Learning from observations

February 1, 2011

Motivation

an agent is acting in an environment

- is making observations
  - How does the environment looks like?
  - Which characteristics the current state has?
- is choosing an action
  - What to do to act right?
- is getting some feedback
  - Was the chosen action the right one?
  - Was the chosen action useful?
  - How to measure the usefulness?
  - Who is giving a feedback?

Inductive Learning

Input

- Instance space $\mathcal{X} = \mathcal{X}_1 \times \ldots \times \mathcal{X}_k$ consisting of instances $\mathbf{x}_i = (x_{i1}, \ldots, x_{ik})$
- Labeling function $c : \mathcal{X} \to \mathcal{C}$
  - classification, if $\mathcal{C} = \mathbb{N}$
  - regression, if $\mathcal{C} = \mathbb{R}$
- Train and Test set
  - $D_k = \{\mathbf{x} \in \mathcal{X} | c(\mathbf{x}) \text{ is known}\}$
  - $D_u \subseteq \mathcal{X} \setminus D_k$

Note, that c is given explicitly!

Inductive Learning

Given the input the aim is to find an approximation $\hat{c}$ of c, such that the $error(\hat{c}, c, D_u)$ is minimal.

- D_u is a (test) set of instances for which labels are unknown
  - How can we measure an error when one of the parameters is unknown?
- $error(\hat{c}, c, D_u)$ is an error measure computed on the test set
  - How would You measure the error?

Error measures

Measured on a sample set D_s, is different for classification and regression problems

- Mean Average Error

$$MAE(\hat{c}, c, D_s) = \frac{1}{n} \sum_{\mathbf{x} \in D_s} |c(x) - \hat{c}(x)|$$

- Root Mean Squared Error

$$RMSE(\hat{c}, c, D_s) = \sqrt{\frac{\sum_{\mathbf{x} \in D_s} (c(x) - \hat{c}(x))^2}{|D_s|}}$$

- Which one is for regression/classification?
- Other different error measures can be found...

Methods, Techniques, ...

Dozens of different approaches and settings

- supervised vs. unsupervised
- model-based vs. memory-based
- rule-based vs. analytical models
- etc.

We will mention just a few of them. If interested, please consider to attend the Machine Learning lecture at the next winter term.

k-Nearest Neighbour

Memory-based/instance-based learning algorithm

- Also called as lazy learning because of no learning phase
- A test instance $\mathbf{x}^t$ is classified according to the majority voting of its k nearest neighbours

$$\hat{c}(\mathbf{x}^t) = arg_j max |\{\mathbf{x} \in N_{\mathbf{x}^t}^k : c(\mathbf{x}) = j\}|$$

- The target value of the test instance $\mathbf{x}^t$ is the average of its k nearest neighbours

$$\hat{c}(\mathbf{x}^t) = \frac{\sum_{\mathbf{x} \in N_{\mathbf{x}^t}^k} c(\mathbf{x})}{k}$$

k-Nearest Neighbour

What are the advantages, disadvantages of this algorithm?

Several variations, improvements

- weighting, optimization by nearest-neighbour search techniques, dimensionality reduction, etc.

There are two **parameters** needed for computation

- the number of neighbours k
- the distance measure
  - Euclidean, Manhattan, ...

How to estimate the *best* values for the parameters?

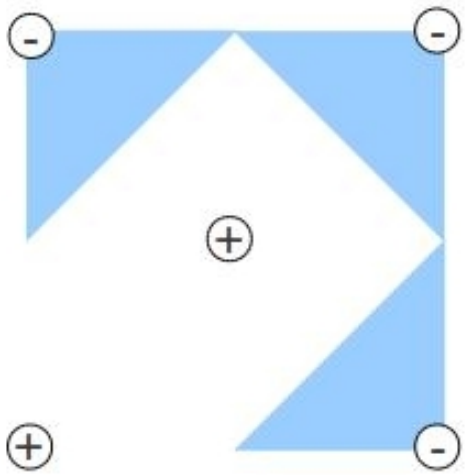
- underfitting vs. overfitting

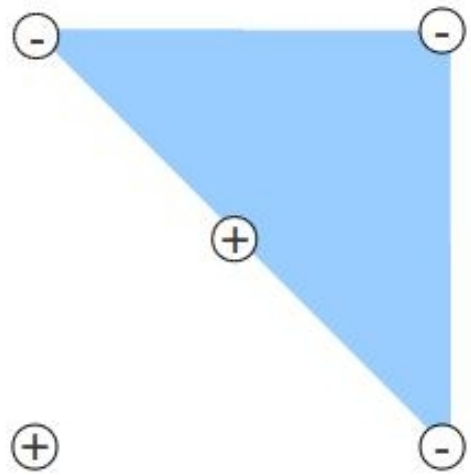
Cross-validation

k-fold cross-validation

- set the parameters to some values
- split the train data into k folds; for each of them do the following
  - the actual fold is used for validation while the other $k-1$ folds are used for training
- average the the errors measured on k folds
- choose the parameters resulting in the lowest error for final training of the model on the whole training set

Decision boundary





If the points can be separated using a linear hyperplane, than we have a linearly separable problem.

