# INF554 - Machine learning I

# École Polytechnique

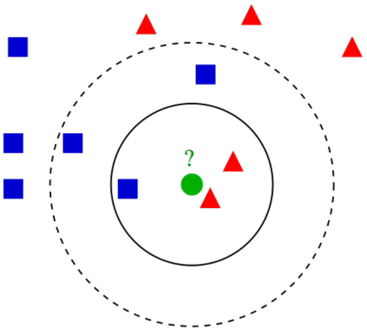
# Lab 2: Supervised Learning

Christos Giatsidis, Jesse Read, Nikolaos Tziortziotis and Michalis Vazirgiannis

September 25, 2017

## 1 Handwritten Digit Recognition with $k$-Nearest Neighbours Classification

The *$k$-Nearest Neighbors algorithm* ($k$-NN) is a simple, very intuitive, and commonly used method in the classification task. For a detailed introduction, refer to the lecture slides, or any machine learning textbook, e.g., [1][1]



In this lab, we will implement and apply the $k$-NN classifier to recognize handwritten digits from the MNIST database[2].

### 1.1 The Data

The MNIST dataset consists of handwritten digit images $(0-9)$ and it is divided in $60,000$ examples for the training set and $10,000$ examples for testing. Figure 1 depicts the first $10$ images of the dataset.
All digit images have been size-normalized and centered in a fixed size image of $28 \times 28$ pixels. Each pixel of the image is represented by a value in the range of $[0,255]$, where $0$ corresponds to black, $255$ to white and anything in between is a different shade of grey. In our case, the pixels are the features of our dataset; therefore, each image (instance) has $784$ features. That way, the training set has dimensions $60,000 \times 784$ and the test set $10,000 \times 784$. Regarding the class labels, each figure (digit) belongs to the category that this digit represents (e.g., digit $2$ belongs to category $2$). Due to time constraints, in the experiments that will be performed in the lab, we will use subsets of the above training and test sets. The code that imports the MNIST dataset has been implemented in `loadMnist.py`.

[1] Available for free online: `https://web.stanford.edu/~hastie/Papers/ESLII.pdf`

[2] The MNIST database: `http://yann.lecun.com/exdb/mnist/`.

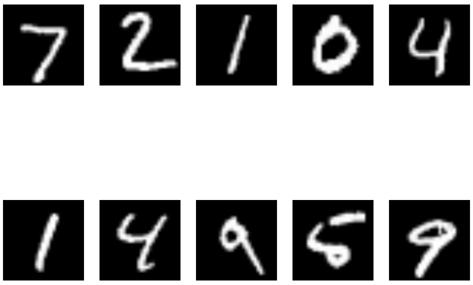


Figure 1: Example of 10 digits of the training set.

### 1.2 The Pipeline

The main file `main.py` loads and splits the data (into train and test sets), displays the first 10 samples (for the sake of inspection) and runs the (initially null) $k$NN classifier. Recall that each instance is a digit with $28 \times 28 = 784$ features (pixels).

Note that since the dataset is relatively large, we keep a subset of the training and test data, due to time constraints of the lab, and the fact that the $k$-NN algorithm is computationally expensive.

The $k$NN classifier is called for each test instance, and the predictions are finally compared with the true labels so as to calculate and display classification accuracy.

### 1.3 The Tasks

1. Run `main.py` and check that the code runs; and check that you understand the flow of the code.
2. Implement the $k$-NN classification algorithm in the file `kNN.py`
3. Run `main.py` and note the accuracy.
4. In `main.py` change the variable $k$, and run again. Note the change in accuracy.
5. Change the size of the training set. Observe the trade-off between accuracy and running time.

## 2 Logistic Regression

The objective of this part is to implement the logistic regression method and use it to find the decision boundary in a classification problem. The problem is to decide the probability that a student will be admitted to the Master's program of a university based on the grades of two courses. *Logistic regression* is a supervised learning algorithm that can be applied to binary or multinomial classification problems. In a binary classification problem (i.e., two classes) we are given a set of training data $\mathcal{D} = \{\boldsymbol{x}_i, c_i\}_{i=1}^{N}$, where $c \in \{0, 1\}$ is a target variable such that $c = 0$ represents class $\mathcal{C}_0$ and $c = 1$ represents class $\mathcal{C}_1$. Logistic regression corresponds to the next model:

$$p(c = 1|\boldsymbol{x}) = \sigma(b + \boldsymbol{x}^\top \boldsymbol{w}), \tag{1}$$

where $b$ is a bias factor (scalar), $\boldsymbol{w}$ is a weight vector. Here $\sigma(\cdot)$ is the logistic sigmoid function defined as:

$$\sigma(x) = \frac{1}{1 + \exp^{-x}}. \tag{2}$$

Input $\boldsymbol{x}$ is classified as $1$ if $p \geq 0.5$, and as $0$ otherwise. The decision boundary separating the two predicted classes is the solution of $b + \boldsymbol{x}^\top \boldsymbol{w} = 0$, which is a point if $\boldsymbol{x}$ is one dimensional, a line if it two dimensional, a plane in the case of three features, etc..

