Logistic Regression and Gradient Ascent

CS 349-02 (Machine Learning)

April 10, 2017

The perceptron algorithm has a couple of issues: (1) the predictions have no probabilistic interpretation or confidence estimates, and (2) the learning algorithm has no principled way of preventing overfitting.[1]

There is another model for learning a hyperplane called logistic regression. Here are the main differences from our perceptron learner:

1. Since we care about the *probability* of a prediction, we define, for a data-point $\boldsymbol{x}$, under a given hyperplane, the probability of a label y:

$$P(y = 1|\boldsymbol{x}; \boldsymbol{w}, b) = \frac{1}{1 + e^{-(\boldsymbol{w}.\boldsymbol{x}+b)}} \tag{1}$$

   For brevity, and to be consistent with textbooks, let's denote both the $\boldsymbol{w}$ and b parameters by θ. For a d-dimensional $\boldsymbol{w} = [w_1, w_2, \ldots w_d]$, θ will represent $[b, w_1, w_2, \ldots w_d]$.

   We will use $h_\theta(\boldsymbol{x})$ to refer to the quantity in Eq. 1, mainly because $P(y = 1|\boldsymbol{x}; \theta)$ is too long to write out.

   Sidenote: the function

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

   is called a sigmoid or logistic function. You can see by plotting it or plugging in values for z that it always takes on a value between 0 and 1.

2. Instead of our class labels y taking on values of 1 or -1 as in the perceptron, we will have them take on values 1 or 0. For example, in your PS2 task, you would assign `positive` to 1 and `negative` to 0 if you were using logistic regression.

   Just like the perceptron, it doesn't matter which class we decide to label as 1 versus 0. It's just that the 1/0 labeling scheme makes the math cleaner compared to a $1/-1$

[1] Averaging, etc. are heuristic approaches with no provable guarantees.

scheme, mainly because we're thinking of the predictions as probabilities (which range from 0 to 1).

3. Logistic regression gives us an explicit objective to aim towards, which can be achieved with several different optimization algorithms. The perceptron learner is an algorithm, and while it is implicitly trying to minimize the number of errors on the training data, it is not a formal, explicit objective. Being able to frame an objective allows us to also prevent overfitting, as we will see later.

Important note on names Logistic regression actually solves a *classification* task where the labels are one of two classes, just like the other (perceptron, kNN) algorithms we've seen so far – **not** a *regression* task where the labels are real numbers. It is a misnomer due to its similarity and historical connection with linear regression.

1 Maximum Likelihood Principle and Likelihood

The Maximum Likelihood Estimation Principle, or MLE, is a general framework in probabilistic machine learning.

The model we're learning is the hyperplane, parametrized by $\boldsymbol{w}$ and b, which as noted in the earlier section, we'll refer to by the single value θ.

The **likelihood** of a given model θ for a dataset $\boldsymbol{X}, \boldsymbol{y}$ (where $\boldsymbol{X}$ is a set of m data-points and $\boldsymbol{y}$ is the corresponding set of labels) is defined as [2]

$$L(\theta) = P(\boldsymbol{y}|\boldsymbol{X}; \theta)$$

MLE says that of *all* the hyperplanes θ that are possible, we should pick the one that has the **highest** likelihood for this data. In order to do this, we should expand out the likelihood definition above as [3]

$$L(\theta) = P(\boldsymbol{y}|\boldsymbol{X}; \theta) = \prod_{i=1}^{m} P(y^{(i)}|\boldsymbol{x}^{(i)}; \theta)$$

[2] The notation $P(a|b; c)$ is the probability of a given (conditioned on) b under a model c.

[3] The notation $\prod_{i=1}^{4} a_i$ stands for the product of a_1, a_2, a_3, and a_3, just like $\sum_{i=1}^{4} a_i$ is notation of the sum of a_1, a_2, a_3, and a_4.

because the probability of a dataset is the product of the probabilities of each point (i.e., the probability of *all* the points occurring).[4]

What is $P(y|\boldsymbol{x};\theta)$? When $y = 1$, it is exactly given by Eq. 1, that is, $h_\theta(\boldsymbol{x})$. When $y = 0$, it is $1 - h_\theta(\boldsymbol{x})$. The equation below captures this:

$$P(y|\boldsymbol{x};\theta) = h_\theta(\boldsymbol{x})^y \cdot (1 - h_\theta(\boldsymbol{x}))^{1-y} \tag{2}$$

Ex. Explain why Eq. 2 gives the correct value of $P(y|\boldsymbol{x};\theta)$ for both possible values of y.

