DATA ANALYTICS USING R

'EXPLORING SCALABILITY CHALLENGES IN R USING H2O'

APRIL 26, 2017

REPORT

CONTENTS

I. What is H2O?

H2O is an open source scalable platform that can be used for data analysis using big-data. One of the main factors that led to the development of H2O was when the now CEO of H2O.ai, SriSatish Ambati got frustrated with the performance of R when using large datasets.

With the methods in place, H2O allows the user to use large datasets without the need to sample the dataset because of performance constraints. Also, it is available to be used in R, Python and Scala and can be used for datasets in Hadoop, cloud (AWS) or the operating system environment.

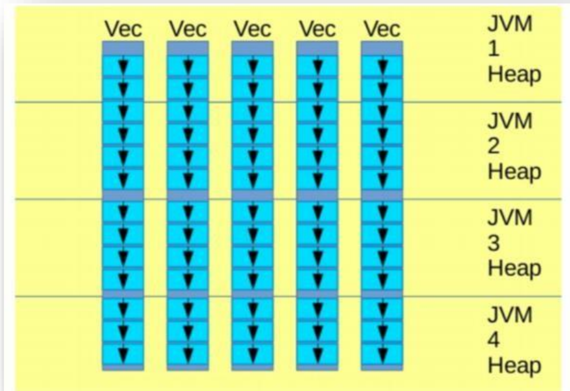
II. Why use H20?

The key factors that have led to increased usage of H2O are the following:

- Seamless integration with Big Data technology like Apache Hadoop & Spark
- High speed - especially matters when working with Big Data
- Allows the use of the whole data without the need to sample
- Offers various ensemble algorithms like random forest, GBM and neural nets which in general take a longer time due to iterative nature of the process

III. How does H20 Work?

With R being implemented by statisticians and not software engineers one of the biggest challenges associated with R is scalability. The current R environment fails when dealing with huge data sets or more complex ensemble models like Random Forest, Gradient Boosting and Deep Learners. H2O has done to R what MapReduce has done for Hadoop.H2O has taken these algorithms' implementations and implemented them using distributed file systems based on Java Virtual Machine Environment.



Structure of how H2O frames are stored as distributed arrays (Source:H2O.ai)

IV. Starting with H2O

For running H2O on R, we would require Java version beyond 1.7 and R version starting 2.13.0. The package for the same can be installed from CRAN and starting an instance of H2O it will set the default heap size to 1GB for a 32-bit Java and 25% of the total memory available for the 64-bit version. Below are the commands for installing and initializing an instance of H2O

```
#installing h2o and loading all the packages on which h20 had dependencies
install.packages("h2o")
library(h2o)

# to start and connect to an H2O instance
localH2O <- h2o.init(ip = "localhost", port =54321, startH2O = TRUE ,max_mem_size = "25g",
nthreads = -1 )
```

In the above code, post installing and loading H2O we initialize an instance of H2O using the function h2o.init and set the IP and port parameters as in the code above. Parameter max_mem_size is used to set the total memory available to Java machine and the parameter nthread can be used to specify the number of parallel threads that need to run. Setting nthreads as -1 allows JVM to use all available threads.

The screenshot below shows the details of the H2O cluster created in R and this information can be viewed at any point of time using h2o.clusterInfo(cluster_name).

```
> localH2Oinstance <- h2o.init(ip = "localhost", port =54321, startH2O = TRUE ,max_mem_size = "25g" )
 Connection successful!

R is connected to the H2O cluster:
    H2O cluster uptime:         3 hours 43 minutes
    H2O cluster version:        3.10.4.4
    H2O cluster version age:    10 days
    H2O cluster name:           H2O_started_from_R_pru_cjz772
    H2O cluster total nodes:    1
    H2O cluster total memory:   12.83 GB
    H2O cluster total cores:    4
    H2O cluster allowed cores:  2
    H2O cluster healthy:        TRUE
    H2O Connection ip:          localhost
    H2O Connection port:        54321
    H2O Connection proxy:       NA
    H2O Internal Security:      FALSE
    R Version:                  R version 3.3.2 (2016-10-31)
```

