# Introduction to learning and analysis of big data
# Exercise 1

Prof. Kontorovich and Dr. Sabato

Fall 2016/7

Submission guidelines, **please read and follow carefully**:

- You may submit the exercise in pairs.
- Submit using the submission system.
- The submission should be a zip file named "ex1.zip".
- The zip file should include **exactly two files in the root - no subdirectories please**. **Anywhere in the exercise where Matlab is mentioned, you can use Octave instead**.
- The files in the zip file should be:
  1. A file called "answers.pdf" - The answers to the questions, including the graphs.
  2. A file called "nn.m" - The Matlab code for the requested function. Note that you can put several auxiliary functions in this file after the definition of the main function. **Make sure that the single file works in Matlab/Octave before you submit it.**
- Getting started with Matlab: You can use the guide in this link: `http://www.mathworks.com/help/pdf_doc/matlab/getstart.pdf?s_tid=int_tut`. There are plenty of other tutorials, documentation and examples on the web. You can run Matlab on the lab computers. Octave is free to download and use.
- Grading: Q.1 (`nn.m` file): 40 points, Q.2: 30 points, Q.3: 30 points.
- For questions use the course Forum, or email `inabd17@gmail.com`.

**Question 1**. Implement a function that runs the **k-nearest-neighbors** algorithm we saw in class on a given training sample, and uses it to predict the labels of examples. The function should be implemented in the submitted file called "nn.m". The first line in the file (the signature of the function) should be:

```
function Ytest_predict = nn(k, m, d, ntest, Xtrain, Ytrain, Xtest)
```

The input parameters are:

- `k` - the number $k$ to be used in the $k$-nearest-neighbor algorithm. You may assume that $k$ is an odd integer.
- m - the size of the training sample $S = ((x_1, y_1), \ldots, (x_m, y_m))$, an integer $m \geq 1$.
- `d` - the number of features in each example in the training sample, and integer $d \geq 1$. So $\mathcal{X} = \mathbb{R}^d$.
- ntest - the number of test examples that the function will need to predict a label for.
- `Xtrain` - a 2-D matrix of size $m \times d$ (note: first number is the number of rows, second is the number of columns). Row $i$ in this matrix is a vector with $d$ coordinates that describes example $x_i$ from the training sample.
- `Ytrain` - a column vector of length $m$ (that is, a matrix of size $m \times 1$). The $i$'s number in this vector is the label $y_i$ from the training sample. You can assume that each label is an integer between 0 and 9.
- `Xtest` - a 2-D matrix of size $ntest \times d$. Each row in this matrix is a vector with $d$ coordinates that describes one example that the function needs to label.

The function returns the variable `Ytest_predict`. This is a column vector of length `ntest` (that is, a matrix of size $ntest \times 1$). Label $i$ in this vector describes the label that the k-nearest-neighbor algorithm predicts for the example in row $i$ of the matrix `Xtest`.

Important notes:

- You may assume all the input parameters are legal.
- You do not need to come up with a very efficient implementation, implement a naive solution.
- The Euclidean distance between two vectors `z1,z2` of the same length can be calculated in Matlab/Octave using `norm(z1-z2)`.

Example for using the function `nn` (here there are only two labels, 0 and 1):

```
>> k=1;
>> m=3;
>> d=2;
>> ntest=4;
>> Xtrain=[1,2;3,4;5,6];
>> Ytrain=[1;0;1];
>> Xtest=[10,11;3.1,4.2;2.9,4.2;5,6];
>> Ytest = nn(k,m,d,ntest,Xtrain,Ytrain,Xtest);
>> Ytest

Ytest =

     1
     0
     0
     1
```

**Question 2**. Test your k-nearest-neighbor implementation on the **hand-written digits recognition** learning problem: In this problem, the examples are images of hand-written digits, and the labels indicate which digit is written in the image. The full data set of images, called MNIST, is free on the web. It includes 70,000 images with the digits 0-9. For this exercise, we will use a smaller data set taken out of MNIST, that only includes images with the digits 3,4,5 and 6, so that there are only four possible labels.

Each image in MNIST has 28 by 28 pixels, and each pixel has a value, indicating how dark it is. Each example is described by a vector listing the $28 \cdot 28 = 784$ pixel values, so we have $\mathcal{X} = \mathbb{R}^{784}$, every example is described by a 748-coordinate vector.

Figure 1: Some examples of images of digits from the data set MNIST

The images in MNIST are split to training images and test images. The test images are used to estimate the success of the learning algorithm: In addition to the training sample $S$ as we saw in class, we have an additional **test sample**, $T$ which also consists of pairs of labeled examples.

The learning algorithm gets $S$, decides on $\hat{h}_S$, and then predicts the labels of the test images in $T$ using $\hat{h}_S$. The error of the prediction rule $\hat{h}_S$ on the test images is $\mathrm{err}(\hat{h}_S, T) = \frac{1}{m}\sum_{(x,y)\in T} \mathbb{I}[\hat{h}_S(x) \neq y]$ (that is, the fraction of test images for which it got the wrong answer). It is a good estimate of the error of $\hat{h}_S$ on the distribution $\mathrm{err}(\hat{h}_S, \mathcal{D})$.