Plugging Eq. 2 back into the likelihood:

$$L(\theta) = \prod_{i=1}^{m} P(y^{(i)}|\boldsymbol{x}^{(i)};\theta) = \prod_{i=1}^{m} h_\theta(\boldsymbol{x}^{(i)})^{y^{(i)}} \cdot (1 - h_\theta(\boldsymbol{x}^{(i)}))^{1-y^{(i)}}$$

Take the log of this expression, to avoid underflow when multiplying small numbers, and because sums are easier to do calculus with than products. Since $\log L(\theta)$ is a monotonic function of $L(\theta)$, maximizing it is the *same* as maximizing $L(\theta)$ itself.

$$\log L(\theta) = \sum_{i=1}^{m} y^{(i)} \cdot \log h_\theta(\boldsymbol{x}^{(i)}) + (1 - y^{(i)}) \cdot \log(1 - h_\theta(\boldsymbol{x}^{(i)})) \tag{3}$$

Our training objective is to find the value of θ – that is, the hyperplane – for which the log likelihood is maximized, compared to all other possible hyperplanes.

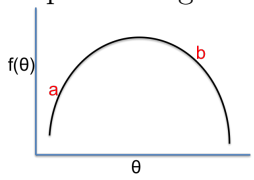
2 Optimization with Gradient Ascent

You already know that you can find the maximum of a function by computing its derivative, setting it to 0, and solving. Unfortunately, that is impossible with Eq. 3 since it is the sum of an arbitrary number of terms and has no closed form.

One trick is called gradient ascent (known as gradient descent when we are minimizing rather than maximizing a function). Consider a convex function like the sketch below, where θ is 1-dimensional. When you are at the certain value of θ, you can calculate $f(\theta)$ and the derivative (slope) f' of the function at that point, but you can't see the rest of the function.

[4]The notation $y^{(i)}$ refers to the label of the i^{th} training point, and $\boldsymbol{x}^{(i)}$ to the feature vector of that point. Parentheses are used to clarify that i is an index, not an exponent.

Imagine walking up a mountain where all you know at any step is (1) your elevation and (2) the slope of the ground.



Starting from point a in the sketch, how would you decide which direction to move in order to reach the peak eventually? There are only two possible directions, right or left. The best decision is to move in the direction of the slope at a, which is positive. Similarly, starting from point b, you would move in the direction of the slope there, which is negative.

Furthermore, it makes sense to take a big step from a, where the magnitude of slope is larger, since this tends to happen further away from the maximum, and a smaller step from b because the magnitude of the slope there is a bit smaller.

Gradient ascent formalizes this intuition. It says that we update θ by the derivative multiplied by η, a constant learning rate. η controls how big a step we make in the direction given by the derivative. [5] If we start at $\theta = a$ at time $t-1$:

$$\theta^{(t)} \rightarrow \theta^{(t-1)} + \eta f'(\theta)|_{\theta=a}$$

2.1 Gradient Ascent in Multiple Dimensions

The same principle and update rule applies to multi-dimensional functions. The only difference is how we compute the derivative. The gradient of a function $f(\theta)$ when θ is 2-dimensional, for example, is given by a vector of partial derivatives at each dimension. We use the gradient symbol $\nabla_\theta f$ instead of f' when dealing with multiple dimensions.

A **partial derivative** is the derivative of a function along a single dimension, treating the other dimensions as constants. So

$$\nabla_\theta f(\theta) = \left[\frac{\partial f}{\partial \theta_1}, \frac{\partial f}{\partial \theta_2}\right]$$

This generalizes to more dimensions. For d-dimensional θ,

$$\nabla_\theta f(\theta) = \left[\frac{\partial f}{\partial \theta_1}, \frac{\partial f}{\partial \theta_2}, \frac{\partial f}{\partial \theta_3} \cdots \frac{\partial f}{\partial \theta_d}\right]$$

[5] Notation: $\theta^{(t)}$ denotes the value of θ at time-step t.

Given a function $f(\theta) = -\theta_1^2 + 5\theta_2^3$,

$$\frac{\partial f}{\partial \theta_1} = -2\theta_1 \text{ (treating } \theta_2 \text{ as constant)}$$
$$\frac{\partial f}{\partial \theta_2} = 15\theta_2^2 \text{ (treating } \theta_1 \text{ as constant)}$$

Therefore,

$$\nabla_\theta f(\theta) = [-2\theta_1, 15\theta_2^2]$$

The gradient of $f(\theta)$ at the point $[2, 1]$, for example, is $[-2 \cdot 2, 15 \cdot 1^2] = [-4, 15]$.

