MLPR Assignment

Paridhi Mishra (s1549106)

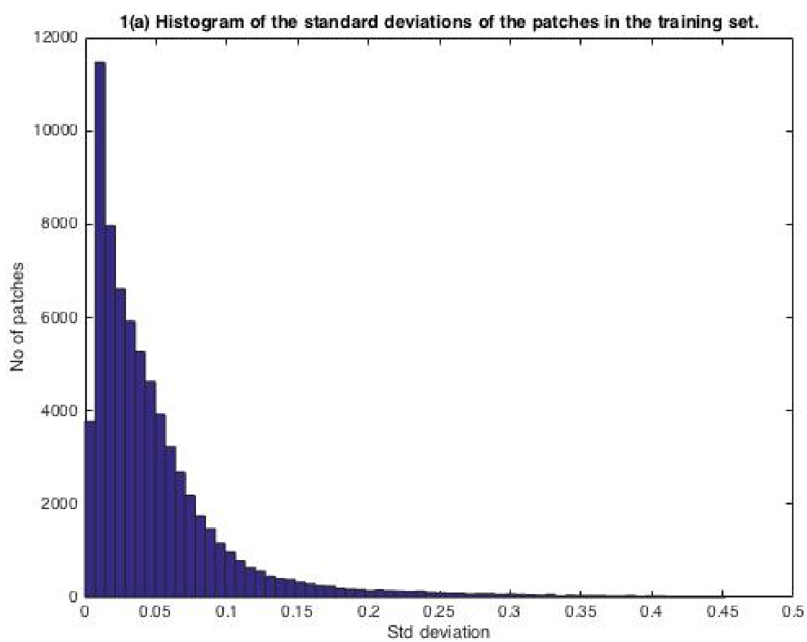
24th Nov 2015

1. The Next Pixel Prediction Task

Q1 Data preprocessing and visualization

(a)

Figure 1: Histogram



Observations from histogram:

1. Many patches have low deviations eg. roughly 3900 patches have 0 std deviations whereas roughly 11,000 out of 70,000 patches have 0.01 std deviation, indicating that a lot of these pixels values are similar, (smooth patches).

2. Since we assume all patches with std dev $< 0.063(4/63)$ are flat patches, from the plot is can be inferred that more than half of the patches are flat.

3. Some patches have very high deviations as well. of the standard deviations of the patches in the training set indicating that they can help in modelling (interesting patches).

Here 64 bins is used because the image has been discretized by 64 grey scales and this way each bin will plot each grey scale value for the image.

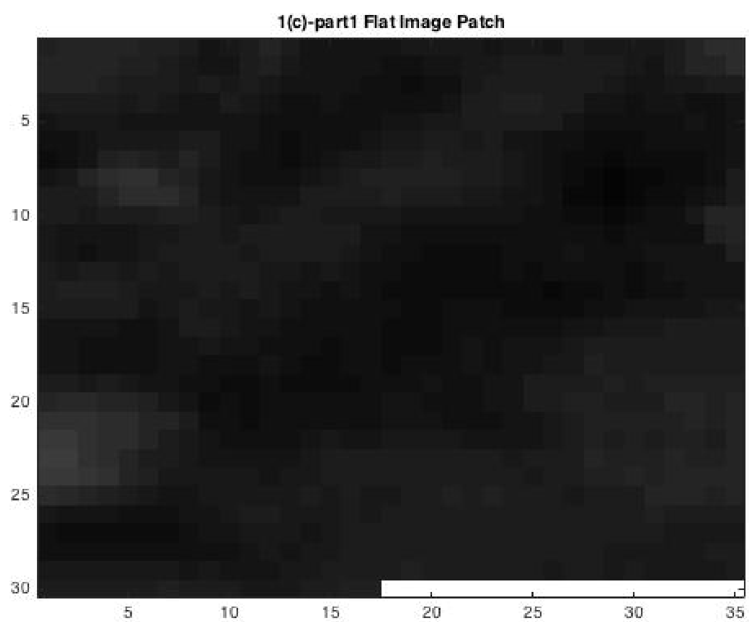
(b)

A simple way is to predict a flat patch is by comparing the y(j) pixel with the mean of all pixels in this same patch. It can also be done by comparing the mean of all pixels of this patch with the mean of x(j,end), x(j,end-34) and y(j). If both values are very similar, the chances of the patch being flat is high.

