EE-411, HomeWork 2 : Classifying digits & learning theory

This homework involves some coding, with the language of your choice. Present your results with graphs, plots, and data.

Submission instruction : The homework must be uploaded on Moodle and the name of the submission must be *name_surname_SCIPER_HW2.ipynb*. All the results must be included in a single notebook. We recommend you to use Google Colab.

1 Classifying digits with Scikit-learn

We are going to use the UCI ML handwritten digits datasets. Using Python and Scikit-learn, they can be downloaded writing : *from sklearn.datasets import load_digits* and then *digits = load_digits()* . This dataset contains $n_{tot} = 1797$ images, each a set of $d = 8 \times 8 = 64$ pixels, with each pixel being an integer between 0 and 16. The image represents handwritten digits, from 0 to 9. Our goal will be to train a classifier to recognize if the digits are representing *even* numbers or *odd* ones [1] .

1) Import and prepare the data :

— Import the dataset using *load_digits(return_X_y=True)*.

— The images are automatically imported as vectors $\mathbf{x}_i, (i = 1, \dots, n)$, but so far the labels are numbers between 0 and 9. Change the targets y_i to correspond to 0/1 for even and odd numbers respectively.

— Split the data into a "training" & a "testing" set (using roughly 60% and 40%).

— Check that these two subsets have roughly the same proportion of labelled numbers.

2) Logistic Regression :

— Using *linear_model.LogisticRegression*, train a classifier based on Logistic regression to decide if an image represents an odd or an even digit, comparing ℓ_2 and ℓ_1 penalty (you can fix the penalty just by setting the corresponding parameter when you call *LogisticRegression()* to generate the method).

— Using *GridSearchCV* with *n_splits=5*, as seen during the exercise sessions, perform cross validation to fix the optimal hyperparameters (the regularization constant for the penalty) studying the interval $\lambda \in [10^{-1}, 10^4]$ on a logarithmic scale. Then, with the resulting classifier, compute the Accuracy on the Test set, i.e. the rate of correctly classified images.

3) Ridge and Hinge : Repeat these operations for Ridge and Hinge (also called SVM) :

— Use *linear_model.RidgeClassifier* to train a classifier based on Ridge, studying $\lambda \in [10, 10^7]$;

— Use *svm.LinearSVC* to train a classifier based on SVM/Hinge, studying $\lambda \in [1, 10^4]$.

— Compute the accuracy of the classifiers on the Test set as before.

4) Random Forest : Repeat the operations described in point 2 for a random forest classifier :

— Use *sklearn.ensemble.RandomForestClassifier* to train a random forest classifier. Study the tuning of the hyperparameter *n_estimators* with cross validation as in previous points, fix the search interval to be *n_estimators* $\in [10, 10^4]$.

— Compute the accuracy of the classifiers on the test set as before.

5) Random Feature :

— An alternative is to consider the random feature classifier. This is done as follows : first we transform the vector $\mathbf{x}$ containing for each image into another vector $\mathbf{u}$ using the following transformation :

$$\mathbf{u}_i = \frac{1}{\sqrt{D}} \sigma(F\mathbf{x_i}) \text{ with, } \sigma(x) = 1/(1 + e^{-x}) \tag{1}$$

1. Things to keep in mind : a) Be cautious when you fix the parameters in a sklearn classifier (e.g. *penalty*, *dual*, *solver*). Not all combinations can be used ! b) Some values of the regularization parameters sometimes lead to a slow convergence of the classifier. It's up to you to do a good compromise, choosing the number *max_iter* of maximum iterations taken for the solvers to converge. c) Warning : while the Ridge classifier uses the usual regularization parameter λ, called α on Scikit-learn, all the other classifiers take the inverse of regularization strength C as a parameter. So remember to invert it if you want to compare the regularization paths ! d) Each CV search should take no more than 5 minutes. If your code is running for much more than that, it means that you have to change some parameters ! (either you are doing something wrong or you are trying to be too precise) e) If you get curious, there are other things you may want to try. For instance scaling and normalizing the data (using sklearn convenient *preprocessing.StandardScale*) and observe how it affects each of the classifiers. You may even try the Kernel methods we discussed during the lectures !

Here the $\mathbf{x_i}$ are vectors of dimension d that contain the values of the pixels for a given image, F is a random Gaussian matrix $F_{D\times d}$, where each element is a Gaussian random number of variance $1/d$ (this can be created using *random.normal* in numpy). The function $\sigma(\cdot)$ is applied on each component. This transformation maps all the original vector $\mathbf{x} \in \mathbb{R}^d$ into seemingly "randomish" vectors $\mathbf{u} \in \mathbb{R}^D$.

