

Data Mining (Mineria de Dades)

Lluís A. Belanche
belanche@lsi.upc.edu

Soft Computing Research Group

Dept. de Llenguatges i Sistemes Informàtics (Software department)

Universitat Politècnica de Catalunya

2011-2012

Data Mining (Part II)

Contents

1. Association Rules
2. Bayesian decision theory
3. Bayesian classifiers: LDA/QDA
4. Bayesian classifiers: Naïve Bayes
5. Bayesian classifiers: Nearest Neighbours
6. Neural Networks: Multilayer Perceptrons
7. Support Vector Machines
8. Ensemble Learning

Artificial neural networks (II)

How could we obtain a model that is non-linear in the parameters
 $\equiv$ a non-linear model? We depart from the basic model:

$$y_k(\mathbf{x}) = g \left(\sum_{i=1}^n w_{ki} x_i + w_{k0} \right), k = 1, \dots, m$$

where g is a sigmoidal function. This model is linear in the parameters
 $\equiv$ a linear model.

The solution is to apply non-linear functions to the input data:

$$y_k(\mathbf{x}) = g \left(\sum_{i=0}^h w_{ki} \phi_i(\mathbf{x}) \right), k = 1, \dots, m$$

We recover the previous “linear” situation making $h = n$ and $\phi_i(\mathbf{x}) = x_i$,
with $\phi_0(\mathbf{x}) = 1$.

Artificial neural networks (II)

Approach 1. Make $\Phi = (\phi_0, \dots, \phi_h)$ a set of predefined functions. This is perfectly illustrated in the case $n = 1$ and polynomial fitting. Consider the problem of fitting the function:

$$p(x) = w_0 + w_1x + \dots + w_hx^h = \sum_{i=0}^h w_i x^i \text{ to } x_1, \dots, x_p$$

This can be seen as a special case of linear regression, where the set of *regressors* is $1, x, x^2, \dots, x^h$. Therefore $\phi_i(\mathbf{x}) = x^i$

The weights $w_0, w_1, \dots, w_h$ can be estimated by standard techniques (ordinary least squares) or by the Delta rule.

Artificial neural networks (II)

What if we have a multivariate input $x = (x_1, \dots, x_n)^t$? The corresponding polynomial is:

$$p(x) = w_0 + \sum_{i_1=1}^n w_{i_1} x_{i_1} + \sum_{i_1=1}^n \sum_{i_2=i_1+1}^n w_{i_1 i_2} x_{i_1} x_{i_2} + \sum_{i_1=1}^n \sum_{i_2=i_1+1}^n \sum_{i_3=i_2+1}^n w_{i_1 i_2 i_3} x_{i_1} x_{i_2} x_{i_3} \dots$$

The number of possible regressors grows as $\binom{n+h}{h}!$

So many regressors (while holding p fixed) causes increasing troubles for estimating their parameters:

- It is quite convenient (sometimes mandatory!) to have more observations p than regressors
- Statistical significance decreases with the number of regressors and increases with p

Artificial neural networks (II)

Approach 2. Why not trying to engineer adaptive regressors? By adapting the regressors to the problem, it is reasonable to expect that we shall need a much smaller number of them for a correct fit.

The basic neural network idea is to *duplicate the model*:

$$y_k(\mathbf{x}) = g \left(\sum_{i=0}^h w_{ki} \phi_i(\mathbf{x}) \right), k = 1, \dots, m$$

where

$$\phi_i(\mathbf{x}) = g \left(\sum_{j=0}^n v_{ij} x_j \right), \text{ with } \phi_0(\mathbf{x}) = 1, x_0 = 1$$

Artificial neural networks (II)

- The effect is that we have now a new set of regressors $\Phi(\mathbf{x}) = (\phi_0(\mathbf{x}), \dots, \phi_h(\mathbf{x}))^t$.
- These regressors are adaptive via the $\mathbf{v}_i$ parameters (called the non-linear parameters)
- Once the regressors are fully specified (i.e, the $\mathbf{v}_i$ parameters are estimated), the remaining task is again a linear fit (via the $\mathbf{w}_k$ parameters).
- What kind of network gives rise to this function? The Multilayer Perceptron or MLP
- Under other choices for the regressors, other networks are obtained:

$$\phi_i(\mathbf{x}) = \exp\left(-\frac{\|\mathbf{x} - \boldsymbol{\mu}_i\|^2}{\sigma_i^2}\right), \text{ is the basic RBF network.}$$

Artificial neural networks (II)

Error functions

Where does the (mean) square error come from? On what grounds?

