TELECOM ParisTech **SD210**
2017/2018 **Djerrab, Garcia, Gori, Lambert, Varni**

TP noté n° 4 : Bagging and Random Forests

In order to be evaluated, your work need to be uploaded as a **unique** file with the following format `nom_prenon.ipynb` on the website of the course (section TP4).

You must upload your work on Éole before the 07th of April 2018 23h59.

The maximum grade is **20** points, assigned in the following way :

— quality of the answers : **15** points,

— quality of the writing, presentation : **4** points,

— absence of bugs : **1** point

If you do not upload your work before the aforementioned deadline, you will get **zero** points.

The exercises are independent and no work will be accepted by email !
DO NOT SEND YOUR WORK BY EMAIL !

- Model aggregation -

We consider a standard supervised problem. Let $\mathcal{D} = \{(X_1, Y_1), \ldots, (X_n, Y_n)\}$ be a given data-set and $\hat{f}_{\mathcal{D}}$ an estimator/classifier (i.e. a function). If the ensemble of Y_i takes values in $\{1, \ldots, K\}$, we call it a multi-class classification problem (with K classes). Instead, if the Y_i take values in $\mathbb{R}$, we call it a regression problem.

An aggregation of models (classifiers/estimators) consists of linearly combining the predictions of each model. In regression, we compute the model $\hat{F}^L_{\mathcal{D}}$ with an aggregation of L estimators $\hat{f}^l_{\mathcal{D}}, l = 1, \cdots, L$:

$$\hat{F}^L_{\mathcal{D}} = \sum_{l=1}^{L} w_l \hat{f}^l_{\mathcal{D}}$$

where $w_l \geq 0$ is a weight.

For the classification, the aggregation can be done with a voting procedure (for instance with a majority rule), or by averaging the probabilities of the classes. If the prediction of a binary classifier $\hat{f}^l_{\mathcal{D}}$ in X corresponds to : $\text{sign}(\hat{f}^l_{\mathcal{D}}(X))$, then the aggregation of models can predict using : $\text{sign}(\sum_{l=1}^{L} w_l \hat{f}^l_{\mathcal{D}}(X))$.

A necessary and sufficient condition for having an aggregation of models more precise than each independent model, is that each model should predict better than by chance and in a different way if we change the input data-sets. The principle of model aggregation is based on the idea that by averaging the predictions of several independent models, we reduce the variance and thus the prediction error.

Math question : Let us consider L binary independent classifiers where the probability of a correct prediction is $p > 0.5$. Then, the prediction of the aggregation of models follows a Binomial distribution with parameters p and L (Why ?).

1) If $p = 0.7$ (which means a probability slightly greater than by chance) and $L = 10$, which is the probability of correct prediction for the aggregation of models ? We could use the implementation of the Binomial distribution in `scipy` :

```
from scipy.stats import binom
rv = binom(L, p)
```

- Bagging -

The *Bagging* (acronym of "Boostrap Aggregation") [**?**] is a classical method for combining models. It consists of taking a simple average of the predictions, *i.e.,* $w_l = 1/L$. In order to produce several estimators, we use different data-sets randomly generated using the technique of *bootstrap*. A *bootstrap* sample is a sample of n points obtained from $\mathcal{D}$ using a uniform random sampling with replacement (*i.e.,* a point may appear multiple times in the same sample).

1) Use Bagging with first decision trees of depth 1 (called *stumps*) and then with decision trees characterized by a greater depth. Use the following code :

```
import numpy as np
from sklearn.tree import DecisionTreeRegressor
from sklearn.ensemble import BaggingRegressor
import matplotlib.pyplot as plt
# Create a random dataset
rng = np.random.RandomState(1)
X = np.sort(5 * rng.rand(80, 1), axis=0)
y = np.sin(X).ravel()
y[::5] += 1 * (0.5 - rng.rand(16))

n_estimators = 10 # L in the text
tree_max_depth = 10
bagging_max_depth = 10
# TODO define the regressor by bagging stumps
# tree = ...
tree.fit(X, y)
# bagging = BaggingRegressor(...)
bagging.fit(X, y)
# Predict
X_test = np.arange(0.0, 5.0, 0.01)[:, np.newaxis]
y_tree = tree.predict(X_test)
y_bagging = bagging.predict(X_test)
# Plot the results
plt.figure(figsize=(12, 8))
plt.plot(X, y, 'o', c="k", label="data")
# TODO add plots for Bagging/Tree
plt.title("Decision Tree Regression")
plt.legend(loc=1, numpoints=1)
plt.show()
```

2) Graphically illustrate the roles of L and of the tree depth (`max_depth`).
3) How can we check that the estimators computed with the decision trees are biased and that the ones based on *bagging* reduce the variance ?
4) Playing with the noise level, show the over-fitting (sur-apprentissage).
5) Show that we can reduce this phenomenon by randomly sub-sampling (sous-échantillonner) without replacement instead than taking the *bootstrap* samples.

Random Forests

The *Random Forests* [**?**], combine the ideas of *Bagging*, *bootstrap* sampling and average, with a random selection of the variables at every node of the tree. For a classification task, the aggregation is made with a majority rule.

6) Evaluate the scores using *Random Forests* with a 7-fold cross-validation on the data-sets `boston`, `diabetes`, `iris` and `digits`. Compare the performances with the ones of a linear SVM. You could use :

```
from sklearn.ensemble import RandomForestRegressor, RandomForestClassifier
```

The *Random Forests*, like *Bagging*, can be used to predict a probability. In order to do that, the probability of belonging to class k is the proportion of trees which predict class k.

7) Using the data-set `iris`, limited to the first two variables/features, show the prediction probabilities for each class. Start from the following script where you will vary the number of random trees (parameter `n_estimators`).

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn import preprocessing
from sklearn.datasets import load_iris
from sklearn.ensemble import RandomForestClassifier
# Parameters
n_estimators = 2
plot_colors = "bry"
plot_step = 0.02
# Load data
iris = load_iris()
X_unscaled, y = iris.data[:, :2], iris.target
# Standardize
X = preprocessing.scale(X_unscaled)
# RF fitting
model = RandomForestClassifier(n_estimators=n_estimators)
clf = model.fit(X, y)
# Plot the decision boundary
x_min, x_max = X[:, 0].min() - 1, X[:, 0].max() + 1
y_min, y_max = X[:, 1].min() - 1, X[:, 1].max() + 1
xx, yy = np.meshgrid(np.arange(x_min, x_max, plot_step),
                     np.arange(y_min, y_max, plot_step))

plt.figure()
for tree in model.estimators_:
    # TODO use predict to obtain the probabilities you will store in Z
    Z = Z.reshape(xx.shape)
    cs = plt.contourf(xx, yy, Z, alpha=1. / n_estimators, cmap=plt.cm.Paired)
plt.axis("tight")
# Plot the training points
for i, c in zip(range(3), plot_colors):
    idx = np.where(y == i)
    plt.scatter(X[idx, 0], X[idx, 1], c=c, label=iris.target_names[i],
                cmap=plt.cm.Paired)
plt.legend(scatterpoints=1)
plt.show()
```

8) Compare the scores between *Random Forests* with a 6-fold cross-validation and the pure decision trees (obtained with `DecisionTreeClassifier`), on the data-set `iris` limited to the first two features/variables. Vary the parameter `max_depth` between 1 and 30. Show that both *Random Forests* and deep decision trees reduce the over-fitting.