V. Exploring data using H2O

1. Importing Data:

Though we can use the regular functions for loading data into R, but when a user is dealing with huge datasets, they can use the following function to seamlessly load huge datasets into R. In case the user is uploading dataset present on HDFS they could use the h2o.importFile function.

bigdata.hex = h2o.importFile(path = "C:/R/Project/New folder/3 gb/c1.csv", destination_frame = "bigdata.hex")

For further use, this h2o object can be converted in the dataframe using as.data.frame(). In case there is a dataframe present in R that needs to be passed into a h2o instance, the function as.h2o() can be used for the same.

On top of this imported data we could run our normal functions like min(), max(), summary(), plot(), colnames() to perform data exploration process.

2. Modeling using H2O

For building the model, we follow the following steps – split the dataset into train, validation and test and then finally build and optimize models based on these. Since the RentHop data is multinomial in nature we have explored the usage of the following models – Gradient Boosting, Random Forest, Deep Learning Algorithms, and Stacked Ensemble models.

3. Splitting the Dataset

H2O provides functions for splitting the dataset into test and train. This function is especially stable when working on huge datasets.

```
splits <- h2o.splitFrame(data = train, ratios = c(0.6,0.2), destination_frames = c("train.hex",
"valid.hex", "test.hex"))
```

4. Building Models

i. Gradient Boosting

The gradient boosting algorithm implementation in H2O along with its tuning capabilities, is one of the most used algorithm in Kaggle Competitions and gives a performance very like the top used packages like XGBoost. Below is the code for implementing the same using H2O.

```
gbm2= h2o.gbm (x= varnames, y="interest_level", training_frame = train ,
               validation_frame = valid, ntrees = 1000, learn_rate = 0.01,
               stopping_metric = "misclassification")
```

In the above function, training_frame and validation_frame is used to specify the training and validation datasets, ntrees for the number of trees, learning_rate specifies how fasts the algorithms tries moving towards the stopping metric's minima, stopping metric is normally an error function based on which the function is optimized. Other than these we can even add parameters like max_depth (for specifying the maximum depth of the trees), distribution (specify the distribution y is following), stopping_tolerance and stopping_rounds (two parameters that specify after how much change and how many rounds without change should the algorithm stop running), sample_rate & col_sample_rate (the percentage of rows and columns sampled randomly (respectively) to build a tree), model_id (id for identifying each model that is built), seed.

Summarizing the GBM function returns the misclassification matrix for the given model.

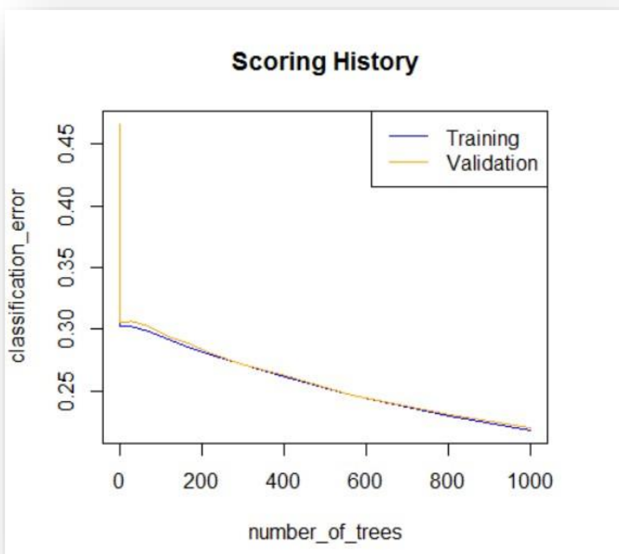
summary(gbm2)

