Classification

Albert Bifet (@abifet)

Paris, 4 October 2016

albert.bifet@telecom-paristech.fr

Classification

Definition

Given n_C different classes, a classifier algorithm builds a model that predicts for every unlabelled instance I the class C to which it belongs with accuracy.

Example

A spam filter

Example

Twitter Sentiment analysis: analyze tweets with positive or negative feelings

Classification

Data set that describes e-mail features for deciding if it is spam.

Example

Contains "Money"	**Domain type**	**Has attach.**	**Time received**	**spam**
yes	com	yes	night	yes
yes	edu	no	night	yes
no	com	yes	night	yes
no	edu	no	day	no
no	com	no	day	no
yes	cat	no	day	yes

Assume we have to classify the following new instance:

Contains "Money"	**Domain type**	**Has attach.**	**Time received**	**spam**
yes	edu	yes	day	?

Majority Class Classifier

Majority Class Classifier

- Training: compute majority class of the dataset
- Prediction:
  - Output majority class of the dataset

k-Nearest Neighbours

k-NN Classifier

- Training: store all instances in memory
- Prediction:
  - Find the k nearest instances
  - Output majority class of these k instances
- Performance:
  - Nearest neighbor search
  - Depends on data structure used:
    - linear search
    - space partitioning (tree)

Bayes Classifiers

Naïve Bayes

- Based on Bayes Theorem:

$$P(c|d) = \frac{P(c)P(d|c)}{P(d)}$$

$$posterior = \frac{prior \times likelikood}{evidence}$$

- Estimates the probability of observing attribute a and the prior probability $P(c)$
- Probability of class c given an instance d:

