SI221 Practical assignment #2: k-Nearest Neighbors - 10/03/2020

Instructions

Read all the instructions below carefully before you start working on the assignment.

- Please submit the source code and a pdf report of your work. Each file must have as title: *TP_kNN_Student1_Student2*.
- Late assignments will not be correct.
- You must do this assignment in groups of 2. Please submit no more than one submission per group.
- Send your work in a zipped file to `giovanna.varni@telecom-paris.fr` `emile.chapuis@telecom-paris.fr` and `michelle.nazareth@telecom-paris.fr` by March 17th at latest.

Objective

This assignment is aimed at coding k-nearest neighbors algorithm (kNN) from scratch in order to classify hand-writing data.

In the following sections you can find a brief summary on how kNN works; a description of the data set you will use; and finally the text of the assignment.

Reminders about kNN

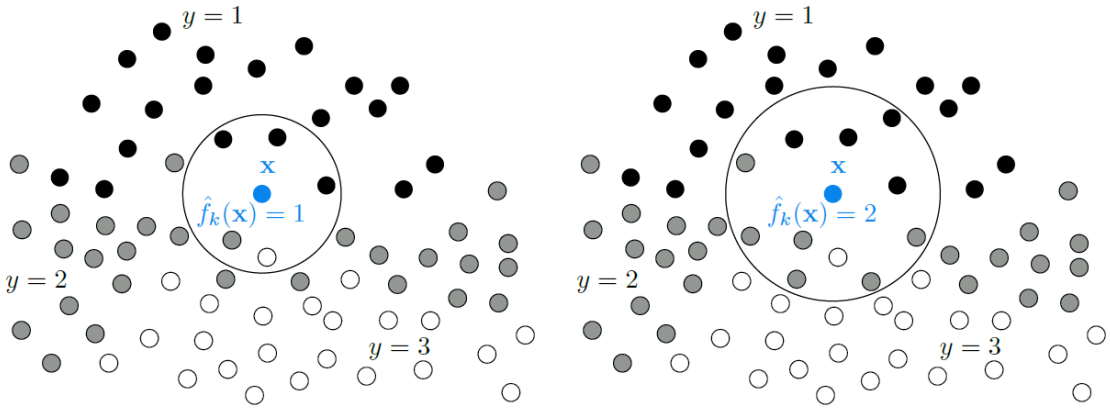
Historical introduction

K-nearest-neighbor (kNN) classification is one of the most fundamental and simple classification methods and should be one of the first choices for a classification study when there is little or no prior knowledge about the distribution of the data. K-nearest-neighbor classification was developed from the need to perform discriminant analysis when reliable parametric estimates of probability densities are unknown or difficult to determine. Some confusion surrounds the history of kNN, partly because it is the result of the work of many people. However, most people ascribe the initial idea to E. Fix and J.L. Hodges that, in an unpublished US Air Force School of Aviation Medicine report in 1951, introduced a non-parametric method for pattern classification. Later, in 1967, T.M. Cover and P.E. Hart worked out some of the formal kNN properties. The history of kNN continued up to 2000, when S. Bermejo and J. Cabestany presented adaptive soft learning kNN classifiers.

Working with kNN

kNN is an intuitive, easily configurable algorithm to handle a classification problem with any number of labels. Its principle is particularly simple: for each new sample $\mathbf{x}$ the set $S_k(\mathbf{x})$ of its k-nearest neighbors among the n learning samples is determined. k must be chosen in the range $[0, n]$, of course. The class assigned to this new sample $\mathbf{x}$ is then the majority class in the set $S_k(\mathbf{x})$. Figure 1 depicts how the method works for the case of three classes.

Figure 1: How kNN works for $k = 5$ (a) and $k = 11$ (b), respectively. In this example, data belongs to three classes having label y=1 or y=2 or y=3.



(a) $k = 5$ (b) $k = 11$

The first step of the algorithm concerns the choice of a distance $d : \mathbb{R}^m \times \mathbb{R}^m \to \mathbb{R}$. For each new sample $\mathbf{x}$, its k-nearest neighbors set $S_k(\mathbf{x})$ is computed using this distance. The following procedure can be followed: for each $\mathbf{x}$ and for each $i = 1, ..., n$, the distance between $\mathbf{x}$ and $\mathbf{x_i}$ is denoted $d_i(\mathbf{x}) = d(\mathbf{x}, \mathbf{x_i})$. The index of the first nearest neighbor of $\mathbf{x}$ among $\mathbf{x_i}$ is defined as:

$$r_1(\mathbf{x}) = i^* \text{ if and only if } d_{i^*}(\mathbf{x}) = \min_{1 \leq i \leq n} d_i(\mathbf{x}).$$

By recurrence, it is possible defining $r_k(\mathbf{x})$ for each integer $1 \leq k \leq n$ as:

$$r_k(\mathbf{x}) = i^* \text{ if and only if } d_{i^*}(\mathbf{x}) = \min_{\substack{1 \leq i \leq n \\ i \neq r_1, ..., r_{k-1}}} d_i(\mathbf{x})$$

The set of the kNN is then expressed by $S_k(\mathbf{x}) = \{\mathbf{x_{r_1}}, \dots \mathbf{x_{r_k}}\}$.

