UNIVERSITY OF VICTORIA

Department of Electrical and Computer Engineering

ECE 503 Optimization for Machine Learning

LABORATORY REPORT

Experiment No: *Lab 3*

Title: *Breast Cancer Diagnosis via Logistic Regression*

Date of Experiment: *Oct.31st, 2025*

Report Submitted on: *Nov.5nd, 2025*

To: *Ruilin Wang*

Names: *Da Zhang (V01062902)*

1. Objective

The goal of this lab experiment is to build and employ a binary classification model (with the normalized Softmax Cost Function) to diagnose breast cancer using the Wisconsin Diagnostic Breast Cancer (WDBC) dataset. Each sample represents a patient and is characterized by 30 real-valued cellular features computed from digitized images of Fine Needle Aspirates (FNA) of breast masses. The objective is to classify each sample as either benign or malignant based on these features.

2. Introduction

The method we employed in this experiment is a binary classification model (with regularized Softmax Cost Function), where the data is split into a training set with the first 285 samples and a training set with the last 284 samples, the optimal augmented weight parameters for the 30 features and 1 biases term is then computed based on minimizing the Softmax Cost Function, $E_{LR}(\hat{w}) = \frac{1}{P}\sum_{p=1}^{P} \log\left(1 + e^{-y_p \widehat{w^T \hat{x}_p}}\right) + \frac{\mu}{2}|\hat{w}|_2^2$ *,on normalized training set to acquire the optimal classifier,* $\tilde{y} = \text{sign}(w^{*T}x + b^*) = \text{sign}(\widehat{w^{*T}}x)$*, using Gradient Descend algorism with Back Tracking Line Search Technique of finding the step sizes on the normalized training set. The prediction models (optimal classifiers) are produced based on different combinations of regularization coefficient u and the number of GD iterations K. After that, the optimal classifiers, decision boundaries, with the optimal augmented weight parameters are applied on the normalized testing set to classify the testing set. Lastly, the models' performances is therefore evaluated in terms of confusion matrixand accuracy.*

