

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 MATLAB 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

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). Everything implemented has been developed from scratch for the MATLAB programming language.

$$y, t \in \{-1, +1\} \quad (1)$$

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

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 , has 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 1 and 2 respectively. All code for the perceptron is held in appendix B.1

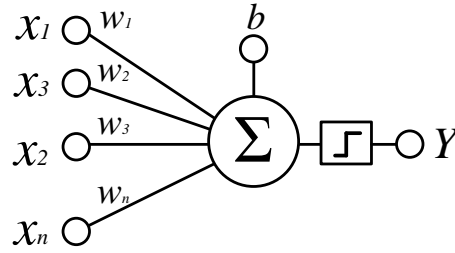


FIGURE 1: Perceptron architecture.

$$\mathbf{Y} = \Theta(\mathbf{X} \cdot \mathbf{W}) \quad (3)$$

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

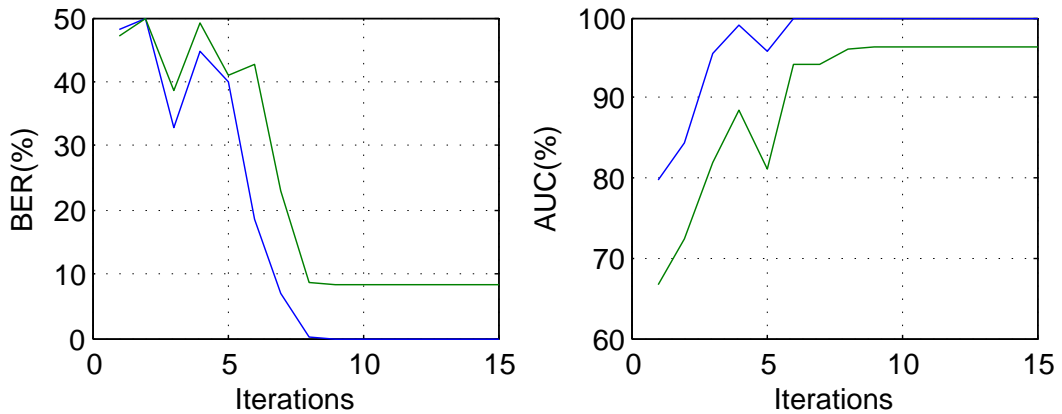


FIGURE 2: Perceptron training on the dexter dataset.

	Train	Valid	Test
BER	0.0000	0.0633	0.0735
AUC	1.0000	0.9779	0.9767

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

	Train	Valid	Test
BER	0.0445	0.0430	0.0432
AUC	0.9889	0.9894	0.9907

TABLE 2: Perceptron results on 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 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 3 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

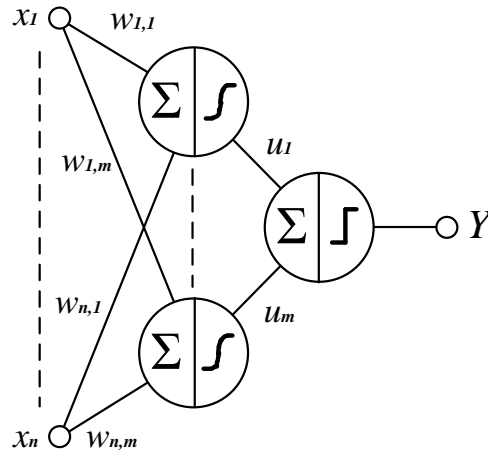


FIGURE 3: MLP architecture.

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

	Train	Valid	Test
BER	0.0120	0.0280	0.0312
AUC	0.9955	0.9864	0.9839

TABLE 3: MLP results on gisette. Submission ID 5549 in Appendix A.

