Singapore University of Technology and Design
Information Systems Technology and Design
50.007 Machine Learning

Logistic Regression & Gradient Descent

---

1 Logistic Regression

We shall now look at a third linear model called *logistic regression* that outputs a probability, i.e., a value between 0 and 1. This new model is similar to regression in that the ouput is real, and is similar to classification in that the output is bounded. Logistic regression has wide application in practice. For example, we would like to predict the occurrence of diabetes based on an individual's blood pressure, height, weight, age, and other features. Clearly, we cannot predict the occurrence of diabetes with any certainty, but we may be able to predict how likely it is to occur given these features. Thus, a probabilistic output would be appropriate than a binary decision. The closer the output is to 1, the more likely the person would have diabetes.

In logistic regression, the output is given by the formula

$$h(\mathbf{x}) = \frac{\exp(\mathbf{w} \cdot \mathbf{x})}{1 + \exp(\mathbf{w} \cdot \mathbf{x})} \ .$$

The *logistic function* $\theta(s) = \frac{\exp(s)}{1+\exp(s)}$ varies continuously between 0 and 1, and its output can be interpreted as a probability for a binary event, e.g., diabetes (+1) or no diabetes (-1). The logistic function is also known as a *sigmoid* function because its shape looks like the letter `s` (Figure 1).

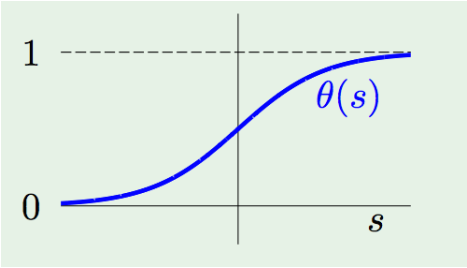

Figure 1: Sigmoid function.

2 Evaluation Function

The target output that logistic regression is trying to learn is a probability (e.g., of a patient being at risk of having diabetes) that depends on the input $\mathbf{x}$ (e.g., the physiological features of the patient). More formally, the target function is $f(\mathbf{x}) = P[y = +1|\mathbf{x}]$. However, the data does not provide the value of f explicitly, and only gives samples generated by this probability, e.g., patients who had diabetes (+1) and patients who did not (-1). Thus, to learn from such data, we need to define a proper evaluation function (objective function) that gauges how close a given hypothesis h is to f in terms of these ± 1 examples.

The evaluation function used in logistic regression is based on the *likelihood* of getting output $y = f(\mathbf{x})$ if the target distribution $P(y|\mathbf{x})$ was captured by our hypothesis $h(\mathbf{x})$. This likelihood is given by

$$P(y|\mathbf{x}) = \begin{cases} h(\mathbf{x}) & \text{for } y{=}{+}1, \\ 1 - h(\mathbf{x}) & \text{for } y{=}{-}1. \end{cases}$$

Substituting $h(\mathbf{x}) = \theta(\mathbf{w} \cdot \mathbf{x})$ in the above equation, and using $\theta(-s) = 1 - \theta(s)$, we get

$$P(y|\mathbf{x}) = \theta(y\, \mathbf{w}{\cdot}\mathbf{x}).$$

Since the training examples $(\mathbf{x}^1, y^1), \ldots, (\mathbf{x}^n, y^n)$ are independently generated, the likelihood of predicting all y^i's from the corresponding $\mathbf{x}^i$'s is given by $\prod_{i=1}^{n} P(y^i|\mathbf{x}^i)$. Hence, we wish to select the hypothesis h (i.e., find its $\mathbf{w}$) that maximizes this probability (this is known as the method of *maximum likelihood*). Equivalently, we can minimize the following:

$$E(\mathbf{w}) = -\frac{1}{n} \ln \left(\prod_{i=1}^{n} P(y^i|\mathbf{x}^i) \right) = \frac{1}{n} \sum_{i=1}^{n} \ln \left(\frac{1}{P(y^i|\mathbf{x}^i)} \right) = \frac{1}{n} \sum_{i=1}^{n} \ln \left(1 + \exp(-y^i\, \mathbf{w}{\cdot}\mathbf{x}^i) \right). \quad (1)$$

Why is this so? And why is this expression computationally more 'convenient'?

