Machine Learning and Pattern Recognition Coursework

Zhen Yi, s1563190

November 24, 2015

NOTE — BEFORE YOU START
The coursework is formated by LaTeX, if you want a doc format copy, please contact me without any hesitation.

Environment for this coursework

For this coursework, I used Matlab as the development tool, and I have some references such as the offical document of Matlab , the papers and the book which are helpful in theory part.

Solution — 2 parts

Part 1

1.a) For this question, I used command `histogram(std_tr, 64)`. Please refer to Figuare 1. The reason why 64 bins is that although we scaled all the data to 64 bins, we don't change the consistency of data. Thus, we still can split the data to 64 bins.
This plot tells us the approximate distribution of the standard deviaion of the

patches. It looks like that the most of standard devation coverge to zero, thus we can assume that most of the patches are flat patches.

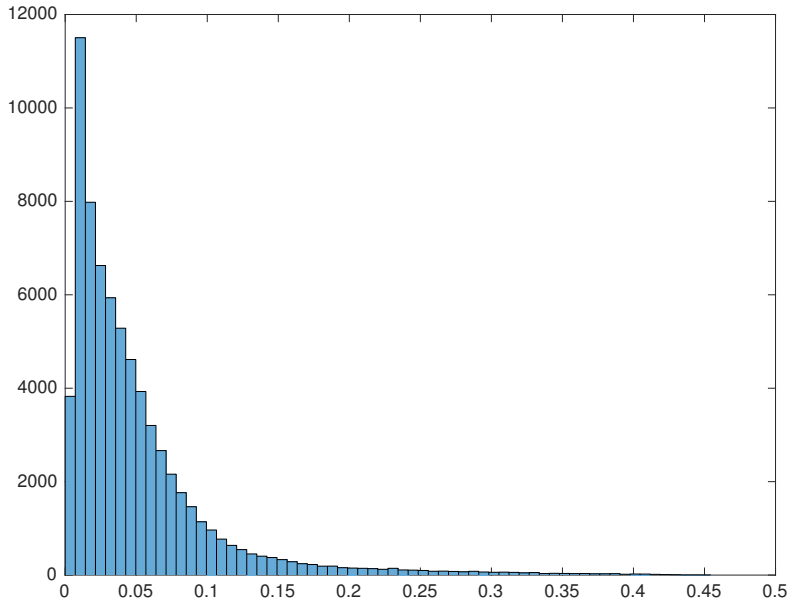

Figure 1: Histogram(using 64 bins)

1.b) This question ask us to predict the target pixel, because we have already had the result of standard deviation. We can simply use such result. The meaning of standard deviaion is that if the closer it is to zero, the more flat it is. Thus, we can use the function **mean**, which means that most the patch can be seen as flat. Thus, the value of the target pixel closes to the mean of the whole patch.

```
1  targetPixel = mean(xtr_1a, 2); %(1.b)
```

1.c) Figure 2 shows one flat and one non-flat image patch, respectively.

```
1  flatPatch = find(std_tr ≤ err);
2  nonflatPatch = find(std_tr > err);
3  nononeFlatPatch = [xtr_1a(nonflatPatch(10000), :) zeros(1, 18)]; ...
       % find a non-flat image patch locates in 2000
```

```
nononeFlatPatch = reshape(nononeFlatPatch, [35, 30])';

oneFlatPatch = [xtr_1a(flatPatch(50), :) zeros(1, 18)]; % find a ...
    non-flat image patch locates in 100
oneFlatPatch = reshape(oneFlatPatch, [35, 30])';

colormap gray;

subplot(1,2,1), imagesc(oneFlatPatch, [0,1]);
subplot(1,2,2), imagesc(nononeFlatPatch, [0,1]);
```

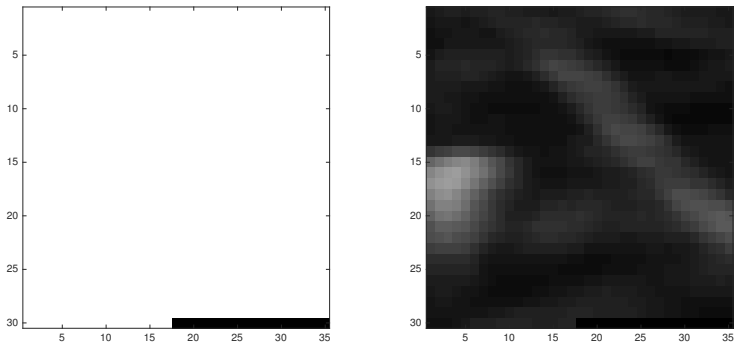

Figure 2: Flat and Non-flat pictures

Note: Here is the full `Matlab` code for the question 1.

```
%load('imgregdata.mat');

xtr_1a = xtr / 63.0; % scaling the training data
ytr_1a = ytr / 63.0;

xte_1a = xte / 63.0; % scaling the test data
yte_1a = yte / 63.0;

std_tr = std(xtr_1a,0,2); % standard derivation

histogram(std_tr, 64); %(1.a)

err = 0 / 64.0;
targetPixel = mean(xtr_1a, 2); %(1.b)

flatPatch = find(std_tr ≤ err);
nonflatPatch = find(std_tr > err);
nononeFlatPatch = [xtr_1a(nonflatPatch(10000), :) zeros(1, 18)]; ...
    % find a non-flat image patch locates in 2000
nononeFlatPatch = reshape(nononeFlatPatch, [35, 30])';

