# Tree based methods LAB

*Enrique J. De La Hoz D.*

*Data Science UTB*

## Building a simple decision tree

The *clientes* dataset contains information about the record of clients in a telecomunication company.

You will use a decision tree to try to learn patterns in the **churn rate**. based on the tenure and the Monthly Payment.

Then, see how the tree's predictions differ for a client leaving the company versus one remaining as an active client.

- Load 'clients.csv'
- Create a training and test dataset.
- Load the rpart package.
- Fit a decision tree model using the training datset with the function rpart(), call it clients_model
  - Supply the R formula that specifies outcome as a function of *tenure* and *Pago_mensual* as the first argument.
  - Leave the control argument alone for now. (You'll learn more about that later!)
- Use predict() with the resulting loan model to predict the outcome for the churn rate. Use the type argument to predict the "class" of the outcome.
- Apply the model to the test dataset and check the results

## Visualizing classification trees

- As Data Scientist it is important to support the results through visualizations
- The structure of classification trees can be depicted visually, which helps to understand how the tree makes its decisions.
- Type clients_model to see a text representation of the classification tree.
- Load the rpart.plot package.
- Apply the rpart.plot() function to the clients_model to visualize the tree.
- See how changing other plotting parameters impacts the visualization by running the supplied command.

## Building and evaluating a larger tree

Previously, you created a simple decision tree that used the client's information to predict the Churn.

Using all of the available applicant data, build a more sophisticated lending model

- The rpart package is loaded into the workspace and the loans_train and loans_test datasets have been created.
- Use rpart() to build a loan model using the training dataset and all of the available predictors. Again, leave the control argument alone.
- Applying the predict() function to the testing dataset, create a vector of predicted outcomes. Don't forget the type argument.

- Create a table() to compare the predicted values to the actual outcome values.
- Compute the accuracy of the predictions using the mean() function.

## Preventing overgrown trees

The tree grown on the full set of clients data grew to be extremely large and extremely complex, with hundreds of splits and leaf nodes containing only a handful of applicants. This tree would be almost impossible for a loan officer to interpret (Difficult to sell).

Using the pre-pruning methods for early stopping, you can prevent a tree from growing too large and complex. See how the rpart control options for maximum tree depth and minimum split count impact the resulting tree.

- Add a maxdepth parameter to the rpart.control() object to set the maximum tree depth to six. Leave the parameter cp = 0. Pass the result of rpart.control() as the control parameter in your rpart() call.
- Check how the test set accuracy of the simpler model compares to the original accuracy.
- First create a vector of predictions using the predict() function.
- Compare the predictions to the actual outcomes and use mean() to calculate the accuracy.
- Add a minsplit parameter to the rpart.control() object to require 70 observations to split. Again, leave cp = 0.
- Again compare the accuracy of the simpler tree to the original.

## Creating a nicely pruned tree

Stopping a tree from growing all the way can lead it to ignore some aspects of the data or miss important trends it may have discovered later.

By using post-pruning, you can intentionally grow a large and complex tree then prune it to be smaller and more efficient later on.

In this exercise, you will have the opportunity to construct a visualization of the tree's performance versus complexity, and use this information to prune the tree to an appropriate level.

The rpart package is loaded into the workspace, along with loans_test and loans_train.

- Use all of the applicant variables and no pre-pruning to create an overly complex tree. Make sure to set cp = 0 in rpart.control() to prevent pre-pruning.
- Create a complexity plot by using plotcp() on the model.
- Based on the complexity plot, prune the tree to a complexity of 0.0014 using the prune() function with the tree and the complexity parameter.
- Compare the accuracy of the pruned tree to the original accuracy of 58.3%. - To calculate the accuracy use the predict() and mean() functions