EE-613: Machine Learning for Engineers
Practical session 1

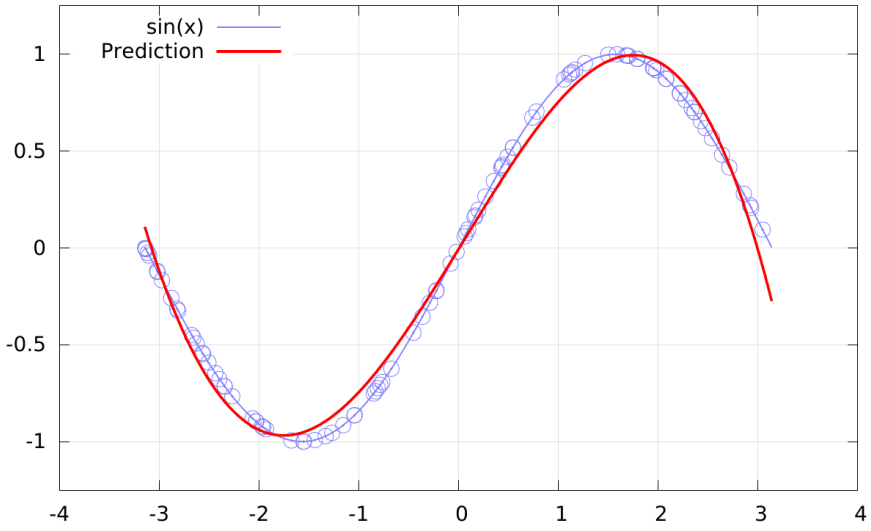
Introduction

Download the archive file, install it, and run the example with:

```
wget http://www.idiap.ch/~fleuret/files/EE613/EE613-pw1.tgz
tar zxvf EE613-pw1.tgz
cd ./EE613/pw1
./do.sh -v example
```

The `-v` option tells the script to display the resulting graph.

You should get on your screen:



Provided code

All this session is based on adding code to the `pw1.cc` file, and visualizing results through the `do.sh` script, which compile and runs `pw1`, and generate graphs using the results files.

The provided code uses `scalar_t` as a floating point type, and defines the function `random_0_to_1` to generate random numbers sampled uniformly into $[0, 1]$.

The source code implements a class `FunctionalRegressor` with the methods

```
scalar_t eval(scalar_t x);
void fit(int nb_samples, scalar_t *x, scalar_t *y);
```

This class is unusable in itself, since it does not implement the two needed virtual functions

```
virtual int nb_basis_functions();
virtual scalar_t value_basis_function(int nf, scalar_t x);
```

The class `PolynomialRegressor` implements them for the functional basis

$$x^d, d = 0, \dots, D.$$

Its constructor takes as parameters the maximum polynomial degree D.

Question 1

Use the `example` function for inspiration on how to answer this question.

Fill the function `question1` so that

1. It creates a training set of 100 samples, with x_n uniformly taken in $[0, 1]$ and

$$y_n = \begin{cases} 0 & \text{if } x_n < 0.5, \\ 1 & \text{otherwise.} \end{cases}$$

2. It successively fits polynomials of degrees 0, 1, 2, 4, 8, 16, and 32, each time computing the value of the fitted function over 1000 successive values taken in $[0, 1]$.

3. It write the results to the file `/tmp/question1.dat`, each line composed of three values: The degree of the polynomial, the x value, and the y value of that polynomial on that x.

When this is done, you can visualize the result with

```
./do.sh -v question1
```

Note that if you write a file `/tmp/question1_train.dat` with one pair of values on each line, the `do.sh` script will display these values as dots. This allows you to visualize the training samples if you want.

Question 2

Write a function

```
scalar_t test_error(int degree,
                    int nb_train_samples, int nb_test_samples);
```

which

1. Generates a training set of `nb_train_samples` samples as in question 1
2. Fits a polynomial of degree `degree`
3. Computes the average quadratic error test error over `nb_test_samples`.

Using this function, fill the function `question2` so that

1. It visits the training set sizes $16, 32, \ldots, 256$, and the degrees $0, 1, \ldots, 10$.
2. For each combination, it writes to the file `/tmp/question2.dat` one line with the number of train samples first, followed by the degree, and the test error obtained with a polynomial fitting and average through 250 runs train/test.

When this is done, you can visualize the result with

```
./do.sh -v question2
```

Question 3

Write a class

```
class HingeRegressor : public FunctionalRegressor {
  int _nb_hinges;
  scalar_t *_hinge_positions;

  virtual int nb_basis_functions();
  virtual scalar_t value_basis_function(int nf, scalar_t x);

public:
  HingeRegressor(int nb_hinges, scalar_t xmin, scalar_t xmax);
  virtual ~HingeRegressor();
};
```

which corresponds to a family of `nb_hinges` functions of the form

$$y = \begin{cases} 0 & \text{if } x_n < \alpha, \\ x - \alpha & \text{otherwise.} \end{cases}$$

with the αs taken at random uniformly between `xmin` and `xmax`.

Using this class, fill the function `question3` so that:

- It generates a training set of 100 samples with x_n uniformly taken in $[-\pi, \pi]$ and $y_n = \sin(x_n)$
- It fits a hinge-based regressor with 15 hinges
- It writes to `/tmp/question3.dat` the values of the fitted function over 1000 successive values taken in $[-\pi, \pi]$. Each line should be of the form x y.

When this is done, you can visualize the result with

```
./do.sh -v question3
```

As for question 1, if you also write a file `/tmp/question3_train.dat`, it will be displayed on the graph as a series of points.

Try the same with an additional noise taken uniformly in $[-0.1, 0.1]$ added to the training ys.

Question 4

Derive analytically

$$\underset{\alpha_1,\ldots,\alpha_K}{\operatorname{argmin}} \; L_\lambda(\alpha_1, \ldots, \alpha_K)$$

with

$$L_\lambda(\alpha_1, \ldots, \alpha_K) = \frac{1}{2} \sum_n \left(\sum_k \alpha_k f_k(x_n) - y_n \right)^2 + \lambda \sum_k \alpha_k^2.$$

Add a method

```
void regularized_fit(scalar_t lambda,
                     int nb_samples, scalar_t *x, scalar_t *y);
```

to the class `FunctionalRegressor`, and use it to look at how regularization allows to control the over-fitting when fitting a polynomial to the sin function with noise.

You can output result files `/tmp/question4.dat` and `/tmp/question4_train.dat` with the same format as in question 3 with

```
./do.sh -v question4
```