SDMML HT 2016 - Part C Problem Sheet 5

1. (**k-Nearest Neighbours, Curse of Dimensionality**) Consider using a k-NN classifier where the real-valued features are uniformly distributed in the p-dimensional unit cube. Suppose we are interested in estimating the distribution over class labels around a test point x by using neighbours within a hyper-cube centred at x.

   (a) Suppose we wish to use a fraction α of the training data to estimate the distribution over class labels at x. What should be the edge length of this hyper-cube to ensure that it includes on average $\alpha\%$ of the training data? If $p = 10$ and $\alpha = 1\%$, compute the edge length of this hyper-cube. In this scenario, is k-NN a "local" algorithm, i.e. using only local neighbours to x?

   (b) Assuming you have access to say $n = 500$ training data, does it appear reasonable to perform k-NN for large values of k (say $k > 10$)? Explain briefly why or why not.

2. (**1-NN risk in binary classification**) Let $\{(X_i, Y_i)\}_{i=1}^n$ be a training dataset where $X_i \in \mathbb{R}^p$ and $Y_i \in \{0, 1\}$. We denote by $g_k(x)$ the conditional density of X given $Y = k$ and assume that $g_k(x) > 0$ for all $x \in \mathbb{R}^p$, and the class probabilities as $\pi_k = \mathbb{P}(Y = k)$. We further denote $q(x) = \mathbb{P}(Y = 1|X = x)$.

   (a) Consider the Bayes classifier (minimizing risk w.r.t. 0/1 loss $\mathbf{1}\{f(X) \neq Y\}$):

   $$f_{\text{Bayes}}(x) = \underset{k\in\{0,1\}}{\arg\max}\, \pi_k g_k(x).$$

   Write the conditional expected loss $\mathbb{P}[f(X) \neq Y|X = x]$ at a given test point $X = x$ in terms of $q(x)$. [The resulting expression should depend *only* on $q(x)$].

   (b) The 1-nearest neighbour (1-NN) classifier assigns to a test data point x the label of the closest training point; i.e. $f_{\text{1NN}}(x) = y$ (class of nearest neighbour in the training set). Given some test point $X = x$ and its nearest neighbour $X' = x'$, what is the conditional expected loss $\mathbb{P}[f_{\text{1NN}}(X) \neq Y|X = x, X' = x']$ of the 1-NN classifier in terms of $q(x), q(x')$?

   (c) As the number of training examples goes to infinity, i.e. $n \to \infty$, assume that the training data fills the space such that $q(x') \to q(x)$, $\forall x$. Give the limit (as $n \to \infty$) of $\mathbb{P}[f_{\text{1NN}}(X) \neq Y|X = x]$. If we denote by $R_{\text{Bayes}} = \mathbb{P}\left[Y \neq f_{\text{Bayes}}(X)\right]$ and $R_{\text{1NN}} = \mathbb{P}[Y \neq f_{\text{1NN}}(X)]$, show that for sufficiently large n

   $$R_{\text{Bayes}} \leq R_{\text{1NN}} \leq 2R_{\text{Bayes}}\left(1 - R_{\text{Bayes}}\right).$$

3. Recall the definition of a one-hidden layer neural network for binary classification in the lectures. The objective function is L_2-regularized log loss:

$$J = -\sum_{i=1}^{n} y_i \log \hat{y}_i + (1 - y_i)\log(1 - \hat{y}_i) + \frac{\lambda}{2}\left(\sum_{jl}(w_{jl}^h)^2 + \sum_l (w_l^o)^2\right)$$

and the network definition is:

$$\hat{y}_i = s\left(b^o + \sum_{l=1}^{m} w_l^o h_{il}\right), \qquad h_{il} = s\left(b_l^h + \sum_{j=1}^{p} w_{jl}^h x_{ij}\right),$$

with transfer function $s(a) = \frac{1}{1+e^{-a}}$.