```

```
oneFlatPatch = [xtr_1a(flatPatch(50), :) zeros(1, 18)]; % find a ...
    non-flat image patch locates in 100
oneFlatPatch = reshape(oneFlatPatch, [35, 30])';

colormap gray;

subplot(1,2,1), imagesc(oneFlatPatch, [0,1]);
subplot(1,2,2), imagesc(nononeFlatPatch, [0,1]);
```

2.a) Figure 3 shows that the points cluster near a point (0.25, 0.25, 0.25) which closes to zero. It looks like a comet. Most of the points cluster around the such centre point (0.25, 0.25, 0.25), the lower bound and upper bound of this body is (0, 0.5]. It also has a tail from (0.5, 1.0).
Substantially, it can be seen as positive linear correlation(from (0,0,0) to (1,1,1)). And the part of the points which below (0.5, 0.5, 0.5) is more related to the linear correlation than the points which above (0.5, 0.5, 0.5).

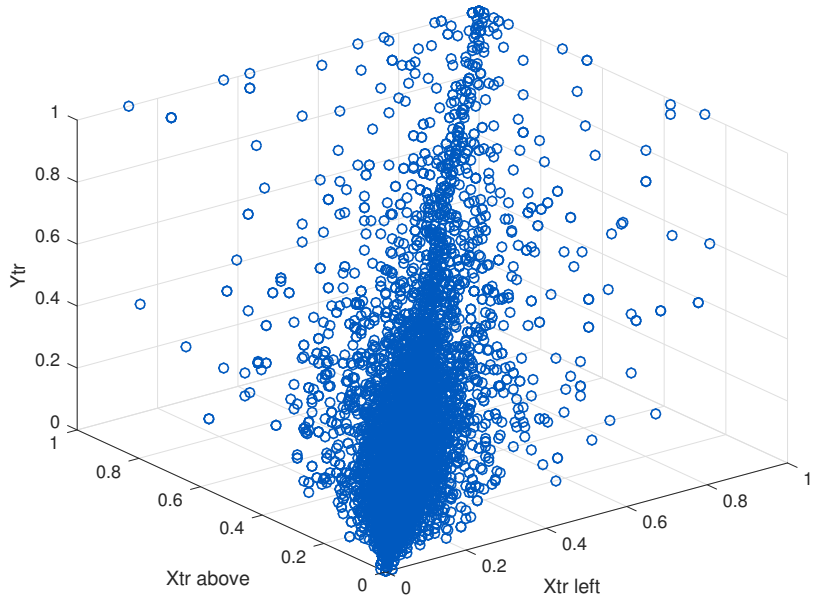

Figure 3: Subsample

2.b) The kernel code is `w = inv(X' * X)* X' * ytr_nf`. We can get the solution of **w** using X and the vector **y**:

$$\log P(y|x,w) = -\frac{1}{2}\log(2\pi\sigma^2) - \frac{1}{2\sigma^2}(y - w^T X)^2$$
$$L(w) = \sum_{i=1}^{n}(\log P(y_i|x_i,w))$$
$$\frac{\partial}{\partial w}L(w) = \frac{\partial}{\partial w}(-\frac{1}{2\sigma^2}\sum_{i=1}^{n}(y_i - w^T X)^2)$$
$$\hat{w} = (X^T X)^{-1} X^T y$$

```
X = [xtr_nf(:,end) xtr_nf(:,end - 34) (ones(1, length(xtr_nf)))'];

w = inv(X' * X) * X' * ytr_nf;
% w = regress(ytr_nf,X); I used this function to check my answer
```

2.c) The linear regression predictor is $y = X * w$. RMSE of traning data is 0.0506, RMSE of test data is 0.0503.
My linear regressor use the closed-form maximum likelihood for weight vector $\vec{w}$, it can work in the simple data sample, however it may not as powerful as we predicting. Because, it dosen't include any iteration to reduce the error.
Now, we need to analyze why RMSE of test data is less than the traning data's. I think, there are more outliers in the traning data. Thus, such circumstance happens.