Logistic regression learns the weights $\boldsymbol{w}$ and bias $b$ so as to maximize the likelihood of the data. The likelihood function can be expressed as:

$$p(\mathcal{D}|b, \boldsymbol{w}) = \prod_{i=1}^{N} p(c = 1|\boldsymbol{x}_i, b, \boldsymbol{w})^{c_i} (1 - p(c = 1|\boldsymbol{x}_i, b, \boldsymbol{w}))^{1-c_i}. \tag{3}$$

Based on the likelihood function we define a cost function by taking the negative logarithm of the likelihood:

$$E(\boldsymbol{w}, b) = -\frac{1}{N} \sum_{i=1}^{N} \left\{ c_i \log \sigma(b + \boldsymbol{x}_i^\top \boldsymbol{w}) + (1 - c_i) \log(1 - \sigma(b + \boldsymbol{x}_i^\top \boldsymbol{w})) \right\} \tag{4}$$

Thus, logistic regression aims to find parameters $(b, \boldsymbol{w})$ in order to minimize the above cost function. This can be achieved taking the gradient of the cost function with respect to parameters $(b, \boldsymbol{w})$ and applying the gradient descent method:

$$\nabla_{\boldsymbol{w}} E = \frac{1}{N} \sum_{i=1}^{N} \left( \sigma(b + \boldsymbol{x}_i^\top \boldsymbol{w}) - c_i \right) \boldsymbol{x}_i. \tag{5}$$

The derivative with respect to the bias is:

$$\frac{dE}{db} = \frac{1}{N} \sum_{i=1}^{N} \left( \sigma(b + \boldsymbol{x}_i^\top \boldsymbol{w}) - c_i \right). \tag{6}$$

## 2.1 The Data

The data that will be used on this task contains the grades of students that were applied to a university Master's program along with the admissions decision (accepted or not - binary). More specifically, each row corresponds to a specific applicant, and the features are the applicant's grades on two exams. The final column denotes the class label that indicates if the student was accepted or not to the Master program. Figure 2 depicts the training data. The axes correspond to the grades on the two courses. The blue dots show the grades of the students that were admitted to the Master's program, while the red × to the students that were rejected.

## 2.2 The Pipeline

The pipeline of the task is in the `logistic/main.py` script file. Initially, we load the data contained in the `data.txt` file. Then, we proceed with the main task. We initialize parameters $\{\beta, \boldsymbol{w}\}$ with zero and call the `minimize()` optimization function of Python[3], that will minimize the value of the `computeCost()` function defined in Eq. (4), with respect to parameters $\{\beta, \boldsymbol{w}\}$. In the `minimize()` function, we also use the `computeGrad()` function, that computes the gradient of the cost function defined in Eqs. (5) and (6). Also note that the `args` parameter contain the training data $\mathbf{X}$ and the corresponding class labels. The function returns the values of parameter $\{\beta, \boldsymbol{w}\}$. Finally, we perform the classification of the data and examine the accuracy of our model.

[3] `scipy.optimize.minimize` function: `http://goo.gl/HoM6Ib`

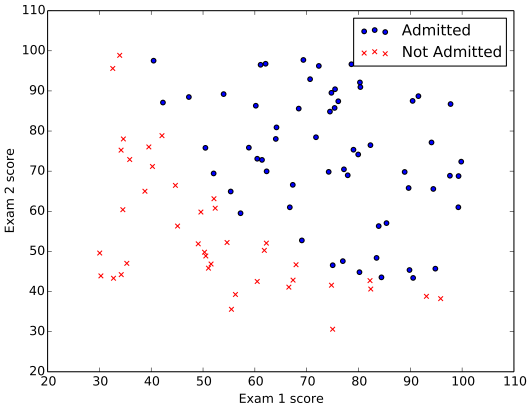


Figure 2: Training data.

## 2.3 The Tasks

1. Run `main.py` and and check that you understand the flow of the code.

2. Implement the *sigmoid* function[4] in the `sigmoid.py` file.

3. Implement the cost (error) function in the `computeCost.py` file.

4. Implement the gradient of the cost function with respect to models' paraments in the `computeGrad.py` file.

5. Apply the logistic function on the data (dot product between test data and parameters) in the `predict.py` file. Calculate the prediction accuracy achieved by applying your model on the data.

## 2.4 Bonus Task

Implement your own optimization function based on the standard stochastic gradient descent algorithm[5]. Use stochastic gradient descent function to minimize the cost function defined in Eq. (4). Compare the accuracy of your solution with that returned by using the standard optimization function provided by Python.

# References

[1] Trevor Hastie, Robert Tibshirani, and Jerome H. Friedman. *The elements of statistical learning: data mining, inference, and prediction, 2nd Edition*. Springer series in statistics. Springer, 2009.

[4] Wikipedia's lemma for *Sigmoid function*: `http://en.wikipedia.org/wiki/Sigmoid_function`.

[5] Wikipedia stochastic gradient descent article: `https://en.wikipedia.org/wiki/Stochastic_gradient_descent`