**Neural and Evolutionary Computing.**

**Lab 1: Classification problems**
**Machine Learning test data repository**
**Weka data mining platform**
**Introduction Scilab**

---

## 1. Classification problems

The main aim of a classification problem is to design a classifier i.e. a system which assigns each input data to a class from a set of predefined classes.

**Examples:**

- Medical diagnosis:
  - the input data are symptoms, results of investigations, characteristics of the patient etc
  - the classes are related with the illness (a frequent case is when there are two classes: healthy and ill)
- Fault diagnosis:
  - the input data are characteristics of a given system
  - the classes corresponds to the system status (functional or with flaws)
- Character recognition:
  - the input data are vectors containing features of the characters
  - the classes corresponds to the characters (letter, digits, other symbols) to be recognized.
- Face recognition:
  - The input data are features extracted from digital images
  - The classes corresponds to the presence or absence of human faces in the image

Categories of data used when designing a classifier:

- *Training data*: data for which the corresponding class is known and are used to design the classifier through a learning process
- *Validation data*: data used to evaluate the behavior of the classifier during the learning process; the classification performance on the validation data is used to decide if the learning process should be stopped or not; early stopping of learning could be useful in order to avoid overtraining and enhance the generalization ability of the classifier
- *Testing data*: data used to evaluate the performance of a trained classifier

The classifiers can be of different types:

- Decision trees
- Decision (classification) rules
- Probabilistic approaches (Bayesian networks)
- Instance based classifiers (e.g. based on the nearest distance to some instances from the training set)
- Neural networks, Support Vector Machines

Main steps in designing a classifier:

- Preprocess the available data (e.g. select/ extract the relevant data features)

- Choose the classifier type (e.g. decision trees, classification rules, distance based, network-based etc.)
- Train the classifier, i.e. adapt the structure and/or estimate the adaptive parameters, such that it correctly classifies the data in the training set.
- Validate the classifier, i.e. evaluate the behavior of the classifier for data not included in the training set.

*Measures of classifiers quality.* The simplest quality measure is the classification accuracy: the ratio between the number of correctly classified instances and the total number of instances.

In the case to binary classification (there are two classes: a class of positive cases and a class of negative cases) the classifier performance can be illustrated using the so-called *confusion matrix*. The confusion matrix contains information about the percent of data which were correctly/ incorrectly classified. Let us consider the case when the first class corresponds to a positive diagnostic (e.g. malignant) and the second class corresponds to a negative diagnostic (e.g. benign). The confusion matrix contains the percent or absolute frequence of:

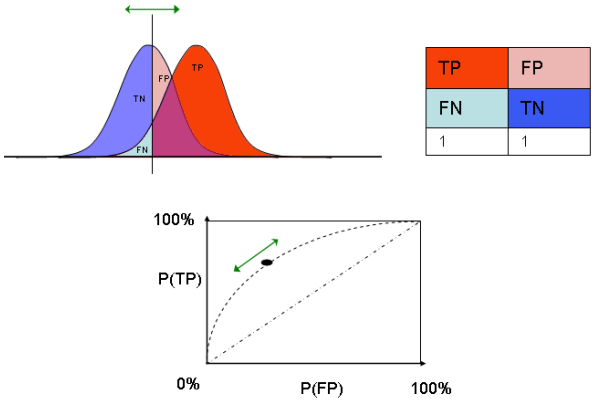
- *True positive cases* (TP): positive cases which were correctly classified in the first class
- *True negative cases* (TN): negative cases which were correctly classified in the second class
- *False positive cases* (FP): negative cases which were incorectly classified in the first class
- *False negative cases* (FN): positive cases which were incorrectly classified in the second class

Based on these values several other quality criteria can be computed:

*sensitivity=TP* / (*TP* + *FN*) (rmk: also called *recall*) *specificity=TN/(TN+FP)*

*precision=TP/(TP+FP)* *F=2*precision*recall/(precision+recall)*

Another possibility to illustrate the performance of a binary classifier depending on a discrimination threshold is the ROC (Receiver Operating Characteristic) curve which consists of points having the coordinates (*1-specificity*, *sensitivity*) and corresponding to different discrimination thresholds (in the case of binary classification the decision that an input belongs to a given class is usually taken based on a value in [0,1] and a threshold: if the value is larger than the threshold then the input belongs to the class otherwise it does not belong).




