Lecture 6. Regression

Spring 2014

Note Title 4/21/2013

Consider learning the conditional distribution $p(y|x)$.
This is often easier than learning the likelihoods $p(x|y)$, the prior $p(y)$, an then compute the posterior $p(y|x)$.
Typically x has many more dimensions than y.
This is considered a regression problem. This is over 200 years old. Gauss finding the planetoid Ceres.

In this lecture, we adress three types of regression problem.

(I) Binary regression. $y \in \{\pm 1\}$

$$p(y|\underline{x};\lambda) = \frac{e^{y\underline{\lambda}\cdot\underline{\varphi}(\underline{x})}}{e^{\underline{\lambda}\cdot\underline{\varphi}(\underline{x})} + e^{-\underline{\lambda}\cdot\underline{\varphi}(\underline{x})}}, \text{ or } P(y|\underline{x};\lambda) = \frac{e^{y\underline{\lambda}\cdot\underline{\varphi}(\underline{x})}}{Z[\underline{x},\underline{\lambda}]}$$

with $Z[\underline{x},\underline{\lambda}] = e^{\underline{\lambda}\cdot\underline{\varphi}(\underline{x})} + e^{-\underline{\lambda}\cdot\underline{\varphi}(\underline{x})}$

(II) Linear Gaussian

$$p(y|\underline{x}:\underline{\lambda}) = \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{(y-\underline{\lambda}\cdot\underline{\varphi}(\underline{x}))^2}{2\sigma^2}} \quad (\text{linear in } \underline{\lambda})$$

This can be extended to vector-valued y.

In both cases, the parameters can be estimated by maximum likelihood (ML).

$$\underset{\text{w.r.t. } \underline{\lambda}}{\text{Minimize}} \quad -\sum_{i=1}^{N} \log P(y_i|\underline{x}_i;\underline{\lambda})$$

This is a convex optimization problem.

We can also include a prior $P(\underline{\lambda})$ and perform MAP optimization.

(III) Non-linear Regression: E.G. Multi-Layer Perceptron (Deep Networks)

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

$M(.,.)$ similarity measure
$g(x:\lambda)$ non-linear in λ.

This leads to non-linear optimization problems
More powerful. More computationally intensive.

(2)

Binary Regression:

$$P(y|\underline{x}) = \frac{e^{y\underline{\lambda}\cdot\underline{\varphi}(\underline{x})}}{e^{\underline{\lambda}\cdot\underline{\varphi}(\underline{x})} + e^{-\underline{\lambda}\cdot\underline{\varphi}(\underline{x})}}$$

Note: this leads to a decision rule:
classify $\underline{x}$ as $y=+1$, if $\underline{\lambda}\cdot\underline{\varphi}(\underline{x}) > 0$
as $y=-1$, if $\underline{\lambda}\cdot\underline{\varphi}(\underline{x}) < 0$
or $\hat{y}(\underline{x}) = \arg\min_y \; y\,\underline{\lambda}\cdot\underline{\varphi}(\underline{x})$

Minimize: $-\sum_{i=1}^{N} \log p(y_i|\underline{x}_i;\underline{\lambda})$

$$= -\sum_{i=1}^{N} y_i\,\underline{\lambda}\cdot\underline{\varphi}(\underline{x}_i) + \sum_{i=1}^{N}\log\{e^{\underline{\lambda}\cdot\underline{\varphi}(\underline{x}_i)} + e^{-\underline{\lambda}\cdot\underline{\varphi}(\underline{x}_i)}\}$$

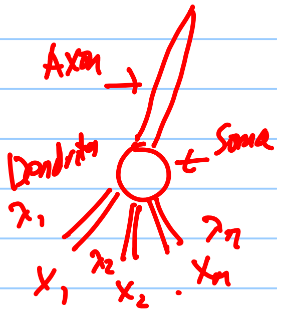
This is a convex function of $\underline{\lambda}$
(check → second term can be written as $\sum_i \underline{\lambda}\cdot\underline{\varphi}(\underline{x}_i)$)
$+ \sum_i \log\{1 + e^{-\underline{\lambda}\cdot\underline{\varphi}(\underline{x}_i)}\}$ ... Hessian is +ve definite.

