Machine Learning
(Course 395)

Computer Based Coursework Manual

Lecturer:
Maja Pantic

CBC Helpers:
Joan Alabort
Bihan Jiang
Ioannis Marras
Brais Martinez
Mihalis Nicolaou
Antonis Oikonomopoulos
Stavros Petridis
Ognjen Rudovic
Jie Shen

Imperial College
London

Table of Contents

1. Introduction

The purpose of this Computer-Based Coursework (CBC) is to provide the students with hands-on experience in implementing and testing basic machine learning techniques. The techniques that will be examined are Decision Trees (DT), Artificial Neural Networks (ANN) and Case Based Reasoning (CBR). Each of these techniques will be used in order to identify six basic emotions from the human facial expression (anger, disgust, fear, happiness, sadness and surprise) based on a labelled set of facial Action Units (AUs). The latter correspond to contractions of human facial muscles, which underlie each and every facial expression, including facial expressions of the six basic emotions. More theory and details on Facial Action Units and their relation to emotions will be given in section 3.
The implementation of the aforementioned techniques requires understanding of these techniques. For this reason, following the lectures of the course is strongly advised.

2. Organization

a) Working Method

Implementation of the algorithms will be done using MATLAB. The students will work in groups of 4 students. They are expected to work together on implementation of each machine learning technique and the related emotion recognizer. The groups will be formed shortly after the first lecture (for lecture schedule see http://ibug.doc.ic.ac.uk/courses/machine-learning-course-395/) and a CBC helper will be assigned to each group. The implementation will be either done from scratch (DT and CBR) or by using special toolboxes (ANN) which do not belong to the standard MATLAB package, but will be provided to the students. After an assignment is completed, the code generated by each group will be evaluated by the CBC helpers. This will be done in the lab and by using a separate test set that will not be available to the students. The implemented algorithms will have to score a predefined minimum classification rate on this unknown test set. In addition, each group must hand in, via the CATe system, a report of approximately **2 pages** (excluding result matrices and graphs), and explaining details of the implementation process of each algorithm along with comments on the acquired results. Your code (which should run on any lab computer) should also be included in the CATe submission. All files (including the report) have to be combined in one archive file.

You should also inform us about your team members by email by 29th January (machinelearningtas@gmail.com) with the following information:

- Student login
- Correspondence email
- Full first Name
- Family Name
- Preferred name (if different from first name).
- Degree, course/study taken, and the current year in that course.

If you cannot form a team with 4 members, then email us the above information and we will assign you to a team. Please note that we only accept requests for groups of 4.

Members in a request for a different group size will be assigned randomly to separate groups.
Deliverables for every assignment will be described at the end of every section describing the assignment in question. Each group is responsible for the way in which the assignments are implemented and the reports are prepared and presented. These reports provide feedback on the performance of the group as a whole. The individual performance of each group member will be evaluated through personal interviews of 10-15 minutes between the CBC helpers and **all** group members. The former will ask random group members about details of the delivered code and report. The interviews will take place during lab hours according to the timetable that will be emailed to the students once groups are formed.
Each group may be asked to attend a final interview with the lecturer and the CBC helpers. These interviews will take place on Friday the 23rd of March, between 10:00am and 1:00 pm. Please make sure that you are available on that day. On that date, the individual performances of the group members will be discussed and final grades will be awarded.

b) Role of the CBC helpers

The role of the CBC helpers is to monitor the implementation of the assignments by the students. The CBC helpers, however, will not make any substantive contribution to the implementation process. Final grading will be exclusively done by the lecturer of the course, who will, nevertheless, ask for the recommendations of the CBC helpers concerning the group progress.

c) Communication

Communication between the students and the CBC helpers is very important, and will be done in labs during the CBC sessions or via email using the following address:

machinelearningtas@gmail.com

Please mention your group number and your assigned helper in any communication; this makes it easier for us to divide the work. In addition, students should visit the website of the course, at
http://ibug.doc.ic.ac.uk/courses/machine-learning-course-395/, in order to download the required data files and various MATLAB functions needed in order to complete the assignments of this CBC. Also many useful links and information will be posted onto this website.

d) Time Management

In total, there are 4 assignments to be completed. As mentioned before, after the completion of each assignment a report of approximately two pages must be handed in and a discussion between the CBC helpers and the group members will take place. The deadlines for handing in each assignment are as follows:

- Assignment 1: No hand in required.
- Assignment 2: Wednesday February 8th .
- Assignment 3: Wednesday February 22nd .

- Assignment 4: Tuesday March 6th .
- Assignment 5: Wednesday March 14th.

e) Grading

After all the assignments are completed and the reports are handed in, every group member will be assigned a grade (therefore the final grade can be different for every student). In this CBC, we expect each group member to actively participate in the implementation of the algorithms, the reports and the interviews between them and the CBC helpers. Each individual assignment will be graded based on the MATLAB code that was developed and the report that was delivered. The final CBC grade for each group member will be a combination of the group grade per assignment (*one_assignment_grade*) and the group member's personal contribution (*overall_personal_mark*) to the group progress, according to the following:

one_assignment_grade = 0.5 report_content + 0.4* implementation + 0.1* report_quality*

four_assignment_ grades = (one_assignment_grade[Ass 2] + one_assignment_grade[Ass 3] + one_assignment_grade[Ass 4] + one_assignment_grade[Ass 5]) / 4

*CBC_grade = 0.6*four_assignment_grades + 0.4*overall_personal_mark*

implementation refers to the code (i.e., whether it works, how it works, what it produces, whether comments and explanations are provided, etc.), *report_content* refers to what is provided in the report (e.g., results, analysis and discussion of the results, novelty in the methodology) and *report_quality* refers to quality of presentation.
NOTE: **CBC accounts for 33% of the final grade for the Machine Learning Course**. In other words, final grade = 0.67*exam_grade + 0.33*CBC_grade. For example, if the exam_grade = 32/100 and the CBC_grade = 80/100, then final_grade = 48/100.

f) Assignment submission guidelines

In order to avoid negative consequences related to CBC assignment submission, *strictly* follow the points listed below.

- You should *work in groups*. Take note that only *one report per group* will be accepted.
- Send a *timely* email to the THs with the *full* list of group members, and the following information for <u>each and every group member</u>:
    - Student login
    - Correspondence email
    - Full first Name
    - Family Name
    - Preferred name (if different from first name).
    - Degree, course/study taken, and the current year in that course.
- The *text in your report* should <u>not exceed 2 pages</u>, and the report should <u>not exceed 5 pages</u>, including text, graphs, results etc.

- Make sure you mention your group number in each of your reports, as well as at each communication with the CBC helpers.
- *Strictly follow* the assignment *submission deadlines and times* specified on CATE.
- Each and every group member *individually has to confirm* that they are part of that particular group, for *each and every assignment submission* (under the pre-determined group leader) before each assignment submission deadline.

g) Outline of the manual

The remaining of this CBC manual is organized as follows. Section 3 introduces the Facial Action Coding System (FACS). It explains the meaning of each AU as well as the relation between the AUs and the six basic emotions. Section 4 introduces the basic system-evaluation concepts including K-fold cross-validation, confusion matrices, recall and precision rates.
Section 5 contains the first (optional) assignment by providing an introduction on MATLAB fundamentals via a number of exercises. Sections 6, 7, and 8 explain the assignments 2-4 and the machine learning techniques that have to be implemented. Section 9 explains the fifth CBC assignment and the *t*- and *ANOVA* tests that will be used to compare the performance between the various implemented techniques.

3. The Facial Action Coding System and the basic emotions

One of the great challenges of our times in computer science research is the automatic recognition of human facial expressions. Machines capable of performing this task have many applications in areas as diverse as behavioural sciences, security, medicine, gaming and human-machine interaction (HMI). The importance of facial expressions in inter-human communication has been shown by numerous cognitive scientists. For instance, we use our facial expressions to synchronize a conversation, to show how we feel and to signal agreement, denial, understanding or confusion, to name just a few. Because humans communicate in a far more natural way with each other then they do with machines, it is a logical step to design machines that can emulate inter-human interaction in order to come to the same natural interaction between man and machine. To do so, machines should be able to detect and understand our facial expressions, as they are an essential part of inter-human communication.

a) FACS

Traditionally, facial expression recognition systems attempt to recognize a discrete set of facial expressions. This set usually includes six 'basic' emotions: anger, disgust, fear, happiness, sadness and surprise. However, the number of possible facial expressions that humans can use numbers about 10,000, many of which cannot be put in one of the six basic emotion categories (think for example of expressions of boredom, 'I don't know', or a brow-flash greeting). In addition, there is more than one ways to display the same feeling or emotion. Therefore, describing a facial expression

in such loose terms as 'happy', 'sad' or 'surprised' is certainly not very exact, greatly depending on who is describing the currently displayed facial expression while leaving a large variation of displayed expressions possible within the emotion classes. The activation of the facial muscles on the other hand can be described very precisely, as each muscle or group of muscles can be said to be either relaxed or contracted at any given time. As every human has the same configuration of facial muscles, describing a facial expression in terms of facial muscle activations would result in the same description of a facial expression, regardless of the person displaying the expression and regardless of who was asked to describe the facial expression. The Facial Action Coding System (FACS[1,2]), proposed by psychologists Ekman and Friesen, describes all the possible facial muscle (de)activations that cause a visible change in the appearance of the face. Every muscle activation that causes visible appearance changes is called an Action Unit (AU). The FACS consists of 44 AUs (see Fig. 1 for examples).

Figure 1: Examples of AUs

b) Action Units and emotions

The same psychologists who proposed the FACS also claimed that there exist six 'basic' emotions (anger, disgust, fear, happiness, sadness and surprise) that are universally displayed and recognized in the same way. As we already mentioned, many research groups have proposed systems that are able to recognize these six basic emotions. Almost all proposed emotion detectors recognize emotions directly from raw data. In this CBC we will use a different approach to emotion detection. Instead of directly classifying a set of features extracted from the images into emotion categories, we will use AUs as an intermediate layer of abstraction. The rules that map AUs present in a facial expression into one of the six basic emotions are given in Table 1. In this CBC we will not use these rules directly but instead we will try to learn emotional classification of AUs using different machine learning techniques. Also, in this CBC we consider the step of AU detection to be solved. Students are provided with a dataset that contains tuples consisting of a list of AUs and the corresponding emotion label.

