Machine learning memo

Daniel Perez

1 Model representation

1.1 Symbols

- m = number of training example
- x's = "input" variable / features
- y's = "output" variable / "target" variable
- (x, y) - one training example
- $(x^{(i)}, y^{(i)})$ - i^{th} training example
- $h : x \rightarrow y$ - hypothesis function (takes input and estimates output)

1.1.1 Linear regression

Also called univariate linear regression.

h is a linear function.

$$h_\theta(x) = \theta_0 + \theta_1 x$$

1.2 Cost function

θ_i are the parameters of the equation. For linear regression: $h_\theta(x) = \theta_0 + \theta_1 x$

Cost function (square error cost function)

$$J(\theta_0, \theta_1) = \frac{1}{2m} \sum_{i=1}^{m} \left(h_\theta(x^{(i)}) - y^{(i)} \right)^2$$

We want to minimize over the parameters θ_0 and θ_1:

$$\min_{\theta_0, \theta_1} J(\theta_0, \theta_1)$$

The square error cost function is one of the most used for regression.

2 Multiple features

2.1 Notations

- n is the number of features
- $x^{(i)}$: input features of i^{th} training example
- $x_j^{(i)}$: input feature j of i^{th} training example

2.2 Hypothesis

$$h_\theta(x) = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \cdots + \theta_n x_n$$

For convenience, $x_0 = 1$, so that

$$x = \begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix} \in \mathbb{R}^{n+1} \qquad \theta = \begin{bmatrix} \theta_0 \\ \theta_1 \\ \theta_2 \\ \vdots \\ \theta_n \end{bmatrix} \in \mathbb{R}^{n+1}$$

we can therefore write

$$\begin{aligned} h_\theta(x) &= \theta_0 x_0 + \theta_1 x_1 + \theta_2 x_2 + \cdots + \theta_n x_n && (1) \\ &= \theta^T x && (2) \end{aligned}$$

which means

$$h_\theta(x) = \underbrace{\begin{bmatrix} \theta_0 & \theta_1 & \cdots & \theta_n \end{bmatrix}}_{\theta^T} \cdot \begin{bmatrix} x_0 \\ x_1 \\ \vdots \\ x_n \end{bmatrix}$$

This is also called Multivariate linear regression.

2.3 Gradient descent

- Parameters: $\theta_0, \theta_1, \cdots, \theta_n$ which we think of as $\theta \in \mathbb{R}^{n+1}$
- Cost function: $J(\theta) = \frac{1}{2m} \sum_{i=1}^{m} \left(h_\theta(x^{(i)} - y^{(i)}) \right)$ where $\theta = \begin{bmatrix} \theta_0 & \theta_1 & \cdots & \theta_n \end{bmatrix}$

Gradient descent: repeat

$$\theta_j := \theta_j - \alpha \frac{\partial}{\partial \theta_j} J(\theta)$$

until we converge.
with simultaneous update. By computing the partial derivative, this gives us

$$\theta_j := \theta_j - \alpha \frac{1}{m} \sum_{i=1}^{m} \left(h_\theta(x^{(i)}) - y^{(i)} \right) x_j^{(i)}$$

2.4 Feature scaling

Use features on the same scale (with same range of values).
e.g.

- Size of the bedroom (100 - 2000feet^2)
- Number of beds (1 - 5)

We will want to have values such as $-1 \leq x_i \leq 1$ Otherwise, gradient descent will converge slowly.

2.5 Mean normalization

Replace x_i by $x_i - \mu_i$, so that the mean of each feature is around 0

Combining both:

$$x_i \leftarrow \frac{x_i - \mu_i}{s_i}$$

where s_i can be either the the range (max - min) or the standard derivation of the feature.

2.6 Learning rate

Gradient descent should decrease after each iteration.

Plotting cost function (y axis) and number of iteration (x axis) helps to make sure gradient descent is working.

If cost function increases with gradient descent increases, use smaller learning rate α. Same if the cost goes up and down.

