machine learning mathematics

mohab metwally

11/2020

Contents

Introduction

0.1 notation

$(x, y) \in R^{n_x}, y \in 0, 1$ are the training dataset, where x is input, and y is corresponding output.

therefore $M = \{(x^1, y^1), (x^2, y^2), ..., (x^m, y^m)\}$ is the training set, and M_{test} is test example.

X is the input of training set and $\in R^{n_x \times m}$, $X = \begin{vmatrix} X^1 & X^2 & ... & X^m \end{vmatrix}$, $X^{(i)}$ is the ith column of length n, and can be written as x, or $x^{(i)}$.

Chapter 1

Logistic Regression as a neural network

1.1 definitions

we need an estimate function $\hat{y}$ for the input x, and weight prameters w$\in$ $R^{n_x}, b \in R$.

logstic function is $\hat{y} = b(y = 1|x)$, and can be defined as follows: $\hat{y} = \sigma(w^T x + b)$, where the sigma function is defined by $\sigma(z) = \frac{1}{1+e^{-z}}$, and notice when z $\to \infty, \sigma = 1, z \to -\infty, \sigma = 0$.

1.2 cost function

loss function is minimizing to the difference between estimation$\hat{y}$, y, can be defined as least squre $L(\hat{y}, y) = \frac{1}{2}(\hat{y} - y)$, but least squares leads to non-convex loss function(with multiple local minimums).

loss function is defined as $L(\hat{y}, y) = -[ylog(\hat{y}) + (1-y)log(1-\hat{y})]$, L$\in$ $[0-1]$.

loss function derivative:

add derivative here

cost function is defined as the average of loss function $J(w, b) = \frac{1}{m} \sum_{i=1}^{m} L(\hat{y^{(i)}}, y)$

1.3 Gradient Descend

gradient descend is a way to tune the weighting parameters with respect to the cost function, the objective is the lean toward the fittest weights with respect to the least cost.

iterate through cost function **J** tuning with respect to weight parameters **w**, **b**.

iterate through: $w := w - \alpha \frac{\partial J}{\partial w}$, $b := b - \alpha \frac{\partial J}{\partial b}$, for tuning w, b for the least **J** possible, such that α is the learning rate of GD.

for simplicity $\partial J/\partial w$ replaced for ∂w, and similarly $\partial J/\partial b$ is replaced for ∂b.

forward propagation, $\partial w = \frac{\partial J}{\partial L}\frac{\partial L}{\partial \hat{y}}\frac{\partial \hat{y}}{\partial z}\frac{\partial z}{\partial w}$, similarly $\partial b = \frac{\partial J}{\partial L}\frac{\partial L}{\partial \hat{y}}\frac{\partial \hat{y}}{\partial z}\frac{\partial z}{\partial b}$.

$\partial L/\partial \hat{y} = \frac{-y}{\hat{y}} + \frac{(1-y)}{1-\hat{y}}$, $\partial \hat{y}/\partial z = \frac{-e^{-z}}{1+e^{-z}} = \hat{y}(1-\hat{y})$.

$\partial L/\partial z = \hat{y} - y$.

then we can deduce that the final iteration gradient descend step after calculating sigma, loss, and cost functions can be $w := w - \frac{\alpha}{m}\sum_{i=1}^{m}\frac{\partial L}{\partial b} = \frac{\alpha}{m}X^T(\hat{y}-y)$, and $b := b - \frac{\alpha}{m}\sum_{i=1}^{m}(\hat{y}-y)$.

Chapter 2

Natural language processing

2.1 Logistic regression classifier

the problem here is how to extract features **X** from the a sentence.

for example how to classify a sentence being positive, or negative, assigning 0 for negative, and 1 for positive, starting for a preprocessed sentence how to turn it into a feature set X.

but there are unnecessary punctuation, conjugation, and stops that need to be get rid of, so first we need to pre-process our dataset as follows:

1. iliminate handles and URLs
2. tokenize the string into words
3. remove stop words such as "and, is, a, on, etc"
4. covert every word into it's stem form
5. lower case transformation

starting with sentence s="i love NLP, therefore i study it" how to classify s=['i', 'love', 'NLP', ',', 'therefore', 'i', 'study', 'it'], $X_m = [1, freqs_{pos}, freqs_{neg}]$.