You will need the following files, which can be found in the course website `http://www.cs.bgu.ac.il/~inabd171/Assignments`: `mnist_all.mat` and `gensmallm.m`

To load all the MNIST data to Matlab, run the following command (if the file `mnist_all.mat` is not in your Matlab path or current directory, add the file path to the name of the file):

```
>> load('mnist_all.mat');
```

After this command you will have the variables:
`train0,train1,...,train9,test0,test1,...,test9` in your Matlab workspace.

To generate a training sample of size `m` with images only of some of the digits, you can use the provided function `gensmallm`, which is used as follows:

```
>> [Xtrain,Ytrain,Xtest,Ytest] = gensmallm(trainA, trainB, testA, testB, m);
```

The function `gensmallm` selects a random subset from the train data and mixes them together in a random order, to generate the training sample `Xtrain,Ytrain`. It also mixes the test images

together and generates the test sample `Xtest,Ytest`. To mix 4 digits together, you can use gensmallm several times, each time adding one more digit to the sample (e.g. add `train3` and `test3` for digit 3).

Answer the following questions in the file "answers.pdf".

(a) Run your `nn` implementation with $k = 1$, on several training sample sizes between $1$ and $500$. **Note: you do not need to run on every size between $1$ and $500$, choose a reasonable number of sizes so that your graph is informative**. For each sample size, calculate the error on the test sample. You can calculate this error, for instance, with the command:

```
>> mean(Ytest ~= Ytest_predict).
```

Submit a plot of the test errors as a function of the training sample size. Don't forget to label the axes. You can use Matlab's `plot` command (or any other plotting software).

(b) Do you observe a trend in the graph? What is it? How do you explain it?

(c) Does the graph look smooth for small sample sizes? What could make the graph appear non-smooth?

(d) Run your `nn` implementation with a training sample size of a 100, for values of $k$ between 1 and 9 (only odd k). Submit a plot of the test errors as a function of $k$.

(e) Do you observe a trend in this graph? Try to explain the results.

(f) Choose one of your runs which had a relatively low error. For the output of this run (provide that run's parameters), provide a *confusion matrix*: A table where each column and each row correspond to one of the digits, and the number in cell (digit1,digit2) is the percentage of images that are actually digit1, and were predicted by the algorithm to be digit2. Explain the findings you observe in the confusion matrix.

**Question 3**. In this question you will show that even in a very simple distribution, 1-nearest-neighbor does not necessarily approach the Bayes optimal error, and can have an error that is arbitrarily close to twice the Bayes-optimal error. You should prove your claims, using basic probability and expectation properties.

(a) Let $\mathcal{X} = \{a, b, c\}$, and let $\mathcal{Y} = \{0, 1\}$ (two labels). Let $\mathcal{D}_a$ be a distribution such that $\mathbb{P}_{(X,Y)\sim\mathcal{D}_a}[X = a] = 1$, and $\eta(a) := \mathbb{P}_{(X,Y)\sim\mathcal{D}_a}[Y = 1 \mid X = a]$.

In the 1-nn algorithm, assume that for any test point $x$, if the training sample $S$ has more than one point that is nearest to $x$ (there's a tie), then one of these points from the sample is selected uniformly at random. Let $\hat{h}_S$ be the (possibly random) prediction rule of the 1-nn algorithm for a given training sample $S$.

Show that for our distribution $\mathcal{D}_a$, a single number that depends on $S$ (call it $\beta_S$), controls $\mathbb{P}[\hat{h}_S(x) = 1]$, for any possible test example $x$. Give a formula for $\beta_S$ as a function of $S = (x_1, y_1), \ldots, (x_m, y_m)$. Note that here the probability is over the randomness of $\hat{h}_S$, since $S$ and $x$ are fixed.

(b) Calculate $\mathrm{err}(\hat{h}_S, \mathcal{D}_a)$ as a function of $\beta_S$ and $\eta(a)$.

(c) Calculate the expectation of $\beta_S$ over samples, that is $\mathbb{E}_{S\sim\mathcal{D}_a^m}[\beta_S]$, as a function of $\eta(a)$.

(d) Calculate the expected error of the 1-nn algorithm for $\mathcal{D}_a$ over random samples, denoted by $\overline{\mathrm{err}} := \mathbb{E}_{S\sim\mathcal{D}_a^m}[\mathrm{err}(\hat{h}_S, \mathcal{D}_a)]$, as a function of $\eta(a)$.

(e) Calculate the error of the Bayes optimal rule for $\mathcal{D}_a$, denoted by $\mathrm{err}_{\text{bayes}}$, as a function of $\eta(a)$.

(f) Show that by setting the value of $\eta(a)$, we can get the ratio $\frac{\overline{\mathrm{err}}}{\mathrm{err}_{\text{bayes}}}$ to be larger than $2-\epsilon$, for any $\epsilon > 0$.