$$w = \begin{bmatrix} 0.4606 \\ 0.5241 \\ 0.0026 \end{bmatrix}$$

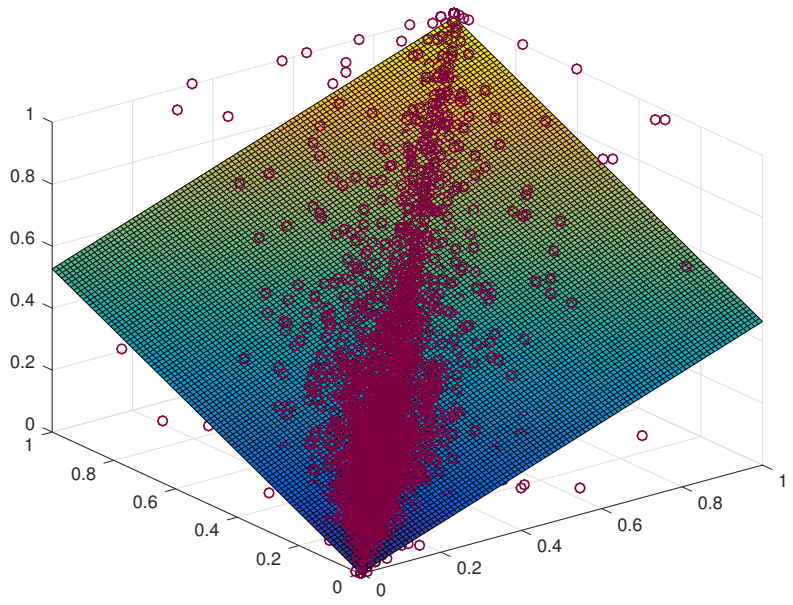

Figure 4: Suface and 3-D plot of test data

Note: Here is the full `Matlab` code for the question 2.

```
load('imgregdata.mat');

position = datasample(1:length(xtr_nf), 10000); %subsample the ...
    postion

xtr_left = [];
xtr_above = [];
ytr_new = [];

for i = 1 : 10000
    xtr_left = [xtr_left; xtr_nf(position(i), end)];
    xtr_above = [xtr_above; xtr_nf(position(i), end - 34)];
    ytr_new = [ytr_new; ytr_nf(position(i))];
end;

scatter3(xtr_left, xtr_above, ytr_new, 'MarkerEdgeColor',[0 .35 ...
    .75]);

xlabel('Xtr left');
ylabel('Xtr above');
zlabel('Ytr');
```

```
20
21 % 2.b
22 X = [xtr_nf(:,end) xtr_nf(:,end - 34) (ones(1, length(xtr_nf)))'];
23
24 w = inv(X' * X) * X' * ytr_nf;
25 % w = regress(ytr_nf,X); I used this function to check my answer
26
27 % 2.c
28 ytr_regress = X * w;
29 RMSE_ytr = sqrt(mean((ytr_regress - ytr_nf).^2));
30 yte_regress = [xte_nf(:,end) xte_nf(:,end - 34) ...
       ones(length(xte_nf), 1)] * w;
31 RMSE_yte = sqrt(mean((yte_regress - yte_nf).^2));
32
33 figure,
34 [dim1, dim2] = meshgrid(0:0.01:1, 0:0.01:1);
35 ysurf =[[dim1(:), dim2(:)], ones(numel(dim1), 1)] * w;
36 surf(dim1, dim2, reshape(ysurf, size(dim1)))
37
38 hold on
39
40 scatter3(xte_nf(:,end), xte_nf(:,end - 34), yte_nf(:), ...
       'MarkerEdgeColor',[0.5 .0 .25]);
```

3.a) In this question, I used 10-fold cross validation and only map the argument of traning data to function. Figure 5 shows the RMSE of 6 different nbf functions.
I have run the **matlab** program 10 times, and plot them in one diagram. You can see that it is fluctuant. Therefore, we should to choose the most stable one from the all 6 possible choice. By observation, I will choose 10 for next problem.

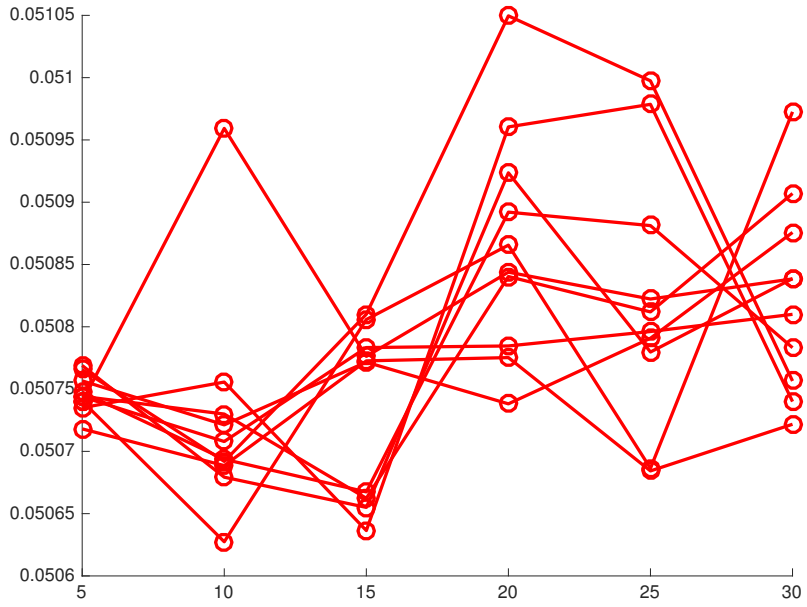

Figure 5: cross-validation RMSE

Note: Here is the full `Matlab` code for the question 3.a.

```
load('imgregdata.mat');
%code for creating RBF and making predictions
%nbf is the number of basis functions
%dim is the dimensionality of input space

nbf = [5,10,15,20,25,30];
format long; % set display precision
options = foptions;
options(1) = 1;
options(14) = 5;
TrainX = xtr_nf(:,[end end - 34]);
%TestX = xte_nf(:,[end end - 34]);

for j= 1:10
for i = 1:6
    dim = 2;
    net = rbf(dim, nbf(i), 1, 'gaussian');
    %10-fold cross validation
    rbff=@(XTRAIN,ytrain,XTEST)(rbffwd(rbftrain(net, options, ...
        XTRAIN, ytrain), XTEST));
    RMSE(i) = sqrt(crossval('mse',TrainX, ytr_nf, 'Predfun',rbff));
```

```
21  end
22
23  hold on;
24  plot(nbf, RMSE,'r','LineWidth',2,'Marker','o','MarkerSize',10);
25  end
```

