Lecture 6. Regression

Prof. Alan Yuille

Summer 2014

Outline

1. Introduction to Regression
2. Binary Regression
3. Linear Regression; Polynomial Regression
4. Non-linear Regression; Multilayer Perceptron

1 Introduction to Regression

Consider learning directly the conditional distribution $p(y|x)$.

This is often easier than learning the likelihood function $p(x|y)$ and the prior $p(y)$ separately. This is because the space of y is typically much lower-dimensional than the space of x. For example, if x is a 10×10 image (e.g., of a face, or a non-face) then this space has an enormous number of dimensions while, by contrast, the output y takes only binary values. It is much easier to learn distributions on lower-dimensional spaces.

The task of estimating y from x is called *regression.* It has a long history. Two hundred years ago it was invented by Gauss to estimate the position of the planetoid Ceres (Gauss's father encouraged him to do work on this problem saying that there was more money in Astronomy than in Mathematics).

In this lecture we address three different types of regression problem.

(I) *Binary regression.* Here $y \in \{\pm 1\}$. We can specify a distribution to be of exponential form (non-parametric ways of doing regression are possible, but we do not have time to discuss them):

$$p(y|x;\lambda) = \frac{e^{y\lambda\cdot\phi(x)}}{e^{\lambda\cdot\phi(x)} + e^{-\lambda\cdot\phi(x)}}.$$

Note that this is of form $p(y|x;\lambda) = \frac{e^{y\lambda\cdot\phi(x)}}{Z[\lambda,x]}$, and because y is binary valued we can compute the normalization term $Z[\lambda,x] = \sum_y e^{y\lambda\cdot\phi(x)} = e^{\lambda\cdot\phi(x)} + e^{-\lambda\cdot\phi(x)}$.

(II) *Linear Regression.* Here y takes a continuous set of values (we can extend this directly to allow y to be vector-valued).

$$p(y|x,\lambda) = \frac{1}{\sqrt{2\pi}\sigma} e^{-(1/2\sigma^2)(y-\lambda\cdot\phi(x))^2},$$

where λ includes the variance σ^2.

This model assumes that the data can be expressed as $y = \lambda\cdot\phi(x) + \epsilon$, where $\lambda\cdot\phi(x)$ is a linear predictor (i.e. it depends on linear coefficients λ) and where ϵ is a random variable (i.e. noise) Gaussianly distributed with zero mean and variance σ^2. To see this, write $p(y|x,\epsilon) = \delta(y-\lambda\cdot\phi(x)+\epsilon)$, $p(\epsilon) = \frac{1}{\sqrt{2\pi}\sigma}\exp -\frac{\epsilon^2}{2\sigma^2}$ and compute $p(y|x) = \int d\epsilon p(y|x,\epsilon)p(\epsilon)$.

(III) *Non-Linear regression.* This is an extension of binary regression.

$$p(y|x) = \frac{1}{Z} e^{M(y,g(x:\lambda))},$$

where $M(\cdot,\cdot)$ is a similarity measure and $g(x:\lambda)$ is non linear in λ. This leads to non-linear optimization problems. It is more powerful but more computationally intensive. Important cases are multi-layer perceptrons and deep networks.

In both cases (I) and (II), the parameters λ can be estimated by Maximum Likelihood (ML) and, as in previous lectures, this corresponding to minimizing a convex energy function and, in some cases (e.g., case II), there will be an analytic expression for the solution. (Sometimes it is good to add a prior $P(\lambda)$ and do MAP estimation, and sometimes a loss function can be added also). In the case (III) the maximum likelihood cannot be solved analytically. An algorithm will be necessary to solve a non-convex optimization problem.