The gradient of $-\sum_{i=1}^{N}\log P(y_i|\underline{x}_i;\underline{\lambda})$ w.r.t. $\underline{\lambda}$
is $-\sum_{i=1}^{N} y_i\,\underline{\varphi}(\underline{x}_i) + \sum_{i=1}^{N}\sum_{y\in\{\pm1\}} y\,\underline{\varphi}(\underline{x}_i)P(y|\underline{x}_i;\underline{\lambda})$

Hence the minimum also balances the statistics: $\hat{\underline{\lambda}}$ s.t.

$$\sum_{i=1}^{N} y_i\,\underline{\varphi}(\underline{x}_i) = \sum_{i=1}^{N}\sum_{y\in\{\pm1\}} y\,\underline{\varphi}(\underline{x}_i)P(y|\underline{x}_i;\hat{\underline{\lambda}}).$$

Steepest Descent: $\underline{\lambda}^{t+1} = \underline{\lambda}^{t} - \Delta\{-\sum_{i=1}^{N} y_i\,\underline{\phi}(\underline{x}_i) + \sum_{i=1}^{N}\sum_{y\in\{\pm1\}} y\,\underline{\phi}(\underline{x}_i)P(y|\underline{x}_i;\underline{\lambda})\}$
or variants, can be used
to minimize $-\sum_{i=1}\log P(y_i|\underline{x}_i;\underline{\lambda})$



Special case - idealized "neuron"

$\underline{\varphi}(\underline{x}) = (x_1, x_2 .\; .\, x_M)$
$\underline{\lambda} = (\lambda_1, \lambda_2, .\; \lambda_M)$

$\underline{\lambda}\cdot\underline{\varphi}(\underline{x}) = \lambda_1x_1 + \lambda_2x_2 + .. + \lambda_Mx_M.$

Integrate-And-Fire: If $\underline{\lambda}\cdot\underline{\varphi}(\underline{x}) > 0$, then the neuron will fire with probability. $\Big|\; P(y|x) = \frac{e^{y\underline{\lambda}\cdot\underline{\phi}(x)}}{e^{\underline{\lambda}\cdot\underline{\phi}(x)} + e^{-\underline{\lambda}\cdot\underline{\phi}(x)}}$

Or threshold $\underline{\lambda} = (\lambda_0, \lambda_1, .. \lambda_n)$
$\underline{\varphi}(\underline{x}) = (1, x_1, \; , x_n)$

$\underline{\lambda}\cdot\underline{\varphi}(\underline{x}) > 0 \quad , \quad \sum_{i=1}^{n}\lambda_i x_i > -\lambda_0 .$

(3) (II) Gaussian Linear Regression.

$$y = \underline{\lambda} \cdot \underline{\varphi}(x) + \underset{\text{zero mean Gaussian}}{\epsilon} \qquad P(\epsilon) = \frac{1}{\sqrt{2\pi}\sigma} e^{-\epsilon^2/2\sigma^2}$$

$$P(y \mid \underline{x}, \underline{\lambda}, \sigma) = \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{1}{2\sigma^2}(y - \underline{\lambda} \cdot \underline{\varphi}(\underline{x}))^2}$$

ML: minimize $- \sum_{i=1}^{N} \log P(y_i \mid \underline{x}_i, \underline{\lambda}, \sigma)$ dataset $\mathcal{X} = \{(\underline{x}_i, y_i) : i = 1, .., N\}$

$$= \sum_{i=1}^{N} \frac{1}{2\sigma^2} (y_i - \underline{\lambda} \cdot \underline{\varphi}(\underline{x}_i))^2 - N \log\{\sqrt{2\pi}\sigma\}.$$

