Classification of Sentinel-2 Multispectral Images using Deep Neural Network in R-H2O (Windows 10)

Zia Ahmed, PhD, Research Associate Professor, RENEW, University at Buffalo

March 22, 2018

This tutorial will show how to implement Deep Neural Network for pixel based supervised classification of Sentinel-2 multispectral images using H20 package in R.

H2O is an open source, in-memory, distributed, fast, and scalable machine learning and predictive analytics platform that allows you to build machine learning models on big data and provides easy productionalization of those models in an enterprise environment. It's core code is written in Java and can read data in parallel from a distributed cluster and also from local culster. H2O allows access to all the capabilities of H2O from an external program or script via JSON over HTTP. The Rest API is used by H2O's web interface (Flow UI), R binding (H2O-R), and Python binding (H2O-Python). Requirement and installation steps in R can be found here here.

We will use the Deep Neural Network algorithm using H20 package in R for image classification. First, we will split "point_data" into a training set (75% of the data), a validation set (12%) and a test set (13%) data.The validation data set will be used to optimize the model parameters during training process.The model's performance will be tested with the data set and then we will predict landuse clasess on grid data set. The point and grid data can be download as rar, 7z and zip format.

Tuning and Optimizations parameters:

- Four hidden layers with 200 neurons and Rectifier Linear (ReLU) as a activation function of neurons.
- The default stochastic gradient descent function will be used to optimize different objective functions and to minimize training loss.
- To reduce the generalization error and the risk of over-fitting of the model, we will use set low values for L1 and L2 regularizations.
- The model will be cross validated with 10 folds with stratified sampling
- The model will be run with 100 epochs.

More details of Tuning and Optimizations parameters of H20 Deep Neural Network for supervised classification can be found here

```
start_time <- Sys.time()
```

Load packages

```
library(rgdal)  # spatial data processing
library(raster) # raster processing
library(plyr)   # data manipulation
```

```r
library(dplyr)  # data manipulation
library(RStoolbox) # ploting spatial data
library(ggplot2) # plotting
library(RColorBrewer)
library(sp)
```

Set working directory

```r
setwd("F:\\My_GitHub\\DNN_H20_R")
```

Load point and grid data

```r
point<-read.csv("point_data.csv", header = T)
grid<-read.csv("grid_data.csv", header = T)
```

Creat data frames

```r
point.data<-cbind(point[c(4:13)],Class=point$Class)
grid.data<-grid[c(4:13)]
grid.xy<-grid[c(3,1:2)]
```

Install H2O

```r
#install.packages("h20")
```

Start and Initialize H20 local cluster

```r
library(h2o)
localH2o <- h2o.init(nthreads = -1, max_mem_size = "50G")
```

Import data to H2O cluster

```r
df<-  as.h2o(point.data)
grid<- as.h2o(grid.data)
```

Split data into train, validation and test dataset

```r
splits <- h2o.splitFrame(df, c(0.75,0.125), seed=1234)
train  <- h2o.assign(splits[[1]], "train.hex") # 75%
valid  <- h2o.assign(splits[[2]], "valid.hex") # 12%
test   <- h2o.assign(splits[[3]], "test.hex")  # 13%
```

Create response and features data sets

```r
y <- "Class"
x <- setdiff(names(train), y)
```