3.b) For this question, the result is better the linear regression, but they are very close. And the method of RBF is totally different from linear regression, because of non-linear feature. And the result of test is better than the result of training.

$$RMSE \quad traning = 0.0505$$
$$RMSE \quad test = 0.0501$$

```
1   load('imgregdata.mat');
2   %code for creating RBF and making predictions
3   %nbf is the number of basis functions
4   %dim is the dimensionality of input space
5   net = rbf(2, 10, 1, 'gaussian');
6
7   options = foptions;
8   options(1) = 1;
9   options(14) = 5;
10
11  rbff= rbftrain(net, options, xtr_nf(:,[end end - 34]), ytr_nf);
12  ypred_tr = rbffwd(rbff, xtr_nf(:,[end end - 34]));
13  ypred_te = rbffwd(rbff, xte_nf(:,[end end - 34]));
14
15  RMSE_ytr = sqrt(mean((ypred_tr - ytr_nf).^2));
16  RMSE_yte = sqrt(mean((ypred_te - yte_nf).^2));
```

4) In this question, we implement a linear regressor with full pixels. The result are : RMSE of traning data is 0.0371, RMSE of test data is 0.0456.
The RMSEs are less than previous result which produced by that linear regressor. The more dimension of the train data is, the less RESE we can get. Therefore, we got the better solution.
Of course, we can get more features from the larger amount of information. The reason why we use non-flat traning data is that we should reduce the outliers. For this example, the over-fitting occurs, thus the gap between RMSE of traning data and RMSE of test data is big.

Note: Here is the full `Matlab` code for the question 4.

```
1   X = [xtr_nf (ones(1, length(xtr_nf)))'];
```

```
2
3  w = inv(X' * X) * X' * ytr_nf;
4  %w = regress(ytr_nf,X); %I used this function to check my answer
5
6  ytr_regress = X * w;
7  RMSE_ytr = sqrt(mean((ytr_regress - ytr_nf).^2));
8  yte_regress = [xte_nf ones(length(xte_nf), 1)] * w;
9  RMSE_yte = sqrt(mean((yte_regress - yte_nf).^2));
```

5.a) In this question, I used the well trained MLP to compute both of traning data and test data. The RMSE of traning data is 0.0333, the RMSE of test data is 0.0473. The RMSE of traning data is less than such data of linear regression. However, the RMSE of test data is greater than such data of linear regression. Only for this example, we can assert that sometimes the performance of NN is not as good as linear regression. I guess the reason is that, they used different traning data or the NN is more appropariate traning data, but not test data. This may cause the over-fitting. Therefore, the result of test data is worse than taht of linear regression.

```
1  %load('imgregdata.mat');
2
3  ypred_tr = mlpfwd(net, xtr_nf);
4  rmse_NNsuball_tr = sqrt(mean(((ytr_nf - ypred_tr).^2)));
5
6  ypred = mlpfwd(net, xte_nf);
7  rmse_NNsuball_te = sqrt(mean(((yte_nf - ypred).^2)));
```

5.b) Referring to the table. I report the result of different seed. The gap between the RMSE of traning data and RMSE of test data gets larger. Because we have tried 5 seeds. We have the different initialization of the corresponding seeds. And the results of the performace are similiar. So we get the conclusion is that limited data cause the over-fitting problem.

Seed	2015	2016	2017	2018	2019
RMSE train	0.0387	0.0363	0.0346	0.0334	0.0347
RMSE test	0.0515	0.0504	0.0515	0.0516	0.0527

```
1  rng(2019, 'twister');
2  nhid = 10;
3  net = mlp(size(xtr_nf, 2), nhid, 1, 'linear');
4
5  options = zeros(1, 18);
6  options(1) = 1;
7  options(9) = 1;
8  options(14) = 200;
```

```
9
10 tic
11 [net, options] = netopt(net, options, xtr_nf(1: 5000, :), ...
       ytr_nf(1:5000,:),'scg');
12 toc
13
14 ypred_tr = mlpfwd(net, xtr_nf(1:5000,:));
15 rmse_NNsuball_tr = sqrt(mean(((ytr_nf(1:5000) - ypred_tr).^2)));
16
17 ypred = mlpfwd(net, xte_nf);
18 rmse_NNsuball_te = sqrt(mean(((yte_nf - ypred).^2)));
```

6) First of all, I think we should compare the methods in different aspects. We cannot compare them directly, because there exsits some different factors which influenced the results.

- Firstly, let we compare the linear regression and RBF network using 2 neighbouring pixels. They have the gap between two RMSEs is very small, which means we can think that they have the similiar performance of the result. However, the efficiency of linear regression is much better than the RBF's. Because we do need to choose the number of functions in RBF network. Thus, at least in this traning data set, the linear regression is better.

