Machine Learning and Pattern Recognition Assignment

Instructors: Iain Murray and Chris Williams

Due: 4pm Tuesday 18 November, 2014 *[Question 4c updated 2014-11-06]*

This assignment is out of 100 marks and forms 20% of your final grade.

Assessed work is subject to University regulations on academic conduct:
`http://tinyurl.com/UoEconduct`
Do not show your code, answers or write-up to anyone else.
Never copy-and-paste material that's not your own into your assignment and edit it.
If you use any publicly-available functions that are not provided with the core of Matlab (as installed on DICE) or in Netlab, you must state where they are from, and explain what they do in your own words. Using any code from other students taking the class is not acceptable.

Late submissions: The School of Informatics policy is that late coursework normally gets a mark of zero. See `http://tinyurl.com/edinflate` for exceptions to this rule. Any requests for extensions should go to the ITO, either directly or via your Personal Tutor.

Submission instructions: You should submit your answers on paper to the ITO office in Appleton Tower by the deadline.

Provide relevant code fragments and plots within your answers where appropriate.

1 Regression

In this question we will consider data with a two-dimensional input space $\mathbf{x}$, and a one-dimensional output y. The data can be found in `regqdat.mat` as part of the tarball `regq.tar`. This file contains a training set with inputs `x_trn` (`num_trn` = 200 datapoints) and corresponding outputs `y_trn`, and also a test set `x_tst` and `y_tst` (with `num_tst` = 200 datapoints).

The data is known to be obtained from the underlying function by adding Gaussian noise of standard deviation `std_n` $= 0.05$.

When you unpack the tarball `regq.tar` and run the script `regq.m` in matlab you will get some data loaded and structures setup which will be useful for answering the questions below.

1. (Data visualization, 5 marks)
   (a) Plot the data in `x_trn`. Comment on any special aspects of the data distribution. Also plot the data in Plot the data in `x_tst` and comment on any differences with `x_trn`.
   (b) Plot the input-output data in 3-d, and comment on the form of the underlying function.

   Hints: the functions `scatter` and `scatter3` may be useful. Also you can rotate 3-d plots in 3-d by activating the "Rotate 3D" option under Tools on the matlab figure window.
2. (Linear regression, 12 marks) Construct the design matrix `X` of the training data:
   ```
   num_trn = size(x_trn,1);
   X = [ones(num_trn, 1) x_trn];
   ```
   We will consider a linear regression model of the form $f(x) = w_0 + w_1 x_1 + w_2 x_2$.
   (a) Solve for the maximum likelihood parameters $\hat{\mathbf{w}}$ of the model and report their values.
   (b) Calculate and report the mean squared error (MSE) of your model on the training and testing sets. Also report the MSE of a dumb model $f(\mathbf{x}) = w_0$; in this case $\hat{w}_0$ is just the mean of `y_trn`.

(c) Report the estimate for the noise variance obtained from the residuals of the linear regression model, and compare this with the true noise variance. Comment on any differences.

3. (Bayesian linear regression, 10 marks) Consider the linear regression model $f(\mathbf{x}|\mathbf{w}) = w_0 + w_1 x_1 + w_2 x_2$. We now assume that $\mathbf{w} \sim \mathcal{N}(\mathbf{0}, I_3)$, i.e. it is drawn from a 3-d multivariate Gaussian distribution with zero mean and the identity covariance matrix.
   (a) Consider the input point $\mathbf{x}^* = (1,\ 2)^T$. Calculate the mean and variance of the corresponding output $f(\mathbf{x}^*)$, treating the weights as random variables.
   (b) Draw three random weight vectors from $\mathcal{N}(\mathbf{0}, I_3)$. Provide surface plots $f(\mathbf{x}|\mathbf{w})$ corresponding to each weight vector, and comment on the form of the prior over functions for the linear regression model.
   Hints: Use `randn` to generate pseudo-random Gaussian random variables. Use `meshgrid` to generate a rectangular grid of (x, y) values, and `surf` to make a surface plot.

