# Assignment 2 - Happy, Sad, Mischievous or Mad?

Most of us do not wear poker faces and through looking at us one may get a basic read on our emotional state. Let us try to give a computer this capability. As any cartoonist knows, the angle of the eyebrows and curve of the lips may indicate emotional state. Figure 1 shows four basic emotional states: happy, sad, mischievous and mad.

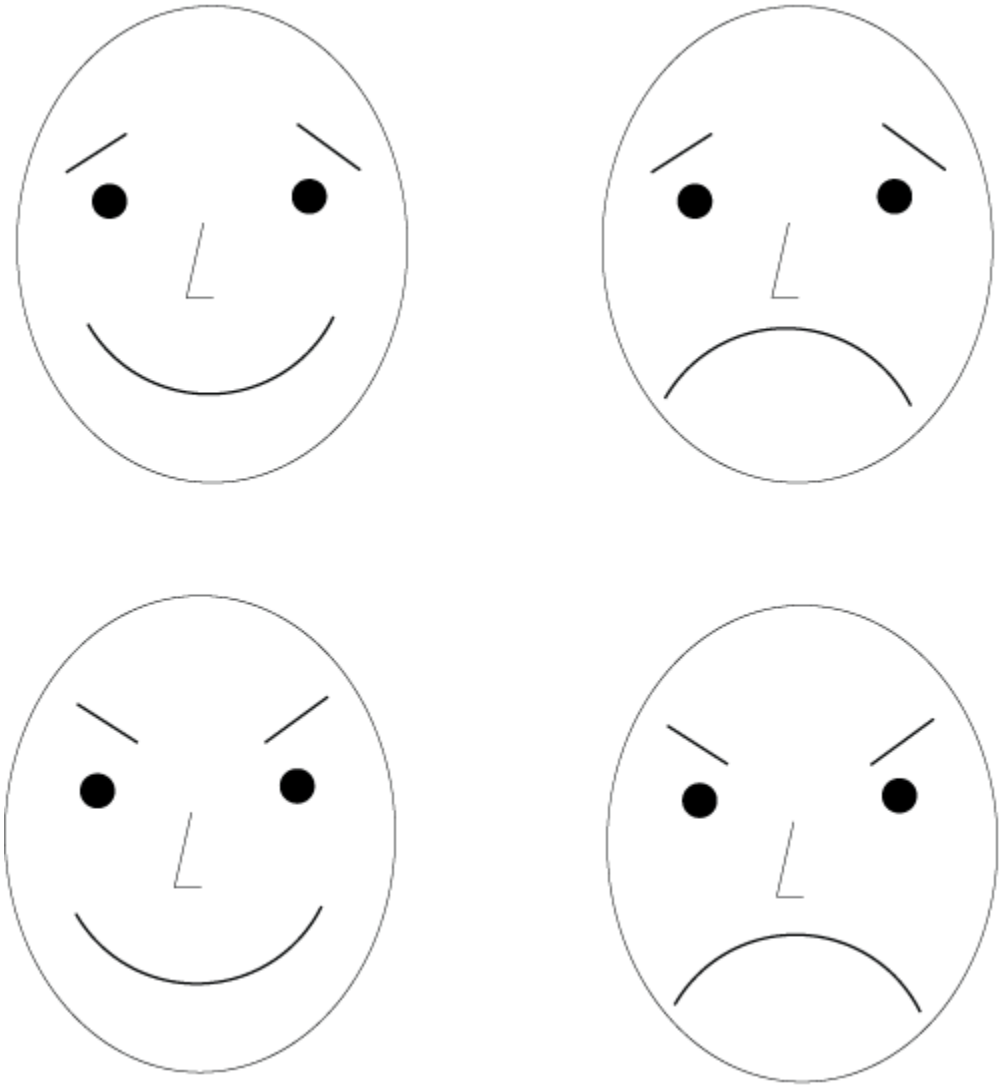

Figure 1: Four basic emotions: Happy (upper left), Sad (upper right), Mischievous (lower left), Mad (lower right)

## Task

In this assignment you are to build a perceptron based classification system that guesses the emotional state of faces presented as input. The inputs are 20 x 20 pixel maps with 32 grey levels from white to black. You may work in groups of two students.

The assignment should be solved in groups of two people. You are given a file with 300 images to train your program on. Your program should divide these files into a set for training and another set for testing the performance of the training (see the lecture slides). Images are stored in an ascii-based file format. Your system must be able to read file of this format, and must write to standard output the classification values for each image id, as described in the Facit-file section of the face-file format.

The training images are stored in training.txt and the correct classification on each image is given in training-facit.txt. See the face-file format spec for a graphic version of all images.

