Artificial Neural Networks – Lab 2
Classical Pattern Recognition

Purpose
To implement (using MATLAB) some traditional classifiers and try these on both synthetic and real data. The exercise should give some insight into the advantages and disadvantages of the different methods.

Presentation
The results from the exercise should be presented individually to one of the lab organizers. If a question or sub-task appears with a box [1] then your respective answer must be presented.

To obtain 1 extra point to be added to the final exam mark, students must be present at the start of the next lab to answer questions on their work. If two students work together in a pair, then both students must be present and able to demonstrate the software that they have written and explain the answers.

Data and Matlab functions
Data and m-files can be downloaded from the course homepage (http://www.aass.oru.se/~tdt/ann). The files you need for this lab are:

boc_decision.m  dis.mat  landsat.mat
plot_dline.m  plot_levels.m  plot_surface.m
sat_features.m  sat_validate.m

Note: Read each task carefully before starting to solve it.

Introduction
Your task is to implement two different classifiers: a minimum distance classifier and a Bayes optimal classifier for normally distributed classes. You will test both classifiers on some artificial data, using some training data to make the classifier and separate test data to estimate the performance of the classifier. Finally, you will apply your pattern recognition skills to a real world problem, by using a Bayes classifier to interpret some satellite images.

The file dis.mat contains the synthetic data. There are three different data sets, which have different distributions. Each data set is 2-dimensional (i.e., each pattern is represented by two features) and contains two classes. For each class there is one matrix with training data and one matrix with test data. See the following table:

<table>
<thead>
<tr>
<th>Set 1</th>
<th>Set 2</th>
<th>Set 3</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>dis1_1</td>
<td>dis2_1</td>
<td>dis3_1</td>
<td>Training class 1</td>
</tr>
<tr>
<td>dis1_2</td>
<td>dis2_2</td>
<td>dis3_2</td>
<td>Training class 2</td>
</tr>
<tr>
<td>dis1_1t</td>
<td>dis2_1t</td>
<td>dis3_1t</td>
<td>Test class 1</td>
</tr>
<tr>
<td>dis1_2t</td>
<td>dis2_2t</td>
<td>dis3_2t</td>
<td>Test class 2</td>
</tr>
</tbody>
</table>
Getting started

The command

```matlab
>>load dis.mat
```
will load in the synthetic data sets. As usual, you can type

```matlab
>>whos
```

...to see which variables have been created. You should notice that each matrix has two rows, corresponding to the two features, and a number of columns, corresponding to the number of patterns.

Plotting the data

To view the data, use the `plot` command, e.g.,

```matlab
>> plot(dis1_1(1,:),dis1_1(2,:),'ro',dis1_2(1,:),dis1_2(2,:),'b+');
```
will plot data set 1, using red circles for the data from class 1 and blue crosses for the data from class 2. The result should look something like figure 1 above. (Use `help` for further information.)

Task 1, The minimum distance classifier

Implement a minimum distance classifier and investigate its performance on the synthetic data.

Write an m-function to train the minimum classifier, by computing the prototype vectors for both classes on the training data. The interface of this function should be

```matlab
function [m1, m2] = mdc_train(tr_class1, tr_class2)
```

You would then train the classifier on data set 1 using

```matlab
[m1, m2] = mdc_train(dis1_1, dis1_2);
```

where the variables `m1` and `m2` are the mean vectors for classes 1 and 2 respectively.

To help you, we have also written an m-function `plot_dline` to plot the decision line for the minimum distance classifier in the same graph as the training data. For example, you can use

```matlab
>>plot_dline(dis1_1, dis1_2);
```
(this calls \texttt{mdc}\_\texttt{train}, so you must write that function first).

Then write an m-function to test the classifier. Recall that the decision function for the minimum distance classifier is given by

\[ d_j(x) = x^T m_j - \frac{1}{2} m_j^T m_j \]

where \( x \) is the pattern vector to be classified, and \( m_j \) is the mean vector of each class \( \omega_j \). It would be a good idea to first write a separate m-function to calculate the value of the decision function for a given class. A suggestion for the interface of this function is

\begin{verbatim}
function result = mdc\_decision(x, m)
\end{verbatim}

where \( x \) is the test pattern vector and \( m \) is the mean vector for that class.

Then you can write the m-function to test the classifier, e.g., using

\begin{verbatim}
function mdc\_test(m1, m2, test\_set)
\end{verbatim}

as the interface. To use this function, you would type

\begin{verbatim}
mdc\_test(m1, m2, dis1\_1t);
mdc\_test(m1, m2, dis1\_2t);
\end{verbatim}

to classify the test patterns from data set 1. In the function \texttt{mdc\_test} you will need a loop to go through the test patterns, calling \texttt{mdc\_decision} to obtain the value of the decision function for each class. You should also write some code to compute the percentage of patterns classified in each class, and display this information at the end of the function.