[1] P. Ekman and W.V. Friesen, The Facial Action Coding System: A Technique for the Measurement of Facial Movement, San Francisco: Consulting Psychologist, 1978
[2] P. Ekman, W.V. Friesen and J.C. Hager, "The Facial Action Coding System: A Technique for the Measurement of Facial Movement", San Francisco: Consulting Psychologist, 2002

Figure 2. Typical smile includes activation of AU6, AU12 and AU25.

Emotion	AUs	Emotion	AUs
Happy	{12}	Fear	{1,2,4}
	{6,12}		{1,2,4,5,20,
Sadness	{1,4}		25||26||27}
	{1,4,11||15}		{1,2,4,5,25||26||27}
	{1,4,15,17}		{1,2,4,5}
	{6,15}		{1,2,5,25||26||27}
	{11,17}		{5,20,25||26||27}
	{1}		{5,20}
Surprise	{1,2,5,26||27}		{20}
	{1,2,5}	Anger	{4,5,7,10,22,23,25||26}
	{1,2,26||27}		{4,5,7,10,23,25||26}
	{5,26||27}		{4,5,7,17,23||24}
Disgust	{9||10,17}		{4,5,7,23||24}
	{9||10,16,25||26}		{4,5||7}
	{9||10}		{17,24}

Table 1. Rules for mapping Action Units to emotions, according to FACS. A||B means 'either A or B'

c) DATA

The data for this CBC will be provided to the students in the form of text files. Each line of these files corresponds to a specific example and contains information about this example, separated by a whitespace. The first column represents the example index, the second column is the emotion label, and the rest of the columns represent the AUs that are present in that example. The students will be provided with a MATLAB function named *loaddata.m* in order to properly read the files and store the information in them in suitable MATLAB variables. The syntax of this function is as follows:

```
[x, y] = loaddata('filename');
```

where *filename* is the full path of the input file (usually a .txt file that is provided) and *x, y* are the outputs:

- A matrix *x*, which is an *N×45* matrix, where *N* is the total number of examples and 45 is the total number of AUs that can be activated or not. In case an AU is activated, the value of the corresponding column will be 1. Otherwise, it will be 0. For instance, the following row

[1 1 0 0 1 0 …0]

would mean that AU1, AU2 and AU5 are activated.

- A vector *y* of dimensions *N×1*, containing the emotion labels of the corresponding examples. These labels are numbered from 1 to 6, and correspond to the emotions anger, disgust, fear, happiness, sadness and surprise respectively.

In addition, the students will be provided with functions that map emotion labels (numbers 1 to 6) to actual emotions (anger, disgust, fear, happiness, sadness, surprise) and back. These files are called *emolab2str.m* and *str2emolab.m* respectively.
During this CBC, the students will work with two types of data: *clean* and *noisy* data, each given as a separate text file. *Clean* data was obtained by human experts. The AU and emotion information in this type of data is considered correct for every example. On the other hand, the AUs in the *noisy* data were obtained by an automated system for AU recognition[3]. Since the system is not 100% accurate, the output of the system can contain wrongly detected AUs and some AUs can be missing.

4. System Evaluation

In this section, the basic system evaluation concepts that will be used throughout this CBC are given. These include:

- K-fold Cross Validation
- The Confusion Matrix
- Recall and Precision Rates
- The F_α-measure

a) Cross Validation

The concept of cross-validation is closely related to overfitting, a concept very important to machine learning. Usually a learning algorithm is trained using some set of training examples, i.e., exemplary situations for which the desired output is known. The learner is assumed to reach a state where it will also be able to predict the correct output for other examples, thus generalizing to situations not presented during training. However, in cases where learning was performed for too long or where training examples are not representative of all situations that can be encountered, the learner may adjust to very specific random features of the training data that have no causal relation to the target function. In the process of overfitting, the performance on the training examples still increases while the performance on unseen data becomes worse. An example of overfitting can be seen in Fig.3. The point where the error on the test set increases is the point where overfitting occurs.
In order to avoid overfitting, it is necessary to use additional techniques, one of which is cross-validation. Cross-validation is the statistical practice of partitioning a sample of data into subsets such that the analysis is initially performed on a single subset, while the other subset(s) are retained for subsequent use in confirming and validating

3 M.F. Valstar and M. Pantic, "Fully automatic facial action unit detection and temporal analysis", Proc. IEEE Int'l Conf. Computer Vision and Pattern Recognition, vol 3, p. 149, 2006

the initial analysis. The initial subset of data is called the *training set*; the other subset(s) are called *test set*.
The process of cross-validation is schematically depicted in Fig.4, where the initial dataset is split *N* times (*N-fold* cross validation) in order to give *N* error estimates. The final error will be the average of these *N* estimates, as shown in the figure.

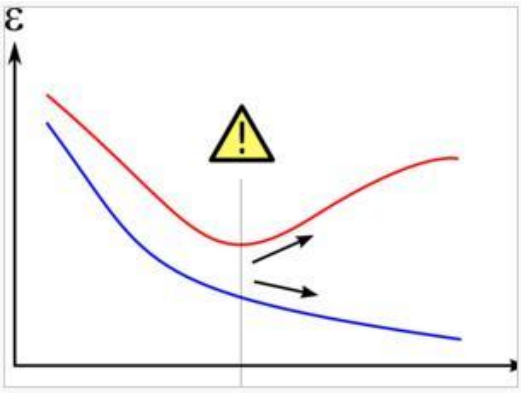

Fig.3: Overfitting/Overtraining in supervised learning (e.g. in a neural network). Training error is shown in blue, validation error in red. If the validation error increases while the training error steadily decreases then a situation of overfitting may have occurred.

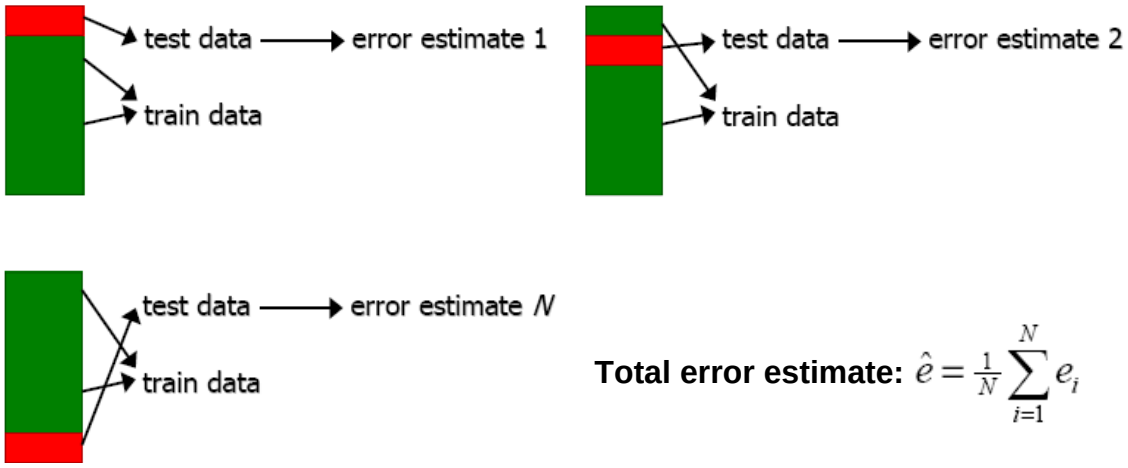

Fig.4: *N-fold* cross validation process. The data is split each time into training and testing datasets. The final prediction error is the mean average error.

b) Confusion Matrix

A confusion matrix is a visualization tool typically used to present the results attained by a learner. Each column of the matrix represents the instances in a predicted class, while each row represents the instances in an actual class. One benefit of a confusion matrix is that it is easy to see if the system is confusing two classes (i.e. commonly mislabelling one as another). In the example confusion matrix below (Table 2), of the 8 actual cats, the system predicted that three were dogs, and of the six dogs, it predicted that one was a rabbit and two were cats. We can see from the matrix that the system in question has trouble distinguishing between cats and dogs, but can make the distinction between rabbits and other types of animals pretty well.

	Cat	Dog	Rabbit
Cat	5	3	0
Dog	2	3	1
Rabbit	0	2	11

Table 2: A simple confusion matrix

c) Recall and Precision Rates

Consider a binary classification problem, that is, containing only positive or negative examples, and a classifier that classifies these examples into two possible classes. A positive example assigned to the positive/negative class is called *True Positive/False Negative*. In the same way the *True Negative/False Positive* terms are defined. Based on this classification, the most obvious way to measure the performance of a classifier is to calculate the correct classification rate, defined as the sum of True Positives and True Negatives, divided by the total number of the examples.
There are cases, however, where the classification rate can be misleading. Consider the two following cases, depicted in the following tables:

Classifier	**TP**	**TN**	**FP**	**FN**	**Recognition Rate**
A	25	25	25	25	50%
B	37	37	13	13	74%

Classifier	**TP**	**TN**	**FP**	**FN**	**Recognition Rate**
A	25	75	75	25	50%
B	0	150	0	50	75%

Table 3: Two different classification scenarios. The top matrix depicts a classification problem using a balanced dataset (same number of positive and negative examples), while in the bottom matrix, the dataset is unbalanced.

It is clear, from the first table that classifier B is better than A, since its classification rate is significantly larger and the number of positive and negative examples is the same. For the second table however, it is clear that, while classifier B correctly classifies the negative examples, it misses all the positive ones, in contrast to classifier A, whose classification performance is more balanced between the two classes. In order to overcome this ambiguity, and to be able to compare the two classifiers, the recall and precision rates are used instead.
Recall and Precision measure the quality of an information retrieval process, e.g., a classification process. Recall describes the completeness of the retrieval. It is defined as the portion of the positive examples retrieved by the process versus the total number of existing positive examples (including the ones not retrieved by the process). Precision describes the actual accuracy of the retrieval, and is defined as the portion of the positive examples that exist in the total number of examples retrieved.

A schematic representation of the recall and precision rates is given in Fig.5. *A* represents the set of True Positives, *B* is the total set of positives and *C* is the set of negative examples that were wrongly classified as positives, that is, False Positives. Based on the recall and precision rates, we can justify if a classifier is better than another, i.e. if its recall and precision rates are significantly better.

