# *ClassIm* User Guide

Ana Osojnik, Federico Danieli
*University of Oxford, Oxford, United Kingdom*

**Abstract**

ClassIm is an open-source Matlab software for image classification, developed during the course *Scientific Computing* taught in the CDT *Industrially Focused Mathematical Modelling* at the University of Oxford. It is responsible for the classification of the images in the MNIST handwritten digits database [1] using a variety of methods the user is allowed to choose from.

This guide is responsible for describing the main routines this software makes use of, so to provide the user a reference for usage, future manipulations and eventual expansions.

*Keywords:* K-nearest neighbours method, K-means clustering, Principal Component Analysis, Linear Discriminant Analysis, Image Classification, Suport vector machines method, Supervised and unsupervised machine Learning

## 1. Code description

### *1.1. Main routine*

The core of the code is summarised in the `main.m` routine, which the user should invoke in order for the software to run. Here, the user is allowed to modify some key parameters to adjust the behaviour of the code. A list containing their description is provided next.

- `method_features` Option to define the main method used for extracting features for classification. Available methods are: *intensity, PCA, PCAs, LDA.*
- `no_vectors` (only relevant with *PCA* and *PCAs*) Number of features that need to be extracted. This value should not exceed the image size. Increasing this value should in general increase the classification accuracy, at the cost of more computational time.

- `no_remove` (only relevant with *PCAs*) Number of first most representative features to remove. This value should not exceed the one of `no_vectors`.

- `method_classification` Specify the method used for conducting classifications as a cell array including two options. The first option is the method itself, available methods are: *knn, svm, kmeans.* The second option enables a choice of standardization of features before classification, inputs are: *standardize, nan.*

- `k` (only relevant with *knn*) Number of closest images to consider to actually assign a class to the test images.

- `size_train` Number of images to pick from the database for training the algorithm. In the database, 60000 train images are present, so the value specified should not exceed it. Images are picked at random from the database.

- `size_test` Number of images to pick from the database for testing the algorithm. In the database, 10000 test images are present, so the value specified should not exceed it. Images are picked at random from the database.

The `main.m` routine invokes `read_data` for importing images from the database, `edit_data` to perform pre-processing on them, `compute_features` to extract the most relevant features for classification and `classify` to perform the actual image classification and compute the corresponding score. All routines are timed for profiling. Their behaviour is described next.

## *1.2. Read data*

This function is responsible for importing images from the MNIST database. As their original format is binary, first they need to be converted into a format that can be used by Matlab. This is done by the internal function `readMNIST`, slightly modified from [2]. The actual train and test images are picked at random without repetitions from the original database. The images labels are collected as well: they will be used to perform classification and to compare results with the ground truth. In order to work properly, the function expects there to be a folder `MNIST` in the directory where the `.m` files are present. Here should be stored the databases extracted from [1].

### *1.3. Edit data*

The image data collected is adjusted by `edit_data`. This routine translates both test and train images removing the average values of the train data. This way, the database is modified to have zero mean, which is considered a desirable property to achieve better results for the classification algorithm[1].

### *1.4. Compute features*

The purpose of `compute_features` is to simplify data representation to be able to perform classification in a faster way. This is usually done by projecting the whole image in a subspace of lower dimension, described by particular features so-called *eigenimages*. This effectively reduces the size of each image from the original to the prescribed `no_vectors` (see Sec.1.1). Comparing images (*e.g.*, computing some sort of distance between them) is hence much faster.

The way this subspace is computed identifies the method: a list of available methods follows, together with their description.

#### *1.4.1. Intensity*

The baseline method: no features are extracted, no projection is performed and data is left untouched. The `classify` routine will then analyse images on a per-pixel basis, comparing the intensity value of each of them.

#### *1.4.2. PCA*

Stands for *Principal Component Analysis*[4]. The eigenimages extracted are the eigenvalues of the covariance matrix of the train data. The ones corresponding to the `no_vectors` largest ones are recovered using Matlab internal function `eigs`. It is possible for the user to remove the first `no_remove` of them. This is because some times they might be associated to general values of picture brightness, hence not relevant for classification. Once the eigenvectors are recovered, both train and test images are projected on them. The projection coefficients represent the reduced images, and they are the output of the function.

[1]Data normalization is a commonly used approach when performing data analysis in general. See for example [3]

### *1.4.3. PCAs*

Stands for *Principal Component Analysis - scaled.* This method is basically identical to PCA, with the sole difference that the eigenvectors are scaled according to their corresponding eigenvalue before performing projection.

