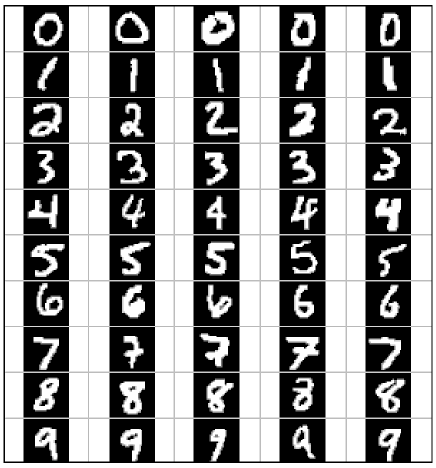
PRACTICAL 4

Machine learning and Pattern recognition

In this practical, we are going to explore different techniques for doing classification. As application example, we will use an OCR problem, where we want to teach the computer to recognise handwritten digits.

We will be using a standard and well-known database in the computer vision scientific community: MNIST digit dataset (http://yann.lecun.com/exdb/mnist/).

This dataset contains more than 70.000 images of scanned handwritten digit. They have been already pre-processed, centred and segmented for us, so it is ideal for us to learn ML techniques and pattern recognition methods on real-world data while spending minimal effort on pre-processing and formatting.



Download from Canvas the provided functions to assist you during the practical as well as the clips and image sequences to be used.

TASK 1: NEAREST NEIGHBOUR CLASSIFIER

Machine learning comprises a set of techniques that allow a computer to learn from examples in order to be able to later apply this knowledge to recognise an object or identify a problem. Machine learning algorithms operate in 2 different phases or stages.

The first stage, called training, is an off-line process where the learning happens. In this first stage, a set of examples are fed into the system, and a model of the problem/object is generated as output.

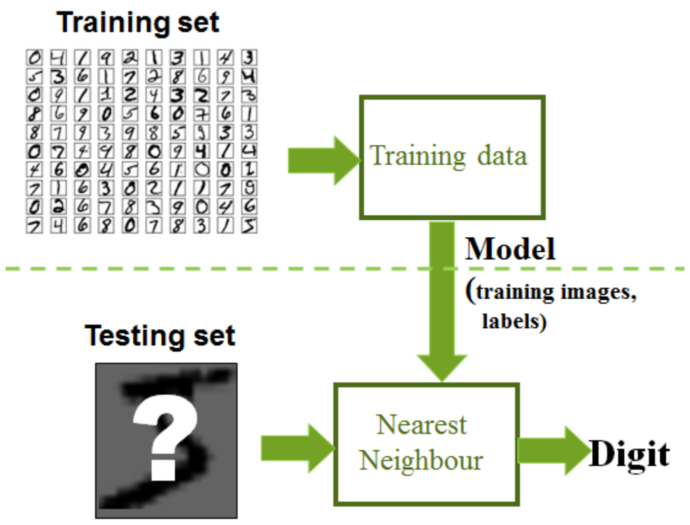
The second stage, called testing, is the process where new images/samples are fed in an online fashion, and the learned model is applied in order to predict the most likely class for each testing sample.

The simplest classifier is the nearest neighbour (NN). In this classification algorithm, a very simple learning takes place, where no real generalisation from the training examples is done. The block diagram of a NN classification system for OCR is depicted in the following figure

STEP 1: Open `OCR_NN_01` and read the code. This script will attempt to create an OCR system for binary code (0/1). For that purpose, it will read the MNIST digit database, extract the digits corresponding to 0 and 1s and use them to train the NN classifier. Be sure that you follow the code through and link each part of the code to the conceptual block diagram.

Now run the training stage of the code and analyse the generated model. How advance is this model?

Please remember that the two hour duration of a practical class will probably not be enough time to complete all sections. There is an expectation that you will work outside of class as an independent learner.



STEP 2: To be able to run the testing stage, we will need to implement the NN classifier. Create and complete the functions:

```
function prediction = NNTesting(testImage, modelNN)
```

and

```
function dEuc=EuclideanDistance(sample1, sample2)
```

The NN classification algorithm follows the following steps:

- Calculate the Euclidean distance between the test sample and all the training samples
- Select the closest training example
- Assign the closest training example's label to the test image

The Euclidean distance equation to be implemented in the second function is:

$$d(sample1, sample2) = |sample1 - sample2|$$
$$= \sqrt{(sample1(1) - sample2(1))^2 + (sample1(2) - sample2(2))^2 + \cdots + (sample1(n) - sample2(n))^2}$$

