X/HEC Datascience for Business Master
2019/2020 **Alexandre Gramfort, Joseph Salmon, Mathurin Massias**

LAB N° 2 : k-nearest neighbor

The files tp_knn_source.py and tp_knn_script.py are on Moodle. They contains functions needed for this Lab.

- REMINDER ABOUT CLASSIFICATION -

Definitions and notation

We remind the setup of multiclass supervised classification.

— $\mathcal{Y}$ is the set of labels. Here we consider L classes, and we choose $\mathcal{Y} = \{1, \ldots, L\}$ to represent the L possible labels. Binary classification corresponds to $L = 2$.

— $\mathbf{x} = (x_1, \ldots, x_p)^\top \in \mathcal{X} \subset \mathbb{R}^p$ is one observation, one example, one point, one sample.

— $\mathcal{D}_n = \{(\mathbf{x}_i, y_i), i = 1, \ldots n\}$ is the full dataset with the samples and their labels.

— We consider a probabilistic model governing random variables X and Y : $\forall i \in \{1, \ldots, n\}, (\mathbf{x}_i, y_i) \overset{i.i.d}{\sim} (X, Y)$.

— We seek to construct from $\mathcal{D}_n$ a function, named classifier, $\hat{f} : \mathcal{X} \mapsto \mathcal{Y}$ which to a new point $\mathbf{x}_{\text{new}}$ gives a label $\hat{f}(\mathbf{x}_{\text{new}})$.

Artificial data generation

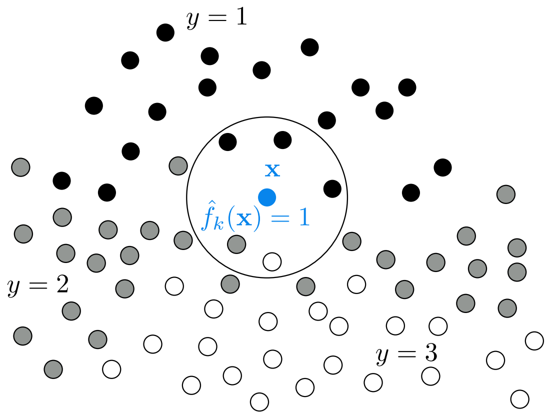
We consider two-dimensional samples, so that we can visualize them ($p = 2$).

1) Śtudy the functions rand_tri_gauss, rand_clown and rand_checkers. What do they yield ? What is the last column ? Generate 4 datasets. Use plot_2d to plot the datasets.

Intuitive approach

k-nn is an intuitive algorithm. Its principle is as follows : for each new point, $\mathbf{x}$ we first find its k-nearest neighbors in the training dataset, denoted $V_k(\mathbf{x})$ The class given to the new point is then the class which is the most represented in $V_k(\mathbf{x})$. An illustration is given in Figure 1 for $L = 3$.

1) propose an adaptation of this algorithm to regression, that is, the case where $\mathcal{Y} = \mathbb{R}$.



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

FIGURE 1 – Exemple of k-nn for $k = 5$ and $k = 11$, with $L = 3$ classes represented in black ($y = 1$), grey ($y = 2$) and white ($y = 3$). For $k = 5$ (left) we predict black at the new point $\mathbf{x}$, while for $k = 11$ (right) we predict grey.

Formal approach

We first choose a distance $d : \mathbb{R}^p \times \mathbb{R}^p \mapsto \mathbb{R}$. For a new point $\mathbf{x}$, we define the k-nearest neighbors $V_k(\mathbf{x})$ in the sense of this distance. We can then proceed as follows : for each $\mathbf{x} \in \mathbb{R}^d$ and each $i = 1, \ldots, n$, let $d_i(\mathbf{x})$ be the distance between $\mathbf{x}$ and $\mathbf{x}_i$: $d_i(\mathbf{x}) = d(\mathbf{x}_i, \mathbf{x})$. We define the first statistic of rank $r_1(\mathbf{x})$ as the index of the nearest neighbor of $\mathbf{x}$ amongst $\mathbf{x}_1, \ldots, \mathbf{x}_n$, that is,