Minimize w.r.t. $\underline{\lambda}$

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

solution $\hat{\underline{\lambda}} = \left\{ \sum_{i=1}^{N} \underline{\varphi}(\underline{x}_i) \underline{\varphi}(\underline{x}_i)^T \right\}^{-1} \sum_{i=1}^{N} y_i \underline{\varphi}(\underline{x}_i)$

Minimize w.r.t. σ $\quad -\frac{1}{\sigma^3} \sum_{i=1}^{N} (y_i - \underline{\lambda} \cdot \underline{\varphi}(\underline{x}_i))^2 - \frac{N}{\sigma}$

solution $\hat{\sigma}^2 = \frac{1}{N} \sum_{i=1}^{N} (y_i - \hat{\underline{\lambda}} \cdot \underline{\varphi}(\underline{x}_i))^2$ //

This estimates the regression coefficients $\underline{\lambda}$ and also the variance.

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

Note: in terms of coordinates

$$\sum_{i=1}^{N} (y_i - \sum_a \lambda_a \varphi_a(x_i))(y_i - \sum_b \lambda_b \varphi_b(x_i))$$

$$\frac{\partial}{\partial \lambda_a} = 2(-1) \sum_{i=1}^{N} \varphi_a(x_i)(y_i - \sum_b \lambda_b \varphi_b(x_i))$$

Then the minimum occurs at

$$\sum_b \left\{ \sum_{i=1}^{N} \varphi_a(x_i) \varphi_b(x_i) \right\} \lambda_b = \sum_{i=1}^{N} y_i \varphi_a(x_i).$$

$\sum_{i=1}^{N} \varphi_a(x_i) \varphi_b(x_i)$ are the coefficients of the ath row and bth column of a matrix.

(4) Spring 2013.

L^1 · Variant .

$y = \underline{\lambda} . \underline{\varphi}(\underline{x}) + \epsilon$, where $P(\epsilon) = \frac{1}{2\sigma} e^{-|\epsilon|/\sigma}$

$$\left(\int_0^\infty e^{-\epsilon/\sigma} d\epsilon = \left[-\sigma e^{-\frac{\epsilon}{\sigma}} \right]_0^\infty = \sigma \right).$$

$$P(y | \underline{x}, \underline{\lambda}, \sigma) = \frac{1}{2\sigma} e^{-\frac{1}{\sigma} | y - \underline{\lambda} . \underline{\varphi}(\underline{x}) |}.$$

Estimate $\underline{\lambda}, \sigma$ by ML

$$\frac{1}{\sigma} \sum_{i=1}^N | y_i - \underline{\lambda} . \underline{\varphi}(\underline{x}_i) | + N \log(2\sigma)$$

$$\hat{\underline{\lambda}} = \arg\min_{\underline{\lambda}} \sum_{i=1}^N | y_i - \underline{\lambda} . \underline{\varphi}(\underline{x}_i) |$$

This is a convex optimization problem - steepest descent and other algorithm will get $\hat{\underline{\lambda}}$

$$\hat{\sigma} = \frac{1}{N} \sum_{i=1}^N | y_i - \hat{\underline{\lambda}} . \underline{\varphi}(\underline{x}_i) |.$$

This is more robust than the previous (Gaussian) method. It is L_1 norm instead of L_2 .

(5) Speific Examples:

$\underline{\lambda} \cdot \underline{\varphi}(\underline{x}) = \omega_0 + \omega_1 x_1 + \dots + \omega_d x_d = \sum_{j=1}^{d} \omega_j x_j + \omega_0$

$\underline{\lambda} = (\omega_0, \omega_1 \dots, \omega_d)$, $\underline{\phi}(\underline{x}) = (1, x_1, \dots x_d)$