Perceptron

A binary classifier

$$c(\mathbf{x}) = sign(\mathbf{w} \cdot \mathbf{x} + w_0 > 0)$$

The aim is to estimate the parameters $\mathbf{w}$, i.e. to find $\hat{\mathbf{w}}$.
If treating the *bias* b as an extra attribute x_0 with values 1, then

$$\hat{c}(\mathbf{x}) = sign(\hat{\mathbf{w}} \cdot \mathbf{x})$$

with an error function

$$err(\hat{c}, c, D_s) = - \sum_{\mathbf{x} \in D_s; c(\mathbf{x}) \neq \hat{c}(\mathbf{x})} c(\mathbf{x})(\hat{\mathbf{w}} \cdot \mathbf{x})$$

Perceptron

For minimizing

$$err(\hat{c}, c, D_s) = - \sum_{\mathbf{x} \in D_s; c(\mathbf{x}) \neq \hat{c}(\mathbf{x})} c(\mathbf{x})(\hat{\mathbf{w}} \cdot \mathbf{x})$$

stochastic gradient descent with derivatives

$$\frac{\partial err}{\partial \hat{\mathbf{w}}} = - \sum_{\mathbf{x} \in D_s; c(\mathbf{x}) \neq \hat{c}(\mathbf{x})} c(\mathbf{x})\mathbf{x}$$

can be used, thus an update at the iteration $i + 1$ for the example $(\mathbf{x}, c(\mathbf{x}))$ will be

$$\hat{\mathbf{w}}^{(i+1)} = \hat{\mathbf{w}}^{(i)} + \alpha(c(\mathbf{x}) - \hat{c}(\mathbf{x}))\mathbf{x}$$

Perceptron

Algorithm

- initialize the weights $\hat{\mathbf{w}}$
- until convergence do
  - for each training example $(\mathbf{x}, c(\mathbf{x}))$ do
    - compute $\hat{c}(\mathbf{x})$
    - if $\hat{c}(\mathbf{x}) \neq c(\mathbf{x})$ update weights as

$$\hat{\mathbf{w}}^{(new)} = \hat{\mathbf{w}}^{(old)} + 2\alpha c(\mathbf{x})\mathbf{x}$$

If the instances are linearly separable, then the perceptron converges.

Linear Regression

Suppose, data are generated as $Y = \beta X + \epsilon$

- labels of instances are a linear combination of their attributes

$$\begin{pmatrix} y_1 \\ \vdots \\ y_n \end{pmatrix} = \begin{pmatrix} \beta_0 \\ \beta_1 \\ \vdots \\ \beta_n \end{pmatrix} \begin{pmatrix} 1 & x_{11} & \dots & x_{1k} \\ \vdots & \vdots & \vdots & \vdots \\ 1 & x_{n1} & \dots & x_{nk} \end{pmatrix} + \begin{pmatrix} \epsilon_1 \\ \vdots \\ \epsilon_n \end{pmatrix}$$

The aim is to fit the instances instead of separating them.

- thus, we should minimize *RMSE*, what means to find a minimum of $||Y - \hat{Y}||^2 = ||Y - \hat{\beta}X||^2$
  - i.e. to solve a system of equations $X^T X \hat{\beta} = X^T Y$

For more information, please refer to the Machine Learning lecture.

Naive Bayes

The probability that an instance $\mathbf{x} = (x_1, \ldots, x_k)$ belongs to a class c is, using the Bayes theorem, the following

$$p(c|x_1, \ldots, x_k) = \frac{p(c)p(x_1, \ldots, x_k|c)}{p(x_1, \ldots, x_k)}$$

