LOGISTIC REGRESSION AND GRADIENT DESCENT

MACHINE LEARNING LECTURE NOTES

Yue Ning
Stevens Institute of Technology
yue.ning@stevens.edu

1 Introduction

Logistic Regression (LR) [2] is a *classification* algorithm used in machine learning to predict discrete outcomes — that is, to determine which category a sample belongs to. It is a discriminative model, meaning it directly learns a function to model the probability of the label y given the features $\mathbf{x}$, $P(y = k \mid \mathbf{x})$, where $k \in \{1, ..., K\}$, rather than modeling the distribution of the features themselves.

Given a set of training samples and their corresponding labels:

$$X = \{\mathbf{x}_1, \ldots, \mathbf{x}_N \mid \mathbf{x}_i \in \mathbb{R}^d\}, \quad Y = \{y_1, \ldots, y_N\},$$

our goal is to estimate $P(y = k \mid \mathbf{x})$ for each class $k \in \{1, \ldots, K\}$. Once trained, the model predicts the label of new samples based on this probability. The sample $\mathbf{x}$ belongs to class k if $P(y = k \mid \mathbf{x})$ has the largest value.

We use *sample* and *example* interchangeably.

1.1 Binary Logistic Regression ($K = 2$)

For a given sample $\mathbf{x}_i$, the output label is a binary value $y_i \in \{0, 1\}$ (positive when $y_i = 1$ and negative when $y_i = 0$). Assuming $\mathbf{w}$ is the model coefficient (parameter), binary LR estimates the probability of $\mathbf{x}$ belonging to the positive class $P(y = 1|\mathbf{x})$ using the Sigmoid function:

$$P(y = 1|\mathbf{x}) = \sigma(\mathbf{w}^\top \mathbf{x}) = \frac{\exp(\mathbf{w}^\top \mathbf{x})}{1 - \exp(\mathbf{w}^\top \mathbf{x})} = \frac{1}{1 - \exp(-\mathbf{w}^\top \mathbf{x})}. \tag{1}$$

Thus, the probability of a given example belonging to the negative class is:

$$P(y = 0|\mathbf{x}) = 1 - \sigma(\mathbf{w}^\top \mathbf{x}). \tag{2}$$

Sigmoid function transforms any real value into the range of $(0, 1)$ as shown in figure 1.

1.2 Decision Boundary

The current prediction function returns a probability score between 0 and 1. In order to map this to a discrete class (true/false, positive/negative), we select a threshold value or tipping point above which we will classify values into class 1 (positive) and below which we classify values into class 0 (negative).

$$\hat{y} = \begin{cases} 1 & \text{if } P(y = 1|\mathbf{x}) \geq 0.5 \\ 0 & \text{else} \end{cases}$$

So the decision boundary is $P(y = 1|\mathbf{x}) = 0.5$ which means $\sigma(\mathbf{w}^\top \mathbf{x}) = \frac{1}{1-\exp(-\mathbf{w}^\top \mathbf{x})} = 0.5$. When we transform this equation, the decision boundary becomes $\mathbf{w}^\top \mathbf{x} = 0$. This is linear with respect to $\mathbf{x}$.

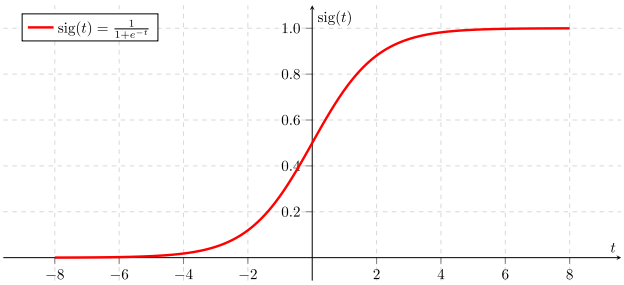

Figure 1: Sigmoid function

2 Multi-class Logistic Regression ($K > 2$)

In multi-class classification, each given sample belongs to only one of K classes: i.e., $y \in \{1, 2, ...K\}$. In this case, we model the probability of a given example belonging to class k as a softmax function [5]:

$$P(y = k|\mathbf{x}) = \text{softmax}(\mathbf{w}_k^\top \mathbf{x}) = \frac{\exp(\mathbf{w}_k^\top \mathbf{x})}{\sum_{c=1}^{K} \exp(\mathbf{w}_c^\top \mathbf{x})} \tag{3}$$

where $\mathbf{w}_k \in \mathbb{R}^d$ is the model parameter for class k. Consider this prediction function to be two steps:

1. Take the inner product of $\mathbf{w}_k$ and $\mathbf{x}$, compute f_k for each $k = 1, ..., K$:

$$f_k = \mathbf{w}_k^\top \mathbf{x} = \sum_{j=1}^{d} w_{kj} x_j \tag{4}$$

2. Apply softmax function to get normalized probability:

$$P(y = k|\mathbf{x}) = \frac{\exp f_k}{\sum_{c=1}^{K} \exp(f_c)} \tag{5}$$

Given the property of softmax, the sum of the probabilities over all K classes is 1:

$$P(y = 1|\mathbf{x}) + \cdots + P(y = K|\mathbf{x}) = \frac{\exp f_1}{\sum_{c=1}^{K} \exp(f_c)} + \cdots + \frac{\exp f_K}{\sum_{c=1}^{K} \exp(f_c)} = \frac{\exp f_1 + \cdots + \exp f_K}{\sum_{c=1}^{K} \exp(f_c)} = 1 \tag{6}$$

