Math 345 - Notes
Logistic Regression
October 29, 2018

Logistic Regression

Let us consider the credit question: given customer data, can a decision be made about their credit?

(a) We used the PLA algorithm to approximate a target function that outputs $+1, -1$ to make a binary decision on credit (approve/deny):

$$f(\mathbf{x}) = \text{sign}(\mathbf{w}^T\mathbf{x})$$

The sign function is a step function leading to a *hard threshold*.

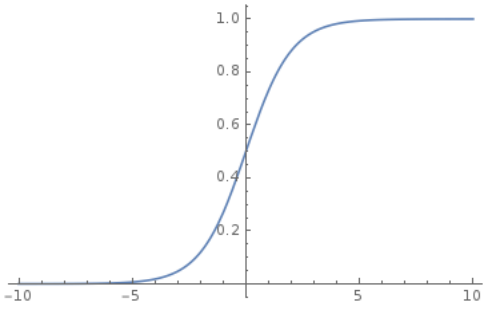
(b) We used linear regression to find a target function that outputs a real value, to estimate a credit score:

$$f(\mathbf{x}) = \mathbf{w}^T\mathbf{x}.$$

The output allows for negative values and is unbounded, hence the target function has *no threshold*.

(c) Suppose we are interested in a bounded function that outputs real values. For example, we may want to approximate the *probability* that a customer will default on their loan, given their customer data. Logistic regression attempts to learn exactly this sort of target function. Using the logistic function, the target function will produce a *soft threshold*.

Let $\theta : \mathbb{R} \to (0, 1)$ be the logistic function $\theta(s) = \dfrac{e^s}{1 + e^s}$.



A useful identity for this function is

$$\theta(-s) = \frac{e^{-s}}{1 + e^{-s}} = \frac{1}{e^s + 1} = 1 - \frac{e^s}{1 + e^s} = 1 - \theta(s). \tag{1}$$

Remark: Another popular function to use is the hyperbolic tangent function $\tanh(s) = \dfrac{e^s - e^{-s}}{e^s + e^{-s}}$, but this function is not as easy to work with.

The idea behind logistic regression is simple: suppose we want to learn the *target function*

$$f(\mathbf{x}) = P(y = +1 \,|\, \mathbf{x}),$$

where $y = +1$ encodes the occurrence of a specific event, such as the customer defaults on loan, the patient has the disease, the message is spam, etc. We use $y = -1$ to mean such event has not occurred. Then the probability of outcome y given the input data $\mathbf{x}$ is

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

We test possible candidates from the set of functions $\mathcal{H}$, with the range of $h(\mathbf{x})$ being $(0, 1)$ to approximate $f(\mathbf{x})$. Let

$$h(\mathbf{x}) = \theta(\mathbf{w}^T\mathbf{x}) = \frac{e^{\mathbf{w}^T\mathbf{x}}}{1 + e^{\mathbf{w}^T\mathbf{x}}}.$$

Then, using equation (1),

$$1 - h(\mathbf{x}) = 1 - \theta(\mathbf{w}^T\mathbf{x}) = \theta(-\mathbf{w}^T\mathbf{x}).$$

Thus, combining with the probability above, we approximate

$$P(y \,|\, \mathbf{x}) \approx \theta(y\, \mathbf{w}^T\mathbf{x})$$

How do we know which function h approximates the target function f best? Note that the function h depends on the coefficient vector $\mathbf{w}$, so we are really trying to find the best $\mathbf{w}$ to approximate f. To measure how close the function h is to f, we will use the method of **maximum likelihood**. This method maximizes how likely it is that we get an output y from the input $\mathbf{x}$ by using the function h. We use the training data to maximize this probability. Suppose our training data is $\mathcal{D} = \{(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_N, y_N)\}$, where the data points are assumed to be independent of each other. Then we maximize

$$P(y_1, y_2, \ldots, y_N \,|\, \mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_N) = \prod_{k=1}^{N} P(y_k \,|\, \mathbf{x}_k).$$

Taking the log of this function and dividing by $-N$ produces an error function that achieves a minimum where $P(y_1, y_2, \ldots, y_N \,|\, \mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_N)$ achieves a maximum. Let

$$\begin{aligned} E_{in}(\mathbf{w}) &= -\frac{1}{N} \log \left(\prod_{k=1}^{N} P(y_k \,|\, \mathbf{x}_k) \right) = -\frac{1}{N} \sum_{k=1}^{N} \log(P(y_k \,|\, \mathbf{x}_k)) \\ &= -\frac{1}{N} \sum_{k=1}^{N} \log(\theta(y_k\, \mathbf{w}^T\mathbf{x}_k)) = -\frac{1}{N} \sum_{k=1}^{N} \log \left(\frac{e^{y_k\, \mathbf{w}^T\mathbf{x}_k}}{1 + e^{y_k\, \mathbf{w}^T\mathbf{x}_k}} \right) \\ &= \frac{1}{N} \sum_{k=1}^{N} \log \left(\frac{1 + e^{y_k\, \mathbf{w}^T\mathbf{x}_k}}{e^{y_k\, \mathbf{w}^T\mathbf{x}_k}} \right) = \frac{1}{N} \sum_{k=1}^{N} \log \left(1 + e^{-y_k\, \mathbf{w}^T\mathbf{x}_k} \right) \end{aligned}$$

Remark: if $y_k\, \mathbf{w}^T\mathbf{x}_k$ is large and positive, the error is small, hence y_k is probably correctly labeled.