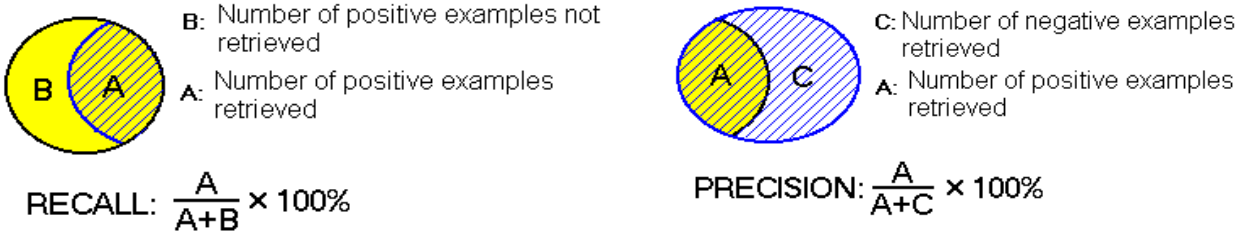

Fig.5: Schematic representation of the recall and precision rates.

d) F_α measure

While recall and precision rates can be individually used to determine the quality of a classifier, it is often more convenient to have a single measure to do the same assessment. The F_α measure combines the recall and precision rates in a single equation:

$$F_\alpha = (1+\alpha)\frac{precision * recall}{\alpha * precision + recall},$$

where α defines how recall and precision will be weighted. In case recall and precision are evenly weighted then the F_1 measure is defined as follows:

$$F_1 = 2\frac{precision * recall}{precision + recall}.$$

e) Binary vs. Multi-class Classification

So far we have only talk about classifiers using datasets with only positive or negative examples. These classifiers are called binary classifiers, since the class labels can only take two values: ±1 (positive/negative). Many real-world problems, however, have more than two classes. In these cases we are talking about multi-class classification. To get M-class classifiers, it is common to construct a set of binary classifiers $f^1 \ldots f^M$, each trained to separate one class from the rest, and combine them by doing the multi-class classification according to the maximal output, that is, by taking:

$$\arg\max_{j=1..M} g^j(x),$$

where $g^j(x) = \mathrm{sgn}(f^j(x))$, and sgn() is the sign function. In case of a tie, i.e. two or more classifiers output +1, one way to break the tie is to choose one class randomly among the classes which were proposed by the classifiers. However, depending on the classifier, there are other ways to break the tie. This is called one versus the rest classification, where each separate classifier is trained on one particular class, the positive class, while all the examples that belong to the rest of the classes are pooled together in order to form the negative class. One possible drawback of this method is that every classifier is trained on a small set of positive examples and a large set of negative ones, leading to an unbalanced training set, as described in section (c).

Pair wise classification trains a classifier for each possible pair of classes. For M classes, this results in (M-1)M/2 binary classifiers. This number is usually larger than the number of one-versus-the-rest classifiers; for instance, if M=10, we need to train 45 binary classifiers rather than 10 as in the method above. When we try to classify a test pattern, we evaluate all 45 binary classifiers, and classify according to which of the classes gets the highest number of votes. A vote for a given class is defined as a classifier putting the pattern into that class. The individual classifiers, however, are usually smaller in size than they would be in the one-versus-the-rest approach. This is for two reasons: first, the training sets are smaller, and second, the problems to be learned are usually easier, since the classes have less overlap.

5. Assignment 1: MATLAB Exercises

Assignment 1 is an optional assignment. It aims to provide a brief introduction to some basic concepts of MATLAB (that are needed in Assignments 2-5) without assessing students' acquisition, application and integration of this basic knowledge. The students, are strongly encouraged to go through all the material, experiment with various functions, and use the MATLAB help files extensively (accessible via the main MATLAB window).
Type “doc” on the MATLAB command line to open the help browser. MATLAB blogs also provide useful information about how to program in MATLAB (http://blogs.mathworks.com).

a) Vectors and Arrays

A vector in MATLAB can be easily created by entering each element between brackets and assigning it to a variable, e.g. :

```
a = [1 2 3 4 5 6 9 8 7]
```

Let's say you want to create a vector with elements between 0 and 20 evenly spaced in increments of 2:

```
t = 0:2:20
```

MATLAB will return:

```
t =
     0   2   4   6   8   10   12   14   16   18   20
```

Manipulating vectors is almost as easy as creating them. First, suppose you would like to add 2 to each of the elements in vector 'a'. The equation for that looks like:

```
b = a + 2

b =
     3   4   5   6   7   8   11   10   9
```

Now suppose you would like to add two vectors together. If the two vectors are the same length, it is easy. Simply add the two as shown below:

```
c = a + b

c =
     4   6   8   10   12   14   20   18   16
```

In case the vectors have different lengths, then an error massage will be generated.
Entering matrices into MATLAB is the same as entering a vector, except each row of elements is separated by a semicolon (;) or a return:

```
B = [1 2 3 4;5 6 7 8;9 10 11 12]

B =
     1     2     3     4
     5     6     7     8
     9    10    11    12

B = [ 1  2  3  4
      5  6  7  8
      9 10 11 12]
```

```
B =
     1     2     3     4
     5     6     7     8
     9    10    11    12
```

Matrices in MATLAB can be manipulated in many ways. For one, you can find the transpose of a matrix using the apostrophe key:

```
C = B'

C =
     1     5     9
     2     6    10
     3     7    11
     4     8    12
```

Now, you can multiply the two matrices B and C together. Remember that order matters when multiplying matrices.

```
D = B * C

D =
    30    70   110
    70   174   278
   110   278   446

D  = C * B

D =
   107   122   137   152
   122   140   158   176
   137   158   179   200
   152   176   200   224
```

Another option for matrix manipulation is that you can multiply the corresponding elements of two matrices using the .* operator (the matrices must be the same size to do this).

```
E = [1 2;3 4]
F = [2 3;4 5]
G = E .* F
E =
     1     2
     3     4
F =
     2     3
     4     5
G =
     2     6
    12    20
```

MATLAB also allows multidimensional arrays, that is, arrays with more than two subscripts. For example,

```
R = randn(3,4,5);
```

creates a 3-by-4-by-5 array with a total of 3x4x5 = 60 normally distributed random elements.

b) Cell arrays and structures

Cell arrays in MATLAB are multidimensional arrays whose elements are copies of other arrays. A cell array of empty matrices can be created with the `cell` function. But, more often, cell arrays are created by enclosing a miscellaneous collection of things in curly braces, `{}`. The curly braces are also used with subscripts to access the contents of various cells. For example

```
C = {A sum(A) prod(prod(A))}
```

produces a 1-by-3 cell array. There are two important points to remember. First, to retrieve the contents of one of the cells, use subscripts in curly braces, for example *C{1}* retrieves the first cell of the array. Second, cell arrays contain *copies* of other arrays, not *pointers* to those arrays. If you subsequently change `A`, nothing happens to `C`.

Three-dimensional arrays can be used to store a sequence of matrices of the *same* size. Cell arrays can be used to store sequences of matrices of *different* sizes. For example,

```
M = cell(8,1);
for n = 1:8
   M{n} = magic(n);
end
M
```

produces a sequence of magic squares of different order:

```
M =
    [           1]
    [ 2x2  double]
    [ 3x3  double]
    [ 4x4  double]
    [ 5x5  double]
    [ 6x6  double]
    [ 7x7  double]
    [ 8x8  double]
```

Structures are multidimensional MATLAB arrays with elements accessed by textual *field designators*. For example,

```
S.name = 'Ed Plum';
S.score = 83;
S.grade = 'B+'
```

creates a scalar structure with three fields.

```
S =
     name: 'Ed Plum'
    score: 83
    grade: 'B+'
```

Like everything else in MATLAB, structures are arrays, so you can insert additional elements. In this case, each element of the array is a structure with several fields. The fields can be added one at a time,

```
S(2).name = 'Toni Miller';
S(2).score = 91;
S(2).grade = 'A-';
```

Or, an entire element can be added with a single statement.

```
S(3) = struct('name','Jerry Garcia',...
              'score',70,'grade','C')
```

Now the structure is large enough that only a summary is printed.

```
S =
1x3 struct array with fields:
    name
    score
    grade
```

There are several ways to reassemble the various fields into other MATLAB arrays. They are all based on the notation of a *comma separated list*. If you type

```
S.score
```

it is the same as typing

```
S(1).score, S(2).score, S(3).score
```

This is a comma separated list. Without any other punctuation, it is not very useful. It assigns the three scores, one at a time, to the default variable `ans` and dutifully prints out the result of each assignment. But when you enclose the expression in square brackets,

```
[S.score]
```

it is the same as

```
[S(1).score, S(2).score, S(3).score]
```

which produces a numeric row vector containing all of the scores.

```
ans =
    83    91    70
```

Similarly, typing

```
S.name
```

just assigns the names, one at time, to `ans`. But enclosing the expression in curly braces,

```
{S.name}
```

creates a 1-by-3 cell array containing the three names.

```
ans =
    'Ed Plum'    'Toni Miller'    'Jerry Garcia'
```

And

```
char(S.name)
```

calls the `char` function with three arguments to create a character array from the `name` fields,

```
ans =
Ed Plum
Toni Miller
Jerry Garcia
```

c) Functions

To make life easier, MATLAB includes many standard functions. Each function is a block of code that accomplishes a specific task. MATLAB contains all of the standard functions such as sin, cos, log, exp, sqrt, as well as many others. Commonly used constants such as pi, and i or j for the square root of -1, are also incorporated into MATLAB.

```
sin(pi/4)

ans =

    0.7071
```

To determine the usage of any function, type `help [function name]` at the MATLAB command window.

MATLAB allows you to write your own functions with the *function* command. The basic syntax of a function is:

```
function [output1,output2] = filename(input1,input2,input3)
```

A function can input or output as many variables as are needed. Below is a simple example of what a function, `add.m`, might look like:

```
function [var3] = add(var1,var2)
%add is a function that adds two numbers
var3 = var1+var2;
```

If you save these three lines in a file called "add.m" in the MATLAB directory, then you can use it by typing at the command line:

```
y = add(3,8)
```

Obviously, most functions will be more complex than the one demonstrated here. This example just shows what the basic form looks like.

d) Loops

If you want to repeat some action in a predetermined way, you can use the *for* or *while loop*. All of the loop structures in MATLAB are started with a keyword such as "for", or "while" and they all end with the word "end".