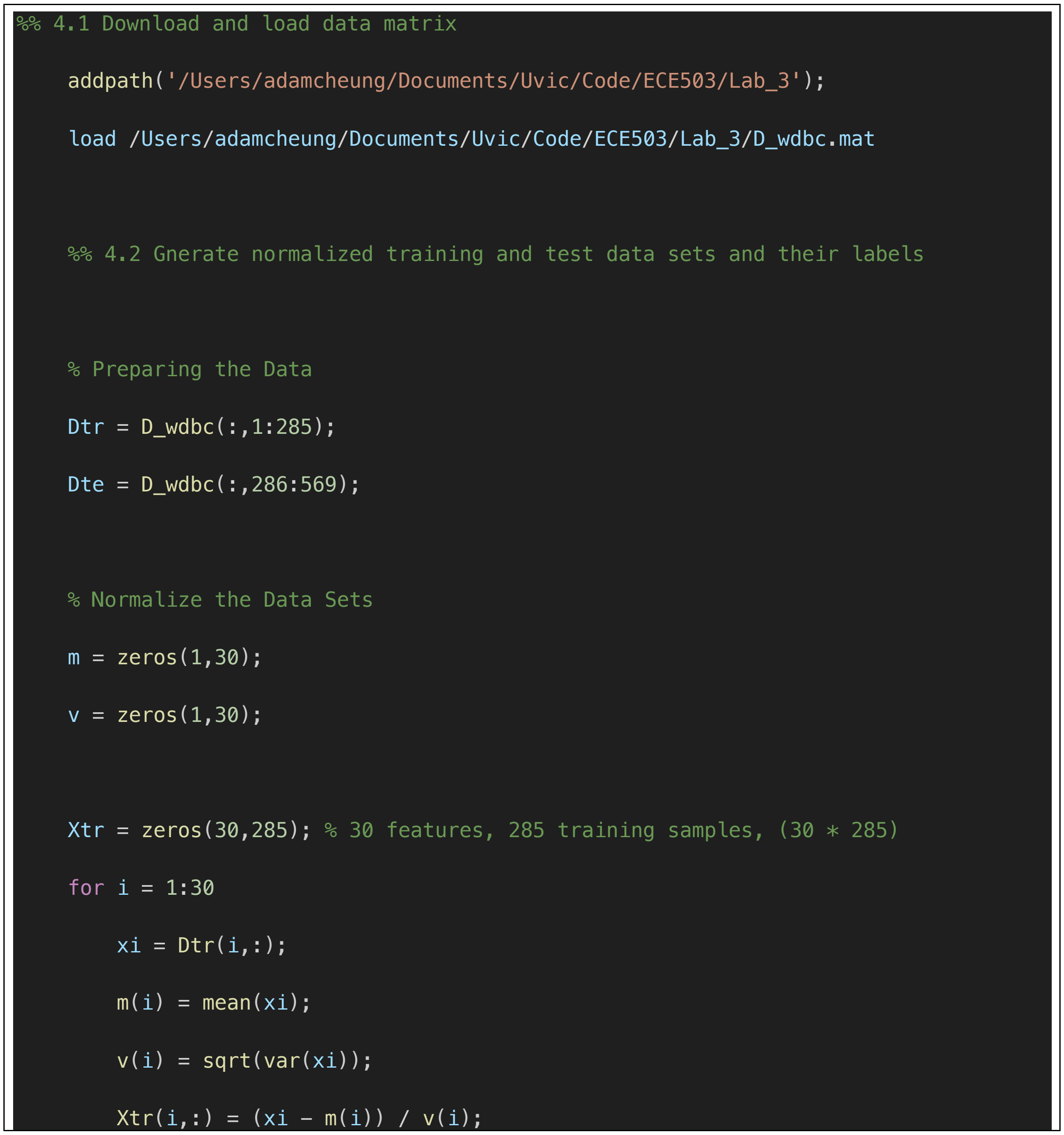
3. Implementation and Results

3.1. Implementation

1. Load resources: bring D_wdbc.mat into workspace and add the LCBR helper folder to the MATLAB path.
2. Partition the data matrix into training (Dtr) and testing (Dte) subsets, then normalize each feature using the training-set means and standard deviations to create Xtr and Xte.
3. Extract the traiing and testing labels (ytr, yte) and build the augmented training matrix Dtr_hat that stacks normalized features with labels for downstream routines.
4. Invoke grad_desc with the pre-prepared regularized softmax cost and gradient functions (f_wdbc, g_wdbc) over the four regularization/iteration configurations to learn weight vectors ws_GD.

5. Form the augmented test matrix Xte_hat, apply each learned weight vector, and take the sign to obtain prediction labels for all 284 test samples.
6. Evaluate the test-set prediction performance on the four differently-configured models and compute confusion matrix and accuracy.