Expecting that the Naive Bayes assumption holds

- $p(x_1, \ldots, x_k|c) = p(x_1|c)p(x_2, \ldots, x_k|c, x_1) = \ldots = p(x_1|c)p(x_2|c, x_1)p(x_3|c, x_1, x_2) \ldots p(x_k|c, x_1, \ldots, x_{k-1}) = p(x_1|c) \ldots p(x_k|c)$

and leaving out the denominator since this is constant, we get

$$p(c|x_1, \ldots, x_k) = p(c) \prod_{j=1}^{k} p(x_j|c)$$

Naive Bayes

- fast computation even for large amount of features

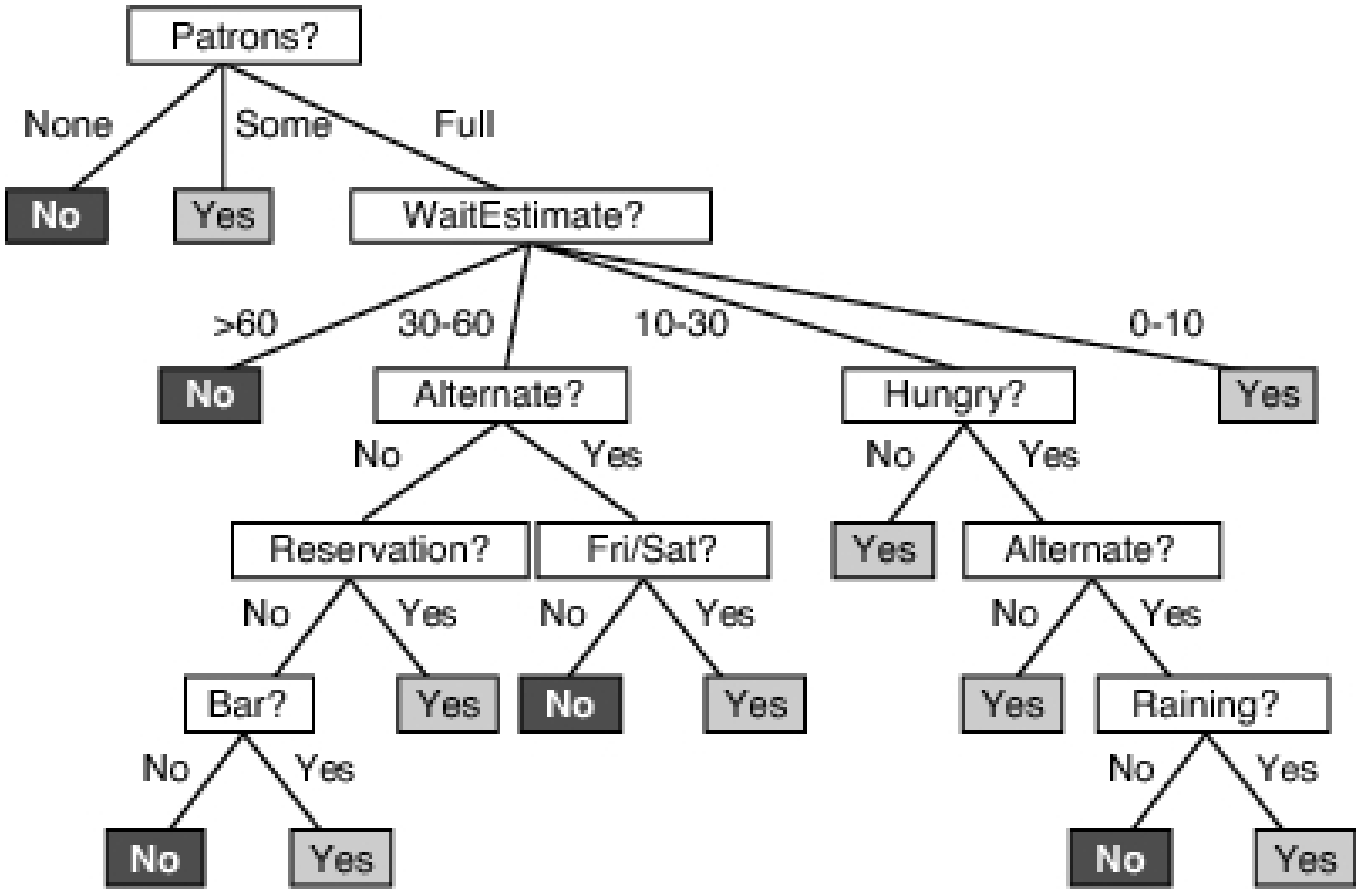
$$p(x_j|c) = \frac{|\{\mathbf{x}' : x'_j = x_j \wedge c(\mathbf{x}') = c\}|}{|\{\mathbf{x}' : c(\mathbf{x}') = c\}|}$$

- works well despite its “unrealistic“ independence assumptions
- easy to implement
- problem o zero probabilities
  - if there is no object in the training data with a particular feature and a particular label
  - smoothing, e.g.

$$p(x_j|c) = \frac{|\{\mathbf{x}' : x'_j = x_j \wedge c(\mathbf{x}') = c\}| + 1}{|\{\mathbf{x}' : c(\mathbf{x}') = c\}| + n}$$