The *for loop* is written around some set of statements, and you must tell MATLAB where to start and where to end. Basically, you give a vector in the "for" statement, and MATLAB will loop through for each value in the vector: For example, a simple loop will go around four times each time changing a loop variable, *j*:

```
for j=1:4,
     j
 end
```

If you don't like the *for loop*, you can also use a *while loop*. The *while loop* repeats a sequence of commands as long as some condition is met. For example, the code that follows will print the value of the *j* variable until this is equal to 4:

```
j=0
while j<5
      j
      j=j+1;
end
```

You can find more information about *for loops* on http://blogs.mathworks.com/loren/2006/07/19/how-for-works/

e) Reading from files / Writing to files

Before we can read anything from a file, we need to open it via the *fopen* function. We tell MATLAB the name of the file, and it goes off to find it on the disk. If it can't find the file, it returns with an error; even if the file does exist, we might not be allowed to read from it. So, we need to check the value returned by *fopen* to make sure that all went well. A typical call looks like this:

```
fid = fopen(filename, 'r');
if (fid == -1)
  error('cannot open file for reading');
end
```

There are two input arguments to *fopen*: the first is a string with the name of the file to open, and the second is a short string which indicates the operations we wish to undertake. The string 'r' means "we are going to read data which already exists in the file." We assign the result of *fopen* to the variable *fid*. This will be an integer, called the "file descriptor," which we can use later on to tell MATLAB where to look for input.

There are several ways to read data from a file we have just opened. In order to read binary data from the file, we can use the *fread* command as follows:

```
A = fread(fid, count)
```

where *fid* is given by *fopen* and *count* is the number of elements that we want to read. At the end of the *fread*, MATLAB sets the file pointer to the next byte to be read. A subsequent *fread* will begin at the location of the file pointer. For reading multiple elements from the file a loop can be used in combination with *fread*.

If we want to read a whole line from the file we can use the *fgets* command. For multiple lines we can combine this command with a loop, e.g. :

```
while (done_yet == 0)

    line = fgets(fid);
    if (line == -1)
      done_yet = 1;
    end
```

Before we can write anything into a file, we need to open it via the *fopen* function. We tell MATLAB the name of the file, and give the second argument 'w', which stands for 'we are about to write data into this file'.

```
fid = fopen(filename, 'w');
if (fid == -1)
  error('cannot open file for writing');
end
```

When we open a file for reading, it's an error if the file doesn't exist. But when we open a file for writing, it's not an error: the file will be created if it doesn't exist. If the file does exist, all its contents will be destroyed, and replaced with the material we place into it via subsequent calls to *fprintf*. Be sure that you really do want to destroy an existing file before you call *fopen*!

There are several ways to write data to a file we have just opened. In order to write binary data from the file, we can use the *fwrite* command, whose syntax is exactly

the same as *fread*. In the same way, for writing multiple elements to a file, *fwrite* can be combined with a loop.
If we want to write data in a formatted way, we can use the *fprintf function, e.g. :*

```
fprintf(fid, '%d %d %d \n', a, b, c);
```

which will write the values of *a,b,c* into the file with handle *fid*, leaving a space between them. The string *%d* specifies the precision in which the values will be written (single), while the string \n denotes the end of the line.
At the very end of the program, after all the data has been read or written, it is good practice to close a file:

```
fclose(fid);
```

f) Avoiding "Divide by zero" warnings

In order to avoid "Divide by zero" warnings you can use the *eps* function. *Eps(X)* is the positive distance from *abs(X)* to the next larger in magnitude floating point number of the same precision as X. For example if you wish to divide A by B, but B can sometimes be zero which will return *Inf* and it may cause errors in your program, then use *eps* as shown:

C = A / B; % If B is 0 then C is *Inf*
C = A / (B + eps); % Even if B is 0 then C will just take a very large value and not *Inf*.

g) Profiler / Debugging

The *profiler* helps you optimize M-files by tracking their execution time. For each function in the M-file, profile records information about execution time, number of calls, parent functions, child functions, code line hit count, and code line execution time. To open the *profiler* graphical user interface select Desktop->Profiler. So if the execution of your code is slow you can use the *profiler* to identify those lines of code that are slow to execute and improve them.

Another useful function that can be used for debugging is the *dbstop* function. It stops the execution of the program when a specific event happens. For example the commands

dbstop if error
dbstop if warning

stop execution when any M-file you subsequently run produces a run-time error/warning, putting MATLAB in debug mode, paused at the line that generated the error. See the MATLAB help for more details. Alternatively, you can use the graphical user interface to define the events that have to take place in order to stop the program. Just select Debug menu -> Stop if Errors/Warnings.

6. Assignment 2: Decision Trees

Decision trees are one of the simplest forms of learning algorithms. A decision tree takes as input an object or situation described by a set of attributes and returns a decision – the predicted value for the input. Classification is done by learning a discrete valued function (assuming discrete inputs and outputs) based on the given situation. A decision tree reaches its decision by performing a sequence of tests. Each internal node in the tree corresponds to a test of the value of one of the properties, and the branches from the node are labelled with the possible values of the test. Problems where each attribute is binary will generate binary trees, i.e. the branching factor of the tree will be 2. Each leaf node in the tree specifies the value to be returned if that leaf is reached. Decision tree representations seem to be very natural for humans, e.g. "How To" manuals are written entirely as a single decision tree stretching over hundreds of pages. A pseudo code of the decision tree learning algorithm is depicted in Table 4.

```
function DECISION-TREE-LEARNING(examples,attribs,targets) returns a decision tree
    examples: set of training examples
    attribs:  list of attributes
    targets:  target labels for the training examples
    ----------------------------------
    if all examples have the same label
    then return a leaf node with that label
    else if attribs is empty
        then return a leaf node with label = MAJORITY-VALUE(targets)
    else
        best ← CHOOSE-BEST-DECISION-ATTRIBUTE(attribs,examples,targets)
        tree ← a new decision tree with root decision attribute best
        for each possible value v_i of best do
            add a branch to tree corresponding to best = v_i
            {examples_i , targets_i}← {elements of examples with best = v_i and
                                       corresponding targets}
            if examples_i is empty
            then return a leaf node with label = MAJORITY- VALUE(targets)
            else subtree ← DECISION-TREE-LEARNING(examples_i ,attribs-{best},
                                                  targets_i)
    return tree
```

Table 4: Pseudo code for the decision tree learning algorithm.

The function MAJORITY-VALUE just returns the majority value of the example goals. The function CHOOSE-BEST-DECISION-ATTRIBUTE measures how "good" each attribute in the set is. Several methods can be applied here, e.g. the ID3 algorithm. Suppose the training set contains *p* positive and *n* negative examples. Then an estimate on the information contained in a correct answer is:

$$I\left(\frac{p}{p+n},\frac{n}{p+n}\right) = -\frac{p}{p+n}\log_2\frac{p}{p+n} - \frac{n}{p+n}\log_2\frac{n}{p+n}$$

Testing of any attribute A will divide the training set E into subsets $E_1,\ldots,E_\upsilon$ according to their values for A, where A can have υ distinct values (e.g. 2 for binary problems). Each subset E_i has p_i positive examples and n_i negative examples, so going along that branch, $I(\ p_i / (p_i+n_i),\ n_i / (p_i+n_i))$ bits of information will be needed to answer the question. So on average, after testing attribute A, we will need:

$$Remainder(A) = \sum_{i=1}^{\upsilon} \frac{p_i + n_i}{p + n} I\left(\frac{p_i}{p_i + n_i}, \frac{n_i}{p_i + n_i}\right)$$

bits of information to classify the example. The information gain from the attribute test is the difference between the original information requirement and the new requirement:

$$Gain(A) = I\left(\frac{p}{p+n}, \frac{n}{p+n}\right) - Remainder(A)$$

Finally, the attribute that is chosen is the one with the largest gain.
For more information on decision trees, see Chapter 3 of *Tom Mitchell's "Machine Learning"* book, especially table 3.1 and the relevant slides you can find here:
http://www.cs.cmu.edu/afs/cs/project/theo-20/www/mlbook/ch3.pdf

a) Implementation

The goal of the first assignment is to implement a decision tree algorithm using MATLAB, along with the accompanying functions, as described in the previous section. The inputs of the implemented algorithm should be the given set of examples, their respective target labels and the emotion label that the resulting tree will correspond to.

a1. Create data

Make sure your data (for this assignment only cleandata_students.txt will be used) is loaded in the workspace using the *loaddata* command. This function outputs an array *x*, which is an *N×45* array, where *N* is the total number of examples and 45 is the number of action units (or features/attributes) and an array *y* of dimensions *N×1*, containing the labels of the corresponding examples. These labels are numbered from 1 to 6, the same as the total number of emotions. Use 'help loaddata' to get more information about the function. In order to construct a decision tree for a specific emotion, the labels in *y* should be remapped according to that particular emotion. For example, if you train for happiness, with label 4, then the labels with that value should be set to 1 (positive examples) and all the others to 0 (negative examples).

a2. Create decision tree

The output of your algorithm should be the resulting tree for the given emotion (hence one tree pre emotion). You are expected to follow the algorithm shown in Table 4, and implement the ID3 algorithm for the attribute selection. The resulting tree must be a MATLAB structure (*struct*) with the following fields:

- *tree.op* : a label for the corresponding node (e.g. the attribute that the node is testing). It must be empty for a leaf node.
- *tree.kids* : a cell array which will contain the subtrees that initiate from the corresponding node. Since the resulting tree will be binary, the size of this cell array must be 1x2, where the entries will contain the left and right subtrees respectively. This must be empty for a leaf node since a leaf has no kids, i.e. tree.kids = [].
- *tree.class* : a label for the returning class. This field can have the following possible values:
    - *0 - 1*: the classification of the examples (*negative-positive*, respectively), if it is the same for all, or as it is defined by the MAJORITY-VALUE function (in the case *attribs* is empty) .
    - It must be empty for an internal node, since the tree returns a label only in a leaf node.

