ELEN0062 - Introduction to machine learning
Project 1 - Classification algorithms

October 2nd, 2024

In this first project, you will get accustomed with some classical machine learning algorithms and concepts, such as under and overfitting. For each algorithm, we ask you to deliver a separate Python script. Make sure that your experiments are reproducible (e.g., by manually fixing random seeds). Add a *brief* report (PDF format, roughly 4 pages without the figures) giving your observations and conclusions. Each project must be carried out by groups of *two* to *three students* and submitted to Gradescope[1] before *October 27, 23:59 GMT+2*. There will be two projects to submit to: one for your Python scripts and one for your report. Note that attention will be paid to how you present your results. Careful thoughts in particular - but not limited to - should be given when it comes to plots. Values for the hyperparameters of all the methods tested are given below to answer the various questions. However, feel free to deviate from these values to support your discussions if you feel it is appropriate.

Files

You are given several files, among which are `data.py` and `plot.py`. The first one provides a function `make_dataset` that generates the binary classification dataset that will be used for the experiments. Unless specified otherwise, we ask you to keep all parameters of the function to their default values (except for the `npoints` argument). This will generate a dataset with two real input variables. The examples are sampled from two circular Gaussian distributions with the same diagonal covariance matrices, centered at $(+1.5, +1.5)$ for the negative class and $(-1.5, -1.5)$ for the positive class. Examples of the negative class are three times more frequent than examples from the positive class. See Figure 1 for an example dataset. For each experiment, you will generate 3000 samples: the first 1000 will be used as the training set and the remaining ones as the testing set.

The file `plot.py` contains a function which depicts the decision boundary of a trained classifier. Note that you should use a dataset independent of the learning set to visualize the boundary.

Only modify the other files, which you must archive before submitting.

1 Decision tree (`dt.py`)

In this section, we will study decision tree models (see `DecisionTreeClassifier` from `sklearn.tree`). More specifically, we will observe how the model's complexity affects the classification boundary. To do so, we will build several decision tree models with `max_depth` values of 1, 2, 4, 8, and `None` (and all other parameters set to their default values). Answer the following questions in your report.

1. Observe how the decision boundary is affected by tree complexity.
   (a) Illustrate and explain the decision boundary for each hyperparameter value.
   (b) Discuss when the model is clearly underfitting/overfitting and detail your evidence for each claim.

[1] `https://www.gradescope.com`. The course entry code is EVGWRX. Please register with your official ULiège email address.

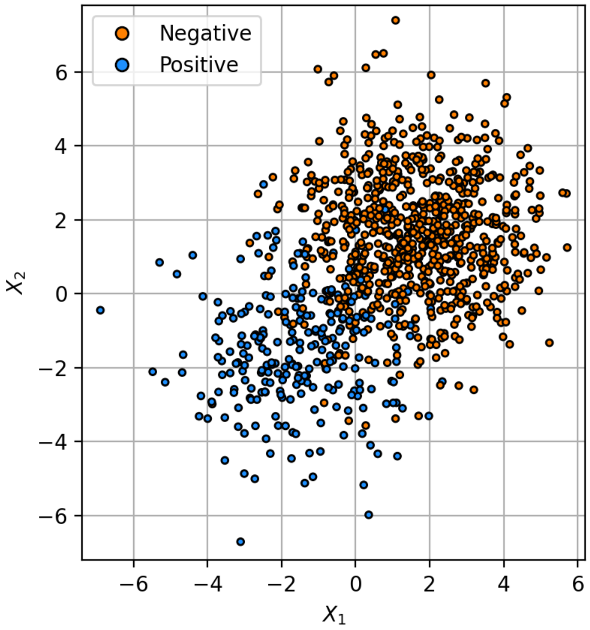

Figure 1: One randomly generated dataset of 1000 points.

   (c) Explain why the model seems more confident when `max_depth` is the largest.

2. Report the average test set accuracies over five generations of the dataset, along with the standard deviations, for each value of `max_depth`. Briefly comment on them.

2 K-nearest neighbors (`knn.py`)

In this section, we will study nearest neighbors models (see the `KNeighborsClassifier` class from `sklearn.neighbors`) on a dataset generated by `make_dataset`. More specifically, we will observe how model complexity impacts the classification boundary. To do so, we will build several nearest neighbor models with `n_neighbors` values of $1, 5, 50, 100$, and 500. Answer the following questions in your report.

1. Observe how the decision boundary is affected by the number of neighbors.

   (a) Illustrate the decision boundary for each value of `n_neighbors`.

   (b) Comment on the evolution of the decision boundary with respect to the number of neighbors.

2. Report the average test set accuracies over five generations of the dataset, along with the standard deviations, for each value of `n_neighbors`. Briefly comment on them.

