**LAB 1: Decision tree & Random Forest with R, Matlab and Python**

**Objectives:**

**Materials**

|  |  |
| --- | --- |
| **File Name** | **Description** |
| *cali.csv* | Calibration data set for predicting wildfire in South Korea |
| *vali.csv* | Validation data set for predicting wildfire in South Korea |
| *DTcode\_R.txt* | DT algorithm code using rpart function in R |
| *RF\_TF.py* | A python code of Random Forest with TensorFlow. |
|  |  |
|  |  |
|  |  |

**PART I: Decision Tree in R**

**Task 1. Setup environment**

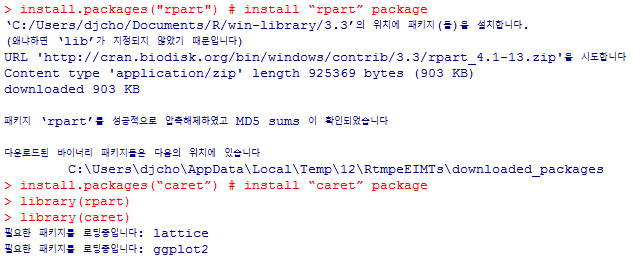
1. Prepare dataset

* Prepare cali.csv as calibration file to make a model and vali.csv file as validation file

**Task 2. Make Decision Tree Model in R**

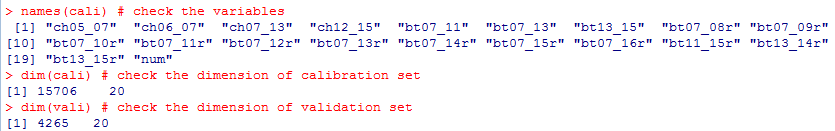
1. Install packages

* First, install the “rpart” and “caret” packages and open library. If you successfully install the package, you can see this result.