(a) Verify that the derivatives needed for gradient descent are:

$$\frac{\partial J}{\partial w_l^o} = \lambda w_l^o + \sum_{i=1}^{n} (\hat{y}_i - y_i) h_{il},$$
$$\frac{\partial J}{\partial w_{jl}^h} = \lambda w_{jl}^h + \sum_{i=1}^{n} (\hat{y}_i - y_i) w_l^o h_{il}(1 - h_{il}) x_{ij}.$$

(b) Suppose instead that you have a neural network for binary classification with L hidden layers, each hidden layer having m neurons with logistic transfer function. Give the parameterization for each layer, and derive the backpropagation algorithm to compute the derivatives of the objective with respect to the parameters. For simplicity, you can ignore bias terms.

4. A **mixture of experts** is an ensemble model in which a number of experts "compete" to predict a label.

Consider a regression problem with dataset $\{(x_i, y_i)\}_{i=1}^n$ and $y_i \in \mathbb{R}$. We have E experts, each associated with a parametrized regression function $f_j(x; \theta_j)$, for $j = 1, \dots, E$ (for example, each expert could be a neural network).

(a) A simple mixture of experts model uses as objective function

$$J(\pi, \sigma^2, (\theta_j)_{j=1}^E) = \sum_{i=1}^{n} \log \sum_{j=1}^{E} \pi_j e^{-\frac{1}{2\sigma^2}\|f_j(x_i;\theta_j) - y_i)\|^2}$$

where $\pi = (\pi_1, \dots, \pi_E)$ are mixing proportions and σ^2 is a parameter.

Relate the objective function to the log-likelihood of a mixture model where each component is a conditional distribution of Y given $X = x$.

(b) Differentiate the objective function with respect to θ_j. Introduce a latent variable z_i, indicating which expert is responsible for predicting y_i, and interpret $\frac{\partial J}{\partial \theta_j}$ in the context of the corresponding EM algorithm. In this context, one needs to use the generalized EM algorithm, where in the M-step gradient descent is used to update the expert parameters θ_j.

(c) A mixture of experts allows each expert to specialize in predicting the response in a certain part of the data space, with the overall model having better predictions than any one of the experts.

However to encourage this specialization, it is useful also for the mixing proportions to depend on the data vectors x, i.e. to model $\pi_j(x; \phi)$ as a function of x with parameters ϕ. The idea is that this **gating network** controls where each expert specializes. To ensure $\sum_{j=1}^{E} \pi_j(x; \phi) = 1$, we can use the softmax nonlinearity:

$$\pi_j(x; \phi) = \frac{\exp(h_j(x; \phi_j))}{\sum_{\ell=1}^{E} \exp(h_\ell(x; \phi_\ell))}$$

where $h_j(x; \phi_j)$ are parameterized functions for the gating network.

The previous generalized EM algorithm extends to this scenario easily. Describe what changes have to be made, and derive a gradient descent learning update for ϕ_j.

5. In this question you will investigate fitting neural networks using the `nnet` library in R. We will train a neural network to classify handwritten digits 0-9. Download files `usps_trainx.data`, `usps_trainy.data`, `usps_testx.data`, `usps_testy.data` from `http://www.stats.ox.ac.uk/~sejdinov/sdmml/data/`.
   Each handwritten digit is 16×16 in size, so that data vectors are $p = 256$ dimensional and each entry (pixel) takes integer values 0-255. There are 2000 digits (200 digits of each class) in each of the training set and test set. You can view the digits with

   ```
   image(matrix(as.matrix(trainx[500,]),16,16),col=grey(seq(0,1,length=256)))
   trainy[500,]
   ```

   Download the R script `nnetusps.R` from the course webpage. The script trains a 1-hidden layer neural network with $S = 10$ hidden units for $T = 10$ iterations, reports the training and test errors, runs it for another 10 iterations, and reports the new training and test errors. To make computations quicker, the script down-samples the training set to 200 cases, by using only one out of every 10 training cases. You will find the documentation for the `nnet` library useful: `http://cran.r-project.org/web/packages/nnet/nnet.pdf`.

   (a) Edit the script to report the training and test error after every iteration of training the network. Use networks of size $S = 10$ and up to $T = 100$ iterations. Plot the training and test errors as functions of the number of iterations. Discuss the results and the figure.

   (b) Edit the script to vary the size of the network, reporting the training and test errors for network sizes $S = 1, 2, 3, 4, 5, 10, 20, 40$. Use $T = 25$ iterations. Plot these as a function of the network size. Discuss the results and the figure.