Suppose we have a set of learning examples

$S = \{(\mathbf{x}_\mu, \mathbf{t}_\mu)\}_{\mu=1,\dots,p}$, where $\mathbf{x}_\mu \in \mathbb{R}^n$, $\mathbf{t}_\mu \in \mathbb{R}^m$ or $\mathbf{t}_\mu \in \{0, 1\}^m$ (assume S is i.i.d.).

- Ideally, we would like to know the statistical properties, namely $p(\mathbf{t}|\mathbf{x})$
- An error function can be derived by *maximizing the likelihood* of S :

$$\mathcal{L} = \prod_{\mu=1}^p p(\mathbf{t}_\mu|\mathbf{x}_\mu)$$

- The different outputs are assumed to be independent, so we can write:

$$p(\mathbf{t}|\mathbf{x}) = \prod_{k=1}^m p(t_k|\mathbf{x})$$

- When convenient, we can also maximize a strictly monotonic function of $\mathcal{L}$, namely the log-likelihood $l = \ln \mathcal{L}$

Artificial neural networks (II)

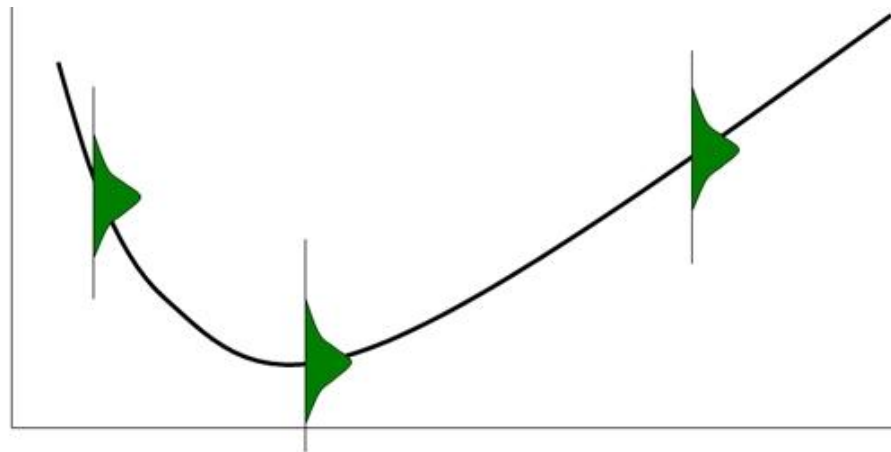
Error functions for regression

For simplicity, take $t_\mu \in \mathbb{R}$. We model functional *ignorance* as stochastic variability: we put a continuous pdf $p(t|X = x)$ around each point x . The optimal solution (the *regression function*) is:

$$y^*(x) = E[t|x] = \int_{\mathbb{R}} t p(t|x) dt$$

In order to model this (recall p is unknown), take each t_k a deterministic function, distorted by a gaussian “noise”:

$t_k = h_k(\mathbf{x}) + \epsilon$, with $\epsilon \sim N(0, \sigma^2)$, $k = 1, \dots, m$. Note ϵ does not depend on k or $\mathbf{x}$ (homocedasticity).



Artificial neural networks (II)

Error functions for regression

Therefore we have:

$$p(\epsilon) = \frac{1}{\sqrt{2\pi}\sigma} \exp \left\{ -\frac{\epsilon^2}{2\sigma^2} \right\}$$

Also note that in this case the optimal function would be:

$$y_k^*(x) = E[t_k|x] = E[h_k(\mathbf{x}) + \epsilon] = E[h_k(\mathbf{x})] + E[\epsilon] = h_k(\mathbf{x})$$

Therefore we **model** $\epsilon = t_k - h_k(\mathbf{x}) \equiv t_k - y_k(\mathbf{x})$ to obtain:

$$p(t_k|\mathbf{x}) = \frac{1}{\sqrt{2\pi}\sigma} \exp \left\{ -\frac{(t_k - y_k(\mathbf{x}))^2}{2\sigma^2} \right\}$$