Decision Tree

Usually “used” in decision making



Decision Tree

Example: training data

Example	Attributes										Target
	Alt	*Bar*	*Fri*	*Hun*	*Pat*	*Price*	*Rain*	*Res*	*Type*	*Est*	*WillWait*
X_1	*T*	*F*	*F*	*T*	*Some*	*$$$*	*F*	*T*	*French*	*0–10*	*T*
X_2	*T*	*F*	*F*	*T*	*Full*	*$*	*F*	*F*	*Thai*	*30–60*	*F*
X_3	*F*	*T*	*F*	*F*	*Some*	*$*	*F*	*F*	*Burger*	*0–10*	*T*
X_4	*T*	*F*	*T*	*T*	*Full*	*$*	*F*	*F*	*Thai*	*10–30*	*T*
X_5	*T*	*F*	*T*	*F*	*Full*	*$$$*	*F*	*T*	*French*	*>60*	*F*
X_6	*F*	*T*	*F*	*T*	*Some*	*$$*	*T*	*T*	*Italian*	*0–10*	*T*
X_7	*F*	*T*	*F*	*F*	*None*	*$*	*T*	*F*	*Burger*	*0–10*	*F*
X_8	*F*	*F*	*F*	*T*	*Some*	*$$*	*T*	*T*	*Thai*	*0–10*	*T*
X_9	*F*	*T*	*T*	*F*	*Full*	*$*	*T*	*F*	*Burger*	*>60*	*F*
X_{10}	*T*	*T*	*T*	*T*	*Full*	*$$$*	*F*	*T*	*Italian*	*10–30*	*F*
X_{11}	*F*	*F*	*F*	*F*	*None*	*$*	*F*	*F*	*Thai*	*0–10*	*F*
X_{12}	*T*	*T*	*T*	*T*	*Full*	*$*	*F*	*F*	*Burger*	*30–60*	*T*

Decision Tree

Algorithm

function DTL(*examples*, *attributes*, *default*) **returns** a decision tree

- **if** *examples* is empty **then return** *default*
- **else if** all *examples* have the same classification **then return** the classification
- **else if** *attributes* is empty **then return** MODE(*examples*)
- **else**
  - $best \leftarrow$ CHOOSE-ATTRIBUTE(*attributes*, *examples*)
  - $tree \leftarrow$ a new decision tree with root test *best*
  - **for each** value v_i of *best* **do**
    - $examples_i \leftarrow$ {elements of *examples* with $best = v_i$}
    - $subtree \leftarrow$ DTL($examples_i$, $attributes - best$, MODE(*examples*))
    - add a branch to *tree* with label v_i and subtree *subtree*
  - **return** *tree*

Decision Tree

How to choose an attribute?

- Entropy

$$I(\frac{p}{p+n}, \frac{n}{p+n}) = -\frac{p}{p+n} log_2 \frac{p}{p+n} - \frac{n}{p+n} log_2 \frac{n}{p+n}$$

- After choosing an attribute A

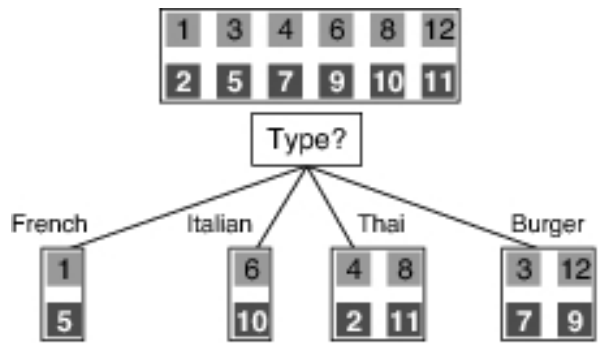
$$Remainder(A) = \sum_{i=1}^{v} \frac{p_i + n_i}{p+n} I(\frac{p_i}{p_i+n_i}, \frac{n_i}{p_i+n_i})$$