Simplest case: $y = \omega_1 x + \omega_0$

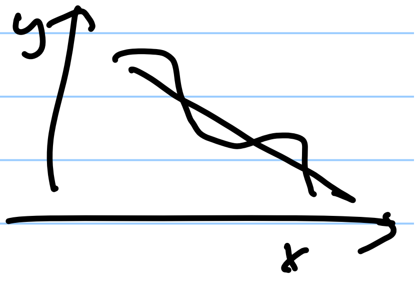
Dataset $\mathcal{X}_N = \{ (x^i, y^i) : i = 1 \text{ to } N \}$

$$E(\omega_1, \omega_0 | \mathcal{X}_N) = \sum_{i=1}^{N} \{ y^i - (\omega_1 x^i + \omega_0) \}^2$$

Minimize: $\frac{\partial E}{\partial \omega_1} = 0$ & $\frac{\partial E}{\partial \omega_0} = 0$

solution
$$\begin{cases} \hat{\omega}_1 = \dfrac{\sum_{i=1}^{N} x^i y^i - \bar{x}\bar{y}N}{\sum_i (x^i)^2 - N\bar{x}^2} \\ \hat{w}_0 = \bar{y} - \omega_1 \bar{x} \end{cases}$$

where $\bar{x} = \sum_{i=1}^{N} x_i / N$, $\bar{y} = \sum_i y_i / N$



A "richer" model can be used
→ eg $\underline{\lambda} \cdot \underline{\phi}(\underline{x}) = \omega_2 x^2 + \omega_1 x + \omega_0$

Too high an order follow the data too closely.

(6) More abstractly

Linear regression: $\lambda . \underline{\Phi}(\underline{x}) = \omega_1 x^i + \omega_0$.

Differentiate energy w.r.t ω_1, ω_0 gives two equations

$$\sum_i y^i = N\omega_0 + \omega_1 \sum_i x^i$$

$$\sum_i y^i x^i = \omega_0 \sum_i x^i + \omega_1 \sum_i (x^i)^2 .$$

Expressed in linear algebra form as $\underline{A}\,\underline{\omega} = \underline{y}$

$$\underline{\underline{A}} = \begin{bmatrix} N & \sum_i x^i \\ \sum_i y^i & \sum_i (x^i)^2 \end{bmatrix}, \quad \underline{\omega} = \begin{bmatrix} \omega_0 \\ \omega_1 \end{bmatrix}, \quad \underline{y} = \begin{bmatrix} \sum_i y^i \\ \sum_i x^i y^i \end{bmatrix}$$

Solved to give $\underline{\omega} = \underline{\underline{A}}^{-1} \underline{y}$.

Polynomial Regression.

$$g(x^i | \omega_k \ldots, \omega_2, \omega_1, \omega_0) = \omega_k (x^i)^k + \ldots + \omega_1 x^i + \omega_0$$

k+1 parameters $\omega_k \ldots \omega_0$

Diff. energy - gives k+1 linear eqn's in k+1 variables.

$$\underline{A}\,\underline{\omega} = \underline{y}.$$

Can write $\underline{\underline{A}} = \underline{\underline{D}}^T \underline{\underline{D}}$, $\underline{y} = \underline{\underline{D}}^T \underline{r}$ $\quad \underline{\underline{D}} = \begin{pmatrix} 1 & x_1 & x_1^k \\ 1 & x_2 & x_2^k \\ 1 & - & - \end{pmatrix}$

solve to get $\underline{\omega} = (D^T D)^{-1} D^T r$ $\quad r = \begin{bmatrix} r^1 \\ r^2 \\ \vdots \\ r^N \end{bmatrix}$

Must adjust the complexity of the model to the amount of data available

Complexity of poly regression is no. parameters k.

Need to pick k to give best generalization error

(7)

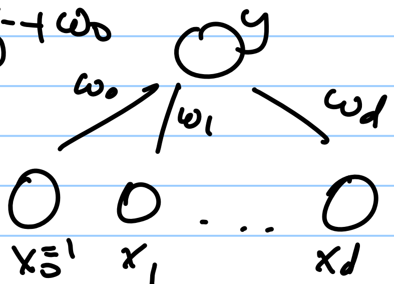
Note Title 4/27/2008

Multilayer Perceptrons

Analogy to Neural Networks in the Brain. - over-simplified.

perceptron. $y = \sum_{j=1}^{d} \omega_j x_j + \omega_0$

idealized neuron



threshold function.

hard $S(a) = \begin{cases} 1, & \text{if } a > 0 \\ 0, & \text{otherwise.} \end{cases}$ || linear discriminant.

soft. $y = \sigma(\underline{\omega}^T \underline{x}) = \dfrac{1}{1 + e^{-\underline{\omega}^T \underline{x}}}$. $\sigma(.)$ sigmoid function.