3.2. MATLAB code



```matlab
%% 4.1 Download and load data matrix

    addpath('/Users/adamcheung/Documents/Uvic/Code/ECE503/Lab_3');

    load /Users/adamcheung/Documents/Uvic/Code/ECE503/Lab_3/D_wdbc.mat


    %% 4.2 Gnerate normalized training and test data sets and their labels


    % Preparing the Data

    Dtr = D_wdbc(:,1:285);

    Dte = D_wdbc(:,286:569);


    % Normalize the Data Sets

    m = zeros(1,30);

    v = zeros(1,30);


    Xtr = zeros(30,285); % 30 features, 285 training samples, (30 * 285)

    for i = 1:30

        xi = Dtr(i,:);

        m(i) = mean(xi);

        v(i) = sqrt(var(xi));

        Xtr(i,:) = (xi - m(i)) / v(i);
    end


    Xte = zeros(30,284); % 30 features, 284 test samples, (30 * 284)

    for i = 1:30

        xi = Dte(i,:);

        Xte(i,:) = (xi - m(i))/v(i);

    end


    ytr = Dtr(31,:); % 285 Training Labels, (1 * 285)

    yte = Dte(31,:); % 284 Test Labels, (1 * 284)


    Dtr_hat = [Xtr; ytr];


    %% 4.3 Prepare functions for evaluating the regularized cost function and its
gradient

    % Refer to f_wdbc.m and g_wdbc created for calculating regularized softmax cost
function and its gradient


    %% 4.4 Modify grad_desc.m according to the Remark


    %% 4.5 Set the initial point w0 = 0 and tun the code to obtain optimized parameters
for the following settings


    % i) w0 = 0 "zeros(31,1)", u = 0 and K = 10

    [ws1_GD, ELRs1_GD, k1_GD] = grad_desc('f_wdbc','g_wdbc',zeros(31,1), Dtr_hat, 0,
10);
```

```
    % ii) w0 = 0 "zeros(31,1)", u = 0.1 and K = 10

    [ws2_GD,  ELRs2_GD,  k2_GD]  =  grad_desc('f_wdbc','g_wdbc',zeros(31,1),  Dtr_hat,
0.1, 10);



    % iii) w0 = 0 "zeros(31,1)", u = 0 and K = 30

    [ws3_GD, ELRs3_GD, k3_GD] = grad_desc('f_wdbc','g_wdbc',zeros(31,1), Dtr_hat, 0,
30);



    % iv) w0 = 0 "zeros(31,1)", u = 0.075 and K = 30

    [ws4_GD,  ELRs4_GD,  k4_GD]  =  grad_desc('f_wdbc','g_wdbc',zeros(31,1),  Dtr_hat,
0.075, 30);



    %% 4.6 Classify the 284 test samples using the optimized models above

    Xte_hat = [Xte; ones(1,size(Xte,2))];



    % Predict test sample using model i) and report confusion matrix & accuracy

    y_pred_1 = sign(ws1_GD' * Xte_hat);



    labelToIdx = @(lbl) (lbl + 3) / 2;      % -1 -> 1, +1 -> 2

    C1_te = zeros(2, 2);

    for i = 1:numel(y_pred_1)

        gtIdx   = labelToIdx(yte(i));

        predIdx = labelToIdx(y_pred_1(i));

        C1_te(predIdx, gtIdx) = C1_te(predIdx, gtIdx) + 1;

    end

    disp(C1_te);
```

```
Accuracy_te_1 = trace(C1_te) / sum(C1_te(:));

disp(Accuracy_te_1);


% Predict test sample using model ii) and report confusion matrix & accuracy

y_pred_2 = sign(ws2_GD' * Xte_hat);


labelToIdx = @(lbl) (lbl + 3) / 2;      % -1 -> 1, +1 -> 2

C2_te = zeros(2, 2);

for i = 1:numel(y_pred_2)

    gtIdx   = labelToIdx(yte(i));

    predIdx = labelToIdx(y_pred_2(i));

    C2_te(predIdx, gtIdx) = C2_te(predIdx, gtIdx) + 1;

end

disp(C2_te);


Accuracy_te_2 = trace(C2_te) / sum(C2_te(:));

disp(Accuracy_te_2);


% Predict test sample using model iii) and report confusion matrix & accuracy

y_pred_3 = sign(ws3_GD' * Xte_hat);


labelToIdx = @(lbl) (lbl + 3) / 2;      % -1 -> 1, +1 -> 2

C3_te = zeros(2, 2);

for i = 1:numel(y_pred_3)
```

```
    gtIdx   = labelToIdx(yte(i));

    predIdx = labelToIdx(y_pred_3(i));

    C3_te(predIdx, gtIdx) = C3_te(predIdx, gtIdx) + 1;

end

disp(C3_te);


Accuracy_te_3 = trace(C3_te) / sum(C3_te(:));

disp(Accuracy_te_3);


% Predict test sample using model iv) and report confusion matrix & accuracy

y_pred_4 = sign(ws4_GD' * Xte_hat);


labelToIdx = @(lbl) (lbl + 3) / 2;      % -1 -> 1, +1 -> 2

C4_te = zeros(2, 2);

for i = 1:numel(y_pred_4)

    gtIdx   = labelToIdx(yte(i));

    predIdx = labelToIdx(y_pred_4(i));

    C4_te(predIdx, gtIdx) = C4_te(predIdx, gtIdx) + 1;

end

disp(C4_te);


Accuracy_te_4 = trace(C4_te) / sum(C4_te(:));

disp(Accuracy_te_4);
```

3.2.1 f_wdbc.m

```
function f = f_wdbc(w,D,mu)
P = size(D,2);
Xtr_hat = [D(1:end-1,:); ones(1,P)];
ytr = D(end,:);
function_sum = 0;
for p = 1:P
    xp = Xtr_hat(:,p);
    yp = ytr(p);
    function_sum = function_sum + log(1 + exp(-yp * (w.' * xp)));
end

f = (function_sum / P) + 0.5 * mu * (w.' * w);
```

3.2.2 g_wdbc.m

```
function g = g_wdbc(w,D,mu)
P = size(D,2);
Xtr_hat = [D(1:end-1,:); ones(1,P)];
ytr = D(end,:);
gradient_sum = zeros(size(w));
for p = 1:P
    xp = Xtr_hat(:,p);
    yp = ytr(p);
    gradient_sum = gradient_sum + (yp * xp) / (1 + exp(yp * (w.' * xp)));
end

g = (mu * w) - (gradient_sum / P);
```