(c)

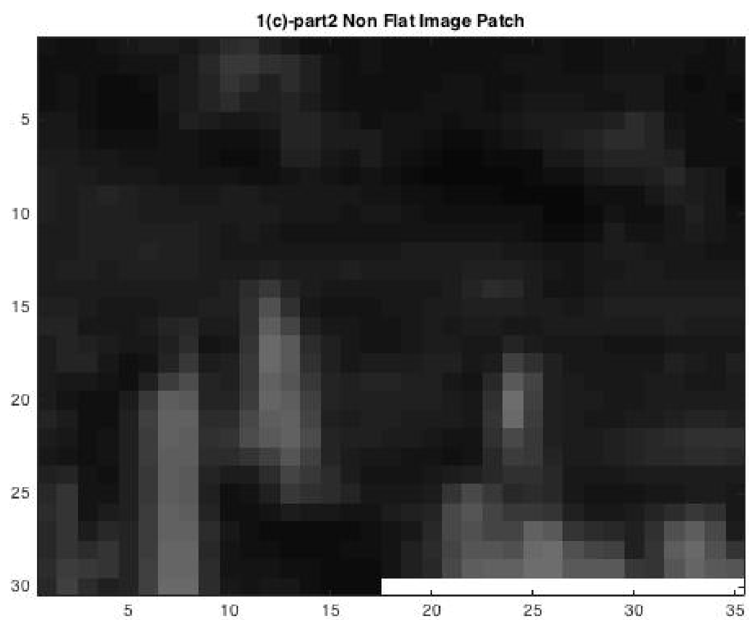
(c) Figures for flat and non-flat image patch is given below. Matlab code at the end of this section.

Figure 2



Matlab Code - Q1.1

Figure 3




```
load('imgregdata.mat');
% First scale all of the data to [0, 1] by dividing each pixel value by 63.
xtr_sc=xtr/63;
ytr_sc=ytr/63;
xte_sc=xte/63;
yte_sc=yte/63;
%compute the standard deviation of each x(j, :) patch
xtr_sd=std(xtr_sc');
% 1(a)- Plot a histogram
figure
hist(xtr_sd,64)
xlabel('Std deviation');
ylabel('No of patches');
title({'1(a) Histogram of the standard deviations of the patches in the training set.'});
rows = size(xtr_sc,1);
% as adviced in the assignment sheet.
threshold = 4/63;
for i = 1 : rows
    if (xtr_sd(i) <= threshold)
```

```
        patch_f=xtr_sc(i,:);
        break;
    end
end
%embed patch with 1's s, increase no of cols from 1032 to 1050 first
patch_embed=horzcat(patch_f,ones(1,18));
% now reshape into (35X30 piel size)
patch_reshaped=reshape(patch_embed,35,30);
figure
colormap gray;
imagesc(patch_reshaped',[0,1]);
title('1(c)-part1 Flat Image Patch');
for i = 1 : rows
    if (xtr_sd(i) > threshold)
        patch_nf=xtr_sc(i,:);
        break;
    end
end
%embed patch with 1's s, increase no of cols from 1032 to 1050 first
patch_embed=horzcat(patch_nf,ones(1,18));
% now reshape into (35X30 piel size)
patch_reshaped=reshape(patch_embed,35,30);
figure
colormap gray;
imagesc(patch_reshaped',[0,1]);
title('1(c)-part2 Non Flat Image Patch')
```

Q1.2 Linear regression with adjacent pixels

(a)

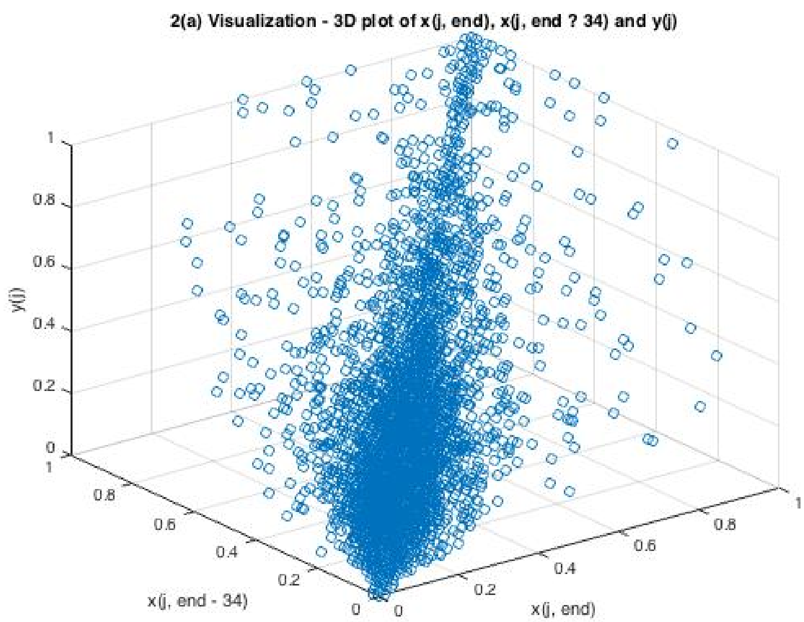
The 3D plot of x(j, end), x(j, end ? 34) and y(j) is presented in Figure 4. For this 3D plot, the data has been subsampled by taking every other row from both datasets Both subsampled sets had roughly 8600 instances after this operation.Strong positive linear correlations between all the 3 pixels are visible in this scatter plot. Both pixels x(j;end) and x(j:end-34) can help in predicting pixel y(j) because of this behaviour. This also indicates that linear regression might be a good technique to explore the interesting statistical trends in this image.

