MLPR Tutorial[1] Sheet 4

Reminders: Attempt the tutorial questions, and ideally discuss them, before your tutorial. You can seek clarifications and hints on the class forum. Full answers will be released.

---

1. **Some computation with probabilities:** It's common to compute with log-probabilities to avoid numerical underflow. Quick example: notice that the probability of 2000 coin tosses, 2^{-2000}, underflows to zero in Matlab, NumPy, or any package using IEEE 64 bit floating point numbers.

   If we have two possible models, M_1 and M_2, for some features, we can define:

   $$\begin{aligned} a_1 &= \log P(\mathbf{x} \,|\, M_1) + \log P(M_1) \\ a_2 &= \log P(\mathbf{x} \,|\, M_2) + \log P(M_2). \end{aligned}$$

   Up to a constant, these 'activations' give the log-posterior probabilities that the model generated the features. Show that we can get the posterior probability of model M_1 neatly with the logistic function:

   $$P(M_1 \,|\, \mathbf{x}) = \sigma(a_1 - a_2) = \frac{1}{1 + \exp(-(a_1 - a_2))}.$$

   Now given K models, with $a_k = \log[P(\mathbf{x} \,|\, M_k)\, P(M_k)]$, show:

   $$\log P(M_k \,|\, \mathbf{x}) = a_k - \log \sum_{k'} \exp a_{k'}.$$

   The $\log \sum \exp$ function occurs frequently in the maths for probabilistic models (not just model comparison). Show that:

   $$\log \sum_k \exp a_k = \max_k a_k + \log \sum_k \exp\left(a_k - \max_{k'} a_{k'}\right).$$

   Explain why the expression is often implemented this way. (Hint: consider what happens when all the a_k's are less than -1000).

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.

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

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.

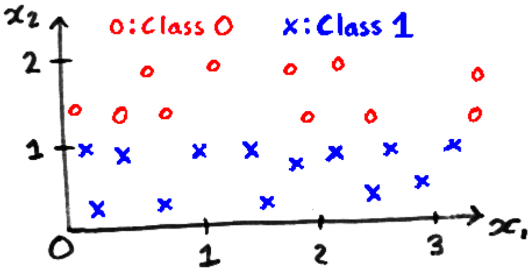
Hints: Use two units in the hidden layer. The unit from the last question will be one of the units, and you will need to design one more. You can then find settings of the weights such that the output unit performs a binary AND operation on the two hidden units.

3. **Learning a transformation:**

The K-nearest-neighbour (KNN) classifier predicts the label of a feature vector by finding the K nearest feature vectors in the training set. The label predicted is the label shared by the majority of the training neighbours. For binary classification we normally choose K to be an odd number so ties aren't possible.

KNN is an example of a *non-parametric* method: no fixed-size vector of parameters is sufficient to summarize the training data. The complexity of the function represented by the classifier can grow with the size of the training data, but the whole training set needs to be stored so that it can be consulted at test time.

a) How would the predictions from regularized linear logistic regression, with $p(y=1\,|\,\mathbf{x}, \mathbf{w}, b) = \sigma(\mathbf{w}^\top\mathbf{x} + b)$, and 1-nearest neighbours compare on the dataset below?



Non-parametric methods can have parameters. We could modify the nearest neighbour rule, by taking a linear transformation of the data $\mathbf{z} = A\mathbf{x}$, and performing nearest neighbours using the new $\mathbf{z}$ features. One loss function, for evaluating possible transformations A, could be the leave-one-out (LOO) classification error. We measure the fraction of errors made on the training set, if each training item has to use another neighbour (not itself) for its classification.

b) Write down a matrix A that would have lower LOO error than using the identity matrix for the dataset above, and explain why it works.

c) Explain whether we can fit the LOO error by gradient descent on the matrix A.

d) Assume that I have implemented some classification method where I can evaluate a cost function c and its derivatives with respect to feature input locations: $\bar{Z}$, where $\bar{Z}_{nk} = \frac{\partial c}{\partial Z_{nk}}$ and Z is an $N \times K$ matrix of inputs.

I will use that code by creating the feature input locations from a linear transformation of some original features $Z = XA$. How could I fit the matrix A? If A is a $D \times K$ matrix, with $K < D$, how will the computational cost of this method scale with D?

You may quote results given in lecture note w5a.