COMP6036: Advanced Machine Learning
Feature Selection Challenge

Ashley J. Robinson
ajr2g10@ecs.soton.ac.uk

School of Electronics and Computer Science
University of Southampton

30th April, 2014

Abstract

All algorithms implemented in this report is written by the author for the MATLA programming language. Initial testing with a linear perceptron using all features in the input space yields good results on the dexter and gisette datasets. The objective is to further optimise the performance on these datasets.

1 Introduction

Everything implemented has been developed from scratch for the MATLAB programming language. Neural networks were chosen as the starting point for investigation as per the author's interests. Throughout datasets taken from and submissions are made to a feature selection challenge (Gunn, 2003).

Each dataset has been divided in three section for training, validation and test submission. The final partition labels remain hidden from the development process. The discrete class labels, t, and the classifier outputs, y, are described by equation (1). Keeping to these output values requires using the thresholding function Θ in equation (2).

$$y, t \in \{-1, +1\} \tag{1}$$

$$\Theta(a) = \begin{cases} +1, & a > 0 \\ -1, & a \leq 0 \end{cases} \tag{2}$$

2 Classifier Build and Test

Artificial Neural Networks (ANNs) have been chosen as the classifier architectures. ANNs are efficient learning machines with biological inspiration (Bishop, 2006). They are parametric models which can be

adapted during supervised training and are scalable in complexity. The error space can be considered as a function of parameters and the training data, $E(w|D)$, therefore many different optimisation algorithms can be used can used to train the model.

2.1 Perceptron

Figure 1 holds the most simple manifestation of an ANN called a perceptron. Each feature of an input pattern is weighted and then summed, along with a bias term, to produce an output. This output passes through a step activation function for binary classification. Capable of producing a linear separation boundary the method serves as starting point for further investigation into the data. The architecture is vectorised in equation. (3) to classify an entire dataset. The offset parameter, b, is included in the matrix and the input patterns are fixed with the unit feature to assist with training the classifier.

Batch training is done iteratively using equation (4) where the error between the output and labels is propagates back to the weights using the transpose of the input patterns. A training parameter, η, hes been manually tuned to 10^{-4}. Figure 2 shows the performance on the dexter dataset over the training process. Tested upon the available datasets the only two reasonable performances were on dexter and gisette which is shown in tables **??** and **??** respectively. All code for the perceptron is held in appendix B.1

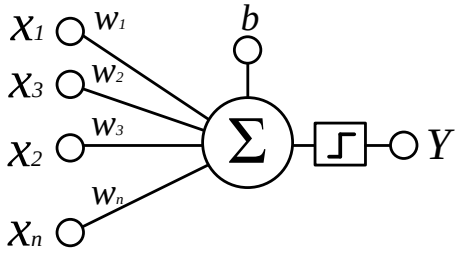


FIGURE 1: Perceptron architecture.

$$\mathbf{Y} = \Theta(\mathbf{X}.\mathbf{W}) \tag{3}$$

$$\mathbf{W}_{t+1} = \mathbf{W}_t + \eta * (\mathbf{Y} - \mathbf{T})\mathbf{X}^T \tag{4}$$

Dataset	**Measure**	**Train**	**Valid**	**Test**
Dexter	BER	0.0000	0.0633	0.0735
Dexter	AUC	1.0000	0.9779	0.9767
Gisette	BER	0.0445	0.0430	0.0432
Gisette	AUC	0.9889	0.9894	0.9907

TABLE 1: Perceptron results on dexter and gisette. Submission ID 5518 in Appendix A.