\textbf{Deliverables}

For each data set (1, 2, 3) you should do the following:

1. Compute the parameters (the prototype vectors) for a minimum distance classifier by using the training data.

   \begin{center}
   \begin{tabular}{ccc}
   Set 1 & Set 2 & Set 3 \\
   \hline
   \text{Class 1} & & \\
   \text{Class 2} & & \\
   \end{tabular}
   \end{center}

2. Plot the training data and the prototype vectors of both classes in the same figure, using different colours or symbols.

3. Calculate the equation of the decision line by hand and compare this with the line produced by \texttt{plot\_dline}.
4. Plot the test data in the same figure as the training data (use different colours or symbols).
   Discuss the possibilities for effective classification based on the form and position of the decision line.

5. Classify both the training vectors and the test vectors using your minimum distance classifier. Calculate the percentage of correctly classified vectors for the training data and test data. Compare with the expected outcome of the classification. What’s the reason for erroneous classifications?

<table>
<thead>
<tr>
<th>Set 1</th>
<th>Set 2</th>
<th>Set 3</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Training data:

Testing data:

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>-------</td>
<td>-------</td>
<td>-------</td>
</tr>
</tbody>
</table>
Task 2, Bayes optimal classifier for normally distributed classes

This task is similar to the last one, except this time we are going to try to improve the performance of the classifier by adding some extra information about the distribution (“shape”) of the data. We actually did this before in Lab 1 with a very simple method: we used histograms to estimate the probability density functions for the two classes. Another approach that we will consider here is to assume some form for the probability functions, e.g., a normal distribution. For 1-dimensional pattern classification, we would therefore need to calculate two parameters to train the classifier: the mean value $m_j$ and the variance $\sigma_j^2$ for each class $j$. For a 2-dimensional problem, as here, the situation is slightly more complicated: for each class, we need to compute the mean vector $\mathbf{m}_j$ (as in task 1 above) and a covariance matrix $\mathbf{C}_j$ which represents the distribution of the data.

Write an m-function to train the Bayes classifier, by computing the mean vector and covariance matrix for each class on the training data. (You could modify your previous functions for the minimum distance classifier.) Use the built-in function $\text{cov}$ to get the covariance. The interface of this function should be

function $[m1, c1, m2, c2] = \text{boc_train}(\text{tr_class1}, \text{tr_class2})$

You would then train the classifier on data set 1 using

$[m1, c1, m2, c2] = \text{boc_train}(\text{dis1}_1, \text{dis1}_2);$

Then write an m-function to test the classifier. Don’t worry too much about the mathematics here – it looks a little complicated, but it’s actually quite easy to use. The main thing is to try to understand how the Bayes classifier represents the distribution of the training data using probability functions. For a Bayes optimal classifier with normally distributed classes, the decision function can be written as follows:

$$d_j(x) = \ln P(\omega_j) - \frac{1}{2} \ln |C_j| - \frac{1}{2} (x - m_j)^T C_j^{-1} (x - m_j)$$

where $\mathbf{m}_j$ is the mean vector and $\mathbf{C}_j$ is the covariance matrix of each class $\omega_j$ (see the handout “Intro- duktion till mönsterigenkänning” for more details). You may be pleased to know that we have written an m-function called $\text{boc_decision}$ to do this for you!

To help you, we have also written an m-function $\text{plot_levels}$ to plot the level curves for the combined decision function $d(x) = d_1(x) - d_2(x)$ (these are the curves or “contours” of equal probability). The value of $d(x)$ would be positive if the test example belongs to class 1, and negative if it belongs to class 2. There is also a similar function $\text{plot_surface}$. These functions both call $\text{boc_train}$, so you must write that function first.

Deliverables

For each data set (1, 2, 3) you should do the following:

1. Compute the parameters (the mean vectors and covariance matrices) for a Bayes optimal classifier by using the training data.

Set 1  Set 2  Set 3

Class 1

Class 2
2. Plot the training data and the mean vectors of both classes in the same figure, using different colours or symbols.

3. Plot the level curves for $d(x) = d_1(x) - d_2(x)$ in the same figure, and indicate the position of the decision curve that separates the two classes (that is the curve where $d(x) = 0$).

4. Plot the test data in the same figure as the training data (use different colours or symbols), Discuss the possibilities for effective classification based on the form and position of the decision curve.

5. Classify both the training vectors and the test vectors using your Bayes optimal classifier. Calculate the percentage of correctly classified vectors for the training data and test data. Compare with the expected outcome of the classification. What’s the reason for erroneous classifications?