- Secondly, in question 4 we used whole image patch to train the linear regressor. And the performance of such regressor is better than the one which used 2 neighbouring pixels(because of less RMSE). The reason why that happens is, in my opinion, the amount of the information is larger. Thus, the cost function can get more features to do the gradient descent(although we used the pseudo-method here). But the RMSE of test data still higher than the traning one. I think this maybe the problem of over-fitting. It is a serious problem. We should decide the initial parameters carefully.

- Thirdly, let we compare neural network with linear regression. In problem 4 and 5, we used all the dimension of the traning data to train the neural network and regressor. Obviously, the method of neural network is much complicated than the linear regression. And the serious problem of overfitting occurrs in the neural network. Thus, the performance of NN is not as good as linear regression.

- Finally, there are some different schemes can strengthen the performance of above methods. 1) We can use different features of the training data in linear regression. For example, we can use the whole row on the left of the predicting pixel and whole columbn on the above of the predicting pixel. The reason is that, we can do the experiment to find which feature

is the best one of the problem. 2) We can add the prior probability. For example, bayes regression and bayes network.

Part 2

1.a) For this question, the code is been showed as follow. I used the basic method to implement the target which augment a 'bias' weight. Thus, we can avoid adding extra inputs and outputs to the likelihood function.

```
load('text_data.mat');
%a: augment the data
x_te = [x_test ones(length(x_test), 1)];
x_tr = [x_train ones(length(x_train), 1)];
```

1.b) The follow routine **t21b** is negative-log-likelihood function:

```
function [Lp, dLp_dw] = t21b(ww, xx, yy)
yy = (yy==1)*2 - 1;

sigmas = 1./(1 + exp(-yy.*(xx*ww))); % Nx1
Lp = -sum(log(sigmas));

if nargout > 1
    dLp_dw = -(((1-sigmas).*yy)' * xx)';
end
```

We can use the above function to compute the data we need. Now, let us compare the two different sets. Firstly, the accuracy of traning data is less than that of test data. It is strange. However, it might be correct because of the features of data or other factors.
For the mean log probability(MLP for short), the baseline is $\log 0.5 = -0.693147$. Thus, both the MLP are bigger than the baseline. And the performance of the test data is still better than the traning one.

$$Accuracy \quad of \quad test = 0.9083$$
$$Accuracy \quad of \quad training = 0.8344$$
$$Mean \quad log \quad probability \quad of \quad test = -0.2881$$
$$Mean \quad log \quad probability \quad of \quad training = -0.4400$$

Here is the full **matlab** code of 1.b.

```
%b
b_tr = rand(101, 1) * 10;
[X, fX, i] = minimize(b_tr, 't21b', 100, x_tr, y_train);

y_prob = 1./(1 + exp(-(x_te*X)));
tot = 0;
for i = 1 : length(y_prob)
    if (y_prob(i) > 0.5)
        temp = 1;
    else
        temp = -1;
    end
    if (temp == y_test(i))
        tot = tot + 1;
    end
end
y_prob_ml = 1./(1 + exp(-y_test.*(x_te*X)));
accuracy = tot / length(y_test);
y_mlog = mean(log(y_prob_ml));

y_prob_tr = 1./(1 + exp(-(x_tr*X)));
tot_t = 0;
for i = 1 : length(y_prob_tr)
    if (y_prob_tr(i) > 0.5)
        temp = 1;
    else
        temp = -1;
    end
    if (temp == y_train(i))
        tot_t = tot_t + 1;
    end
end
y_prob_tr_ml = 1./(1 + exp(-y_train.*(x_tr*X)));
accuracy_t = tot_t / length(y_train);
y_mlog_t = mean(log(y_prob_tr_ml));
%standard error
std_err_a = sqrt(accuracy.*(1-accuracy)/size(y_prob, 1));
std_err_atr = sqrt(accuracy_t.*(1-accuracy_t)/size(y_prob_tr, 1));
std_err = sqrt(var(log(y_prob_ml))/size(y_prob, 1));
std_err_tr = sqrt(var(log(y_prob_tr_ml))/size(y_prob_tr, 1));
```

1.c) Here we limited the data to first 100 traning cases. The accuracy and the mean log probability descend dramatically. I think the reason is that the traning cases is not enough to train the log regressor well. For this example, the first 100 cases cannot represent the general feature of the whole data sets.

$$Accuracy \quad of \quad test = 0.7520$$
$$Mean \quad log \quad probability \quad of \quad test = -inf$$

Here is the full **matlab** code of 1.c.

```
load('text_data.mat');

x_te = [x_test ones(length(x_test), 1)];
x_tr = [x_train ones(length(x_train), 1)];

b_tr = ones(101, 1) ;
[X, fX, i] = minimize(b_tr, 't21b', 10000, x_tr(1:100,:), ...
    y_train(1:100,:));

y_prob_tr = 1./(1 + exp(-(x_te*X)));
tot_t = 0;
for i = 1 : size(y_prob_tr, 1)
    if (y_prob_tr(i) > 0.5)
        temp = 1;
    else
        temp = -1;
    end
    if (temp == y_test(i))
        tot_t = tot_t + 1;
    end
end
accuracy = tot_t / size(y_test, 1);
y_mlog_t = mean(log(y_prob_tr));
```