2.2 Multi Layer Perceptron (MLP)

Section 2.1 can be expanded by concatenating the architecture to produce a multi layer perceptron. They contain hidden layers and when used with non-linear activation functions can produced

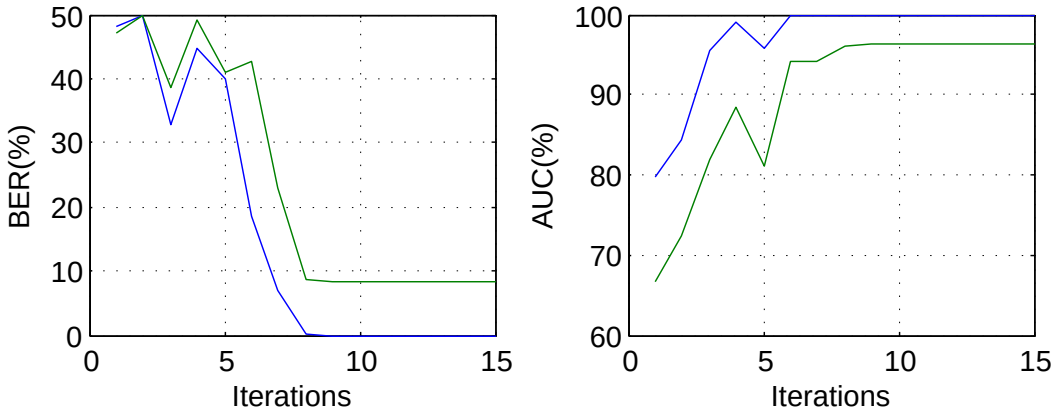


FIGURE 2: Perceptron training on the dexter dataset.

a non-linear separation boundary (Prügel-Bennett, 2001). Only one hidden layer is implemented for simplicity as shown in figure 3. The hidden nodes use an offset *tanh* and output uses the Θ function.

When testing the MLP all the features are used with 50 hidden nodes. Table **??** is trained using a value of η set to 10^{-4} yet shows no improvement. The training set is learnt quickly and no improvement is shown in the validation set. Table **??** does show an improvement over the perceptron with η set to 10^{-5}. The taring set is not full learnt and a steady increase in performance is shown during training in figure 4. Code for this implementation is held in appendix B.2

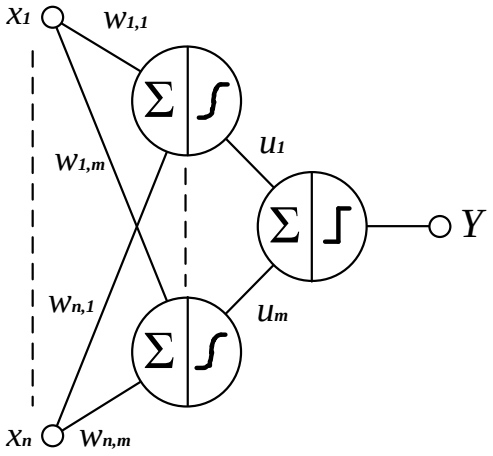


FIGURE 3: MLP architecture.

$$y = \Theta\left(b + \sum_{i=1}^{m} u_i \tanh\left(c_i + \sum_{j=1}^{n} x_j w_{j,i}\right)\right) \tag{5}$$

Dataset	Measure	Train	Valid	Test
Dexter	BER	0.0000	0.0833	0.1110
Dexter	AUC	1.0000	0.9704	0.9632
Gisette	BER	0.0120	0.0280	0.0312
Gisette	AUC	0.9955	0.9864	0.9839

TABLE 2: MLP results on dexter and gisette. Submission ID 5549 in Appendix A.