4. (RBF regression, 23 marks). We will now use a mapping from 2-d into 25 dimensions via a set of Gaussian radial basis functions (RBFs) on a 5×5 grid.
   RBFs are implemented in Prof. Ian Nabney's Netlab toolbox[1]. This toolbox is provided in the tarball `regq.tar`. See especially the functions `rbf` and `rbffwd`. Make sure that you read through the documentation and online help (e.g. type `help rbf` at the matlab prompt) so you know what the various functions do.
   The 5×5 grid of RBFs is set up and visualized using the netlab functions below (this code is available in the file `regq.m`):

```
gridside = 5; % create 5x5 grid of RBFs
rbf_net = create_rbf(gridside);
%
% create grid for plotting
%
[gridx1, gridx2] = meshgrid(-2:.05:2, .5:.05:3.5);
grid_dim = size(gridx1);
grid_phi = eval_rbf_bases(rbf_net, [gridx1(:), gridx2(:)]);
%
% plot the basis functions
%
figure
for ii = 1:gridside^2
    surf(gridx1, gridx2, reshape(grid_phi(:,ii), grid_dim));
    hold on
end
hold off
```

   First of all the provided function `create_rbf` is used to set up a RBF network, set its centres to a grid of locations reasonable for the $\mathbf{x}$-space of the data, and to scale the width of each RBF to be the mean distance of each centre to its nearest neighbour. You do not need to adjust these settings.
   Then a call to the provided function `eval_rbf_bases` is made; this evaluates the RBF basis functions at a $M \times 2$ matrix of data points, where each row is a 2-d point, and there are M data points.
   If you type `rbf_net` at the matlab prompt after running `create_rbf` you will get information about this structure. Do a `help rbf` at the matlab prompt to learn more about what these fields mean.
   To do Bayesian linear regression with the RBF basis we need to specify a prior on the hidden-to-output weights of the RBF network (assuming that the RBF centres and widths are fixed). These parameters are called `rbf_net.w2` for the RBFs, and `rbf_net.b2` for the bias weight (corresponding to w_0 for linear regression). In netlab, the weight prior is assumed to have the form $\mathcal{N}(\mathbf{0}, \alpha^{-1} I)$, with the field `rbf_net.alpha` corresponding to α.

1. `http://www1.aston.ac.uk/eas/research/groups/ncrg/resources/netlab/`

Recall that for Bayesian linear regression with a basis[2] $\boldsymbol{\phi}(\mathbf{x})$, we have that

$$y(\mathbf{x}) = \mathbf{w}^T\boldsymbol{\phi}(\mathbf{x}) + \text{noise}$$

where the noise is iid for each datapoint, with distribution $\mathcal{N}(\mathbf{0}, \sigma_\eta^2)$. With a Gaussian $\mathcal{N}(\mathbf{0}, \alpha^{-1}I)$ prior on the weights, the posterior is given by:

$$\begin{aligned} p(\mathbf{w}|\Phi, \mathbf{y}, \sigma_\eta^2) &= \mathcal{N}(\mathbf{w}|\mathbf{w}_N, V_N) \\ \mathbf{w}_N &= \frac{1}{\sigma_\eta^2} V_N \Phi^T \mathbf{y}, \\ V_N &= \sigma_\eta^2(\alpha\sigma_\eta^2 I + \Phi^T\Phi)^{-1}, \end{aligned}$$

where Φ is the design matrix. Also the predictive distribution at a test point $\mathbf{x}^*$ is given by

$$p(y^*|\mathbf{x}^*, \mathcal{D}) \sim \mathcal{N}(\mathbf{w}_N^T\boldsymbol{\phi}^*, (\boldsymbol{\phi}^*)^T V_N \boldsymbol{\phi}^* + \sigma_\eta^2),$$

where $\boldsymbol{\phi}^*$ is the vector of basis functions evaluated at $\mathbf{x}^*$.
We assume the `rbf_net.w2` and `rbf_net.b2` parameters are drawn from a $\mathcal{N}(\mathbf{0}, I_{26})$ distribution, corresponding to setting `rbf_net.alpha` to 1 (as set in the `create_rbf` function). (This is equivalent to setting the α parameter in the weight prior described above to 1.)

(a) Draw three random parameter vectors from $\mathcal{N}(\mathbf{0}, I_{26})$. Plot the corresponding regression surface for each set of parameters. Comment on the form of the prior over functions obtained for this RBF network, and say if you think looks reasonable for the data you have been given.

(b) We are using a weight prior $\mathcal{N}(\mathbf{0}, \alpha^{-1}I)$ with $\alpha = 1$. What effect would increasing α have on the form of the functions drawn?