3.2.3 grad_desc.m

```
% To implement the gradient descent algorithm.
% Example: [xs,fs,k] = grad_desc('f_rosen','g_rosen',[0; 2],D,mu,50);
function [xs,fs,k] = grad_desc(fname,gname,x0,D,mu,K)
format compact
format long
xk = x0;
for k = 1:K
  gk = feval(gname,xk,D,mu);
  dk = -gk;
  ak = bt_lsearch2019(xk,dk,fname,gname,D,mu);
  xk = xk + ak*dk;
end
disp('solution:')
xs = xk
disp('objective function at solution point:')
fs = feval(fname,xs,D,mu)
format short
disp('number of iterations performed:')
```

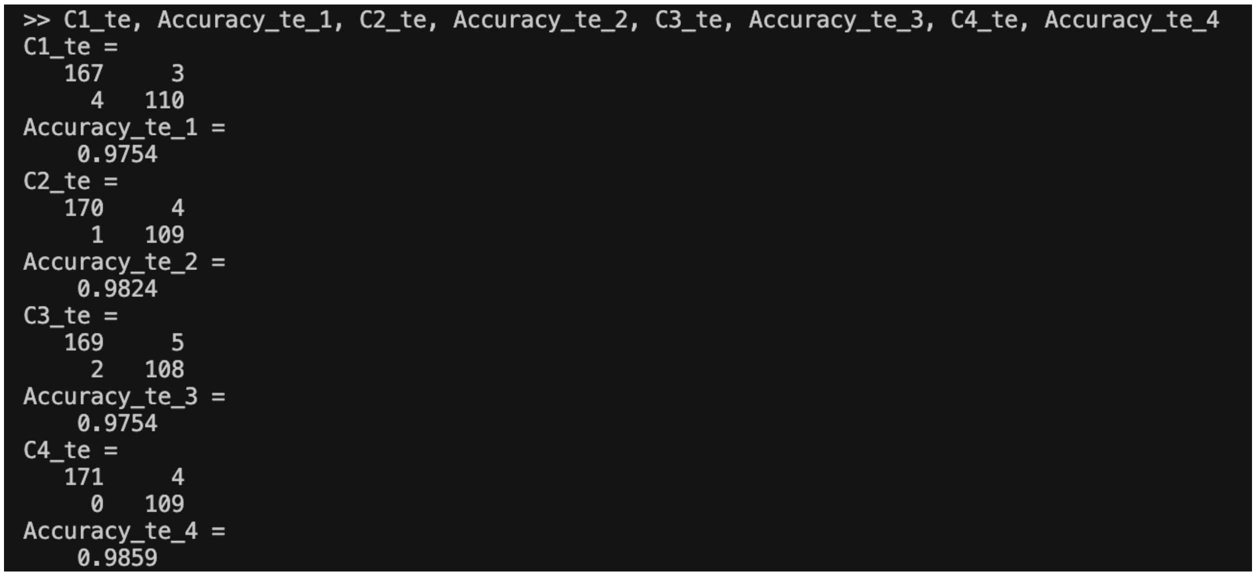
```
k
```

3.2.4 bt_lsearch2019.m

```
function a = bt_lsearch2019(x,d,fname,gname,p1,p2)
rho = 0.1;
gma = 0.5;
x = x(:);
d = d(:);
a = 1;
xw = x + a*d;
parameterstring ='';
if nargin == 5
   if ischar(p1)
      eval([p1 ';']);
   else
      parameterstring = ',p1';
   end
end
if nargin == 6
   if ischar(p1)
      eval([p1 ';']);
   else
      parameterstring = ',p1';
   end
   if ischar(p2)
      eval([p2 ';']);
   else
      parameterstring = ',p1,p2';
   end
end
eval(['f0 = ' fname '(x' parameterstring ');']);
eval(['g0 = ' gname '(x' parameterstring ');']);
eval(['f1 = ' fname '(xw' parameterstring ');']);
t0 = rho*(g0'*d);
f2 = f0 + a*t0;
er = f1 - f2;
while er > 0
     a = gma*a;
     xw = x + a*d;
     eval(['f1 = ' fname '(xw' parameterstring ');']);
     f2 = f0 + a*t0;
     er = f1 - f2;
end
if a < 1e-5
   a = min([1e-5, 0.1/norm(d)]);
end
```

3.3 Results

Results for 4.6.1 Report the confusion matrix for each case



```
>> C1_te, Accuracy_te_1, C2_te, Accuracy_te_2, C3_te, Accuracy_te_3, C4_te, Accuracy_te_4
C1_te =
   167     3
     4   110
Accuracy_te_1 =
    0.9754
C2_te =
   170     4
     1   109
Accuracy_te_2 =
    0.9824
C3_te =
   169     5
     2   108
Accuracy_te_3 =
    0.9754
C4_te =
   171     4
     0   109
Accuracy_te_4 =
    0.9859
```