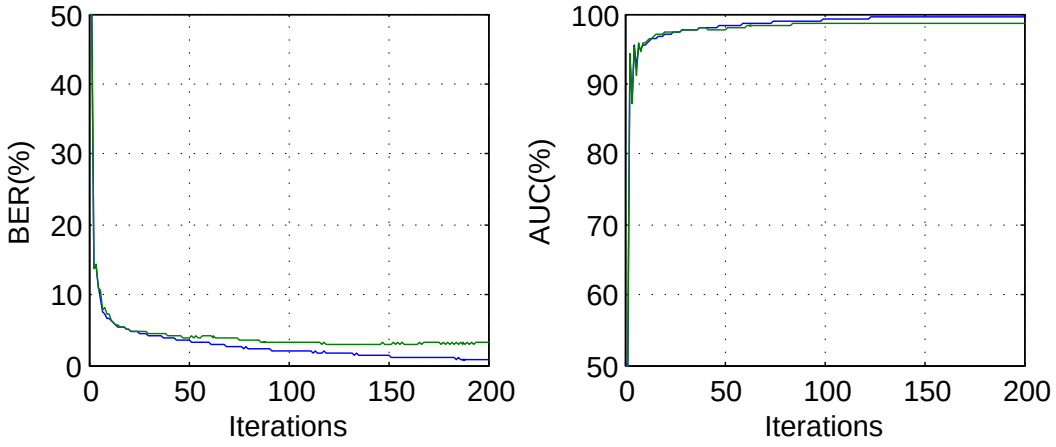


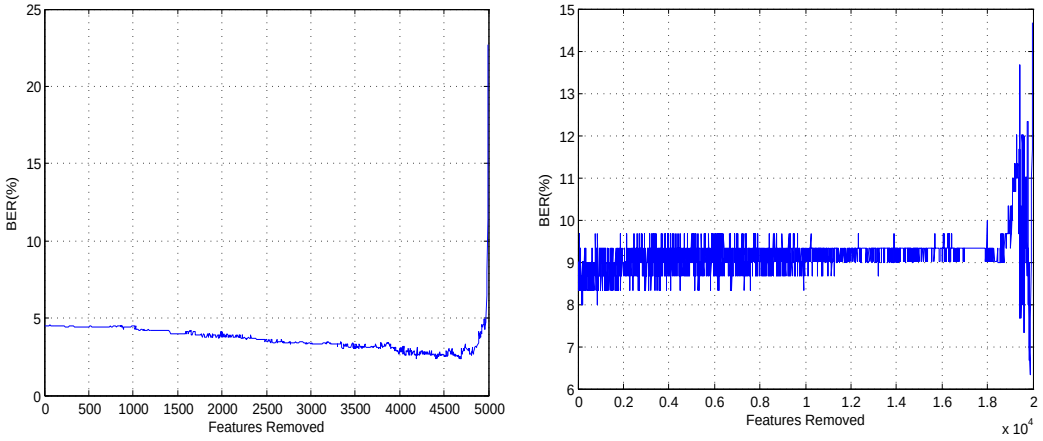
FIGURE 4: MLP training on the gisette dataset.

3 Feature Selection

3.1 Perceptron Weights

When trained the weights of a percptron give an indication of how the each feature contributes to the classifier output. This is done in a iterative process where the five features with the smallest weights are removed and the net is retrained. Figure 5 shows how this process effects the performance of the percptron on each dataset.

Gisette gradually improves in performance before sharply increasing as nearly all the features are removed. Dexter remains stable but becomes erratic after 18000 features have been removed. In table **?** features are selected just before the perforamce decreases sharply attempting to minimise the number of features. No improvement was made even when passed though the MLP but the attempt was to generalise by removing as many features as possible.



(a) Removing features from gisette.

(b) Removing features from dexter.

FIGURE 5: Feature reduction using perceptron weights.

Dataset	**Features**	**Probes (%)**	**Measure**	**Train**	**Valid**	**Test**
Dexter	60	0.3	BER	0.0233	0.0833	0.0825
Dexter	60	0.3	AUC	0.9911	0.9543	0.9646
Gisette	30	0.6	BER	0.0410	0.0520	0.0483
Gisette	30	0.6	AUC	0.9498	0.9393	0.9398

TABLE 3: Feature reduction results on dexter and gisette. Submission ID 5760 in Appendix A.

4 Results

5 Conclusions

References

C.M. Bishop. *Pattern Recognition and Machine Learning*. Information Science and Statistics. Springer, 1 edition, 2006.