- For sufficiently small α, $J(\theta)$ should decrease on every iteration
- But if α is too small, gradient descent can be too slow

When choosing α, try a range of values: $\cdots$, 0.001, 0.003, 0.01, 0.03, 0.1, 0.3, 1, $\cdots$

2.7 Normal equation

Given X is the matrix containing features, with $x_0 = 1$ and y is a vector containing the results,

$$\theta = (X^T X)^{-1} X^T y$$

In details:

$$x^{(i)} = \begin{bmatrix} x_0^{(i)} \\ x_1^{(i)} \\ x_2^{(i)} \\ \vdots \\ x_n^{(i)} \end{bmatrix} \qquad X = \begin{bmatrix} \cdots \left(x^{(1)}\right)^T \cdots \\ \cdots \left(x^{(2)}\right)^T \cdots \\ \vdots \\ \cdots \left(x^{(m)}\right)^T \cdots \end{bmatrix}$$

X will have a dimension of $m \times (n+1)$

3 Logistic regression

Algorithm to classify a data set in different categories.

3.1 Hypothesis

$$h_\theta(x) = g(\theta^T x) \tag{3}$$

$$g(z) = \frac{1}{1 + e^{-z}} \tag{4}$$

$g(z)$ has the following property:

$$g(z) \geq 0.5 \leftrightarrow z \geq 0$$

3.2 Cost function

We want the cost function to be convex, so we cannot use the same function as for linear regression.

We define the cost function as follow:

$$J(\theta) = \frac{1}{m}\sum_{i=1}^{m}\mathrm{Cost}(h_\theta(x), y)$$

For linear regression, the $\mathrm{Cost}(h_\theta(x), y)$ was the squared error.

For logistic regression, we use

$$\mathrm{Cost}(h_\theta(x), y) = \begin{cases} -\log\left(h_\theta(x)\right) & \text{if } y = 1 \\ -\log\left(1 - h_\theta(x)\right) & \text{if } y = 0 \end{cases}$$

More intuitively, if we predicted with certainty $y = 0$ but it turned out that $y = 1$, we pay a very large cost.

On the opposite, if we predicted with certainty that $y = 0$ and it was correct, the cost is very close to 0.

The above function can be rewritten as follow:

$$\mathrm{Cost}(h_\theta(x), y) = -y\log\left(h_\theta(x)\right) - (1-y)\log\left(1 - h_\theta(x)\right)$$

we therefore get

$$J(\theta) = -\frac{1}{m}\left[\sum_{i=1}^{m} y^{(i)}\log\left(h_\theta(x^{(i)})\right) + (1-y^{(i)})\log\left(1 - h_\theta(x^{(i)})\right)\right]$$

we then need to compute $\min_\theta J(\theta)$, which can be achieved by using the gradient descent.

We compute the partial derivative as follow

$$\frac{\partial}{\partial\theta_j} = \frac{1}{m}\sum_{i=1}^{m}\left(h_\theta(x^{(i)}) - y^{(i)}\right)x_j^{(i)}$$

3.3 Optimizations

There are more optimized algorithms that gradient descent that can be used to compute the parameters. For example:

- Conjugate gradient
- BFGS
- L-BFGS

Advantages:

- No need to manually pick α
- Often fater than gradient descent

Disadvantages

- More complex

These can be used easily in octave as follow. Given the following example,

$$\theta = \begin{bmatrix} \theta_1 \\ \theta_1 \end{bmatrix} \tag{5}$$

$$J(\theta) = (\theta_1 - 5)^2 + (\theta_2 - 5)^2 \tag{6}$$

$$\frac{\partial}{\partial \theta_1} J(\theta) = 2(\theta_1 - 5) \tag{7}$$

$$\frac{\partial}{\partial \theta_2} J(\theta) = 2(\theta_2 - 5) \tag{8}$$

```
function [jVal, gradient] = costFunction(theta)
  jVal = (theta(1) - 5)^2 + (theta(1) - 5)^2;
  gradient = zeros(2, 1);
  gradient(1) = 2 * (theta(1) - 5);
  gradient(2) = 2 * (theta(2) - 5);
end

options = optimset('GradObj', 'on', 'MaxIter', 100);
initialTheta = zeros(2, 1);
[optTheta, functionVal, exitFlag] = fminunc(@costFunction, initialTheta, options);
```

3.4 Multiclass classification

We can use the one-vs-all algorithm.

Train a logistic regression classifier $h_\theta^{(i)}(x)$ for each class i to predict the probability that $y = i$
On a new input x, to make a prediction, pick the class i that maximizes

$$\max_i h_\theta^{(i)}(x)$$

4 Regularization

4.1 Overfitting

If we have too many features, the learned hypothesis may fit the training set very well ($J(\theta) = \frac{1}{2m}\sum_{i=1}^{m}(h_\theta(x^{(i)}) - y^{(i)})^2 \approx 0$), but fail to generalize to new examples.