To find the minimum of this function, we want to set the gradient with respect to $\mathbf{w}$ equal to zero. We first find the gradient (remember to use chain rule!):

$$\begin{aligned}
\nabla_{\mathbf{w}} E_{in}(\mathbf{w}) &= \nabla_{\mathbf{w}} \left[\frac{1}{N} \sum_{k=1}^{N} \log \left(1 + e^{-y_k \, \mathbf{w}^T \mathbf{x}_k} \right) \right] \\
&= \frac{1}{N} \sum_{k=1}^{N} \nabla_{\mathbf{w}} \log \left(1 + e^{-y_k \, \mathbf{w}^T \mathbf{x}_k} \right) \\
&= \frac{1}{N} \sum_{k=1}^{N} \frac{1}{1 + e^{-y_k \, \mathbf{w}^T \mathbf{x}_k}} \left(e^{-y_k \, \mathbf{w}^T \mathbf{x}_k} \right) (-y_k \mathbf{x}_k) \\
&= \frac{1}{N} \sum_{k=1}^{N} \frac{-y_k \mathbf{x}_k}{1 + e^{y_k \, \mathbf{w}^T \mathbf{x}_k}} \\
&= -\frac{1}{N} \sum_{k=1}^{N} y_k \mathbf{x}_k \, \theta(-y_k \, \mathbf{w}^T \mathbf{x}_k).
\end{aligned}$$

While computing the gradient is easy (hence a good reason to use this function), solving $\nabla_{\mathbf{w}} E_{in}(\mathbf{w}) = \mathbf{0}$ is not trivial. We use the gradient descent algorithm or the stochastic gradient descent algorithm to find a $\mathbf{w}$ that minimizes $E_{in}(\mathbf{w})$ instead.

Logistic Regression Algorithm (with Gradient Descent)

1. Set the initial weights $\mathbf{w}_0$ and step size η
2. For $t \geq 0$,
   - find the gradient $\mathbf{g}_t = -\frac{1}{N} \sum_{k=1}^{N} \frac{y_k \mathbf{x}_k}{1 + e^{y_k \, \mathbf{w}_t^T \mathbf{x}_k}}$
   - update $\mathbf{w}_{t+1} = \mathbf{w}_t - \eta \, \mathbf{g}_t$.
3. Stop when "done"
4. Return final $\mathbf{w}_t$.

Remarks:

(a) To initialize $\mathbf{w}_0$, one can set it to $\mathbf{0}$. Another option is to initialize each coordinate in $\mathbf{w}(0)$ by independently sampling from a normal distribution with mean zero and small variance.

(b) To end the algorithm, one can run it for a fixed (thousands) number of steps, or run it until $\|\mathbf{g}_t\|$ drops below a certain small threshold (since minimum error is achieved at $\mathbf{g}_t = \mathbf{0}$), or a combination of both.

(c) Instead of using a constant step η, one can use variable η_t, typically with η_t decreasing.

Instead of using the error from all N data points, one can use error from one data point uniformly picked at random from the training set. Let

$$e_k(\mathbf{w}) = \log\left(1 + e^{-y_k \mathbf{w}^T \mathbf{x}_k}\right)$$

be the error from data point $(\mathbf{x}_k, y_k)$. Then the update step in the gradient descent algorithm will be based only on the error from this point, as described below:

Logistic Regression Algorithm (with Stochastic Gradient Descent)

1. Set the initial weights $\mathbf{w}_0$ and step size η
2. For $t \geq 0$,
   - pick one data point from $\mathcal{D}$ uniformly at random. Suppose it is $(\mathbf{x}_k, y_k)$.
   - find the gradient $\mathbf{g}_t = \nabla e_k(\mathbf{w}_t) = -\dfrac{y_k \mathbf{x}_k}{1 + e^{y_k \mathbf{w}_t^T \mathbf{x}_k}}$
   - update $\mathbf{w}_{t+1} = \mathbf{w}_t - \eta\, \mathbf{g}_t$.
3. Stop when "done"
4. Return final $\mathbf{w}$.

Remarks:

(a) the computational cost of using the stochastic version is cheaper by a factor of N

(b) the stochastic version is more wiggly, but in the long run it averages out.

(c) Stochastic Gradient Descent Algorithm is as efficient the Gradient Descent Algorithm, since on average the change at each iteration is

$$\begin{aligned}
\mathbb{E}\left[-\eta \nabla e(\mathbf{w})\right] &= -\eta \sum_{k=1}^{N} P(\text{pick data point } k) \nabla e_k(\mathbf{w}) \\
&= -\frac{\eta}{N} \sum_{k=1}^{N} \nabla e_k(\mathbf{w}) \\
&= -\frac{\eta}{N} \sum_{k=1}^{N} \frac{y_k \mathbf{x}_k}{1 + e^{y_k \mathbf{w}^T \mathbf{x}_k}} \\
&= -\eta \nabla E_{in}(\mathbf{w}).
\end{aligned}$$

Thus, the expected change in each iteration in the stochastic version is equal to the change in the regular version.