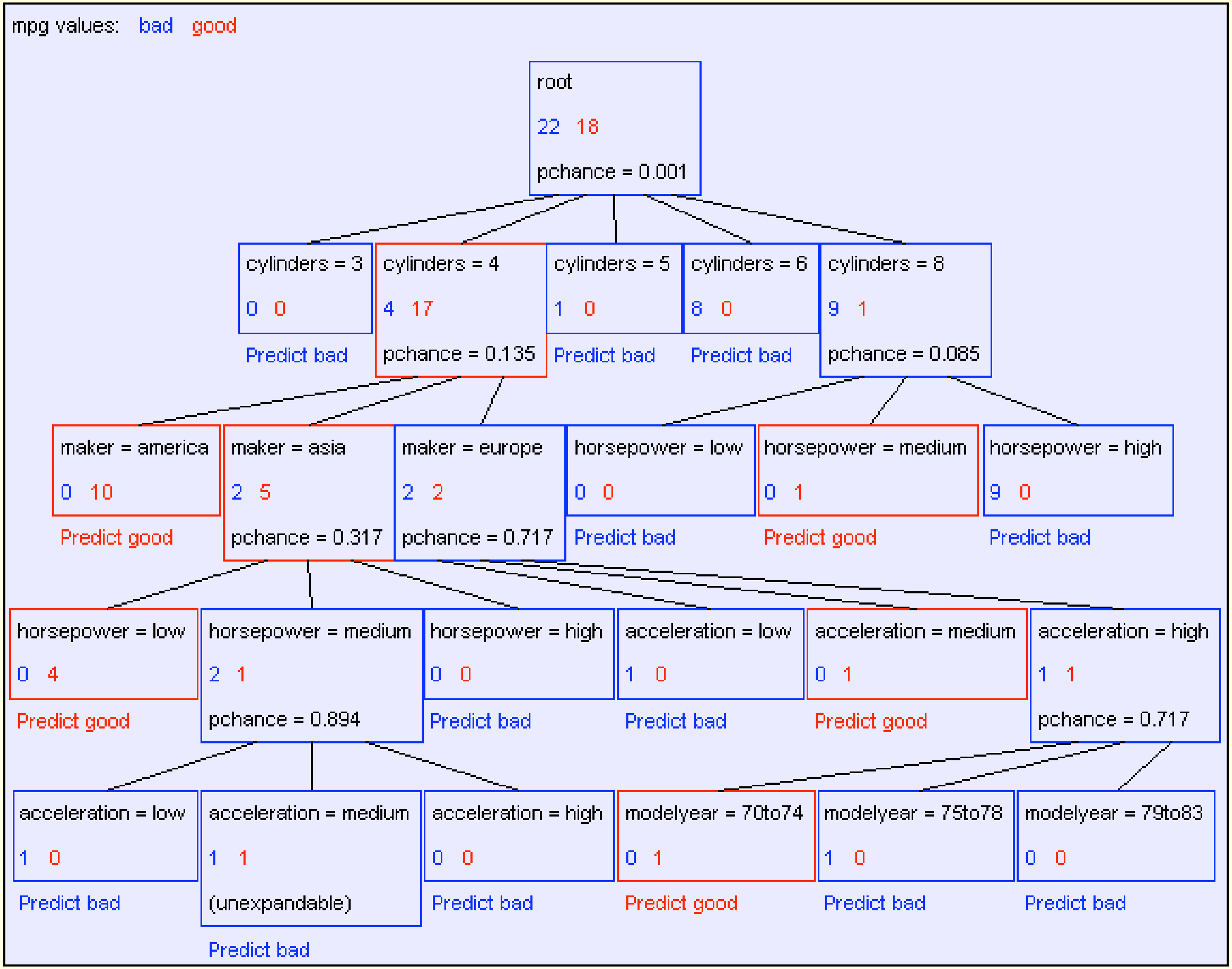
Second level of tree



Recursively build a tree from the seven records in which there are four cylinders and the maker was based in Asia

(Similar recursion in the other cases)

The final tree



Classification of a new example

- Classifying a test example
- Traverse tree
- Report leaf label

Learning decision trees is hard!!!

- Learning the simplest (smallest) decision tree is an NP-complete problem [Hyafil & Rivest ’76]

- Resort to a greedy heuristic:
  - Start from empty decision tree
  - Split on **next best attribute (feature)**
  - Recurse
- How to choose the best attribute and the value for a split?