Steve Gunn. Feature selection challenge, 2003. [Online].

Adam Prügel-Bennett. *COMP3008: Machine Learning*. University of Southampton, 2001. Lecture Notes.

A Submission ID URLs

- 5518: http://www.nipsfsc.ecs.soton.ac.uk/description/?id=5518
- 5549: http://www.nipsfsc.ecs.soton.ac.uk/description/?id=5549
- 5760: http://www.nipsfsc.ecs.soton.ac.uk/description/?id=5760

B Code Listings

B.1 Perceptron

```
clear
addpath('../Common')
dataset={'dexter'};%,'gisette'};
method=('SLP')
where_my_data_is='../';                                % This is the path to your
    data and results are
data_dir=[where_my_data_is 'Data']                     % Wehre you put the five data
    directories dowloaded.
output_dir=[where_my_data_is 'Results/' method]        % The outputs of a given
    method.
status=mkdir(where_my_data_is, ['Results/' method]);
zip_dir=[where_my_data_is 'Zipped'];                   % Zipped files ready to go!
status=mkdir(where_my_data_is, 'Zipped');
for k=1:length(dataset)
    % Input and output directories
    data_name=dataset{k};
    input_dir=[data_dir '/' upper(data_name)];
    input_name=[input_dir '/' data_name]
    output_name=[output_dir '/' data_name]
    fprintf('\n/|\\-/|\\-/|\\-/|\\ Loading and checking dataset %s
    /|\\-/|\\-/|\\-/|\\\n\n', upper(data_name));
    % Data parameters and statistics
    p=read_parameters([input_name '.param'])
    fprintf('-- %s parameters and statistics -- \n\n', upper(data_name));
    print_parameters(p);
    % Read the data
    fprintf('\n-- %s loading data --\n', upper(data_name));
    X_train=[]; X_valid=[]; X_test=[]; Y_train=[]; Y_valid=[]; Y_test=[];
    load([data_dir '/' data_name]);
    fprintf('\n-- %s data loaded --\n', upper(data_name));
    % Try some method
    fprintf('\n-- Begining... --\n\n');
    idx = feat_select(X_train);
    low_valid = 1;
    for i=1:5
        [c,idx_feat] = train_graph( X_train, Y_train, X_valid, Y_valid,idx);
        % Classifier has been trained, prediction only
        [Y_resu_train, Y_conf_train]    = predict( X_train, c,  idx_feat    );
        [Y_resu_valid, Y_conf_valid]    = predict( X_valid, c,  idx_feat    );
        % Blance error for train and validiate
        errate_train                    = balanced_errate(Y_resu_train, Y_train);
        errate_valid                    = balanced_errate(Y_resu_valid, Y_valid);
        % AUC error for train and validiate
        auc_train                       = auc(Y_resu_train.*Y_conf_train, Y_train);
        auc_valid                       = auc(Y_resu_valid.*Y_conf_valid, Y_valid);
```

```
42          % User output
43          fprintf('%d:    Train: errate= %5.2f%%, auc= %5.2f%%    Validation set: errate
        = %5.2f%%, auc= %5.2f%%   \n',i, errate_train*100, auc_train*100,errate_valid*100,
        auc_valid*100);
44          if(errate_valid < low_valid)
45              low_valid = errate_valid;
46              keep = c;
47          end
48      end
49      % Classifier has been trained, prediction only
50      [Y_resu_train, Y_conf_train]    = predict( X_train, keep,   idx_feat    );
51      [Y_resu_valid, Y_conf_valid]    = predict( X_valid, keep,   idx_feat    );
52      [Y_resu_test, Y_conf_test]      = predict( X_test,  keep,   idx_feat    );
53      % Blance error for train and validiate
54      errate_train                    = balanced_errate(Y_resu_train, Y_train);
55      errate_valid                    = balanced_errate(Y_resu_valid, Y_valid);
56      % AUC error for train and validiate
57      auc_train                       = auc(Y_resu_train.*Y_conf_train, Y_train);
58      auc_valid                       = auc(Y_resu_valid.*Y_conf_valid, Y_valid);
59      % User output
60      fprintf('Training set: errate= %5.2f%%, auc= %5.2f%%\n', errate_train*100,
        auc_train*100);
61      fprintf('Validation set: errate= %5.2f%%, auc= %5.2f%%\n', errate_valid*100,
        auc_valid*100);
62      % Write out the results
63      save_outputs([output_name '_train.resu'], Y_resu_train);
64      save_outputs([output_name '_valid.resu'], Y_resu_valid);
65      save_outputs([output_name '_test.resu'], Y_resu_test);
66      save_outputs([output_name '_train.conf'], Y_conf_train);
67      save_outputs([output_name '_valid.conf'], Y_conf_valid);
68      save_outputs([output_name '_test.conf'], Y_conf_test);
69      save_outputs([output_name '.feat'], idx_feat);
70      fprintf('\n-- %s results saved, see %s* --\n', upper(data_name), output_name);
71  end % Loop over datasets
72  % Zip the archive so it is ready to go!
73  zip([zip_dir '/' method], ['Results/' method], where_my_data_is);
74  fprintf('\n-- %s zip archive created, see %s.zip --\n', upper(data_name), [zip_dir '/'
        method]);
```