- Information gain

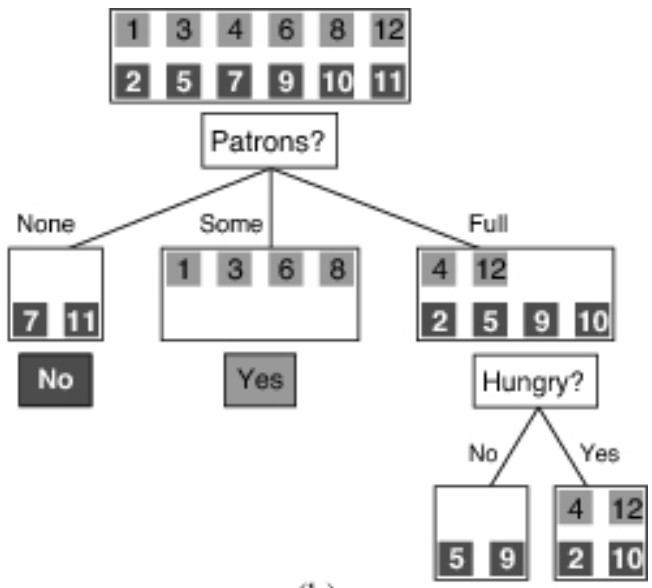
$$Gain(A) = I(\frac{p}{p+n}, \frac{n}{p+n}) - Remainder(A)$$

Decision Tree

How to choose an attribute?



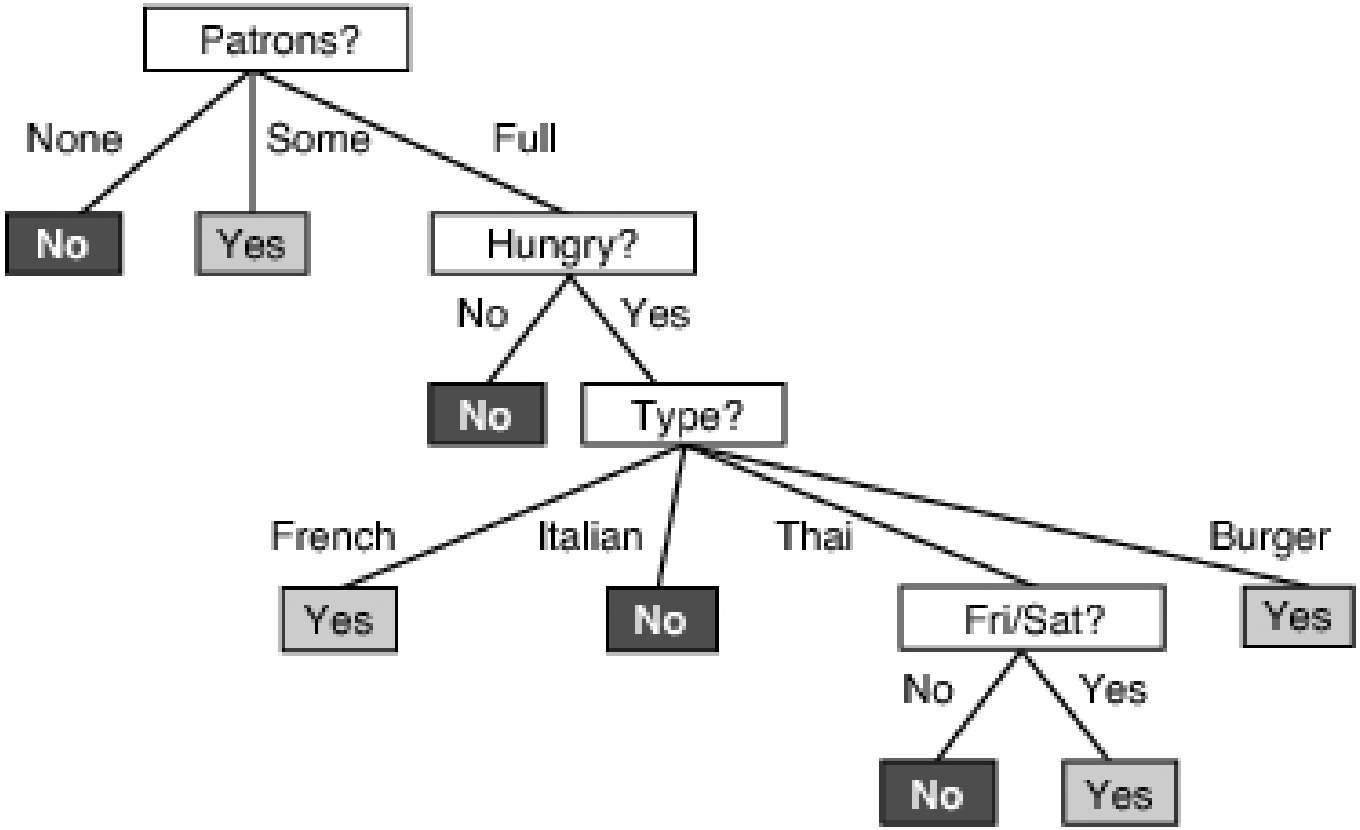
(a)



(b)

Decision Tree

An induced tree from the example



Prunning

- against overfitting