There are a variety of different algorithms to train a perceptron from labelled examples

Example: Quadrate error.

$E(\underline{\omega} \mid \underline{x}^t, y^t) = \frac{1}{2}(y^t - \underline{w} \cdot \underline{x}^t)^2$

update rule $\Delta \omega_j^t = -\Delta \frac{\partial E}{\partial \omega_j} = +\Delta (y^t - \underline{w} \cdot \underline{x}^t) \underline{x}^t$

$E(\{\underline{\omega}_i\}_i \mid \underline{x}^t, y^t) = -\sum_i (r_i^t \log y_i^t + (1 - r_i^t) \log(1 - y_i^t))$

$r^t = \text{sigmoid}(\underline{\omega}^T{}_i \underline{x}^t)$.

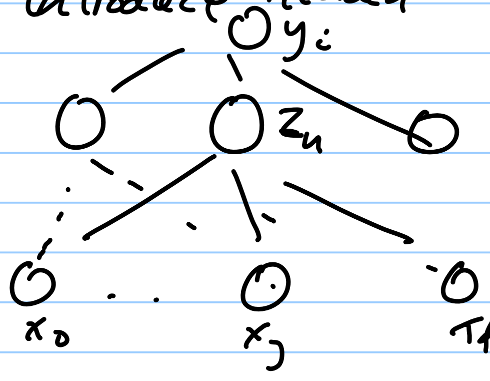
update rule $\Delta w_j^t = -\eta(r^t - y^t) x_j^t$.

Update = Learning Factor . (Desired Output - Actual Output) x Input.

Multilayer Perceptron.

A single layer perceptron can only approximate liner function of the input — i.e. the set of perceptrons has limited capacity.

Multilayer perceptrons were invented to increase the capacity — introduce hidden units (nodes)



$$z_h = \sigma(\underline{\omega}_h^T \underline{x})$$

$$= \frac{1}{1 + \exp\{-\sum_{j=1}^{d} \omega_{hj} x_j + \omega_{i0})\}},$$

$h = 1, .. H.$

Output $y_i = \underline{v}_i^T z = \sum_{h=1}^{H} v_{ih} z_h + v_{i0}.$

Other output function — e.g. $y_i = \sigma(\underline{v}_i^T \underline{z})$.

<u>Many</u> levels can be specified.

What do the hidden units represent?

Unclear., but many people have tried to explain them.

Any input-output function can be represented as a multilayer perceptron with enough hidden units — infinite capacity.

[9]

How to train a multilayer perceptron?

Unknown parameters – the weights ω_{hj}, v_{ij}.

Define an error function:

e.g. $$E[\omega, v] = \sum_i \Big(y_i - \sum_h v_{ih} \; \sigma \big(\sum_j \omega_{hj} x_j \big) \Big)^2$$

update $$\Delta \omega_{hj} = \frac{-\partial E}{\partial \omega_{hj}}.$$ computed by the chain rule.

$$\Delta v_{ik} = \frac{-\partial E}{\partial v_{ih}}$$ computed directly

Define $r_k = \sigma(\sum_j w_{kj} x_j)$, $E = \sum_i (y_i - \sum_k v_{ik} r_k)^2$

$$\frac{\partial E}{\partial \omega_{kj}} = \frac{\partial E}{\partial r_k} \cdot \frac{\partial r_k}{\partial \omega_{kj}}$$