2 Binary Regression

$$p(y|\vec{x};\vec{\lambda}) = \frac{e^{y\vec{\lambda}\cdot\vec{\phi}(\vec{x})}}{e^{\vec{\lambda}\cdot\vec{\phi}(\vec{x})} + e^{-\vec{\lambda}\cdot\vec{\phi}(\vec{x})}}.$$

Note that this corresponds to a decision rule $\hat{y} = sign(\vec{\lambda}\cdot\vec{\phi}(\vec{x}))$. Or, equivalently, $\hat{y}(\vec{x}) = \arg\max_y y\vec{\lambda}\cdot\vec{\phi}(\vec{x})$. We obtain this here by taking the log-likelihood ratio $\log\frac{p(y=1|\vec{x};\vec{\lambda})}{p(y=-1|\vec{x};\vec{\lambda})}$.

To perform ML on this model we need to minimize:

$$F(\vec{\lambda}) = -\sum_{i=1}^{N}\log p(y_i|\vec{x}_i;\vec{\lambda}) = -\sum_{i=1}^{N} y_i\lambda\cdot\vec{\phi}(\vec{x}_i) + \sum_{i=1}^{N}\log\{e^{\vec{\lambda}\cdot\vec{\phi}(\vec{x})} + e^{-\vec{\lambda}\cdot\vec{\phi}(\vec{x})}\},$$

where $\mathcal{X} = \{(\vec{x}_i, y_i) : i = 1, ..., N\}$ is the training dataset.

It can be checked that $F(\vec{\lambda})$ is a convex function of $\vec{\lambda}$ (compute the Hessian, then use Cauchy-Schwartz to show it is positive semi-definite.).

The gradient of $F(\vec{\lambda})$ with respect to $\vec{\lambda}$ can be computed to be:

$$\frac{\partial F}{\partial\vec{\lambda}} = -\sum_{i=1}^{N} y_i\vec{\phi}(\vec{x}_i) + \sum_{i=1}^{N}\sum_{y\in\{\pm1\}} y\vec{\phi}(\vec{x}_i)p(y|\vec{x}_i,\vec{\lambda}). \tag{1}$$

Hence the ML estimate – at $\hat{\vec{\lambda}}$, such that $\frac{\partial F}{\partial\vec{\lambda}}(\hat{\vec{\lambda}}) = 0$ – balances the statistics of the data (first term in equation (1) –with the model statistics (second term in equation (1) where the expected over x is based on the data (i.e. regression only learns a probability model for y and not for x).

Usually we cannot solve equation (1) analytically to solve for $\hat{\lambda}$. Instead, we can solve for λ by doing steepest descent (is there an analogy to GIS? check! yes, easy to derive one). I.e.

$$\vec{\lambda}^{t+1} = \vec{\lambda}^t - \Delta\{-\log p(y|\vec{x},\vec{\lambda})\}$$

$$= \vec{\lambda}^t - \Delta\{-\sum_{i=1}^{N} y_i\vec{\phi}(\vec{x}_i) + \sum_{i=1}^{N}\sum_{y\in\{\pm1\}} y\vec{\phi}(x_i)p(y|\vec{x}_i,\vec{\lambda})\}.$$

2.1 Special Case of Binary Regression: the Artificial Neuron

An artificial model of a neuron is obtained by setting $\vec{\phi}(\vec{x}) = (x_1, ..., x_M)$, where the x_i are scalar values. This is illustrated in figure (1). In this case $\vec{\lambda} \cdot \vec{\phi}(\vec{x}) = \sum_{i=1}^{M} \lambda_i x_i$. The x_i are thought of as the input to the neuron and their strength in weighted by the synaptic strength λ_i. The weighted inputs are summed and at the cell body, the soma, the artificial neuron fires with probability $p(y = 1|\vec{x})$, given by:

$$p(y|\vec{x}) = \frac{e^{y\vec{\lambda}\cdot\vec{\phi}(\vec{x})}}{e^{\vec{\lambda}\cdot\vec{\phi}(\vec{x})} + e^{-\vec{\lambda}\cdot\vec{\phi}(\vec{x})}}$$

This is called integrate-and-fire. In practice, we can add another term λ_0 to the summation which acts as a threshold for firing (i.e. $\vec{\phi}(\vec{x}) = (1, x_1, ..., x_M)$ and $\vec{\lambda} = (\lambda_0, \lambda_1, ..., \lambda_M)$). In this case, $\vec{\lambda} \cdot \vec{\phi}(\vec{x}) > 0$, i.e., $\sum_{i=1}^{M} \lambda_i x_i > -\lambda_0$