(c) Obtain the *maximum a posteriori* (MAP) value of the parameters, as shown below (this code is also available in the script `regq.m`)

```
D = gridside^2 + 1;  % number of parameters (incl const term)
trn_phi = eval_rbf_bases(rbf_net, x_trn);
trn_Phi = [ones(num_trn, 1) trn_phi]; % design matrix
w_rbf_map = (trn_Phi'*trn_Phi + rbf_net.alpha*std_n^2*eye(D))\trn_Phi'*y_trn;
rbf_net.w2 = w_rbf_map(2:end); % store values back to model
rbf_net.b2 = w_rbf_map(1);
```

Calculate and report the mean squared error (MSE) of your model on the training and testing sets using the MAP parameters.

(d) Calculate and report the mean squared error (MSE) of your model on the training and testing sets using the maximum likelihood parameters (which can be obtained by a simple modification of the code above). Comment on any differences of these errors with the MAP values, and also make a comparison with the linear regression results.

(e) Report the maximum likelihood estimate for the noise variance obtained from the residuals, and compare this with the true noise variance. Comment on any differences.

(f) It is particularly interesting to look at the predictive variance of the RBF model. Make a plot of the predictive variance in the domain $x_1 = [-4, 4]$ and $x_2 = [-1, 5]$, and comment on the structure you see.

The second part of the assignment starts on the next page...

2. In the equations below the bias weight is included along with all of the others in the basis.

2 Game prediction/ranking and logistic regression

In this part you'll model data from the KGS Go server, `http://www.gokgs.com/`, an online service where millions of two-player games are played every year between hundreds of thousands of users. The service models and reports the ability of each player. Most players choose to play with those of similar ability, so most games are closely-matched. It is hard to beat the baseline prediction that the probabilities of each user winning any game is a half.

The data: A small subset of data has been selected, biased towards games with new users. These games were not all evenly matched, while the system and players worked out how good they were relative to everyone else. In the real system, adapting to new users quickly is important, so performing well on subsets like this one is important.

Training data `train_games.txt` has $N = 319$ rows, with three columns: 1) p_1, ID of player 1; 2) p_2, ID of player 2; 3) y, did player 1 win (as 1 or 0)? The data has been pre-processed to make the player ID's of the D different players take on integers from $1 \ldots D$. Test data, `test_games.txt` is in the same format, with $M = 115$ rows.

In the game, player one plays with white pieces and player two plays with black pieces. It's not necessarily obvious how this knowledge would change predictions. Black plays first, but the scoring system is meant to compensate for this advantage. (Some of you may know that the game of Go has a handicap system, which can give a further advantage to the black player. Handicapped games have been omitted from the data provided however.)

Predicting game outcomes as classification: Given a setting of players one and two, (p_1, p_2), predicting whether p_1 won the game ($y = 1$) or not ($y = 0$) is a classification task. We could consider passing the $(p_1^{(n)}, p_2^{(n)}, y^{(n)})$ training tuples to any classification model, and using the model to predict future outcomes. The fitted parameters within the classifier might (or might not!) have an interesting interpretation that tells us something about the players.

Naive Bayes is a simple classifier. Given input features $\mathbf{p} = (p_1, p_2)$ and a binary 'class label' y indicating whether p_1 won, we can write down a joint model of a row of data as follows:

$$P(y, p_1, p_2) = P(y)\, P(p_1 \,|\, y)\, P(p_2 \,|\, y). \tag{1}$$

To generate data for a new game, the model first decides whether the first player p_1 won the game (without knowing who that player is). After generating y, the model then chooses the identities of the two players independently. Specifying the model requires probabilities such as $P(p_2 = 65 \,|\, y = 0)$: "given that p_1 lost, but not the identity of that player, what's the probability that p_2 was player 65?"

We can estimate all of the required probabilities from the training data:

$$\hat{P}(y) = \frac{n_y + \alpha}{N + 2\alpha}, \qquad \hat{P}(p_i \,|\, y) = \frac{n_{y,i,p_i} + \alpha}{n_y + \alpha D},$$

where in the N training items, n_y is the number of times the third column is set to y, and n_{y,i,p_i} is the number of times column i is set to p_i when the third column is y. The counts are smoothed, preventing estimates of zero, with a positive constant α. The conditional distributions are likely to be uneven so we picked a fairly small, 'responsive' value of $\alpha = 0.1$. We know that $P(y = 1)$ is likely to be close to a half, so we could have picked a larger α to estimate the class probabilities. However, there is a lot of data for that estimate, so choosing α is unlikely to be as important.

