# Introduction to machine learning-Handout: logistic regression

Yue Guan

May 8, 2020

## 1 Logistic Regression

We first check out the logistic regression for **binary classification**. The idea about logistic regresssion is that given the input vector $\boldsymbol{x}$, we want to assign a **probability of the model output** to $f(\boldsymbol{x})$ to represent the chance of the input belonging to positive class ( the probability of the negative class will be $1 - f(\boldsymbol{x})$), i.e. $y = 1$. To summerize the input vector $\boldsymbol{x}$, we can use a linear model $\boldsymbol{w}^T\boldsymbol{x} + b$ [1].

A first though of converting the output of the linear model to a value in $[0, 1]$ would be just by thresholding using the unit step function , as shown in (1).

$$z(\boldsymbol{x}) = \begin{cases} 1 & h = \boldsymbol{w}^T\boldsymbol{x} + b \geq 0 \\ 0 & h = \boldsymbol{w}^T\boldsymbol{x} + b < 0 \end{cases} \tag{1}$$

However, a soft version should be preferred as it will make gradient based learning easier. Thus we use a soft version of step function, which is the Sigmoid function $\sigma(\cdot)$, as shown in (2).

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

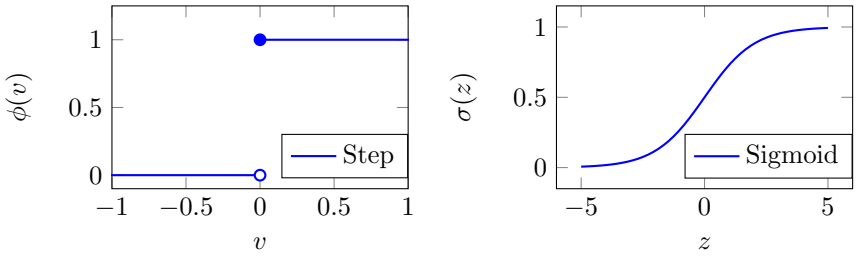
The plot for these two functions are shown side by side in Fig 1

Now given the input $\boldsymbol{x}$, we will using the following equation (3) to crunch the input to an output representing the probability of positive sample.

$$f(\boldsymbol{x}) = \sigma(\boldsymbol{w}^T\boldsymbol{x} + b) \tag{3}$$

We also want to define a loss function as the learning goal. The choice of the loss function should be better to:

1. align with the learning goal as close as possible,
2. be friendly for the learning method.




(a) step fucntion. (b) sigmoid function.

Figure 1: Step function and its soft version, i.e. Sigmoid function.

In our binary classification problem, we choose to use binary cross entropy loss (4):

$$L = \sum_{i=0}^{N} -y_i \ln \frac{1}{1+e^{-(\boldsymbol{w}^T \boldsymbol{x}_i + b)}} - (1-y_i) \ln(1 - \frac{1}{1+e^{-(\boldsymbol{w}^T \boldsymbol{x}_i + b)}}) \tag{4}$$

, remember the entropy is the average negative logarithm of the probability.

## 2 Stochastic gradient descent

In this part, we want to minimize the loss fucntion (4) with the help of stochastic gradient descent (SGD). We will go through gradient descent (GD) first and then introduce the SGD.

We can get the loss gradient with respect to model parameters by the chain rule (5) (use $\boldsymbol{w}$ as an example.).

$$\frac{\partial L_i}{\partial \boldsymbol{w}} = \frac{\partial L_i}{\partial z_i} \frac{\partial z_i}{\boldsymbol{w}} \tag{5}$$

. For the Sigmoid function, we have (6)

$$\begin{aligned} \frac{\partial L}{\partial z_i} &= \frac{-y_i}{1+e^{z_i}} + \frac{1-y_i}{1+e^{-z_i}} \\ \frac{\partial z_i}{\partial \boldsymbol{w}} &= \boldsymbol{x}_i \end{aligned} \tag{6}$$

.

