(Concepts of) Machine Learning

Neural networks for classification

After this session you should be able to:

- Create multilayer neural networks in Matlab;
- Train and test multilayer neural networks.

Classification

Classification problems form an important application area of machine intelligence.

Assume that you have a special discrete-valued variable called the **class**; this is denoted by C. The variable C takes values in $\{c_1, c_2, \ldots c_m\}$. Let's now assume that m=8; then we have 8 classes: c_1=1, c_2=2,.., c_8=8.

The classification problem aims to specify what value the class variable C has for a given object, taking into account a set of measurements on the object, e.g., A, B, …etc. These measurements are called *features*. Thus in classification we want to learn a mapping from *features* to *Classes.*

Notation

C is the class

A, B, etc (the measurements) are called the features

Classifiers

In order to solve this problem we need a machine, called **classifier**. A classifier is a mapping from a feature space (a d-dimensional vector) to a set of class labels {1,2,…,m}. Thus, a classifier partitions a feature space into m decision regions. The line or surface separating any 2 classes is the decision boundary.

Applications of Classification

Medical Diagnosis: classification of cancerous cells.

Credit card and Loan approval: Most major banks.

Speech recognition: IBM, Dragon Systems, AT&T, Microsoft, etc.

Optical Character/Handwriting Recognition: Post Offices, Banks, Gateway, Motorola, Microsoft, Xerox, etc.
Email classification: classify email as "junk" or "non-junk".

Classification Accuracy (success)
Assume we have N feature vectors. Assume that you know the true class label for each feature vector. We can measure how accurate a classifier is by counting how many feature vectors have been classified correctly.

Accuracy = percentage of feature vectors correctly classified.
Training accuracy = accuracy on training data.
Test accuracy = accuracy on new data not used in training.

Training Data and Test Data
Training data: Labelled data used to build a classifier.
Test data: These are new data, not previously seen during training, which are used to estimate how well the classifier works.

Memorisation versus Generalisation
Better training accuracy leads in memorising the training data.
Better testing accuracy results in *good* generalisation over new data.
In general, we would like our classifier to perform well on new test data, not just on training data, i.e. we would like it to generalise well to new data.

Training and Testing Data
Training Data:
`Dtrain = { [x(1), c(1)] , [x(2), c(2)] , …, [x(N), c(N)] }`
`N`: pairs of feature vectors and class labels

Feature Vectors and Class Labels:
`x(i)` is the *i*-th training data feature vector.

In MATLAB this could be the *i*-th row of an `N×d` matrix. `c(i)` is the class label of the *i*-th feature vector. In general, `c(i)` can take *m* different class values, e.g., c = 1, c = 2, ...

Examples of Training and Test Data
Speech Recognition
Training data: words recorded and labelled in a laboratory
Test data: words recorded from new speakers, noisy conditions

Prediction of proteins localisation
Training data: Proteins localised in 8 different localisation sites.
Test data: *Ecoli* Proteins, previously unknown, should be classified in one of these sites.

Classification with a Feedforward Network

Let us attempt to build a classifier that can identify the gender of a crab from its physical measurements. Six physical characteristics of a crab are considered: species, frontallip, rearwidth, length, width and depth. The problem on hand is to identify the gender of a crab given the observed values for each of these 6 physical characteristics.

Step1: Preparing (or loading) the Data

```
[x,t] = crab_dataset;
```

% where `x` → input matrix of size (6,200), each row corresponding to each characteristic taken into account & `t` → target matrix of size (2,200) wherein Female crabs are represented with a one in the first element, male crabs with a one in the second element. (All other elements are zero) and 200 → no. of crab samples in the dataset

