Oxford DL Course Notes

pjhades

March 17, 2016

Course: https://www.cs.ox.ac.uk/people/nando.defreitas/machinelearning/

Lecture 1

Some introduction ...

Lecture 2 - Linear Regression

The loss function is also called energy, which is:

$$J(\theta) = (\mathbf{y} - \mathbf{X}\boldsymbol{\theta})^T(\mathbf{y} - \mathbf{X}\boldsymbol{\theta}) = \sum_{i=1}^{n}(y_i - \mathbf{x}_i^T\boldsymbol{\theta})^2$$

Optimization: Differentiation, $\boldsymbol{\theta} = (\mathbf{X^T X})^{-1}\mathbf{X^T y}$

Lecture 3 - Maximum Likelihood

Univariate Gaussian:

$$p(x) = \frac{1}{\sqrt{2\pi\sigma^2}}e^{-\frac{(x-\mu)^2}{2\sigma^2}}$$

Sampling from a univariate Gaussian – use **probability integral transformation**(see *Probability and Statistics* 4ed Page 170):

- Generate a random number from a uniform distribution on $[0, 1]$
- Apply to it the inverse of the normal c.d.f.

Variance-covariance matrix:

$$Cov(\mathbf{x}) = E[(\mathbf{x} - E(\mathbf{x}))(\mathbf{x} - E(\mathbf{x}))^T] = \begin{bmatrix} Var(x_1) & Cov(x_1, x_2) & \dots & Cov(x_1, x_d) \\ Cov(x_2, x_1) & Var(x_2) & \dots & Cov(x_2, x_d) \\ \vdots & \vdots & \dots & \vdots \\ Cov(x_n, x_1) & Cov(x_n, x_2) & \dots & Cov(x_n, x_d) \end{bmatrix}$$

where we compute the pairwise covariance among the features.

And the covariance matrix for the entire dataset:

$$Cov(\mathbf{X}) = \mathbf{X^T X}$$

where $\mathbf{X}$ is the data minus the mean, and each entry c_{ij} in the result holds the covariance between feature i and j computed over the entire sample.

Multivariate Gaussian:

p.d.f.:

$$p(\mathbf{x}|\boldsymbol{\mu}, \boldsymbol{\Sigma}) = \frac{1}{(2\pi)^{d/2}|\boldsymbol{\Sigma}|^{1/2}} \exp\{-\frac{1}{2}(\mathbf{x} - \boldsymbol{\mu})^T \boldsymbol{\Sigma}^{-1}(\mathbf{x} - \boldsymbol{\mu})\}$$

which can be derived from the joint density of two Gaussians.

Likelihood for linear regression

If the data consists of linear function + Gaussian noise:

$$y_i = \mathbf{x_i}^T \boldsymbol{\theta} + \mathcal{N}(0, \sigma^2)$$

Then the likelihood is:

$$p(\mathbf{y}|\mathbf{X}, \boldsymbol{\theta}, \sigma) = (2\pi\sigma^2)^{-n/2} e^{-\frac{1}{2\sigma^2}(\mathbf{y} - \mathbf{X}\boldsymbol{\theta})^T(\mathbf{y} - \mathbf{X}\boldsymbol{\theta})}$$

where the exponential is just the loss.

This is a usually seen form of likelihood functions:

$$p(\text{data}|\text{parameters}) = \text{constant} \cdot e^{-\text{loss}}$$

To optimize, we do the same thing in different ways:

- Minimizing loss – minimize the difference between estimated value and the true value (energy)
- MLE – maximize the likelihood of observing the given data

Lecture 4 - Nonlinear Ridge Regression, Risk, Regularization, Cross Validation

Linear regression + weight decay regularization.

Loss:

$$J(\theta) = (\mathbf{y} - \mathbf{X}\boldsymbol{\theta})^T(\mathbf{y} - \mathbf{X}\boldsymbol{\theta}) + \lambda\boldsymbol{\theta}^{\boldsymbol{T}}\boldsymbol{\theta}$$

The result is $\hat{\boldsymbol{\theta}} = (\mathbf{X}^{\mathbf{T}}\mathbf{X} + \lambda\mathbf{I})^{-1}\mathbf{X}^{\mathbf{T}}\mathbf{y}$

Use RBFs to introduce nonlinearity.

Lecture 5 - Optimization

XXX-Batch

Batch uses all training data $\{\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_n\}$ in the gradient computation. Mini-batch uses only a part. Online or SGD uses just one.

Newton's method

Update rule:

$$\boldsymbol{\theta}_{k+1} = \boldsymbol{\theta}_k - \mathbf{H}_k^{-1}\mathbf{g}_k$$

where the last term is the product of Hessian and gradient.

Newton's method uses second-order Taylor expansion (steepest descent is of order 1):

$$f(\boldsymbol{\theta}) = f(\boldsymbol{\theta}_k) + \mathbf{g}_k^T(\boldsymbol{\theta} - \boldsymbol{\theta}_k) + \frac{1}{2}(\boldsymbol{\theta} - \boldsymbol{\theta}_k)^T\mathbf{H}_k(\boldsymbol{\theta} - \boldsymbol{\theta}_k)$$

What this does is to use the Taylor expansion to approximate the loss function around the current point $\boldsymbol{\theta}_k$, and get the minimum of the approximation, which will act as a bound for the descending, adjusting the length of our steps according to the curvature of the loss function.

Applying Newton's method to linear regression produces the same result.

Lecture 6 - Logistic Regression as a Simple ANN

Logistic regression

- Use sigmoid function to convert a score to a value between 0 and 1.
- Each sample is then from a Bernoulli trial: the sigmoid function tells us the probability that some sample input results in a label 1
- Weights are usually estimated by MLE
- The error measure (cross-entropy) can be derived from the log-likelihood
- Optimization is done by gradient descent

Softmax

Multi-output logistic regression.

Each class has a weight vector $\boldsymbol{\theta}_k$, and the probability is given by:

$$p(y = k|\mathbf{x}, \boldsymbol{\theta}) = \frac{e^{\boldsymbol{\theta}_k^T \mathbf{x}}}{\sum_{i=1}^{K} e^{\boldsymbol{\theta}_i^T \mathbf{x}}}$$

Lecture 7 - Backpropagation: A Modular Approach

The most important idea introduced in this lecture is the modular structure or view of neural networks.

Each layer in the network can be regarded as an individual function box with inputs, outputs, and optional parameters:

- Inputs: data, or outputs produced by the previous layer
- Outputs: final result, or input for the next layer
- Parameters: control the behavior of this layer's functions

Based on this view, we can take derivatives without referring to other layers. The computation (forward pass) and derivatives (backward pass) of each layer can be derived separately.

Adding a new layer to the network is just like adding a new utility in the pipe.

Lecture 8 - Neural Networks

Nothing new.

Lecture 9 - Convnets