LAB 1: DECISION TREE & RANDOM FOREST WITH R, MATLAB AND PYTHON

OBJECTIVES:

Objective 추가

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.

Data description 추가

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

'rpart' 및 'caret' 라이브러리 설명 pdf 추가하기

1) Install packages
   - First, install the "rpart" and "caret" packages and open library. If you successfully install the package, you can see this result.

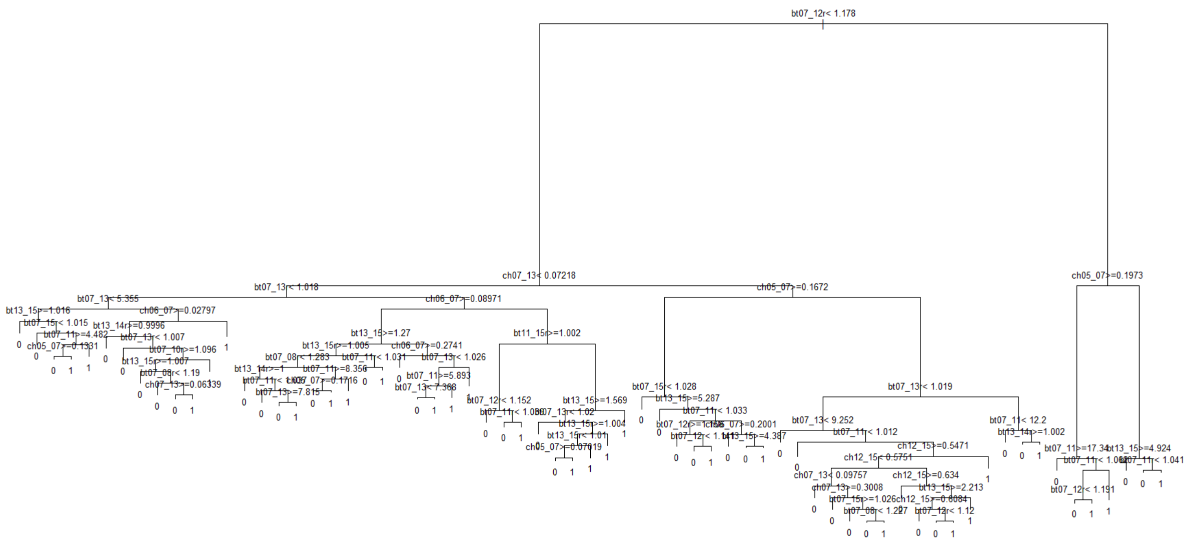
```
> install.packages("rpart") # install "rpart" package
'C:/Users/djcho/Documents/R/win-library/3.3'의 위치에 패키지(들)을 설치합니다.
(왜냐하면 'lib'가 지정되지 않았기 때문입니다)
URL 'http://cran.biodisk.org/bin/windows/contrib/3.3/rpart_4.1-13.zip'을 시도합니다
Content type 'application/zip' length 925369 bytes (903 KB)
downloaded 903 KB

패키지 'rpart'를 성공적으로 압축해제하였고 MD5 sums 이 확인되었습니다

다운로드된 바이너리 패키지들은 다음의 위치에 있습니다
        C:\Users\djcho\AppData\Local\Temp\12\RtmpeEIMTs\downloaded_packages
> install.packages("caret") # install "caret" package
> library(rpart)
> library(caret)
필요한 패키지를 로딩중입니다: lattice
필요한 패키지를 로딩중입니다: ggplot2
```

2) 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.

RStudio에서 Dimension이 얼마나 되는지 보여주기

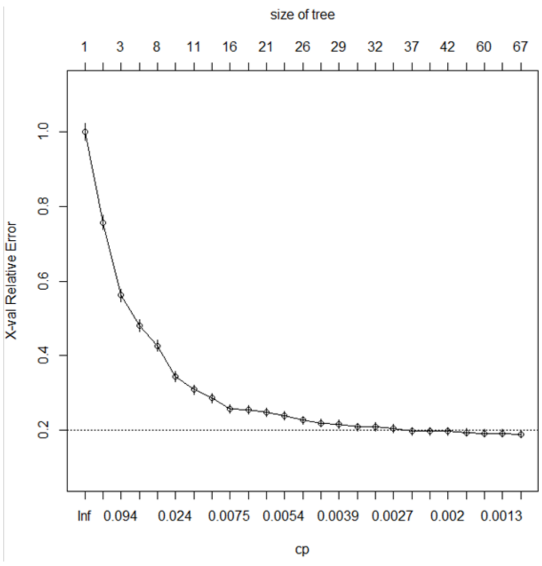
```
> names(cali) # check the variables
 [1] "ch05_07"  "ch06_07"  "ch07_13"  "ch12_15"  "bt07_11"  "bt07_13"  "bt13_15"  "bt07_08r" "bt07_09r"
[10] "bt07_10r" "bt07_11r" "bt07_12r" "bt07_13r" "bt07_14r" "bt07_15r" "bt07_16r" "bt11_15r" "bt13_14r"
[19] "bt13_15r" "num"
> dim(cali) # check the dimension of calibration set
[1] 15706    20
> dim(vali) # check the dimension of validation set
[1] 4265   20
```

3) 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.

