CS4100/CS5100 Assignment 2

This assignment must be submitted by **30th November 2021, at 5pm**.

Instructions

1. This assignment accounts for 15% of the total grade of CS4100 / CS5100.
2. You can use either R, Python or MATLAB to complete this assignment.
3. You should submit files with the following names:
   - `RF.r`, which contains your source code for implementing Random Forest;
   - `AHC.r`, which contains your source code for implementing Agglomerative Hierarchical Clustering (AHC);
   - `report.pdf`, which contains the numerical results for both Random Forest and AHC, and a discussion about them.
4. In order to submit, copy all your submission files into a directory, e.g., `DA2`, and create a compressed file `DA2.zip`. **We only accept `.zip` compressed files, do not use other compression software.** Upload `DA2.zip` to Moodle through one of the following pages:

   Assignment 2 for CS5100

   Assignment 2 for CS4100
5. The files you submit cannot be overwritten by anyone else, and they cannot be read by any other student. You can, however, overwrite your submission as often as you like, by resubmitting, though only the last version submitted will be kept.
6. Submissions after the deadline will be accepted but they will be recorded as late and are subject to College Regulations on late submissions.
7. All your submissions will be graded anonymously, so **your name or student ID should not appear anywhere in your submission**.

> **Note:** All the work you submit should be solely your own work. Coursework submissions are routinely checked for this.

Learning outcomes assessed

The learning outcomes assessed are:

- develop, validate, evaluate, and use effectively machine learning models;
- apply methods and techniques such as ensemble methods and clustering;
- extract value and insight from data;
- implement machine-learning algorithms.

In this assignment, you are asked to complete two tasks: (1) implementing random forest, and (2) implementing agglomerative hierarchical clustering (AHC). Lab Worksheet 6 contains a section dedicated to Object Oriented Programming (OOP), which might be helpful for you to complete this assignment.

Task 1: Implementing Random Forest (50 marks)

You are asked to implement random forest for regression. Random forest was discussed in Teaching Week 6. **You are not allowed to use any existing implementations of decision trees or random forests in R or any other language, and you must code random forest from first principles.**

You should apply your random forest program to the `Boston` dataset to predict `medv`. In other words, `medv` is the label, and the other 13 variables in the dataset are the attributes.

Split the dataset randomly into two equal parts, which will serve as the training set and the test set. Use your birthday (in the format MMDD) as the seed for the pseudorandom number generator. The same training set and test set should be used throughout this assignment. You need to complete the following parts:

(a) Generate $B = 100$ bootstrapped training sets (BTS) from the training set.

(b) Use each BTS to train for a decision tree of height $h = 3$. Be reminded that you are implementing random forest, so at each node you do not consider all attributes, but only a sample of them.

(c) Find the training MSE and test MSE. Include it in your report.

(d) Repeat the above parts using different values of B and h. In your report, plot the training MSE and test MSE as functions of B or/and h, and discuss your observations.

In the code file, you should leave comments to clearly indicate which of your code snippets deals with which part (among (a), (b), (c) or (d) above). This helps the graders to understand your code more easily. Feel free to include in your report anything else that you find interesting.

Training for a Decision Tree

In Task 1(b), you are asked to train for a decision tree of height 3 (see Figure 1 for an example of such a decision tree). To begin, you should do the following steps at the root of the decision tree:

(i) When there are p attributes, sample $\lceil p/3 \rceil$ attributes without replacement.

(ii) Use the sampled attributes to determine the optimal split at the root of the decision tree. Recall that the optimal split is the split that minimizes the sum of the two RSS (residual sum of squares) at the two child nodes of the root. See the discussion below to refresh your memory on how to compute RSS.

(iii) The optimal split gives you a partition of the BTS into two smaller datasets. Now, repeat steps (i) and (ii) on each of the smaller datasets, to train for a decision tree of height 2.

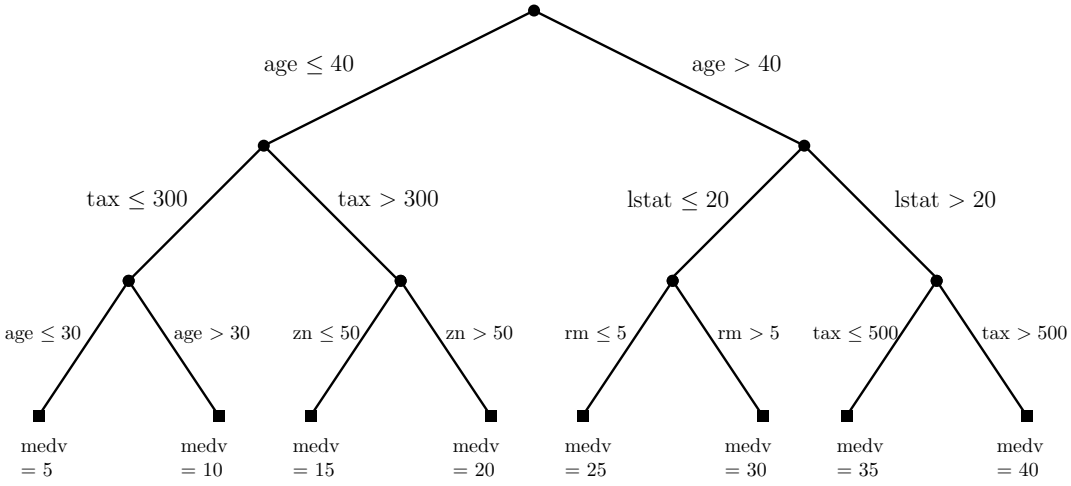

Figure 1: An example of a decision tree of height 3.