Console ~/

```
> summary(gbm2)
Model Details:
==============

H2OMultinomialModel: gbm
Model Key:  GBM_model_R_1493174250716_1
Model Summary:
  number_of_trees number_of_internal_trees model_size_in_bytes min_depth max_depth mean_depth min_leaves max_leaves
1            1000                     3000          1097317598         5         5    5.00000         10         32
  mean_leaves
1    27.78933

H2OMultinomialMetrics: gbm
** Reported on training data. **

Training Set Metrics:
=====================

Extract training frame with `h2o.getFrame("train.hex")`
MSE: (Extract with `h2o.mse`) 0.1752949
RMSE: (Extract with `h2o.rmse`) 0.4186823
Logloss: (Extract with `h2o.logloss`) 0.5149219
Mean Per-Class Error: 0.4391641
Confusion Matrix: Extract with `h2o.confusionMatrix(<model>,train = TRUE)`)
=========================================================================
Confusion Matrix: vertical: actual; across: predicted
       high   low medium  Error             Rate
high    866   938    904 0.6802 =  1,842 / 2,708
low      34 22982    973 0.0420 = 1,007 / 23,989
medium  133  4536   3174 0.5953 =  4,669 / 7,843
Totals 1033 28456   5051 0.2177 = 7,518 / 34,540

Hit Ratio Table: Extract with `h2o.hit_ratio_table(<model>,train = TRUE)`
=======================================================================
Top-3 Hit Ratios:
  k hit_ratio
1 1  0.782339
2 2  0.954777
3 3  1.000000
```

```
Variable Importances: (Extract with `h2o.varimp`)
=================================================

Variable Importances:
       variable relative_importance scaled_importance percentage
1         price        55274.785156          1.000000   0.228898
2   building_id        47013.191406          0.850536   0.194686
3    manager_id        44890.882812          0.812140   0.185897
4      bedrooms        23851.207031          0.431502   0.098770
5          hour        13778.770508          0.249278   0.057059
6    num_photos        11948.094727          0.216158   0.049478
7  num_features        11134.541992          0.201440   0.046109
8      latitude         8769.466797          0.158652   0.036315
9     longitude         8422.038086          0.152367   0.034876
10    bathrooms         7738.314453          0.139997   0.032045
11   listing_id         5137.980469          0.092953   0.021277
12         mday         1867.085327          0.033778   0.007732
13         wday          817.627625          0.014792   0.003386
14         yday          762.360474          0.013792   0.003157
15        month           75.641563          0.001368   0.000313
```

plot(gbm2)



Hyper-Parameter search

We can perform hyper-parameter search, which helps in model tuning and finding the optimized model. Below is the code to find the best value for max_depth:

hyper_params = list(max_depth = seq(1,10,2))

grid <- h2o.grid(hyper_params = hyper_params, search_criteria = list(strategy = "Cartesian"),

algorithm="gbm", grid_id="depth_grid", x = varnames, y = "interest_level", training_frame = testData, ntrees = 100, learn_rate = 0.05)

The hyper_params parameter in the h2o.grid function takes the list of values to iterate through for the parameter to be optimized, search_criteria is an optional parameter used to specify more advanced search strategies. Other than these there are parameters for algorithm to be optimized, similar to model_id we have grid_id here.

The output is as shown below, which can later be sorted by any other metrics as required

```
Console ~/
+                 learn_rate = 0.05)
  |==========================================================================================
> grid
H2O Grid Details
================

Grid ID: depth_grid
Used hyper parameters:
  -  max_depth
Number of models: 5
Number of failed models: 0

Hyper-Parameter Search Summary: ordered by increasing logloss
  max_depth          model_ids             logloss
1         9 depth_grid_model_4 0.07809272874125063
2         7 depth_grid_model_3 0.13476834294817208
3         5 depth_grid_model_2 0.26961415503056224
4         3 depth_grid_model_1 0.48001237265324825
5         1 depth_grid_model_0  0.6747465830288698
> |
```

ii. Random Forest

Like the implementation in R, the Random Forest can be easily implemented in H2O using the following code. And like the GBM function in h2o, random forest also offers an array of parameters to add to the function. Some of these parameters are ntrees, max_depth, stopping_rounds, stopping_tolerance, model_id, seed. These parameters have meaning similar to that in GBM.