Ex. Compute the gradient of $f(\theta) = \log 2\theta_1 + \theta_1\theta_2 - \theta_2^4 + e^{\theta_3}$ at [2, 3, 0].

2.2 Gradient Ascent Applied to Logistic Regression

Eq. 3 gives us a function to maximize to find the optimal hyperplane θ. Using the gradient ascent approach gives us this update rule at time step t:

$$\theta^{(t)} \rightarrow \theta^{(t-1)} + \eta \nabla \log L(\theta)|_{\theta=\theta^{(t-1)}} \tag{4}$$

Let's compute $\nabla \log L(\theta)$:

$$\nabla_\theta \log L(\theta) = \nabla_\theta \sum_{i=1}^{m} y^{(i)} \cdot \log h_\theta(\boldsymbol{x}^{(i)}) + (1 - y^{(i)}) \cdot \log(1 - h_\theta(\boldsymbol{x}^{(i)})) \tag{5}$$
$$= \sum_{i=1}^{m} \frac{y^{(i)}}{h_\theta(\boldsymbol{x})} \nabla_\theta h_\theta(\boldsymbol{x}^{(i)}) + \frac{1 - y^{(i)}}{1 - h_\theta(\boldsymbol{x}^{(i)})}(-\nabla_\theta h_\theta(\boldsymbol{x}^{(i)})) \tag{6}$$

Evaluating $\nabla_\theta h_\theta(\boldsymbol{x})$ Recall that θ was our shorthand for the combination of $\boldsymbol{w}$ and b that parameterize the hyperplane, so $h_\theta(\boldsymbol{x}) = \dfrac{1}{1 + e^{-(\boldsymbol{w}\boldsymbol{x}+b)}}$. Its derivative with respect to θ is

$$-1 \cdot (1 + e^{-(\boldsymbol{w}\boldsymbol{x}+b)})^{-2} \cdot e^{-(\boldsymbol{w}\boldsymbol{x}+b)} \cdot -\boldsymbol{x}$$
$$= \boldsymbol{x} \cdot \frac{1}{1 + e^{-(\boldsymbol{w}\boldsymbol{x}+b)}} \cdot \frac{e^{-(\boldsymbol{w}\boldsymbol{x}+b)}}{1 + e^{-(\boldsymbol{w}\boldsymbol{x}+b)}}$$

Conveniently, $\dfrac{e^{-(\boldsymbol{w}\boldsymbol{x}+b)}}{1 + e^{-(\boldsymbol{w}\boldsymbol{x}+b)}} = 1 - \dfrac{1}{1 + e^{-(\boldsymbol{w}\boldsymbol{x}+b)}}$, so the above quantity becomes

$$\boldsymbol{x} \cdot h_\theta(\boldsymbol{x}) \cdot (1 - h_\theta(\boldsymbol{x}))$$

We put this together with Eq. 6. (To reduce congestion, I removed the $^{(i)}$ notation to denote the i^{th} example. Read all mentions of $\boldsymbol{x}$ and y in the computation below as $\boldsymbol{x}^{(i)}$ and $y^{(i)}$ respectively.)

$$\begin{aligned}
\nabla_\theta \log L(\theta) &= \sum_{i=1}^{m} \frac{y}{h_\theta(\boldsymbol{x})} \boldsymbol{x} \cdot h_\theta(\boldsymbol{x}) \cdot (1 - h_\theta(\boldsymbol{x})) + \frac{y-1}{1 - h_\theta(\boldsymbol{x})} \boldsymbol{x} \cdot h_\theta(\boldsymbol{x}) \cdot (1 - h_\theta(\boldsymbol{x})) \\
&= \sum_{i=1}^{m} y \cdot \boldsymbol{x} \cdot (1 - h_\theta(\boldsymbol{x})) + (y-1) \cdot \boldsymbol{x} \cdot h_\theta(\boldsymbol{x}) \\
&= \sum_{i=1}^{m} \boldsymbol{x} \cdot y \cdot (1 - h_\theta(\boldsymbol{x})) + (y-1) \cdot h_\theta(\boldsymbol{x}) \\
&= \sum_{i=1}^{m} \boldsymbol{x} \cdot (y - h_\theta(\boldsymbol{x})) \qquad (7)
\end{aligned}$$