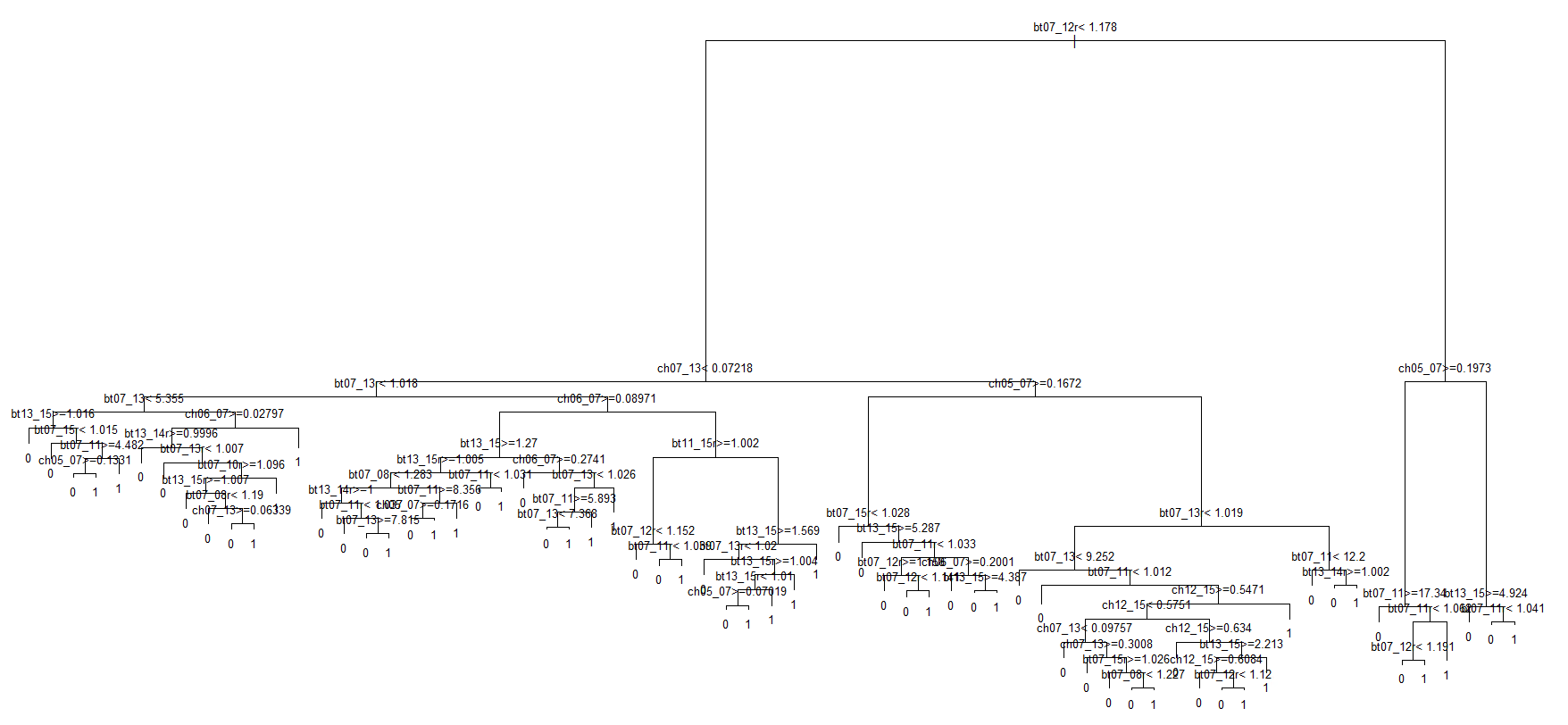
1. Read data file

* Read the data file.
* If you open the cali.csv and vali.csv files, you can notice that used variables and dimensions.



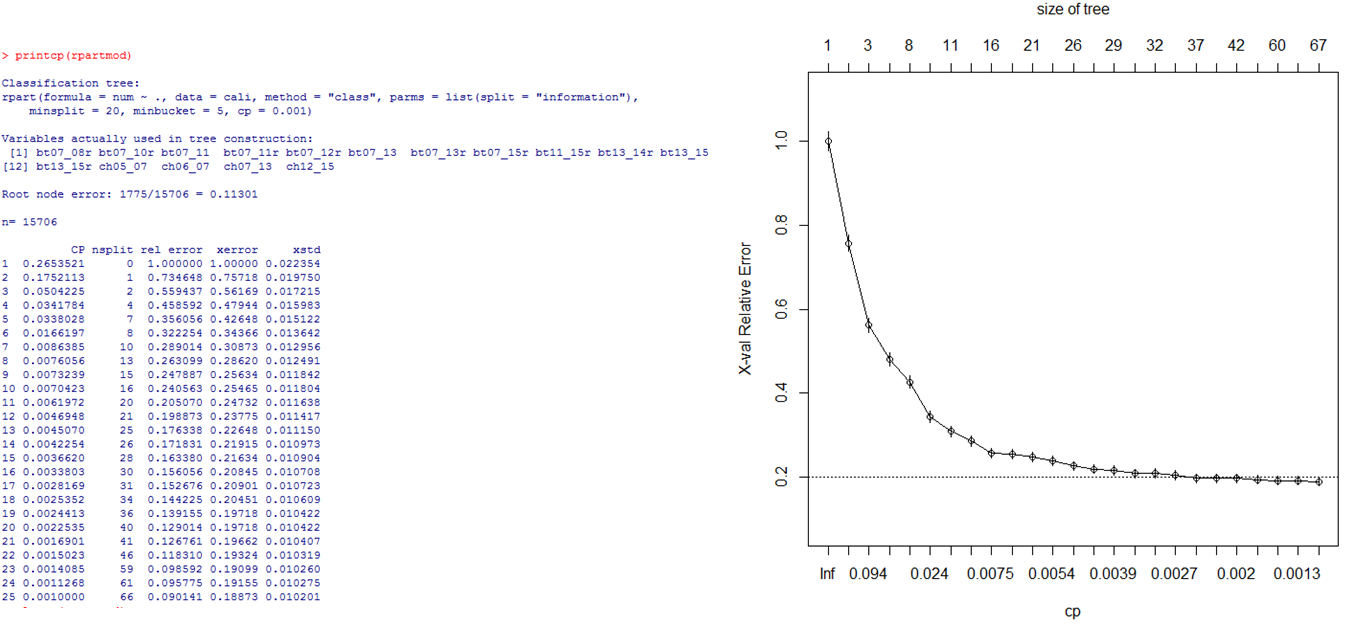
1. Build the model

* In this step, you can easily build the model using rpart function.
* In this lap, classification method and information gain was used. Also, each value of minsplit, minbucket and cp is 20, 5 and 0.001.
* If you want to plot the made tree, then it can be plotted using plot and text function like below figure.