Finally, classification of the sample $\mathbf{x}$ is done through majority voting, resolving this:

$$\hat{f}_k(\mathbf{x}) \in \underset{y \in \mathcal{Y}}{\text{argmax}} \left(\sum_{\mathbf{j=1}}^{\mathbf{k}} \mathbb{1}_{\{y_{r_j} = y\}} \right) \tag{1}$$

where $\hat{f}$ is a classifier associating to each new sample $\mathbf{x}$ one label $\hat{f}(\mathbf{x})$, and $\mathcal{Y}$ is the labels's set.

Getting started!

1 k-NN classification: Synthetic dataset

Generate a dataset $\{\mathbf{x}_i, y_i\}_{i=1}^{300}$ as follows. The label variable y_i takes value uniformly over $\{0, 1, 2\}$. Given y_i, the domain variable $\mathbf{x}_i \in \mathbb{R}^2$ is generated according to an independent bivariate Gaussian distribution $\mathbf{x}_i \sim \mathcal{N}(\mu, \sigma^2\mathbf{I})$ with mean vector $\mu \in \mathbb{R}^2$ given by:

$$\begin{cases} \mu = (-1, 0) \text{ if } y_i = 0 \\ \mu = (1, 0) \text{ if } y_i = 1 \\ \mu = (0, 1) \text{ if } y_i = 2. \end{cases}$$

Split the dataset in two groups: training data (2/3) and test data (1/3).

1. Generate one dataset with $\sigma^2 = 0.10$. Using the entire dataset as a plot training points and the test points, given by the algorithm, for $K \in \{1, 2, 5, 10\}$. Comment on what happens with the decision boundary when K increases.
   Hint: Use matplotlib.pyplot.countourf to plot the decision boundary.

2. Fix the variance value $\sigma^2 = 0.05$ and fix $K = 5$. Run the algorithm over 50 randomly generated datasets, compute the average error e and its standard deviation s. Repeat with $\sigma^2 \in \{0.10, 0.15, 0.20, 0.25\}$ and $K \in \{1, 2, 10\}$. Plot $e(\sigma^2, K)$ and $s(\sigma^2, K)$ by fixing first K then by fixing σ^2. Comment.

2 k-NN regression: Szeged-weather dataset

The Szeged-weather dataset is a daily/hourly summary for Szeged, Hungary area, between 2006 and 2016, in terms of temperature, humidity, apparent temperature, pressure, wind speed, among other measurements. This dataset can be downloaded at `https://www.kaggle.com/budincsevity/szeged-weather/data`.

We want to predict the apparent temperature given humidity and temperature. Apparent temperature is a notion of temperature that reflects human perception. The prediction, given by the k-nearest neighbors algorithm, is computed by taking into account the average of the values of K nearest neighbors.

1. Visualize the dataset in terms of temperature (x-axis), humidity (y-axis), and apparent temperature (color).

2. Consider the first 2000 samples of the dataset and set $K = 1$. Permute the order of the 2000 samples uniformly at random, and split the data set into 4/5 and 1/5. Each fold is then used once as a test while the 4 remaining folds form the training set. Compute the average of error e. Repeat five times and compute the average error and its standard deviation s. Represent for $K \in \{2, 3, 4, 5, 6, 7, 8, 9, 10\}$ and plot $e(K)$ and $s(K)$.

3 k-NN classification: MNIST dataset

The MNIST dataset is a large dataset of handwritten digits (from 0 to 9) and their labels (attention! the label associated to 0 is 10) that is commonly used for training various image processing systems. You can download data at `https://www.dropbox.com/sh/r1qu378v4fylh0b/AAA9AVXwKymcqyDcuIlfyAGDa?dl=0`in the folder TP_KNN. Find the training set and test set, and the corresponding labels in the `data_app.mat` and in the `data_test.mat` files, respectively.

The following Python code will allow you to read the data:

```
import scipy.io
train = scipy.io.loadmat('data_app.mat')
test = scipy.io.loadmat('data_test.mat')
```

`train` and `test` are Python dictionaries. To access digits you can use the key `'x'`, whereas to access labels you can use the key `'S'`. Each digit is represented as a 28 x 28 gray-scale image arranged as a row of `'x'`. There is a row for each digit.

1. Visualize the dataset. To do that, you can check the function `subplots` in the `matplotlib` library. In order to embed figures in your notebook, include the line `\%matplotlib inline` once at the beginning of your notebook. In order to be sure that the images will be displayed correctely, convert and normalize them in such a way: `image_c = image.astype(float) / 255`. Pay attention: use the images processed in this way *throughout the assignment* (also when you will compute distances).
   Plot the histograms of the labels corresponding to these two sets: what can you say about data distribution?

2. Consider the Euclidean distance between two vectors associated with each image of the training dataset $\mathbf{x}_i^{(\text{training})}$ and each image of the test dataset $\mathbf{x}_i^{(\text{test})}$ as distance $d\left(\mathbf{x}_i^{(\text{training})}, \mathbf{x}_i^{(\text{test})}\right)$. Classify the test dataset for $K = 1$ and compute the error rate of the classification. Repeat for $K \in \{3, 5\}$. Write a function `knn` to represent the kNN algorithm.

3. A confusion matrix allows visualization of the performance of an algorithm. Each row of this matrix represents the instances in a actual class while each column represents the instances in a predicted class. In our case, we have a matrix 10×10 where the element (i, j) denotes the number of samples belonging to class i that are classified by the algorithm as belonging to class j. Compute and display the confusion matrix. Write a function `confusion_matrix`.