4.1.1 Adressing overfitting

Options:

1. Reduce number of features
   - Manually select which features to keep
   - Model selection algorithm
2. Regularization
   - Keep all the features, but reduce the magnitude/values of parameters θ_j
   - Works well when we have a lot of features, each of which contributes a bit to predicting y

4.2 Cost function

Small values for parameters $\theta_0, \theta_1, \cdots, \theta_n$

- Simpler hypothesis
- Less prone to overfitting

To implement it, we can change the cost function as follow

$$J(\theta) = \frac{1}{2m}\left[\sum_{i=1}^{m}\left(h_\theta(x^{(i)}) - y^{(i)}\right)^2 + \lambda\sum_{i=1}^{m}\theta_j^2\right]$$

λ is called the regularization parameter.

4.3 Regularized linear regression

With

$$J(\theta) = \frac{1}{2m}\left[\sum_{i=1}^{m}\left(h_\theta(x^{(i)}) - y^{(i)}\right)^2 + \lambda\sum_{i=1}^{m}\theta_j^2\right]$$

we want to compute $\min_\theta J(\theta)$.
We have

$$\frac{\partial}{\partial\theta_j}J(\theta) = \frac{1}{m}\sum_{i=1}^{m}\left(h_\theta(x^{(i)}) - y^{(i)}\right)x_j^{(i)} + \frac{\lambda}{m}\theta_j$$

therefore we only need to separate the computation of θ_0 and θ_j $(j \geq 1)$ so we do not regularize θ_0.

4.3.1 Gradient descent

The gradient descent update for θ_j looks like

$$\begin{aligned}\theta_j &= \theta_j - \alpha\left[\frac{1}{m}\sum_{i=1}^{m}\left(h_\theta(x^{(i)}) - y^{(i)}\right)x_j^{(i)} + \frac{\lambda}{m}\theta_j\right] && (9)\\ &= \theta_j\left(1 - \alpha\frac{\lambda}{m}\right) - \alpha\frac{1}{m}\sum_{i=1}^{m}\left(h_\theta(x^{(i)}) - y^{(i)}\right)x_j^{(i)} && (10)\end{aligned}$$

We usually want $\left(1 - \alpha\frac{\lambda}{m}\right) < 1$

4.3.2 Normal equation

Given

$$X = \begin{bmatrix}\left(x^{(1)}\right)^T \\ \vdots \\ \left(x^{(m)}\right)^T\end{bmatrix} \qquad y = \begin{bmatrix}y^{(1)} \\ \vdots \\ y^{(m)}\end{bmatrix}$$

we are looking for $\min_\theta J(\theta)$
The normal equation in this case is given by

$$\theta = \left(X^T X + \lambda \begin{bmatrix} 0 & & & \\ & 1 & & \\ & & \ddots & \\ & & & 1 \end{bmatrix} \right)^{-1} X^T y$$

4.4 Regularized logistic regression

We update our cost function to look as follow:

$$J(\theta) = -\frac{1}{m} \left[\sum_{i=1}^{m} y^{(i)} \log \left(h_\theta(x^{(i)}) \right) + (1 - y^{(i)}) \log \left(1 - h_\theta(x^{(i)}) \right) \right] + \frac{\lambda}{2m} \sum_{j=1}^{n} \theta_j^2$$

To implement the gradient descent, we need to use the same steps as for linear regression.

5 Neural networks

Neural network is made of

1. An input layer
2. 0 or more hidden layers
3. An output layer

5.1 Example and computations

Given a neural network with 3 units in the input layer, and a single hidden layer, the computations are defined as follow:

$$a_1^{(2)} = g \left(\Theta_{10}^{(1)} x_0 + \Theta_{11}^{(1)} x_1 + \Theta_{12}^{(1)} x_2 + \Theta_{13}^{(1)} x_3 \right) \tag{11}$$
$$a_2^{(2)} = g \left(\Theta_{20}^{(1)} x_0 + \Theta_{21}^{(1)} x_1 + \Theta_{22}^{(1)} x_2 + \Theta_{23}^{(1)} x_3 \right) \tag{12}$$
$$a_3^{(2)} = g \left(\Theta_{30}^{(1)} x_0 + \Theta_{31}^{(1)} x_1 + \Theta_{32}^{(1)} x_2 + \Theta_{33}^{(1)} x_3 \right) \tag{13}$$
$$h_\Theta(x) = g \left(\Theta_{10}^{(2)} a_0^{(2)} + \Theta_{11}^{(2)} a_1^{(2)} + \Theta_{12}^{(2)} a_2^{(2)} + \Theta_{13}^{(2)} a_3^{(2)} \right) \tag{14}$$

