In this first project, we ask you to write several Python scripts to answer the different questions below. One separate script is required for each of the three questions. Make sure that your experiments are reproducible (e.g., by fixing manually random seeds). Add a brief report (pdf format, 4 pages max.) giving your observations and conclusions.

Each project must be done by group of 2 students. Send a zip archive with the pdf report and python scripts before October 26, 23:59 GMT+2 to a.joly@ulg.ac.be with subject "ELEN0062][project-1]last-name1-last-name2" where last-name1 and last-name2 are the names of the members of the group.

1 Decision tree (dt.py)

With the provided function make_cross, generate a training set (X_train, y_train) of 150 samples and a testing set (X_test, y_test) of 1500 samples.

1. Build a decision tree model (sklearn.tree.DecisionTreeClassifier) on X_train, y_train. Compare visually\(^1\) the prediction of the model on X_test to the ground truth y_test. Comment your observations.

2. Observe visually the effect of the max_depth parameter on the decision frontier.

3. Evaluate the accuracy on the learning and testing sets for various values of the max_depth parameter and plot the corresponding error curve.

4. What is the best value to choose for the max_depth parameter? Justify your answer.

\(^1\)i.e., represent the decision frontier obtained by the estimator in the input space. To do that, you can use the function plot_boundary from plot.py that allows you to plot the decision surface of an estimator and a scatter plot of a set of points.

2 K-nearest neighbors (knn.py)

Re-use the same training and testing sets as in question 1.

1. Build a k-nearest neighbors model (sklearn.neighbors.KNeighborsClassifier) on X_train, y_train. Compare visually the prediction of the model on X_test to the ground truth y_test. Comment your observations.

2. Analyse visually the effect of the n_neighbors parameter on the decision frontier.

3. Evaluate the accuracy on the learning and testing sets for increasing values of the n_neighbors parameter and plot the corresponding error curve.

4. What is the best value to choose for the n_neighbors parameter? Justify your answer.
3 Stochastic gradient descent (sgd.py)

We want now to consider a linear model:

\[ y = w^T x \] (1)

Such model can be used for binary classification by coding one class with an output \( y = +1 \) and the other class with an output \( y = -1 \). A new example described by its input vector \( x' \) is then classified according to the sign of \( w^T x' \).

We propose to use stochastic gradient descent with squared loss to train this model. This algorithm minimizes the average squared loss on the training set, which is defined as:

\[ \text{Err}(w) = \frac{1}{n} \sum_{i=1}^{n} \text{Err}_i(w) \text{ with } \text{Err}_i(w) = \frac{1}{2}(w^T x_i - y_i)^2. \]

The idea of stochastic gradient descent is to go iteratively through the training examples (in random order) and for each of them to update the weights according to the following rule:

\[ w(t+1) \rightarrow w(t) - \eta_t \Delta_w E_{i_t}(w(t)), \] (2)

where \( i_t \in \{1, \ldots, n\} \) denotes the randomly chosen training example at step \( t \) and \( \Delta E_{i_t}(w(t)) \) is the gradient of \( E_{i_t} \) computed at \( w(t) \), which in the case of squared loss becomes:

\[ \Delta_w E_{i_t}(w(t)) = (w(t)^T x_{i_t} - y_{i_t}) x_{i_t}. \]

The learning rate \( \eta_t \) is a parameter of the algorithm. It can either be fixed a priori or chosen to be decreasing with \( t \). The algorithm starts from \( w(0) = 0 \) and applies the update in (2) for examples a fixed number \( t_{\text{max}} \) of times, each from a example randomly drawn (with replacement) from the training set. The final model is taken as the average over all computed vectors:

\[ \frac{1}{t_{\text{max}}} \sum_{t=1}^{t_{\text{max}}} w(t). \]

Note that the decision frontier of the linear model in (1) will always go through the origin. One simple way to relax this constraint is to add a dummy feature (i.e., a new column) of constant value 1 for all examples.

1. Implement your own stochastic gradient descent estimator according to the above description and following the scikit-learn convention (http://scikit-learn.org/dev/developers/).
   The parameters of the algorithm should be the following:
   - \( \text{n\_iter} \): the number of iterations \( t_{\text{max}} \).
   - \( \text{learning\_rate\_init} \) (\( \eta_0 \)): a parameter that defines the learning rate through the following formula:
     \[ \eta_t = \eta_0. \]
   * Suggestion: Fill in the class whose template is given in sgd.py.

2. Re-using the training and testing sets from question 1, build a linear model using your stochastic gradient algorithm on \( X_{\text{train}}, y_{\text{train}} \) and compare visually the predictions of the model on \( X_{\text{test}} \) to the ground truth \( y_{\text{test}} \). Comment your observations.

3. Imagine and implement a way to modify the input data in order to improve the performance of the model for this specific problem. Compare visually the predictions of the model on \( X_{\text{test}} \) to the ground truth \( y_{\text{test}} \). Comment your observations.
4. Evaluate the accuracy on the learning and testing sets for various values of the n_iter parameter and plot the corresponding error curve (with learning_rate_init=0.01). Comment your observations.

5. Evaluate the accuracy on the learning and testing sample set for various values of the learning_rate_init parameter and plot the corresponding error curve (with n_iter=1000). Comment your observations.

6. What is the best accuracy that you can obtained by optimizing both learning_rate_init and n_iter? What are the optimal values of these two parameters?

**Remark**

Whenever you are optimizing the hyper-parameters, you might want to average your results over several experiments in order to get averaged performance and smoother error curves.