<table>
<thead>
<tr>
<th>Set 1</th>
<th>Set 2</th>
<th>Set 3</th>
</tr>
</thead>
</table>

Training data:

<p>| | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
</table>

Testing data:

<p>| | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
</table>

6. Compare the results for the Bayes classifier with those of the minimum distance classifier. Your conclusion?
Task 3, Vegetation classification in satellite images

In this task you will make use of your skills in pattern recognition to automate the surveying of forest areas via satellite. The Landsat satellites were originally launched by NASA for studies of natural environments. The Landsat 4 and 5 orbit the earth at an altitude of 705 km with an orbital time of 98.5 minutes. Among the sensors of the Landsats there are a multi-spectral scanner that scans an area in four different wavelengths: 0.5-0.6 (green), 0.6-0.7 (red), 0.7-0.8 (red/infrared) and 0.8-1.1 (infrared) [µm]. That is, one scan of a certain area give rise to four images where each one depicts the same area but in different wavelengths. Compare with a common RGB image which consists of three sub-images which carry information about the red, blue and green components of the image. Each image is stored as a grey scale image with 8 bits resolution. Figure 2 shows how each pixel is composed of the four images. Your task is to design a classifier that can be used to classify each pixel in a scan from the multi spectral scanner to belong to one of three classes \{water, thick forest, sparse forest\}. Raw satellite images together with images that can be used for training can be found in the file \texttt{landsat.mat}. You can look at the images with command \texttt{image} (see online help for more information). The following tables summarize the data available in \texttt{landsat.mat}.

<table>
<thead>
<tr>
<th>Full satellite scan (for validation)</th>
<th>Training scan, sparse forest</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td>Size</td>
</tr>
<tr>
<td>mss1</td>
<td>256 × 256</td>
</tr>
<tr>
<td>mss2</td>
<td>256 × 256</td>
</tr>
<tr>
<td>mss3</td>
<td>256 × 256</td>
</tr>
<tr>
<td>mss4</td>
<td>256 × 256</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Training scan, thick forest</th>
<th>Training scan, water</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td>Size</td>
</tr>
<tr>
<td>v21</td>
<td>16 × 16</td>
</tr>
<tr>
<td>v22</td>
<td>16 × 16</td>
</tr>
<tr>
<td>v23</td>
<td>16 × 16</td>
</tr>
<tr>
<td>v24</td>
<td>16 × 16</td>
</tr>
</tbody>
</table>

The three training scans were carefully selected by a human expert as representative samples of “water”, “thick forest” and “sparse forest”. You will use this data to train and test a Bayes classifier, and then validate the trained system on the full satellite scan. You will need to write new versions of your functions \texttt{boc_train} and \texttt{boc_test} because now we have three classes and the input data is 4-dimensional. To help you we have provided m-functions to carry out most of the necessary steps for feature extraction and validation, described as follows.

Feature Extraction

Use the provided m-function \texttt{sat_features}. This will automatically pre-process the scan data and create some matrices which are suitable for input to a classifier, as follows:
<table>
<thead>
<tr>
<th>Name</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>sp_forest</td>
<td>Training class 1 (sparse forest)</td>
</tr>
<tr>
<td>th_forest</td>
<td>Training class 2 (thick forest)</td>
</tr>
<tr>
<td>water</td>
<td>Training class 3 (water)</td>
</tr>
<tr>
<td>sp_forest_t</td>
<td>Testing class 1 (sparse forest)</td>
</tr>
<tr>
<td>th_forest_t</td>
<td>Testing class 2 (thick forest)</td>
</tr>
<tr>
<td>water_t</td>
<td>Testing class 3 (water)</td>
</tr>
</tbody>
</table>

Have a look at the code in `sat_features.m` and try to work out what it does.

**Validation**

Because this is a real world problem, we will now use a third data set to validate the performance of the trained classifier. Here we will pretend that we are really using the system on-board the satellite, so we won’t know what the true classes are. We will classify the full satellite scan, and display the classified pixels in three different colours for the three different classes. We have provided an incomplete m-function `sat_validate` to do this. At the moment, this will just create randomly coloured images. Modify this function to validate your trained classifier and display the classified images in an appropriate colour scheme.

**Deliverables**

Do the following:

1. Compute the parameters (the mean vectors and covariance matrices) for the Bayes optimal classifier by using the training data.

   - sparse forest
   - thick forest
   - water

   Class 1

   Class 2

   Class 3

2. Classify both the training vectors and the test vectors using your trained classifier. Calculate the percentage of correctly classified vectors for the training data and test data.

3. Validate your classifier on the full satellite scan. Display the coloured image.
Run `sat_features` again, and then repeat the whole experiment. What happens and why?

Be prepared to demonstrate all of your programs to the lab organizer.