3 Optimization

In the previous section, we do not use indices for examples because all the equations are applied to one example. In the optimization process, we define a loss function (or objective function) over all training examples to optimize (maximize or minimize). In multi-class LR, we optimize the LR model by maximizing the likelihood which is the probability of correct class y given feature vector $\mathbf{x}$ of all training examples. In the below likelihood function, i is the index for examples and k is the index for classes:

$$\prod_{i=}^{N} \prod_{k=1}^{K} P(y_i = k|\mathbf{x}_i)^{y_{ik}}. \tag{7}$$

We use $y_{ik} = 1$ to denote $y_i = k$ which means the current sample $\mathbf{x}_i$ belongs to class k.

Or equivalently, we can minimize the negative log probability of that class:

$$-\log P(y_i = k|\mathbf{x})^{y_{ik}} = -y_{ik} \log \big(\frac{\exp f_k}{\sum_{c=1}^{K} \exp f_c}\big). \tag{8}$$

The total loss over all training examples becomes:

$$\mathcal{L}_{\text{multi}} = -\frac{1}{N}\sum_{i=1}^{N}\sum_{k=1}^{K} y_{ik} \log\big(\frac{\exp f_k}{\sum_{c=1}^{K}\exp f_c}\big). \tag{9}$$

For binary classification where $y_i = \{0, 1\}$, the loss can be simplified into:

$$\mathcal{L}_{\text{binary}} = -\frac{1}{N}\sum_{i=1}^{N}\Big(y_i \log\sigma(\mathbf{w}^\top\mathbf{x}) + (1-y_i)\log(1-\sigma(\mathbf{w}^\top\mathbf{x}))\Big). \tag{10}$$

This loss is also recognized as cross-entropy loss. In information theory, cross-entropy measures the distance between two probability distributions:

$$H(\mathbf{P}, \mathbf{Q}) = \sum_{i=1}^{D} p_i \log(q_i), \tag{11}$$

where $P = (p_1, ..., p_D)$ and $Q = (q_1, ..., q_D)$ are two probability distributions.

In multi-class LR, the true distribution refers to one-hot encoding of the label. For label k when k is the true label for a given sample, the one-hot encoding is defined as a vector whose element is 1 at index k, and 0 everywhere else.

The loss of one training example $\mathbf{x}$ in class k defined as cross-entropy is given by:

$$\begin{aligned} L(\mathbf{x}, y, \mathbf{w}) &= H(y, \hat{y}) \\ &= -\sum_{c} y_c \log\hat{y}_c \\ &= -\log\hat{y}_k \\ &= -\log\frac{\exp\mathbf{w}_k^\top\mathbf{x}}{\sum_{c=1}^{K}\exp\mathbf{w}_c^\top\mathbf{x}} \end{aligned} \tag{12}$$

3.1 Estimation

We apply *Gradient Descent* [1, 6] to find the best $\mathbf{w}$ that minimizes the training loss $\mathcal{L}$ defined in the last section. Gradient descent is a first-order optimization algorithm used to minimize a loss function by iteratively moving in the direction of steepest descent as defined by the negative of the gradient [4]. In this procedure, we first initialize the model parameter $\mathbf{w}$ and then iterate the training dataset to update the parameter. At each epoch, we update $\mathbf{w}$ by the following policy:

(Batch) Gradient Descent

- Initialize $\mathbf{w}$
- For t in $1, ..., T$ (epoches):
  - Calculate the total training loss J and its gradient with respect to $\mathbf{w}$: $\nabla_{\mathbf{w}}\mathcal{L}$
  - Update $\mathbf{w}$ by $\mathbf{w}^{t+1} = \mathbf{w}^t - \eta\nabla_{\mathbf{w}}\mathcal{L}$

where η is the learning rate (or step size) and $\nabla_{\mathbf{w}}\mathcal{L}$ is the gradient of $\mathcal{L}$ with respect to $\mathbf{w}$. For a binary LR, the derivative of the loss function with respect to $\mathbf{w}$ is:

$$\nabla_{\mathbf{w}}\mathcal{L}_{\text{binary}} = \frac{1}{N}\sum_{n=1}^{N}(P(y=1|\mathbf{x}) - y_n)\mathbf{x}_n = \frac{1}{N}\sum_{n=1}^{N}(\sigma(\mathbf{w}^\top\mathbf{x}_n) - y_n)\mathbf{x}_n. \tag{13}$$

Note that the derivative of a sigmoid function is $\frac{d\sigma}{da} = \sigma(a)(1-\sigma(a))$.