LISTING 1: run.m

```
1   function [param,idx_feat]=train(X_train, Y_train, X_valid, Y_valid, feat)
2       eta=1e-4;
3       X=X_train(:,feat);                      % All features
4       patterns=size(X,1);
5       ones_row=ones(patterns,1);
6       X=horzcat(X,ones_row);                  % Offset on all input patterns
7       features=size(X,2);
8       W=rand(features,1)-0.5;                 % Init weights to zero
9       for t=1:30
10          x(t) = t;
11          Y=tnah(X*W)';                   % Do
12          deltaW = (Y_train - Y)';
13          W = W + eta.*(deltaW*X)';           % Train
14      end
15      param.W = W(1:(end-1));                 % Split
16      param.b = W(end);
17      idx_feat = feat;
```

LISTING 2: train.m

```
1   function [param,idx_feat]=train(X_train, Y_train, X_valid, Y_valid, feat)
```

```
    eta=1e-2;
    X=X_train(:,feat);                          % All features
    patterns=size(X,1);
    ones_row=ones(patterns,1);
    X=horzcat(X,ones_row);                      % Offset on all input patterns
    features=size(X,2);
    W=zeros(features,1)-0.5;                        % Init weights to zero
    for t=1:20
        x(t) = t;
        Y=tanh(X*W);                        % Do
        deltaW = (Y_train - Y)';
        W = W + eta.*(deltaW*X)';               % Train
        param.W = W(1:(end-1));
        param.b = W(end);
        % Predict
        [Y_resu_train, Y_conf_train]    = predict( X_train, param, feat);
        [Y_resu_valid, Y_conf_valid]    = predict( X_valid, param, feat);
        % Blance error for train and validiate
        err_bal_train(t)                = balanced_errate(Y_resu_train, Y_train)*100;
        err_bal_valid(t)                = balanced_errate(Y_resu_valid, Y_valid)*100;
        % AUC error for train and validiate
        err_auc_train(t)                = auc(Y_resu_train.*Y_conf_train, Y_train)
    *100;
        err_auc_valid(t)                = auc(Y_resu_valid.*Y_conf_valid, Y_valid)
    *100;
    end
    figure;
    title('Perceptron Training')
    subplot(1,2,1)
    plot(x,err_bal_train,x,err_bal_valid)
    ylabel('BER(%)')
    xlabel('Iterations')
    grid on;
    subplot(1,2,2)
    plot(x,err_auc_train,x,err_auc_valid)
    ylabel('AUC(%)')
    xlabel('Iterations')
    grid on ;
    idx_feat = feat;
```