Computation of RSS

To refresh your memory, we discuss how RSS is computed. Suppose you are given a dataset $\mathcal{D}$. For any attribute "`attr`" and number s, you can split $\mathcal{D}$ into $\mathcal{D}_1$ and $\mathcal{D}_2$, where $\mathcal{D}_1$ contains all observations with the attribute values less than or equal to s, and $\mathcal{D}_2$ contains the remaining observations. Let $\hat{y}_{\leq}$ denote the average of the label values in $\mathcal{D}_1$. Then the RSS of $\mathcal{D}_1$ is

$$\texttt{RSS}_1 \ := \sum_{i \in \mathcal{D}_1} (y_i - \hat{y}_{\leq})^2,$$

where i denotes an index of an observation, and y_i denotes the label value of the i-th observation.

Analogously, let $\hat{y}_{>}$ denote the average of the label values in $\mathcal{D}_2$. The RSS of $\mathcal{D}_2$ is

$$\texttt{RSS}_2 \ := \sum_{i \in \mathcal{D}_2} (y_i - \hat{y}_{>})^2.$$

The optimal split is determined by the attribute "`attr`" and the number s that produce the smallest sum of $\texttt{RSS}_1$ and $\texttt{RSS}_2$.

See the next page for Task 2.

Task 2: Implementing Agglomerative Hierarchical Clustering (50 marks)

You are asked to implement agglomerative hierarchical clustering (AHC). AHC was discussed in Teaching Week 7. **You are not allowed to use any existing implementations of AHC in R or any other language, and you must code AHC from first principles.**

You should apply your AHC program to the `NCI microarray` dataset which can be downloaded from the module's Moodle page. This dataset has 64 columns and 6830 rows, where each column is an observation (a cell line), and each row represents a feature (a gene). Therefore, the dataset is represented via its transposed data matrix. You can load the dataset using the following code snippet in R.

```
ncidata <- read.table("ncidata.txt")
ncidata <- t(ncidata)
```

After executing the above code snippet, `ncidata` has 64 rows, and each row is an observation. Each observation is a vector in $\mathbb{R}^{6830}$.

You need to complete the following parts:

(a) Implement AHC with the following linkage functions: single linkage, complete linkage, average linkage and centroid linkage. Your output should be a data structure that represents a dendrogram. We will discuss how to design a class for the nodes in a dendrogram below.

(b) Implement a function `getClusters` that takes a dendrogram and a positive integer K as arguments, and its output is the K clusters obtained by cutting the dendrogram at an appropriate height.

(c) In your report, use the `getClusters` function to discuss the performance of AHC with the four different linkage functions when applied to the `NCI microarray` dataset.

In the code file, you should leave comments to clearly indicate which of your code snippets deals with which part (among (a), (b) or (c) above). This helps the graders to understand your code more easily. Feel free to include in your report anything else that you find interesting.

Class for Nodes in a Dendrogram

In Task 2(a), you are asked to implement a dendrogram. Supposed we are working with a dataset which contains n observations. At the bottom of a dendrogram is n leaves, where each leaf is a cluster containing one observation. The height of each leaf is zero. Then in each iteration, you use the linkage function to determine which two clusters to be merged. After $(n-1)$ iterations, you have only one cluster left and the algorithm terminates.

In a dendrogram, the "object" is a node, which can be either an internal node or a leaf (see Figure 2 for an example). Each node corresponds to one cluster. A leaf does not have any child. An internal node has two children.

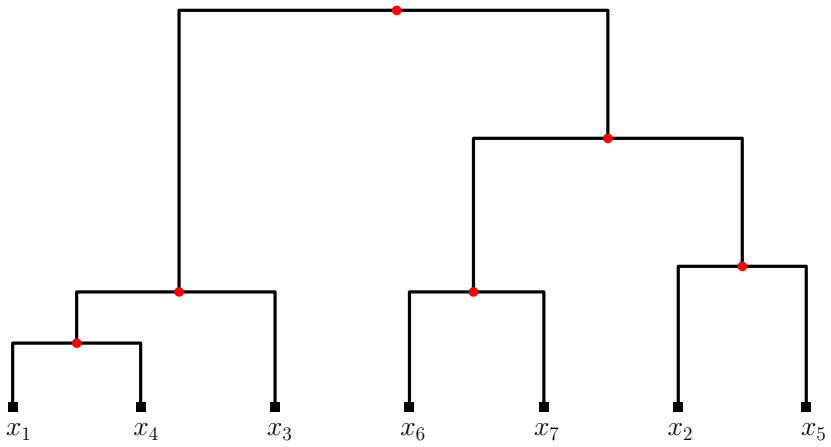

Figure 2: An example of a dendrogram of 7 observations. Each node marked with ■ is a leaf, which corresponds to a cluster of one observation. Each node marked with • is an internal node, which has two child nodes.

When designing a class for the node objects, the class should have at least the following fields:

- `is.leaf`: a logical variable that indicates whether the node is a leaf or an internal node;
- `height`: a numeric variable that stores the height of the node;
- `observations`: a list or vector that stores all observations (or indices of these observations) in the cluster represented by the node;
- `left.node`: if the node is an internal node, `left.node` points to its left child; and
- `right.node`: if the node is an internal node, `right.node` points to its right child.

These will be sufficient for completing Task 2(a). To complete Task 2(b), you may need to add more fields to the class.

Marking criteria

To be awarded full marks, you need both to submit correct code and to obtain correct results on the given datasets. Even if your results are not correct, marks will be awarded for correct or partially correct code. Correctly completing Task 1(b) will give you 30 marks. Correctly completing Task 2(a) will give you 30 marks.