Entropy

- Entropy characterizes our uncertainty about our source of information
- ***More uncertainty, more entropy!***
    - *Information Theory interpretation: $H(Y)$ is the expected number of bits needed* to encode a randomly drawn value of Y *(under most efficient code)*

Information gain

- Advantage of attribute – decrease in uncertainty
  - Entropy of Y before you split

$$H(Y) = -\sum_{i=1}^{k} P(Y = y_i) \log_2 P(Y = y_i)$$

  - Entropy after split
    - Weight by probability of following each branch, i.e., normalized number of records

$$H(Y \mid X) = -\sum_{j=1}^{v} P(X = x_j) \sum_{i=1}^{k} P(Y = y_i \mid X = x_j) \log_2 P(Y = y_i \mid X = x_j)$$

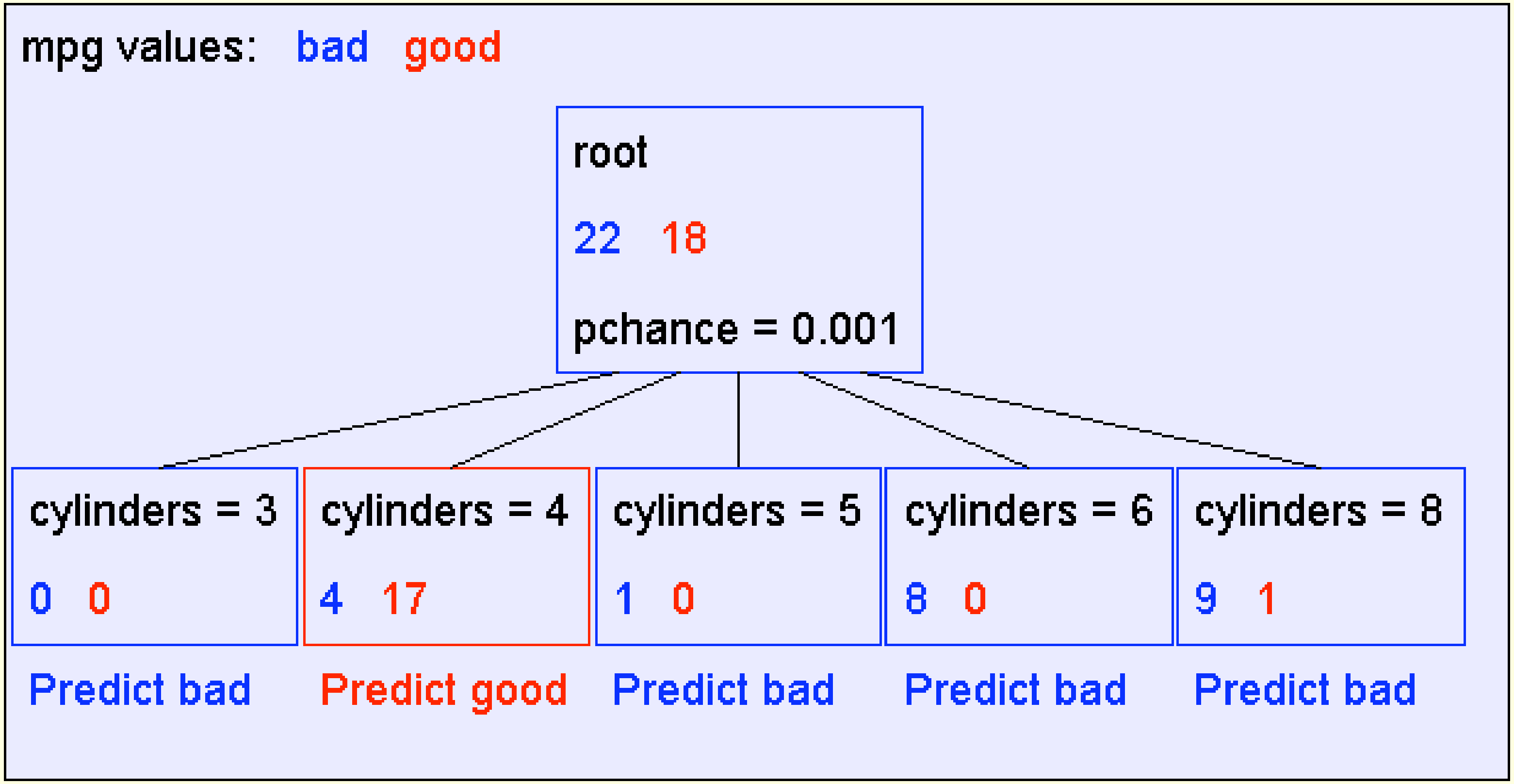
- Information gain is difference

$$IG(X) = H(Y) - H(Y \mid X)$$

Learning decision trees

- Start from empty decision tree
- Split on **next best attribute (feature)**
  - Use, for example, information gain to select attribute
  - Split on $\arg\max_i IG(X_i) = \arg\max_i H(Y) - H(Y \mid X_i)$
- Recurse