1. **The Zermelo/Bradley–Terry model (10 marks)**

   (a) **Game prediction as Logistic regression:** Zermelo (1929) proposed a model where the probability of player p_1 winning a game against player p_2 is given by:

   $$P(p_1 \text{ beats } p_2) = \frac{u_{p_1}}{u_{p_1} + u_{p_2}}, \tag{2}$$

   where u_{p_1} and u_{p_2} give the underlying, positive-valued abilities of the two players. An equivalent model has been rediscovered several times; Bradley and Terry (1952) wrote the most cited paper on this model.

We can reparameterize the positive-valued ability of player d as an unconstrained parameter, w_d, by setting $u_d = \exp(w_d)$. The parameters for all D players being modelled can be put into a single $D \times 1$ vector, $\mathbf{w}$.

Show that the model in (2) can be written as a logistic regression model:

$$P(y=1 \mid \mathbf{x}) = \frac{1}{1+\exp(-\mathbf{w}^\top \mathbf{x})}, \tag{3}$$

where $y=1$ if player p_1 beats player p_2, and $\mathbf{x}$ is a feature vector encoding that players p_1 and p_2 were playing, and that y should indicate whether p_1 won. Your explanation should include a description of how to construct the feature vector $\mathbf{x}$ from (p_1, p_2).

(b) **Comparing and contrasting the approaches:**

Why do the models discussed so far, based on Naive Bayes and logistic regression, use different input feature representations, $\mathbf{p} = (p_1, p_2)$ and $\mathbf{x}$? Discuss whether it might work to run logistic regression with two inputs $\mathbf{p} = (p_1, p_2)$ and/or Naive Bayes with feature vector $\mathbf{x}$.

Show that the logistic regression model treats p_1 and p_2 symmetrically. That is, the probability of winning a game against a particular player doesn't depend on whether you are 'player one' (white) or 'player two' (black) in the pairing. How could the model be modified so that it could model an advantage/disadvantage in being player p_1 (white)?

Give any advantages of disadvantages that you see of each of the two approaches over the other. In particular, explain whether one of the models is more prone to players 'gaming the system'. That is, if player A repeatedly plays and wins against a very weak player, is one of the classifiers more prone than the other to predict that player A will win all of their future games, regardless of their opponent?

2. **Fitting and testing the models (40 marks)**

(a) **Naive Bayes:** The Matlab function `game_naive_bayes_demo.m` fits the naive Bayes model to the training set as described above, and returns probabilistic predictions of whether p_1 wins in each game in the test set. That is, it returns a vector of M probabilities, $p(y^{(m)} = 1 \mid p_1^{(m)}, p_2^{(m)})$ for each row m in the test set.

Obtain the probabilistic predictions by running the code, and create hard predictions, by thresholding the probabilities at 0.5. Report the classifier's accuracy: the fraction of test games correctly predicted. Also report the mean log probability of the true test y labels under the model's predictive probabilities.

A good answer will report standard errors on these test scores, which are noisy estimates of the scores that would be obtained if we had access to an infinite test set.

Does the model perform better than the simplest baseline: predicting $P(y=1 \mid \mathbf{p}) = 0.5$, regardless of who is playing?

(b) **Logistic regression:** The Matlab function `game_loglike.m` evaluates the log-likelihood of the logistic regression model (3), and its partial derivatives with respect to the weights $\mathbf{w}$. As its documentation states, this function takes the original features $\mathbf{p}$ directly, rather than requiring you to convert them into the feature vector $\mathbf{x}$ discussed in question 1a.

Use the provided generic gradient-based optimization routine, `minimize.m` by Carl Rasmussen. *Maximize* the likelihood, by writing a `negative_loglike.m` function and minimizing it. If you wish, you may use another minimizer instead, for example one that comes with Netlab.

Numerical problems in the provided code mean that the optimizer doesn't actually find parameters that completely maximize the log-likelihood. Explain a way to change the numerically fitted weights that would make the true log-likelihood slightly higher. (Hint: consider the weight for a player who won all their training games.)

Use the parameters fitted by the minimize routine to create a vector of test predictions $p(y^{(m)}=1 \mid p_1^{(m)}, p_2^{(m)})$, under the logistic regression (Zermelo/Bradley–Terry) model. How does the model's accuracy and mean log probability compare to Naive Bayes on the test set? Also make both models 'predict' the labels on the training set, and compare their performances on that set. Briefly comment on what these results illustrate.

(c) **Regularization:** We can limit the size of the weights $\mathbf{w}$ by adding $\lambda\mathbf{w}^\top\mathbf{w}$ to the negative log-likelihood.

What predictions result from the weights fitted with this regularization in the extreme limits where λ is very small and very large?