This tree structure is essential for the visualization of the resulting tree by using the ***DrawDecisionTree.m*** function, which is provided. Alternatively, a different tree structure can be chosen, provided that a visualization function will also be given.

a3. Evaluation

Now that you know the basic concepts of decision tree learning, you can use the clean dataset provided to train 6 trees, one for each emotion, and visualize them using the DrawDecisionTree function. Then, evaluate your decision trees using 10-fold cross validation. 6 trees should be created in each fold, and each example needs to be classified as one of the 6 emotions. You should expect that slightly different trees will be created per each fold, since the training data that you use each time will be slightly different. Use your resulting decision trees to classify your data in your test set. Write a function:

- y = testTrees(T, x),

which takes your trained trees (all six) T and the features x as returned by *loaddata* and produces a vector of label predictions y in the format of *loaddata*. Think how you will combine the six trees to get a single output for a given input sample.

(***Hint***: 10-fold cross validation with trees will be used again in the last assignment so you may wish to write a function that takes as inputs the input samples and the targets and returns a 6x6 confusion matrix).

b) Deliverables

For the completion of this part of the CBC, the following have to be submitted electronically via CATE:

1. The code used to create the decision trees, including the loading and transforming of the data. Save the 6 trees trained on the whole dataset with the MATLAB command *save*.

2. A report of approximately two pages (excluding result matrices and tree graphs) containing the following:
   - The acquired decision trees, trained on the whole available dataset (6 in total, for 6 different emotions).
   - Each example needs to get only a single emotion assigned to it, between 1 and 6. Explain how you made sure this is always the case in your decision tree algorithm.
   - Have you encountered any ambiguity when attempting to classify a given sample, if yes, what strategies did you use to deal with this?
   - Average cross validation classification results, that include:
     - Confusion matrix.

     (***Hint:*** you should get a single 6x6 matrix)

     (***Hint***: you will be asked to produce confusion matrices in almost all the assignments so you may wish to write a general purpose function for computing a confusion matrix)
     - Average recall and precision rates per class.

     (***Hint:*** you can derive them directly from the previously computed confusion matrix)
     - The F_1-measure derived from the recall and precision rates of the previous step.

     This implies that the students will have to write a small script that splits the given dataset into training and test sets.
   - Answer the following question in your report: Pruning is an important issue in trees. Run the *pruning_example* function, which is provided, and briefly explain how it works. It uses the MATLAB built-in functions for trees. Two figures should be generated showing two different curves. Include those figures in your report and explain why each curve has this shape. What is the difference between them?

HINT : Make sure that you save your results! You will need to use them again for the completion of Assignment 5!

c) Grading scheme

Grade = 0.5* Report content + 0.4* Code + 0.1* Report quality

Code (total : 100)

- Results : 60
- Comments : 15
- Quality : 25

Report content (total : 100)

- Implementation details : 20
- Tree figures: 10
- Results of the cross validation experiments, analysis and explanation : 10
- Confusion matrix : 10
- Recall/precision : 10

- Answer of the ambiguity question : 20
- Answer of the pruning question : 20

Report quality (total : 100)

- Quality of presentation.

7. Assignment 3: Artificial Neural Networks

An Artificial Neural Network (ANN) or simply Neural Network is a network of interconnected neurons (see fig. 6) that is able to learn non-linear real-valued, discrete-valued or vector-valued functions from examples. They are inspired by the observation that biological learning systems are composed of very complex webs of neurons. Every neuron has a number of real-valued inputs, which can be the outputs of other neurons, and uses a mathematical function called the transfer function to compute one real-valued output. There are three types of neurons which are most commonly used: the *perceptron*, which takes a number of real valued inputs and computes a binary output as a linear combination of the inputs, the *linear unit* which takes a number of real valued inputs and computes a continuous output as a linear combination of the inputs, and the *sigmoid unit* which takes a number of real valued inputs and computes a continuous output as a non-linear, sigmoid function of the inputs.
The way the various neurons are connected is called the network topology. The neurons are interconnected with each other by synapses, the weights of which define the output of the neuron. In the training phase, it is these weights that need to be learned. Together with the topology and the type of neurons the weights completely define the ANN.
The procedure used to update the weights in order to minimize the mistakes made by the classifier is called the training rule, or learning law. In Tom Mitchell's "Machine Learning" book the *backpropagation* method is described. In this CBC we will experiment with different versions of this training rule.

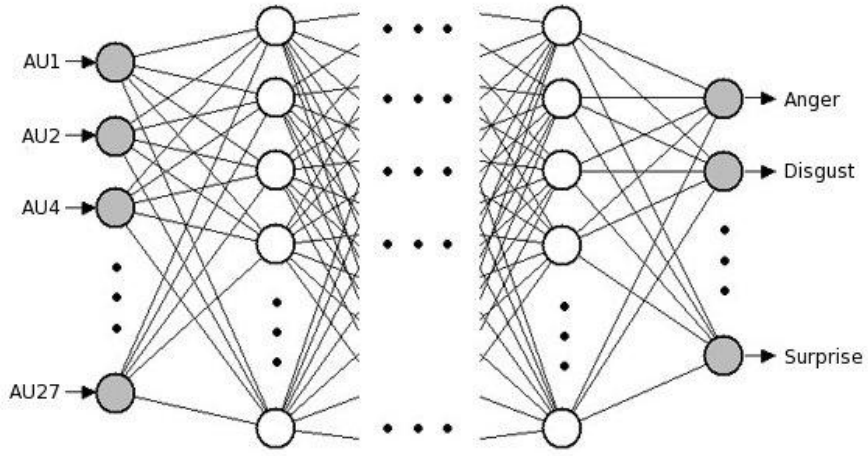

Figure 6.Artificial Neural Network with Action Units as inputs and emotions as outputs

a) Implementation

To complete assignment 2, you will work with the MATLAB Neural Networks toolbox. The *MATLAB version* that you should use for this assignment is 64-bit MATLAB R2011b installed under 64-bit Ubuntu Linux 11.04 on the LAB machines**.**
The command *nntoo*l will open a dialog box where you can define the topology of

your network, the data you want to use, the transfer function of the neurons, the learning law, and the function you want to use to measure the performance of the network. The transfer function can be specified per hidden layer, while the learning law and the performance measure are for the entire network. For the cross-validation evaluation of the ANNs you cannot use this GUI however and you should write your own cross-validation script using the toolbox's command line functions. It is recommended that you read through the MATLAB help section on Neural Network Design before starting with this assignment. For this assignment we will be building feed-forward backpropagation neural networks. **Make sure that in all your experiments you use the version of the MATLAB that is installed on the LAB machines.**

a1. Create data

Make sure your data is loaded in the workspace using the *loaddata* command. For the Neural Networks implementation of MATLAB, the data (i.e. AUs) should be arranged such that every column represents one sample. The emotion labels should be arranged in a matrix so that again each column represents the label of one sample. Every row of this matrix corresponds to the target values of the network's output nodes. Since every sample can have only one emotion label, only one element per column can have a value '1'. The row number that corresponds to the active emotion label for that sample should contain a value of '1', all other rows of that column should contain the value '0'. So a sample with the label 'happiness', a value of 4 in the data returned by *loaddata*, would be represented by the vector $[0\ 0\ 0\ 1\ 0\ 0]^T$. This data transformation described above is performed by the provided function *ANNdata.m*:

```
[x, y] = loaddata('cleandata_students.txt');
[x2, y2] = ANNdata(x, y);
```

a2. Create network using nntool

To get a basic understanding of the neural networks, you should first play with the MATLAB Graphical User Interface (GUI) for NNs. A number of examples for using the GUI for NNs are provided below. Make sure the data is in the correct format, as specified above. Next, run the *nnstart* function. This will open a GUI that provides links to new and existing Neural Network Toolbox™ GUIs and other resources (see Figure 7).

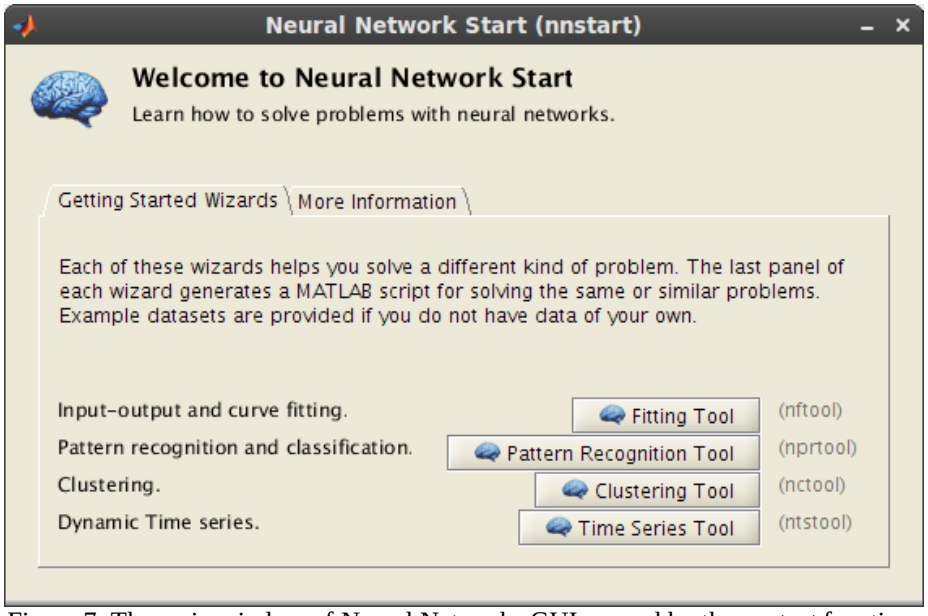

Figure 7: The main window of Neural Networks GUI opened by the *nnstart* function.

There is an excellent demo on MATLAB's website on getting started with Neural Networks. See http://www.mathworks.com/products/neuralnet/description1.html.

Based on the demo, import the training data as inputs and as targets. Make sure you import the AU data as 'inputs' and the emotion labels as 'targets'. Next create a new network. Please note that the last layer you define is the output layer and as such should have the same number of neurons as the number of categories in which you want to classify your data. Choose a training function using the NN training tool (*nntraintool*). Be aware that some training functions are very slow, so don't start training with too many epochs (the default 100 should be fine). When you feel confident that the NN is learning fast enough, you can increase the number of epochs if you wish. Note the training error. Toy with some of the training parameters. What are the influences of the learning rate and of the number of epochs used? Keep in mind that these questions must be addressed in your report.

a3. Evaluation

Now that you have a basic understanding of Neural Networks (NNs), you will have to train two types of neural networks: (1) single six-output neural network, and (2) six single-output neural networks, where each output should represent one of the six emotions to be classified. For the training, use the clean dataset and the command line functions. To create the new networks, use the function 'feedforwardnet'. Read the help pages of feedforwardnet, network/configure, network/train and network/sim. Note that you may have to set the trainParam.show, trainParam.epochs, trainParam.goal, trainParam.mu, trainParam.lr, and trainParam.goal values of your network prior to training. As the choice of the network's topology and its parameters influencing the classification performance and training time, we suggest that you experiment with the following parameters of the network: (a) number of hidden layers, (b) number of neurons in hidden layers, (c) learning rate, (d) different activation / transfer function, (e) different training functions (only ones that work with batch training).