Deep Learning Model

```r
dl_model <- h2o.deeplearning(
              model_id="Deep_Learning",                  # Destination id for this model
              training_frame=train,                      # Id of the training data frame
              validation_frame=valid,                    # Id of the validation data frame
              x=x,                                       # a vector predictor variable
              y=y,                                       # name of reponse vaiables
              standardize=TRUE,                          # standardize the data
              score_training_samples=0,                  # training set samples for scoring (0 for all)
              activation = "RectifierWithDropout",       # Activation function
              score_each_iteration = TRUE,
              hidden = c(200,200,200,200),               # 4 hidden layers, each of 200 neurons
              hidden_dropout_ratios=c(0.2,0.1,0.1,0),    # for improve generalization
              stopping_tolerance = 0.001,                # tolerance for metric-based stopping criterion
              epochs=100,                                # the dataset should be iterated (streamed)
              adaptive_rate=TRUE,                        # manually tuned learning rate
              l1=1e-6,                                   # L1/L2 regularization, improve generalization
              l2=1e-6,
              max_w2=10,                                 # helps stability for Rectifier
```

```
              nfolds=10,                                  # Number of folds for K-fold cross-validation
              fold_assignment="Stratified",               # Cross-validation fold assignment scheme
              keep_cross_validation_fold_assignment = TRUE,
              seed=125,
              reproducible = TRUE,
              variable_importances=T
        )
```

Model Summary

```
#summary(dl_model)
#capture.output(print(summary(dl_model)),file =  "DL_summary_model_01.txt")
```

Mean error

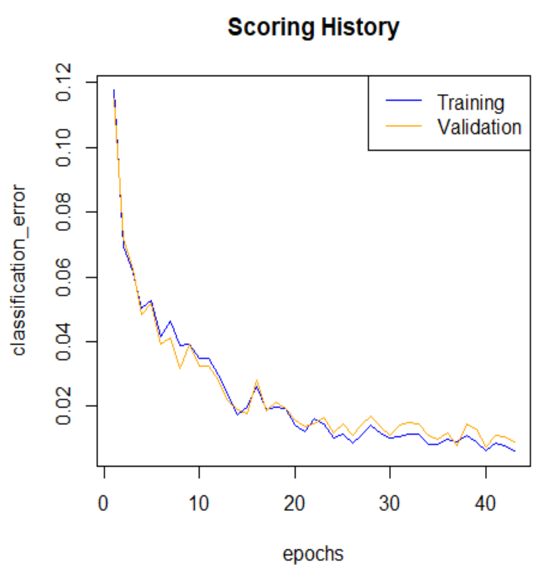
```
h2o.mean_per_class_error(dl_model, train = TRUE, valid = TRUE, xval = TRUE)

##       train       valid        xval
## 0.005951827 0.008541536 0.009924872
```

Scoring history

```
scoring_history<-dl_model@model$scoring_history
#write.csv(scoring_history, "scoring_history_model_02.csv")
```

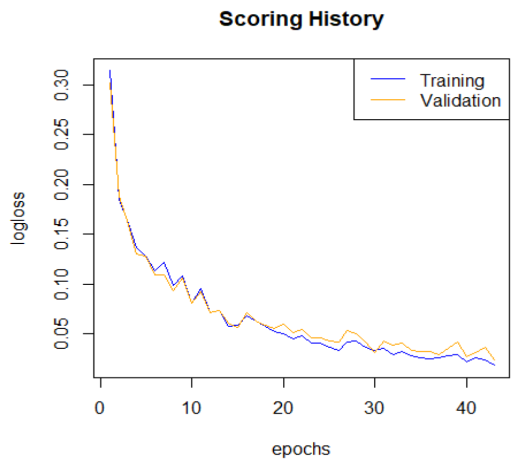
Plot the classification error

```
plot(dl_model,
     timestep = "epochs",
     metric = "classification_error")
```



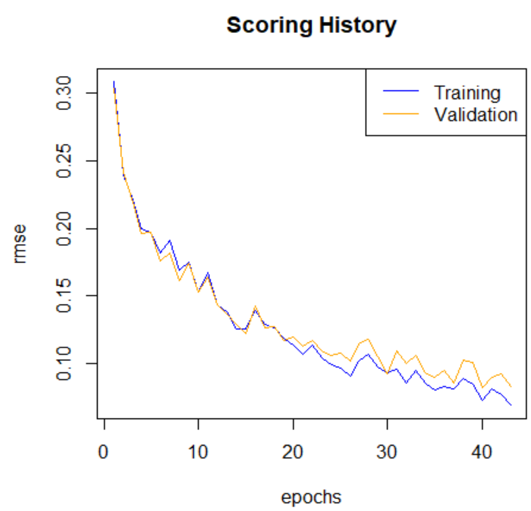
Plot logloss

```
plot(dl_model,
     timestep = "epochs",
     metric = "logloss")
```