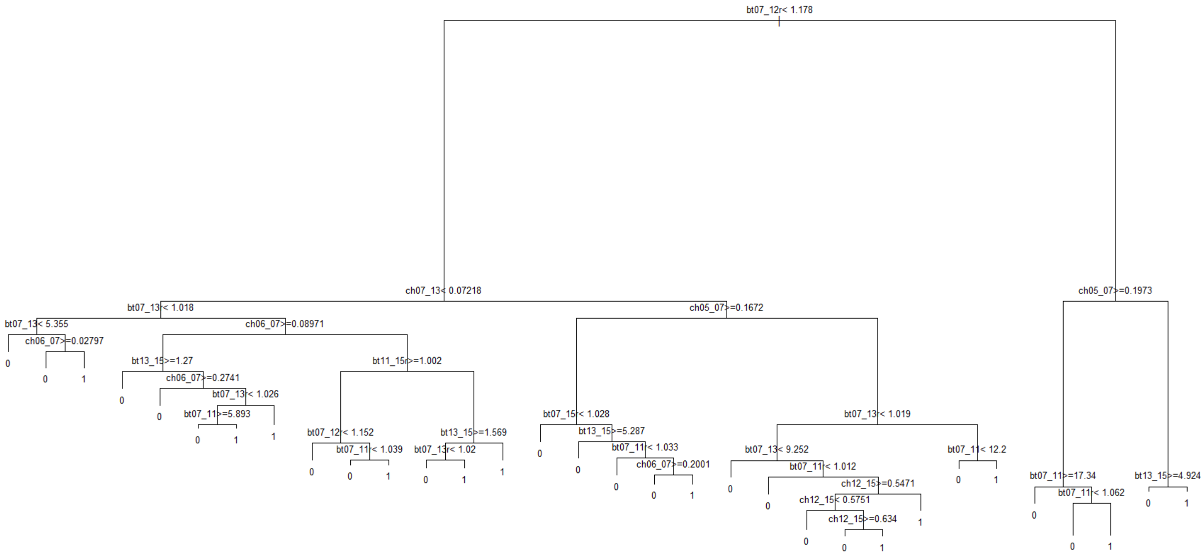


4) Prune process
   - When you use the decision tree, be careful overfitting.
   - So in order to prevent overfitting, pruning process is needed.
   - Pruning criteria is complexity parameter(cp). cp value can be checked by using printcp or plotcp function like below figure.

```
> printcp(rpartmod)

Classification tree:
rpart(formula = num ~ ., data = cali, method = "class", parms = list(split = "information"),
    minsplit = 20, minbucket = 5, cp = 0.001)

Variables actually used in tree construction:
 [1] bt07_08r bt07_10r bt07_11  bt07_11r bt07_12r bt07_13  bt07_13r bt07_15r bt11_15r bt13_14r bt13_15
[12] bt13_15r ch05_07  ch06_07  ch07_13  ch12_15

Root node error: 1775/15706 = 0.11301

n= 15706

          CP nsplit rel error  xerror     xstd
1  0.2653521      0  1.000000 1.00000 0.022354
2  0.1752113      1  0.734648 0.75718 0.019750
3  0.0504225      2  0.559437 0.56169 0.017215
4  0.0341784      4  0.458592 0.47944 0.015983
5  0.0338028      7  0.356056 0.42648 0.015122
6  0.0166197      8  0.322254 0.34366 0.013642
7  0.0086385     10  0.289014 0.30873 0.012956
8  0.0076056     13  0.263099 0.28620 0.012491
9  0.0073239     15  0.247887 0.25634 0.011842
10 0.0070423     16  0.240563 0.25465 0.011804
11 0.0061972     20  0.205070 0.24732 0.011638
12 0.0046948     21  0.198873 0.23775 0.011417
13 0.0045070     25  0.176338 0.22648 0.011150
14 0.0042254     26  0.171831 0.21915 0.010973
15 0.0036620     28  0.163380 0.21634 0.010904
16 0.0033803     30  0.156056 0.20845 0.010708
17 0.0028169     31  0.152676 0.20901 0.010723
18 0.0025352     34  0.144225 0.20451 0.010609
19 0.0024413     36  0.139155 0.19718 0.010422
20 0.0022535     40  0.129014 0.19718 0.010422
21 0.0016901     41  0.126761 0.19662 0.010407
22 0.0015023     46  0.118310 0.19324 0.010319
23 0.0014085     59  0.098592 0.19099 0.010260
24 0.0011268     61  0.095775 0.19155 0.010275
25 0.0010000     66  0.090141 0.18873 0.010201
```



