ELEN0062 - Introduction to machine learning
Project 1 - Classification algorithms

September 22nd, 2021

In this first project, you will get accustomed with some classical machine learning algorithms and concepts, such as under- and over-fitting. 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 3 pages without the figures) giving your observations and conclusions. Each project must be carried out by groups of *two* to *three students* and submitted as a `tar.gz` file on Montefiore's submission platform (`http://submit.montefiore.ulg.ac.be`) before *October 17, 23:59 GMT+2*. 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.

Files

You are given several files, among which `data.py` and `plot.py`. The first one generates binary classification datasets with two real input variables. More precisely, the examples are sampled from two circular gaussian distributions with the same covariance matrices, centered at $(+1.5, +1.5)$ for the negative class and $(-1.5, -1.5)$ for the positive class. In the following, you will work with `make_unbalanced_dataset`, where the negative class is three times more present, an example of which is displayed in Figure 1. You can generate datasets of 3000 samples. The first 1000 will be used as training set and the remaining ones as testing set, unless specified explicitly otherwise. The second file contains a function which depicts the decision boundary of a trained classifier. Note that you should use a dataset independent of the training set to visualize the boundary and make relevant observations.

The other files must be completed and archived together with the report.

1 Decision tree (`dt.py`)

In this section, we will focus on decision tree models (see the `DecisionTreeClassifier` class from `sklearn.tree`). More specifically, we will observe how model complexity impacts the classification task. For that purpose, we will build several decision tree models with `min_samples_split` values of $2, 8, 32, 64, 128, 500$. Answer the following questions.

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

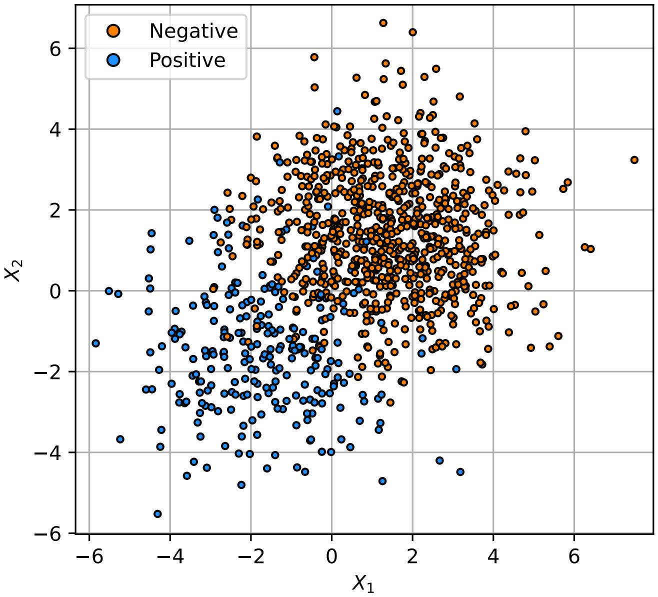

Figure 1: Unbalanced dataset.

(c) Explain how prediction confidence is computed in a decision tree. Why does the model seem more confident when `min_samples_split` is the smallest.

2. Based on what you have observed, how could you visually guess that:

   (a) Boundaries are associated to decision trees?

   (b) The problem is unbalanced?

3. Report the average test set accuracies (over five generations of the dataset) along with the standard deviations for each value of `min_samples_split`. Briefly comment on them.

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

In this section, we will focus on nearest neighbors models (see the `KNeighborsClassifier` class from `sklearn.neighbors`). Similarly to decision trees, we will observe how model complexity impacts the classification task. 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.

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. Use a ten-fold cross validation strategy to optimize the value of the `n_neighbors` parameter:

   (a) Explain your methodology.

   (b) Report the optimal value of `n_neighbors` and the mean accuracy obtained over the ten folds. Do they corroborate your decision boundary-based intuition? Justify your answer.

3. Observe the evolution of the optimal value of `n_neighbors` with respect to the size of the training set. To do so:

   (a) Compute and plot the evolution of mean test accuracies (on a fixed test set of size 500), for every possible number of neighbors, for the following training set sizes N: $\{50, 150, 250, 350, 450, 500\}$, over ten generations of the dataset.

   (b) Plot the optimal value of `n_neighbors` with respect to the training set size, for all sizes studied in the previous point.

   Briefly comment on what you observe. What can you say about the relationship between k and N?

3 Logistic regression (`logistic_regression.py`)

Despite its name, logistic regression is a classification method. In the case of a binary classification problem, it estimates the conditional probability of the positive class with the following expression:

$$P(Y = +1|\mathbf{x_i}; w_0, \mathbf{w}) = \frac{1}{1 + \exp(-w_0 - \mathbf{w}^T \mathbf{x}_i)} \tag{1}$$

where $\theta = (w_0, \mathbf{w}^T)$ are the trainable parameters. The use of the sigmoid function $1/(1 + \exp(-x))$ ensures that the estimated probability will be in the unit range $[0, 1]$.

To train this model, one usually finds θ^* that minimizes the negative log-likelood loss function over the training set computed as:

$$\mathcal{L}(\theta) = -\frac{1}{N} \sum_{i=1}^{N} \log P(Y = y_i|\mathbf{x}_i, \theta) \tag{2}$$

In our case, we will learn θ in the simplest possible way: with a batch-mode vanilla gradient descent. The idea is to iteratively update the parameter θ according to the following rule:

$$\theta^{(t+1)} \leftarrow \theta^{(t)} - \eta \nabla_\theta \mathcal{L}(\theta) \tag{3}$$

where η is an hyper-parameter called the learning rate, which is introduced to prevent overfitting.

In the present case, the gradient of the loss function is given by

$$\nabla_\theta \mathcal{L}(\theta) = \frac{1}{N} \sum_{i=1}^{N} \left[P(Y = +1|\mathbf{x}_i, \theta) - y_i\right] \mathbf{x}'_i, \tag{4}$$

where $\mathbf{x}'_i = (1, x_i^{(1)}, x_i^{(2)})^T$ is the ith feature vector to which a constant 1 has been introduced as first component.

Implement your own logistic regression estimator according to the above description and following the scikit-learn convention (`http://scikit-learn.org/dev/developers/`). The estimator should have two hyper-parameters: the learning rate η and the number of gradient descent iterations to perform.
Suggestion: Fill in the template given in `logistic_regression.py`.

Answer the following questions:

1. Show that the decision boundary of a logistic regression model is linear.
2. Show that maximizing the likelihood of the observed data is equivalent to minimizing the negative log-likelihood function, i.e. that maximizing $P(D \mid \theta)$ is the same as minimizing Equation (2), where D is the learning set of input-output pairs.
3. Show that Equation (4) is indeed the gradient of the negative log-likelihood (Equation (2)).
4. How do you propose to set the initial value of θ?
5. Illustrate the decision boundary on the dataset and briefly comment on the results. Specify the values of the hyper-parameters.
6. Report the average accuracy (over five generations of the dataset) along with its standard deviation. Specify the hyper-parameters you used.
7. Study the effect of the number of iterations (for a fixed but relevant learning rate) on both the decision boundary and the error.
8. Based on what you have observed for each model considered (DT, k-NN, and logistic regression) and on the nature of the problem, which model do you think fits best? Justify your answer.