(b)

Bias weight has been added as 3nd column to matrix of x(j, end) and x(j, end - 34), and denoted as X. Let Y represent the vector for y(j) values. MLE solution $\mathbf{w} = \mathbf{(inv(X^\top\ X))X^\top y}$

The inv() function above is exponent to the power -1 (the inverse function).

Figure 4: .



(c)

The linear regression predictor is implemented in matlab as shown in code below. The weight vector values are: w = [0.46064 ; 0.52412 ; 0.00256].The root mean squared error on the training and test sets are RMSE training = 0.0506 and RMSE test = 0.0503 . Both values are very similar which indicates that the model is a good model and the training data was not over fitted. Had that been the case, RMSE of the test would have been much higher that RMSE of the training set.

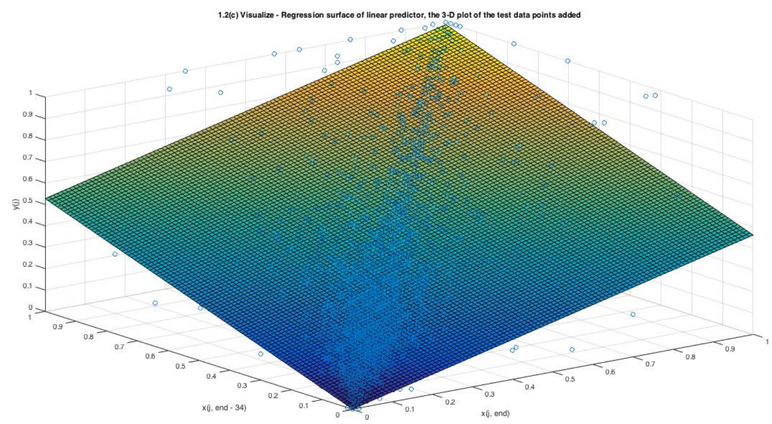
The regression surface of this linear predictor in 3-D using Matlab function surf() is in Figure 5 below. From this plot, the data is linearly separable as almost all test points like above this regression surface. which means that the linear classifier is successful in predicting the pixel values(y(j)) given the training set of pixels.

Matlab Code - Q1.2- Linear Regression - 2 variables

```
load imgregdata.mat xte_nf yte_nf xtr_nf ytr_nf
% 1.2(a) - Subsampled by taking every other row from both datasets
% Both subsampled sets have roughly 8600 instances
xtr_ss=xtr_nf(1:2:end,:);
ytr_ss=ytr_nf(1:2:end,:);

```

Figure 5




```
figure
plot(xtr_ss(:,end-34),xtr_ss(:,end),'r:+');

figure
scatter3(xtr_ss(:,end),xtr_ss(:,end-34),ytr_ss);
title({'2(a) Visualization - 3D plot of x(j, end), x(j, end ? 34) and y(j)'});
xlabel('x(j, end)');
ylabel('x(j, end - 34)');
zlabel('y(j)');

ytr=ytr_nf;
yte=yte_nf;

% 1.2(b) Denote the feature matrix as X, where each row is 3-dimensional,
% the first 2 dimensions are x(j, end) and x(j, end ? 34), and the third is simply 1.
xtr=[(xtr_nf(:,end)) , (xtr_nf(:,end-34)) , ones((size(xtr_nf,1)),1)];
xte=[(xte_nf(:,end)) , (xte_nf(:,end-34)) , ones((size(xte_nf,1)),1)];
X=xtr;
y=ytr;
% max likelihood weight(w) = inv(X'*X)*X'*y
w=((X'*X)^(-1))*(X'*y);
% w = [0.46064 ; 0.52412 ; 0.00256]
ytr_pr=xtr*w;
yte_pr=xte*w;
% Squared errors
```

```
ytr_se=(ytr_nf-ytr_pr).^2;
yte_se=(yte_nf-yte_pr).^2;
% Root Mean of squared errors
rmse_tr=sqrt(mean(ytr_se));
rmse_te=sqrt(mean(yte_se));
% RMSE training = 0.0506 , RMSE test = 0.0503
% 1.2(c) Visualize the regression surface of this linear predictor in 3-D using Matlab function surf().
figure
[dim1,dim2] = meshgrid(0:0.01:1,0:0.01:1);
ysurf = [[dim1(:), dim2(:)], ones(numel(dim1),1)]*w;
surf(dim1, dim2, reshape(ysurf, size(dim1)));
hold on;
scatter3(xte_nf(:,end),xte_nf(:,end-34), yte_nf);
xlabel('x(j, end)');
ylabel('x(j, end - 34)');
zlabel('y(j)');
title('1.2(c) Visualize - Regression surface of linear predictor, the 3-D plot of the test data points added');
```

Q1.3 RBF regression with adjacent pixels

(a)

The RBF model was tried with following number of basis functions 5, 10, 15, 20, 25, 30. using crossval function. The plot for the cross-validation RMSE against the number of RBFs used, is shown in Figure 6 below. Hence best model selected is for nbf = 10. The matlab code which implements this RBF model is shown below.