2.a) For this question, we need to compute the derivatives of both $\mathbf{w}$ and ϵ. Such result can be used as the function which compute the log regression with the label noise.
The original function showed as below:

$$P(y|\mathbf{x}, \mathbf{w}, \epsilon) = (1-\epsilon)\sigma(y\mathbf{w}^T\mathbf{x}) + \epsilon/2, \qquad y \in \{-1, +1\}$$

Firstly, we should compute the derivative of $\mathbf{w}$:

$$\mathcal{L}(\mathbf{w}, \epsilon) = \sum_{n=1}^{N} \log P(y^{(n)}|x^{(n)}, \mathbf{w}, \epsilon)$$
$$\nabla_w \mathcal{L}(\mathbf{w}, \epsilon) = \frac{\partial}{\partial w} \sum_{n=1}^{N} \log P(y^{(n)}|x^{(n)}, \mathbf{w}, \epsilon)$$
$$\nabla_w \mathcal{L}(\mathbf{w}, \epsilon) = \frac{\partial y^{(n)}\mathbf{w}^T x^{(n)}}{\partial w} \frac{\partial}{\partial y^{(n)}\mathbf{w}^T x^{(n)}} \sum_{n=1}^{N} \log((1-\epsilon)\sigma(y^{(n)}\mathbf{w}^T\mathbf{x}^{(n)}) + \epsilon/2)$$
$$\nabla_w \mathcal{L}(\mathbf{w}, \epsilon) = y^{(n)}\mathbf{x}^{(n)} \sum_{n=1}^{N} \left[\frac{(1-\epsilon)\sigma(y^{(n)}\mathbf{w}^T x^{(n)})(1-\sigma(y^{(n)}\mathbf{w}^T x^{(n)}))}{(1-\epsilon)\sigma(y^{(n)}\mathbf{w}^T\mathbf{x}^{(n)}) + \epsilon/2} \right]$$

Secondly, we should compute the derivative of ϵ :

$$\mathcal{L}(\mathbf{w}, \epsilon) = \sum_{n=1}^{N} \log P(y^{(n)}|x^{(n)}, \mathbf{w}, \epsilon)$$
$$\nabla_\epsilon \mathcal{L}(\mathbf{w}, \epsilon) = \frac{\partial}{\partial \epsilon} \sum_{n=1}^{N} \log P(y^{(n)}|x^{(n)}, \mathbf{w}, \epsilon)$$
$$\nabla_\epsilon \mathcal{L}(\mathbf{w}, \epsilon) = \frac{\partial}{\partial \epsilon} \sum_{n=1}^{N} \log((1-\epsilon)\sigma(y^{(n)}\mathbf{w}^T\mathbf{x}^{(n)}) + \epsilon/2)$$
$$\nabla_\epsilon \mathcal{L}(\mathbf{w}, \epsilon) = \sum_{n=1}^{N} \left[\frac{\frac{1}{2} - \sigma(y^{(n)}\mathbf{w}^T x^{(n)})}{(1-\epsilon)\sigma(y^{(n)}\mathbf{w}^T\mathbf{x}^{(n)}) + \epsilon/2}\right]$$

Here are the **matlab** code of the above two derivatives.

```
function [Lp, dLp_dw] = t22f1(ww, eps, xx, yy)
yy = (yy==1)*2 - 1;
% derivative of w
sigmas = 1./(1 + exp(-yy.*(xx*ww))); % Nx1
Lp = sum(log((1-eps)*sigmas+(eps/2)));

if nargout > 1
    dLp_dw = ...
        (((1-eps)*sigmas.*(1-sigmas)./((1-eps)*sigmas+eps/2).*yy)' ...
        * xx)';
end
```

```
function [Lp, dLp_dw] = t22f2(eps, ww, xx, yy)
yy = (yy==1)*2 - 1;
%derivative of epsilon
sigmas = 1./(1 + exp(-yy.*(xx*ww))); % Nx1
Lp = sum(log((1-eps)*sigmas+(eps/2)));

if nargout > 1
    dLp_dw = sum((0.5-sigmas)./((1-eps)*sigmas+(eps/2)));
end
```

Now we can use the routine **checkgrad.m** to check the derivatives of above two functions. This routine provides the difference between my gradients and their numerial approximation. However, because of the precision, I found that there might not exsit the largest difference.

```
load('text_data.mat');
b_tr = rand(101, 1) * 10;
x_te = [x_test ones(length(x_test), 1)];
x_tr = [x_train ones(length(x_train), 1)];
checkgrad('t22f1', b_tr, 1e-3, rand(1, 1),  x_tr, y_train);
checkgrad('t22f2', rand(1, 1), 1e-3, b_tr,  x_tr, y_train);
```