Listing 3: trainGraph.m

```
function [Y_resu, Y_conf] = predict(X_test, param, idx_feat)
    Y_score=X_test(:,idx_feat)*param.W+param.b;
    Y_resu=unitVec(Y_score)';
    Y_conf=abs(Y_score);
```

Listing 4: predict.m

```
function idx=feat_select(X)
    for i=1:size(X,2)
        idx(i) = i;
    end
```

Listing 5: featSelect.m

B.2 Multi Layer Perceptron

```
clear
addpath('../Common')
```

```
dataset={'dexter'};
method=('MLP')
where_my_data_is='../';                                % This is the path to your
    data and results are
data_dir=[where_my_data_is 'Data']                     % Wehre you put the five data
    directories dowloaded.
output_dir=[where_my_data_is 'Results/' method]        % The outputs of a given
    method.
status=mkdir(where_my_data_is, ['Results/' method]);
zip_dir=[where_my_data_is 'Zipped'];                   % Zipped files ready to go!
status=mkdir(where_my_data_is, 'Zipped');


for k=1:length(dataset)

    % Input and output directories
    data_name=dataset{k};
    input_dir=[data_dir '/' upper(data_name)];
    input_name=[input_dir '/' data_name]
    output_name=[output_dir '/' data_name]
    fprintf('\n/|\\-/|\\-/|\\-/|\\ Loading and checking dataset %s
    /|\\-/|\\-/|\\-/|\\\n\n', upper(data_name));

    % Data parameters and statistics
    p=read_parameters([input_name '.param'])
    fprintf('-- %s parameters and statistics -- \n\n', upper(data_name));
    print_parameters(p);

    % Read the data
    fprintf('\n-- %s loading data --\n', upper(data_name));
    X_train=[]; X_valid=[]; X_test=[]; Y_train=[]; Y_valid=[]; Y_test=[];
    load([data_dir '/' data_name]);
    fprintf('\n-- %s data loaded --\n', upper(data_name));

    % Try some method
    fprintf('\n-- Begining... --\n\n');
    idx = feat_select(X_train,Y_train,200);

    [params,idx_feat] = train( X_train, Y_train, X_valid, Y_valid, idx);


    % Classifier has been trained, prediction only
    [Y_resu_train, Y_conf_train]    = predict( X_train, params, idx );
    [Y_resu_valid, Y_conf_valid]    = predict( X_valid, params, idx );
    [Y_resu_test, Y_conf_test]      = predict( X_test,  params, idx_feat    );


    % Blance error for train and validiate
    errate_train                    = balanced_errate(Y_resu_train, Y_train);
    errate_valid                    = balanced_errate(Y_resu_valid, Y_valid);

    % AUC error for train and validiate
    auc_train                       = auc(Y_resu_train.*Y_conf_train, Y_train);
    auc_valid                       = auc(Y_resu_valid.*Y_conf_valid, Y_valid);

    % User output
    fprintf('Training set: errate= %5.2f%%, auc= %5.2f%%\n', errate_train*100,
    auc_train*100);
    fprintf('Validation set: errate= %5.2f%%, auc= %5.2f%%\n', errate_valid*100,
    auc_valid*100);


```

```
59
60      % Write out the results
61      % ---- Note: the class predictions (.resu files) are mandatory.
62      % ---- Please also provide the confidence value when available, this will
63      % ---- allow us to compute ROC curves. A confidence values can be the absolute
64      % ---- values of a discriminant value, it does not need to be normalized
65      % ---- to resemble a probability.
66      save_outputs([output_name '_train.resu'], Y_resu_train);
67      save_outputs([output_name '_valid.resu'], Y_resu_valid);
68      save_outputs([output_name '_test.resu'], Y_resu_test);
69      save_outputs([output_name '_train.conf'], Y_conf_train);
70      save_outputs([output_name '_valid.conf'], Y_conf_valid);
71      save_outputs([output_name '_test.conf'], Y_conf_test);
72      save_outputs([output_name '.feat'], idx_feat);
73      fprintf('\n-- %s results saved, see %s* --\n', upper(data_name), output_name);
74
75  end % Loop over datasets
76
77  % Zip the archive so it is ready to go!
78  zip([zip_dir '/' method], ['Results/' method], where_my_data_is);
79  fprintf('\n-- %s zip archive created, see %s.zip --\n', upper(data_name), [zip_dir '/'
        method]);
```