In terms of the partial derivatives, for our $d+1$-dimensional hyperplane θ, this gradient is the vector

$$\left[\frac{\partial}{\partial \theta_0} \log L(\theta), \frac{\partial}{\partial \theta_1} \log L(\theta), \frac{\partial}{\partial \theta_2} \log L(\theta), \ldots \frac{\partial}{\partial \theta_d} \log L(\theta) \right]$$

For $j \geq 1$, where $\boldsymbol{x}_j$ is the value of the point $\boldsymbol{x}$ in dimension j:

$$\frac{\partial}{\partial \theta_j} \log L(\theta) = \sum_{i=1}^{m} \boldsymbol{x}_j^{(i)} \cdot (y^{(i)} - h_\theta(\boldsymbol{x}^{(i)}))$$

For $j = 0$, because $\boldsymbol{x}$ doesn't have a value in that dimension,

$$\frac{\partial}{\partial \theta_0} \log L(\theta) = \sum_{i=1}^{m} (y^{(i)} - h_\theta(\boldsymbol{x}^{(i)}))$$

This gives us the gradient for our update (refer back to Eq. 4). Updating each dimension j at a time:

$$\begin{aligned}
\theta_j^{(t)} \rightarrow \theta_j^{(t-1)} + \eta \sum_{i=1}^{m} \boldsymbol{x}_j^{(i)} \cdot (y^{(i)} - h_{\theta^{(t-1)}}(\boldsymbol{x}^{(i)})) \text{ for all } j \geq 1 \\
\theta_0^{(t)} \rightarrow \theta_0^{(t-1)} + \eta \sum_{i=1}^{m} y^{(i)} - h_{\theta^{(t-1)}}(\boldsymbol{x}^{(i)}) \qquad (8)
\end{aligned}$$

When implementing the algorithm, it is faster to treat $[\theta_1, \theta_2, \ldots \theta_d]$ as a vector and and do a single vector addition rather than write a for-loop over the different dimensions j. Going back to our original $\boldsymbol{w}$, b notation is better for this. The above equation is re-written as

$$w^{(t)} \rightarrow w^{(t-1)} + \eta \sum_{i=1}^{m} \boldsymbol{x}^{(i)} \cdot (y^{(i)} - h_{\theta^{(t-1)}}(\boldsymbol{x}^{(i)})) \text{ for all } j \geq 1$$

$$b^{(t)} \rightarrow b^{(t-1)} + \eta \sum_{i=1}^{m} y^{(i)} - h_{\theta^{(t-1)}}(\boldsymbol{x}^{(i)}) \tag{9}$$

2.2.1 Algorithm for Logistic Regression Gradient Ascent (batch version)

Just like with the perceptron, we start with all-zeros for $\boldsymbol{w}$ and b. The algorithm makes a sweep through the data, computes the gradient, and updates the hyperplane. This repeats for $Maxiter$ epochs.

- $\boldsymbol{w}$ = all-zeros, $b = 0$
- for *epoch* in [1, 2, ... Maxiter]:
  - gradw = all zeros (to compute the first gradient in Eq. 9)
  - gradb = 0 (to compute the second gradient in Eq. 9)
  - for $\boldsymbol{x}^{(i)}, y^{(i)}$ for $i \in [1, m]$ in training data:
    - gradw $\leftarrow$ gradw $+\boldsymbol{x}^{(i)} \cdot (y^{(i)} - h_\theta(\boldsymbol{x}^{(i)}))$
    - gradb $\leftarrow$ gradb $+(y^{(i)} - h_\theta(\boldsymbol{x}^{(i)}))$
  - $\boldsymbol{w} \leftarrow \boldsymbol{w} + \eta$ gradw
  - $b \leftarrow b + \eta$ gradb

When do we stop? We could see when our likelihood stops growing much. It's usually not a good idea to do this in practice. Instead, we'll run it all the way until $Maxiter$. Unlike the perceptron, there's usually always room for update, since it's not an all-or-nothing error prediction.

2.3 Stochastic Gradient Ascent

The summation in the above equation is expensive, since we need to go through all the training points just to make *one* update.

In contrast, the perceptron algorithm did "online" updates, where *online* refers to updating the model parameters after each training point, rather than the Eq. 9 strategy of going through the entire "batch" of training examples for an update.