## 2. Neural Networks for classification problems

The most common neural network architecture used in solving classification problems is the *feedforward* one characterized by:

- An *input layer* having as many units as attributes are in data
- One or several *hidden layers* (as the number of hidden units is larger the model extracted by the neural network is more complex – but this is not necessarily beneficial for the problem to be solved as it can lead to model overfitting)
- An *output layer* having as many units as classes

The output signal corresponding to an input vector (that containing the features of the object to be classified) specifies the class to which belongs the object. There are at least two ways of interpreting the outputs of a neural network used for classification:

- In the most common case, the class is specified by the index of the aoutput unit producing the largest value
- In the case when the output values belong to (0,1) and their sum equals 1 then they can be interpreted as probabilities (the value corresponding to unit i corresponds to the probability that the input vector belongs to class i).

If the values produced by the output neurons are in (0,1), the correct answers (to be used in the learning process) are vectors containing one value equal to 1 on the position corresponding to the right class and equal to 0 on all the other positions.

The *learning* process is controlled by a *cross-validation technique* which consists in dividing the initial set of data into three slices:

- *Training data*: the data used in the training algorithm to compute the adjustments for connections weights
- *Validation data*: when the classification error on these data starts to increase, the training process is stopped (these data are not used to compute the weight adjustments but only to decide if the network has generalization ability)
- *Test data*: these data are not used in the training process but only to evaluate the quality of the classifier

**3. Weka data mining platform (**http://www.cs.waikato.ac.nz/ml/weka/ **)**

- Weka is a collection of machine learning algorithms (including several types of neural networks: multilayer perceptron, radial basis functions networks) designed to solve various data mining tasks: data visualization, attribute selection, classification, clustering, association rules extraction.

- It is open source and the methods can be used from the Explorer GUI or directly from a Java code. It allows to compare different methods (Experimenter GUI) or to define workflows of data mining tasks (Knowledge Flow GUI).

**Exercise 1.** Open the file *breast_cancer_nominal.csv* in Weka and compare the accuracy of the following classifiers:

- Classification rules: Rules-> OneR, Rules-> ConjunctiveRules, Rules->NNge
- Decision trees: Trees->J48, Trees->RandomForest
- Nearest Neighbor: Lazy->IBk
- Bayesian Networks: bayes->NaiveBayes
- Neural Networks: functions->MultilayerPerceptron, functions->RBFnetwork,
- Support Vector Machines: functions->SMO

**Hint:** All classifiers are in the Classify panel of the Weka Explorer. For neural networks classifiers in order to see a graphical representation of the network architecture set GUI option in the panel opened when click on the classifier bar.

**4. Machine Learning Repository (**http://archive.ics.uci.edu/ml/ )

This archive contains a lot of data from various domains (life sciences, physical sciences, computer science and engineering) which can be used to train, validate and test classifiers and other machine learning techniques.

**Exercise 2.** Visit the repository, download one of the dataset for classification (at your choice) and use it to test classifiers from Weka (see Ex. 1)

**Appendix 1: Scilab** - Open source software for numerical computation
(http://www.scilab.org/)

Scilab is an interpreted programming language which offers support for computational tasks arising in linear algebra, polynomials operations, interpolation and approximation, linear and quadratic optimization, differential equations, signal processing, statistics and graphics.

In Scilab the basic object is the matrix, both the vectors and the scalars being particular cases of matrices.

*Some general aspects:*

- Scilab is case-sensitive
- Being an interpreter, the variables do not have to be declared but they should have a value assigned; the assignment operator is =
- The predefined constants have names with the prefix % (e.g. %pi, %i, %e, %t (true), %f (false))
- The relational operators are: == (equal), ~= or <> (unequal), <=, >=
- The logical operators are: ~ (not), & (and), | (or)
- The result of an evaluation which is not explicitly assigned to a variable is implicitly assigned to the object ans which can be used in the following command
- The commands specified on the same line should be separated by ; (this separator has also the effect of inhibiting the visualization of the last evaluation result).
- The strings are specified using doubl quotes (") and their concatenation can be realized using +
- The line comments can be specified by //