LISTING 6: run.m

```
1   function [best,feat_out]=train(X_train, Y_train, X_valid, Y_valid, feat_in)
2       eta=1e-4;
3       X=X_train(:,feat_in);                          % Subset of features
4       X_val=X_valid(:,feat_in);
5       patterns=size(X,1);
6       ones_row=ones(patterns,1);
7       X=horzcat(X,ones_row);                    % Offset on all input patternus
8       features=size(X,2);
9       hiddens=10;
10      w_in=rand(hiddens,features)-0.5;          % Weights from input to each hidden layer
        node
11      w_out=rand(1,hiddens)-0.5;              % Weights from hidden layer to output
12      b_out = 0;
13      low_valid = 100;
14      for i=1:300
15          disp(i)
16          % Forward
17          for j=1:hiddens
18              Y_hidden(j,:) = tanh(X*w_in(j,:)');
19          end
20          Y = tanh(Y_hidden'*w_out')';
21          % BackProp
22          E = (Y_train' - Y);
23          delta_in = eta.*E*X;
24          for j=1:hiddens
25              w_in(j,:) = w_in(j,:) + w_out(j)*delta_in;
26          end
27          delta_out = eta.*E*Y_hidden';
28          w_out = w_out + delta_out;
29          b_out = b_out - eta*sum(E);
30          % performace
31          x(i) = i;
32          param.w_in = w_in(:,(1:(end-1)));
33          param.b_in = w_in(:,end);
34          param.w_out = w_out;
35          param.b_out = b_out;
36          [Y_resu_train, Y_conf_train]    = predict( X_train, param,feat_in);
```

```
37          [Y_resu_valid, Y_conf_valid]     = predict( X_valid, param,feat_in);
38          err_train(i)                     = balanced_errate(Y_resu_train, Y_train)*100;
39          err_valid(i)                     = balanced_errate(Y_resu_valid, Y_valid)*100;
40          auc_train(i)                     = auc(Y_resu_train.*Y_conf_train, Y_train)
        *100;
41          auc_valid(i)                     = auc(Y_resu_valid.*Y_conf_valid, Y_valid)
        *100;
42          if(err_valid(i) < low_valid)
43              low_valid = err_valid(i);
44
45              best.w_in = w_in(:,(1:(end-1)));
46              best.b_in = w_in(:,end);
47              best.w_out = w_out;
48              best.b_out = b_out;
49          end
50      end
51      figure;
52      subplot(1,2,1);
53      plot(x,err_train,x,err_valid);
54      xlabel('Iterations')
55      ylabel('BER(%)')
56      grid on;
57      subplot(1,2,2);
58      plot(x,auc_train,x,auc_valid);
59      xlabel('Iterations')
60      ylabel('AUC(%)')
61      grid on;
62      feat_out = feat_in;
```

LISTING 7: train.m

```
1  function [Y_resu, Y_conf] = predict(X_test, param, feat)
2      X=X_test(:,feat);
3      hiddens=size(param.w_in,1);
4
5      for j=1:hiddens
6          Y_hidden(j,:) = tanh(X*param.w_in(j,:)'+param.b_in(j));
7      end
8      Y_score =(Y_hidden'*param.w_out')+param.b_out;
9      Y_resu=unitVec(Y_score)';
10     Y_conf=abs(Y_score);
```

LISTING 8: predict.m

```
1  function idx=feat_select(X)
2      for i=1:size(X,2)
3          idx(i) = i;
4      end
```

LISTING 9: featSelect.m