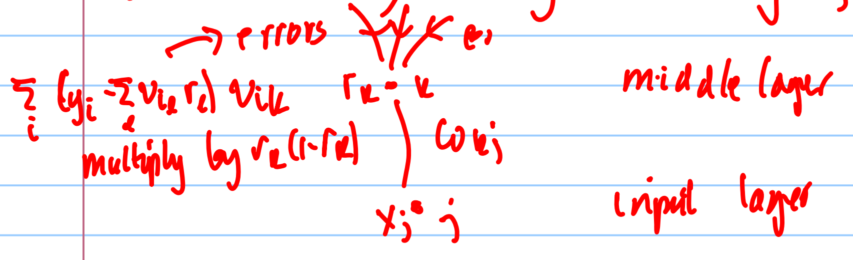
$$\frac{\partial E}{\partial r_k} = -2 \sum_i \Big(y_i - \sum_l v_{il} r_l \Big) v_{ik}$$

$$\frac{\partial r_k}{\partial \omega_{kj}} = x_j \, \sigma'\big(\sum_j \omega_{kj} x_j \big)$$

$$\sigma'(z) = \frac{d}{dz} \sigma(z) = \sigma(z) \{ 1 - \sigma(z) \}.$$

Hence $$\frac{\partial E}{\partial \omega_{hj}} = -2 \sum_i \underbrace{(y_i - \sum_l v_{il} r_l)}_{\text{error at output layer}} \underbrace{v_{ik}}_{\text{weights from middle layer to output layer.}} x_k \, \sigma(\sum_j \omega_{kj} x_j) \{ 1 - \sigma(\sum_j \omega_{kj} x_j) \}$$

This is called backpropagation. The error at the output layer is propagated back to the nodes at the middle layer $\sum_i (y_i - \sum_l v_{il} r_l) v_{ik}$ where it is multiplied by the activity $r_k(1 - r_k)$ at that node, and by the activity x_j at the input.



(10)

Learning in Batch Mode:

Put all data into an energy function - ie. Sum the errors over all the training data. Update the weights - equations above - by summing over all the data.

Alternatively, online learning.

At time t, select one element $(\underline{x}^t, y^t)$ from the training set at random.
Perform one iteration of steepest descent only.
Then select another element at random.

Online learning can be shown to converge and may be better at avoiding local minima in the energy function than batch methods.
Also, online is suitable if we keep getting new input over time - this happens in many real world applications.

Critical Issues for multilayer perceptrons

— how many hidden units to use?

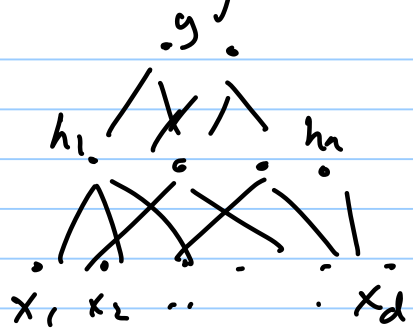
<u>Book desribes</u> severl techniques for dealing with this → for example, having more hidden units than you need and then penalizing the weights.

E.G. Add term $\sum_{h,j}(\omega_{hj}^2) + \sum_{i,h}(v_{ih}^2)$ to the cost function → intuition, if the weights are small to a hidden unit, then the hidden unit is not used.

In practice, some of the most effective multi layer perceptrons are those which the structure was hand designed.

Multilayer Perceptrons / SVM / AdaBoost

(next two lectures)



$$y_i = \sum_h v_{ih} h_h$$
$$h_h = \sum_j \sigma\left(\sum w_{hj} x_j\right).$$

SVM can also be represented in this way

$$y = \text{sign}\left(\sum_\mu \alpha_\mu y_\mu \underline{x}_\mu \cdot \underline{x}\right)$$

hidden units response $\underline{x}_\mu \cdot \underline{x} = h_\mu$.

$$y = \text{sign}\left(\sum_\mu \alpha_\mu y_\mu h_\mu\right)$$

Advantage of SVM — number of hidden units is given by the no. of support vectors.

$\rightarrow \{\alpha_\mu\}$ specified by maximizing the primal problem (well defined algorithm to perform this minimization).