CSC2515 – Assignment #2

Due: Nov.9, 2004, 2pm at the **START** of class
Worth: 18%
Late assignments not accepted.

1 Adapting Centres in Radial Basis Networks (2%)

In this question you will find the derivatives which would allow you to adapt the centres in a radial basis network.

Recall that a radial basis network is a generalized linear model of the form

$$y = \sum_j w_j h_j(\mathbf{x})$$
$$h_j(\mathbf{x}) = \exp\left[-\alpha_j (\mathbf{x} - \mathbf{z}_j)^\top (\mathbf{x} - \mathbf{z}_j)\right]$$

Assume we are using a Gaussian noise model so that the likelihood becomes squared error:

$$E \propto \sum_n (y_n - \sum_j w_j h_j(\mathbf{x}_n))^2$$

As you know, for fixed α_j and fixed $\mathbf{z}_j$, the maximum likelihood weights under this Gaussian noise can be found by linear regression. But if $\mathbf{z}_j$ and α_j are unknown, we would like to adapt them.

- Compute the gradient $\partial E / \partial \mathbf{z}_k$ of E with respect to $\mathbf{z}_k$, given a finite training set $\{\mathbf{x}_n, y_n\}$ and the current values of $\mathbf{z}_j, \alpha_j$ for all j.

- Compute the gradient $\partial E / \partial \log \alpha_k$ of E with respect to $\log \alpha_k$, given a finite training set $\{\mathbf{x}_n, y_n\}$ and the current values of $\mathbf{z}_j, \alpha_j$ for all j.

2 Pseudo-Bayesian Linear Regression (4%)

In this question you will dabble in Bayesian statistics and extend the probabilistic model formulation to include not only the data but also the weights.

Consider the following conditional model of a scalar output y given some inputs $\mathbf{x}$:

$$\begin{aligned} p(y|\mathbf{x}) &= \int_{\mathbf{w}} p(y|\mathbf{x}, \mathbf{w}) p(\mathbf{w}) d\mathbf{w} \\ p(y|\mathbf{x}, \mathbf{w}) &= \mathcal{N}(y; \mathbf{w}^\top \mathbf{x}, b) \\ p(\mathbf{w}) &= \mathcal{N}(\mathbf{w}; \mathbf{0}, a\mathbf{I}) \end{aligned}$$

Here the weight prior $p(\mathbf{w})$ is a Gaussian with mean zero and covariance $a\mathbf{I}$ ($\mathbf{I}$ is the identity matrix) and the data model is a conditional linear-Gaussian with mean $\mathbf{w}^\top \mathbf{x}$ and variance b.

Notice that the model generates a value of y given $\mathbf{x}$ by stochastically selecting a set of weights and using these to linearly combine the components of $\mathbf{x}$. Given this generative model, the correct way to make predictions on future data given a finite training set is to *average* over all possible settings of the weights, letting each one make a prediction, and weight the predictions in the average by the posterior of the weights given the training data.

- Find the posterior distribution over weights $p(\mathbf{w}|\{\mathbf{x}_n, y_n\}, a, b)$ given a finite training set $\{\mathbf{x}_n, y_n\}$ and the variances a, b. Hint: your answer should be another Gaussian distribution, now over $\mathbf{w}$. To do this, you can either form the joint Gaussian $p(y, \mathbf{w}|\mathbf{x})$, marginalize out the weights to form $p(y|\mathbf{x})$ and then use Bayes rule to find $p(\mathbf{w}|\mathbf{x}, y)$ or you can try completing the square, using vector and matrix operations and the apply the conditioning formulas from the probability and statistics review notes.
- Show that the ridge regression weights, which minimize the cost $\lambda \mathbf{w}^\top \mathbf{w} + \sum_n (y_n - \mathbf{w}^\top \mathbf{x}_n)^2$ are equal to the mean (and mode) of the posterior above.
- Give the ridge regression penalty λ in terms of the variances a, b above.
- Integrate out the posterior over the weights, and work out the conditional predictive distribution over a new output y_{new} given a new test input $\mathbf{x}^{new}$ and the training set:

$$p(y_{new}|\mathbf{x}^{new}, \{\mathbf{x}_n, y_n\}, a, b) = \int_{\mathbf{w}} p(y^{new}|\mathbf{x}^{new}, \mathbf{w}) p(\mathbf{w}|\{\mathbf{x}_n, y_n\}, a, b) d\mathbf{w}$$

It is permissible to consult external sources (textbooks, research reports on the web, notes from Bayesian stats courses, etc) when working out the answer to this question but it is not permissible to copy an answer directly from another source without understanding it and working it out for yourself.