Create a function to compute this penalized negative log-likelihood and its gradients. You should check that your gradients are consistent with finite differences of the function values. You may wish to use the provided `checkgrad.m` by Carl Rasmussen, as used in `game_loglike_check.m` to check the log-likelihood routine. Alternatively you could use Netlab's `gradchek` routine.

We have determined (so that you don't have to) that $\lambda = 0.5$ is a reasonable value. We used 20-fold cross-validation[1] on the training set. Fit the regularized model with this setting, $\lambda = 0.5$. How does applying this regularization perform on the test set compared to the maximum likelihood fit?

3. ***Entirely optional:*** **Hierarchical logistic regression and MCMC** ***(0 marks)***

We will provide an answer to this question, and feedback on any attempts. However, you won't gain any extra credit for it: there is no penalty for not submitting an answer to this part.

A hierarchical model says that the weights are Gaussian distributed, but with unknown variance:

$$P(\log\lambda) = \mathrm{Uniform}[\log\lambda;\, a, b]$$
$$P(\mathbf{w}) = \mathcal{N}(\mathbf{w}; 0, \tfrac{1}{2\lambda}I) = \left(\tfrac{\lambda}{\pi}\right)^{D/2} \exp(-\lambda\mathbf{w}^\top\mathbf{w}),$$

with $P(y=1 \mid \mathbf{w}, \mathbf{x})$ as in Equation (3). While not strictly well defined, it's common to take the 'improper' limit, $a \to -\infty$ and $b \to \infty$. This limit prefers no single value of $\log\lambda$ over another, and does not restrict its range. (Almost all of the prior mass is on extreme values, which may or may not matter in practice.)

The log-posterior of this model, up to a constant, is the log-likelihood $L(\mathbf{w})$, plus a term for the log prior:

$$\log P(\mathbf{w}, \log\lambda \mid \text{data}) = L(\mathbf{w}) - \lambda\mathbf{w}^\top\mathbf{w} + \tfrac{D}{2}\log\lambda + \text{const.} \tag{4}$$

(a) What is the largest log-likelihood that any model can have given a training set of N binary outcomes? What is the log-likelihood of zero weights, $\mathbf{w} = \mathbf{0}$? Hence show that the log-posterior above is globally maximized by setting the weights to zero and making λ infinite. (Hence the need for cross validation above when setting λ. We could restrict the prior range $\log\lambda \in [a, b]$, but then how would we set *that*?)

(b) Use the provided `slice_sample.m` Markov chain Monte Carlo routine to approximately sample from the hierarchical posterior in (4) for 1,000 iterations. Report any choices that you needed to make. Plot λ against iteration number. Does the sampler seem to have reached steady-state behaviour? Does the value of λ found by cross-validation seem reasonable?

Hint: in your code, put $\mathbf{w}$ and $\log\lambda$ into a single vector `theta`, and write a function to evaluate the log posterior (up to a constant) of this vector. Give the slice sampling routine a function handle to this routine.

1. See the model selection lecture and references. We will provide example code with the assignment solutions.

(c) Explain how to use the samples of $\mathbf{w}$ to predict the game outcomes on the test set. Compare the result of sampling to the fitted regularized model.

(d) Explain how to use the samples to find the probability that player 1 is better than player 2, that is $P(w_1 > w_2)$. Can any of the fitted classifiers answer this question?

Reminder: all of part 3 above, on the hierarchical model and MCMC (in small type) is entirely optional and carries no credit.

3 Notes on MATLAB

- **Remember, there are only a limited number of licences for MATLAB. After you have finished using MATLAB, quit from the MATLAB session so that others can work.**
- Under the Resources heading on the PMR page `http://www.inf.ed.ac.uk/teaching/courses/pmr/` there are a number of MATLAB tutorials listed.
- You can find out more about most MATLAB functions by typing `help` followed by the function name. Also, you can find the `.m` file corresponding to a given function using the `which` command.
- `close all` closes all the figures. It helps if things get cluttered.
- Read about plots in the "Introduction to MATLAB" linked from the PMR homepage. Recall that the current figure can be saved as a PDF file `myplot.pdf` using `print -dpdf myplot.pdf`.

References

R. A. Bradley and M. E. Terry. Rank analysis of incomplete block designs: I. The method of paired comparisons. *Biometrika*, 39(3/4):324–345, 1952.

E. Zermelo. Die Berechnung der Turnier-Ergebnisse als ein Maximumproblem der Wahrscheinlichkeitsrechnung. *Mathematische Zeitschrift*, 29(1):436–460, 1929.