A Decision Stump



mpg values: bad good

root

22 18

pchance = 0.001

cylinders = 3

0 0

Predict bad

cylinders = 4

4 17

Predict good

cylinders = 5

1 0

Predict bad

cylinders = 6

8 0

Predict bad

cylinders = 8

9 1

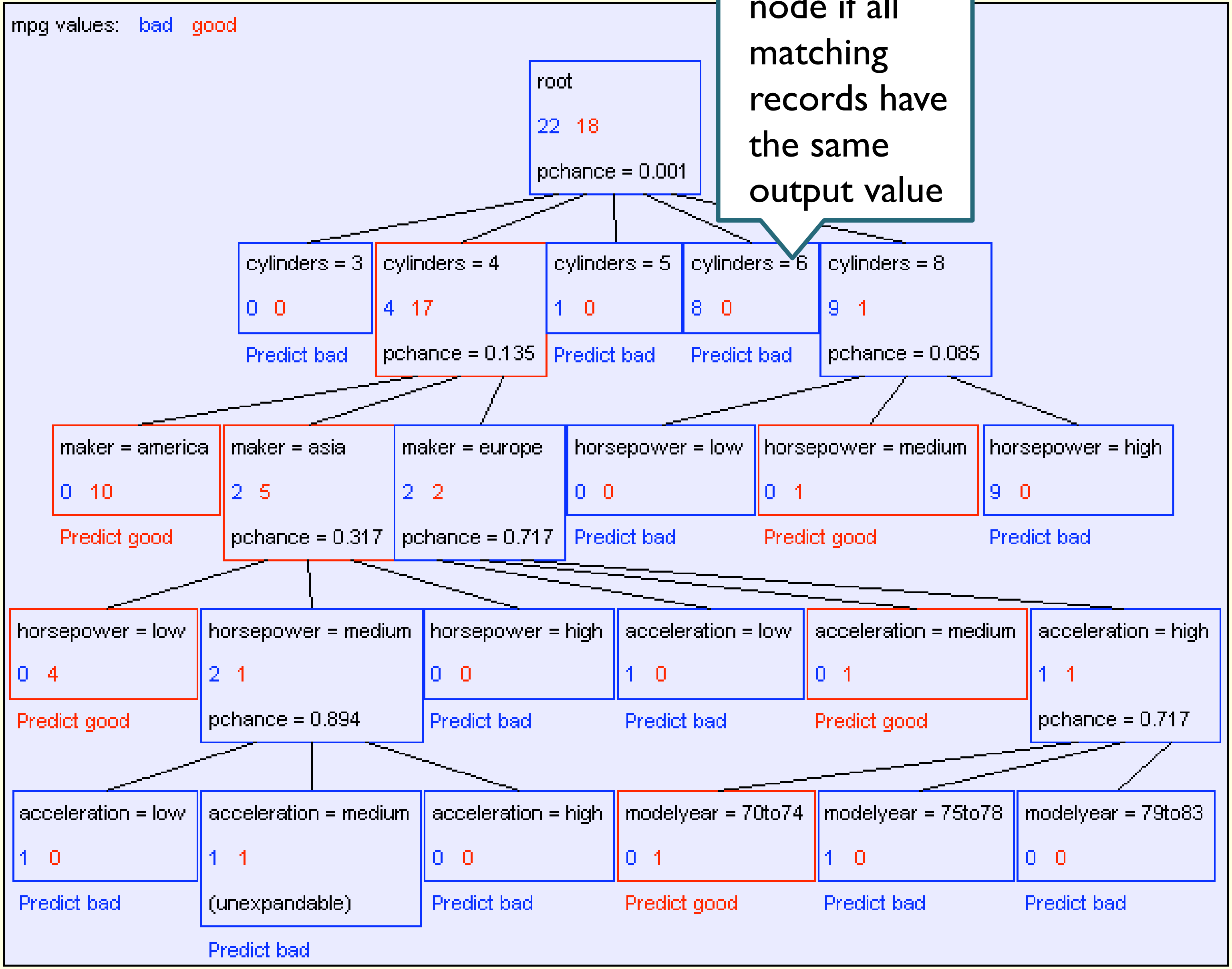