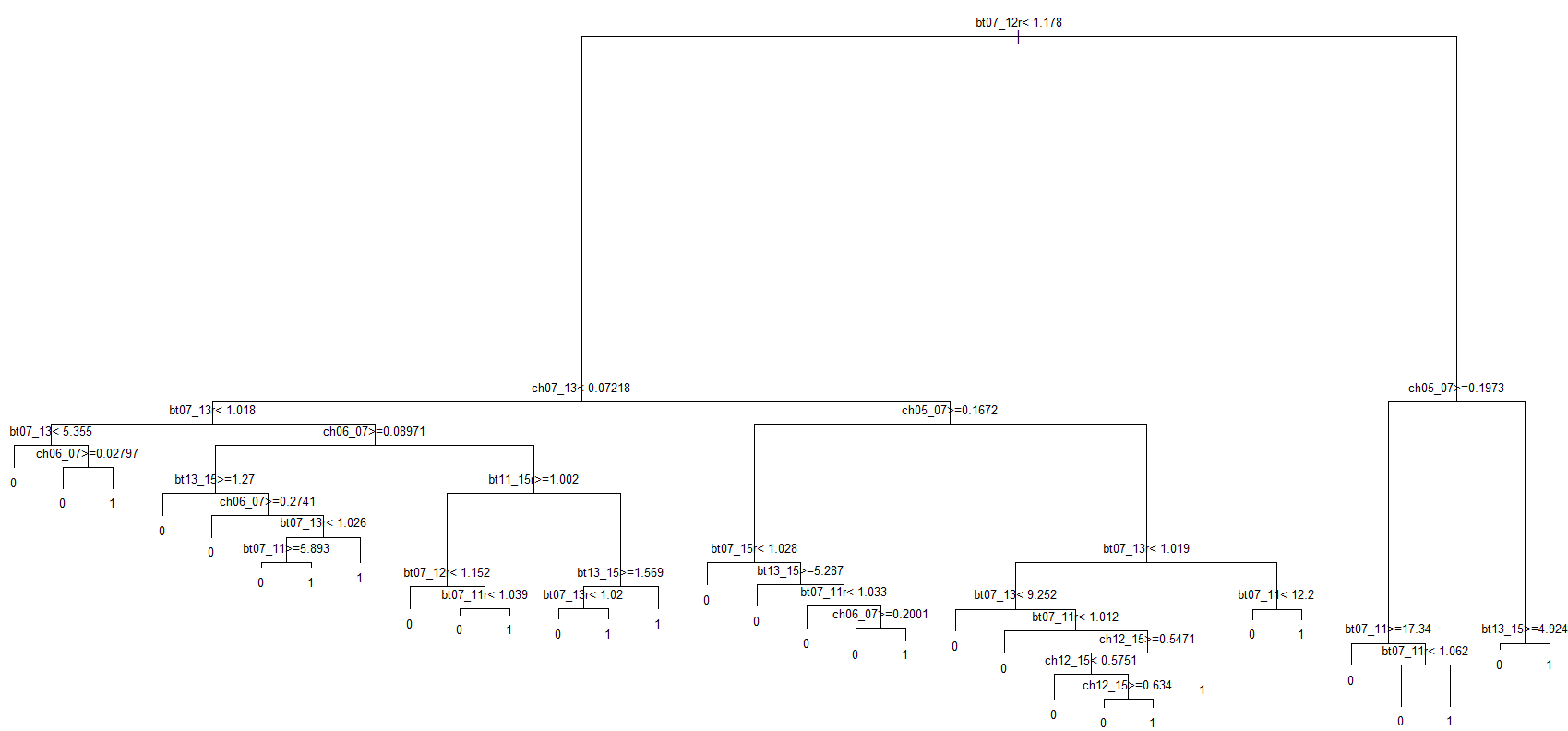


1. Prune process

* When you use the decision tree, be careful overfitting.
* So in order to prevent overfitting, pruning process is needed.
* Pruning criteria is complexity parameter(cp). cp value can be checked by using printcp or plotcp function like below figure.