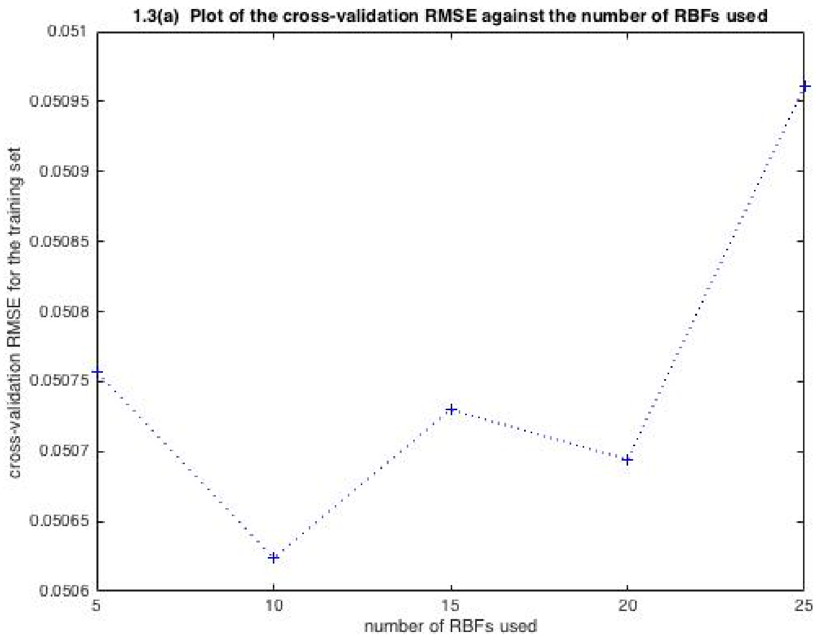
(b)

Choosing nbf = 10 (for which rmse is 0.05063), the RBF network is trained using all the non-flat training data . The values are -RMSE training = 0.0498 and RMSE test = 0.0526. Again, both the values are very close indicated a good fit for this model over given data. There is a slight improvement in RMSE compared to the previous linear regression model.

Matlab Code - Q1.3 - RBF

```
load imgregdata.mat xte_nf yte_nf xtr_nf ytr_nf
xtr_nf=[xtr_nf(:,end), xtr_nf(:,end-34)];
xte_nf=[xte_nf(:,end), xte_nf(:,end-34)];
options = foptions; % nbf is the number of basis functions
options(1) = 1; % Display EM training
options(14) = 5; % number of iterations of EM
count=1;
rmse = [0,0,0,0,0];
n_rbf = [5,10,15,20,25];
```

Figure 6




```matlab
while (count <= 5)% nbf using the candidates {5, 10, 15, 20, 25, 30}.
rbf_fn=@(xtr,ytr,xte)(rbffwd(rbftrain((rbf(2, n_rbf(count), 1, 'gaussian')),options,xtr,ytr),xte));
rmse(count) = sqrt(crossval('mse',xtr_nf,ytr_nf,'Predfun',rbf_fn));
count=count+1;
end
figure
plot(n_rbf,rmse,'b+:');
xlabel('number of RBFs used');
ylabel('cross-validation RMSE for the training set');
title('1.3(a)  Plot of the cross-validation RMSE against the number of RBFs used');
net_tr = rbf(2, 10, 1, 'gaussian');% 1.3(b)  best model nbf = 10
net_tr = rbftrain(net_tr,options,xtr_nf,ytr_nf);
ytr_pr = rbffwd(net_tr,xtr_nf);
ytr_se=(ytr_nf-ytr_pr).^2;
rmse_tr=sqrt(mean(ytr_se)); % rmse tr = 0.0498
net_te = rbf(2, 10, 1, 'gaussian');
net_te = rbftrain(net_te,options,xte_nf,yte_nf);
yte_pr = rbffwd(net_te,xte_nf);
yte_se=(yte_nf-yte_pr).^2;
```

```
rmse_te=sqrt(mean(yte_se)); % rmse te = 0.0504
```

Q1.4 Linear regression with all pixels

Extending the model and implementing the linear regression with all pixels, RMSE training = 0.0456 and RMSE test = 0.0371 .Compared to its simpler version of linear model, it performs better on both the training and test sets, but marginally only. Given that so many extra parameters were provided for this model, the performance has not improved significantly. It indicates that the 2 previous pixels alone can predict the target pixel in a satisfactory manner. Compared to the RBF model too, if performs a little better as seen from the RMSE values, but since there is no concrete proof that it is a better model that RBF.