$$P(c|d) = \frac{P(c)\prod_{a \in d} P(a|c)}{P(d)}$$

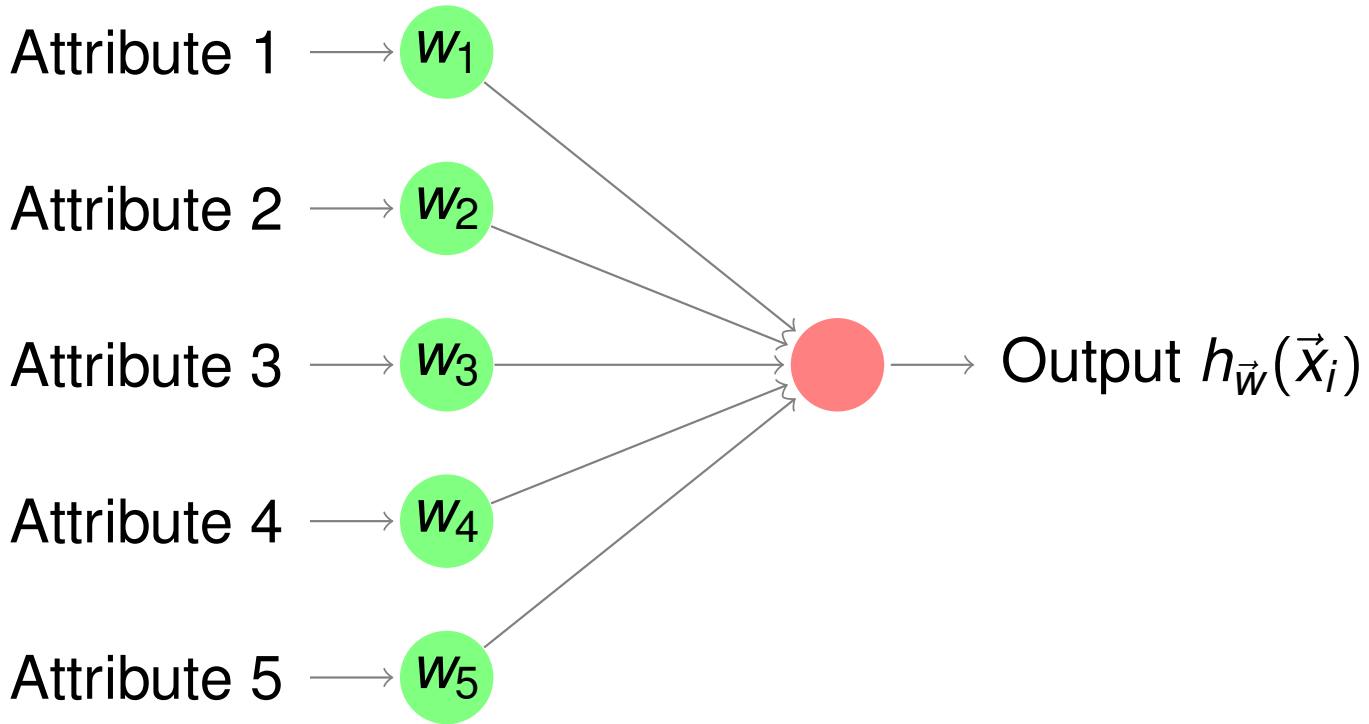
Bayes Classifiers

Multinomial Naïve Bayes

- Considers a document as a bag-of-words.
- Estimates the probability of observing word w and the prior probability $P(c)$
- Probability of class c given a test document d:

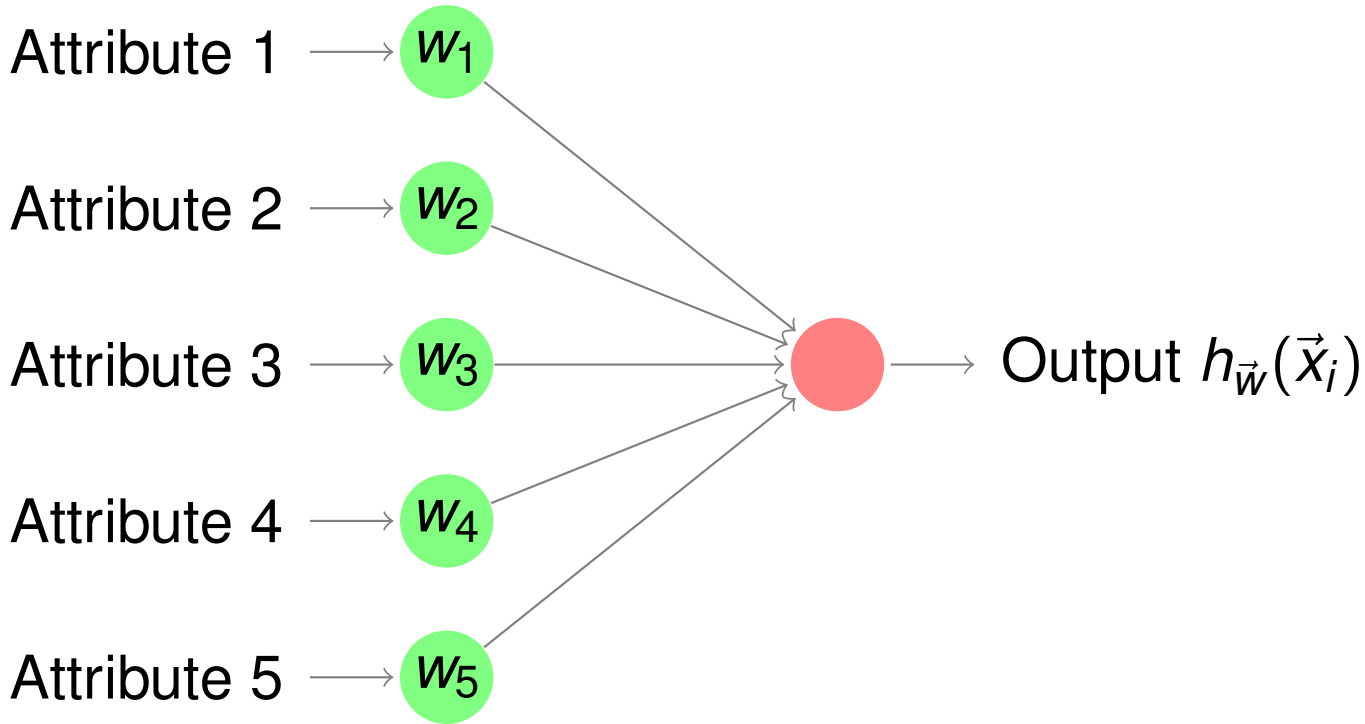
$$P(c|d) = \frac{P(c)\prod_{w \in d} P(w|c)^{n_{wd}}}{P(d)}$$