Given multiple classes, we calculate the gradient of the loss function with respect to each $\mathbf{w}_k$ where $k = 1, ..., K$ (if you are interested in how to derive the gradient of softmax, see this reference [3]):

$$\frac{\partial}{\partial \mathbf{w}_k}\mathcal{L}_{\text{multi}} = \frac{\partial}{\partial \mathbf{w}_k}\Big(-\frac{1}{N}\sum_{i=1}^{N}\sum_{k=1}^{K} y_{ik}\log\frac{\exp f_k}{\sum_{c=1}^{K}\exp f_c}\Big)$$
$$= \frac{1}{N}\sum_{i=1}^{N}\Big(P(y_i = k|\mathbf{x}_i) - y_{ik}\Big)\mathbf{x}_i \tag{14}$$

One problem with gradient descent algorithm is that it can be very slow because $\mathcal{L}$ is the sum of each example loss. Thus, one efficient solution is *Stochastic Gradient Descent (SGD)*. It updates the gradient based on one training example:

Stochastic Gradient Descent (SGD)

- Initialize $\mathbf{w}$
- For t in $1, ..., T$ (epoches):
  - Randomly shuffle the training data
  - For $(\mathbf{x}_n, y_n)$ in the training data:
    - Calculate the loss on the current example $J(\mathbf{x}_n, y_n)$ and its gradient with respect to $\mathbf{w}$: $\nabla_{\mathbf{w}} J(\mathbf{x}_n, y_n)$
    - Update $\mathbf{w}$: $\mathbf{w}^{t+1} = \mathbf{w}^t - \eta\nabla_{\mathbf{w}} J(\mathbf{x}_n, y_n)$
    - Convergence test and early stop

SGD provides an online update solution with new examples, but it has a high variance that causes the objective function to fluctuate heavily. A trade-off solution is *Mini-batch Gradient Descent*:

Mini-batch Gradient Descent

- Initialize $\mathbf{w}$
- For t in $1, ..., T$ (epoches):
  - Randomly sample a mini-batch from the training data
    - Calculate the loss on the mini-batch $J(\text{mini-batch})$ and its gradient with respect to $\mathbf{w}$: $\nabla_{\mathbf{w}} J(\text{mini-batch})$
    - Update $\mathbf{w}$: $\mathbf{w}^{t+1} = \mathbf{w}^t - \eta\nabla_{\mathbf{w}} J(\text{mini-batch})$
    - Convergence test and early stop

It reduces the variance of the parameter updates, which can lead to more stable convergence. It can also make use of highly optimized matrix optimizations (common in state-of-the-art deep learning libraries) that make computing the gradient w.r.t. a mini-batch very efficient.

4 Regularization

Regularization is a technique to avoid overfitting. In multi-class LR models, we can add a L2 norm regularization in the loss function and it then becomes:

$$\mathcal{L}_{\text{multi}} = -\frac{1}{N}\sum_{i=1}^{N}\sum_{k=1}^{K} y_{ik}\log\big(\frac{\exp f_k}{\sum_{c=1}^{K}\exp f_c}\big) + \lambda\sum_{k=1}^{K}||\mathbf{w}_k||^2, \tag{15}$$
$$= -\frac{1}{N}\sum_{i=1}^{N}\sum_{k=1}^{K} y_{ik}\log\big(\frac{\exp f_k}{\sum_{c=1}^{K}\exp f_c}\big) + \lambda\sum_{k=1}^{K}\sum_{j=1}^{d} w_{kj}^2. \tag{16}$$

The hyperparameter λ controls the strength of the regularization. We can use a validation set to choose the best value for λ. Similarly, the binary loss of LR becomes:

$$\mathcal{L}_{\text{binary}} = -\frac{1}{N}\sum_{i=1}^{N}\Big(y_i \log \sigma(\mathbf{w}^\top \mathbf{x}) + (1-y_i)\log(1-\sigma(\mathbf{w}^\top \mathbf{x}))\Big) + \lambda||\mathbf{w}||^2. \tag{17}$$

When we have a regularization term in the loss function, the gradient w.r.t $\mathbf{w}$ will need to be updated.

References

[1] A.-L. Cauchy. Méthode générale pour la résolution des systèmes d'équations simultanées. *C. R. Acad. Sci. Paris*, 25:536–538, 1847.

[2] J. Cramer. The early origins of the logit model. *Studies in History and Philosophy of Science Part C: Studies in History and Philosophy of Biological and Biomedical Sciences*, 35(4):613 – 626, 2004.

[3] S. Ji and Y. Xie. Logistic regression: From binary to multi-class. `http://people.tamu.edu/~sji/classes/LR.pdf`.

[4] A. Ng. CS229 lecture notes. `https://cs229.stanford.edu/lectures-spring2022/main_notes.pdf`.

[5] S. O. Ogun. You don't really know softmax. `https://ogunlao.github.io/2020/04/26/you_dont_really_know_softmax.html`.

[6] S. Ruder. An overview of gradient descent optimization algorithms. `http://arxiv.org/abs/1609.04747`, 2016.