- `[net] = feedforwardnet([S1, S2...SNl], trainFcn)`
  Si - Size of ith hidden layer, for Nl layers (not including input and output layers).
  trainFcn - Training function, default = 'trainlm' (Levenberg-Marquardt backpropagation). Note that you may change various fields of the returned neural network descriptor to customise the network's properties.

- `[net] = configure(net, x, y)`
  Configures the input and output layers of the neural network with respect to the given set of inputs (x) and targets (y).

- `[net] = train(net, x, y)`
  Trains a network using input data (x) and targets (y).

- `[t] = sim(net, x)`
  Simulates a network in feed-forward. In other words, it gives a prediction of labels (t) given a set of inputs (x).

A simple example on how to use the command line for experimenting with NNs is provided below.

Let `P` represent the training input and `T` represent the targets:

```
P = 0:0.01:1;
T = sin(-pi/2 + P * 3 * pi);
```

To create a network with one hidden layer of five neurons the following command are executed:

```
net = feedforwardnet(5);
net = configure(net, P, T);
```

To train the network for 100 epochs, and plot the output the following commands are executed:

```
net.trainParam.epochs = 100;
net = train(net, P, T);
Y = sim(net, P);
plot(P, T, P, Y, 'r.');
```

The network's output is plotted in Figure 8.

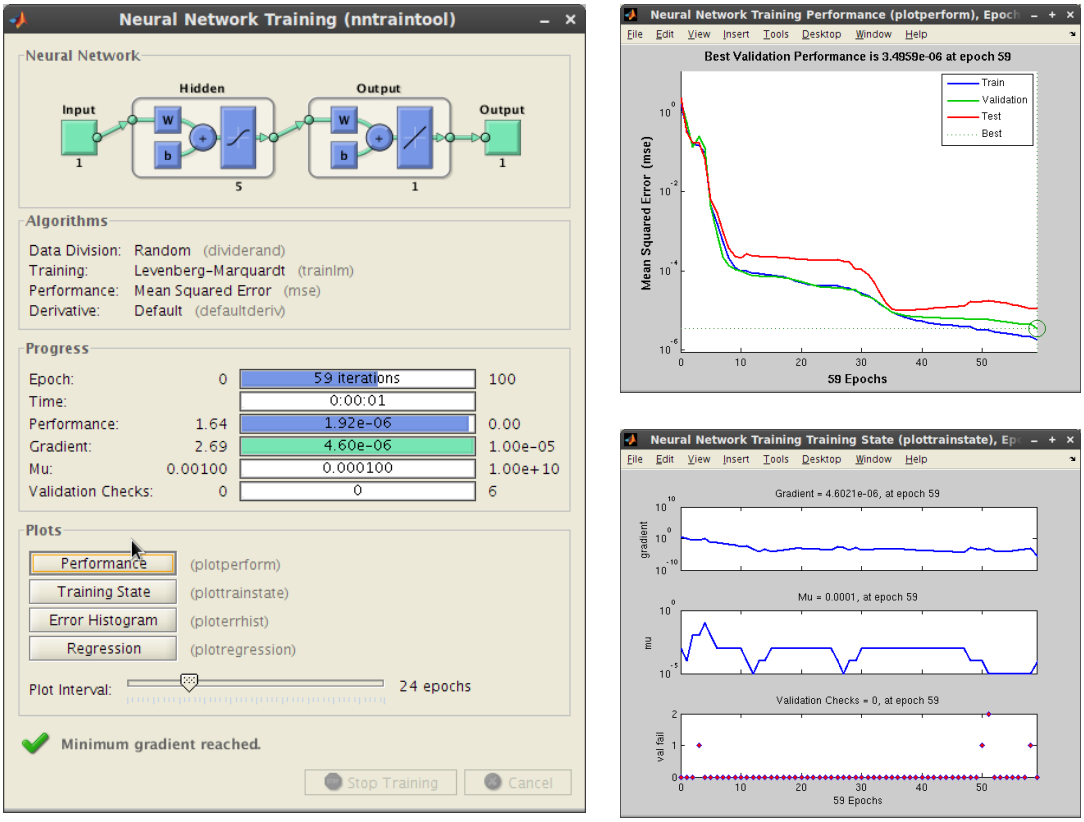

Figure 8: An example NN with 1 hidden layer of 5 neurons created using the function feedforwardnet.

Now getting back to the assignment, in order to transform the neural network output into the format used throughout this CBC, use the function '*NNout2labels*' provided to you. More specifically,

- `[y] = NNout2labels(t)`
  where *t* is the output of a feed-forwarded Neural Network and *y* is the corresponding output in the format used throughout this CBC (i.e., 1 = anger, 2 = disgust etc.).

Write a function:

- y = testANN(net, x),

which takes your trained network *net* and the features x as returned by *loaddata* and produces a vector of label predictions y in the format of *loaddata*..

b) Deliverables

For the completion of this part of the CBC, the following have to be submitted electronically via CATE:

1. MATLAB material, containing:
   - The resulting *optimal* six-output NN and six single-output NNs created by training on the whole dataset provided to you. Save the trained network with the command 'save'.

- The code used to train the network, including the loading and transforming of the data, initialisation of the NNs and setting of network parameters.

2. Report of approximately two pages (excluding result matrices) containing:

(a) Discussion of the network parameters (e.g. learning rate, epochs, topology, etc.). What criteria have you used to choose *optimal* topology/parameters of the networks? Compare the *optimal* parameters of both types of the networks (the single six-output network and six single-outputs networks). Explain what strategy you employed to ensure good generalisation ability of the networks and overcome the problem of *overfitting*, if encountered (support this by experimental results).

(b) Perform 10-fold cross-validation for both types of networks. Cross-validation should be performed in the same way as in Assignment 2 (with a script that splits the given dataset into training and test sets). 10-fold cross-validation should be performed using the optimal topology and best parameters obtained in 2(a), i.e., for six-output and single-output NNs. Note that in the case of 6 networks, each example must be classified as one of the 6 emotions. Plot the performance (only F1 measure) per fold of each network type in the same figure.

(c) Report also the average results of the 10-fold cross-validation (similarly to the trees) for both types of networks (single six-output NN and six single-outputs NNs):
   - confusion matrices,
   - recall and precision rates per class,

   (***Hint:*** you can derive them directly from the previously computed confusion matrix)
   - the F_1-measure derived from the recall and precision rates of the previous step.

(d) Is there any difference in the classification performance of the two different classification approaches. Discuss the advantages / disadvantages of using 6 single-outut NNs vs. 1 six-output NNs.

HINT : Make sure that you save your results! You will need to use them again for the completion of Assignment 5!

c) Grading scheme

Grade = 0.5* Report content + 0.4* Code + 0.1* Report quality

Code (total : 100)
- Results : 60
- Comments : 15
- Quality : 25

Report content (total : 100)
- Implementation details: 20
- 2a. (learning rate, epochs, topology, learning law, and overfitting) : 40

- 2b. (cross validation figure) : 10
- 2c. (average results) : 10
- 2d (discussion of classification approaches): 20

Report quality (total : 100)

- Quality of presentation

8. Assignment 4: Case Based Reasoning

Case Based Reasoning (CBR) is an example-based lazy-learning machine learning technique that solves new problems by adapting previously successful solutions to similar problem. A CBR system learns by storing problems and corresponding suitable solutions. When a novel problem, i.e., a new AU vector, is presented to the CBR the system suggests a solution that worked well for a similar problem in the past. Fig. 9 shows the 4 steps that a CBR system uses:

RETRIEVE the most similar case(s)
REUSE the case(s) to attempt to solve the current problem
REVISE the proposed solution if necessary
RETAIN the new solution as a part of a new case

Please note that REVISE will not be used in this exercise.

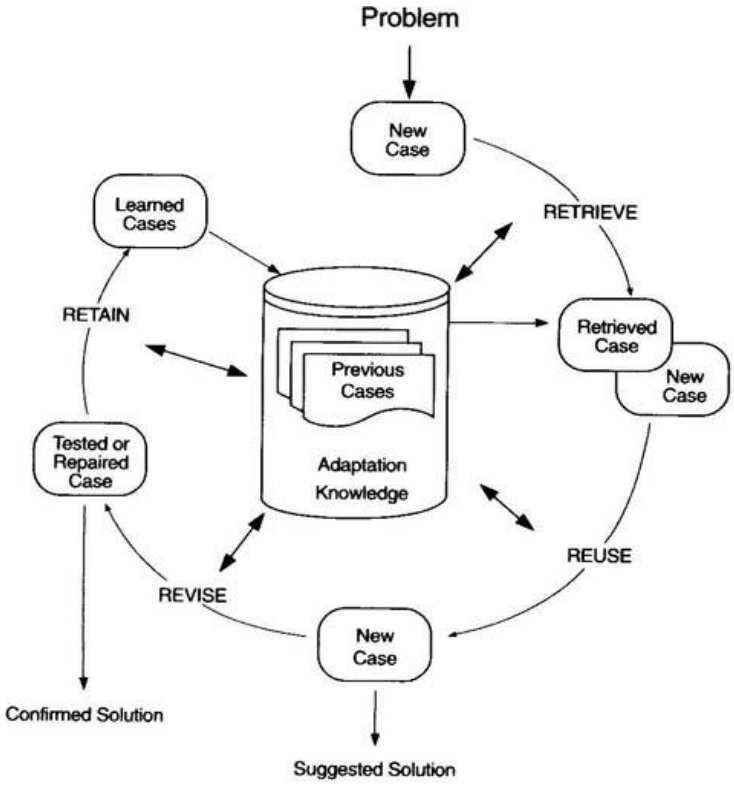

Figure 9. The cycle of a Case Based Reasoning system

a) Implementation

To complete assignment 3, you need to implement a CBR system, i.e., to build the various parts of a case based reasoning system step by step. Implement cases and the CBR system as two separate MATLAB structures.

a1. Implement cases

Implement a case structure. Select carefully the fields you will use (e.g., typicality, solution, problem description). Typicality can, for instance, be the number of times the same case is found during training. Implement a function that takes a vector of AUs and one scalar label as input and returns a case. The input AU vectors should contain the AUs which are active, i.e. if a binary vector returned by *loaddata* is nonzero at position 9 and 20 then the input AU vector should be [9 20]. The scalar label must be a number in the range 1:6, similar to the elements of the label data returned by the function *loaddata*. It must be possible to construct a case without a solution being associated to it. This is of importance when implementing the REUSE function.

