Logistic Regression Problems

Huy Tr.

October 3rd, 2016

In the previous post, we learned **Linear Regression** to predict a continuous value by using a linear function as a base line that fit to the data distribution. Now we take another problem: **classification** and let's see what is the different between this two types.

1 The Rune Checking Problem

So, we have the problem in a DotA 2 match:

Sven joined the battle as an member of the Radiant, who has around 1500 MMR, so he start with a *450* gold boot and took the middle lane, which should belongs to *Pudge*.

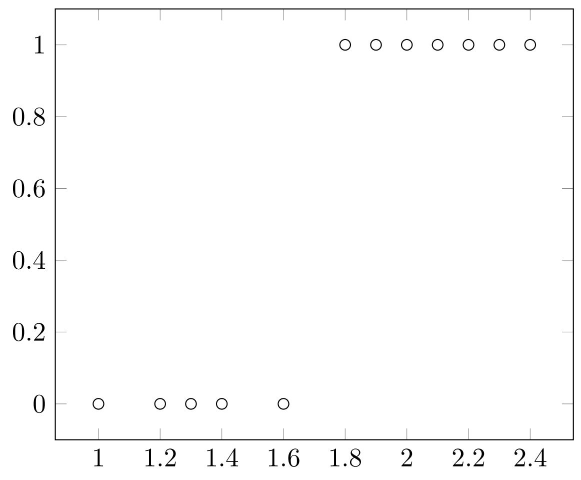
The nightmare didn't stop here, during the match, he left his lane to check for runes in the river every *0.2* minutes!

Here is the check log that Sven made during his early game:

Table 1: Sven's Rune Checklog

Check Time	Found Rune
1.0	No
1.2	No
1.3	No
1.4	No
1.6	No
1.8	Yes
1.9	Yes
2.0	Yes
2.1	Yes
2.2	Yes
2.3	Yes
2.4	Yes

Let's take a look at the plot of this check log:



In order to keep the match from being ruined by Sven's ignorance, we need to help him figure out when is the best time to check for runes, so he don't waste too much time travelling to the river again.

The idea is to write a computer program that Sven can use at any time, this program will take the current match time and tell him now is a **good time** or **bad time** to go check for the runes.

2 The Classification

The problem as described above should be called **classification**, since we only have a limit number of answers and the predict function should always return one of the answers.

For example, with **binary classification**, the y values should only takes 0 (**negative class**) and 1 (**positive class**).

Classification can also be found in *spam detection*, *document classification*, *object recognition*, *etc...*

3 Logistic Regression

Logistic Regression is the approach to solve classification by ignoring the fact that y is discrete-valued and use the old good **linear regression** algorithm to predict y by x. But we need to change our **hypothesis function** $h_\theta(x)$:

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

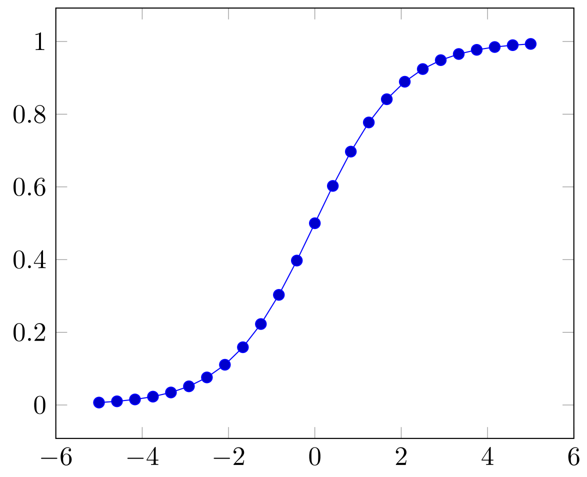
Which:

$$g(z) = \frac{1}{1+e^{-z}} \tag{2}$$

Note that:

$$z = \theta_0 + \theta_1 x \tag{3}$$

This called **logistic function** or **sigmoid function**



Notice that $g(z)$ tends towards 1 as $z \to \infty$ and towards 0 as $z \to -\infty$, and $g(z)$ (or $h(x)$ as well) is always bounded between 0 and 1.

We also have the **derivative of the sigmoid function** as follow:

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

4 Optimizing with Gradient Descent

We can use the same **Gradient Descent Algorithm** as we used in **Linear Regression** to solve **Logistic Regression** problem.

