CS229: Machine Learning

Junchuan Zhao

May 17, 2022

Abstract

This document includes my notes of CS229: Machine Learning.

1 Lecture 2

1.1 Linear Regression

Hypothesis: $h(x) = \sum_{j=0}^{n} \theta_j x_j$ (n: number of features)
Linear Regression: the hypothesis is the linear combination of the training dataset.
Loss function (goal): $\min_{\theta} \frac{1}{2} \sum_{i=1}^{m} (h_\theta(x^i) - y^i)^2$
Parameters: θ, m, n, x, y

1.2 Gradient Descent

1.2.1 Batch Gradient Descent

Basic Ideas: start with some θ, keep changing θ to reduce $J(\theta)$, repeat until convergent
$\theta_j := \theta_j - \alpha \frac{\partial}{\partial \theta_j} J(\theta)$ (α: learning rate)
$\frac{\partial}{\partial \theta_j} J(\theta) = (h_\theta(x) - y) \cdot x_j$
$\theta_j := \theta_j - \alpha \sum_{i=1}^{m} (h_\theta(x)^i - y^i) \cdot x_j^i$ (α: learning rate)
Batch Gradient Descent: all the training data as a batch when optimizing the loss function. (-): not good for large scale dataset, very expensive.

1.2.2 Stochastic Gradient Descent

Algorithm 1 Stochastic Gradient Descent

Require: the parameter of j^{th} feature
Ensure: the updated parameter of j^{th} feature
1: **for** $i = 1$ to m **do**
2: $\quad \theta_j := \theta_j - \alpha \cdot (h_\theta(x)^i - y^i) \cdot x_j^i$;
3: **end for**

stochastic gradient descent heads over to the global optimum.

1.2.3 Total Equations

A Total Equation for Stochastic Gradient Descent: $\nabla_\theta J(\theta) = \mathbf{0}$
If A is square ($A \in \mathbb{R}_{n\times n}$)
Trace of A: $tr(A) = \sum_i A_{ii}$
Features of Trace:

- $tr(A) = tr(A^T)$
- $f(A) = tr(AB)$, $\nabla_A f(A) = B^T$
- $tr(AB) = tr(BA)$
- $tr(ABC) = tr(CAB)$
- $\nabla_A tr(AA^TC) = CA + C^TA$

Loss Function: $\nabla_\theta J(\theta) = \frac{1}{2}(X\theta - y)^T(X\theta - y)$
$= \frac{1}{2}(\theta^TX^T - y^T)(X\theta - y)$
$= \frac{1}{2}(\theta^TX^TX\theta - \theta^TX^Ty - y^TX\theta)$
$= X^TX\theta - X^Ty$
Loss Function: $X^TX\theta - X^Ty = \mathbf{0} \iff X^TX\theta = X^Ty$
$\theta = (X^TX)^{-1}X^Ty$

2 Lecture 3

2.1 Locally Weighted Regression

"Parametric" learning algorithm: fit fixed set of parameters (θ_i) to data.
"Non-parametric learning algorithm": amount of data/parameters you need to keep grows (linearly) with the size of data.

Linear Regression: to evaluate h at certain x
Fit θ to minimize
$\frac{1}{2}\sum_i (y^i - \theta^Tx^i)^2$,
return θ^Tx

Linear Regression: to evaluate h at local region of x
Fit θ to minimize
$\sum_i w^i(y^i - \theta^Tx^i)^2$, where w^i is a "weight function".
common choice for w^i is $w^i = e^{(-\frac{(x^i-x)^2}{2\tau^2})}$
If $|x^i - x|$ is small, then $w^i \approx 1$.
If $|x^i - x|$ is large, then $w^i \approx 0$.
τ: bandwidth, control a larger or narrower window.

2.2 Why Square Error?

Assume $y^i = \theta^Tx^i + \epsilon^i$, where $\epsilon^i \sim N(0, \sigma^2)$ models effects of random noise
$p(y^i|x^i;\theta) = \frac{1}{\sqrt{2\pi}\sigma}e^{-\frac{(y^i-\theta^Tx^i)^2}{2\sigma^2}}$
$\iff y^i|x^i;\theta \sim N(\theta^Tx^i, \sigma^2)$

Difference between Likelihood (L) and Probability (P): L variess parameters, P varies datapoints. Assume the datapoints are IID,

The "likelihood" of θ: $L(\theta) = p(\boldsymbol{y}|\boldsymbol{x};\theta) = \prod_{i=1}^{m} p(y^i|x^i;\theta)$

$l(\theta) = \log \prod_{i=1}^{m} \frac{1}{\sqrt{2\pi}\sigma} e^{(\cdots)} = \sum_{i=1}^{m} [\log \frac{1}{\sqrt{2\pi}\sigma} + \log e^{(\cdots)}]$

$= m \log \frac{1}{\sqrt{2\pi}\sigma} - \sum_{i=1}^{m} \frac{(y^i - \theta^T x^i)^2}{2\sigma^2}$

MLE: maximum likelihood estimation. Choose θ to maximize $L(\theta)$

i.e. choose θ to minimize $\frac{1}{2}\sum_{i=1}^{m}(y^i - \theta^T x^i)^2$, which is actually $J(\theta)$

2.3 Classification

Binary Classification: $y \in \{0, 1\}$

2.3.1 Logistic Regression

We want $h_\theta(x) \in [0, 1]$, so we define

$h_\theta(x) = g(\theta^T x) = \frac{1}{1+e^{-\theta^T x}}$

Sigmoid Function: $g(z) = \frac{1}{1+e^{-z}}$

An important thing is that: the partial derivative of the sigmoid function is a **concave** function, which means it has the global maximum.

