Machine Learning - Notes

François Lux

30 August, 2017

Contents

Machine Learning - Notes

Notation - Vectorized Expressions:

n_x: number of features
m: number of datasamples
$X = (n_x, m)$-Matrix, containing the data-samples, one sample each column
$w^T = (1, n_x)$-Vector, containing the weights for each feature
b = bias-scalar, extended to a vector (broadcasting)
$\hat{y} = a$: calculated probability of classification y = 1

1) Logistic Regression - Classification

Given input x, find the Probability that y = 1, where 0 and 1 denote the two possibilities of a binary classification problem.

Logistic Regression is still a linear classifier (Decision boundary is linear).
Linear: $z = wx + b$

Linear Regression references Gaussian distribution while Logistic regression references Binomial distribution.

So as we want a probability between 0 and 1 we use sigmoid-function.

$$\sigma(z) = \frac{1}{1 + e^{-z}}$$

Loss-function (for one example):

$$L(\hat{y}, y) = \frac{1}{2}(\hat{y} - y)^2$$

But optimization problem becomes non-convex (many local optima). So this causes problems with Gradient Descent.

Instead we use:

$$L(\hat{y}, y) = -(y \ log\hat{y} + (1-y) \ log(1-\hat{y}))$$

First case: y = 1: $L(\hat{y}, y) = - \ log\hat{y}$; wants a large $\hat{y}$ in order to minimize loss-function.

Second case: y = 0: $L(\hat{y}, y) = -log(1-\hat{y}))$; wants a small $\hat{y}$ in order to minimize loss-function.

Cost-function(for all test-data):

$$J(w, b) = \frac{1}{m} \sum_{i=1}^{m} L(\hat{y}^i, y^i)$$

Gradient Descent:

Initialize w and b, most often with 0 or random (does not matter too much because Loss-function is convex).

Repeat {
$w = w - \alpha \frac{dJ(w,b)}{dw}$
$b = b - \alpha \frac{dJ(w,b)}{db}$
}

If gradient is negative then update is positive, if gradient is positive the update is negative. Alpha is the learning rate.

Backpropagation: Using the chainrule to calculate the partial derivative of the final output (J or L) with respect to the input variables.

Vectorization

(for loops) are around 300 times slower than vectorized calculations because these can be calculated in parallel using SIMD(See test-script vectorization.py).

$Z = [z_1, z_2, ..., z_m] = w^T X + b = np.dot(w.T, X) + b$
$A = [a_1, a_2, ..., a_m] = [\sigma(z_1), \sigma(z_2), ...] = \sigma(Z)$

We need to calculate the derivative of the loss-function L with respect to the weights and the bias in order to update them.

This is done using the chainrule, so we first calculate

$$\frac{dL(a, y)}{da} = -\frac{y}{a} + \frac{1-y}{1-a}$$

and then we calculate

$$\frac{da(z)}{dz} = a(1-a)$$

From there follows that

$$\frac{dL}{dz} = \frac{dL(a, y)}{da} \cdot \frac{da(z)}{dz} = a - y$$

The last step then is

$$\frac{dL}{dw_1} = x_1 \cdot dz$$

$$\frac{dL}{dw_2} = x_2 \cdot dz$$

$$\frac{dL}{db} = dz$$

Vectorized:

$$dZ = A - Y$$

,

$$db = \frac{1}{m} np.sum(dZ)$$

$$dw = \frac{1}{m} X dz^T$$