3 Gradient Descent

To train logistic regression (find $\mathbf{w}$ that minimizes expression 1), we can try an approach similar to linear regression by setting $\nabla E(\mathbf{w}) = 0$. Unfortunately, unlike linear regression, $\nabla E(\mathbf{w})$ for logistic regression is amenable to an analytic solution. Although we cannot analytically set the gradient to zero, we can *iteratively* to do using a new algorithm *gradient descent*. Gradient descent is a general algorithm that can be used to train many other learning models with smooth evaluation functions.

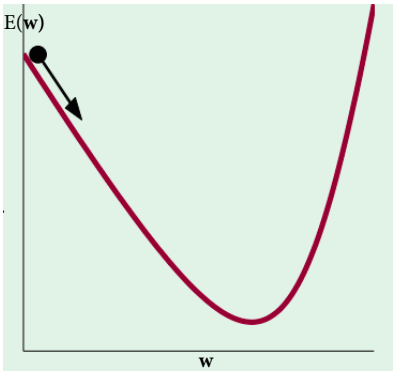

Figure 2: A convex function.

To build intuition, let us consider an analogy of a ball rolling down a hilly surface. If the ball is placed on a hill, it will roll down and come to rest at the bottom of a valley. The same idea underlies gradient descent. $E(\mathbf{w})$ is a 'surface' in a high-dimensional space. At step 0, we start somewhere on this surface with weights $\mathbf{w}^0$, and roll down, thereby decreasing $E(\mathbf{w})$. Like in the ball in the analogy, we may come to rest at a valley that is not the lowest of the entire

surface. Such a valley is known as a *local minimum*. However, $E(\mathbf{w})$ has a nice property – it is *convex*[1] (Figure 2). This means that it has a single *global minimum* (one valley), and no matter where we start, we will always come to rest at that minimum.

Note that the gradient $\nabla E(\mathbf{w})$ points in the direction where $E(\mathbf{w})$ increases. Hence, to minimize $E(\mathbf{w})$, we should take a small step in the opposite direction. Simply put, we modify the weights from iteration t to $t+1$as follows:

$$\mathbf{w}^{t+1} = \mathbf{w}^t - \eta \nabla E(\mathbf{w}^t)$$

where η is a positive real number known as the *learning rate* or *step size*. The learning rate η can be set by holding out some of the training data and using it to tune η to get good performance. A typically good choice for η is around 0.1 (a purely practical observation).

```
1  Initialize weights w^0
2  for t = 0, 1, 2, ... do
3     Compute gradient ∇E(w^t)
4     Update weights: w^{t+1} = w^t − η ∇E(w^t)
5     Iterate to next step unless termination condition is met
6  Return final weights
```

To execute line 3 of the gradient descent algorithm, we need to compute the gradient of logistic regression. Verify that

$$\nabla E(\mathbf{w}) = -\frac{1}{n} \sum_{i=1}^{n} \frac{y^i \mathbf{x}^i}{1 + \exp(y^i \, \mathbf{w} \cdot \mathbf{x}^i)} \; . \tag{2}$$

On line 1, the weights are typically initialized randomly, for example, by choosing each weight independently from a Gaussian distribution with zero mean and a small variance. On line 5, we need to decide whether to terminate. Usually a combination of conditions is used to determine this. Typically we terminate when a maximum number of iterations is reached, or the change in $E(\mathbf{w})$ between iteration is small (below some threshold) and the gradient is small ($||\nabla E(\mathbf{w})|| = \sqrt{\nabla E(\mathbf{w}) \cdot \nabla E(\mathbf{w})}$ falls below some threshold).

4 Stochastic Gradient Descent (SGD)

The gradient descent algorithm is a *batch* method, i.e., the gradient is computed on the full batch of training examples in order to update the weights. A more efficient version of gradient descent uses only one training example to estimate the gradient. In *stochastic* gradient descent, we choose a training example $(\mathbf{x}^i, y^i)$ uniformly at random (i.e., stochastically), and consider the evaluation function $e^i(\mathbf{w})$ with respect to that one example. For logistic regression,

$$e^i(\mathbf{w}) = \ln\left(1 + \exp(-y^i \, \mathbf{w} \cdot \mathbf{x}^i)\right) \; .$$