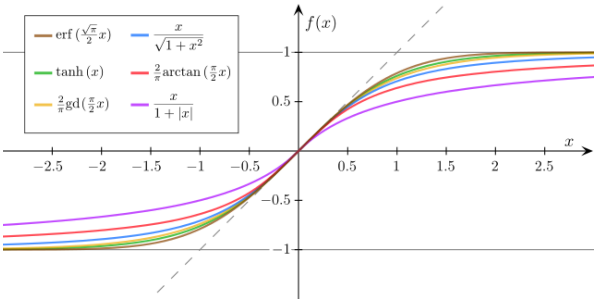

Figure 1: The graph of sigmoid function.

Different from Linear Regression $h_\theta(x) = \theta^T x$, Logistic Regression is $h_\theta(x) = \frac{1}{1+e^{-\theta^T x}}$.

$p(y = 1|x;\theta) = h_\theta(x)$, $p(y = 0|x;\theta) = 1 - h_\theta(x)$

In sum, $p(y|x;\theta) = h_\theta(x)^y (1 - h_\theta(x))^{1-y}$

$L(\theta) = p(\boldsymbol{y}|\boldsymbol{x};\theta) = \prod_{i=1}^{m} h_\theta(x^i)^{y^i}(1 - h_\theta(x^i))^{1-y^i}$

$l(\theta) = \log(L(\theta)) = \sum_{i=1}^{m} y^i \log h_\theta(x^i) + (1 - y^i)\log(1 - h_\theta(x^i))$

Choose θ to maximize $l(\theta)$, we use Batch gradient ascent.

Batch Gradient Ascent:

$\theta_j := \theta_j + \alpha \frac{\partial}{\partial \theta_j} l(\theta)$

$\theta_j := \theta_j + \alpha \sum_{i=1}^{m}(y^i - h_\theta(x^i))x_j^i$, same as the one in linear regression. (A larger category called generalized linear model (GLM))

Note: no **normal equation** for logistic regression solution.

2.3.2 Newton's Method

The basic idea: have some function f, we want to fit θ, **s.t.** $f(\theta) = 0 \iff$ want maximizing $l(\theta)$ $\iff$ want $l'(\theta) = 0$

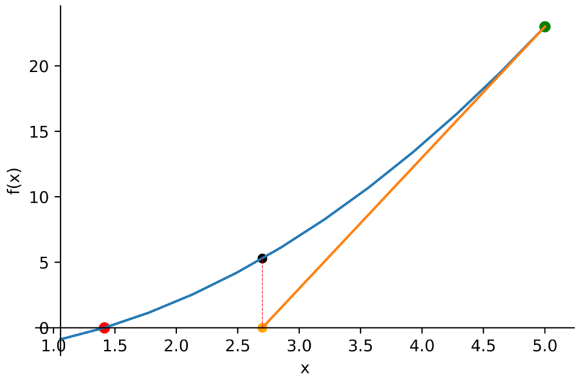

Figure 2: Newton's Method.

$\theta^{t+1} := \theta^t - \frac{f(\theta^t)}{f'(\theta^t)}$, let $f(\theta) = l'(\theta)$
$\theta^{t+1} := \theta^t - \frac{l'(\theta^t)}{l''(\theta^t)}$

"**Quadratic convergent**": 0.01 error $\longrightarrow$ 0.0001 error $\longrightarrow$ 0.00000001 error.

When θ is a vector: $(\theta \in \mathbb{R}^{n+1})$
$\theta^{t+1} := \theta^t + \alpha H^{-1} \nabla_\theta l(\theta)$
where

$$\nabla_\theta l(\theta) = \begin{bmatrix} \frac{\partial l}{\partial \theta_{00}} & \frac{\partial l}{\partial \theta_{01}} & \cdots & \frac{\partial l}{\partial \theta_{0n}} \\ \frac{\partial l}{\partial \theta_{10}} & \frac{\partial l}{\partial \theta_{11}} & \cdots & \frac{\partial l}{\partial \theta_{1n}} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial l}{\partial \theta_{n0}} & \frac{\partial l}{\partial \theta_{n1}} & \cdots & \frac{\partial l}{\partial \theta_{nn}} \end{bmatrix}$$

$H \in \mathbb{R}^{n+1 \times n+1}$ is the Hessian Matrix, $H_{ij} = \frac{\partial^2 l}{\partial \theta_i \partial \theta_j}$
Note: If the dataset is large, the size of H will be too large to compute.