### *1.4.4. LDA*

Stands for *Linear Discriminant Analysis*[5]. Due to the redundancy of the dataset (dark areas present in every images), the classical LDA consistently fails. To overcome this problem, first a PCA is applied to remove this redundancy, and the data is projected on a subset of large dimension. Next, the LDA is performed on the already reduced data to further reduce dimension. This procedure involves first dividing the train data in classes (according to their label, *i.e.*, the digit they represent); next, computing inter- and intra-class covariance matrices; finally, solving a generalised eigenvalue problem to compute the eigenimages.

## *1.5. Classify*

The function `classify` takes either original or reduced image projections and based on those classifies data into clusters. Search for classes is performed via different algorithms which identify three methods. `knn` and `svm` perform supervised learning whereby we feed in the chosen reduced image projection, whereas `kmeans` performs unsupervised clustering with original images as input (choose `intensity` as `method_features` in `main`).

### *1.5.1. k-nearest neighbours method*

k-nearest neighbours algorithm (`knn`) is a method that finds for every input test point (image) $k$ closest points in the training data set based on some measure of distance. The measure used here is Euclidean distance and the search for data points with smallest distances is performed via k-d trees (a binary space-partitioning method). The variable `k` provides an option to specify the number of neighbours to be found. Class of a test image is chosen to be the most frequent (the `mode`) class among the $k$ neighbours found. `knn` can be used in conjunction with any `method_features`.

### *1.5.2. Support vector machines method*

The basic support vector machines is a binary learning algorithm which maps training data so it is separated by a clear gap which is as wide as possible. Test data is then mapped into the same space. The method in this program is an adaptation and performs multiple binary support vector machines classifications of test data for each digit and then aggregates these results according to an error minimisation scheme. For more details see the MATLAB function `fitcecoc`. The `svm` method can be used with any `method_features`.

### *1.5.3. k-means clustering method*

*k*-means clustering (`kmeans`) aims to partition images into as many classes as there are in the training set. The clusters of the training set are chosen as to minimise the sum of distance functions (squared Euclidean distance) of each cluster point to the cluster mean, called the centroid, via an efficient heuristic algorithm. Given the cluster centroids found for the training set, test images are assigned to a cluster with the closest centroid of training data. This is done using an exhaustive `knn` search as before with the training centroids as input to find the closest centroid neighbour. One should set `method_features='intensity'`, i.e. original images as features, when using this method. However, choosing other methods of computing features is not going to produce an error.

## 2. Examples

As an usage example, here is reported the parameters setup for a classification that uses PCA to extract features, keeps all of the 10 most relevant ones, and performs classification using knn checking against 10 closest neighbours. The classification is done using the whole test and train database.

```
method_features = 'PCA';
no_vectors = 10;
no_remove = 0;

method_classification = {'knn','standardize'};
k = 10;

size_train = 60000;
size_test = 10000;
```

The Matlab output, reporting classification score and time (in seconds) necessary for the execution of each sub function, looks like the following.

```
Time to read data =1.8166
Time to edit data =0.34139
Time to compute features =0.84186
Time to perform classification =2.3334
Computation complete! Classification score =0.9258
```

## 3. Performance testing

The code also involves adaptations of the `main` file in which classification is run multiple times with some varying method parameter. Performance scores are plotted and plots saved to a `.png` file. Performance testing files are named as `main_*.m` where `*` is the name of the varying parameter, for example in `main_k.m` the parameter `k` is varied. The advanced user who intends to experiment the effects of varying function parameters towards classification performance, is invited to take these functions as templates.

[1] Y. LeCun, C. Cortes, C. Burges, The mnist database of handwritten digits, `http://yann.lecun.com/exdb/mnist/`, 1998.

[2] M. Sirotenko, Function to import images from mnist database, `https://uk.mathworks.com/matlabcentral/fileexchange/24291-cnn-convolutional-neural-network-class/content/ver\%200.83/readMNIST.m`, 2009.

[3] C. Date, An Introduction to Database Systems, Addison-Wesley Longman, 1999.

[4] I. T. Jolliffe, Principal Component Analysis, Springer, 2002.

[5] R. O. Duda, P. E. Hart, D. G. Stork, Pattern Classification, Wiley Interscience, 2000.