Matlab Code - Q1.4 - Linear Regression - all pixels

```
load imgregdata.mat xte_nf yte_nf xtr_nf ytr_nf
% max likelihood weight(w) = inv(X'*X)*X'*y
weight = ((xtr_nf'*xtr_nf)^(-1))*(xtr_nf'*ytr_nf);
ytr_pr = xtr_nf*weight;
yte_pr = xte_nf*weight;
% Squared errors
ytr_se=(ytr_nf-ytr_pr).^2;
yte_se=(yte_nf-yte_pr).^2;
% Root mean squared errors
rmse_tr=sqrt(mean(ytr_se)); % RMSE training = 0.0456
rmse_te=sqrt(mean(yte_se)); % RMSE test = 0.0371
```

Q1.5 Neural Network with all pixels

(a)

Using the well trained MLP provided, a neural network was built with 10 hidden units. The RMSE training = 0.0333 and RMSE test = 0.0473.

Comparing its RMSEs with those from linear regression, there is a small difference only. What is noticeable is that the linear regression model seems to be less overfitting that the NN since the difference if test and training values is smaller. At the same time, the RMSE of training of NN is better than linear model which indicates makes it harder to compare both these models.

(b)

Using first 5000 data points to train the MLP and running the training 5 times using different random seeds, the resulting RMSE's are given below. The Matlab code is provided at the end of the section.

All the training and test RMSE values for the small dataset (first 5000 points) are higher that what was observed in 1.5.a. Even though different initial values (random seeds) were used for 5 different times, it did not show improvement This suggests that this subset of data taken might not be very representative of the original dataset. It is also possible that since NNs adapts its network as it learns from data, the lesser data it saw in this part is the reason for higher error values. If so, then it is expected to perform better with larger datasets and over time.

Training RMSE	Test RMSE
0.0475	0.0499
0.0488	0.0515
0.0485	0.0515
0.0486	0.0527
0.0489	0.0526

Matlab Code - Q1.5

```
load welltrainedMLP.mat
load imgregdata.mat xte_nf yte_nf xtr_nf ytr_nf
% net struture is provided with data
ytr_pr = mlpfwd(net, xtr_nf);
yte_pr = mlpfwd(net, xte_nf);
rmse_tr = sqrt(mean((ytr_nf - ytr_pr).^2));  % RMSE on training set
rmse_te = sqrt(mean((yte_nf - yte_pr).^2));  % RMSE on test set
%part(b) - first 5000 data points
arr_rmse_tr=[0 0 0 0 0];
arr_rmse_te=[0 0 0 0 0];
for i= [2015,2016,2017,2018,2019]
  rng(i,'twister')
  nhid = 10; % number of hidden units
  net = mlp(size(xtr_nf,2), nhid, 1, 'linear');% Set up the network
  options = zeros(1,18); % Set up vector of options for the optimiser.
  options(1) = 1; % This provides display of error values.
  options(9) = 1; % Check the gradient calculations.
  options(14) = 200; % Number of training cycles.
  % Train using scaled conjugate gradients.
  [net, options] = netopt(net, options, xtr_nf(1:5000,:), ytr_nf
      (1:5000,:), 'scg');
  ytr_pr = mlpfwd(net, xtr_nf);
  yte_pr = mlpfwd(net, xte_nf);
  tr_rmse = sqrt(mean(((ytr_nf - ytr_pr).^2)));  % RMSE on training set
  te_rmse = sqrt(mean(((yte_nf - yte_pr).^2)));  % RMSE on test set
  arr_rmse_tr(i-2014)=tr_rmse;
  arr_rmse_te(i-2014)=te_rmse;
end
```

Q1.6 Discussion

Above we have considered linear regression and RBF network using 2 neighbouring pixels, and linear regression and a neural network on all pixels.

Compare these methods Especially for all pixels case, NN has the advantage that it can model non linearities in multiple pixels automatically when compared to a linear regression model which implements a statistical model . For example incase the training data has noise, NN is better in mapping hidden and nonlinear input-output dependencies ..But with the freedom of adding multiple hidden layers and nodes, the NN is also more prone to overfitting of the data that linear model.

In this particular task of pixel prediction for image compression, NN model minimizes the training root mean squared error, thus the neural networks prediction, leads to better results.

Briefly, what experiment would you try next if you wanted to improve on these predictors?

As number of pixels grow, I would like to try which of above two models handle large numerical data better. Also, once implemented, which model is better at adjusting with the changes, say for example accomodating the historical data The next experiment I would try is changing the dataset and running the models on these new datasets since comparing the models on a single dataset can be dangerous.