Artificial neural networks (II)

Error functions for regression

So which error should we use? Let us try to define and minimize the negative log-likelihood as the *error*:

$$\begin{aligned} -l &= -\ln \mathcal{L} = -\ln \prod_{\mu=1}^p p(\mathbf{t}_{\mu} | \mathbf{x}_{\mu}) = -\ln \prod_{\mu=1}^p \prod_{k=1}^m p(t_{\mu k} | \mathbf{x}_{\mu}) = -\sum_{\mu=1}^p \sum_{k=1}^m \ln p(t_{\mu k} | \mathbf{x}_{\mu}) \\ &= -\sum_{\mu=1}^p \sum_{k=1}^m \left\{ \ln \left(\frac{1}{\sqrt{2\pi}\sigma} \right) - \frac{(t_{\mu k} - y_k(\mathbf{x}_{\mu}))^2}{2\sigma^2} \right\} = \sum_{\mu=1}^p \sum_{k=1}^m \left\{ \frac{1}{2} \ln(2\pi\sigma^2) + \frac{(t_{\mu k} - y_k(\mathbf{x}_{\mu}))^2}{2\sigma^2} \right\} \\ &= \frac{mp}{2} (\ln(2\pi) + 2\ln\sigma) + \frac{1}{2\sigma^2} \sum_{\mu=1}^p \sum_{k=1}^m (t_{\mu k} - y_k(\mathbf{x}_{\mu}))^2 \end{aligned}$$

The first term is out of our control –it does not depend on the model $y_k(\cdot)$. So we should minimize:

$$E \equiv \frac{1}{2} \sum_{\mu=1}^p \sum_{k=1}^m (t_{\mu k} - y_k(\mathbf{x}_{\mu}))^2 = \frac{1}{2} \sum_{\mu=1}^p \|\mathbf{t}_{\mu} - \mathbf{y}(\mathbf{x}_{\mu})\|^2$$

Artificial neural networks (II)

Error functions for classification

The goal in classification is to model the posterior probabilities for every class $P(\omega_k|\mathbf{x})$. In two-class problems, we model by creating an ANN with one output neuron ($m = 1$) to represent $y(\mathbf{x}) = P(\omega_1|\mathbf{x})$; therefore $1 - y(\mathbf{x}) = P(\omega_2|\mathbf{x})$.

Suppose we have a set of learning examples

$S = \{(\mathbf{x}_\mu, t_\mu)\}_{\mu=1,\dots,p}$, where $\mathbf{x}_\mu \in \mathbb{R}^n, t_\mu \in \{0, 1\}$ (assume S is i.i.d.).

We take the convention that $t_\mu = 1$ means $\mathbf{x}_\mu \in \omega_1$ and $t_\mu = 0$ means $\mathbf{x}_\mu \in \omega_2$, to **model**:

$$P(t|\mathbf{x}) = \begin{cases} y(\mathbf{x}) & \text{if } \mathbf{x}_\mu \in \omega_1 \\ 1 - y(\mathbf{x}) & \text{if } \mathbf{x}_\mu \in \omega_2 \end{cases}$$

which can be more conveniently expressed as $P(t|\mathbf{x}) = y(\mathbf{x})^t(1 - y(\mathbf{x}))^{1-t}$, $t = 0, 1$.

Artificial neural networks (II)

Error functions for classification

This is a Bernoulli distribution. Assuming again an i.i.d sample, the likelihood is:

$$\mathcal{L} = \prod_{\mu=1}^p y(\mathbf{x}_{\mu})^{t_{\mu}} (1 - y(\mathbf{x}_{\mu}))^{1-t_{\mu}}$$

So which error should we use? Let us define and minimize again the negative log-likelihood as the *error*:

$$E \equiv -\ln \mathcal{L} = -\sum_{\mu=1}^p \{t_{\mu} \ln y(\mathbf{x}_{\mu}) + (1 - t_{\mu}) \ln(1 - y(\mathbf{x}_{\mu}))\}$$

which is known as the “cross-entropy” and can be generalized to more than two classes.