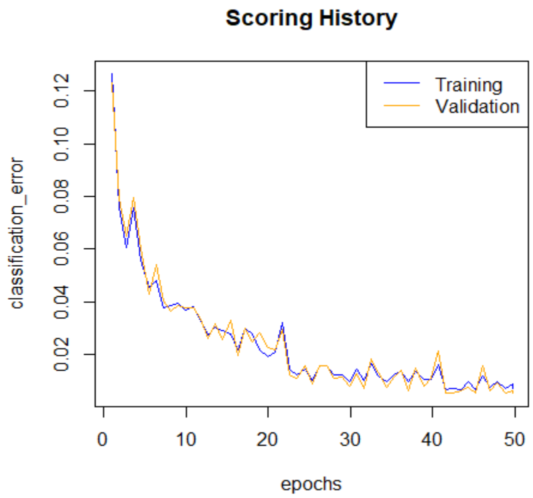
Plot RMSE

```
plot(dl_model,
     timestep = "epochs",
     metric = "rmse")
```



Cross-validation Error

```
# Get the CV models from the deeplearning model object` object
cv_models <- sapply(dl_model@model$cross_validation_models,
                    function(i) h2o.getModel(i$name))
# Plot the scoring history over time
plot(cv_models[[1]],
     timestep = "epochs",
     metric = "classification_error")
```



Cross validation result

```
print(dl_model@model$cross_validation_metrics_summary%>%.[,c(1,2)])

##                                mean           sd
## accuracy                   0.989942   0.00306049
## err                     0.010057977   0.00306049
## err_count                      18.2     5.681549
## logloss                 0.032726314   0.01081141
## max_per_class_error      0.02872035 0.0097002955
## mean_per_class_accuracy  0.99011856 0.0029896488
## mean_per_class_error    0.009881463 0.0029896488
## mse                     0.008144272 0.0023309013
## r2                       0.99390364 0.0017807437
## rmse                    0.088454194  0.012651616

#capture.output(print(dl_model@model$cross_validation_metrics_summary%>%.[,c(1,2)]),file =  "DL_CV_model_01.txt")
```

