UNIVERSITY OF CAMBRIDGE

MPhil in Machine Learning, Speech and Language Technology
2015-2016

N.B. Handed in 2:30pm Friday. but Katherine on Leave.

MODULE COURSEWORK FEEDBACK

Student Name: Max Chamberlin

Module Title: Intro to ML

CRSiD: MEC68

Module Code: MLSALT 1

College: Clare

Coursework Number: 2

I confirm that this piece of work is my own unaided effort and conforms with the Department of Engineering guidelines on plagiarism

I declare the word count for this piece is: —

Student's Signature: MaxChamberlin.

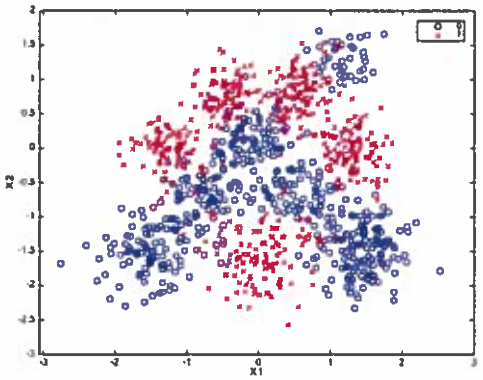
Date Marked:

Marker's Name:

This piece of work has been completed to a standard which is *(please give mark as appropriate)*:

Marker's Comments:

Question A)



A classifier with a linear boundary will do very poorly. The data points are not linearly separable and what's more: any boundary that classifies most of the positive points correctly will also classify many of the negative points incorrectly.

Question B)

I created a function to split the data into test and training sets. It works by randomly permuting the matrix Xy (This is just the matrix of inputs and corresponding outputs). Once this has been done it returns the first 20% of the data as the test set and the last 80% as the training set. We want to randomise the data because we cannot be sure the data was input in a random fashion.

```
% A function to randomly partition data into test-set and training-set
function [Test_X,Test_y,Train_X,Train_y]= partition(X,y)
        Xy=[X,y];
        [l,w]= size(X);
        Xy_rnd = Xy(randperm(l),:);  % random permutation of indeces of Xy
        p= round (0.2*l);            % Take the first 20% of the data as Test
        Test_X= Xy_rnd(1:p,1:w);      Test_y= Xy_rnd(1:p,w+1);
        Train_X=Xy_rnd(p+1:l,1:w);    Train_y=Xy_rnd(p+1:l,w+1);
end
```

Question C)

$$\begin{aligned}\mathfrak{L}(\beta) &= \ln P(\mathbf{y}|\mathbf{X},\beta) = \sum_n \ln\, \mathrm{P}(y^{(n)}|\tilde{\mathbf{x}}^{(n)},\beta) \\ &= \sum_n [y^{(n)} \ln\, \sigma(\beta^{\mathrm{T}}\tilde{\mathbf{x}}^{(n)}) + (1-y^{(n)}) \ln\, \sigma(-\beta^{\mathrm{T}}\tilde{\mathbf{x}}^{(n)})]\end{aligned}$$

Taking derivatives, and using $\frac{\partial \ln \sigma(z)}{\partial z} = \sigma(-z)$, and defining $z_n \stackrel{\text{def}}{=} \beta^{\mathrm{T}}\tilde{\mathbf{x}}^{(n)}$, we have:

$$\begin{aligned}\frac{\partial \mathcal{L}(\beta)}{\partial \beta} &= \sum_n [y^{(n)}\sigma(-z_n)\tilde{\mathbf{x}}^{(n)} - (1-y^{(n)})\sigma(z_n)\tilde{\mathbf{x}}^{(n)}] \\ &= \sum_n [y^{(n)}(\sigma(-z_n)+\sigma(z_n))\tilde{\mathbf{x}}^{(n)} - \sigma(z_n)\tilde{\mathbf{x}}^{(n)}]\end{aligned}$$