Implement a function that computes the similarity of two cases. One example of a similarity measure would be the difference in vector lengths of the problem descriptions of two cases, i.e., AU vectors. However, there exist much better measures. Students are advised to consult the papers uploaded on the Machine Learning Website.

a2. Implement CBR system

Implement a CBR system. Take care how you will initialize it and how cases will be stored in it. For instance, think of the following: what will happen if you try to add a case to the CBR system that is already known to the system? Implement a function/method to create a CBR system initialized with a large number of cases in one go (useful for the cross-validation assignment). Call this function *CBRinit*:

- `[cbr] = CBRinit(x, y)`

where *x* is an [n x 45] matrix of AU examples which has the same format as the matrix of AUs returned by the function *loaddata*, and *y* is a vector of length *n* of labels using the same format as the labels returned by *loaddata.* Convert the input vector *x* to the appropriate format as described above inside this function.

Implement the functions RETRIEVE, REUSE and RETAIN as follows:

- `[case] = retrieve(cbr, newcase),`
  where *cbr* is a non-empty CBR system, *newcase* a novel case without a solution associated to it, and *case* is the case that matches best with *newcase* according to the implemented similarity measure.

- `[solvedcase] = reuse(case, newcase)`
  where *case* is the best-matching case previously returned by RETRIEVE and *newcase* is the novel case presented to the CBR system. REUSE attaches the solution of *case* to *newcase*, resulting in *solvedcase*.

- `[cbr] = retain(cbr, solvedcase)`
  updates the CBR system by storing the solved case *solvedcase.*
  (***Hint:*** Think what will happen if you try to add a case to the CBR system that is already known to the system.

Initialize the CBR with a small number of the training samples from the AUs and emotions data provided to you and toy with the system. Provide it with some new samples and check the solution that the CBR suggests. Use the RETAIN function to store the provided case-solution pair after you have checked it.

a3. Evaluation

To evaluate the performance of your CBR system in terms of classification accuracy, use 10-fold cross validation, initializing the CBR with 90% of the data for each fold and check how well the suggested solutions for the test samples agree with the emotion labels associated with those samples. Keep track of the classification scores per fold, just recall and precision, and the average score over all folds, just recall, precision and confusion matrix. What happens if the same problem (set of AUs) has two different solutions (emotion labels)? Answer these questions in your report.
Write a function:

- y = testCBR(CBR, x),
  which takes your trained CBR system and the features x as returned by *loaddata* and produces a vector of label predictions y in the format of *loaddata*..

(***Hint***: 10-fold cross validation using a CBR system will be used again in the last assignment so you may wish to write a function that takes as inputs the input samples and the targets and returns a 6x6 confusion matrix).

b) Deliverables

For the completion of this part of the CBC, the following have to be submitted electronically via CATE:

1. MATLAB material:
   - The resulting Case Based Reasoning system including all functions. The CBR should be initialised using the whole dataset provided to you. Save the trained CBR with the MATLAB command 'save'.
   - All code used to create and validate your CBR system.

2. Report of approximately two pages (excluding result matrices) containing:
   - Explanation of the implementation details of both the case and the CBR structures. Report on the difficulties encountered. Explain how the functions RETRIEVE, REUSE and RETAIN work in your implementation.
   - Explain why your similarity measure works best. What different similarity measures have you compared (compare at least three different measures).
   - Average cross validation classification results (This implies that you will have to write a small script that splits the given dataset into training and test sets), that include:

- Confusion matrix.

(***Hint:*** you should get a single 6x6 matrix, including only the true positives and the false positives, that is, ignoring false negative classifications)

- Average recall and precision rates per class.

(***Hint:*** you can derive them directly from the previously computed confusion matrix)

- The F_1-measure derived from the recall and precision rates of the previous step.

NOTE: You are free to choose how to implement the CBR system in terms of the structure employed. A simple system would impose no hierarchy amongst the cases and apply 1-NN to retrieve similar cases based on the chosen similarity measure. A more complex system would impose a clustered hierarchy and/or apply k-Nearest Neighbor (k-NN) where k > 1. In the report you should also discuss any problems or issues you run into, e.g. what happens when there is a tie when retrieving similar cases from the system?

HINT : Make sure that you save your results! You will need to use them again for the completion of Assignment 5!

c) Grading scheme

Grade = 0.5* Report content + 0.4* Code + 0.1* Report quality

Code (total : 100)

- Results : 60
- Comments : 15
- Quality : 25

Report content (total : 100)

- Implementation details of *Case* : 10
- Implementation details of *Retrieve* : 5
- Implementation details of *Reuse* : 5
- Implementation details of *Retain* : 5
- Distance measures – motivation : 20
- Distance measures – explanation : 10
- Results (cross validation, comparison of distance measures, precision, recall etc.) : 45

Report quality (total : 100)

- Quality of presentation.

9. Assignment 5: *T-test* and *ANOVA* test

a) T-test

The t-test assesses whether the means of two distributions are *statistically* different from each other. Consider the three situations shown in Fig. 10. The first thing to notice about the three situations is that the difference between the means is the same in all three. But, you should also notice that the three situations don't look the same. The left-most example shows a case with low variability. The centre situation shows a case with moderate variability within each group, while the right-most example shows the high variability case. Clearly, we would conclude that the two groups appear most different or distinct in the low-variability case. This is because there is relatively little overlap between the two bell-shaped curves. In the high variability case, the group difference appears least striking because the two bell-shaped distributions overlap so much. This leads us to a very important conclusion: when we are looking at the differences between scores for two groups, we have to judge the difference between their means relative to the spread or variability of their scores. The t-test does just this.

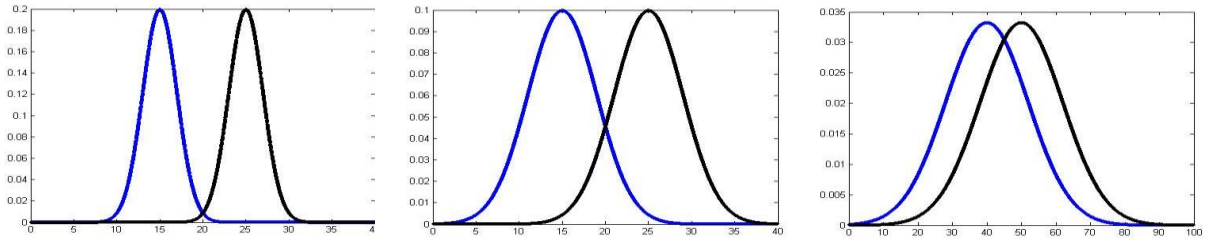

Fig.10: Three scenarios for differences between means

The formula for the t-test is a ratio. The top part of the ratio is just the difference between the two means or averages. The bottom part is a measure of the variability or dispersion of the scores:

$$t = \frac{\bar{x}_T - \bar{x}_C}{SE(\bar{x}_T - \bar{x}_C)},$$

Where $\bar{x}_T$, $\bar{x}_C$ are the means of the corresponding groups. The denominator of the above equation is called *Standard Error of Difference* and is given by:

$$SE(\bar{x}_T - \bar{x}_C) = \sqrt{\frac{\mathrm{var}_T}{n_T} + \frac{\mathrm{var}_C}{n_C}},$$

Where var_T, var_C are the variances of the two groups and n_T, n_C are the sample sizes of the groups respectively. The t-value will be positive if the first mean is larger than the second and negative if it is smaller.

Once the t-value is computed, a table of significance has to be used in order to determine if the ratio is large enough to say that the difference between the groups is not likely to have been a chance finding. An example of such a table is given in Fig. 11.

df	α 0.20	0.10	0.05	0.02	0.01	0.005	0.002	0.001
1	3.078	6.314	12.706	31.820	63.657	127.321	318.309	636.619
2	1.886	2.920	4.303	6.965	9.925	14.089	22.327	31.599
3	1.638	2.353	3.182	4.541	5.841	7.453	10.215	12.924
4	1.533	2.132	2.776	3.747	4.604	5.598	7.173	8.610
5	1.476	2.015	2.571	3.365	4.032	4.773	5.893	6.869
6	1.440	1.943	2.447	3.143	3.707	4.317	5.208	5.959
7	1.415	1.895	2.365	2.998	3.499	4.029	4.785	5.408
8	1.397	1.860	2.306	2.897	3.355	3.833	4.501	5.041
9	1.383	1.833	2.262	2.821	3.250	3.690	4.297	4.781
10	1.372	1.812	2.228	2.764	3.169	3.581	4.144	4.587
11	1.363	1.796	2.201	2.718	3.106	3.497	4.025	4.437
12	1.356	1.782	2.179	2.681	3.055	3.428	3.930	4.318
13	1.350	1.771	2.160	2.650	3.012	3.372	3.852	4.221
14	1.345	1.761	2.145	2.625	2.977	3.326	3.787	4.140
15	1.341	1.753	2.131	2.602	2.947	3.286	3.733	4.073
16	1.337	1.746	2.120	2.584	2.921	3.252	3.686	4.015
17	1.333	1.740	2.110	2.567	2.898	3.222	3.646	3.965
18	1.330	1.734	2.101	2.552	2.878	3.197	3.610	3.922
19	1.328	1.729	2.093	2.539	2.861	3.174	3.579	3.883
20	1.325	1.725	2.086	2.528	2.845	3.153	3.552	3.850

Fig.11: Sample of significance table used for the *t-test.*

To test the significance, you need to set a risk or significance level α. This means that α times out of a hundred you would find a statistically significant difference between the means even if there was none (i.e., by "chance"). A typical value is 0.05. In addition, you need to determine the degrees of freedom (df) for the test. In the *t-test*, the degrees of freedom are the sum of the samples in both groups minus 2. If the calculated *t* value is above the threshold chosen for statistical significance (taken from the table), then the null hypothesis that the two groups come from distributions with equal means and equal but unknown variances is rejected in favour of the alternative hypothesis, which typically states that the means of the groups differ.

b) Analysis of Variance Test (ANOVA)

An *ANOVA* (Analysis of Variance) test, sometimes called an *F* test, is closely related to the *t* test. The major difference is that, where the *t* test measures the difference between the means of two groups, an *ANOVA* tests the difference between the means of two or more groups. Similarly to the *t* test, the *ANOVA* test is a test on the null

hypothesis H_o that the means of the distributions are the same, against the alternative hypothesis H_a, that is, at least two means are unequal.
For the analysis of the *ANOVA* test, let us use the following notation:

- g is the number of groups we want to compare.
- $\mu_1, \mu_2, \ldots, \mu_g$ are the means of the distributions we want to compare.
- $n_1, n_2, \ldots, n_g$ are the sample sizes.
- $\bar{Y}_1, \bar{Y}_2, \ldots, \bar{Y}_g$ are the sample means.
- $\sigma_1, \sigma_2, \ldots, \sigma_g$ are the sample standard deviations.