Predict bad

Base Cases

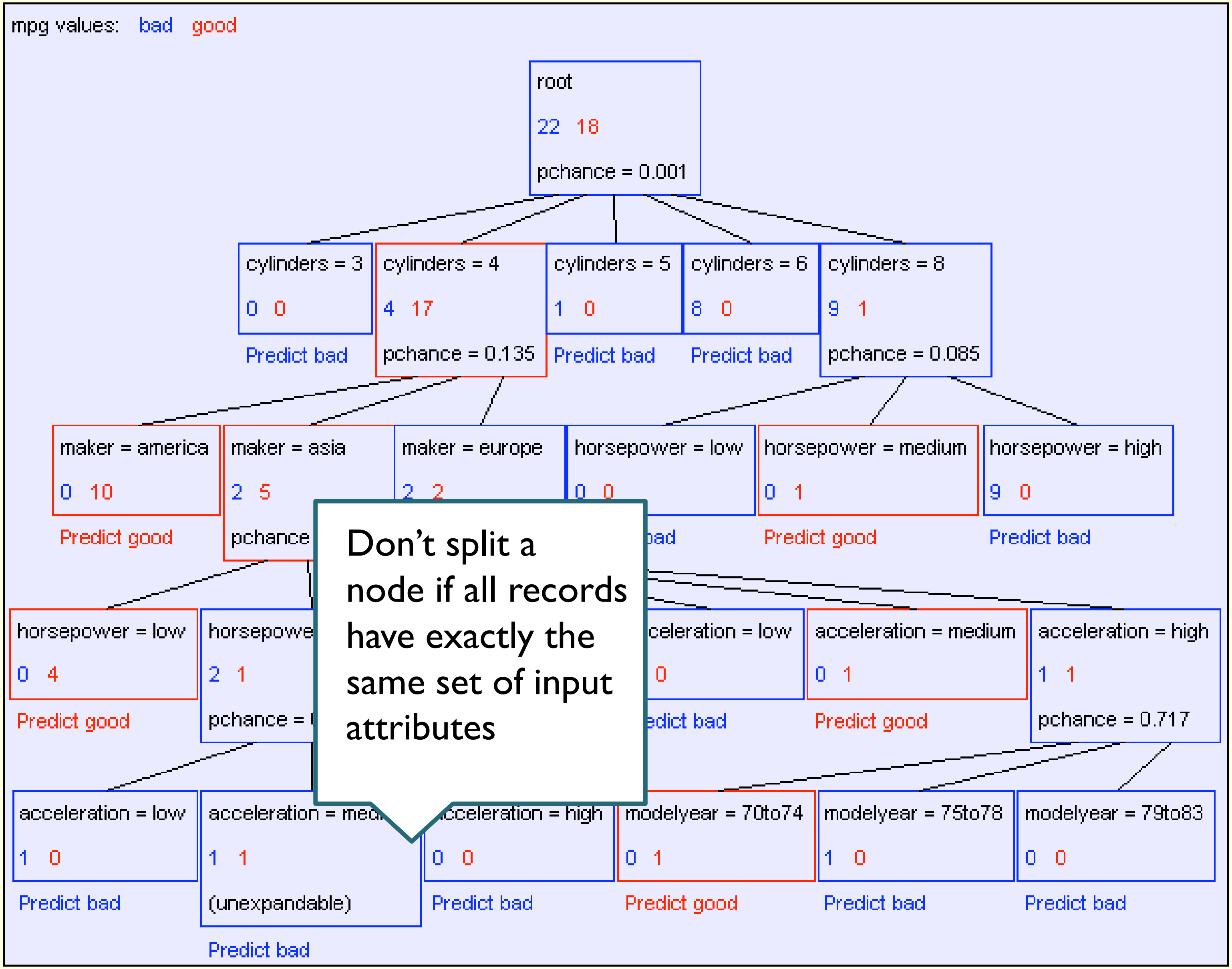
- Base Case One: If all records in current data subset have the same output then **don’t recurse**
- Base Case Two: If all records have exactly the same set of input attributes then **don’t recurse**