But note that $\sigma(-x)+\sigma(x) = 1 \Rightarrow$

```
function [ beta ] = boldDriver(Val_X,Val_y,Train_X,Train_y,n)
%Initialisation parameters
[~,w]=size(Train_X); beta = zeros(w,1); z=zeros(n,1); f=zeros(n,1); eta=1;
z(1)=likelihood(Train_X,Train_y,beta); f(1)=likelihood(Val_X,Val_y,beta);
% main loop
for i=2:n
    grad=Train_X'*(Train_y- logsig(Train_X*beta));
    beta_prev=beta;
    beta=beta+grad.*eta;
    z(i)=likelihood(Train_X,Train_y,beta);
    % likelihood has decreased so undo changes and decrease learning rate
    if z(i)<z(i-1)
      eta=eta*0.5;
      beta=beta_prev; z(i)=z(i-1);
      %learnign rate is fine so increase it
    else
          eta=eta*1.05;
    end
beta;
end
```

Question F)

		predicted label $\hat{y}$	
		0	1
true label y	0	fraction of true negatives = 0.66667	fraction of false positives= 0.33333
	1	fraction of false negatives = 0.28421	fraction of true positives = 0.71579

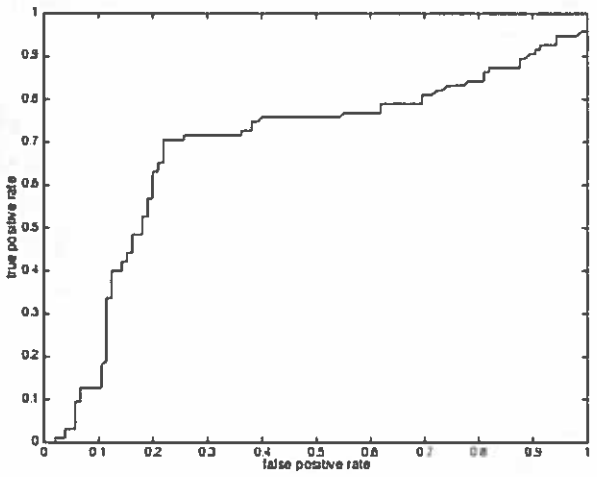
Training likelihood:0.5387 , Test likelihood: 0.5243

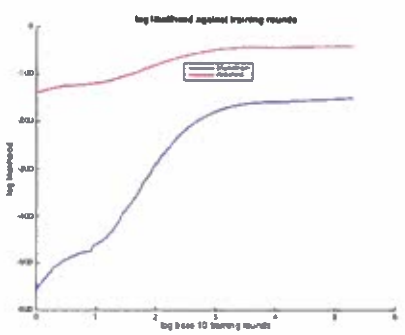
Please see **QE's graph** for the plot of the log likelihoods.

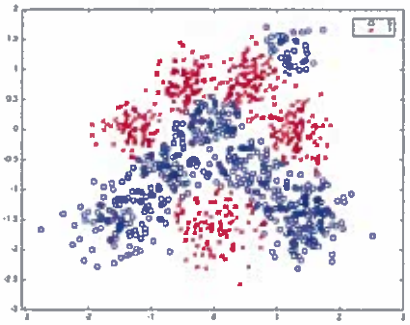
Question G)

The area under the curve, AUC is: 0.6842

In the diagram below we show a ROC curve for the training set. I think the classifier seems to give quite a poor performance. It is not classifying points at random because it is far away from a straight line, but at the same time it seems to be quite far from an ideal classifier which would have an area under the curve of 1. Judging by areas the classifier is closer in performance to a random classifier than to the 'ideal' classifier. The proportion of false negatives and false positives is fairly high at around 0.3. Indeed: 1-AUC=0.3, so we would expect our classifier to misclassify about a third of the points.