Perceptron/Neuron



- Data stream: $\langle \vec{x}_i, y_i \rangle$
- Classical perceptron: $h_{\vec{w}}(\vec{x}_i) = \mathrm{sgn}(\vec{w}^T \vec{x}_i)$,
- Minimize Mean-square error: $J(\vec{w}) = \frac{1}{2} \sum (y_i - h_{\vec{w}}(\vec{x}_i))^2$

Perceptron/Neuron

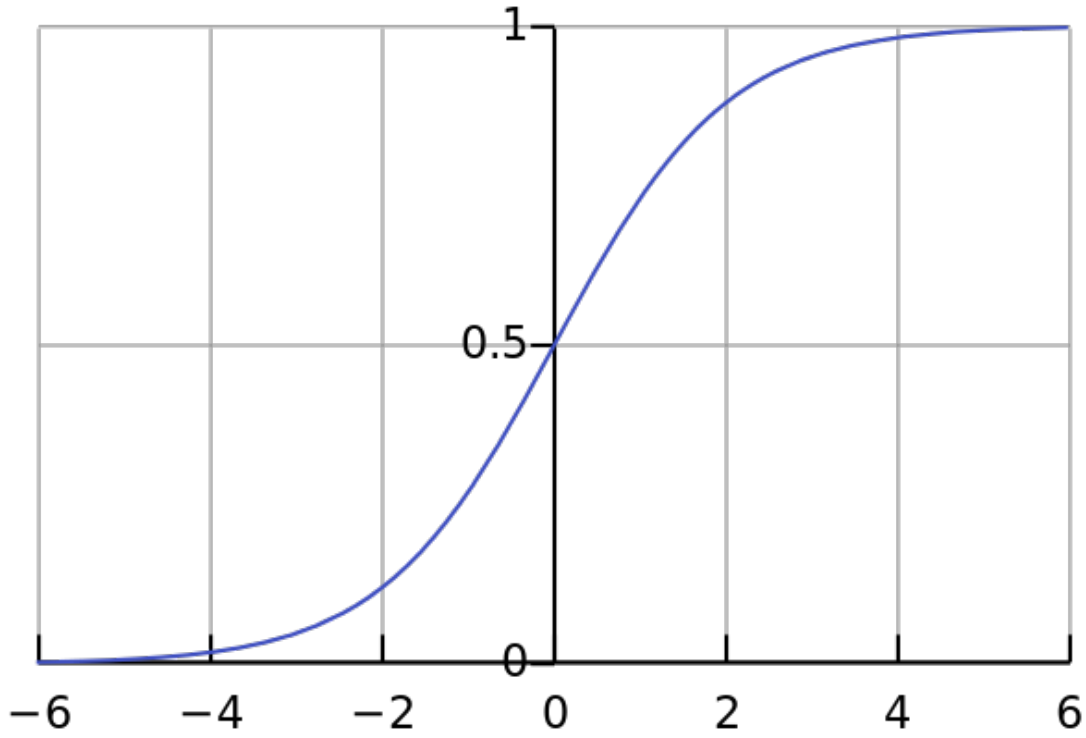


- We use sigmoid function $h_{\vec{w}} = \sigma(\vec{w}^T \vec{x})$ where

$$\sigma(x) = 1/(1 + e^{-x})$$

$$\sigma'(x) = \sigma(x)(1 - \sigma(x))$$

Sigmoid function



$$\sigma(x) = 1/(1 + e^{-x})$$

Perceptron/Neuron

- Minimize Mean-square error: $J(\vec{w}) = \frac{1}{2}\sum(y_i - h_{\vec{w}}(\vec{x}_i))^2$
- Stochastic Gradient Descent: $\vec{w} = \vec{w} - \eta \nabla J \vec{x}_i$
- Gradient of the error function:

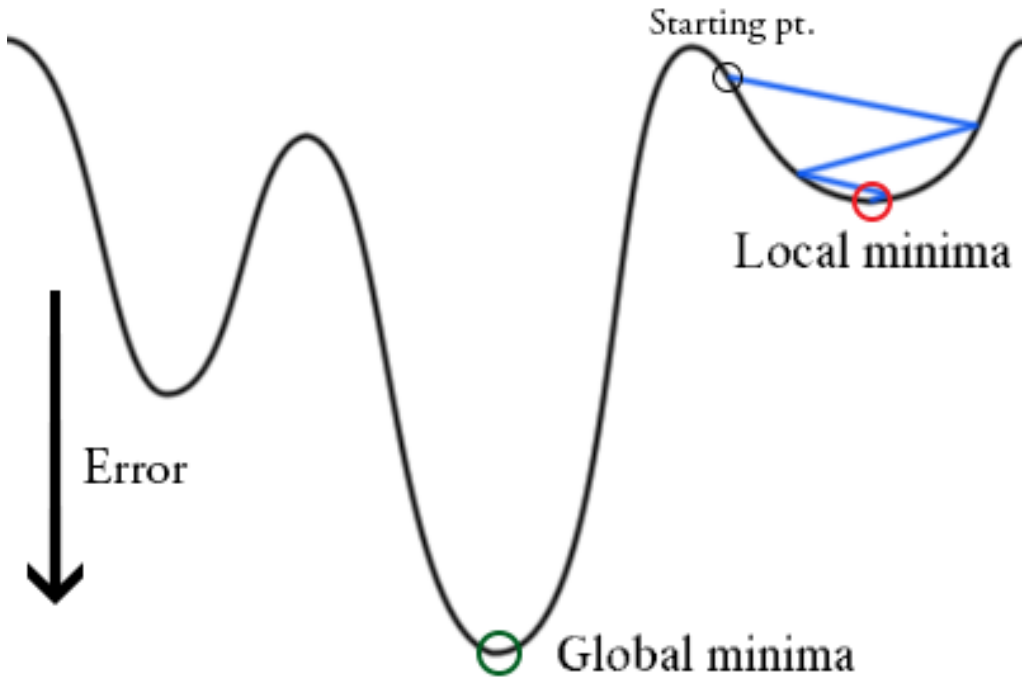
$$\nabla J = -\sum_i (y_i - h_{\vec{w}}(\vec{x}_i)) \nabla h_{\vec{w}}(\vec{x}_i)$$

$$\nabla h_{\vec{w}}(\vec{x}_i) = h_{\vec{w}}(\vec{x}_i)(1 - h_{\vec{w}}(\vec{x}_i))$$