   - 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



5) 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

```
> rpartpred<-predict(ptree, vali, type='class')
>
> library('e1071')
> confusionMatrix(rpartpred, vali$num) # Calculate the accuracy of the model
Confusion Matrix and Statistics

          Reference
Prediction    0    1
         0 3825   83
         1   50  307

               Accuracy : 0.9688
                 95% CI : (0.9632, 0.9738)
    No Information Rate : 0.9086
    P-Value [Acc > NIR] : < 2.2e-16

                  Kappa : 0.8049
 Mcnemar's Test P-Value : 0.005524

            Sensitivity : 0.9871
            Specificity : 0.7872
         Pos Pred Value : 0.9788
         Neg Pred Value : 0.8599
             Prevalence : 0.9086
         Detection Rate : 0.8968
   Detection Prevalence : 0.9163
      Balanced Accuracy : 0.8871

       'Positive' Class : 0
```

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

randomForest library 설명 pdf 추가하기

1) Install packages
   - First, install the "randomForest" packages and open library. If you successfully install the package, you can see this result.

```
> install.packages("randomForest")
Installing package into 'C:/Users/IRIS/Documents/R/win-library/3.3'
(as 'lib' is unspecified)
trying URL 'https://cran.rstudio.com/bin/windows/contrib/3.3/randomForest_4.6-12.zip'
Content type 'application/zip' length 179054 bytes (174 KB)
downloaded 174 KB

package 'randomForest' successfully unpacked and MD5 sums checked

The downloaded binary packages are in
        C:\Users\IRIS\AppData\Local\Temp\RtmpiiBzyM\downloaded_packages
> library(randomForest)
randomForest 4.6-12
Type rfNews() to see new features/changes/bug fixes.
```

2) 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.

```
# Read the data file
setwd("C:/Users/IRIS/Desktop/LAB2") # Set the path
calib<-read.csv(file="cali.csv") # Read the data file
n<-ncol(calib)
predictors<-calib[,1:(n-1)] # define predictors
response<-calib[,n] # define dependent
response<-as.factor(response)
```

3) 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.

```
# Make Random Forest model
treemodel.rf<-randomForest(x=predictors,y=response,ntree=10,localImp=TRUE)
importance(treemodel.rf) #see the variable importance
```

```
> importance(treemodel.rf) #see the variable importance
                  0         1 MeanDecreaseAccuracy MeanDecreaseGini
ch05_07  7.202203  5.201076             7.967823        351.98080
ch06_07  8.369702 11.402924             9.313473        307.30054
ch07_13  8.527303  3.863105             9.376689        173.67743
ch12_15  3.074672  2.950725             3.601961         80.75164
bt07_11  6.836213  1.211885             7.560387        155.09809
bt07_13  6.238629  3.385584             6.366157        169.40801
bt13_15  5.369043  8.015471             5.777114        165.08580
bt07_08r 3.288405  2.258688             4.352566        230.84018
bt07_09r 3.148676  2.463775             3.520286        157.58547
bt07_10r 1.961045  3.899565             2.183724        134.04839
bt07_11r 5.986419  3.098012             5.927089         91.56857
bt07_12r 3.066523  3.764299             3.904269        467.29463
bt07_13r 5.209843  3.794028             5.638488        143.54200
bt07_14r 3.095469  2.998907             3.585063         46.22862
bt07_15r 3.035397  2.263025             3.514363         67.19976
bt07_16r 2.634851  2.365692             3.051921        139.87489
bt11_15r 4.430801  3.702815             6.728866         72.97421
bt13_14r 3.800408  6.786711             4.929768         59.91621
bt13_15r 3.963724  3.876486             4.459240        145.90795
```

4) 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.".

```
# Calibration result
treemodel.pred_calib<-predict(treemodel.rf,newdata=calib)
write.table(treemodel.pred_calib,"cali_result.csv",sep=",",append=FALSE)
save(treemodel.rf,file = "wildfire_RF.RData")

# Validation result
valid<-read.csv(file="vali.csv")
treemodel.pred_valid<-predict(treemodel.rf,newdata=valid)
write.table(treemodel.pred_valid,"vali_result.csv",sep=",",append=FALSE)
```

```
> treemodel.rf

Call:
 randomForest(x = predictors, y = response, ntree = 10, localImp = TRUE)
               Type of random forest: classification
                     Number of trees: 10
No. of variables tried at each split: 4

        OOB estimate of  error rate: 1.94%
Confusion matrix:
      0    1 class.error
0 13670  127   0.0092049
1   175 1573   0.1001144
```