2.b) In this problem, we will write the parameter of ϵ as the result of taking the logistic sigmod of an unconstrained parameter a:

$$\epsilon = \sigma(a) = \frac{1}{1+exp(-a)}$$

Now, we compute the derivative of a :

$$\mathcal{L}(\mathbf{w}, a) = \sum_{n=1}^{N} \log P(y^{(n)}|x^{(n)}, \mathbf{w}, \sigma(a))$$
$$\nabla_a \mathcal{L}(\mathbf{w}, a) = \frac{\partial}{\partial a} \sum_{n=1}^{N} \log P(y^{(n)}|x^{(n)}, \mathbf{w}, \sigma(a))$$
$$\nabla_a \mathcal{L}(\mathbf{w}, a) = \frac{\partial}{\partial a} \sum_{n=1}^{N} \log((1-\sigma(a))\sigma(y^{(n)}\mathbf{w}^T\mathbf{x}^{(n)}) + \sigma(a)/2)$$
$$\nabla_a \mathcal{L}(\mathbf{w}, a) = \sum_{n=1}^{N} \left[\frac{\sigma(a)(1-\sigma(a))(\frac{1}{2}-\sigma(y^{(n)}\mathbf{w}^T x^{(n)}))}{(1-\sigma(a))\sigma(y^{(n)}\mathbf{w}^T\mathbf{x}^{(n)})+\sigma(a)/2)}\right]$$

Here is the full **matlab** code of the function of 2.b.

```
load('text_data.mat');
%a: augment the data
x_te = [x_test ones(length(x_test), 1)];
x_tr = [x_train ones(length(x_train), 1)];

[X, fX, i] = minimize([ones(101,1)',0]', @t23f, 10000, x_tr, ...
    y_train);

w = X(1:  end -1);
eps = 1./(1+exp(-w(end)));
y_prob = 1./(1 + exp(-(x_te*w)));
tot = 0;
for i = 1 : length(y_prob)
    if (y_prob(i) > 0.5)
        temp = 1;
    else
        temp = -1;
    end
    if (temp == y_test(i))
        tot = tot + 1;
    end
end

y_prob_ml = 1./(1 + exp(-y_test.*(x_te*w)));
accuracy = tot / length(y_test);
y_mlog = mean(log(y_prob_ml));
```

Now, we use the program to compute accuracy and mean log probability(MLP for short). We can get the new result as below:

$$accuracy = 0.9108$$
$$mean \quad log \quad probability = -1.3486$$

For this result, we found that the MLP is less than the last problem. And the accuracy is similiar to that problem. We can observe the predicting value of y, and the value are much more extreme. The reason why that happens is that, I think, we use the constrained parameter to build the sigmoid function of ϵ.
On the other hand, because the training data of the problem may not always reliable. Thus, in this problem, we lead into the label noise model. If $s_n = 1$ then the label is chosen with a uniform random choice and ignore the original $\mathbf{x}$ features. Therefore, we may guess the correct choice which is not reliable by balance the original function with constrained parameter.

Here is the full **matlab** code of the function of derivative.

```
function [Lp, dLp_dw] = t22f1n(w, x, y)
y = (y==1)*2 - 1;
a = w(end);
ww = w(1:  end - 1);
eps = 1./(1+exp(-a));
sigmas = 1./(1 + exp(-y.*(x*ww))); % Nx1
Lp = -sum(log((1-eps)*sigmas+(eps/2)));

a = -(((1-eps)*sigmas.*(1-sigmas)./((1-eps)*sigmas+eps/2).*y)' * ...
    x)';
b = -sum((0.5-sigmas)*eps*(1-eps)./((1-eps)*sigmas+(eps/2)));
dLp_dw = [a', b]';
```

3.a) The largest log-likelihood that any model can given a training set of N binary outcomes equal to the log-likelihood if a model were somehow able to predict every training label correctly. Thus, it is just 1.

$$\text{Let } w = 0 \text{ in}$$
$$\log P(\epsilon, \mathbf{w}, \log\lambda|data) = \mathcal{L}(\mathbf{w}, \epsilon) - \lambda\mathbf{w}^T\mathbf{w} + \tfrac{D}{2}\log\lambda + const$$

We can get following result:

$$\log P(\epsilon, 0, \log\lambda|data) = \tfrac{1}{2} + \tfrac{D}{2}\log\lambda + const$$

By questions 1 and 2 in part 2, we have already known the derivative of the $\mathcal{L}(\mathbf{w}, \epsilon)$ showed as below:

$$\nabla_\epsilon \mathcal{L}(\mathbf{w}, \epsilon) = \sum_{n=1}^{N} \left[\frac{\frac{1}{2} - \sigma(y^{(n)} \mathbf{w}^T x^{(n)})}{(1-\epsilon)\sigma(y^{(n)} \mathbf{w}^T \mathbf{x}^{(n)}) + \epsilon/2} \right]$$

Now, we need to solve the extrema of above function:

$$\begin{gathered} 0 = \sum_{n=1}^{N} \left[\frac{\frac{1}{2} - \sigma(y^{(n)} \mathbf{w}^T x^{(n)})}{(1-\epsilon)\sigma(y^{(n)} \mathbf{w}^T \mathbf{x}^{(n)}) + \epsilon/2} \right] \\ \vdots \\ \sigma(y \mathbf{w}^T x) = 0 \\ w = 0 \end{gathered}$$

To sum up, the function $\log \lambda$ is the strictly increasing function. Thus, the log-posterior above is globally maximized by setting the weights to zero and making λ infinite.