Fig 1. Confusion Matrix and Classification Accuracy for classifier applied on training data with different model configurations (u = 0, K = 10; u = 0.1, K = 10; u = 0, K = 30; u = 0.075, K = 30)

	Actual -1	**Actual +1**
Predicted -1	Truth Negative (TN)	False Negative (FN)
Predicted +1	False Positive (FP)	Truth Positive (TP)

Fig.2 Reference for Confusion Matrices constructed

Results for 4.6.2 Comment on the results obtained

Across all four runs the diagonals dominates, so predictions and ground truth align for around 97% to 99% of the 284 test samples. Model ii) and Model iv) edge ahead that both misclassify only 5 example each, giving accuracies of 0.9824 and 0.9859 respectively. Model i) and Model iii) tie at 0.9765 accuracy with 7 mistakes apiece (3 false negatives, 4 false positives for model i) and 5 false negatives, 2 false positives for model iii)). Interpret (1,2) entry of the confusion as false negatives (actual +1, predicted -1) and (2,1) as false positives (actual -1, predicted +1). Minimizing false negatives may matter most if class +1 is malignant, so model ii) and model iv) are more attractive because they reduce that count to four or fewer.

4. Discussion

From the all model variants with different configurations of regularization coefficient and iteration choices, the results produced are different, meaning regularization and iterations choices matter. The two configurations with moderates regularization (u = 0.1 and u = 0.075) and higher iteration count reduce error slightly compared with the unregularized/shot-run baselines, indicating that guarding against overfitting and allowing extra descent steps both help.

Error asymmetry is small but important: false negatives (upper-right entries of the confusion matrices) drop from 3-5 to 4 or fewer in the better models; if "+1" denotes malignant cases, those entries quantify missed malignancies and are the key safety risk in the event of breast cancer diagnosis.

False Positive (lower-left entries) stay around 0-4 across runs; even the best models still flag a few benign samples are malignant, which could prompt unnecessary follow-up compared to missing a malignancy.

By comparing the results about misclassification, recall and specificity:

Model i):

7 misclassification (4 FP, 3 FN); recall = 110/113 = 0.9735; specificity = 167/171 = 0.9766

Model ii):

5 misclassification (1 FP, 4 FN); recall = 109/113 = 0.9646; specificity = 170/171 = 0.9942

Model iii):

7 misclassification (2 FP, 5 FN); recall = 108/113 = 0.9558; specificity = 169/171 = 0.9883

Model iv):

4 misclassifications (0 FP, 4FN); recall = 109/113 = 0.9646; specificity = 171/171 = 1

All four models keep error low, but model iv) is the standout; it makes only four mistakes, achieves perfect specificity and still maintains respectable recall (0.9646). Model ii) trades one extra false positive for the same recall as model iv); depending on how costly a false alarm is compared to a missed malignancy, either configuration could be preferable. Model i) and iii) with no regularizations lag slightly; their recall drops to 97% and 96% respectively, and the extra false positives/negatives bring total misclassification up to seven, so they offer no advantage over ii) or iv).

5. Conclusion

In this experiment, we successfully implemented a binary classification model with Regularized Softmax Cost Function to classify breast cancer cases using GD with BTLS. GD Regularized Softmax training produces consistently strong set performance on the WDBC split that every configuration of the model yields a sharply diagonal confusion matrix and accuracy in the range of 97% to 99%, confirming descent generalization beyond the training set. Increasing the iteration budget and adding moderate regularization improves robustness: model ii) and iv) reduce misclassification to 5 and 4 respectively, showing that a controlled amount of shrinkage plus longer optimization beats the unregularized baselines, where the regularization coefficients are set to zero. The dominant risk remains false negatives (missed malignancies); model iv) strikes the best balance by eliminating false positives entirely while keeping recall near 96%, whereas model ii) delivers a similar recall at the cost of one false alarm – acceptable if sensitivity is slightly relaxed in favour of specificity. Model i) and iii) offers no practical benefit: their extra false positives/negatives dilute recall to 96% - 97%, without improving specificity or accuracy, so they are dominated by the regularized alternatives. Overall, the experiment supports deploying the u = 0.075 and K = 30 configuration (Model iv))when false positives are costly or the u = 0.1and K = 10 configuration (Model ii)) when a tiny increase in specificity can be traded for an extra safety margin against missed detections..

6. References

None.