Base Case 1

Don't split a node if all matching records have the same output value



Base Case 2



Don't split a node if all records have exactly the same set of input attributes

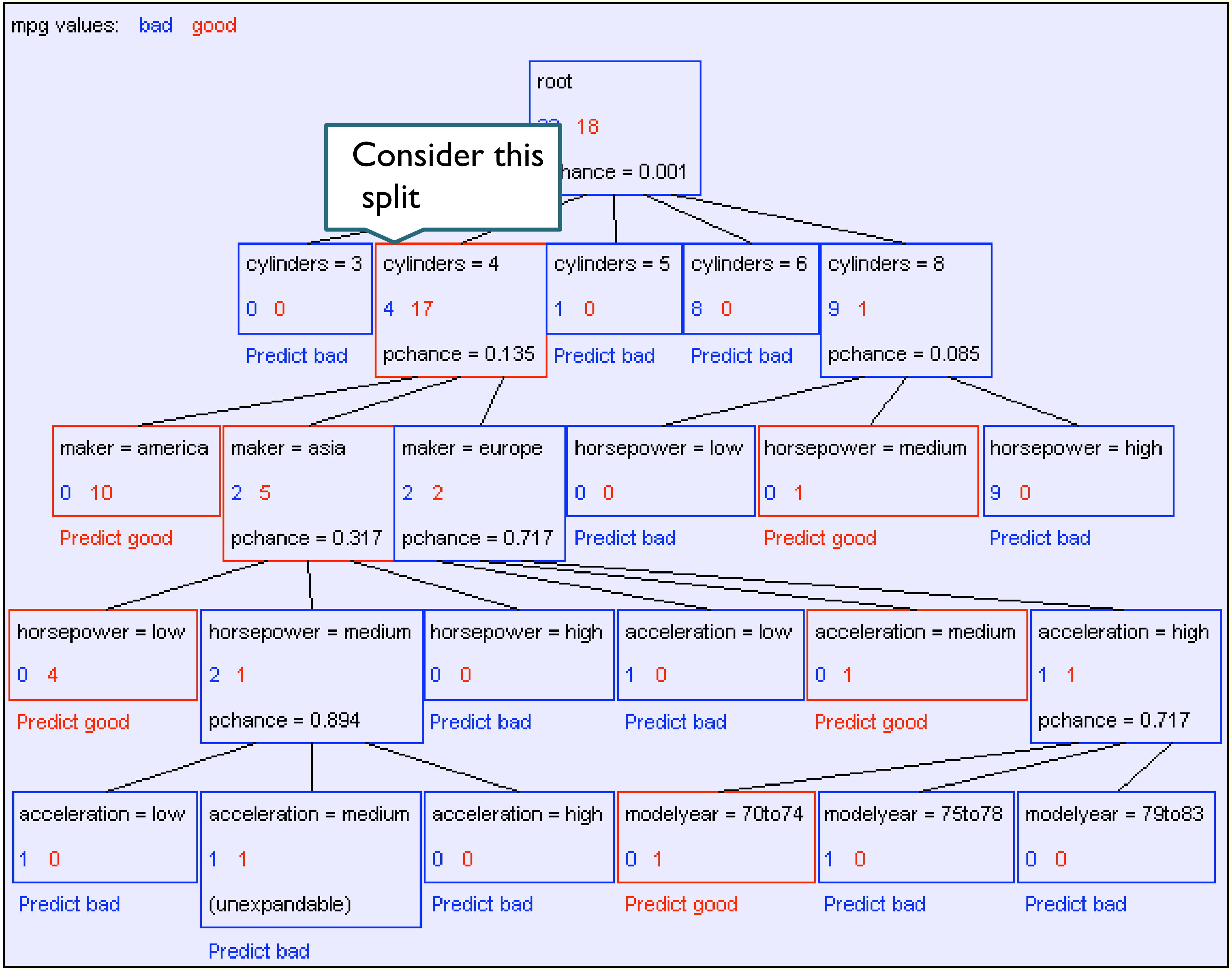
Basic Decision Tree Building Summarized

- BuildTree(*DataSet,Output*)
- If all output values are the same in *DataSet, return a leaf node that says* "predict this unique output"
- If all input values are the same, return a leaf node that says "predict the majority output"
- Else find attribute *X with highest Info Gain*
- Suppose *X has* n_X *distinct values (i.e. X has arity* n_X*).*
    - Create and return a non-leaf node with n_X *children.*
    - The *i'th child should be built by calling* BuildTree(*DSi,Output*)

      Where *DSi built consists of all those records in DataSet for which X = ith* distinct value of X.