3.b) For this question, we should implement the function and make the choice of the parameters. The parameters of the slice_sample.m should be decided carefully. For this question, I set all the value of w is 1, and the value of ϵ and $\log \lambda$ are 0.5 and $\log 1$. The width which I set is below 0.2(0.1 for plot the scatter plot). The reason is that, we can get more priciser result.
Figure 6 shows the scatter plot of $\log \lambda$ against ϵ. We can assert that, in this question, our posterior beliefs about them are independent. Because we do the simple addition of the two different parts $\mathcal{L}(w, \epsilon)$ and $\frac{D}{2} \log \lambda$. Therefore they do not affect each other.

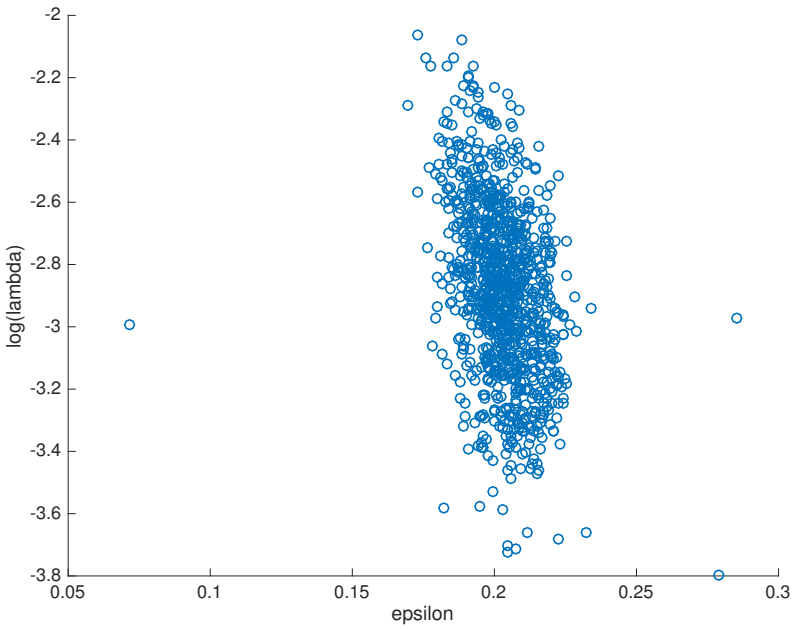

Figure 6: The log(lambda) and epsilon

```
load('text_data.mat');
%a: augment the data
x_te = [x_test ones(length(x_test), 1)];
x_tr = [x_train ones(length(x_train), 1)];

samples = slice_sample(1000, 0, @t23f, [ones(101,1)', 0.5, ...
    log(1)]', 0.2, true, x_tr, y_train);

xlabel('epsilon');
ylabel('log(lambda)');
hold on;
scatter(samples(end-1,:), samples(end,:));
```

```
function logP = t23f(w, x, y)
y = (y==1)*2 - 1;
loglambda = w(end);
eps = w(end-1);
ww = w(1:  end - 2);
sigmas = 1./(1 + exp(-y.*(x*ww))); % Nx1
Lp = sum(log((1-eps)*sigmas+(eps/2)));
lambda = exp(loglambda);
logP = Lp - lambda*(ww')*ww + length(w)/2*loglambda ;
```

3.c) Because of the following estimate:

$$\int f(x)P(x)\,dx \approx \frac{1}{S}\sum_{s=1}^{S} f(x^{(s)}).$$

Above equation tells us that we should compute all of the prediction by each $\vec{w}$ and do the sum of the results (for this question, we should compute 1000 results of each sample). Finally, we compute the mean of it.

We can compute the data, and get the accuracy 0.9151. So it is better than the previous model. Because we compute the mean of the 1000 samples. And some of the prediction of the samples maybe very good. That's why we got the better result.

Here is the full **matlab** code of 3.

```
load('text_data.mat');
%a: augment the data
x_te = [x_test ones(length(x_test), 1)];
x_tr = [x_train ones(length(x_train), 1)];

samples = slice_sample(1000, 0, @t23f, [ones(101,1)', 0.5, ...
    log(1)]', 0.2, true, x_tr, y_train);

xlabel('epsilon');
ylabel('log(lambda)');
hold on;
scatter(samples(end-1,:), samples(end,:));

for i = 1 : 1000
    y_pred = y_pred + x_te * samples(1:end-2,i);
end

y_prob = mean(y_pred, 2);

tot = 0;
for i = 1 : length(y_prob)
    if (y_prob(i) > 0.5)
        temp = 1;
    else
        temp = -1;
    end
    if (temp == y_test(i))
        tot = tot + 1;
    end
end
```

```
30
31  accuracy = tot / length(y_prob);
```

Conclusion

What I have learned Basically, I learned how to use **matlab** to write some simple routines. Futhur more, the topic of comparing the different models and how to set the initial guess data are all very important. The ability to compare the two or more models is what I have grasped.

Where should I make the improvement I have to say that this is the most tough coursework which I have seen. I work hard and spend so much time on it. However, the methods which I have used are not good enough. And some of the princples cannot be undertood deeply by me. Thus, I should do more on the theoritical part. Ann then, I think I can report better. The best practice is that I should study more resource both on papers and examples.