We define

$$\Theta_{10}^{(1)} x_0 + \Theta_{11}^{(1)} x_1 + \Theta_{12}^{(1)} x_2 + \Theta_{13}^{(1)} x_3 = z_1^{(2)}$$

so that

$$a_1^{(2)} = g \left(z_1^{(2)} \right)$$

Given the above example, we have

$$x = a^{(1)} = \begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \end{bmatrix} \qquad z^{(2)} = \begin{bmatrix} z_1^{(2)} \\ z_2^{(2)} \\ z_3^{(3)} \end{bmatrix}$$

and we can use the following computations.

$$z^{(2)} = \Theta^{(1)} a^{(1)} \tag{15}$$

$$a^{(2)} = g\left(z^{(2)}\right) \tag{16}$$

To compute the next layer (output layer for this example), add $a_0^{(2)} = 1$, and repeat:

$$z^{(3)} = \Theta^{(2)} a^{(2)} \tag{17}$$

$$h_\Theta(x) = a^{(3)} = g\left(z^{(3)}\right) \tag{18}$$

5.2 Cost function

- $\left(x^{(1),y^{(1)}}\right), \left(x^{(2),y^{(2)}}\right), \cdots, \left(x^{(m),y^{(m)}}\right)$: training set
- L: total number of layers in network
- s_l: number of units (not couting bias unit) in layer l

$$J(\Theta) = -\frac{1}{m}\left[\sum_{i=1}^{m}\sum_{k=1}^{K} y_k^{(i)} \log\left(h_\Theta(x^{(i)})\right)_k + (1 - y_k^{(i)}) \log\left(1 - (h_\Theta(x^{(i)}))_k\right)\right] \tag{19}$$

$$+ \frac{\lambda}{2m} \sum_{l=1}^{L-1}\sum_{i=1}^{s_l}\sum_{j=1}^{s_l+1} \left(\Theta_{ji}^{(l)}\right) \tag{20}$$

5.3 Back propagation algorithm

$\delta_j^{(l)}$ is the "error" of node j in layer l.
For each ouput unit (here layer $L = 4$)

$$\delta_j^{(4)} = a_j^{(4)} - y_j \tag{21}$$

$$\delta_j^{(3)} = \left(\Theta^{(3)}\right)^T \delta^{(4)}. \star g'\left(z^{(3)}\right) \tag{22}$$

$$\delta_j^{(2)} = \left(\Theta^{(2)}\right)^T \delta^{(3)}. \star g'\left(z^{(2)}\right) \tag{23}$$

where

$$g'\left(z^{(3)}\right) = a^{(3)}. \star \left(1 - a^{(3)}\right)$$

and

$$\frac{\partial}{\partial \Theta_{ij}^{(l)}} J(\Theta) = a_j^{(l)} \delta_i^{(l+1)} \qquad \text{if } \lambda = 0$$

5.3.1 Algorithm

- Set $\Delta_{ij}^{(l)} = 0$ (for all l, i, j)
- For $i = 1$ to m
  - Set $a^{(1)} = x^{(i)}$

- Perform forward propagation to compute $a^{(l)}$ for $l = 2, 3, \cdots, L$
  - Using $y^{(i)}$, compute $\delta^{(L)} = a^{(L)} - y^{(i)}$
  - Compute $\delta^{(L-1)}, \delta^{(L-2)}, \cdots, \delta^{(2)}$
    $\Delta_{ij}^{(l)} = \Delta_{ij}^{(l)} + a_j^{(l)} \delta_i^{(l+1)}$

- $D_{ij}^{(i)} = \frac{1}{m}\Delta_{ij}^{(l)} + \lambda\Theta_{ij}^{(l)} \quad$ if $j \neq 0$
- $D_{ij}^{(i)} = \frac{1}{m}\Delta_{ij}^{(l)} \quad$ if $j = 0$

6 Evaluating algorithms

Precision:

$$\frac{\#\text{true positives}}{\#\text{true positives + false positives}}$$

Recall:

$$\frac{\#\text{true positives}}{\#\text{true positives + false negatives}}$$

F_1 score:

$$2\frac{PR}{P+R}$$