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

By recursion, we can define $r_k(\mathbf{x})$ for any $1 \le k \le n$:

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

The set of k-nearest neighbors of $\mathbf{x}$ is then : $V_k(\mathbf{x}) = \{\mathbf{x}_{r_1}, \ldots, \mathbf{x}_{r_k}\}$. Finally, the decision function to classify $\mathbf{x}$ is a vote by majority, solving :

$$\hat{f}_k(\mathbf{x}) \in \operatorname*{arg\,max}_{y \in \mathcal{Y}} \left(\sum_{j=1}^{k} \mathbb{1}_{\{y_{r_j} = y\}} \right). \tag{2}$$

The module sklearn.neighbors of scikit-learn (*cf.* `http://scikit-learn.org/stable/modules/neighbors.html`) implements the algorithms based on nearest neighbors.

2) Fill the KNNClassifier to reimplement the decision function described above. Check your predictions by comparing them to the results of KNeighborsClassifier de scikit-learn, on the toy datasets introduced above.

For quicker computation, you will now use the functions from scikit-learn instead of your implementation.

3) Run this algorithm on the generated datasets, using the classical Euclidean distance $d(\mathbf{x}, \mathbf{v}) = \|\mathbf{x} - \mathbf{v}\|_2$.
4) Vary k. What happens when $k = 1$? $k = n$? Plot these cases on one dataset. When is the frontier simple ? complicated ?
5) What is the fraction of errors on your training data when $k = 1$? and on test data ?
6) Plot the error curves as a function of k on one of the datasetsfor n taking values 100, 500 and 1000. What is the best k ? Is it the same for all datasets ? Be careful to evaluate the error on the testing data. You can use the class ErrorCurve.
7) What are the pros and cons of this method ?
8) Apply this method the the DIGITS dataset with different choices of $k \geq 1$. Refer to http://scikit-learn.org/stable/_downloads/plot_digits_classification.py to load the data.
9) Compute the confusion matrix $(\mathbb{P}\{Y = i, C_k(X) = j\})_{i,\, j}$ associated to your classifier C_k. Refer to http://scikit-learn.org/stable/auto_examples/plot_confusion_matrix.html.
10) Propose a method to choose k and implement it. You can use LOOCurve.
11) A popular variant is to use weights for the j-th neighobr : $e^{-d_j^2(\mathbf{x})/h}$ (for a parameter h controlling the level of weighting) : we replace Equation (2) by :

$$\hat{f}_k(\mathbf{x}) \in \underset{y \in \mathcal{Y}}{\arg\max} \left(\sum_{j=1}^{k} \exp\left(-\frac{d_j^2(\mathbf{x})}{h} \right) \mathbb{1}_{\{y_{r_j} = y\}} \right) . \tag{3}$$

Implement this version in your KNNClassifier and compare to scikit-learn (passing the weights function as a parameter to the constructor of KNeighborsClassifier). You could get inspiration from _weight_func of scikit-learn : https://github.com/scikit-learn/scikit-learn/blob/master/sklearn/neighbors/tests/test_neighbors.py. Test the impact of h on the classification frontiers.

- To go further -

Global details available at [HTF09, Chapitre 13]. For a theoretical understanding of the method, refer to [DGL96, Chapitre 11], and for the limits of the method when $k = 1$, http://certis.enpc.fr/%7Edalalyan/Download/DM1.pdf. Finally, for algorithmic considerations, one can read http://scikit-learn.org/stable/modules/neighbors.html#brute-force and following paragraphs.

Références

[DGL96] L. Devroye, L. Györfi, and G. Lugosi. *A probabilistic theory of pattern recognition*, volume 31 of *Applications of Mathematics (New York)*. Springer-Verlag, New York, 1996. 3

[HTF09] T. Hastie, R. Tibshirani, and J. Friedman. *The elements of statistical learning*. Springer Series in Statistics. Springer, New York, second edition, 2009. http://www-stat.stanford.edu/~tibs/ElemStatLearn/. 3