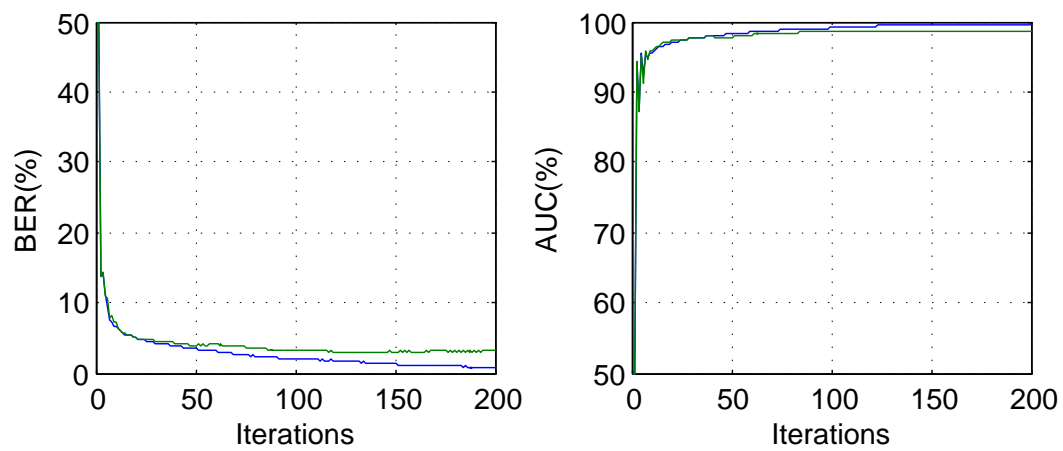


FIGURE 4: MLP training on the gisette dataset.

3 Feature Selection

4 Results

5 Conclusions

References

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

B Code Listings

B.1 Perceptron

```

1 clear
2 addpath( '../Common' )
3 dataset={ 'dexter' }; %,'gisette' };
4 method=('SLP')
5 where_my_data_is='../'; % This is the path to your
    data and results are
6 data_dir=[where_my_data_is 'Data'] % Where you put the five data
    directories downloaded.
7 output_dir=[where_my_data_is 'Results/' method] % The outputs of a given
    method.
8 status=mkdir( where_my_data_is , [ 'Results/' method] );
9 zip_dir=[where_my_data_is 'Zipped']; % Zipped files ready to go!
10 status=mkdir( where_my_data_is , 'Zipped' );
11 for k=1:length( dataset )
12     % Input and output directories
13     data_name=dataset{ k };
14     input_dir=[data_dir '/' upper( data_name )];
15     input_name=[input_dir '/' data_name]
16     output_name=[output_dir '/' data_name]
17     fprintf( '\n\\-\\-\\-\\-\\- Loading and checking dataset %s
    \\-\\-\\-\\-\\-\\n', upper( data_name ) );
18     % Data parameters and statistics
19     p=read_parameters( [input_name '.param'] )
20     fprintf( '-- %s parameters and statistics -- \n\n', upper( data_name ) );
21     print_parameters( p );
22     % Read the data
23     fprintf( '\n-- %s loading data --\n', upper( data_name ) );
24     X_train=[]; X_valid=[]; X_test=[]; Y_train=[]; Y_valid=[]; Y_test=[];
25     load( [data_dir '/' data_name] );
26     fprintf( '\n-- %s data loaded --\n', upper( data_name ) );
27     % Try some method
28     fprintf( '\n-- Begining ... --\n\n' );
29     idx = feat_select( X_train );
30     low_valid = 1;
31     for i=1:5
32         [c,idx_feat] = train_graph( X_train, Y_train, X_valid, Y_valid,idx );
33         % Classifier has been trained, prediction only
34         [Y_resu_train, Y_conf_train] = predict( X_train, c, idx_feat );
35         [Y_resu_valid, Y_conf_valid] = predict( X_valid, c, idx_feat );
36         % Blance error for train and validate
37         errate_train = balanced_errate( Y_resu_train, Y_train );
38         errate_valid = balanced_errate( Y_resu_valid, Y_valid );
39         % AUC error for train and validate
40         auc_train = auc( Y_resu_train.*Y_conf_train, Y_train );
41         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 % Balance error for train and validate
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 validate
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=1:10
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)
2     eta=1e-5;

