Toulouse School of Economics

Worksheet 2 – Bagging

Worksheet 2

Camille Mondon

February 12, 2024

This worksheet[1] illustrates bagging for classification. For various classification methods, the bagging classifier is compared to the original classifier. We use datasets from the UCI Machine Learning repository; these are available in

```
library(mlbench)
```

Bagging of trees

Exercise 1. This exercise aims at predicting the presence of some type of structure in the ionosphere from radar data. Data are contained in the dataset `Ionosphere`

```
data(Ionosphere)
```

and will be analyzed using *classification trees* (packages `rpart` and `rpart.plot`):

```
library(rpart)
library(rpart.plot)
```

1. Using

```
help(Ionosphere)
```

read the description of the dataset. How many possible predictors does the dataset include and what are their types (continuous or categorical)? How many observations are there?

2. Using the function `sample`, split the dataset into a training set (of size 201) and a test set (of size 150).
3. Using the function `rpart` (with default parameters and no pruning), design —on the training set— a classification tree for predicting `Class` from all the other variables. Print a summary and a graphical representation of this procedure.
4. Find the misclassification rate on the training set. Repeat the exercise on the test set. What do you learn by comparing both errors?
5. With the aim to use it later with the function `boot` (from the package `boot`), write a function, of two arguments `x` and `d`, that

[1] The content of this worksheet is strongly based on a worksheet designed by Nathalie Vialaneix and Davy Paindaveine.

- identifies the bootstrap sample with identifier `d` coming from the dataset `x`,
- trains a classification tree on it, and
- returns the resulting prediction for the test set (encode the prediction as 1 for "good" and 0 for "bad")

Test the function with `x` being the training set obtained in Question 2 and `d` being an identifier obtained using the function `sample`.

6. Aiming at a bagging approach, use the function written in the previous question to obtain predictions for the test set from $B = 100$ bootstrap samples. What is the resulting final prediction for the test set obtained from bagging? What is the resulting test error? Compare it with the test error obtained from the direct approach.

7. Repeat the whole procedure $M = 100$ times (each time splitting randomly the original data set into a training set and a test set) to obtain 100 training errors, test errors and bagging test errors (use a `for` loop). Use boxplots to compare the distributions of these three types of errors.

Using the `R` package `ipred`

Exercise 2. This exercise will illustrate how the package `ipred` can be used for bagging. We will use the dataset `PimaIndiansDiabetes`:

```
data(PimaIndiansDiabetes)
```

The goal is to predict diabete from description of people in the Pima indians population.

1. Using

```
help(PimaIndiansDiabetes)
```

read the description of the dataset. How many possible predictors does the dataset include and what are their types? How many observations are there?

2. Using the same approach as in the previous exercise (with the package `boot`, test samples of size 300, $B = 100$ bootstrap samples, and $M = 100$ replicates of the splitting into training and test sets), use boxplots to compare the distributions of the three types of misclassification errors (train, test from the direct approach, and test from the bagging approach). Use the function `system.time` to evaluate how much time the whole procedure requires.

3. In this question, the package `ipred` is used to obtain a bagging prediction.

```
library(ipred)
```

Split the dataset into training and test sets of the same size as in Question 2 and use the function `bagging` to define a bagging classification procedure (still based on $B = 100$ bootstrap samples). Obtain the bagging predictions in the test set (use the function `predict`) and evaluate the corresponding test error. Obtain the out-of-bag (OOB) error (check the help page of the function `bagging`).

4. Repeat this $M = 100$ times. In each replication, obtain

a. the training error for a single classification tree,
b. the corresponding test error,
c. the bagging test error, and
d. the OOB error.

Use the function `system.time` to evaluate much time the whole procedure requires; how does the computation time compare with the one in Question 2? Use boxplot to compare the distributions of the errors (a)–(d).

Remark. The package `ipred` can be used directly for bagging trees but it also provides a simple way to use bagging with other methods. For an example, see

```
help(ipredknn)
```

Bagging with kNN

Exercise 3. This exercise will compare (the direct and bagging approaches for) kNN classifiers[2] with (the direct and bagging approaches for) classification trees. kNN classifiers are known to be rather stable. The comparison will be carried out on a letter recognition problem:

```
data(LetterRecognition)
```

The goal is to predict handwritten letters from a description of the images.

1. Using

```
help(LetterRecognition)
```

2. Split the dataset into a test set and a training set, of respective sizes 6,000 and 14,000.
3. Use the function `rpart` to fit a classification tree to the train set. Determine the resulting test error.
4. Use the function `bagging` to obtain the bagging-of-trees test error (with $B = 100$ bootstrap samples).
5. Use the function `tune.knn` of the package `e1071` for tuning a kNN classifier with an optimal number of neighbors chosen between 1 and 10 with a 10-fold CV strategy. Check the help pages

```
help(tune.knn)
help(tune.control)
```

and in particular the arguments `k`, `tunecontrol` (for `tune.knn`) and `sampling` and `cross` (for `tune.control`). What is the optimal number of neighbors found by CV?

```
library(class)
library(e1071)
```

6. Use this optimal value of k to train a kNN classifier with the function `knn` (use the argument `test` to compute the predictions for the test set). Compute the resulting (direct) test error.
7. Using the `boot` function, implement a bagging of kNN classifiers (keep throughout k equal to the optimal number above). Compute the resulting bagging test error.

[2] See http://en.wikipedia.org/wiki/K-nearest_neighbors_algorithm if you do not know this method.