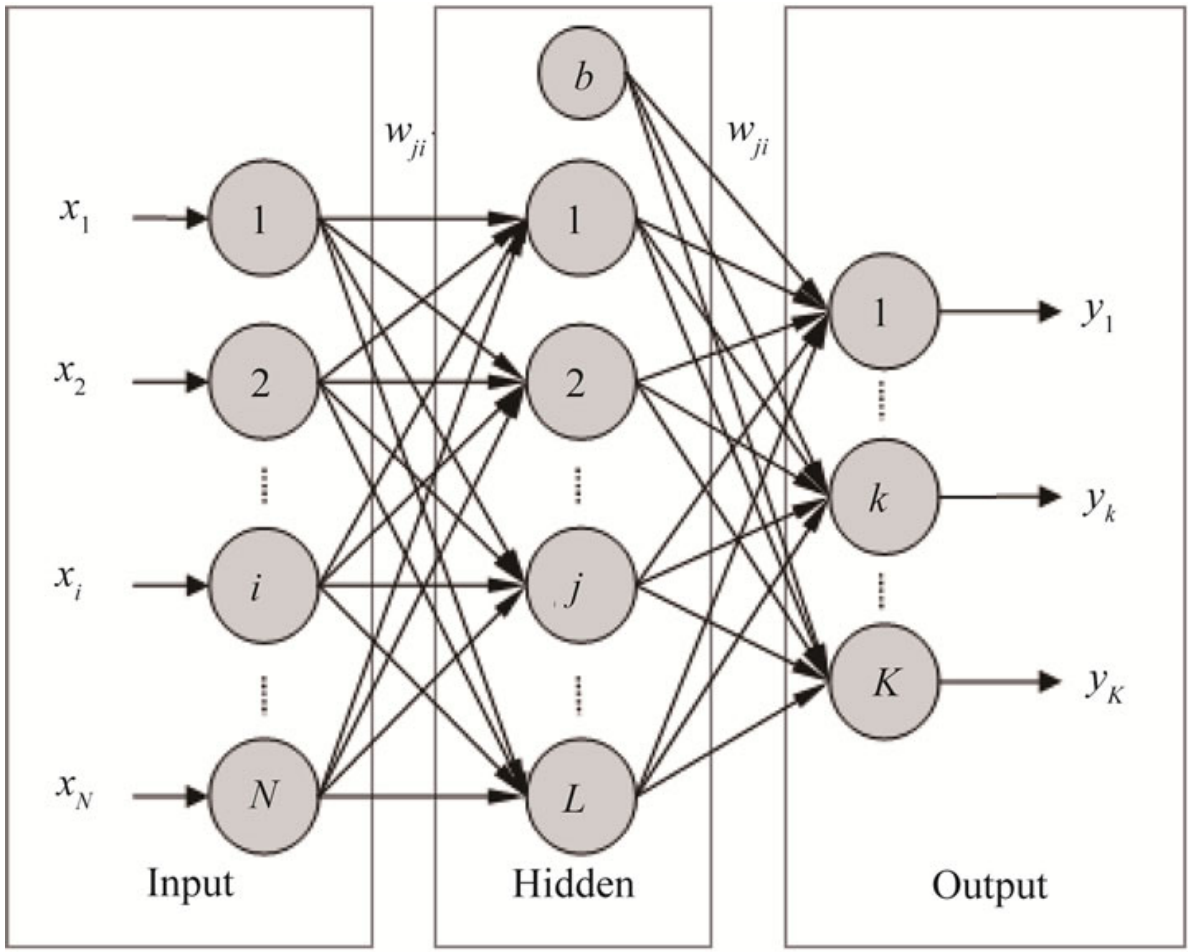
- Weight update rule

$$\vec{w} = \vec{w} + \eta \sum_i (y_i - h_{\vec{w}}(\vec{x}_i)) h_{\vec{w}}(\vec{x}_i)(1 - h_{\vec{w}}(\vec{x}_i))\vec{x}_i$$