Stochastic gradient ascent (SGD) is an online updating variant of traditional (batch) gradient ascent. The idea is to iterate through the labeled training data-points $\boldsymbol{x}^{(i)}, y^{(i)}$ in epochs just like the perceptron. On each iteration, we compute the gradient only at that training point (rather than summing up over the dataset), and immediately update our hyperplane.

This is a noisy approximation of the true gradient, but it leads to faster learning. Here's the modified algorithm:

- $\boldsymbol{w}$ = all-zeros, $b = 0$
- for *epoch* in [1, 2, ... Maxiter]:
  - for $\boldsymbol{x}^{(i)}, y^{(i)}$ in training data:
    - $\boldsymbol{w} \leftarrow \boldsymbol{w} + \boldsymbol{x}^{(i)} \cdot (y^{(i)} - h_\theta(\boldsymbol{x}^{(i)}))$
    - $b \leftarrow b + (y^{(i)} - h_\theta(\boldsymbol{x}^{(i)}))$

A midway compromise between the more accurate but expensive batch gradient ascent and the faster but noisier SGD is to divide the data into "mini-batches", accumulate the gradient on each mini-batch, and update.

3 Regularization

The above learning procedures could cause $\boldsymbol{w}$ to become very large. This leads to *overfitting*, because

$$h_\theta(\boldsymbol{x}) = \frac{1}{1 + e^{-(\boldsymbol{w}.\boldsymbol{x}+b)}}$$

will end up being 0 or 1 most of the time, and we lose the agnosticism of the logistic regression (an advantage over the perceptron).

The great thing about having an explicit objective function is that we can throw in other preferences! So if I don't want my something to become too large, I'll simply ask the objective to **penalize** its L2 norm.

This is called L2 (or 'ridge') regularization, and is achieved with a small tweak to Eq. 3.

$$\begin{aligned}\text{Objective}(\theta) &= \log L(\theta) - \frac{\alpha}{2}||\theta||^2 \\ &= \sum_{i=1}^{m} y^{(i)} \cdot \log h_\theta(\boldsymbol{x}^{(i)}) + (1 - y^{(i)}) \cdot \log(1 - h_\theta(\boldsymbol{x}^{(i)})) - \frac{\alpha}{2}\sum_{j=1}^{d} \theta_j^2\end{aligned}$$

where α is a hyperparameter[6] denoting the regularization weight, determining how *much* to penalize large models at the expense of the log likelihood. Notice that I have approximated $||\theta||^2$ as $\sum_{j=1}^{d} \theta_j^2$ rather than $\sum_{j=0}^{d} \theta_j^2$. That is, we're only regularizing the $\boldsymbol{w}$ coefficients, not b, which is θ_0, because in practice, b doesn't grow that large and we prefer not to penalize it.

Computing the gradient of $\frac{\alpha}{2}\sum_{j=1}^{d} \theta_j^2$:

$$\begin{aligned}&\frac{\alpha}{2}\left[\frac{\partial}{\partial\theta_1}\sum_{j=1}^{d}\theta_j^2, \frac{\partial}{\partial\theta_2}\sum_{j=1}^{d}\theta_j^2, \ldots \frac{\partial}{\partial\theta_d}\sum_{j=1}^{d}\theta_j^2\right] \\ &= \frac{\alpha}{2}[2\theta_1, 2\theta_2, \ldots 2\theta_d] = \alpha[\theta_1, \theta_2, \ldots \theta_d] \text{ which is simply } \alpha\boldsymbol{w}\end{aligned}$$

The regularized function's derivative is an adaptation of our original gradient to account for this new term. The SGD update therefore becomes

$$\boldsymbol{w} \leftarrow \boldsymbol{w} + \boldsymbol{x}^{(i)} \cdot (y^{(i)} - h_\theta(\boldsymbol{x}^{(i)})) - \alpha\boldsymbol{w}$$

and batch gradient ascent update becomes

$$\boldsymbol{w} \leftarrow \boldsymbol{w} + \sum_{i=1}^{m}\left(\boldsymbol{x}^{(i)} \cdot (y^{(i)} - h_\theta(\boldsymbol{x}^{(i)}))\right) - \alpha\boldsymbol{w}$$

The update for b remains the same as before because we decided not to penalize it.

There are other types of regularization, like L1 (or 'lasso') regularization, which leads to *sparse* models (weight vectors with many zeros). However, the L1 norm is not differentiable at 0, and requires some tricks to approximate the gradient with "subgradients". We won't cover L1 regularization, but you should know its existence and understand why it's used.

[6]Tune η and α on development data. Dang those hyperparameters!