Step2: Building the classifier

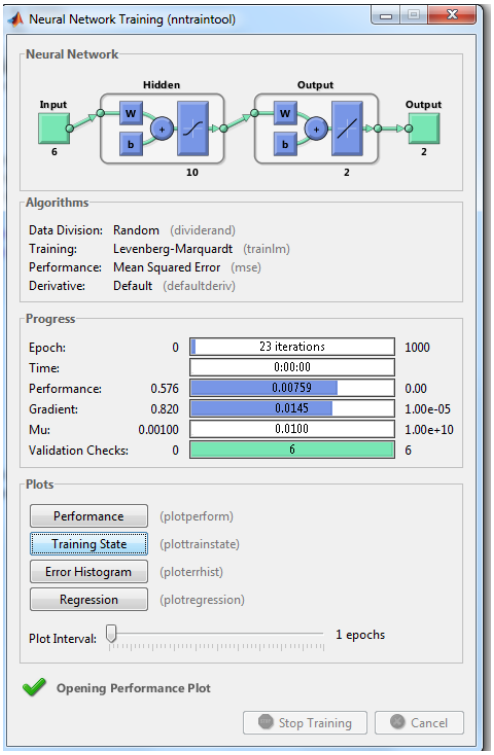
```
net = feedforwardnet(10);
```

% creates a network with single hidden layer with 10 neurons

Step3: Training the network

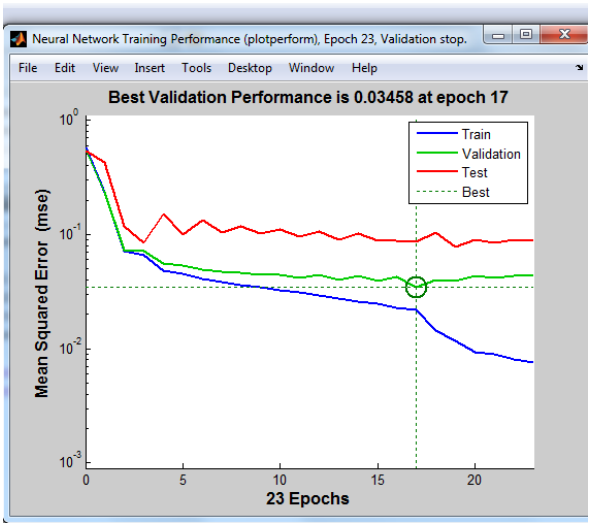
```
net.trainParam.epochs = 100;    % max. epochs/iterations
net.trainparam.lr = 0.3;        % learning rate
net.trainParam.mc = 0.6;        % momentum constant
[net,tr] = train(net,x,t);      % network training
```

% neural network training tool will appear on screen now, it looks like this:



Step4: plot performance

% when the training stops click on Performance button, you will be able to see performance plot on training, validation and testing, it should look like this:



Step5: test the classifier

% retrieve the patterns that were sampled as testing patterns

```
testX = x(:,tr.testInd);
testT = t(:,tr.testInd);
```

% test classifier on the new data

```
testY = net(testX);
```

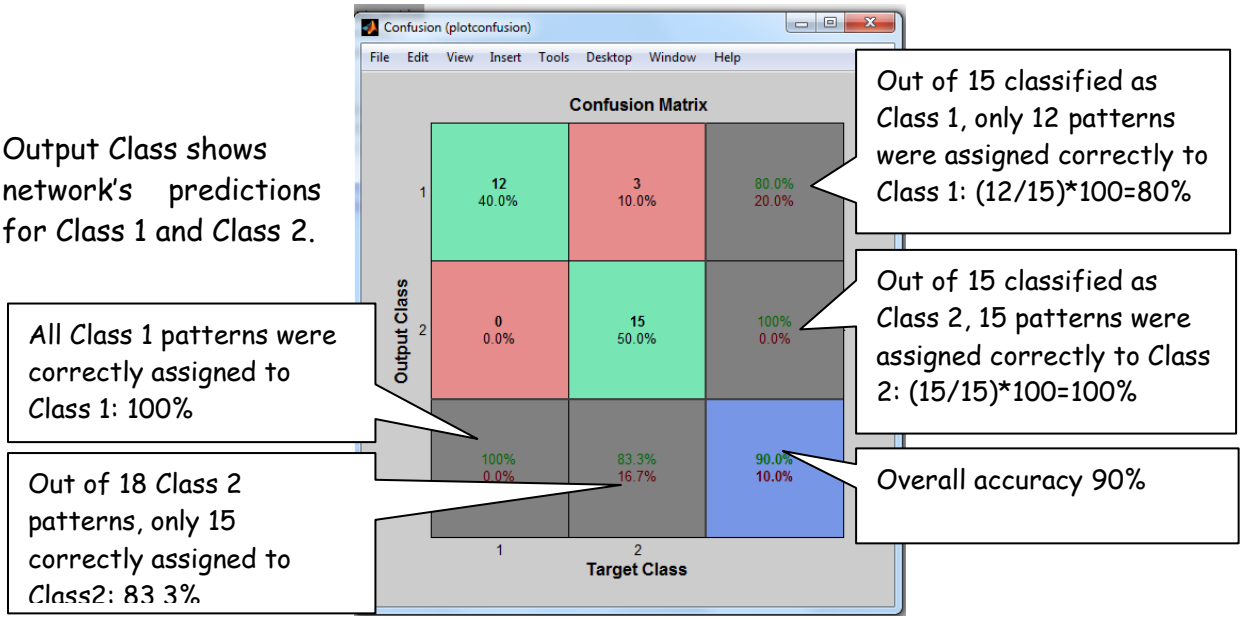
```
testIndices = vec2ind(testY)
```

Step6: plot test performance

One measure of how well the neural network has fit the data is the confusion plot. The confusion matrix shows the percentages of correct and incorrect classifications. Correct classifications are the green squares on the matrices diagonal. Incorrect classifications form the red squares. If the network has learned to classify properly, the percentages in the red squares should be very small, indicating few misclassifications.

```
plotconfusion(testT,testY)
```

% you should get a plot like this. Rows represent predicted classes, whilst columns represent true/actual classes. Diagonal cells show correct classifications. Off-diagonal cells show classifier mistakes.