— Perform the same classification as before with ℓ_2-Logistic regression, now using the vector $\mathbf{u}$ *instead* of the vector $\mathbf{x}$. How does the accuracy behave as D increases ? (try with $D = 2d, D = 4d, D = 8d \ldots$). We suggest to use $\lambda \in [10^{-7}, 10^{-2}]$ for the CV analysis.

2 Statistical Learning with Nearest-Neighbors

Here, we are going to follow the steps we took in lecture 3 to understand the limitation of learning with nearest neighbors. We shall assume that data are generated from an unknown distribution $\mathbb{P}(\mathbf{X}, Y)$, with $\mathbf{X} \in \mathbb{R}^d$, and $Y \in \mathbb{R}$, with variance $\text{var}[Y] = \sigma^2$. Our task is to find a function $f(\mathbf{x})$ that "learns" to output the corresponding y.

1. First, we assume that we are a "genie" that knows the distribution $\mathbb{P}(\mathbf{X}, Y)$. Show that the "best" estimator $f(X)$ in terms of minimizing the expected population risk $\mathbb{E}_{\mathbf{X},Y}[(f(\mathbf{X}) - Y)^2]$ is given by $f_{\text{Bayes}}(X) = \mathbb{E}(Y|\mathbf{X})$. This will serve as our reference point.
2. We are given a set of point $\mathbf{x}_{1,\ldots,n}$ and their labels $y_{1,\ldots,n}$ and consider the KNN function (where $\mathcal{N}_k(\mathbf{X})$ is the set of k-nearest neighbors of the point $\mathbf{X}$) :

$$f_{\text{KNN}}(\mathbf{X}) = \frac{1}{k} \sum_{i \in \mathcal{N}_k(\mathbf{X})} y_i \tag{2}$$

Show that the difference between the expected population risk (where we also average over the labels Y_i, $i = 1, \ldots, n$) and the Bayes population risks follows a Bias-Variance decomposition given by

$$\mathcal{R}_{\text{KNN}} - \mathcal{R}_{\text{Bayes}} = \mathbb{E}_{\{Y^{(i)}\},\mathbf{X},Y}\left[(Y - f_{\text{KNN}}(\mathbf{X}))^2 - (Y - f_{\text{Bayes}}(\mathbf{X}))^2\right] = b^2 + \text{v} \tag{3}$$

$$\text{with} \quad b^2 = \mathbb{E}_{\mathbf{X}}\left[\left(\frac{1}{k}\sum_{i \in \mathcal{N}_k(\mathbf{X})} f_{\text{Bayes}}(\mathbf{x}_i) - f_{\text{Bayes}}(\mathbf{X})\right)^2\right] \quad \text{and} \quad \text{v} = \frac{\sigma^2}{k} \tag{4}$$

3. To bound the bias term, we make the following assumptions : (i) we suppose that the regression function $f_{\text{Bayes}}(\mathbf{x})$ is is L-Lipschitz, and (ii) that the $\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_n$ are are evenly spaced on a d-dimensional unit (hyper-)cube of volume 1. Show that this leads to

$$\mathcal{R}_{\text{KNN}} - \mathcal{R}_{\text{Bayes}} \leq L^2 \left(\frac{k}{n}\right)^{\frac{2}{d}} + \frac{\sigma^2}{k} \tag{5}$$

Pause to think about the relationship between our results and L, k, n, and d. Does this dependence qualitatively make sense ?

From now on, for the rest of the homework, we shall assume $L = \sigma = 1$ for simplicity.

4. To get an idea of what the terms in the bound look like and what the best k might be, plot the individual terms in the bound for the excess risk and the bound itself for $n = 100$ and $d = 1$ as a function of k to visualize the trade-off. Find the value of k that minimizes the excess risk. Repeat for different values of n and d.
5. For general n and d, compute analytically the best k, denoted $k_*(n, d)$, as a function of n and d. How does $k_*(n, d)$ changes with d and n ? Does this relation with respect to n make sense ?
6. Plug this value of $k_*(n, d)$ in the bound (5). Obtain an optimal estimate for the bound on the excess squared risk for k-nearest-neighbor regression in terms of n and d. Pause to think about the result.
7. To understand and get an intuition on the meaning of this result, plot the number of samples n required to achieve excess squared risk of 0.1 as a function of the dimension d. Do you find anything worrying about this plot ?