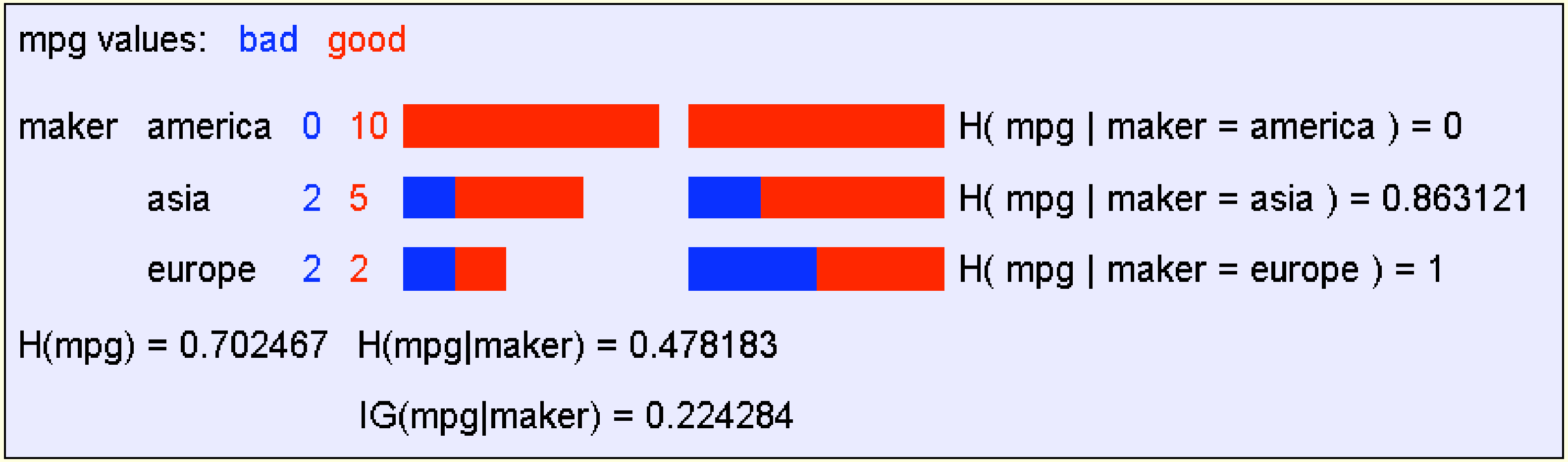
Decision trees will overfit

- Standard decision trees are have no learning biased
  - Training set error is always zero!
    - (If there is no label noise)
  - Lots of variance
  - Will definitely overfit!!!
  - Must bias towards simpler trees
- Many strategies for picking simpler trees:
  - Fixed depth
  - Fixed number of leaves
  - Or something smarter…



Consider this split

A statistical test



mpg values: bad good

maker	america	0	10	H(mpg | maker = america) = 0
	asia	2	5	H(mpg | maker = asia) = 0.863121
	europe	2	2	H(mpg | maker = europe) = 1

H(mpg) = 0.702467 H(mpg|maker) = 0.478183

IG(mpg|maker) = 0.224284

- Suppose that mpg was completely uncorrelated with maker.
- What is the chance we’d have seen data of at least this apparent level of association anyway?

Using to avoid overfitting

- Build the full decision tree as before
- But when you can grow it no more, start to prune:
  - Beginning at the bottom of the tree, delete splits in which have extreme low chance to appear//*pchance* > *MaxPchance*
  - Continue working your way up until there are no more prunable nodes

What you need to know about decision trees

- Decision trees are one of the most popular data mining tools
  - Easy to understand
  - Easy to implement
  - Easy to use
  - Computationally cheap (to solve heuristically)
- Information gain to select attributes (ID3, C4.5,…)
- Presented for classification, can be used for regression and density estimation too
- Decision trees will overfit!!!
  - Zero bias classifier ! Lots of variance
  - Must use tricks to find "simple trees", e.g.,
    - Fixed depth/Early stopping
    - Pruning

Decision trees in Matlab

- Use classregtree class
- Create a new tree:

  t=classregtree(X,Y), X is a matrix of predictor values, y is a vector of n response values
- Prune the tree:

  tt = prune(t, alpha, pChance) alpha defines the level of the pruning
- Predict a value

  y= eval(tt, X)

Performance Estimation

- Need to produce a single, final model
- But also estimate its performance
- Why estimate performance
  - Know what to expect out of a model / system
  - Select the best model out of all possible models
  - Compare different learning algorithms
- Probably the most underestimated problem in machine learning, data mining, pattern recognition

Ideal Performance Estimation

1. Learn a model from samples in S (train-set)
2. Install the model in its intended operational environment
3. Observe its operation for some time, for new cases S’
4. Label with a gold-standard the cases in S’ (test-set)
5. Estimate the performance of the model on S’

Ideal Performance Estimation

Golden Rule:

Simulate: learn from S, make operational, **test** on **new** samples S’

Simulating the Ideal


Train

Test

- Randomly split original data
- Learn on Train
- Test on Test
- Called hold-out estimation
- Can it go wrong?