For example, this is the **Stochastic Gradient Descent** algorithm:

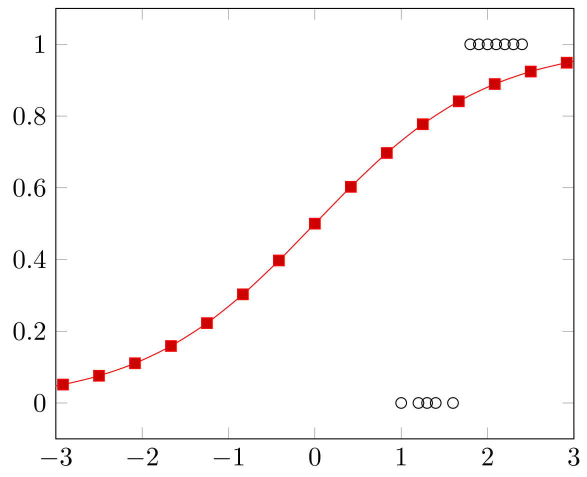
```
loop
    for i $\to$ m do
        for every j do $\theta_j = \theta_j + \alpha(h_\theta(x^{(i)}) - y^{(i)})x_j$
        end for
    end for
end loop
```

5 Solving Rune Checking Problem with Logistic Regression

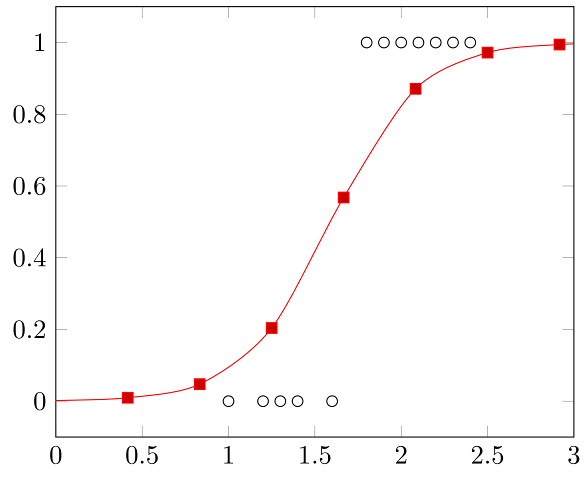
Now, let's get back to our **Rune Checking** problem and see how can we use **Logistic Regression** to solve it.

Starting with our data, we define a **hypothesis function** as a **sigmoid function**

$$h(x) = \frac{1}{1 + e^{-(\theta_0 + \theta_1 x)}} \tag{5}$$



What we need to do is fitting this function to our data, in the other words, we optimizing θ_0 and θ_1 to make the function become like the following plot:

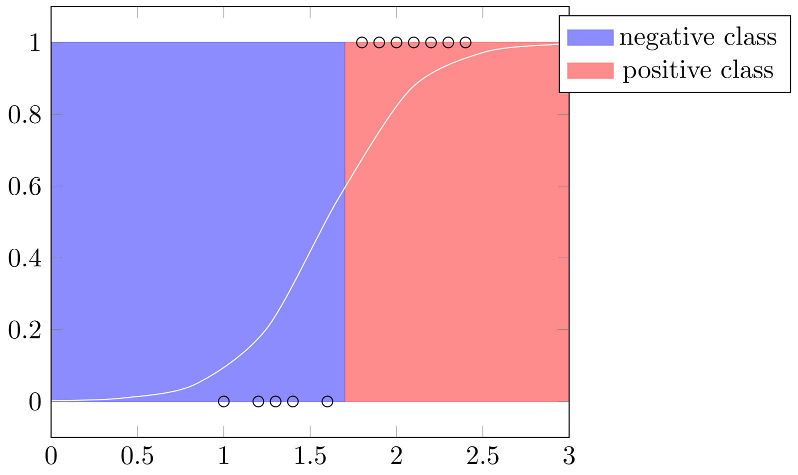


By running **Gradient Descent**, we found that $\theta_0 = -6.266031$ and $\theta_1 = 3.923966$. This mean our **hypothesis function** now become:

$$h(x) = \frac{1}{1 + e^{-(-6.266031 + 3.923966x)}} \tag{6}$$

Later on, if you feed some x value to this function, you will get the result as number between $0 \rightarrow 1$.

At the point this number reached 0.5, it called **decision boundary**, everything on the left of the **decision boundary** will be the **negative class** and on the right will be the **possitive class**



In the **Rune Checking** problem, if the given time is in the **positive** zone, so it's a **good time** to go for rune, otherwise, it's a **bad time**, don't go anywhere.