Question I)

-

len=0.01		predicted label $\hat{y}$	
		0	1
true label y	0	fraction of true negatives = 0.99048	fraction of false positives= 0.0095238
	1	fraction of false negatives = 0.77895	fraction of true positives = 0.22105

AUC=0.7583; Training Likelihood: 0.9998; Test Likelihood: 0.5307

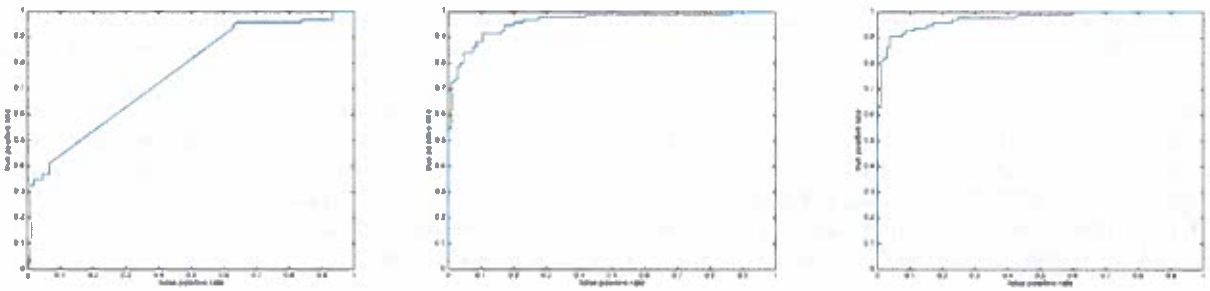
len=0.1		predicted label $\hat{y}$	
		0	1
true label y	0	fraction of true negatives = 0.91429	fraction of false positives= 0.085714
	1	fraction of false negatives = 0.11579	fraction of true positives = 0.88421

AUC=0.9612 ;Training likelihood:0.8727; Test Likelihood:0.7725

len=1		predicted label $\hat{y}$	
		0	1
true label y	0	fraction of true negatives = 0.95238	fraction of false positives= 0.047619
	1	fraction of false negatives = 0.094737	fraction of true positives = 0.90526

AUC= 0.9731; Training Likelihood: 0.8282 ;Test likelihood 0.8153;

From left to right: graphs for the roc curves l=0.01, l=0.1,l=1



As we can see, the models with length scale l=1 and l=0.1, are undeniably better classifiers than the simple classifier trained using the original inputs. They have greater training and test likelihoods and the areas under the ROC curves are higher. However, it could be argued that the model with length scale 0.01 offers little improvement to the original model. Although, the AUC is higher, the roc curve shows a strong straight-line component which is characteristic of random classifiers. And although the training likelihood is high, the model has very poor performance on the test data with a likelihood of 0.53. This is little better than random guessing. The fraction of false negatives has also gone up. So in summary, for l=0.01 the model might only be marginally better than the original model.

Explanation: The length scales of 1 and 0.1 are better because they offer a smoother probability distribution for the data. With a length-scale of 0.01, the distribution for positive or negative points becomes very spiky- with high probabilities focused on the the exact locations of the training data that taper off quickly as we move away. We can see this by looking at the inputs: with a length scale of $l = 0.01$, the term $\tilde{\mathbf{x}}_{m+1}^{(n)}$ becomes very large if $n = m$ and almost 0 if $n \neq m$. So the matrix of data-points, will essentially be a diagonal matrix. As such, the classifier just has to put a very high positive (if $\tilde{\mathbf{x}}_{m+1}^{(n)}$ is positive) or negative (if $\tilde{\mathbf{x}}_{m+1}^{(n)}$ is negative) weight on $\tilde{\mathbf{x}}_{m+1}^{(n)}$ to maximise the likelihood of the data. But then β becomes very large in each component,

len=0.01		predicted label $\hat{y}$	
		0	1
true label y	0	fraction of true negatives = 0.99048	fraction of false positives= 0.0095238
	1	fraction of false negatives = 0.8	fraction of true positives = 0.2

len=0.1		predicted label $\hat{y}$	
		0	1
true label y	0	fraction of true negatives = 0.94286	fraction of false positives= 0.057143
	1	fraction of false negatives = 0.13684	fraction of true positives = 0.86316

len=1		predicted label $\hat{y}$	
		0	1
true label y	0	fraction of true negatives = 0.92381	fraction of false positives= 0.07619
	1	fraction of false negatives = 0.094737	fraction of true positives = 0.90526