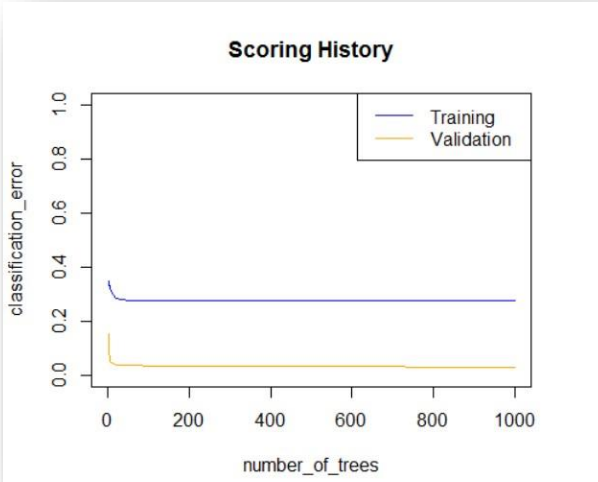
rf1 = h2o.randomForest(training_frame = train, validation_frame = valid, x=varnames,y="interest_level", model_id = "rf1", ntrees = 1000)

summary(rf1)

```
Hit Ratio Table: Extract with `h2o.hit_ratio_table(<model>,train = TRUE)`
=======================================================================
Top-3 Hit Ratios:
  k hit_ratio
1 1  0.724754
2 2  0.930805
3 3  1.000000


H2OMultinomialMetrics: drf
** Reported on validation data. **

Validation Set Metrics:
=====================

Extract validation frame with `h2o.getFrame("valid.hex")`
MSE: (Extract with `h2o.mse`) 0.07230787
RMSE: (Extract with `h2o.rmse`) 0.2689012
Logloss: (Extract with `h2o.logloss`) 0.2669147
Mean Per-Class Error: 0.06391979
Confusion Matrix: Extract with `h2o.confusionMatrix(<model>,valid = TRUE)`)
=========================================================================
Confusion Matrix: vertical: actual; across: predicted
       high  low medium  Error          Rate
high    512   43      2 0.0808 =    45 / 557
low       0 4717      1 0.0002 =   1 / 4,718
medium    1  173   1397 0.1108 = 174 / 1,571
Totals  513 4933   1400 0.0321 = 220 / 6,846

Hit Ratio Table: Extract with `h2o.hit_ratio_table(<model>,valid = TRUE)`
=======================================================================
Top-3 Hit Ratios:
  k hit_ratio
1 1  0.967864
2 2  0.999562
3 3  1.000000
```

```
Variable Importances: (Extract with `h2o.varimp`)
=================================================

Variable Importances:
       variable relative_importance scaled_importance percentage
1    manager_id      1345609.750000          1.000000   0.179706
2   building_id      1044124.562500          0.775949   0.139443
3         price       853520.562500          0.634300   0.113987
4          hour       532084.750000          0.395423   0.071060
5    num_photos       482706.031250          0.358727   0.064465
6      latitude       465312.781250          0.345801   0.062142
7  num_features       460738.531250          0.342401   0.061532
8     longitude       440556.625000          0.327403   0.058836
9          mday       381053.781250          0.283183   0.050890
10   listing_id       345279.093750          0.256597   0.046112
11         yday       335581.031250          0.249390   0.044817
12     bedrooms       299559.062500          0.222620   0.040006
13         wday       272677.906250          0.202643   0.036416
14    bathrooms       149795.671875          0.111322   0.020005
15        month        79246.992188          0.058893   0.010583
>
```

plot(rf1)



