A robust logistic regression model

Real data is often imperfect. The labels attached to training examples are sometimes incorrect. The logistic/softmax models can model noisy labels by reducing the magnitude of the weight vector. However, a corrupted example with an extreme input $\mathbf{x}$ could force the weight vector in some direction to be small, even if many other examples, with less extreme $\mathbf{x}$, indicate otherwise. This problem occurs most readily for the negative log-likelihood loss, because a single example can have arbitrarily large loss.

There are various solutions to this issue. We could limit the magnitude of the feature vector: some practitioners only use binary features, or scale their feature vectors to have length one. We could try to identify outliers and discard them. We could intervene in the optimization procedure, and limit the size of the update from any individual example.

In this note we take a probabilistic modelling approach. We take the existing model of the labels, and modify it to include a model of label corruption. We can then derive a new expression for the log likelihood and proceed as before. We're not saying that you should routinely use this particular robust classifier — it's just an example of how writing down a probabilistic model for your data can create a "new method". After modifying the model we immediately get a new negative log-likelihood cost function. We'll optimize it using standard methods, but the results will depend on the model, so it can look like we have an entirely new method.

We will assume that a binary choice $m \in \{0, 1\}$ was made for each observation, about whether to corrupt the label:

$$P(m \mid \epsilon) = \text{Bernoulli}(m;\, 1-\epsilon) = \begin{cases} 1-\epsilon & m = 1 \\ \epsilon & m = 0. \end{cases} \tag{1}$$

With probability $(1-\epsilon)$ the model sets $m=1$ and generates a label using the normal logistic regression process (to keep notation simple, we'll assume binary classification). Otherwise, with probability ϵ, the model sets $m=0$ and picks the label uniformly at random:

$$P(y=1 \mid \mathbf{x}, \mathbf{w}, m) = \begin{cases} \sigma(\mathbf{w}^\top \mathbf{x}) & m = 1 \\ \frac{1}{2} & m = 0. \end{cases} \tag{2}$$

If we knew the indicator variable m for each example, we'd use it as a mask: we'd ignore the irrelevant points where $m=0$, and fit the remaining points as usual. However, because we don't observe the indicator variable m, we need to write an expression for the probability of just the label we observe. To include the unknown choice m on the right-hand side, we need to "marginalize" (sum) it out using the sum rule:

$$\begin{aligned} P(y=1 \mid \mathbf{x}, \mathbf{w}, \epsilon) &= \sum_{m \in \{0,1\}} P(y=1, m \mid \mathbf{x}, \mathbf{w}, \epsilon) && (3) \\ &= \sum_{m \in \{0,1\}} P(y=1 \mid \mathbf{x}, \mathbf{w}, m)\, P(m \mid \epsilon) && (4) \\ &= (1-\epsilon)\sigma(\mathbf{w}^\top \mathbf{x}) + \epsilon \frac{1}{2}. && (5) \end{aligned}$$

In the second line we use the product rule. In general the last term would be $P(m \mid \mathbf{x}, \mathbf{w}, \epsilon)$, but in this model we decided to make the choice independent of the input position and the weights. The first term in the product rule doesn't include ϵ in this case, because once we know m, knowing ϵ doesn't help predict the label.

Now we need the gradients of the log probability. As we've mentioned in passing, it's possible to automatically get gradients given code for our cost function (and we'll study how in a later week). However, for now we'll rearrange the gradients by hand to see if they are interpretable.

As in the previous logistic regression note, we use $\sigma_n = \sigma(z^{(n)}\mathbf{w}^\top\mathbf{x}^{(n)})$ as the probability of getting the nth label correct under standard logistic regression, where $z^{(n)} = (2y^{(n)} - 1)$ is a $\{-1, +1\}$ version of the label. After some manipulation, we can find the expression:

$$\nabla_{\mathbf{w}} \log P(z^{(n)} \mid \mathbf{x}^{(n)}, \mathbf{w}) = \frac{1}{1 + \frac{1}{2}\frac{\epsilon}{1-\epsilon}\frac{1}{\sigma_n}} \nabla_{\mathbf{w}} \log \sigma_n, \tag{6}$$

where $\nabla_{\mathbf{w}} \log \sigma_n = (1-\sigma_n) z^{(n)} \mathbf{x}^{(n)}$ was the gradient of the log-probability for the original logistic regression model. As a check, we recover the original model when $\epsilon = 0$.

If we implemented the above equation we could check it with finite differences. If you derive a derivative that looks different, you could also check it's correct using finite differences, and/or numerically check it against the equation provided.

In the original model, the gradient is large for an extreme $\mathbf{x}$ where the label is poorly predicted. In the robust model, the contribution from the gradient is small if the probability of being correct, σ_n, is much smaller than the probability of a corruption, ϵ. Really extreme outliers will be nearly ignored.

We could set ϵ by hand, for example to 0.01. We could try a grid of settings — perhaps uniform on a log scale if we don't know how small epsilon should be. Or we could optimize ϵ along with $\mathbf{w}$ using gradient methods. However, most gradient-based optimizers need the parameters to be unconstrained, whereas ϵ is constrained to $[0, 1]$.

[The website version of this note has a question here.]

1 Reflection

You might be uncomfortable with the somewhat arbitrary model and assumptions in this note. If so, it's worth reflecting on whether you were as uncomfortable with the original logistic regression, where we made a stronger assumption, that $\epsilon = 0$! All models are wrong, but one way to test how much their assumptions matter is to add free parameters and see whether the data justify using them.

You might have written down a different data corruption process. Perhaps you would rather have a corruption process that flips the correct label, rather than generating a label uniformly at random. You would be free to work through that model, and obtain a different cost function for ϵ. You might ask if you have a genuinely different model, or just a different parameterization of the same model.

The negative log likelihood of the robust regression model is not convex for $\epsilon > 0$. (Review convexity if you can't say why.) So, we can't provide the same guarantees about being able to find the minimum of this loss as for logistic regression. However, that doesn't seem a strong argument to abandon the idea. If concerned about doing worse than standard logistic regression, we could initialize a fit with weights using $\epsilon = 0$ (or ϵ very small), and try to improve the model from there.

An optional (hard) exercise for keen students: We could write down a model that states that the standard logistic regression model with no label noise is correct for our data, but that the inputs $\mathbf{x}$ are noisy. Why did I not go that route? That is, if you write down a model that says $\mathbf{x}$ could be corrupted by noise, do you get stuck? If so, where and why? You'd have to make arbitrary choices for the corruption process (for example adding Gaussian noise to real-valued inputs). Then if you have a firm grasp on the rules of probabilistic inference, you could write down a likelihood. But could you compute it?