Note that your program must be able to handle comment lines and blank lines automatically. You may i.e. not assume some fixed number of comment lines in the beginning of the file. No comments or blank lines are embedded in the actual pixel data.

Your solution will be tested on a set of images, to see how accurately your perceptron classifies unknown faces. This set was randomly drawn from the same population as the original 300 image set. In order to pass the assignment your solution must correctly classify at least 65% of the new test images.

When your program is tested it will be trained on a set of images/facits which contains an unknown number of images, and tested against another set, also with an unknown number of images. It is therefore important that your program can adapt to a varying number of images. It is also important that you divide the given file into the training and test sets with a fixed proportion, not a fixed number of images. Part of the challenge is to decide how large part of the images are to be used for traing and for performance evaluation.

Example:

You are given a file with 300 images and a facit file with the same number of images. You decide to use 2/3 of the images for training and the remainder for performance testing. Your program uses an algorithm that trains the network until some performance criteria is fulfilled.

Whey you have submitted your solution your program will be run with a file with an unknown number of images. This means that your program will use 2/3 of these for training and the rest for performance evaluation. After the network is trained it will be tested on a new set of images. It is this test that you must pass with 65 % correct classification.

**NOTE: Your solution must follow the execution and output specifications given below. Part of this assignment is to be able to write a program that follows given specifications. A solution that does the right classifications but fails to comply with these specifications will not pass.**

**Under course material -> ANN-Assignment2/ in Cambro you will find a zip archive named FaceTest.zip. This contains instructions on how to run your program to make sure it is compliant with the test procedure used when grading your submission.**

# Implementation

The solution may be implemented in any programming language available in the CS Linux environment, including any machine in the Linux laboratories. You are of course free to develop your program on any operating system, as long as you ensure that it can be compiled and executed on our Linux system. See the support pages for an introduction to the Linux/Unix environment. You may use third-party software for components in your program, such as data pre-processing. However, third-party software may not be used for the perceptron and the learning mechanism implementation, you must implement these parts yourself.

**Please note** that the resulting classification should be written to standard output (the terminal window), and that the ordering of the result is important - the resulting classification values must be in the same order as the images in the test-file.

**In order to facilitate grading, your program must be able to execute in one of the following ways:**

## Python

If your solution is implemented in Python, it should be able to execute using the following schell command, assuming that all source files are located in the current folder:

```
python faces.py training-file.txt training-facit.txt test-file.txt
```

This program should then train on the images in training file using correct classifications from training-facit.txt, and then run a classification on the images in test-file.txt.

Note that the sizes of all files are unknown, your program must adapt to a varying number of images. There is always an equal number of images and facits.

### Java

If your solution is implemented in Java, it should be able to compile and execute using the following schell commands, assuming that all source files are located in the current folder:

```
javac *.java
```

```
java Faces training-file.txt training-facit.txt test-file.txt
```

This program should then train on the images in training file using correct classifications from training-facit.txt, and then run a classification on the images in test-file.txt.

### Other languages

If you've not implemented your solution in Python or Java, we ask that your program can be compiled and executed by calling two bash scripts, in the following way:

```
bash compile.sh
```

```
bash faces.sh training-file.txt training-facit.txt test-file.txt
```

faces.sh should then train on the images in training file using correct classifications from training-facit.txt, and then run a classification on the images in test-file.txt. You can achieve this by creating a schell script. See this tutorial for more information.

### Program output

When executing your solution as described above, the program should print text to standard out that are completely compliant with the Facit-file section of the face-file format. This means that every line that is not part of the classification result should be marked as a comment. This is important since your solution will be automatically executed and the result parsed as if it was a Facit-file.

## What to hand in

You should hand in a complete and well-written report (in pdf format) and source code (in a compressed archive) electronically in Cambro.

The report may be brief but it is important that you fulfill the following requirements:

- The report must have a title page including all names, CAS-ids, and usernames at computing science for all of group members, and course name, course code, assignment name, and the teachers name.

- Describes how your solution works on a level that does not lose interesting details. Make some sensible breakdown into headings
    - Most important is that you describe your perceptron. Clearly describe how many nodes you use and what each node code for.
- Describe how your work went and which problems and issues you faced when developing the program.
- Use clear, well written **English** language
- You do **not** have to include your source code in the report

The source code should align to the following:

- Comment the code in a sensible way, follow JavaDoc, PyDoc if existing for the language you use
- Structure your code in a suitable way, such that it is easy to follow that is going on
- Name classes, variables, methods, modules, etc. such that the names clearly describe their function and **follow the specifications given above**
- The code you hand in should not contain sections that are not used, or unnecessary. Make sure that you understand every part of the code you hand in.