*Specifying matrices:*

- Explicitly, by specifying all elements (the elements on the same row should be separated by , or space and the rows should be separated by ; or enter):
  A=[a11,a12,…a1n; a21,a22,…,a2n; …;am1, am2,…,amn]

- Implicitly, by using functions which generate matrices:
  - zeros(m,n) : matrix with m rows and n columns and elements equal to 0
  - ones(m,n): matrix with m rows and n columns and elements equal to 1
  - rand(m,n): matrix with m rows and n columns and elements randomly generated in (0,1)

*Operations with matrices.*

- Finding the size: size(mat) returns [nr rows, nr columns]
- Reorganizing a matrix: matrix(mat, nr rows, nr columns) returns a matrix with the specified size and elements taken row by row from the object specified as the first parameter
- Accessing the elements: mat(row index,column index).
  Obs: the indices can be individual values or ranges specified as inf:sup. The last index of a row or column can be specified by $. For instance mat($,$-1) specifies the element on the last row and the column before the last one.
  Obs: the indices start with 1
- Changing a matrix:
  - Change an element: mat(i,j)=val

- Add a row: mat=[mat; el1, el2,…,eln]
- Add a column: mat=[mat'; el1, el2,…,elm]' (the operator ' denotes the transpose)
- Remove a row: mat(i,:)=[]
- Remove a column: mat(:,j)=[]

- Arithmetical operations: all arithmetical operations are vectorized; in order to specify operations at the level of elements the operators should be prefixed by . (dot). For instance, A*B returns the algebraic product of matrices A and B ans A.*B returns the matrix which is obtained by multiplying the corresponding elements from the matrices.

*Other types of objects in Scilav:*

- Structures:
  - struct(fieldname1,value1, fieldname2,value2,…., fieldnamen,valuen)
  - Example: date=struct('day',3,'month','october','year',2014)
  - Element specification: StructureName.Fieldname(e.g.: date.day is 3)

- Heterogeneous lists:
  - Simple list: list(Element1,Element2,…,Elementn)
    Remark: the elements are specified using indices
  - Typed list: tlist(Names,Element1,Element2,…,Elementn) ;
    Example: d=tlist(['date','day','month','year'],3,'october',2014)
    Remark: the elements can be specified both by indexing abd by qualification:
    d(2) is identical to d.day

*Instructions*

- **If statement:**

```
if (condition) then
        <statements 1>
else
        < statements 2>
end
```

Variant:

```
if (condition1) then
        < statements 1>
elseif (condition 2)
        < statements 2>
else
        < statements 3>
end
```

- **Select statement:**

```
select < selector>
case < val 1>
        < statements 1>
case <val 2>
```

```
        < statements 2>
…
case <val n>
        < statements n>
else
        <other statements >
end
```

- **for statement**

```
for contor=inf:step:sup
        < statements >
end
```

Remark: If step is 1 the it can be ommitted. The value of the step can be less than 0. The iteration can be done over the elements of a vector (one row matrix):

```
for contor=vector
        < statements >
end
```

- **while statement**

```
while(conditie)
        < statements >
end
```

*Functions*

Scilab functions can be used to compute several results (specified by output variables in the function header):

```
function [output1, …,outputm]=functionName(input1,…,inputn)
        <function body>
endfunction
```

*Graphics*

Several graphics functions:

- plot - one-dimensional functions
- fplot3d and contour - surfaces
- paramfplot2d – curves given by parametric equations
- polarplot – polar coordinates representation

Example:

```
// one-dimensional function to be plotted
function y=f(x)
   y=x*x/10+sin(x)
endfunction

x=-2*%pi:0.1:2*%pi
clf
plot(x,f)

// surface
function y=f2arg(x1, x2) // function with 2 arguments
   y = x1 **2 + x2 **2;
endfunction
x1data = linspace ( -1 , 1 , 100 ); // this is equivalent with x1data = -1:0.1:1;
x2data = linspace ( -1 , 1 , 100 );
contour ( x1data , x2data , f2arg , 10)  // contour plot
pause
clf                              // clean the screen
fplot3d( x1data , x2data , f2arg)       // surface plot
```