Model performance with Test data set

Compare the training error with the validation and test set errors

```
h2o.performance(dl_model, newdata=train)     ## full train data

## H2OMultinomialMetrics: deeplearning
##
## Test Set Metrics:
## =====================
##
## MSE: (Extract with `h2o.mse`) 0.004795433
## RMSE: (Extract with `h2o.rmse`) 0.06924907
## Logloss: (Extract with `h2o.logloss`) 0.01829286
## Mean Per-Class Error: 0.005951827
## Confusion Matrix: Extract with `h2o.confusionMatrix(<model>, <data>)`
## =========================================================================
## Confusion Matrix: Row labels: Actual class; Column labels: Predicted class
##         Class_1 Class_2 Class_3 Class_4 Class_5  Error            Rate
## Class_1    4150      15       1       0       0 0.0038 =    16 / 4,166
## Class_2      48    3249      23       0       0 0.0214 =    71 / 3,320
## Class_3      11       0    6102       2       0 0.0021 =    13 / 6,115
## Class_4       0       0       9    3730       0 0.0024 =     9 / 3,739
## Class_5       0       0       0       0     694 0.0000 =      0 / 694
```

```
## Totals     4209    3264    6135    3732     694 0.0060 = 109 / 18,034
##
## Hit Ratio Table: Extract with `h2o.hit_ratio_table(<model>, <data>)`
## =======================================================================
## Top-5 Hit Ratios:
##   k hit_ratio
## 1 1  0.993956
## 2 2  1.000000
## 3 3  1.000000
## 4 4  1.000000
## 5 5  1.000000
```

```r
h2o.performance(dl_model, newdata=valid)     ## full validation data
```

```
## H2OMultinomialMetrics: deeplearning
##
## Test Set Metrics:
## =====================
##
## MSE: (Extract with `h2o.mse`) 0.00687076
## RMSE: (Extract with `h2o.rmse`) 0.08289005
## Logloss: (Extract with `h2o.logloss`) 0.02419423
## Mean Per-Class Error: 0.008541536
## Confusion Matrix: Extract with `h2o.confusionMatrix(<model>, <data>)`)
## =========================================================================
## Confusion Matrix: Row labels: Actual class; Column labels: Predicted class
##         Class_1 Class_2 Class_3 Class_4 Class_5  Error         Rate
## Class_1     657       3       0       0       0 0.0045 =    3 / 660
## Class_2       8     529       7       0       0 0.0276 =   15 / 544
## Class_3       3       0     999       1       0 0.0040 =  4 / 1,003
## Class_4       0       0       4     602       0 0.0066 =    4 / 606
## Class_5       0       0       0       0     111 0.0000 =    0 / 111
## Totals      668     532    1010     603     111 0.0089 = 26 / 2,924
##
## Hit Ratio Table: Extract with `h2o.hit_ratio_table(<model>, <data>)`
## =======================================================================
## Top-5 Hit Ratios:
##   k hit_ratio
## 1 1  0.991108
## 2 2  1.000000
## 3 3  1.000000
## 4 4  1.000000
## 5 5  1.000000
```

```r
h2o.performance(dl_model, newdata=test)     ## full test data
```

```
## H2OMultinomialMetrics: deeplearning
##
## Test Set Metrics:
## =====================
##
## MSE: (Extract with `h2o.mse`) 0.00642109
## RMSE: (Extract with `h2o.rmse`) 0.0801317
## Logloss: (Extract with `h2o.logloss`) 0.02786502
## Mean Per-Class Error: 0.007264487
## Confusion Matrix: Extract with `h2o.confusionMatrix(<model>, <data>)`)
## =========================================================================
## Confusion Matrix: Row labels: Actual class; Column labels: Predicted class
##         Class_1 Class_2 Class_3 Class_4 Class_5  Error         Rate
## Class_1     704       3       1       0       0 0.0056 =    4 / 708
## Class_2       8     553       4       0       0 0.0212 =   12 / 565
```

```
## Class_3       3       0     976       0       0 0.0031 =    3 / 979
## Class_4       0       0       4     624       0 0.0064 =    4 / 628
## Class_5       0       0       0       0     107 0.0000 =    0 / 107
## Totals      715     556     985     624     107 0.0077 = 23 / 2,987
##
## Hit Ratio Table: Extract with `h2o.hit_ratio_table(<model>, <data>)`
## =======================================================================
## Top-5 Hit Ratios:
##   k hit_ratio
## 1 1  0.992300
## 2 2  1.000000
## 3 3  1.000000
## 4 4  1.000000
## 5 5  1.000000

#capture.output(print(h2o.performance(dl_model,test)),file =
"test_data_model_01.txt")
```

Confusion matrix

```
train.cf<-h2o.confusionMatrix(dl_model)
print(train.cf)
valid.cf<-h2o.confusionMatrix(dl_model,valid=TRUE)
print(valid.cf)
test.cf<-h2o.confusionMatrix(dl_model,test)
print(test.cf)
#write.csv(train.cf, "CFM_train_model_01.csv")
#write.csv(valid.cf, "CFM_valid_model_01.csv")
#write.csv(test.cf, "CFM_test_moldel_01.csv")
```

Grid Prediction

```
g.predict = as.data.frame(h2o.predict(object = dl_model, newdata = grid))
```