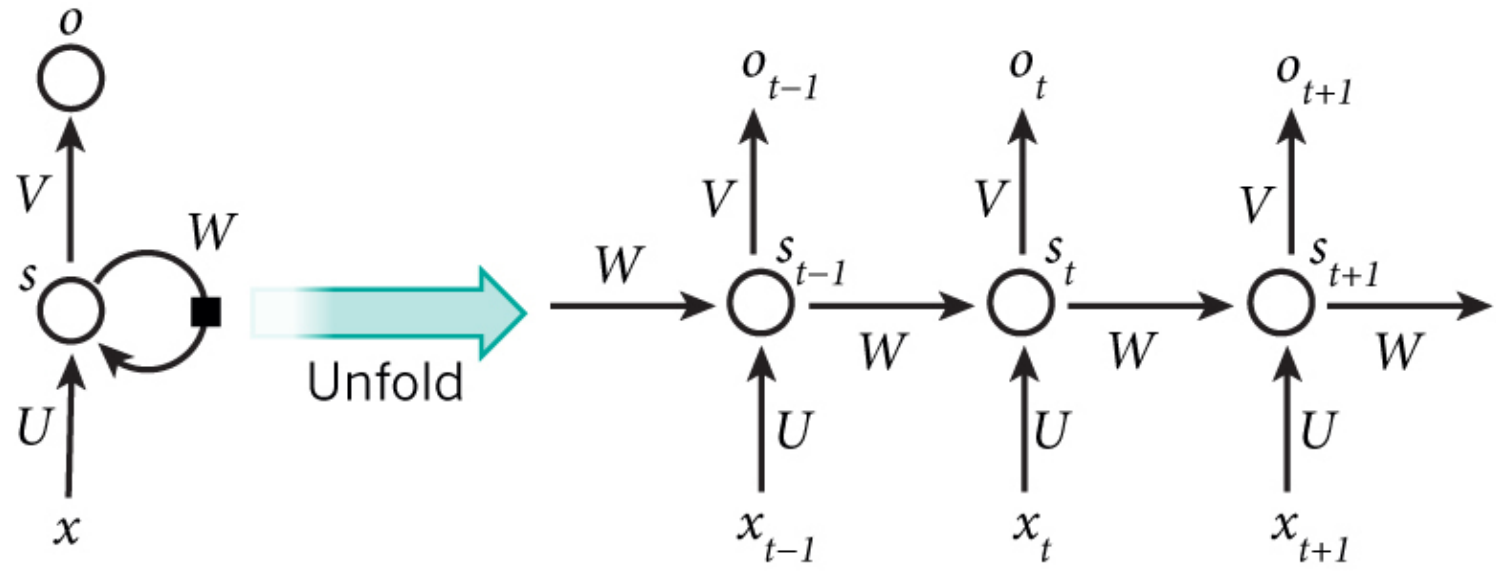
Stochastic Gradient Descent



Artificial Neural Network



Recurrent Neural Network



A recurrent neural network and the unfolding in time of the computation involved in its forward computation. Source: Nature

Classification

Data set that describes e-mail features for deciding if it is spam.

Example

Contains "Money"	**Domain type**	**Has attach.**	**Time received**	**spam**
yes	com	yes	night	yes
yes	edu	no	night	yes
no	com	yes	night	yes
no	edu	no	day	no
no	com	no	day	no
yes	cat	no	day	yes

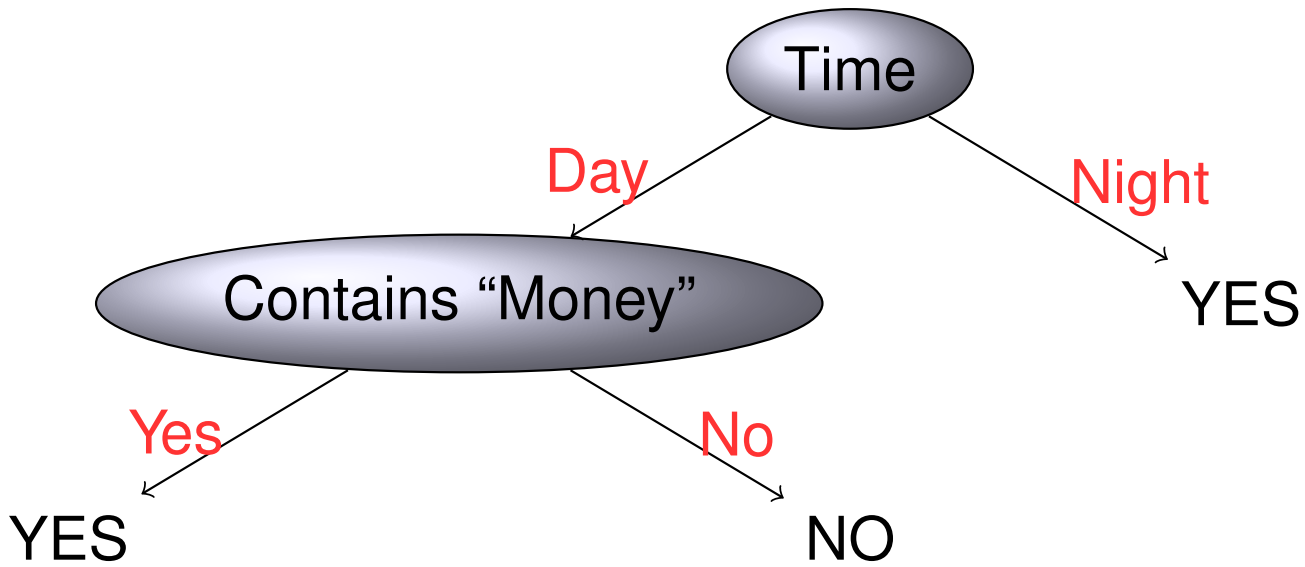
Assume we have to classify the following new instance:

Contains "Money"	**Domain type**	**Has attach.**	**Time received**	**spam**
yes	edu	yes	day	?

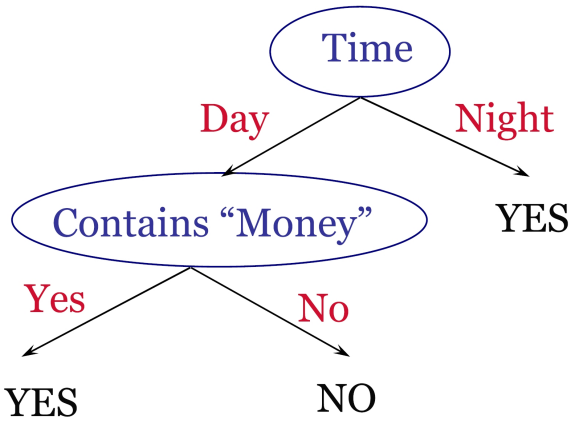
Classification

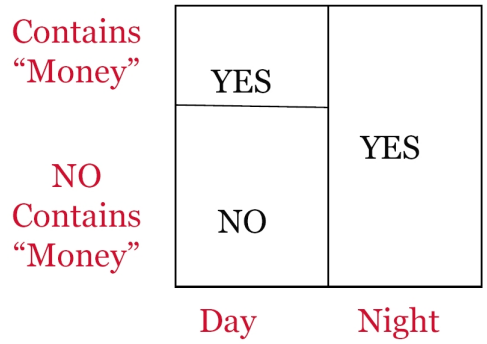
- Assume we have to classify the following new instance:

Contains "Money"	Domain type	Has attach.	Time received	spam
yes	edu	yes	day	?



Decision Trees

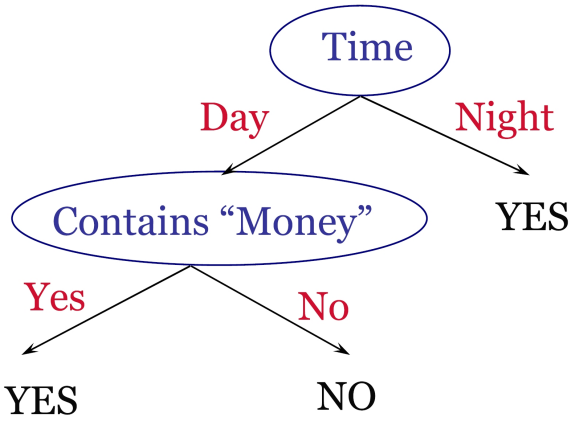


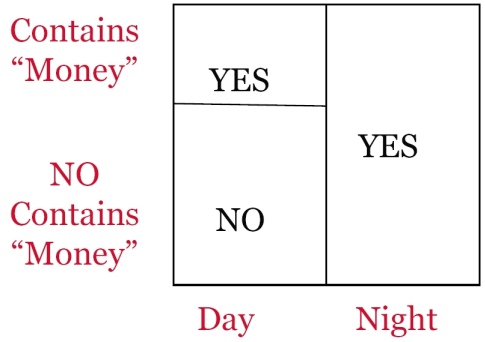


Basic induction strategy:

- $A \leftarrow$ the "best" decision attribute for next *node*
- Assign A as decision attribute for *node*
- For each value of A, create new descendant of *node*
- Sort training examples to leaf nodes
- If training examples perfectly classified, Then STOP, Else iterate over new leaf nodes

Decision Trees





Splitting Measures:

- Gini impurity

$$I = \sum_{i=1}^{J} p_i(1 - p_i)$$

- Information Gain

$$Entropy = -\sum_{i=1}^{J} p_i \log p_i$$

Bagging

Example

Dataset of 4 Instances : A, B, C, D

Classifier 1: B, A, C, B
Classifier 2: D, B, A, D
Classifier 3: B, A, C, B
Classifier 4: B, C, B, B
Classifier 5: D, C, A, C

Bagging builds a set of *M* base models, with a bootstrap sample created by drawing random samples with replacement.

Random Forests

- Bagging
- Random Trees: trees that in each node only uses a random subset of the attributes

Random Forests is one of the most popular methods in machine learning.

Boosting

The strength of Weak Learnability, Schapire 90

A boosting algorithm transforms a weak learner into a strong one

Boosting

A formal description of Boosting (Schapire)

- given a training set $(x_1, y_1), \ldots, (x_m, y_m)$
- $y_i \in \{-1, +1\}$ correct label of instance $x_i \in X$
- for $t = 1, \ldots, T$
  - construct distribution D_t
  - find weak classifier

$$h_t : X \to \{-1, +1\}$$

with small error $\varepsilon_t = \Pr_{D_t}[h_t(x_i) \neq y_i]$ on D_t

- output final classifier

Boosting

AdaBoost

1: Initialize $D_1(i) = 1/m$ for all $i \in \{1, 2, ..., m\}$
2: **for** t = 1,2,...T **do**
3: Call **WeakLearn**, providing it with distribution D_t
4: Get back hypothesis $h_t : X \to Y$
5: Calculate error of $h_t : \varepsilon_t = \sum_{i:h_t(x_i) \neq y_i} D_t(i)$
6: Update distribution

$$D_t : D_{t+1}(i) = \frac{D_t(i)}{Z_t} \times \begin{cases} \varepsilon_t/(1-\varepsilon_t) & \text{if } h_t(x_i) = y_i \\ 1 & \text{otherwise} \end{cases}$$

where Z_t is a normalization constant (chosen so D_{t+1} is a probability distribution)

7: **return** $h_{fin}(x) = \arg\max_{y \in Y} \sum_{t:h_t(x)=y} -\log \varepsilon_t/(1-\varepsilon_t)$

Boosting

AdaBoost

1: Initialize $D_1(i) = 1/m$ for all $i \in \{1, 2, ..., m\}$
2: **for** t = 1,2,...T **do**
3: Call **WeakLearn**, providing it with distribution D_t
4: Get back hypothesis $h_t : X \to Y$
5: Calculate error of $h_t : \varepsilon_t = \sum_{i:h_t(x_i) \neq y_i} D_t(i)$
6: Update distribution

$$D_t : D_{t+1}(i) = \frac{D_t(i)}{Z_t} \times \begin{cases} \varepsilon_t & \text{if } h_t(x_i) = y_i \\ 1 - \varepsilon_t & \text{otherwise} \end{cases}$$

where Z_t is a normalization constant (chosen so D_{t+1} is a probability distribution)

7: **return** $h_{fin}(x) = \arg\max_{y \in Y} \sum_{t:h_t(x)=y} -\log \varepsilon_t/(1-\varepsilon_t)$

Stacking

Use a classifier to combine predictions of base classifiers

Example

- Use a perceptron to do stacking
- Use decision trees as base classifiers