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

classification/regression 후에 mapping 보여주기

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','OOB
PredictorImportance','on');
```

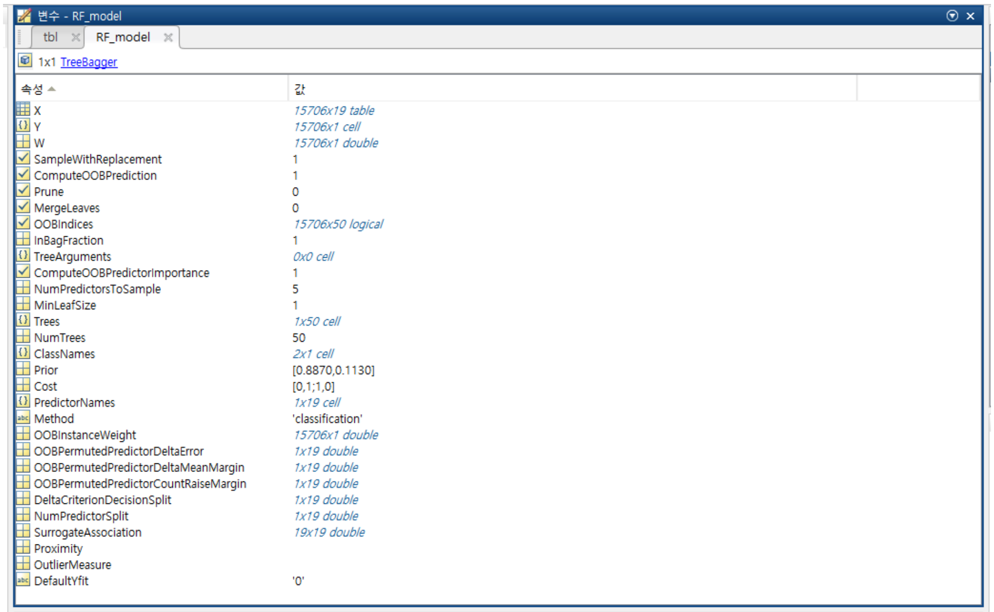


Figure 1 developed RF model

```
% 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';
```

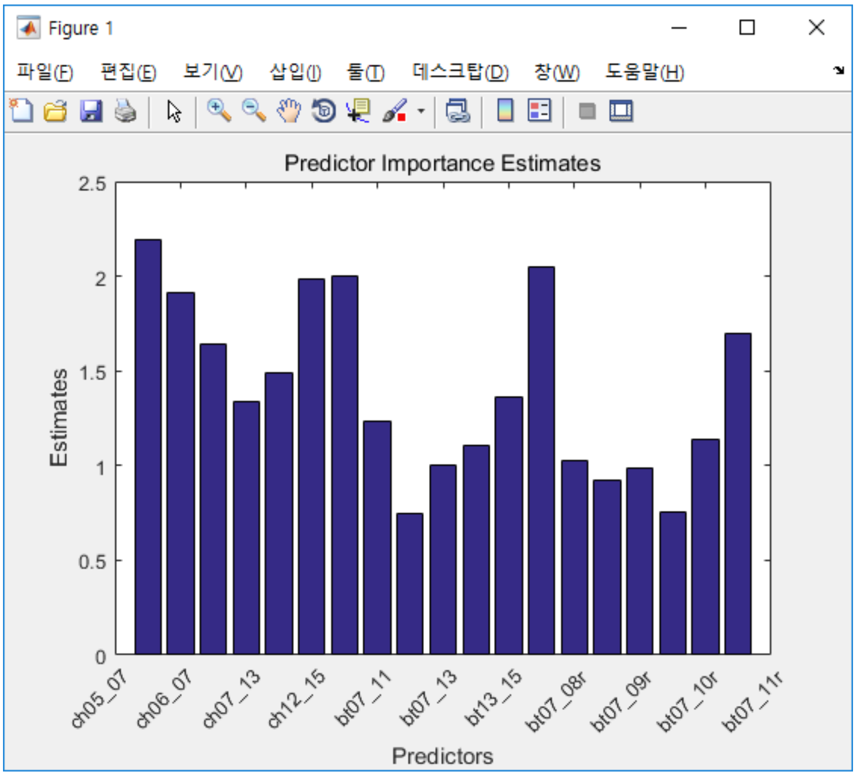


Figure 2 Variable importance

2) 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%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).