# CMPT459: Data Mining, Spring 2024

**Instructor: Martin Ester**
**TAs: Arash Khoeini, Ali Alimohammadi**

**Assignment 1**

In this assignment, you will implement a Random Forest, using the files provided, without using any third-party library except NumPy and pandas. You will then apply your Random Forest to classify people into those who earn less than 50k and higher than 50k based on their features. The given dataset consists of 14 features (6 continuous and 8 categorical). You can find more information about this dataset here.

## Introduction to Random Forests

Random forest, like its name implies, consists of a large number of individual decision trees that operate as an ensemble. Each individual tree in the random forest spits out a class prediction and the class with the most votes becomes our model's prediction.

The fundamental concept behind random forest is a simple but powerful one — the wisdom of crowds. In data mining language, the reason that the random forest model works so well is that a large number of relatively uncorrelated models (trees) operating as a committee will outperform any of the individual constituent models.

The low correlation between models is the key. Just like how investments with low correlations (like stocks and bonds) come together to form a portfolio that is greater than the sum of its parts, uncorrelated models can produce ensemble predictions that are more accurate than any of the individual predictions. The reason for this wonderful effect is that the trees protect each other from their individual errors (as long as they don't constantly all err in the same direction). While some trees may be wrong, many other trees will be right, so as a group the trees are able to move in the correct direction.

So how does random forest ensure that the behaviour of each individual tree is not too correlated with the behaviour of any of the other trees in the model? It uses the following two methods:

**Bagging (Bootstrap Aggregation) —** Decisions trees are very sensitive to the data they are trained on — small changes to the training set can result in significantly different tree structures. Random forest takes advantage of this by allowing each individual tree to randomly sample from the dataset with replacement, resulting in different trees. This process is known as bagging.

Please notice that with bagging we are not subsetting the training data into smaller chunks and training each tree on a different chunk. Rather, if we have a sample of size N, we are still feeding each tree a training set of size N (unless specified otherwise). But instead of the original training

data, we take a random sample of size N with replacement. For example, if our training data was [1, 2, 3, 4, 5, 6] then we might give one of our trees the following list [1, 2, 2, 3, 6, 6]. Notice that both lists are of length six and that "2" and "6" are both repeated in the randomly selected training data we give to our tree (because we sample with replacement).
**Feature Randomness —** In a normal decision tree, when it is time to split a node, we consider every possible feature and pick the one that produces the most separation between the observations in the left node vs. those in the right node. In contrast, each tree in a random forest can pick only from a random subset of features. This forces even more variation amongst the trees in the model and ultimately results in lower correlation across trees and more diversification. (source)

You are provided with a file random_forest.py which includes two classes:

- **Node:** This is the data structure that you will use to create the tree. Each Node object is a leaf node by default (node.is_leaf = True) unless you set its 'children'. Then it becomes a decision node. Note that using the node.set_children method will set node.is_leaf to False automatically. A decision node is simply a node object which (1) has a name same as the feature name it uses to split, (2) has these properties set: node.children, node.is_numerical, node.threshold (only for numerical nodes) and (3) has node.is_leaf = False. For more details about this class please look into the implementation.
- **RandomForest:** This is the class you need to complete in order to finish this assignment. Please read the implementation documentation carefully before doing any coding.
  The main methods you will need to complete are *fit*, *predict*, *split_node*, *gini*, and *entropy*. You are also allowed to add new helper methods if you wish.

You are also provided with the file *main.py* which you will use to run your implementation. It accepts the following arguments:

- --train-data: Train data path
- --test-data: Test data path
- --criterion: The function to measure the quality of a split. Supported criteria are "gini" for the Gini index and "entropy" for the information gain.
- --maxdepth: The maximum depth of the tree.
- --min-sample-split: The minimum number of samples required to split an internal node.
- --max-features: The number of features to consider when looking for the best split
- – n_classifiers: The number of trees to be trained

**[IMPORTANT]** You should submit a *[student-id].zip* file. The zip file should include the following:

- a *report.pdf* file
- a *random_forest.py*
- a *main.py*

Running *python3.8* command on *main.py* file with the mentioned arguments must print out your train and the test accuracy of task #1.

**Deadline:** make your submission no later than **23:59 pm on February 13th**. You will lose %10 of the marks for submissions after this deadline, as long as it's not more than 24 hours late. You will lose all the marks for submissions after that time.

**Libraries:** No third-party library is allowed except NumPy and pandas.
You MUST provide YOUR OWN implementation. It MUST be implemented from scratch i.e. not using scikit-learn libraries. You will be marked on the correctness of your implementation. You are also encouraged to do any data preprocessing that you think might help the classification accuracy.

a) [10 marks] Present a pseudo-code for a simple Decision Tree:
   i) The Gini index is used as a split criterion. [2 marks]
   ii) The trees are grown deep, i.e. they are grown until all training examples corresponding to a leaf node belong to the same class. [3 marks]
   iii) Works both on categorical and numeric data. [5 marks]

b) [90 marks] Implement the Random Forest algorithm using Python. Your implementation must include the following functions (method signatures might be different based on the specific needs of your implementation):

   - *fit*: samples the dataset and generates *self.n_classifiers* trees by calling *generate_tree*. [5 marks]
   - *split_node:* splits a node into its children [5 marks]
   - *predict*: returns the aggregated prediction of all the trees on the given dataset. You should use voting to aggregate predictions. [5 marks]
   - *gini: measures Gini index* [5 marks]
   - *entropy: measures information gain* [5 marks]

   Use your implementation to complete the following tasks:

   1. Train a Random Forest model n_classifiers and maxdepth set to 10, min_sample_split=20 and max_features set to 11. Use the Gini index as the criterion. Report the accuracy of the training and testing data. [15 marks]
   2. Do the same experiment as 1 but this time with Information Gain as the criterion. How does this change the accuracy of training and test datasets? Discuss it in your report. [10 marks]

3. In your report explain why the accuracy of the training data is not equal to 100%. [5 marks]
    a. Is there any way to train a tree that yields 100% accuracy on the training dataset? How would that affect the accuracy of the test data? [5 marks]
    b. What parameters should you change to get a tree with perfect training accuracy? Explain if such a classification model has a high variance or bias? [10 marks]
4. Train 10 Random Forest models with min_sample_split=max_features=n_classifiers=10 and maxdepth=[1,2,3,...,10]. Use the Gini index as the criterion. Plot the accuracy of the 10 models on the test dataset in your report. Discuss your results. [20 marks]