1 Logistic Regression

In this exercise we focus on the classification problem. This is just like the regression problem, except that the values y we now want to predict take on only a small number of discrete values. For now, we will focus on the **binary** classification problem in which y can take on only two values, 0 and 1. For instance, if we are trying to build a spam classifier for email, then $\mathbf{x}$ may be some features of a piece of email, and y may be 1 if it is a piece of spam mail, and 0 otherwise. The value 0 is also called the negative class, whereas the value 1 the positive class. Additionally, they are sometimes denoted by the symbols '–' and '+'. Given a training example $\mathbf{x}^{(i)}$, the corresponding $y^{(i)}$ is also called the **label** of it.

1.1 Theory

We could approach the classification problem ignoring the fact that y is discrete-valued, and use our old linear regression algorithm to try to predict $\underline{x}$ given y. However, it is easy to construct examples where this method performs very poorly. Intuitively, it also does not make sense for $h(\underline{x})$ to take values larger than 1 or smaller than 0 when we know that $y \in [0, 1]$. To fix this, let change the form for our hypotheses $h(\underline{x})$. We will choose

$$h(\underline{x}) = g\left(\underline{\theta}^T \underline{x}\right) = \frac{1}{1 + e^{-\underline{\theta}^T \underline{x}}},$$

where

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

is called the **logistic** (or the **sigmoid**) function. Notice that $g(z)$ tends towards 1 as $z \to \infty$, and $g(z)$ tends towards 0 as $z \to -\infty$. Moreover, $g(z)$, and hence also $h(\underline{x})$, is always bounded between 0 and 1. As before, we are keeping the convention of letting $x_0 = 1$, so that $\underline{\theta}^T \underline{x} = \theta_0 + \sum_{j=1}^{m} \theta_j x_j$.

So, given the logistic regression model, how do we fit $\underline{\theta}$ for it? Following how we saw least squares regression could be derived as the maximum likelihood estimator under a set of assumptions, let endow our classification model with a set of probabilistic assumptions, and then fit the parameters via maximum likelihood. Let us assume that $P(y = 1|\underline{x}; \underline{\theta}) = h(\underline{x})$ and

$P(y = 0|\underline{x}; \underline{\theta}) = 1 - h(\underline{x})$. Note that this can be written more compactly as

$$P(y|\underline{x}; \underline{\theta}) = [h(\underline{x})]^y \times [1 - h(\underline{x})]^{1-y}.$$

Assuming that we have n training examples, we can then write down the likelihood of the parameters as

$$\begin{aligned}\mathcal{L}(\underline{\theta}) &= P(\underline{y}|\mathbf{X}; \underline{\theta}) \\ &= \prod_{i=1}^{n} P\left(y_i|\underline{x}_i; \underline{\theta}\right) \\ &= \prod_{i=1}^{n} \left[h\left(\underline{x}_i\right)\right]^{y_i} \times \left[1 - h\left(\underline{x}_i\right)\right]^{1-y_i}\end{aligned}$$

The goal is to maximize the log likelihood

$$\log \mathcal{L}(\underline{\theta}) = \sum_{i=1}^{n} \left(y_i \log h\left(\underline{x}_i\right) + (1 - y_i) \log\left(1 - h(\underline{x}_i)\right)\right), \tag{2}$$

or, alternatively, to minimize the cost function $\mathcal{J}(\underline{\theta})$ is defined as

$$\mathcal{J}(\underline{\theta}) = -\frac{1}{n} \log \mathcal{L}(\underline{\theta}) \tag{3}$$

How do we minimize the cost function? Similar to our derivation in the case of linear regression, we can use Gradient Descent.

The gradient of the cost function $\mathcal{J}(\underline{\theta})$ is given by

$$\frac{\partial \mathcal{J}(\underline{\theta})}{\partial \theta_j} = \frac{1}{n} \sum_{i=1}^{n} \left[h\left(\underline{x}_i\right) - y_i\right] x_{ij}, \;\; j = 1, 2, \ldots, m. \tag{4}$$

and the corresponding matrix form

$$\frac{\partial \mathcal{J}(\underline{\theta})}{\partial \underline{\theta}} = \frac{1}{n} \mathbf{X}^T [h(\mathbf{X}) - \underline{y}] \tag{5}$$

then, the gradient descent algorithm updates the parameters using

$$\underline{\theta} = \underline{\theta} - \frac{\partial \mathcal{J}(\underline{\theta})}{\partial \underline{\theta}} \tag{6}$$

1.2 Python Exercise

In this part of the exercise, you will implement the logistic regression to predict if a person has cancer or not. We employ the existing dataset from `sklearn` named `breast cancer wisconsin dataset`. This dataset contains in total 569 examples, among them 212 examples are labelled as malignant (M or 0) and 357 examples are marked as benign (B or 1). Each example is a vector of 30 dimensions. The part of loading and pre-processing have been done for you as we re-use the code from the previous exercises.

Step 1: Implement the sigmoid, cost and gradient functions. Similar to the previous exercises, you have to implement the functions `compute_cost` and `compute_gradient`. Also, you need to write code to calculate the probability of having cancer, namely implement the `sigmoid` function. At the end of this step, you will use the function `approximate_gradient` to verify if your implementation of the `compute_gradient` function is correct.

Step 2: Update model parameters using mini-batch gradient descent. In this step, you will need to write your code to update the model's parameters using the function `compute_gradient` that you have implemented.

Step 3: Predict the probabilities of having caner and drawing the confusion matrix. In this step, you use the trained model to predict the probabilities of having caner using the measurements from the test set. You will need to call function `sigmoid` you have implemented before. Moreover, you will need to evaluate the performance of your model using accuracy, which is the percentage of correctly classified examples over the total number of examples in the test set. Then, you will draw a confusion matrix illustrating your classification result.