Due to the complexities which increase as the network and hidden layers increase in an adaptive NN model, I would like to compare the models in terms of execution time.It might be possible that the NN model is unnecessarily slow for this task.

Q.2. Robust modelling

Q2.1 Fitting the baseline model

(a)

The bias feature was added in the logistic regression model by augmenting the data and adding an extra dimension. The bias weight bw (d+1) recorded is

(b)

(b) Maximizing the likelihood

Test set Accuracy = 0.3194 Mean Log Probability = -0.7140 Std error = 0.0116
Training set Accuracy = 0.2977 Mean Log Probability = -0.7311 Std error = 0.0057

Performance of predictions on training and test

We know that a baseline that predicts P(y|x)=0.5 makes no assumptions about the data, so it will have error of 0.5 for every prediction. Mean probability, however, will predict closer to the label that it predicts as more common. So long as the label it predicts as more common IS more common, a baseline that uses mean probability will be better.

(c)

(c) Limited training data

Avg log probability = -0.6196 Because a small training set was used, many of the weights in the weight vector quicky minimize to either 0 or 1, indicating that in this small training set, all feature vectors with a certain value for a certain feature had the same label. In our concrete text classification example, this would indicate that for instance that only sports-labelled documents contained the word "pass." When this weight vector is used to classify a document, the linear regressor returns values of zero or one. This is an artifact of weight vectors that over-generalize the likelihood of seeing a given feature in a document of a given label, but the effect is to make the regressor entirely certain of a document's label. If the regressor returns a probability of 0 for any of the documents in the test set, then the mean log probability becomes -Inf +/- NaN.

Matlab Code - Q2.1 - Fitting the baseline model

```
load text_data.mat;
x_train = [x_train ones(size(x_train,1),1)]; %training and set expanded
x_test  = [x_test ones(size(x_test,1),1)]; % and bias feature added
%weights = ones(101,1);  % initialize using all 1's weight vector
weights=rand(101,1); % initialize randomly weight vector

%create a negative-log-likelihood function
[NLp, dNLp_dw] = log_like_negative(weights, x_train, y_train);

% Minimize it given the training data x_train and y_train
minimized_weights = minimize(weights, @log_like_negative, 1, x_train, y_train);
bias_weight=weights(101); %bias weight

% using fitted weights, find the probability that y= +1 for each of the test
% inputs x_test.
prob_te = 1./(1 + exp(-y_test.*(x_test*minimized_weights)));
% the probability is compared to y_test which has values(-1 or +1), and
% for prediction of label, prob >= 0.5, round is used to roundoff the prob.
prob_te_rd=round(prob_te);
acc_te=mean(prob_te_rd == y_test);
var_te=var(prob_te_rd == y_test);
std_err_te = sqrt(var_te/size(y_test,1));

% find mean log probability that predictions assign to test labels
mean_log_prob_te = mean(log(prob_te));

% performance of predictions on training set
prob_tr = 1./(1 + exp(-y_train.*(x_train*minimized_weights)));
prob_tr_rd=round(prob_tr);
acc_tr=mean(prob_tr_rd == y_train);
var_tr=var(prob_tr_rd == y_train);
std_err_tr = sqrt(var_tr/size(y_train,1));
mean_log_prob_tr = mean(log(prob_tr));

weights_ltd = rand(101,1);

% Fit the model with only the first N = 100 training cases
min_weights_ltd= minimize(weights_ltd, @log_like_negative, 100, x_train(1:100,:), y_train(1:100));
% probability of test set being classified as +1
prob_ltd = 1./(1 + exp(-y_train(1:100).*(x_train(1:100,:)*
```

```
    minimized_weights)));
avg_log_prob = (sum(log(prob_ltd)))/100;
```

Q2.2 Label noise model

(a)

(a) Modifying the likelihood:

Functions created which return log likelihood of this model, given data are: $\log_{l}ike_{n}oise_{w}()andlog_{l}ike_{n}oise_{e}()w.r.twar$

(b)

(b) Fitting a constrained parameter:

Create a function that evaluates the negative log-likelihood of the new model and evaluates the derivatives with respect to w and a. Hence fit both w and a. You are advised to wrap the function from the previous part, rather than starting from scratch. Report the fitted noise level e=?(a). Given that the test labels were checked more carefully, predict them using the newly fitted weights but using the original model (1). Report and interpret the new test accuracy and mean log probability.