3 Regularizing Linear Mixtures of Experts (2%)

In class I did not tell you anything about how to regularlize mixtures of experts models. This question asks you to work out the details for one method and to suggest some more.

Recall that the mixture of experts model uses a logistic regression gate to compute $p(j|\mathbf{x})$ given the input vector $\mathbf{x}$, and then stochastically selects an expert according to these probabilities. With linear experts and a multivariate output $\mathbf{y}$, each expert's output is given by a linear-Gaussian model: $p(\mathbf{y}|j,\mathbf{x}) = \mathcal{N}(\mathbf{y};\mathbf{U}_j\mathbf{x},\Sigma)$. The final output distribution is a mixture of all the expert models, weighted by the (conditional) prior the gate places on them given the input: $p(\mathbf{y}|\mathbf{x}) = \sum_j p(j|\mathbf{x})\mathcal{N}(\mathbf{y};\mathbf{U}_j\mathbf{x},\Sigma)$. The model is trained to maximize the average log conditional likelihood of the training outputs given the training inputs.

- One obvious way to regularlize a linear mixture of experts model using squared error is to do *ridge regression* at the experts instead of maximum likelihood regression. This means the objective function would become:

$$\ell^{new} = \sum_n \log \sum_j p(j|\mathbf{x}_n)\mathcal{N}(\mathbf{y}_n;\mathbf{U}_j\mathbf{x}_n,\Sigma) + \lambda \sum_{ijk} U_{ijk}^2$$

  where U_{ijk} is the weight from input i to output k in expert j.
  Recall that the gradient with respect to an expert's weights $\mathbf{U}_j$ in the original linear MOE was:

$$\partial\ell^{old}/\partial\mathbf{U}_j = \Sigma^{-1}\sum_n p(j|\mathbf{x}_n,\mathbf{y}_n)\,(\mathbf{y}_n - \mathbf{U}_j\mathbf{x}_n)\,\mathbf{x}_n^\top$$

  which uses the *posterior* probability of each expert:

$$p(j|\mathbf{x}_n,\mathbf{y}_n) = \frac{p(j|\mathbf{x}_n)p(\mathbf{y}_n|j,\mathbf{x}_n)}{\sum_k p(k|\mathbf{x}_n)p(\mathbf{y}_n|k,\mathbf{x}_n)}$$

- When we use ridge regression, the new likelihood function will, of course, have different gradients with respect to the expert weights. Derive the new gradient $\partial\ell^{new}/\partial\mathbf{U}_j$ with respect to an expert's weights $\mathbf{U}_J$ under the ridge regression likelihood above. Assume the gate is fixed, so that the functions $p(j|\mathbf{x})$ are known and do not change. Assume also that the output covariance Σ is known and fixed and the same for all experts. Notice that in this question I am asking you to consider a multivariate output $\mathbf{y}$ instead of just a scalar output and so each expert has a matrix of weights instead of just a vector.

- Suggest three other, different ways of regularizing a linear mixture of experts architecture to prevent overfitting. One way should regularlize the gate, one way should regularlize the individual experts (differently than the ridge regression you worked out above), and one way should regularize the whole architecture.

4 Fully Observed Trees (10%)

In this assignment you will train a fully observed tree model on word-document vectors taken from USENET articles.