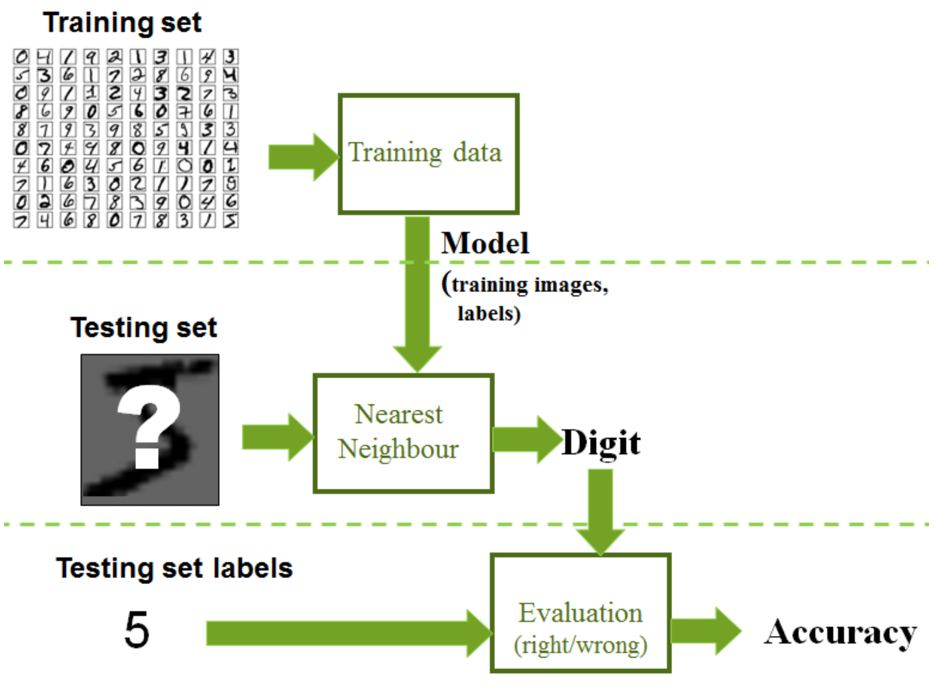
STEP 3: Run the full script now. Write down the Accuracy result given by your system. We will use it to compare it with other alternative in the following tasks.

There are two things to note. The first one is how a machine learning algorithm is evaluated (see the figure below). Once the blind testing is done, the labels of the annotated testing images are used to compare the answer given by your blind classification (i.e. your prediction). This comparison is summarised in the accuracy value, which is calculated as:

$$Accuracy = \frac{\text{Number of test images identified correctly}}{\text{Number of test images}}$$

The second thing to note is that the testing images are different from the ones used in training (no training image is used again for testing) and that no information about the labelling/annotation is used during the testing, but only in the third evaluation stage. All this is crucial to ensure that the results from your machine learning algorithm are trustworthy and you are not trying to cheat or make the problem easier for your algorithm.

In order to understand why given a training image for testing is cheating, answer the following question. What will be the minimum Euclidean distance in case we pass a training image to the NNTesting function?



TASK 2: K-NEAREST NEIGHBOUR EXTENSION

1-NN classifier is not a very robust option. The final decision is based on a matching against a single training example. This means that, if the selected training sample is corrupted, noisy or an outlier, our classification will be probably wrong.

In order to avoid this problem, an extension called K-NN can be applied. In this new version, not one but the k closet neighbours to the test sample are selected. Then, each of this K neighbours vote with their corresponding classes to the final decision. The class with more votes, is the one selected for classification.

STEP 1: Implement the function

```
function prediction = KNNTesting(testImage, modelNN, K)
```

which will be a modified version of the previous `NNTesting`. This function should work for any value of K. [HINT1: in order to implement this function you can use the function `sort`. Check in the documentation all the output parameters provided by `sort`]. [HINT 2: a possible implementation for the voting step is to create a histogram for the labels. If you do not like histograms, then the function `mode` can be of great help].

STEP 2: Replace the function `NNTesting` with `KNNTesting`. Test the accuracy of the systems for different values of K= 1 (should give the same that NN), 3, 5, 10. What conclusion can you extract?

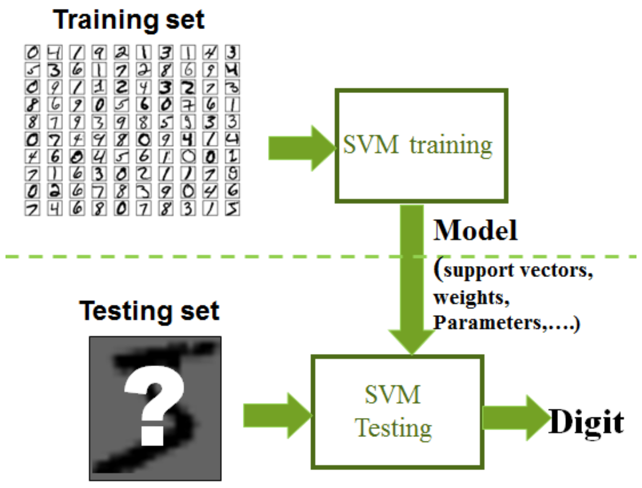
TASK 3: BINARY SVM CLASSIFIER

In this task we want to perform the same classification than before but applying a theoretically better classification technique, SVM.