Target shows the "true" classes

To view overall percentages of correctly classified or misclassified, use the following command

```
[c,cm] = confusion(testT,testY)
fprintf('Percentage Correct Classification   : %f%%\n', 100*(1-c));
fprintf('Percentage Incorrect Classification : %f%%\n', 100*c);
```

% wherein `c` → confusion values, i.e. fraction of samples misclassified and `cm` → S-by-S confusion matrix, where `cm(i,j)` is the number of samples whose target is the `i`th class but was classified as `j`

Upon typing the `fprintf` commands, what percentages did you get for correctly classified/misclassified results? Try improving performance by increasing epochs.

Are you happy with the performance? Does the desired target look close to the actual values?

Multilayer Neural Networks

This category of networks can be effectively put to use for many types of problems like function fitting, pattern recognition etc. Multilayer feedforward networks have one or more hidden layers of neurons followed by an output layer of linear or sigmoid neurons. Multiple layers of neurons with nonlinear transfer functions allow the network to learn nonlinear relationships between input and output vectors. An example problem is described below:

Classification of Wine

The aim is to build a classifier that can that can classify wines from three wineries using thirteen different attributes, such as alcohol, colour intensity, hue etc. This is an example of a pattern recognition problem, where inputs are associated with different classes, and our aim is to create a neural network that not only classifies the known wines properly, but can generalize to accurately classify wines that were not used to design the solution.

- ***Preparing the Data***
  For classification, data for training a neural network are organised into two matrices, the input matrix **X** and the target matrix **T**. Each ith column of the input matrix has thirteen elements representing a wine whose winery is already known. Each corresponding column of the target matrix has three elements, consisting of two zeros and a 1 in the location of the associated winery. A dataset is already available and it can be loaded using the command:

  1. `[x,t] = wine_dataset;`
  % This commands loads the wine dataset. The matrix **x** contains the inputs and the matrix **t** consists of the target elements. Note that both **x** and **t** have 178 columns. These represent 178 wine sample attributes (inputs) and associated winery class vectors (targets).

Input matrix **x** has thirteen rows, for the thirteen attributes. Target matrix **t** has three rows, as for each example we have three possible wineries.

- ***Building the Neural Network Classifier***
  The next step is to create a neural network that will learn to classify the wines. For this task it is enough to use a one-hidden-layer feed forward neural networks.

  2. `net = patternnet(10);`
  % Creates a feedforward network with a single hidden layer with 10 neurons. You may also use the command **feedforwardnet** for building a network.

- ***Training the Classifier***
  The created network has to be trained. The samples are automatically divided into training, validation and test sets. The training set is used to train the network. Training continues as long as the network continues improving on the validation set. The test set provides a completely independent measure of network accuracy.

  3. `[net,tr] = train(net,x,t);`
  % The network is trained in a supervised mode on inputs **x** based on the desired targets **t**. To see how the network's performance improved during training, either click the **Performance** button in the training tool, or call **plotperform**. Performance is measured in terms of mean squared error, and shown in log scale.

- ***Testing the Classifier***
  The trained neural network can now be tested with the testing samples. This gives a sense of how well the network will do when applied to data from the real world. The network outputs are in the range 0 to 1, so we use **vec2ind** function to get the class indices as the position of the highest element in each output vector.

  4. `testX = x(:,tr.testInd);`
  `testT = t(:,tr.testInd);`
  % Samples out 27 samples from the input and target dataset for testing purposes.

  5. `testY = net(testX);`
  % Computes the test output

6. `testIndices = vec2ind(testY)`
% Convert vectors to indices; that is, the function is applied to get the class indices as the position of the highest element in each output vector.

- ***Performance Assessment***
One method of assessing the performance is to plot the so-called *confusion matrix*. The confusion matrix shows the percentages of correct and incorrect classifications.

7. `plotconfusion(testT,testY)`
% This command plots the confusion matrix. Note that this plot is also generated automatically in `nntraintool` when you use the `patternnet` command. If you use the `feedforwardnet` command (see Step 2, above) then you'll have to separately plot this matrix using the confusion matrix plot command. Correct classifications are shown as **green squares** on the matrices diagonal. Incorrect classifications form the **red squares**. If the network has learned to classify properly, the percentages in the red squares should be very small, indicating few misclassifications.

8. `[c,cm] = confusion(testT,testY)`
% This command also creates the classification confusion matrix. Here `c` refers to the confusion value, which is equal to the fraction of samples misclassified, and `cm(i,j)` is the number of samples whose target is the *i*-th class but was classified as *j*.

9. `fprintf('Percentage Correct Class:%f%%\n', 100*(1-c));`
`fprintf('Percentage Incorrect Class: %f%%\n', 100*c);`
% The above commands prints the percentages of correct and incorrect classifications on screen.