for a set of strings $S = s_1, s_2, .., s_n$, matching each string against a vocabulary of all words to end up with two vectors of word frequency, we create positive frequency vector freqs(w,1), and negative frequency vector freqs(w,0) such that w stand for sentence word, and such that $X_m = [1, \sum_w freqs(w, 1), \sum_w freqs(w, 0)]$

for example in training sets s_1, s_2, s_1="i love NLP, therefore i study it" labled as positive, and s_2="society no longer in need for black magic, or superstition" we can extract pos-neg features against vocabulary V=['i', 'love', 'NLP', ',', 'therefore', 'study', 'it', 'society', 'no', 'longer', 'in', 'need', 'for', 'black', 'magic', 'or', 'superstition'], pos-freqs = [2, 1, 1, 1, 1, 1, 0, 0, 0, ..., 0], neg-freqs = [0, 0, 0, 1, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1]. X_m=[1, 8, 11]. for m training sets, $X_m = \begin{vmatrix} 1 & x_1^{(1)} & x_2^{(1)} \\ 1 & x_1^{(2)} & x_2^{(2)} \\ 1 & x_1^{(3)} & x_2^{(3)} \\ ... & & \\ 1 & x_1^{(m)} & x_2^{(m)} \end{vmatrix}$

let's see how this fits inside the Gradient Descend algorithm, for $\sigma(X^i, \theta^i) = \frac{1}{1+e^{-z}}$, we add the bias b to the X^i itself, and therefore $z = \theta^T X^i$.

therefore for a positive z we get $\sigma > 0.5$, and inversely for negative z we have $\sigma < 0.5$.

now we ready for Gradient Descend, given initial parameters θ, we predict, or evaluate the logistic:

$\sigma(X^i, \theta_j)$, then loss function $L(\hat{y}, y) = -[\hat{y}log(\hat{y}) - (1-y)log(1-\hat{y})]$, gradient $\nabla = \partial J / \partial \theta_j = \frac{X^T}{m}(\hat{y} - y)$, updating $\theta_j := \theta_j - \alpha\nabla$. iterate through gradient descend k times.

2.2 Naive Bayes classifier

Chapter 3

Neural Networks

3.1 Lingua franca

- RELU Activation Function: It turns out that using the Tanh function in hidden layers is far more better. (Because of the zero mean of the function). Tanh activation function range is [-1,1] (Shifted version of sigmoid function). Sigmoid or Tanh function disadvantage is that if the input is too small or too high, the slope will be near zero which will cause us the gradient decent problem. RELU stands for rectified linear unit, it's a rectifier Activation function and can be defined as $f(x) = x^+ = max(0, x)$ or $\begin{cases} 0 & x \leq 0 \\ x & x > 0 \end{cases}$ relu shows to be better replacement to sigmoid function σ for the reason that it help in vanquishing gradient problem.

- Neuron: is a linear regression algorithm denoted by z, or a, $z = W^T X + b$, such that W is the the weight vector of the network.

- Shallow Layers: also known as Hidden Layers, is a set of neurons, for example of the network of composed of input X, and output Y, with at least a single layer $L1$, and at most 2 layers, then the forward propagation will be as follows: we calculate the logistic function for the first layer (1), $z_i^1 = w^T X_i + b_i$, $\hat{Y}^{(1)} = \sigma(z_i^1)$, then we proceed to calculate the final logistic evaluation for the output layer with $\hat{Y}^{(1)}$ as an input instead of X, and so on we proceed replacing $\hat{Y}^{(i)}$ instead of X as new input.

- Layer: layer $L_{(i)}$ is $\hat{Y}^{(i)} = [\hat{y}_1^{(i)}, \hat{y}_2^{(i)}, ..., \hat{y}_n^{(i)}]$ such that n is the length of the layer $L_{(i)}$. each $\hat{y}_j^{(i)}$ is weighted with unique weight vector with previous layer $L_{(i-1)}$.
- Neural Network(NN): is a set of interconnected layers, $< X, L_1, L_2, ..., L_m, Y >$
- Deepness: shallow layer as defined to consist of 1-2 hidden layers, but on the other hand Deep Network is consisting of more than 2 inner, or hidden layers.

we discussed in previous chapter that $\frac{\partial \hat{y}}{\partial z}$, is actually for the logistic activation function σ only, we need to calculate the same derivation for tanh, and RELU.

for $tanh(z) = \frac{e^z - e^{-z}}{e^z + e^{-z}}$ is $\frac{\partial \hat{y}}{\partial z} = 1 - tanh(z)^2$

for relu activation function $\frac{\partial \hat{y}}{\partial z} = \begin{cases} 0 & z < 0 \\ 1 & z \geq 0 \end{cases}$

in NN there are plenty of parameters to worry about, for example the weights need to be initialized randomly, with small values, and b can be initialized as zero.

Appendices

Appendix A

Introduction to probabilities