Softmax regression

Softmax[1] regression is a generalization of logistic regression to "multiclass classification": each label can take on one of $K \geq 2$ discrete settings. Some textbooks will simply call this generalization "logistic regression" as well.

As previously discussed one way to handle multiclass classification, is to represent the each label with a length-K one-hot encoding:

$$\mathbf{y} = [0 \;\; 0 \;\; \ldots \;\; 0 \;\; 1 \;\; 0 \;\; \ldots \;\; 0]. \tag{1}$$

If data-point n has label c, then $y_k^{(n)} = \delta_{kc}$, where δ_{kc} is a Kronecker delta.

We could attempt to predict the elements of $\mathbf{y}$ one at a time, with K separate one-vs-rest classifiers. We would need a method to combine the outputs of these classifiers. Instead we'll fit a probabilistic model that is explicit about how to predict the whole vector.

Our model will have parameters W and outputs a vector $\mathbf{f}$. We interpret the vector output $\mathbf{f}$ as a probability distribution over which bit in the target vector $\mathbf{y}$ is turned on, giving the model:

$$P(y_k = 1 \,|\, \mathbf{x}, W) = f_k(\mathbf{x}; W). \tag{2}$$

We can start by using a separate regression vector $\mathbf{w}^{(k)}$ for each class, and create a positive score, by exponentiating a linear combination of features:

$$s_k = e^{(\mathbf{w}^{(k)})^\top \mathbf{x}} \tag{3}$$

Moving the features $\mathbf{x}$ in the direction $\mathbf{w}^{(k)}$ increases the score for class k. In particular, if $w_d^{(k)}$ is positive, then large values of x_d give class k a large score.

However, probabilities have to add up to one, so our output vector is normalized:

$$f_k = \frac{s_k}{\sum_{k'} s_{k'}} = \frac{e^{(\mathbf{w}^{(k)})^\top \mathbf{x}}}{\sum_{k'} e^{(\mathbf{w}^{(k')})^\top \mathbf{x}}}. \tag{4}$$

Here k' is a dummy index used to sum over all of the classes. Our vector-valued function $\mathbf{f}$ is parameterized by $W = \{\mathbf{w}^{(k)}\}_{k=1}^K$. The classes now *compete* with each other. If $w_d^{(k)}$ is large, then large x_d doesn't necessarily favour class k. The normalization means that the probability of class k could be small if $w_d^{(j)}$ is larger for another class j.

If W is a $K \times D$ matrix, we might write the above function as $\mathbf{f}(\mathbf{x}; W) = \text{softmax}(W\mathbf{x})$.

1 Fitting the model

We want our function evaluated at a training input, $\mathbf{f}(\mathbf{x}^{(n)})$, to predict the corresponding target $\mathbf{y}^{(n)}$. At best, they are equal, in which case we confidently predict the correct label. However, if the labels are inherently noisy, so that the same input could be labelled differently, making perfect predictions every time is impossible.

Matching targets $\mathbf{y}$ and function values $\mathbf{f}$ by least squares would be *consistent*: if a setting of the parameters can make $\mathbf{f}$ match the probabilities of the labels under the real process that is generating the data, then we will fit those parameters given infinite data. However, "maximum likelihood" estimation is also consistent, and for well-specified models has faster asymptotic convergence. Compared to least squares, maximum likelihood heavily penalizes

1. I believe the term *softmax* was coined by John S. Bridle (1990). Probabilistic Interpretation of Feedforward Classification Network Outputs, with Relationships to Statistical Pattern Recognition. In: F.Fogleman Soulie and J.Herault (eds.), Neurocomputing: Algorithms, Architectures and Applications, Berlin: Springer–Verlag, pp. 227–236.

confident but wrong function values, which may or may not be seen as desirable. (Consider the maximum loss that can be obtained on a single example under the two cost functions.)

We first find the gradients of the model's log-probability of a single observation: a label c when the input features are $\mathbf{x}$:

$$\log P(y_c = 1 \mid \mathbf{x}, W) = \log f_c = (\mathbf{w}^{(c)})^\top \mathbf{x} - \log \sum_{k'} e^{(\mathbf{w}^{(k')})^\top \mathbf{x}} \tag{5}$$

$$\nabla_{\mathbf{w}^{(k)}} \log f_c = \delta_{kc}\mathbf{x} - \frac{1}{\sum_{k'} e^{(\mathbf{w}^{(k')})^\top \mathbf{x}}} \mathbf{x}\, e^{(\mathbf{w}^{(k)})^\top \mathbf{x}} \tag{6}$$

$$= (y_k - f_k)\,\mathbf{x}. \tag{7}$$

In the last line we have substituted $y_k = \delta_{kc}$ and the definition of f_k.[2]

To apply stochastic gradient ascent[3] on the log-likelihood we would take a training example $(\mathbf{x}^{(n)}, \mathbf{y}^{(n)})$ and move the weight vector for each class parallel to the input $\mathbf{x}^{(n)}$. The size and sign of the move depends on the disparity between the binary output $y_k^{(n)}$ and the prediction f_k.

Alternatively we could use an optimizer that uses a batch of examples, requiring the gradient

$$\nabla_{\mathbf{w}^{(k)}} \sum_n \log f_{c^{(n)}}^{(n)} = \sum_n (y_k^{(n)} - f_k(\mathbf{x}^{(n)}))\,\mathbf{x}^{(n)}, \tag{8}$$

or minus that value for the gradient of the batch's negative log probability.

2 Redundant parameters and binary logistic regression

For the special case of two classes, the softmax classifier gives:

$$P(y = 1 \mid \mathbf{x}, W) = \frac{e^{(\mathbf{w}^{(1)})^\top \mathbf{x}}}{e^{(\mathbf{w}^{(1)})^\top \mathbf{x}} + e^{(\mathbf{w}^{(2)})^\top \mathbf{x}}} \tag{9}$$

$$= \frac{1}{1 + e^{(\mathbf{w}^{(2)} - \mathbf{w}^{(1)})^\top \mathbf{x}}} = \sigma\big((\mathbf{w}^{(1)} - \mathbf{w}^{(2)})^\top \mathbf{x}\big). \tag{10}$$

The prediction only depends on the vector $(\mathbf{w}^{(1)} - \mathbf{w}^{(2)})$. In logistic regression we normally fit that single vector directly as "$\mathbf{w}$".

We can fit two vectors, as in the softmax classifier formulations. The parameters are not well specified: adding a constant vector $\mathbf{a}$ to both class's weight vectors gives the same predictions. However, if we add a regularization term to our cost function, the cost function for this model will be minimized by a unique setting of the weights.

Alternatively we could set the weight vector for one of the classes to zero, and fit the rest of the parameters. For example, recovering logistic regression by setting $\mathbf{w}^{(2)} = \mathbf{0}$.

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

---

2. It may take some time to digest all of the notation here. We index into the vector of function outputs using the observed label c. We didn't use k for this index so we can use k to index the weight vector we are computing gradients for. We need the gradients for $k \neq c$ as well as $k = c$. The dummy index k' is used because we need to consider all of the weight vectors $\{\mathbf{w}^{(k')}\}_{k'=1}^K$, when computing the gradient with respect to a particular weight vector $\mathbf{w}^{(k)}$.

3. Optimization texts normally talk about gradient *descent* and minimizing functions. Gradient descent on the negative log-likelihood amounts to the same thing. We take a negative step in the direction of the gradient of the negative likelihood, and the two negatives cancel.