Matlab Code - Q2.2

```
load text_data.mat;
x_train = [x_train ones(size(x_train,1),1)]; %training and set expanded
x_test  = [x_test ones(size(x_test,1),1)]; % and bias feature added
weights = ones(101,1);  % initialize using all 1's weight vector
%2.2(a) modifying the likelihood
[lp1, dw] = log_like_noise_w(weights,x_train,y_train,0.1);
grad_w = checkgrad(@lr_loglike_noise_w, weights, 1e-1, x_train, y_train
    , 0.1);
[lp2, de] = log_like_noise_e(0.1,x_train,y_train,weights);
grad_e = checkgrad(@lr_loglike_noise_e, 0.1, 1e-1, x_train, y_train,
    weights);
%2.2(b) fitting a constrained param : adding a
weights = rand(102,1);
min_weights = minimize(weights, @log_like_noise_a, 100, x_train,
    y_train);
e = 1./(1 + exp(-min_weights(end)));
% mlp for noisy weights on initial model
mlp_noise_te = mean(log(1./(1 + exp(-x_test * min_weights.*y_test))));
% probability of test set being classified as +1
ytest = x_test * min_weights;
probability_yte = 1./(1 + exp(-ytest));
```

Q2.3 Hierarchical model and MCMC

(a)

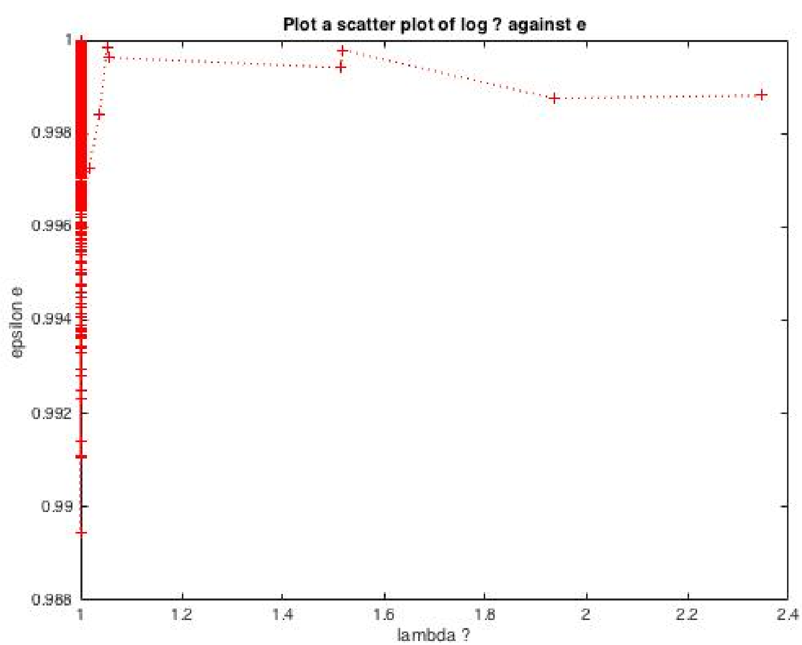
If a model were somehow able to predict every training label correctly, the binary noise variable from Eq 3 will become 0 and the log likelihood will behave as a model with no noise. For w = 0, the likehihood will become sum(log(1/(1+e^0))) $= sum(log(0.5)) = N * (-0.6)$
$N is the number of training/test samples.$

(b)

Putting w, e and log lambda into a single vector, and writing a wrapper to the log posterior function to evaluate Eq(6), slice sample fn was called. Step out values was toggled between false and true, and a number of combinations of width, burn and N were tried. For initialization of the row vector with e,lambda,w, rand fn was called to set the values.

The function returned N samples of e,lambda,w. Plot of lambda against e is shown below. Since the step-out was set to false, the function did not performed as expected.

Figure 7



(c)

The slice sampler was hung indefinitely with step out = true. All the samples returned with step-out = false are valid, even though the distribution is not as expected. These observations can be utilized as part of test sets of the same model. The values observed are epsilon values are high - between 0.9 and 1.0 and the lambda values are between 1 and 3. Since the function did not complete the execution as expected, the comparison is not possible.

Matlab Code - Q2.3 - Hierarchial Model and MCMC

```
load text_data.mat;
x_train = [x_train ones(size(x_train,1),1)]; %training and set expanded
x_test  = [x_test ones(size(x_test,1),1)]; % and bias feature added
%weights = ones(101,1);  % initialize using all 1's weight vector
weights=rand(101,1); % initialize randomly weight vector
dim=size(weights,1);
log_lambda=0.1; % initilize with random value btw 0 and 1
e=0.5; % initilize with random value btw 0 and 1
% handle of log_posterior passed through slice_sample
log_p=log_posterior(e,log_lambda,weights,dim,x_train,y_train);
% set the values for slice_sample
N=10; % no of iterations
burn=0; % default value
widths=1; % default value
step_out=false; % value set to true hangs the code.
rng(0,'twister');
init=init_params(dim); % row vector initialized
log_p_s = @(args) log_posterior(args{:}, dim, x_train, y_train);
% wrapper created so as to hide extra parameters
log_s_s = @(vector) log_p_s(split_params(vector));
% make sure that the initial row vector is inside the prob density area
assert(~isinf(log_s_s(init))); % before passing it to slice_sample
%MCMC slice sampling for heirarchial posterior
result = slice_sample(N, burn, log_s_s, init, widths, step_out);
% the result contains 3 params - e, lambda and w
params_cells=split_params(result);
e=params_cells(1);
log_lambda=params_cells(2);
weights=params_cells(3);
res_lambda=exp(log_lambda{1,1});
res_e=e{1,1};
res_weights=weights{1,1};
% scatter plot to confirm posterior belie about log lambda and e
plot(res_lambda,res_e,'r:+');
xlabel('lambda ?');
ylabel('epsilon e ');
title({'Plot a scatter plot of log ? against e'});
```