The basic principle of *ANOVA* is to compute two different estimates of the population variance:

- The within groups estimate pools together the sums of squares of the observations about their means:

$$WSS = \sum_{i=1}^{g} \sum_{j=1}^{n_i} (Y_{ij} - \bar{Y}_i)^2$$

- The between groups estimate, calculated with reference to the grand mean, that is, the mean of all the observations pooled together:

$$BSS = \sum_{i=1}^{g} n_i (\bar{Y}_i - \bar{Y})$$

  The between groups estimate will be a good estimate of the sample variance if and only if the null hypothesis is true, since only then the grand mean be a good estimate of the mean of each group.

The *ANOVA* F test statistic is the ratio of the between estimate and the within estimate:

$$F = \frac{\text{Between estimate}}{\text{Within estimate}} = \frac{BSS/(g-1)}{WSS/(N-g)},$$

When the null hypothesis is false, the between estimate tends to overestimate the population variance, so it tends to be larger than the within estimate. Then, the F test statistic tends to be considerably larger than 1. N is the sum of all sample sizes and $df_1 = g - 1$, $df_2 = N - g$ are the degrees of freedom of an F probability distribution, as shown in Fig.12.

Similar to the t test, in order to determine if the null hypothesis is true, a significance level α has to be determined. The significance level corresponds to the shaded area

of Fig. 12, and defines a threshold on the F value, called critical value F_{CRIT}. If the computed F value is larger than F_{CRIT}, then there is evidence that at least one of the examined distributions is different than the rest.

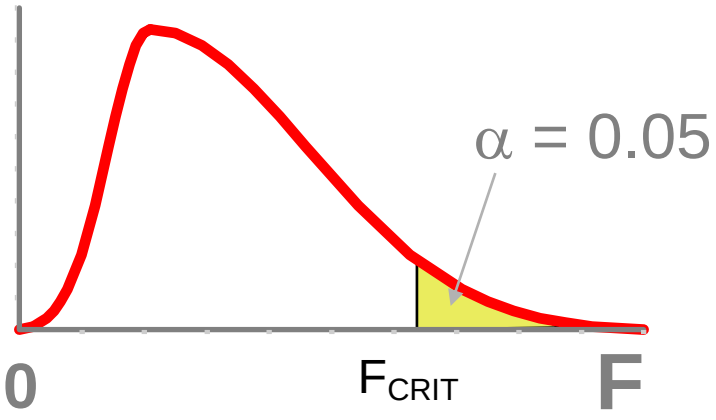

Fig.12: The F probability distribution

c) Multiple Comparisons

The ANOVA test determines whether there exists a significant difference among the means of different groups. In this sense it is a preliminary test that informs us if we should continue the investigation of the data at hand. If the null hypothesis (no difference among the group means) is accepted, then there is not much we can learn, and we are finished with the analysis. However, if the null hypothesis is rejected, we usually want to undertake a thorough analysis in order to determine which groups are statistically significant to each other.

An obvious (but naive) solution to this problem would be to apply a series of t-tests between every possible pair of the available groups. Formally, given K groups, one should perform $K(K-1)/2$ different t-tests. Following this approach, however, poses a significant problem. When you perform a simple t-test of one group mean against another, you specify a significance level that determines the cutoff value of the t statistic. For example, you can specify the value alpha = 0.05 to insure that when there is no real difference, you will incorrectly find a significant difference no more than 5% of the time. When there are many group means, there are also many pairs to compare. If you applied an ordinary t-test in this situation, the alpha value would apply to each comparison, so the chance of incorrectly finding a significant difference would increase with the number of comparisons.

Multiple comparison procedures are designed to address this problem, by providing an upper bound on the probability that any comparison will be incorrectly found significant. In essence, multiple comparison procedures compensate for the number of comparisons by modifying the specified significance level. A typical example is the Bonferroni correction, which states that if the desired significance level for the whole

family of tests is (at most) α, then one should test each of the individual tests at a significance level of α/n.

d) Implementation

For this assignment you do not have to implement the algorithms for the *t*, *ANOVA* and multiple comparison tests by yourselves. MATLAB provides suitable functions for these tests that are easy and convenient to use. More specifically:

d1. T-test

```
H = TTEST2(X,Y,ALPHA)
```

- Performs a T-test of the hypothesis that two independent samples, in the vectors X and Y, come from distributions with equal means, and returns the result of the test in H.
- H==0 indicates that the null hypothesis ("means are equal") cannot be rejected at the α % significance level.
- H==1 indicates that the null hypothesis can be rejected at the α % level. The data are assumed to come from normal distributions with unknown, but equal, variances. X and Y can have different lengths.

d2. *ANOVA* test

```
[P, ANOVATAB,STATS] = ANOVA1(X)
```

- Performs an *ANOVA* test for comparing the means of two or more groups of data. It returns the following:
    - a P-value for the null hypothesis that the means of the groups are equal. The P-value is essentially a significance level. That is, if you define a significance level α and the returned P-value is smaller than α, then you can reject the null hypothesis that all data come from populations with the same mean.
    - the ANOVA table values as the cell array ANOVATAB.
    - A STATS structure of statistics useful for performing a multiple comparison of means.
- If X is a matrix, ANOVA1 treats each column as a separate group, and determines whether the population means of the columns are equal.

d3. Multiple Comparison test

```
C = MULTCOMPARE(STATS, PARAM1, VAL1, PARAM2, VAL2,…)
```

- performs a multiple comparison test using the information in the STATS structure (returned by ANOVA1, see **d2**), and returns a matrix C of pairwise comparison results. The latter contains the results of the test in the form of a five-column matrix. Each row of the matrix represents one test, and there is one row for each pair of groups. The entries in the row indicate the means being compared, the estimated difference in means, and a confidence interval for the difference. For example, suppose one row contains the following entries:

  2.0000 5.0000 1.9442 8.2206 14.4971

  These numbers indicate that the mean of group 2 minus the mean of group 5 is estimated to be 8.2206, and a 95% confidence interval for the true difference of the means is [1.9442, 14.4971]. In this example the confidence interval does not contain 0.0, so the difference is significant at the 0.05 level. If the confidence interval did contain 0.0, the difference would not be significant at the 0.05 level.

- PARAM1, VAL1, PARAM2, VAL2..: Parameters used by the function and respective parameter values. Typical parameters are (see MATLAB help for more details):
  - 'alpha': Scalar between 0 and 1 that specifies the significance level (default is 0.05).
  - 'display': Either 'on' (the default) to display a graph of the estimates with comparison intervals around them, or 'off' to omit the graph.
  - 'ctype': Specifies the method used for the multiple comparison, e.g. 'Bonferroni'.

d) Deliverables

To complete this assignment, the following actions should be performed:

- Train your 3 algorithms using the whole *clean* data, test using the *noisy* data and report the classification results, just F_1 measures for each emotion. Which algorithm performed better overall in terms of the F_1 measure, i.e has the highest average F_1 measure?
- Can we claim that this algorithm is a better learning algorithm than the others in general? Yes or No? And why?
- Perform 10-fold cross validation on the *clean* data using your algorithms (Use the results obtained from the previous assignments). For each algorithm calculate the F_1 measure per fold. So you will end up with a sample of 10 values per algorithm per emotion. Use the ANOVA test in order to determine if there is significant difference between the algorithms. If there is such a case, use the multiple comparison test in order to determine which algorithms are statistically different, and compare the outcome with the one you get if you use the t-test only.
  (***Hint:*** You should perform 18 *t* tests in total (3 classifiers per emotion, 6 emotions), 6 *ANOVA* tests (one per emotion) and 6 multiple comparison tests (one per emotion)).

- Perform 10-fold cross validation on the *noisy* data using your algorithms. Similarly to above, calculate the F_1 measures per fold. Use the ANOVA test in order to determine if there is significant difference between the algorithms. If there is such a case, use the multiple comparison test in order to determine which algorithms are statistically different, and compare the outcome with the one you get if you use the t-test only.
- Provide comments as to why some of the examined algorithms performed better than the others.
- Suppose that we want to add some new classes of new emotions to the existing dataset. Which of the examined algorithms are more suitable for incorporating the new classes in terms of extra engineering effort? Which algorithms need to undergo radical changes in order to include new classes?
- What assumptions do the t-test and ANOVA test make concerning the variances of the compared groups? Can you think of other procedures in case some of these assumptions do not hold?

Deliver your results in a report of approximately three pages via CATE.

c) Grading scheme

Grade = 0.8* Report content + 0.2* Report quality

Code will not be graded for this assignment.

Report (total: 100)

- Experimental results using clean and noisy data : 10
- T-test using clean data : 5
- Anova test using clean data : 5
- Multiple comparisons test using clean data : 5
- T-test using noisy data : 10
- Anova test using noisy data : 10
- Multiple comparisons test using noisy data : 10
- Better algorithm question : 10
- Discussion of results : 15
- Discussion of adding classes : 10
- Discussion of assumptions: 10

Report quality (total: 100)

- Quality of presentation.