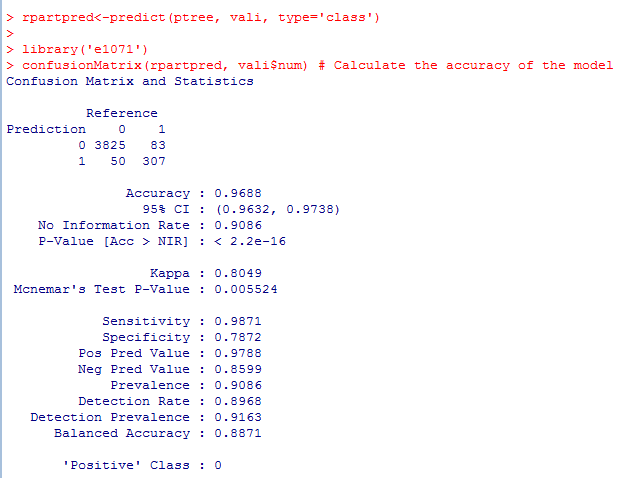


* From cp results, we can know that there is no big difference from below 0.003
* By using prune function, we can make a new tree.
* Let’s plot new tree algorithm using plot and text



1. Apply the decision tree to data

* Use the predict function.
* Open library ‘e1071’ for using confusionMatrix function
* Use the confusionMatrix function, then you can check the results



**PART II: Random Forest in R**

**Task 1. Setup Environment**

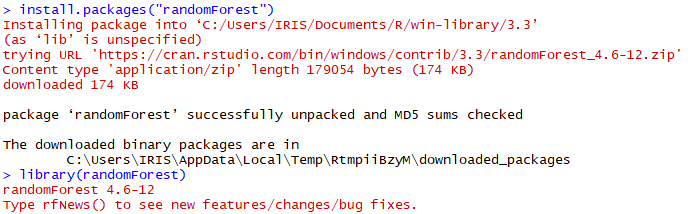
1. Prepare dataset

* Prepare cali.csv as calibration file to make a model and vali.csv file as validation file

**Task 2. Make Random Forest Model in R**

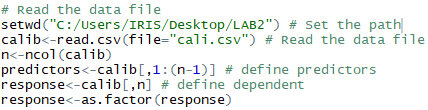
1. Install packages

* First, install the “randomForest” packages and open library. If you successfully install the package, you can see this result.



1. Read data file

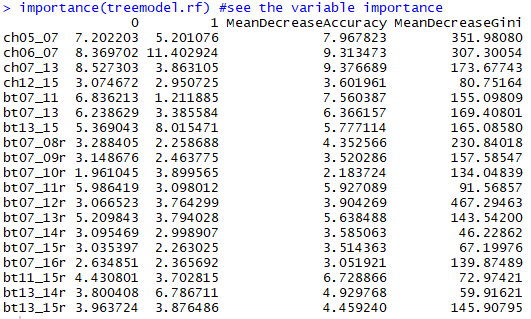
* Read the data file and predictors(x) and response(y). If you open the cali.csv file, you can notice that last column is response data.



1. Build the model

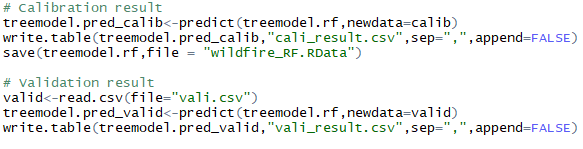
* In this step, you can easily build the model using randomForest package. You are also able to see importance like below figure.

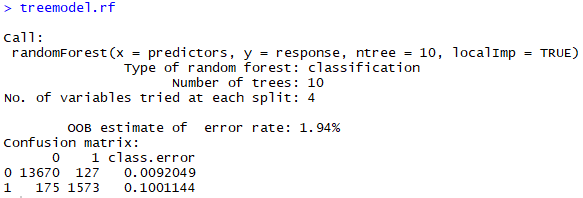




1. Apply the Random Forest model to data

* Apply the model to calibration and validation data and save the result. If you successfully do this, you can get cali\_result.csv and vali\_result.csv file in your working directory. You can also see the detail information in Console with just typing “treemodel.rf.”.