```

```

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=1:200
10         x(t) = t;
11         Y=tanh(X*W);                                    % Do
12         deltaW = (Y_train - Y)';
13         W = W + eta.*(deltaW*X)';                       % Train
14         param.W = W(1:(end-1));
15         param.b = W(end);
16         % Predict
17         [Y_resu_train, Y_conf_train] = predict(X_train, param, feat);
18         [Y_resu_valid, Y_conf_valid] = predict(X_valid, param, feat);
19         % Blance error for train and validate
20         err_bal_train(t) = balanced_errate(Y_resu_train, Y_train)*100;
21         err_bal_valid(t) = balanced_errate(Y_resu_valid, Y_valid)*100;
22         % AUC error for train and validate
23         err_auc_train(t) = auc(Y_resu_train.*Y_conf_train, Y_train)
24         *100;
25         err_auc_valid(t) = auc(Y_resu_valid.*Y_conf_valid, Y_valid)
26         *100;
27     end
28     figure;
29     title('Perceptron Training')
30     subplot(1,2,1)
31     plot(x,err_bal_train,x,err_bal_valid)
32     ylabel('BER(%)')
33     xlabel('Iterations')
34     grid on;
35     subplot(1,2,2)
36     plot(x,err_auc_train,x,err_auc_valid)
37     ylabel('AUC(%)')
38     xlabel('Iterations')
39     grid on ;
40     idx_feat = feat;

```

LISTING 3: trainGraph.m

```

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

```

LISTING 4: predict.m

```

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

```

LISTING 5: featSelect.m

B.2 Multi Layer Perceptron

```

1 clear
2 addpath(' ../Common')
3 dataset={'dexter'};

```



```

4 method=('MLP')
5 where_my_data_is='../'; % This is the path to your
    data and results are
6 data_dir=[where_my_data_is 'Data'] % Where you put the five data
    directories downloaded.
7 output_dir=[where_my_data_is 'Results/' method] % The outputs of a given
    method.
8 status=mkdir(where_my_data_is, ['Results/' method]);
9 zip_dir=[where_my_data_is 'Zipped']; % Zipped files ready to go!
10 status=mkdir(where_my_data_is, 'Zipped');
11
12
13 for k=1:length(dataset)
14
15     % Input and output directories
16     data_name=dataset{k};
17     input_dir=[data_dir '/' upper(data_name)];
18     input_name=[input_dir '/' data_name]
19     output_name=[output_dir '/' data_name]
20     fprintf('\n\\-\\-\\-\\-\\-\\-\\-\\- Loading and checking dataset %s
    \\-\\-\\-\\-\\-\\-\\-\\-\\n\\n', upper(data_name));
21
22     % Data parameters and statistics
23     p=read_parameters([input_name '.param'])
24     fprintf('-- %s parameters and statistics -- \\n\\n', upper(data_name));
25     print_parameters(p);
26
27     % Read the data
28     fprintf('\n-- %s loading data --\\n', upper(data_name));
29     X_train=[]; X_valid=[]; X_test=[]; Y_train=[]; Y_valid=[]; Y_test=[];
30     load([data_dir '/' data_name]);
31     fprintf('\n-- %s data loaded --\\n', upper(data_name));
32
33     % Try some method
34     fprintf('\n-- Begining... --\\n\\n');
35     idx = feat_select(X_train,Y_train,200);
36
37     [params,idx_feat] = train( X_train, Y_train, X_valid, Y_valid, idx);
38
39
40     % Classifier has been trained, prediction only
41     [Y_resu_train, Y_conf_train] = predict( X_train, params, idx );
42     [Y_resu_valid, Y_conf_valid] = predict( X_valid, params, idx );
43     [Y_resu_test, Y_conf_test] = predict( X_test, params, idx_feat );
44
45
46     % Blance error for train and validate
47     errate_train = balanced_errate(Y_resu_train, Y_train);
48     errate_valid = balanced_errate(Y_resu_valid, Y_valid);
49
50     % AUC error for train and validate
51     auc_train = auc(Y_resu_train.*Y_conf_train, Y_train);
52     auc_valid = auc(Y_resu_valid.*Y_conf_valid, Y_valid);
53
54     % User output
55     fprintf('Training set: errate= %5.2f%%, auc= %5.2f%%\\n', errate_train*100,
    auc_train*100);
56     fprintf('Validation set: errate= %5.2f%%, auc= %5.2f%%\\n', errate_valid*100,
    auc_valid*100);
57
58
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 patterns
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