Appendix A - Additional Code

```
function log_post = log_posterior(e, log_lambda, ww, D, xx, yy)

if (e<=0 || e>1) % zero prior density
    log_post = -Inf;
    return;
end
if (log_lambda<=0 || log_lambda>1)
    log_post = -Inf;
    return;
end
sigmas = 1./(1 + exp(-yy.*(xx*ww)));
lambda=exp(log_lambda);
% Dropped one constant term = -log(pi)
log_prior = -lambda.*((ww')*ww) + (D/2)*log_lambda;
log_like = sum(log(((1-e)*sigmas)+(e/2)));
log_post = log_like + log_prior;

end
```

```
function [params] = split_params(vector)
% slice_sample returns a matrix, this fn
% extracts the 3 sampled parameters
[rows,cols]=size(vector);
%initial case while starting slice_sample
if rows == 1
    e=vector(1);
    log_lambda=vector(2);
    weights=vector(3:end);
    params={e,log_lambda,weights'};
    return
else
    e=vector(1,:);
    log_lambda=vector(2,:);
    weights=vector(3:end,:);
    params={e,log_lambda,weights}; % the returned result of slice_sample
end
```

```
function params = init_params(dim)
% initializes the row vector to be passed to slice_sample()
params=ones(1,dim+2);
params(1)=rand(); % 1st param is epsilon (e) - range btw 0 and 1
params(2)=rand(); % 2nd param is log(lambda) - range btw 0 and 1
b=9.376307381550576e+04; % range of weight vector - random value
a=-7.460274498610420e+05;
for i = 3 : dim+2 % 3rd element till last element is weight vector
```

```
    params(i)=(b-a).*rand() + a;
end
```

```
function [NLp, dNLp_dw] = log_like_negative(ww, xx, yy)
% this fn returns the negative log likelihood
yy = (yy==1)*2 - 1;
sigmas = 1./(1 + exp(-yy.*(xx*ww)));
NLp = -sum(log(sigmas));
if nargout > 1% additionally returns the derivate w
    dNLp_dw = (((-1)*(1-sigmas).*yy)' * xx)';
end
```

```
function [Lp, dLp_dw] = log_like_noise_w(ww, xx, yy, e)
% this fn returns the log likelihood for a model with noise e
% and the gradient w
yy = (yy==1)*2 - 1;
sigmas = 1./(1 + exp(-yy.*(xx*ww))); % Nx1
Lp = sum(log((1-e)*sigmas+(e/2)));
if nargout > 1
    dLp_dw = (((((1-e)./((1-e)*sigmas+e/2)).*sigmas.*(1-sigmas)).*yy)'
        * xx)';
end
```

```
function [Lp, dLp_de] = log_like_noise_e(e, xx, yy, ww)
yy = (yy==1)*2 - 1;
% this fn returns the log likelihood for a model with noise e
% and the gradient e
sigmas = 1./(1 + exp(-yy.*(xx*ww))); % Nx1
Lp = sum(log((1-e)*sigmas+(e/2)));
if nargout > 1
    dLp_de = sum((-1.*sigmas + (1/2)).* (1./((1-e)*sigmas+(e/2))));
end
```

```
function [NLp, dLp_dw,dLp_da] = log_like_noise_a(ww, xx, yy)
yy = (yy==1)*2 - 1;
% this fn returns the -ve log likelihood for a model with noise
% unconstrained parameeter a and returns derivative wrt w and a
a = ww(end,:);
ww = ww(1:end-1,:);
e = 1/(1 + exp(-a));
sigmas = 1./(1 + exp(-yy.*(xx*ww)));
[Lp, dLp_dw] = log_like_noise_w(ww,xx,yy,e);
NLp=-Lp;
dLp_dw=dLp_dw*a;
dLp_da= sum(((log((1-log(a))*sigmas+(log(a)/2))).^(-1))*(((1/a)*sigmas)
    +(0.5*a)));
```

Appendix B - References

1.http://uk.mathworks.com/help/
2.http://mlg.eng.cam.ac.uk/zoubin/tut06/mcmc.pdf
3..http://homepages.inf.ed.ac.uk/imurray2/teaching