MLPR Tutorial[1] Sheet 6

Reminder: If you need more guidance to get started on a question, seek clarifications and hints on the class forum. Move on if you're getting stuck on a part for a long time. Full answers will be released after the last group meets.

---

1. **Classification with unbalanced data**

   Classification tasks often involve rare outcomes, for example predicting click-through events, fraud detection, and disease screening. We'll restrict ourselves to binary classification, $y \in \{0, 1\}$, where the positive class is rare: $P(y\!=\!1)$ is small.

   We are likely to see *lots* of events before observing enough rare $y\!=\!1$ events to train a good model. To save resources, it's common to only keep a small fraction f of the $y\!=\!0$ 'negative examples'. A classifier trained naively on this sub-sampled data would predict positive labels more frequently than they actually occur. For Bayes classifiers we could set the class probabilities based on the original class counts (or domain knowledge). This question explores what to do for logistic regression.

   We write that an input-output pair occurs in the world with probability $P(\mathbf{x}, y)$, and that the joint probability of an input-output pair $(\mathbf{x}, y)$ *and* keeping it (k) is $P(\mathbf{x}, y, k)$. Because conditional probabilities are proportional to joint probabilities, we can write:

$$P(y \,|\, \mathbf{x}, k) \propto P(\mathbf{x}, y \,|\, k) \propto P(\mathbf{x}, y, k) = \begin{cases} P(\mathbf{x}, y) & y = 1 \\ f\, P(\mathbf{x}, y) & y = 0. \end{cases}$$

   a) Show that if:

$$P(y \,|\, \mathbf{x}) \propto \begin{cases} c & y = 1 \\ d & y = 0, \end{cases}$$

   then $P(y\!=\!1 \,|\, \mathbf{x}) = \sigma(\log c - \log d)$, where $\sigma(a) = 1/(1 + e^{-a})$.

   [This exercise should be revision, given Tutorial 4, Q3.]

   b) We train a logistic regression classifier, with a bias weight, to match subsampled data, so that $P(y = 1 \,|\, \mathbf{x}, k) \approx \sigma(\mathbf{w}^\top \mathbf{x} + b)$. Use the above results to argue that we should add $\log f$ to the bias parameter to get a model for the real-world distribution $P(y \,|\, \mathbf{x}) \propto P(y, \mathbf{x})$.

   Have we changed the bias in the direction that you would expect?

   c) We now consider a different approach. We may wish to minimize the loss:

$$L = -\mathbb{E}_{P(\mathbf{x},y)}[\log P(y \,|\, \mathbf{x})] \,.$$

   Multiplying the integrand by $1 = \frac{P(\mathbf{x},\mathbf{y}\,|\,k)}{P(\mathbf{x},\mathbf{y}\,|\,k)}$ changes nothing, so we can write:

$$\begin{aligned} L &= -\int \sum_y P(\mathbf{x}, y \,|\, k)\, \frac{P(\mathbf{x}, y)}{P(\mathbf{x}, y \,|\, k)} \log P(y \,|\, \mathbf{x})\, \mathrm{d}\mathbf{x} \\ &= -\mathbb{E}_{p(\mathbf{x},y\,|\,k)}\left[\frac{P(\mathbf{x}, y)}{P(\mathbf{x}, y \,|\, k)} \log P(y \,|\, \mathbf{x})\right] \\ &\approx -\frac{1}{N} \sum_{n=1}^{N} \frac{P(\mathbf{x}^{(n)}, y^{(n)})}{P(\mathbf{x}^{(n)}, y^{(n)} \,|\, k)} \log P(y^{(n)} \,|\, \mathbf{x}^{(n)}), \end{aligned}$$

   where $\mathbf{x}^{(n)}, y^{(n)}$ come from the subsampled data. This manipulation is a special case of a trick known as *importance sampling*, which we will see later in lectures

1. Q2 is based on a previous sheet by Amos Storkey, Charles Sutton, and/or Chris Williams

in another context. We have converted an expectation under the original data distribution, into an expectation under the subsampling distribution. We then replaced the formal expectation with an average over subsampled data.

Use the above idea to justify multiplying the gradients for $y\!=\!0$ examples by $1/f$, when training a logistic regression classifier on subsampled data.

d) *Hard:* Compare the two methods that we have considered for training models based on subsampled data, giving some pros and cons of each.

2. **Building a toy neural network**

Consider the following classification problem. There are two real-valued features x_1 and x_2, and a binary class label. The class label is determined by

$$y = \begin{cases} 1 & \text{if } x_2 \geq |x_1| \\ 0 & \text{otherwise.} \end{cases}$$

a) Can this function be perfectly represented by logistic regression, or a feedforward neural network without a hidden layer? Why or why not?

b) Consider a simpler problem for a moment, the classification problem

$$y = \begin{cases} 1 & \text{if } x_2 \geq x_1 \\ 0 & \text{otherwise.} \end{cases}$$

Design a single 'neuron' that represents this function. Pick the weights by hand. Use the hard threshold function

$$\Theta(a) = \begin{cases} 1 & \text{if } a \geq 0 \\ 0 & \text{otherwise,} \end{cases}$$

applied to a linear combination of the $\mathbf{x}$ inputs.

c) Now go back to the classification problem at the beginning of this question. Design a two layer feedforward network (that is, one hidden layer with two layers of weights) that represents this function. Use the hard threshold activation function as in the previous question.

Hints: Use two units in the hidden layer. The unit from the last question will be one of the units, and you will need to design one more. You can then find settings of the weights such that the output unit performs a binary AND operation on the two hidden units.

3. **Regression with input-dependent noise:**

In lectures we turned a representation of a function into a probabilistic model of real-valued outputs by modelling the residuals as Gaussian noise:

$$p(y\,|\,\mathbf{x}, \theta) = \mathcal{N}(y;\, f(\mathbf{x};\theta),\, \sigma^2).$$

The noise variance σ^2 is often assumed to be a constant, but it could be a function of the input location $\mathbf{x}$ (a "heteroscedastic" model).

A flexible model could set the variance using a neural network:

$$\sigma(\mathbf{x})^2 = \exp(\mathbf{w}^{(\sigma)\top}\mathbf{h}(\mathbf{x};\theta) + b^{(\sigma)}),$$

where $\mathbf{h}$ is a vector of hidden unit values. These could be hidden units from the neural network used to compute function $f(\mathbf{x};\theta)$, or there could be a separate network to model the variances.

a) Assume that $\mathbf{h}$ is the final layer of the same neural network used to compute f. How could we modify the training procedure for a neural network that fits f by least squares, to fit this new model?

b) In the suggestion above, the activation $a^{(\sigma)} = \mathbf{w}^{(\sigma)\top}\mathbf{h} + b^{(\sigma)}$ sets the log of the variance of the observations.

   i) Why not set the variance directly to this activation value, $\sigma^2 = a^{(\sigma)}$?

   ii) *Hard:* Why not set the variance to the square of the activation, $\sigma^2 = (a^{(\sigma)})^2$?

c) Given a test input $\mathbf{x}^{(*)}$, the model above outputs both a guess of an output, $f(\mathbf{x}^{(*)})$, and an 'error bar' $\sigma(\mathbf{x}^{(*)})$, which indicates how wrong the guess could be.

   The Bayesian linear regression and Gaussian process models covered in lectures also give error bars on their predictions. What are the pros and cons of the neural network approach in this question? Would you use this neural network to help guide which experiments to run?