2DI70 - Statistical Learning Theory

Nearest Neighbors Classification and
Handwritten Digit Classification

1 Introduction

Sometimes, simple ideas can be surprisingly good. This is the case with one of the oldest, but still rather popular *learning rule*, known as the *k-nearest neighbor* rule (abbreviated k-NN in this document). Consider the setting of supervised learning. Suppose you have a training data set $\{X_i, Y_i\}_{i=1}^n$, where $X_i \in \mathcal{X}$ and $Y_i \in \mathcal{Y}$, where $\mathcal{X}$ should be a metric space (that is, a space endowed with a way to measure distances). As usual, our goal is to learn a prediction rule $f : \mathcal{X} \to \mathcal{Y}$ that is able to do "good" predictions on unseen data.

The idea of k-NN is remarkably simple. Given a point $x \in \mathcal{X}$ for which we want a prediction, we simply look for the k "closest" points in the training set and make a prediction based on a majority vote (classification) or average (regression) of the neighbor labels. That is as simple as that. Computationally this might seem cumbersome, particularly for large datasets. But one can use clever computational tricks to ensure this can be done quickly.

In this assignment, which is divided in two parts, you will: (i) get a first-hand experience with this method by implementing it and choosing the "right" set of tunable parameters in a sound way; (ii) analyze the performance of this method in some generality and get a better understanding why it is sensible.

To make the explanation more concrete let us consider the problem of handwritten digit classification (which is the topic of part I): given a low resolution image of a handwritten digit we would like to classify it as one of the digits in $\{0, 1, \dots, 9\}$. More specifically our images have 28x28 pixels, each pixel taking values in $\{0, 1, 2, \dots, 255\}$. Therefore $\mathcal{X} = \{0, 1, \dots, 255\}^{28\times 28}$ and $\mathcal{Y} = \{0, 1, \dots, 9\}$.

2 The k-NN rule

Let $d : \mathcal{X} \times \mathcal{X} \to [0, +\infty)$ be a metric[1] in $\mathcal{X}$. Let $x \in \mathcal{X}$ be an arbitrary point in $\mathcal{X}$. Given x let us reorder each pair of the training data as

$$\left(X_{(1)}(x), Y_{(1)}(x)\right),\ \left(X_{(2)}(x), Y_{(2)}(x)\right), \dots, \left(X_{(n)}(x), Y_{(n)}(x)\right)\ ,$$

so that

$$d(x, X_{(1)}(x)) \le d(x, X_{(2)}(x)) \le \cdots \le d(x, X_{(n)}(x))\ .$$

[1]A metric or distance is a function that must satisfy the following properties: (i) $\forall x \in \mathcal{X}\ d(x,x) = 0$; (ii) $\forall x, y \in \mathcal{X}\ d(x,y) = d(y,x)$ (symmetry); (iii) $\forall x, y, z \in \mathcal{X}\ d(x,y) \le d(x,z) + d(z,y)$ (triangle inequality).

Note that the ordering depends on the specific point x (hence the cumbersome notation), and might not be unique. In that case we can break ties in some pre-defined way (e.g., if to points are at equal distance from x the point that appears first in the original dataset will also appear first in the ordered set). The k-NN rule (for classification) is defined as

$$\hat{f}_n(x) = \arg\max_{y \in \mathcal{Y}} \left\{ \sum_{i=1}^{k} \mathbf{1}\left\{ Y_{(i)}(x) = y \right\} \right\} \ .$$

In other words, just look among the k-nearest neighbors and choose the class that is represented more often. Obviously, there might be situations where two (or more) classes appear an equal number of times. In such situations one can break these ties according to a pre-specified rule (e.g., take the most common class that appears first in $\mathcal{Y}$).

The performance of the method described above hinges crucially on the choice of two parameters: k, the number of neighbors used for prediction and; $d : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$, the distance metric used to define proximity of two points in the feature space. There are many possible choices for d, and a naïve but sensible starting point is to consider the usual Euclidean distance: if $x, y \in \mathbb{R}^d$ then the Euclidean distance is simply given by $\sqrt{\sum_{i=1}^{d}(x_i - y_i)^2}$.

3 The MNIST dataset

This MNIST dataset[2] is a classical dataset to demonstrate machine learning methods, and is still used often as a benchmark to compare methodologies. This dataset is provided as *comma-separated value* (csv) files in CANVAS. The training set `MNIST_train.csv` consists of 60000 images of handwritten digits and the corresponding label (provided by a human expert). The test set `MNIST_test.csv` consists of 10000 images of handwritten digits and the corresponding labels (this file will be provided only closer to the assignment deadline). In addition, in CANVAS you will also find two smaller training and test sets, `MNIST_train_small.csv` (3000 examples) and `MNIST_test_small.csv` (1000 examples). These will be used for a large part of the assignment, to avoid the struggles associated with large datasets and to test out your implementations.

The format of the data is as follows: each row in the `.csv` file has 785 entries and corresponds to a single example. The first entry in the row is the "true" label, in $\mathcal{Y} = \{0, 1, \ldots, 9\}$ and the 784 subsequent entries encode the image of the digit – each entry corresponding to a pixel intensity, read in a lexicographical order (left-to-right then top-to-bottom). Pixel intensities take values in $\{0, 1, \ldots, 255\}$. The Matlab function `showdigit.m` in CANVAS will take as input a row of this data and display the corresponding digit image. Figure 3 shows several examples from `MNIST_train.csv`.

Ultimately the goal is to minimize the probability of making errors. For the purposes of this assignment we will use simply the 0/1 loss. This means that the empirical risk is simply the average number of errors we make. If $\{X'_i, Y'_i\}_{i=1}^{m}$ denotes the pairs of features/labels

[2]See `http://yann.lecun.com/exdb/mnist/` for more details and information.