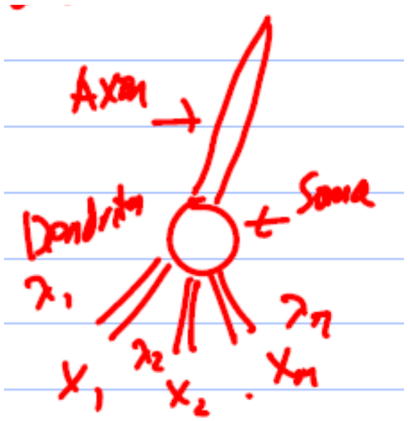

Figure 1: An artificial model of a neuron. The inputs are $x_1, ..., x_M$ at the dendrites, the synaptic strengths are $\lambda_1, ..., \lambda_M$, the cell body (soma) calculates the weighted sum of the inputs $\sum_{i=1}^{M} \lambda_i x_i$ and fires a spike down the axon with probability $p(y = 1|x)$. This provides input to another neuron..

3 Gaussian Linear Regression

Now, consider continuous linear regression with a Gaussian model. Here y is a scalar output (a continuous number).

$$y = \vec{\lambda} \cdot \vec{\phi}(\vec{x}) + \epsilon,$$

where ϵ is a noise term which we can model with Gaussian: $p(\epsilon) = \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{\epsilon^2}{2\sigma^2}}$ The probability distribution of y is

$$p(y|\vec{x}, \vec{\lambda}) = \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{(y - \vec{\lambda} \cdot \vec{\phi}(\vec{x}))^2}{2\sigma^2}}.$$

Maximum Likelihood (ML) estimation minimizes:

$$F(\vec{\lambda}, \sigma) = \frac{1}{2\sigma^2} \sum_{i=1}^{N} (y_i - \vec{\lambda} \cdot \vec{\phi}(\vec{x}_i))^2 + N \log \sqrt{2\pi}\sigma.$$

We can minimize, and obtain analytic expressions for $\hat{\vec{\lambda}}$ and $\hat{\sigma}^2$ by differentiating $F(\vec{\lambda}, \sigma)$ with respect to $\vec{\lambda}$ and σ and setting the derivatives to be zero.

This gives an analytic solution for $\hat{\vec{\lambda}}$:

$$-\frac{1}{\sigma^2} \sum_{i=1}^{N} (y_i - \vec{\lambda} \cdot \vec{\phi}(\vec{x}_i)) \vec{\phi}(\vec{x}_i) = 0$$

$$\hat{\vec{\lambda}} = \{\sum_{i=1}^{N} \phi(\vec{x}_i) \phi(\vec{x}_i)^T\}^{-1} \sum_{i=1}^{N} y_i \vec{\phi}(\vec{x}_i),$$

where T denotes vector transpose and $^{-1}$ denotes matrix inverse. To see this, write $F(\vec{\lambda})$ using coordinate summation for the dot product terms – i.e. $(y_i - \vec{\lambda} \cdot \vec{\phi}_b(\vec{x}_i))^2 = (y_i - \sum_a \vec{\lambda}_a \vec{\phi}_a(\vec{x}_i))^2$. Then the solution is $\hat{\lambda}_a = \{\sum_{i=1}^{N} \phi_a(x_i)\phi_b(x_i)\}^{-1} \sum_{i=1}^{N} y_i \phi_a(x_i)$, where we are taking the inverse of the matrix with row and column entries indexed by a and b.

We also get an analytic solution for $\hat{\sigma}$:

$$\hat{\sigma}^2 = \frac{1}{N} \sum_{i=1}^{N} (y_i - \hat{\vec{\lambda}} \cdot \vec{\phi}(\vec{x}_i)).$$

Hence we use MLE to estimate the regression coefficients $\hat{\vec{\lambda}}$ and the variance $\hat{\sigma}^2$.

This can be generalized to allow for vector-valued output.