K-Fold Cross-Validation

Train	Train	Train	Test	Train

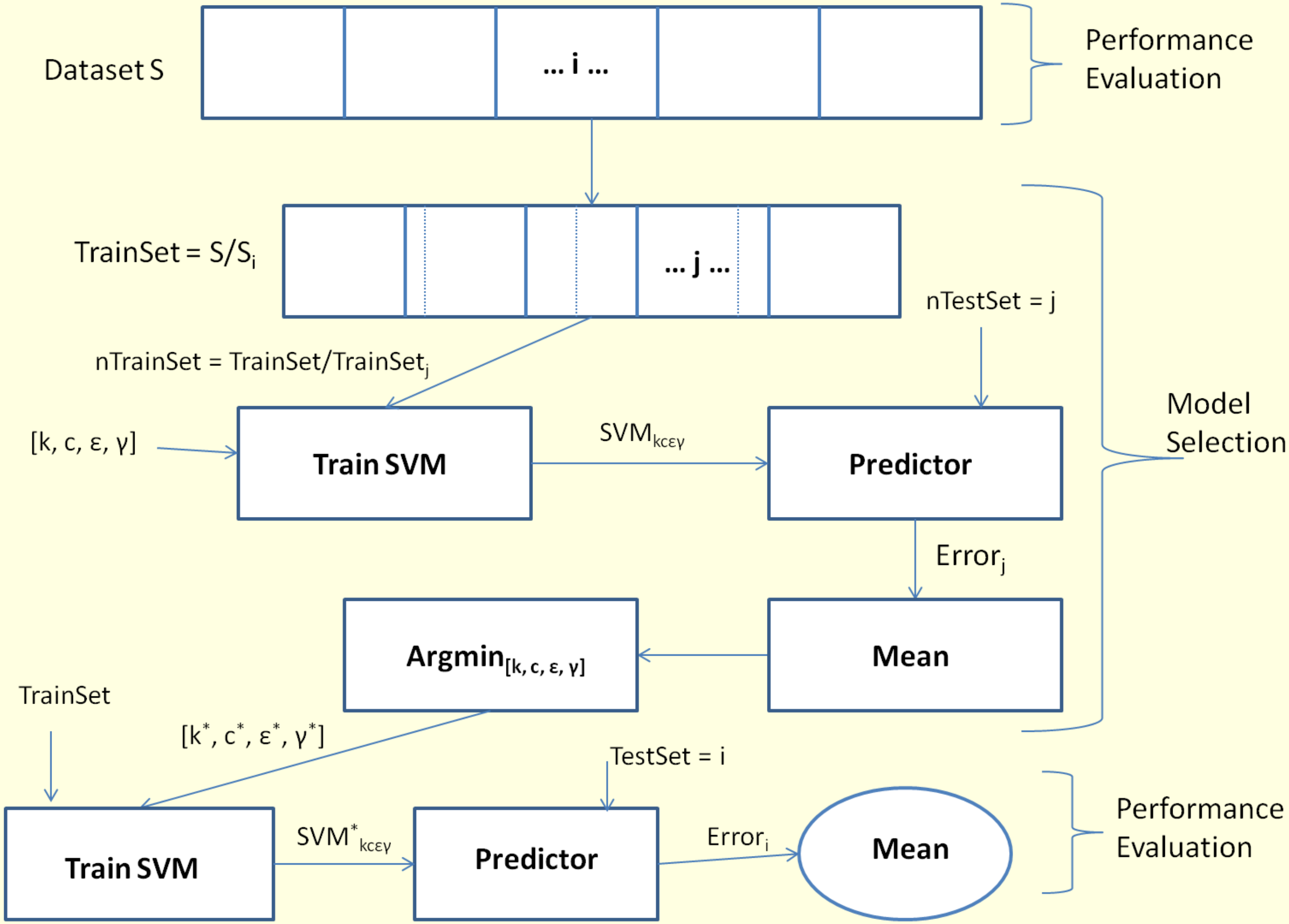
- Split to K-folds
- Cross-Validation(Data **D,** number K**)**
  - Randomly split **D** to K folds
  - Returned Model: f(D)

Nested Cross Validation

- Different combinations of model parameters
- n = 5 folds

Input: Dataset S, n, k, γ, ε, c

Output: Mean(Error_i)



Application on VoIP in Wireless Networks

Motivation

- Wide use of wireless services for communication
- Quality of Service (QoS):
  - Objective network-based metrics (e.g., delay, packet loss)
- Quality of Experience (QoE):
  - Objective and subjective performance metric (e.g., E-model, PESQ)
  - Objective factors: network, application related
  - Subjective factors: users expectation (MOS)

Problem Definition

- Users are not likely to provide QoE feedback
    - unless bad QoE is witnessed
- Estimation of QoE
    - difficult because of the many contributing factors using Opinion Models
- Use of machine learning algorithms for the estimation of the QoE
    - based on QoS metrics

Proposed Method

- Nested Cross Validation training of
  - ANN Models
  - GNB Models
  - Decision Trees models
- Preprocessing of data: normalization

Dataset

- 25 users
- 18 samples (segments of VoIP calls)
- Each user evaluated all the segments with QoE score
- 10 attributes as predictors

Dataset

- Predictors
  - **average delay, packet loss, average jitter, burst ratio, average burst interarrival, average burst size, burst size variance, delay variance, jitter variance, burst interarrival variance**
- QoE score

Experiments and Results

- For ANN we tested different values of nodes at the first and the second hidden layer, with and no normalization of the data
- In this table we can see some statistics from the error which appears from the difference between the estimated QoE and the real QoE