1. Exercise
   * Manually adjust the option default values (using ntree=, mtry=, nodesize=, maxnodes=, localImp=, etc.) and compare the result.

**PART II: Random Forest in Matlab**

**Task 1. Make Random Forest Model in Matlab**

1. Read data file

* First, read the data file. If you open the cali.csv file, you can notice that last column is response data.

tbl=readtable('cali.csv','format','%f%f%f%f%f%f%f%f%f%f%f%f%f%f%f%f%f%f%f%C');

1. Build the model

* You can easily build the model using TreeBagger Function. You are also able to identify relative variable importance.

% Built random forest model

RF\_model= TreeBagger(50,tbl,'num','Method','classification','OOBPrediction','on','OOBPredictorImportance','on');

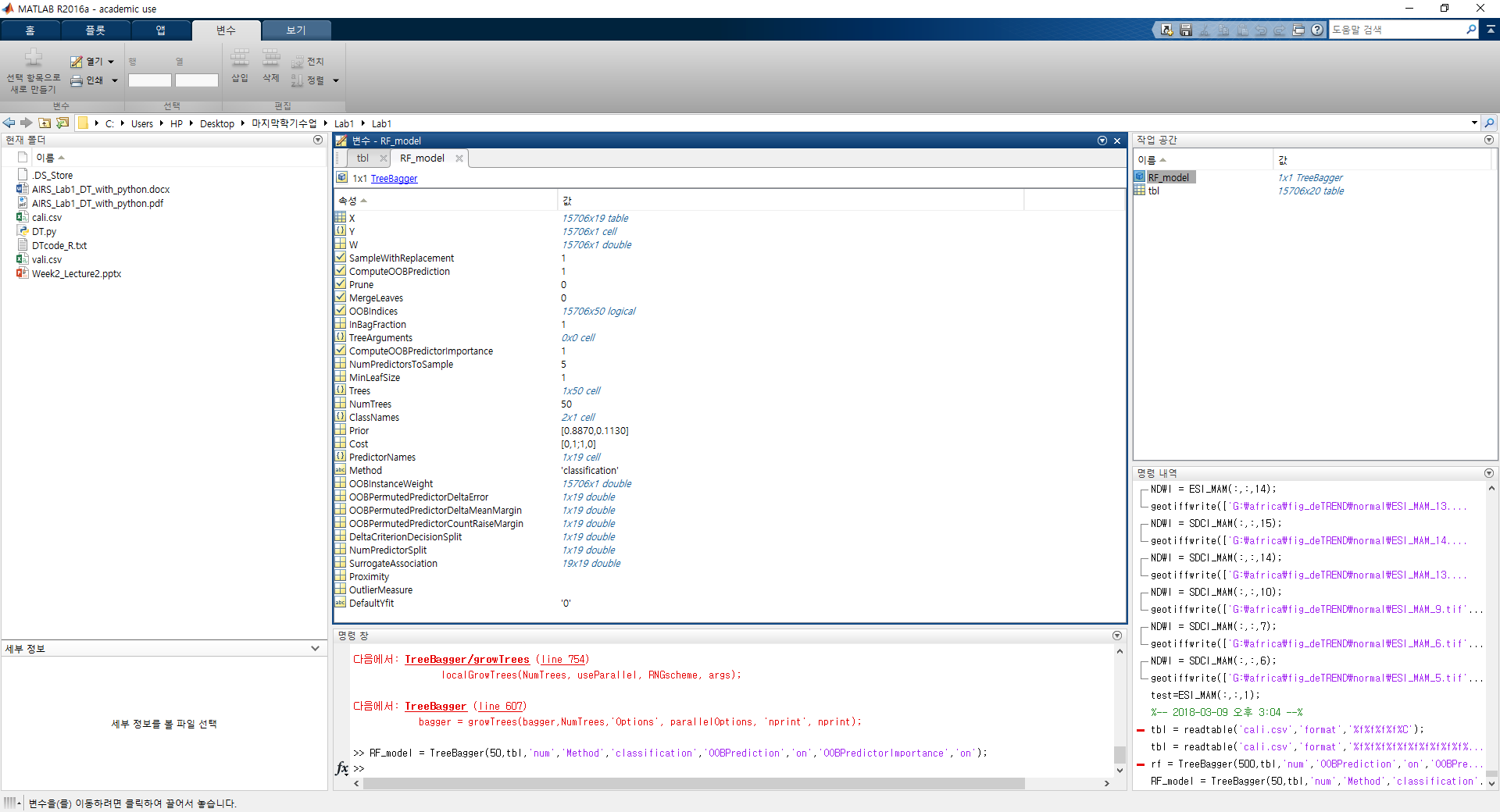


Figure 1 developed RF model

% Variable importance

imp = RF\_model.OOBPermutedPredictorDeltaError

% Make bar graph of the variable importance

figure;

bar(imp);

title('Predictor Importance Estimates');

ylabel('Estimates');

xlabel('Predictors');

h = gca;

h.XTickLabel = RF\_model.PredictorNames;

h.XTickLabelRotation = 45;

h.TickLabelInterpreter = 'none';

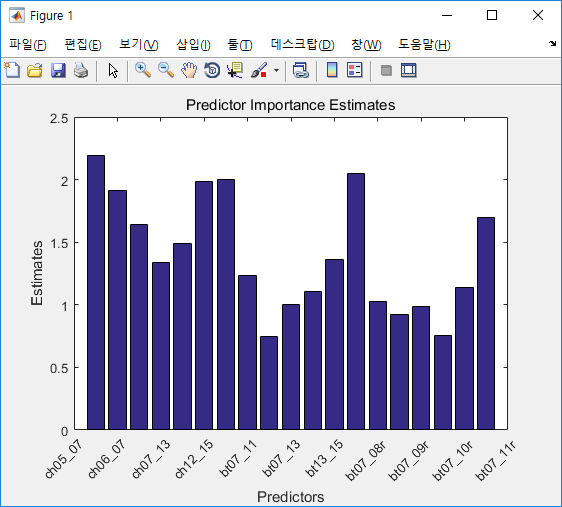


Figure 2 Variable importance

1. Validation or prediction

* In this step, you can validate the developed model using validation data (‘vali.csv’).

% Load Validation data

X\_val= readtable('vali.csv','format','%f%f%f%f%f%f%f%f%f%f%f%f%f%f%f%f%f%f%f%C');

% validation

pred = predict(RF\_model, X\_val);

**PART III: Random Forest in Python with TensorFlow**

**Task 1. Import your libraries**

Load the necessary libraries before starting.