3 Perceptron (`perceptron.py`)

In this section, we will study a simple classification model based on a single neuron, also called a perceptron classifier. A single neuron computes a (numerical) prediction using the following formula:

$$\hat{f}(\mathbf{x};\mathbf{w}) = \sigma(w_0 + \sum_{j=1}^{p} w_j x_j), \tag{1}$$

where $\mathbf{x} = (x_1, \ldots, x_p)$ denotes the p input variables, σ is an activation function and $\mathbf{w} = [w_0, w_1, \ldots, w_p] \in \mathbb{R}^{p+1}$ is the vector of trainable parameters. Let us assume a binary classification problem with the output class defined in $\{0, 1\}$. The output of the neuron is turned into a class prediction using the following rule:

$$\hat{y}(\mathbf{x};\mathbf{w}, d_{th}) = \begin{cases} 1, & \text{if } \hat{f}(\mathbf{x};\mathbf{w}) \geq d_{th}, \\ 0, & \text{otherwise.} \end{cases} \tag{2}$$

where d_{th} is some decision threshold. In this project, we propose to use a sigmoid function as the activation function:

$$\sigma(x) = \frac{1}{1 + e^{-x}}. \tag{3}$$

In this case, the output of the neuron will always be between 0 and 1 and can be interpreted as an estimate of the class conditional probability of class 1, i.e.:

$$\hat{P}(Y = 1|\mathbf{x};\mathbf{w}) = \hat{f}(\mathbf{x};\mathbf{w}). \tag{4}$$

A natural decision threshold is therefore $d_{th} = 0.5$ that you will use in all experiments.

To train the model, one usually finds $\mathbf{w}^*$ that minimizes some loss function. Given the interpretation of $\hat{f}$ as the class conditional probability, we will use here the cross-entropy loss defined as:

$$\mathcal{L}(\mathbf{x}, y, \mathbf{w}) = -y \log(\hat{f}(\mathbf{x};\mathbf{w})) - (1 - y) \log(1 - \hat{f}(\mathbf{x};\mathbf{w})), \tag{5}$$

where $\mathbf{x}$ is an instance and $y \in \{0, 1\}$ its observed class. We will minimise the sum of the loss over the learning set in the simplest way, i.e., using stochastic gradient descent. The idea is to iterate over all learning pairs $(\mathbf{x}, y)$ and update the parameter vector $\mathbf{w}$ according to the following rule:

$$\mathbf{w} \leftarrow \mathbf{w} - \eta \nabla_{\mathbf{w}} \mathcal{L}(\mathbf{x}, y, \mathbf{w}) \tag{6}$$

where η is an hyperparameter called the learning rate, which is introduced to prevent overfitting. Another hyperparameter of the algorithm is the number of epochs, T, i.e., the number of passes over the entire learning set.

Answer the following questions in your report

1. Derive the mathematical expression of the gradient $\nabla_{\mathbf{w}} \mathcal{L}(\mathbf{x}, y, \mathbf{w})$, which is necessary for implementing the stochastic gradient descent algorithm.

2. Implement your own perceptron estimator according to the above description and following the scikit-learn convention (`http://scikit-learn.org/dev/developers/`). *Suggestion: Fill in the class whose template is given in `perceptron.py`.* Briefly explain and motivate your implementation choices in the report (e.g., explain how you initialize the weights and in which order you are you going through the learning examples in each epoch).

3. Using the same sizes for the learning and test sets (i.e., respectively 1000 and 2000) as in previous sections:

   (a) Illustrate the decision boundary of the perceptron classifier for different values of the learning rate η. Fix the number of epochs T to 5 and use the values of η in $\{1 \times 10^{-4}, 5 \times 10^{-4}, 1 \times 10^{-3}, 1 \times 10^{-2}, 1 \times 10^{-1}\}$.

   (b) Comment on the evolution of the decision boundary with respect to η.

4. Report the average test set accuracies over five generations of the dataset, along with the standard deviations, for each value of η. Briefly comment on them.

4 Method comparison (`mc.py`)

We would like now to compare the three methods for this specific dataset. Since all methods depend on hyperparameters, we need to find a way to tune them. Below, you can tune each of them by exploring the set of candidate values suggested in the previous sections.

Answer the following questions in your report:

1. Explain a way to tune the value of `max_depth`, `n_neighbors`, and η using only the learning set.

   cross-validation

2. Implement this method and use it to compute the average test set accuracies over at least five generations of the dataset, along with the standard deviations, for the tuned decision tree, k-nearest neighbors methods, and perceptron.

3. Repeat the previous experiment but this time by adding 200 new input features to the dataset that are pure Gaussian noise and thus not related to the class. You can use the argument `n_irrelevant` of the function `make_dataset` to add them to the generated dataset. These features are all centered at 0 and have the same standard deviation as the original two features.

4. Discuss the performance and the ranking of the different methods in the two settings, i.e., without and with the noisy features. Try to explain the possible differences between these two settings.