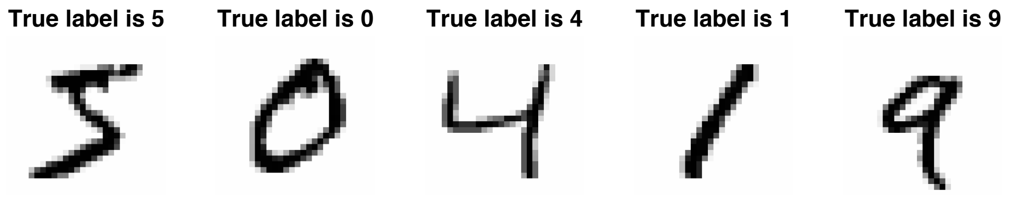

Figure 1: First five examples from `MNIST_train.csv` and the corresponding labels provided by a human expert.

in a test set and $\{\hat{Y}_i'\}_{i=1}^m$ denotes the corresponding inferred labels by the k-NN rule then the empirical risk on the test set is given by $\frac{1}{m}\sum_{i=1}^m \mathbf{1}\left\{\hat{Y}_i' \neq Y_i'\right\}$.

4 PART I - Computational Assignment

The goal of the first part of the assignment is to implement "from scratch" a nearest neighbor classifier. This means you will not be allowed to use existing implementations and libraries, and should make only use of standard data structures and mathematical operations. The only "exception" is that you are allowed to use a sorting subroutine (i.e., a function that, given a vector of numerical values, can sort them in ascending order and give the correspondent reordered indexes). The rational for the above restrictions is for you to experience what are the critical aspects of your implementation, and understand if it is scalable to big datasets. For this assignment you are allowed to use any language or command interpreter (preferably a high-level language, but not necessarily so). You will not be judged on your code, but rather on your choices and corresponding justification.

You should prepare a report (in English) and upload it via CANVAS. The report should be self-contained and document in a clear and concise way the problem, your solution and results. It is import that for you to have a critical attitude, and comment on the your choices and results. The report for part I should be submitted as a single .pdf file. In addition, submit a separate file with the code/script you used (you will not be graded on this, but if needed we might look at it to better understand the results in your report). In your report you should do the following experiments and answer the questions below.

a) Write down you implementation of k-NN neighbors (using as training data `MNIST_train_small.csv`) and report on its accuracy to predict the labels in both the training and test sets (respectively `MNIST_train_small.csv` and `MNIST_test_small.csv`). For this question use the simple Euclidean distance. Make a table of results for $k \in \{1, \ldots, 20\}$, plot your the empirical training and test loss as a function of k, and comment on your results.

b) Obviously the choice of the number of neighbors k is crucial to obtain good performance. This choice must be made WITHOUT LOOKING at the test dataset. Although one can use rules-of-thumb, a possibility is to use cross-validation. *Leave-One-Out Cross-Validation* (LOOCV) is extremely simple in our context. Implement LOOCV to estimate the risk of the k-NN rule for $k \in \{1, \ldots, 20\}$. Report these LOOCV risk estimates[3] on a table and plot them as well the empirical risk on the test dataset (that you obtained in (a)). Given your results, what would be a good choice for k?

c) Obviously, the choice of distance metric also plays an important role. Consider a generalization of the Euclidean distance, namely L_p distances (also known as Minkowski distances). For $x, y \in \mathbb{R}^d$ define

$$d_p(x, y) = \left(\sum_{i=1}^{d} |x_i - y_i|^p \right)^{1/p} \quad ,$$

where $p \geq 1$. Use leave-one-out cross validation to simultaneously choose a good value for k and $p \in [1, 15]$.

d) **(this question is more open)** Building up on your work for the previous questions suggest a modification of the distance metric or some pre-processing of the data that you consider appropriate to improve the performance of the NN method. Note that, any choices you make should be done solely based on the training data (that is, do not optimize the performance of your method on the test data). Clearly justify ALL the choices you made and describe the exact steps you took. Someone reading your report should be able to exactly replicate your results.

Now that you implemented and tested your methodologies in a smaller scale, let us see how these methods scale to the real problem. For the remaining questions you will use the full MNIST training and test sets.

e) Make use of either the Euclidean distance or d_p with your choice of p in part (c) (use only one or the other). Determine a good value for k using leave-one-out cross validation when considering the full training set (60000 examples). Was your implementation able to cope with this large amount to data? Did you have to modify it in any way? If so, explain what you did. What is the risk estimate you obtain via cross-validation?

[3] Recall that these estimates use only the information on the training dataset.

f) **(it is only possible to answer this question after I provide you the file `MNIST_test.csv`)** Using the choice of k in part (e) compute the loss of your method on the test set provided. How does this compare with the cross-validation estimate you computed in (e)? Would you choose a different value for k had you been allowed to look at the test dataset earlier?

g) **Bonus question:** each training example is currently a high-dimensional vector. A very successful idea in machine learning is that of dimensionality reduction. This is typically done in an unsupervised way - feature vectors are transformed so that most information is preserved, while significantly lowering their dimension. A possibility in our setting is to use Principal Component Analysis (PCA) to map each digit image to a lower dimensional vector. There is an enormous computational advantage (as computing distances will be easier) but there might be also an advantage in terms of statistical generalization. Use this idea for this problem and choose a good number of principal components to keep in order to have good accuracy (again, this choice should be solely based on the training data). Document clearly all the steps of your procedure. In this question you are allowed to use an existing implementation of PCA or related methods.