The gradient of this single example is used for the weight update in the same manner as the batch approach. The gradient is

$$\nabla e^i(\mathbf{w}) = \frac{-y^i \mathbf{x}^i}{1 + \exp(y^i \, \mathbf{w} \cdot \mathbf{x}^i)}$$

[1] A convex function has a characteristic 'bowl' shape. More formally, a convex function $f(\mathbf{z})$ is one where $f(\alpha \mathbf{z} + (1-\alpha)\mathbf{z}') \leq \alpha f(\mathbf{z}) + (1-\alpha) f(\mathbf{z}')$ for any $\mathbf{z}, \mathbf{z}'$ and $\alpha \in [0, 1]$.

and the weight update is

$$\mathbf{w}^{t+1} = \mathbf{w}^t - \eta \nabla e^i(\mathbf{w}^t) \ .$$

To understand how SGD works, let's examine the expected change in weight (the expectation is with respect to the random training example selected). Since $(\mathbf{x}^i, y^i)$ is picked uniformly at random from $(\mathbf{x}^1, y^1), \ldots, (\mathbf{x}^n, y^n)$, the expected weight change is

$$\eta . \frac{1}{n} \sum_{i=1}^{n} \nabla e^i(\mathbf{w}) \ ,$$

which is exactly the same as the deterministic weight change using the batch approach (compare with Equation 2). Thus, in expectation, the stochastic approach proceeds in the right direction, but has some random fluctuations, which cancel each other out in the long run. A big advantage of the stochastic approach is that it is computationally cheaper by a factor of n because we compute the gradient with only one (rather than n) example.

In practice, SGD is successful, often beating its batch counterpart.

5 Linear Regression Revisited

Recall that in linear regression we analytically minimize the evaluation function

$$E(\mathbf{w}) = \frac{1}{2} \sum_{i=1}^{n} (y^i - \mathbf{w} \cdot \mathbf{x}^i)^2$$

by computing the inverse of a matrix. This approach takes $O(d^3)$ time, which can be expensive when the number of dimensions d in each example is large. As it turns out, $E(\mathbf{w})$ is convex. Hence we can use (stochastic) gradient descent to circumvent the $O(d^3)$ problem. We simply plug the gradient of $E(\mathbf{w})$ into the gradient descent algorithms above, and iteratively obtain the optimal $\mathbf{w}$.

6 Linear Classification Revisited

Recall that the perceptron algorithm only works for linearly separable data, and its evaluation function is $E(\mathbf{w}) = \sum_{i=1}^{n} [\![y^i(\mathbf{w} \cdot \mathbf{x}^i) \leq 0]\!]$. This expression is also known as *zero-one* loss.

We shall look at an alternative evaluation function. Let $e(y\,\mathbf{w}\cdot\mathbf{x}) = \max\{1 - y\,\mathbf{w}\cdot\mathbf{x},\ 0\}$. This expression forces the classifier predictions to be more than correct: the agreement $z = y\,\mathbf{w}\cdot\mathbf{x}$ must be ≥ 1 before $e(z)$ is zero. If we reduce the agreement below 1 (or ≤ 0 for misclassified points), $e(z)$ increases linearly, and get larger as predictions get more incorrect (i.e., when $z < 1$, $e(z)$ is simply $1 - z$). We define our new evaluation function as

$$E(\mathbf{w}) = \frac{1}{n} \sum_{i=1}^{n} e(y^i\,\mathbf{w}\cdot\mathbf{x}^i) = \frac{1}{n} \sum_{i=1}^{n} \max\{1 - y^i\,\mathbf{w}\cdot\mathbf{x}^i,\ 0\} \ .$$

As it turns out, this new evaluation function is convex in $\mathbf{w}$, and we can simply plug its gradient into the (stochastic) gradient descent algorithm to find the optimal $\mathbf{w}$. This evaluation function is also known as the *hinge loss*.

There is one technical issue we have to address. The function $E(\mathbf{w})$ is not everywhere differentiable. $E(\mathbf{w})$ is a sum of piece-wise linear functions, and thus it is also a piece-wise linear

function. This means that it will have 'kinks', i.e., points where it is not differentiable. At those points, we have several possible gradients. For our purpose of minimizing $E(\mathbf{w})$, we only need to select one possible gradient at any point regardless of how many choices there are. We can do this by simply picking any gradient right around a 'kink'.