COMP6036: Advanced Machine Learning
Feature Selection Challenge

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

School of Electronics and Computer Science
University of Southampton

29th 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

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.

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

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

2 Classifier Architectures

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

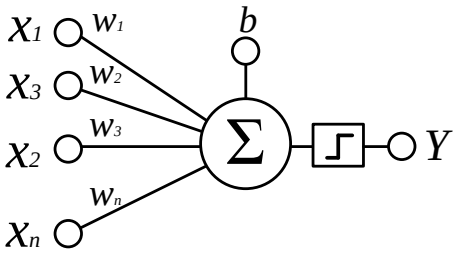

FIGURE 1: Perceptron architecture.

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

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

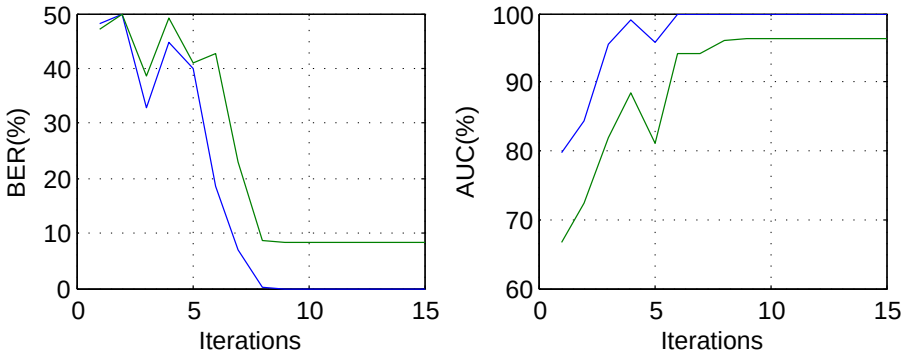

FIGURE 2: Perceptron training on the dexter dataset.

2.2 Multi Layer Perceptron

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

	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.

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.

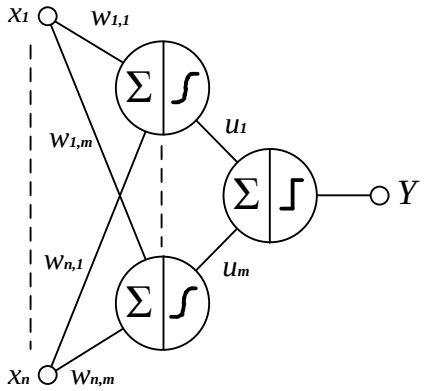

FIGURE 3: Multi layer perceptron architecture.

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

2.3 Testing

The classifier was tested upon the the *Dexter* dataset using all of the features avaialble in the input patterns. The training iterations are shown in shonw in figure 4 and can be matched to the results held in table 3. Manual tuning was used set the number of hidden nodes to 100 and η to a value of 5×10^{-5}. This performs as expted upon the test data which is only slighly worse than the validation set performance.

	Train	Valid	Test
BER	0.0033	0.0833	0.0890
AUC	1.0000	0.9761	0.9739

TABLE 3: MLP results on Dexter. Submission ID 5459 A.

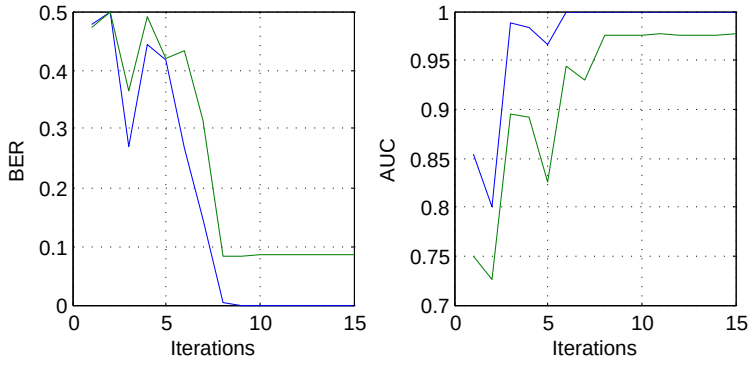

FIGURE 4: Training iterations of the MLP.

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

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);
        % User output
        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);
        if(errate_valid < low_valid)
```

```
            low_valid = errate_valid;
            keep = c;
        end
    end
    % Classifier has been trained, prediction only
    [Y_resu_train, Y_conf_train]    = predict( X_train, keep,   idx_feat    );
    [Y_resu_valid, Y_conf_valid]    = predict( X_valid, keep,   idx_feat    );
    [Y_resu_test, Y_conf_test]      = predict( X_test,  keep,   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);
    % Write out the results
    save_outputs([output_name '_train.resu'], Y_resu_train);
    save_outputs([output_name '_valid.resu'], Y_resu_valid);
    save_outputs([output_name '_test.resu'], Y_resu_test);
    save_outputs([output_name '_train.conf'], Y_conf_train);
    save_outputs([output_name '_valid.conf'], Y_conf_valid);
    save_outputs([output_name '_test.conf'], Y_conf_test);
    save_outputs([output_name '.feat'], idx_feat);
    fprintf('\n-- %s results saved, see %s* --\n', upper(data_name), output_name);
end % Loop over datasets
% Zip the archive so it is ready to go!
zip([zip_dir '/' method], ['Results/' method], where_my_data_is);
fprintf('\n-- %s zip archive created, see %s.zip --\n', upper(data_name), [zip_dir '/'
    method]);
```

LISTING 1: run.m

```
function [param,idx_feat]=train(X_train, Y_train, X_valid, Y_valid, feat)
    eta=1e-4;
    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=rand(features,1)-0.5;                 % Init weights to zero
    for t=1:10
        x(t) = t;
        Y=unitVec(X*W)';                    % Do
        deltaW = (Y_train - Y)';
        W = W + eta.*(deltaW*X)';           % Train
    end
    param.W = W(1:(end-1));                 % Split
    param.b = W(end);
    idx_feat = feat;
```

LISTING 2: train.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 3: predict.m

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

LISTING 4: featSelect.m