4.1 What to do

- Using the data provided in `a2newsgroups.mat`, train a fully observed tree model with the optimal structure. Training the model structure should give you an *undirected* tree. Using this, you can pick a root node arbitrarily and form a *directed tree.*
- The matrix `documents` contains one column per USENET article.
  Each of the 100 rows is a binary variable indicating whether a certain keyword appears in the posting or not. The cell array `wordlist` has the 100 words corresponding to these rows. The data has 100 (binary) features and 16242 training cases. (In case you are interested, the array `newsgroup` tells you the toplevel newsgroup category of each posting, `1=comp.*, 2=rec.*, 3=sci.*, 4=talk.*`, although we won't use it in this assignment.)

4.2 What to hand in

- Any *compact, clear and unambiguous* representation of the optimal (undirected) tree structure found by your algorithm. For example, you might print out all 99 pairs of words `(word_i--word_j)` corresponding to edges chosen by the tree learning algorithm. Or you might want to draw a picture of the graph (see below).
  DO NOT hand in lists of numbers corresponding to the indices of pairs of features chosen – convert everything back to words using the `wordlist` cell array. Since the order of the words is arbitrary, the number of the word index is meaningless, and printing it out tells someone else nothing about your model.
- After finding the optimal tree structure for the 100 variables, you may choose any node you wish as the root and form a directed graph. Different choices of root node will result in different output when you visualize a directed tree, but of course the undirected model structure always remains the same. Chose a root word. Hand in a *compact, clear and unambiguous* representation of a directed tree structure created by directing all arcs in the optimal structure away from a your chosen root. For example, you might print out a breadth-first list of directed links `(word_i->word_j)` in the tree. Alternately, you might want to draw a picture (see below).
- A histogram (with at least 100 bins) of the log likelihoods of the training cases under your optimal model. Notice that these likelihoods are the same no matter how you chose to convert your undirected tree into a directed tree.
  Compute and print out the min,max,mean and median log likelihood.
  Which training case (give its number) has the worst (lowest) log likelihood?
  Print out the list of keywords appearing in this posting.

4.3 Reminders

- Be careful when computing the mutual information weights for pairs of variables that have zero counts for some of their joint settings. In this case the mutual information should be zero, but a careless calculation will lead to division by zero or log of zero errors in the code.
- Amongst other things, your code will need to:
  - Compute the marginal count for every feature being on or off summed across all documents.
  - Compute the joint counts for every pair of features being on together, off together, on-off or off-on, summed across all documents.
  - Convert these counts into mutual information weights.
    Be careful to do this properly for joint counts which are zero!
  - Find the best spanning tree over the features given these weights. Depending on how you compute the weights, you will either need a minimum or maximum weight spanning tree.
    (This will in turn affect how you call `mwst.m` if you choose to use it.)
  - Select a root for the tree and direct all edges away from it to get a directed graph.

4.4 Some implementation help

- To save you some trouble, I've written the function `mwst.m` for you, which finds minimum or maximum weight spanning trees given a weight matrix.
- Here is a trick for computing the 2 by 2 joint count table of two binary column vectors `v1` and `v2`:

```
counts12 = reshape(hist(v1+2*v2,0:3),2,2);
```

  but make sure you understand it before using it!
- The data is almost a megabyte (in MATLAB , stored as a sparse matrix), so be careful about making copies of it in memory. You should be able to do everything you need without making another (sparse or nonsparse) copy of the whole dataset.

4.5 Automatic Graph Layout

- If you want to be really fancy, you can try using the function `dottree.m` and the automatic graph layout programs `dot` and `neato` to draw pictures of your final structures. It is very cool looking, but you will not get any more or any fewer marks for using this method or a simpler method to display your results.
  Caution: do not waste too much time trying to get this to work, since it doesn't get you any more marks. It is just for fun. Get the rest of the assignment finished first.
- Call `dottree(tree,'fname',wordlist)` and it will write out two files, `fname.u` and `fname.d`.
  Now call the `neato` and `dot` programs to convert these files into pictures.
- Information and downloads for `dot` and `neato` are available at `http://www.graphviz.org/`. The programs are already installed in `/local/bin/` on CDF (and in `/pkgs/graphvis/linux/bin/` on CSLAB if you want to use them in your own research later).

IMPORTANT: please do your computing on CDF or your own machine and not on CSLAB machines since they have limited MATLAB licenses.