# Import libraries  
import tensorflow as tf  
from tensorflow.contrib.tensor\_forest.python import tensor\_forest # Random forest in TF  
from tensorflow.python.ops import resources  
import numpy as np  
import pandas as pd

Tensorflow's RandomForest library is *tensor\_forest*. If not installed, use the Anaconda prompt. At the Anaconda prompt, you can install the required libraries by typing *pip install 'library\_name'* or *conda install 'library\_name'*.

**Task 2. Load wildfire data**

Load wildfire data. First, you need to set the working path. Here Pandas library was used to read csv file as DataFrame format. To convert DataFrame into the array, numpy.array function was used.

After loading wildfire data split them into X and Y for training.

# Load wildfire data  
work\_path = '/Users/dhan/Dropbox/Archive/\_coursework/2018\_1st/AI\_RS/week2/lab/Lab1' # Define your work path  
cali\_path = work\_path + '/' + 'cali.csv'  
vali\_path = work\_path + '/' + 'vali.csv'  
cali = np.array(pd.read\_csv(cali\_path, dtype='float32'))  
vali = np.array(pd.read\_csv(vali\_path, dtype='float32'))  
  
cali.shape # You can check the shape of calibration dataset. [15707 samples, 19 variables, 1 label]  
vali.shape # You can check the shape of validataion dataset. [4266 samples, 19 variables, 1 label]  
  
# Split your data into X and Y. Here, the last column is the true value.  
X\_cali = cali[:,:-1]  
Y\_cali = cali[:,-1]  
X\_vali = vali[:,:-1]  
Y\_vali = vali[:,-1]

**Task 3. Set the parameters**

Before building Random Forest model, some parameters should be set. You can compare the results by changing these parameters.