Note that the gradient with respect to $\boldsymbol{w}$ is a vector and each element corresponds to one input sample. The convention to update the weight is taking the empirical average of the whole dataset for GD:

$$\boldsymbol{w}_{t+1} = \boldsymbol{w}_t - \alpha \frac{1}{N} \sum_{i}^{N} \frac{\partial L}{\partial z_i} \tag{7}$$

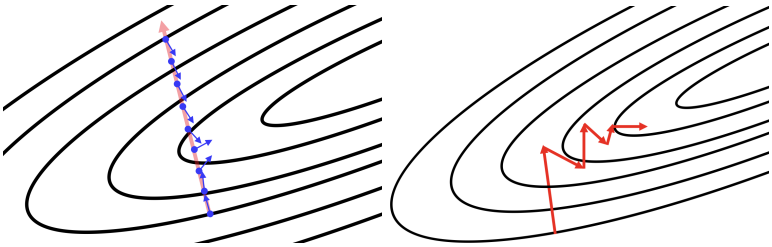


Figure 2: Gradient and GD trace

When the dataset becomes large, a practical concern is that this whole batch GD process will become infeasible because the data cannot fit into the machine's memory. SGD goes to the other end where it only randomly sample one instance with replacement a time and use this **single sample** to calculate the gradient and update the parameters. In next epoch, another sample is used to get the gradient and update the parameter. Since there is only one sample used at a time. there is likely no memory issue.

However, updating the parameter by going through the data one by one may not be efficient to cover the distribution of the dataset. We can sample a small batch of data to enjoy the benefit of SGD while stay away from the memory issue. This GD approach with a small batch samples is called **mini-batch SGD**, which is is the de facto approach to train a deep learning model.

# 3 Input preprocessing

We usually need to preprocess the raw datasets presented by the learning problems. One common practice is to normalize the data into $[0, 1]$ interval or normalize the data to have zero mean and unit variance. This is often necessary steps to try during training. An example what may happened is shown in Fig 2.

The fact is that most algorithms just treat each input feature equally and use Euclidian distance between samples to measure the distances between them. This leads features with large numeric value shadows other features and make the training process difficult.

# 4 Balancing samples

The class population in the training dataset needs to be balanced. For example, there is one dataset contains 100 samples, among which, 99 samples are the positive cases and only 1 sample is the negative case. Then a classifier always returns positive will still achieve 99% accuracy. To overcome the problem, there is oversampling/undersamping method as shown in Fig 3. If there is too much data from one class, then we can randomly sample a subset of the data from that class to make the samples from each class is almost the same.

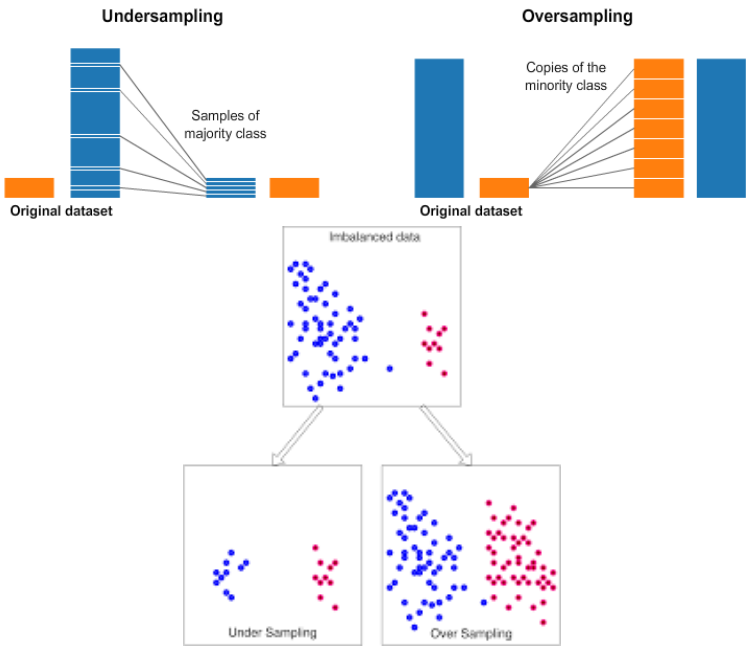



Figure 3: Unbalanced data. Plots are from web.

# References

[1] C. Bishop. *Pattern Recognition and Machine Learning*. 2006.