	ANN
Mean error	0.9018
Median error	0.6181
Std error	1.0525

Experiments and Results

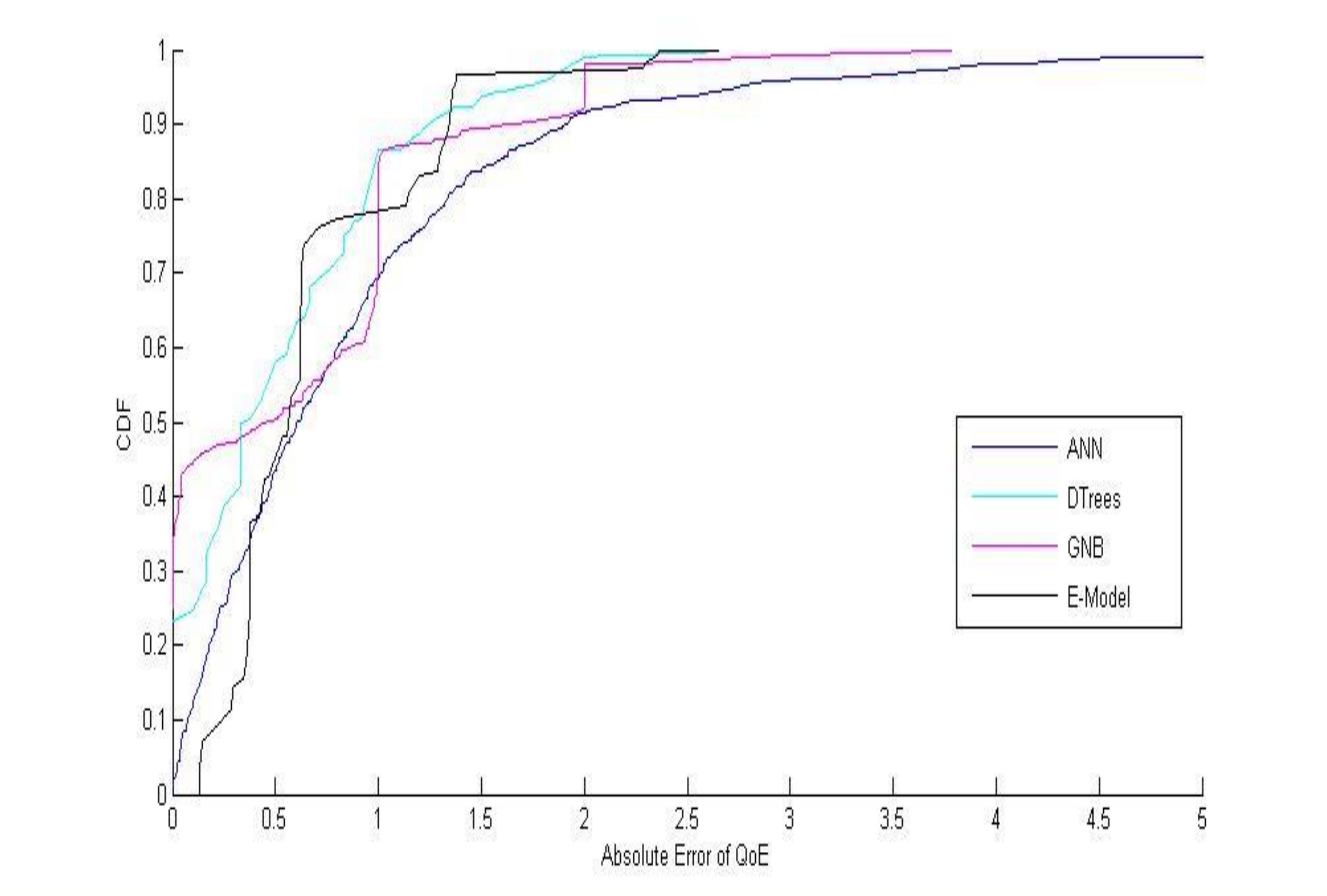
- In order to train the GNB models we use the data with normalization or not.
- Statistics from the error of this model:

	GNB
Mean error	0.9018
Median error	0.6181
Std error	1.0525

Experiments and Results

- For the Decision Trees we used different values of alpha (a) parameter which defines the pruning level of the tree.
- Statistics:

	Decision Trees
Mean error	0.5475
Median error	0.3636
Std error	0.5395


CDF
1
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0
0
0.5
1
1.5
2
2.5
3
3.5
4
4.5
5
Absolute Error of QoE
ANN
DTrees
GNB
E-Model

Material

Sources:

- Lectures from Machine Learning course CS577