STEP 1: Save the previous script with a new name (OCR_SVM_01.m) and replace the NN training and testing functions by `SVMtraining` and `SVMTesting` respectively. These functions will be using internally a SVM toolbox provided. In order to be able to use this toolbox, the toolbox folder should be added into the path. This is done by adding at the beginning of the script:

```
addpath  .\SVM-KM\
```

STEP 2: Open the SVM training and testing functions and understand the code and what is normally expected in a toolbox function to work (understand and tune adequately the input parameters, format the input date, collect the output). Our classifier will follow now this block diagram:



STEP 3: Run the script and write down the results. How do they compare against the NN and k-NN? Also observe the cloud of points and the support vectors drawn. Was the obtained SVM result as expected given the difficulty of the problem?

[Note] There are many SVM toolboxes and libraries that operate in a very similar fashion. Matlab has its own implementation in the statistics toolbox. If you wish to compare how this works, the functions `SVMtraining_v2` and `SVMTesting_v2` are provided for you to play with.

TASK 4: MULTICLASS CLASSIFICATION

Now that we understand the basics, we can target a more complex and challenging problem. To do so, we will create a classifier able to identify the 10 different digits from 0 to 9.

STEP 1: Save the previous script with new names `OCR_NN_09` and `OCR_SVM_09`. Remove the lines in testing and training that select only the 0 and 1s. This will make that the classifiers are fed with all the different digits images and labels. Since these are many images, increase the sampling variable to 50.

STEP 2: Open again the function `SVM_training`. Note that, although SVM is a binary classifier, there are strategies to extend it to multiclass.

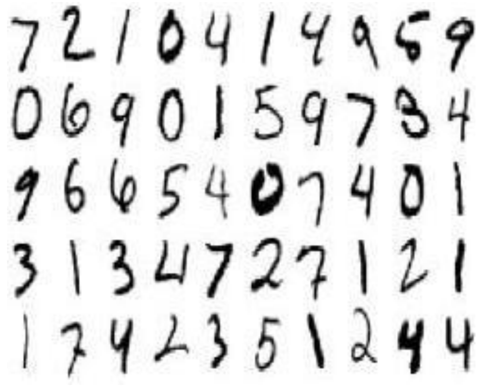
STEP 3: Run the training and testing. Observe now the cloud of points and think about the complexity of the problem. Compare both classifiers now and decide which one is better. [Hint: if the NN accuracy rate is too low, you may want to rethink if the function `NNTesting/KNNTesting` is valid for more than 2 labels]

STEP 4: Play with the SVM parameters to see if you can optimise the results. Try also using non-Gaussian kernels to see if the kernel trick to deal with non-linearity improves the results.

STEP 5: Compare your results against those ones described in the MNISt digit database webpage.

TASK 5: FROM CLASSIFICATION TO DETECTION: SLIDING WINDOW

In our final task, we are going to use our best previous classifier to apply to a scanned images where someone wrote down a few telephone numbers.



STEP 1: Run the best classifier from the previous task. Save the generated model into a file using the Matlab command:

```
save detectorModel model
```

STEP 2: Open the script `OCR_Detector.m`, which will help you on solving this task. Load the model you just saved in the previous step using the command:

```
load detectorModel
```

In addition, complete the line 56, using the testing function of your chosen classifier.

STEP 3: If you try to run the script you will notice that the recognition does not work. This is due to the fact that we need to convert the digits into a format as close as the training as possible. This will imply:

- Convert the digit images into doubles with the pixel values in the range 0 to 1, instead of 0 to 255
- Invert the colour schema (i.e get the negative image of the digit), since in training digits were white on black background and now in this figure they are the opposite
- Re-sample the digit since Training images were 28x28.
- Transform the matrix into a vector (flatten the digit)

[HINT: Matlab function `imresize` and `reshape` will help you in some of the previous steps]

STEP 4: Run the script and check the resulting Accuracy. Is the number similar to the one obtained in previous tasks? Come up with a theory why that is.

STEP 5: Open the function `preprocessingDigit` and try to understand what it tries to achieve. Do you think it will improve results? Call the function to process the digit just before transforming them into vector. What is the accuracy now?

We have completed our first digit detector using a simple sliding window strategy. However, this detector is working thank to 2 assumptions that we are not always true: that we know the size of the digit in the image and that all crops will contain a valid digit.

STEP 6: To fully understand what happens when these assumptions are not true, change the for loops for these ones:

```
county=0;
for r=1:5:size(I,1)
    predictedRow=[];
    county=county+1;
    countx=0;
    for c= 1:5:size(I,2)
        countx=countx+1;
```

What is the result? Is that useful?

First modification that we will need will consist on using the classifier not only for obtaining a prediction but also for measuring how confident is the classifier in its own prediction. This can be done:

- In a k-NN classifier: by evaluating the distance to the closet neighbour/ average distance to the k-closest neighbours
- In SVM: by evaluating the distance to the support vector (variable `maxi`)

STEP 7: Save all the values of that distance into a matrix map

```
map(county,countx) =maxi;
```

Display the matrix map as an image using `imagesc`(). Evaluate the result and think on a strategy to detect the digits and discard the non-digit crops.