EE-613: Machine Learning for Engineers
Practical session 10
(Decision Trees)

1 Download and test

```
wget http://www.idiap.ch/~fleuret/files/EE613/EE613-pw10.tgz
tar zxvf EE613-pw10.tgz
cd EE613/pw10
wget http://www.idiap.ch/~fleuret/files/EE613/mnist.tgz
tar zxvf mnist.tgz
./do.sh -v demo graph
```

after a couple of minutes, you should have obtained a printout of the data set check, followed by the compilation and the computation itself, ending with.

```
Depth 19 ... done (0.283333% / 14.47%).
```

and a display of an error graph.

2 Programming

2.1 Compilation

You can compile the source code with `make -k`.

It is suggested to implement each question in the corresponding function

```
void computation_question1(VignetteSet *train_image_set,
                           VignetteSet *test_image_set) {
  ...
}
```

and to test it by calling the `./do.sh question1`, `./do.sh question2`, etc.

Note that during your development, if you compile the code with

```
make -k DEBUG=yes
```

the `VignetteSet::pixel()` method will check for out-of-bound errors.

2.2 Classes

A **VignetteSet** contains small gray-scale images, each with an integer label (note that with the MNIST data set width = height = 24).

```
class VignetteSet {
public:
  inline int nb_vignettes();
  inline int nb_classes();
  inline int width();
  inline int height();

  inline unsigned char pixel(int p, int x, int y);
  inline unsigned char label(int p);

  void load_mnist_format(char *picture_file_name,
                         char *label_file_name);
```

```

  void bootstrap(int nb, VignetteSet *vs);
};
```

A **Classifier** is a virtual object that is trainable and can predict the label of a vignette.

```
class Classifier {
public:
  virtual void train(VignetteSet *train_set) = 0;
  virtual int predict(VignetteSet *vs, int n_vignette) = 0;
  int nb_errors(VignetteSet *vs);
};
```

A **DecisionTree** is a subclass of **Classifier** implementing a simple decision tree

```
class DecisionTree : public Classifier {
public:
  DecisionTree(QuestionSet *question_set,
               int depth_max,
               int nb_min_samples_for_split,
               int nb_questions_for_optimization);
  virtual ~DecisionTree();
  virtual void train(VignetteSet *train_set);
  virtual int predict(VignetteSet *vs, int n_vignette);
};
```

A **QuestionSet** is a virtual object standing for a set of Boolean functions of vignettes.

```
class QuestionSet {
public:
  virtual int nb_questions() = 0;
  virtual bool response(int n_question,
                        VignetteSet *vs, int n_vignette) = 0;
};
```

The **PixelQuestionSet** is a set of $28 \times 28 = 784$ questions, each testing if a certain pixel of the vignette is greater than 128.

3 Questions

Question 1: Performance vs. training set size

Train one tree with 16, 32, 64, etc. up to $2^{15} = 32,768$ samples, and compute the test error each time. Use depth max 10, number of samples to split 10, and look at 100 questions at every node for training.

Help: Use `VignetteSet::bootstrap` to extract sub-sets from the full training sets.

Question 2: New feature set

Write a sub-class `MinQuestionSet` of `QuestionSet` which implements questions which test that the minimum of the pixel values over rectangles of size $p \times q$ is greater than 128. Compute the same test errors vs. depths as in the demo program. Try with $p = 4,\ q = 1$ and $p = 4,\ q = 4$.

Help: Both the .h and .cc files of that new class will be extremely similar to those of the `PixelQuestionSet` class.

Question 3: Forest

Write a subclass `Forest` of the class `Classifier` which implements a voting procedure with trees. Compute the same errors as in the demo program with 5 trees.

Help: Here is the declaration of that class

```
class Forest : public Classifier {
  int _nb_trees;
  DecisionTree **_trees;
public:

  Forest(int nb_trees,
         QuestionSet *question_set,
         int depth_max,
         int nb_min_samples_for_split,
```

```
          int nb_questions_for_optimization);
  ~Forest();
  void train(VignetteSet *train_set);
  int predict(VignetteSet *vs, int n_vignette);
};
```

Question 4: Nearest-Neighbors

Implement a sub-class `NearestNeighbors` of `Classifier` which implements the k-NN classifier. Compute the test error vs. the parameter k. You may reduce the training set size to run at reasonable speed.

Help: The classifier will have to keep a pointer to a `VignetteSet`.