Stop h20 cluster

```
h2o.shutdown(prompt=FALSE)

## [1] TRUE
```

Extract Prediction Class

```
grid.xy$Class<-g.predict$predict
str(grid.xy)

## 'data.frame':    35700 obs. of  4 variables:
##  $ ID   : int  1 2 3 4 5 6 7 8 9 10 ...
##  $ x    : int  677775 677785 677795 677805 677815 677825 677835 677845 677855 677865 ...
##  $ y    : int  4764065 4764065 4764065 4764065 4764065 4764065 4764065 4764065 4764065
4764065 ...
##  $ Class: Factor w/ 5 levels "Class_1","Class_2",..: 1 1 1 1 1 1 3 1 2 1 ...

grid.xy.na<-na.omit(grid.xy)
```

Join Class Id Column

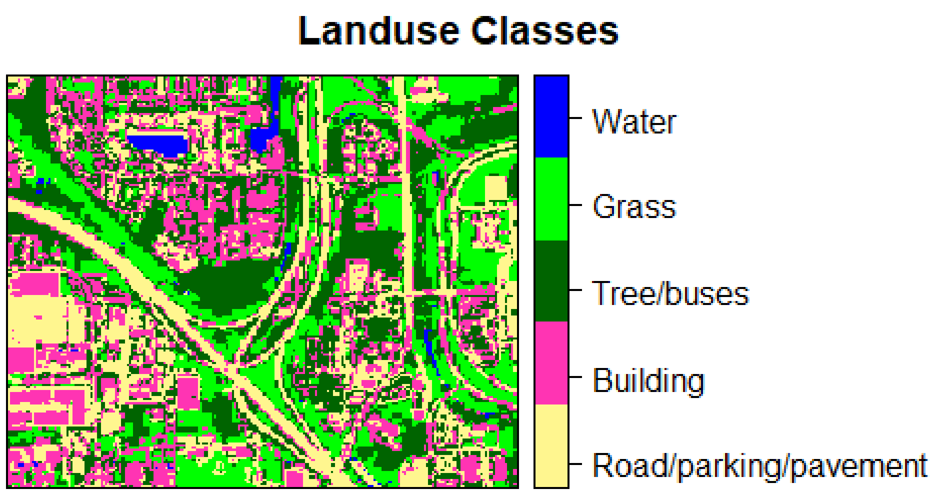
```
ID<-read.csv("Landuse_ID_h20.csv", header=TRUE)
new.grid<-join(grid.xy.na, ID, by="Class", type="inner")
#write.csv(new.grid, "Predicted_Landuse_Class.csv")
```

Convert to raster and write

```
x<-SpatialPointsDataFrame(as.data.frame(new.grid)[, c("x", "y")], data = new.grid)
r <- rasterFromXYZ(as.data.frame(x)[, c("x", "y", "Class_ID")])
#writeRaster(r,"predicted_Landuse.tiff","GTiff",overwrite=TRUE)
```

Plot and Save as a tiff file

```
myPalette <- colorRampPalette(c("khaki1","maroon1", "darkgreen","green", "blue"))
lu<-spplot(r,"Class_ID", main="Landuse Classes" ,
      colorkey = list(space="right",tick.number=1,height=1, width=1.5,
              labels = list(at = seq(1,4.8,length=5),cex=1.0,
              lab = c("Road/parking/pavement" ,"Building", "Tree/buses", "Grass", "Water"))),
              col.regions=myPalette,cut=4)
lu
```




```
# Save as tif file
windows(width=4, height=4)
tiff( file="FIGURE_Landuse_Class.tif",
      width=4,
      height=4,
      units = "in",
      pointsize = 12,
      res=600,
      restoreConsole = T,
      compression =  "lzw",
      bg="transparent")
print(lu)
dev.off()

## png
##   2
```

Run time

```r
end_time <- Sys.time()
end_time - start_time

## Time difference of 30.42456 mins
```

Conclusions

This simple pixel-based satellite image classification algorithm with deep neural network with H20-R able to identify urban objects with very high accuracy. It may be use full for landuse classification for urban environment monitoring as well as planning purpose. Also, may use full for agricultural landuse classification.