# Parameters  
num\_steps = 100 # Total steps to train  
num\_classes = 2 # The binary wildfire detection  
num\_features = 19 # Total 19 variables  
num\_trees = 100  
max\_nodes = 1000

**Task 4. Set the tf.Placeholders**

In TensorFlow, it is needed to set the *placeholder* before build a structure. *Placeholder* is one of the unique variable type of Tensorflow. A *placeholder* is simply a variable that we will assign data later. It allows us to create our operations and build our computation graph, without needing the data. In TensorFlow terminology, we then **feed** data into the graph through these placeholders.

# Input and Target data  
X = tf.placeholder(tf.float32, shape=[None, num\_features])  
# For random forest, labels must be integers (the class id)  
Y = tf.placeholder(tf.int32, shape=[None])

**Task 5. Build a Random Forest model.**

A Random Forest model can be built using ***tensor\_forest***. First, we need to assign the parameters from using *tensor\_forest.ForestHParams().* ‘HParams’ means the hyper parameters.

# Random Forest Parameters  
hparams = tensor\_forest.ForestHParams(num\_classes=num\_classes,  
 num\_features=num\_features,  
 num\_trees=num\_trees,  
 max\_nodes=max\_nodes).fill()  
  
# Build the Random Forest  
forest\_graph = tensor\_forest.RandomForestGraphs(hparams)  
# Get training graph and loss  
train\_op = forest\_graph.training\_graph(X, Y)  
loss\_op = forest\_graph.training\_loss(X, Y)

To get the accuracy, the accuracy calculation method should be defined.

# Compare prediction and true value  
correct\_prediction = tf.equal(tf.argmax(infer\_op, 1), tf.cast(Y, tf.int64))  
accuracy\_op = tf.reduce\_mean(tf.cast(correct\_prediction, tf.float32))  
  
# Initialize the variables (i.e. assign their default value) and forest resources  
init\_vars = tf.group(tf.global\_variables\_initializer(),  
 resources.initialize\_resources(resources.shared\_resources()))

**Task 6. Run the model.**

After building graphs, it is needed to initialize the variables with *tf.global\_variables\_initializer().* And run the model using *sess.run* feeding the X and Y data into placeholders with *feed\_dict*.

# Initialize the variables (i.e. assign their default value) and forest resources  
init\_vars = tf.group(tf.global\_variables\_initializer(),  
 resources.initialize\_resources(resources.shared\_resources()))  
  
# Start TensorFlow session  
sess = tf.Session()  
  
# Run the initializer  
sess.run(init\_vars)  
  
# Training  
for i in range(1, num\_steps + 1):  
 # Prepare Data  
 \_, l = sess.run([train\_op, loss\_op], feed\_dict={X: X\_cali, Y: Y\_cali})  
 if i % 10 == 0 or i == 1:  
 acc = sess.run(accuracy\_op, feed\_dict={X: X\_cali, Y: Y\_cali})  
 print('Step %i, Loss: %f, Acc: %f' % (i, l, acc))  
  
# Test Model  
print("Test Accuracy:", sess.run(accuracy\_op, feed\_dict={X: X\_vali, Y: Y\_vali})) # vali accuracy  
pred = sess.run(tf.argmax(infer\_op,1), feed\_dict={X: X\_vali, Y: Y\_vali}) # binary prediction results

Check the accuracies per each iteration. Test Accuracy is calculated using validation data.

Step 1, Loss: -0.000000, Acc: 0.886986  
Step 10, Loss: -28.320000, Acc: 0.958551  
Step 20, Loss: -217.600006, Acc: 0.980262  
Step 30, Loss: -540.280029, Acc: 0.988985  
Step 40, Loss: -928.460022, Acc: 0.992996  
Step 50, Loss: -998.000000, Acc: 0.993506  
Step 60, Loss: -998.000000, Acc: 0.993506  
Step 70, Loss: -998.000000, Acc: 0.993506  
Step 80, Loss: -998.000000, Acc: 0.993506  
Step 90, Loss: -998.000000, Acc: 0.993506  
Step 100, Loss: -998.000000, Acc: 0.993506  
Valiation Accuracy: 0.977726

**Assignment**

1. Run Random Forest model with R, Matlab and Python code changing parameters. Compare the accuracies and running times. Report your results in a document and submit to TA (dhan@unist.ac.kr).