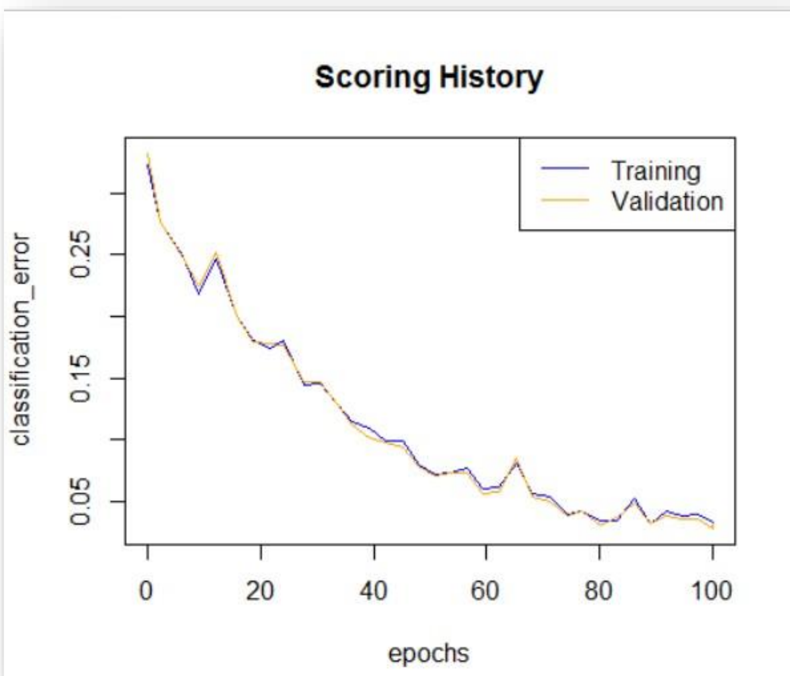
iii. Deep Learning

Deep learning models can be implemented in H2O using the following code. For deep learning some of the parameters that can be used are training_frame and validation_frame (for specifying the train and the validation dataset), model_id, hidden is used to specify the information about the hidden layers (count of elements in the assigned vector is the number of layers and each number specifies the number of neurons in each hidden layer), epochs (specifies how many times would the neural net be exposed to each observation in the dataset at least once), variable_importance (returns variable importance, by default set to false)

```
deepLearningModel <- h2o.deeplearning(x=varnames, y= "interest_level",
                                       training_frame = train,
                                       hidden=c(32,32,32),
                                       validation_frame = valid, epochs = 100)

summary(deepLearningModel)
```

```
Console ~/

Validation Set Metrics:
=====================

Extract validation frame with `h2o.getFrame("valid.hex")`
MSE: (Extract with `h2o.mse`) 0.02272624
RMSE: (Extract with `h2o.rmse`) 0.1507523
Logloss: (Extract with `h2o.logloss`) 0.07986127
Mean Per-Class Error: 0.06148502
Confusion Matrix: Extract with `h2o.confusionMatrix(<model>,valid = TRUE)`)
=========================================================================
Confusion Matrix: vertical: actual; across: predicted
       high  low medium  Error          Rate
high    502   13     42 0.0987 =    55 / 557
low       4 4680     34 0.0081 =   38 / 4,718
medium   12  110   1449 0.0777 =  122 / 1,571
Totals  518 4803   1525 0.0314 =  215 / 6,846

Hit Ratio Table: Extract with `h2o.hit_ratio_table(<model>,valid = TRUE)`
=======================================================================
Top-3 Hit Ratios:
  k hit_ratio
1 1  0.968595
2 2  0.997371
3 3  1.000000
```

plot(deepLearningModel)



Exp l oring deep l earni ng Tensorflow mod el s usin

***g H2O* Dependencies:**

- train.json, vgg_16.ckpt – has the weights
- imagenet_classes.txt – has the classes

Input:

- train.json, vgg_16.ckpt
- imagenet_classes.txt

Intermediary Output:

- images.csv (a*fter extracting the images)*

Final Output:

- image_classification.csv

- **About Tensorflow**

Tensorflow is an open-source library written in python and C++ which was a byproduct of the Google brain Team and is used for machine learning purposes. It uses nodes and edges for mathematical computation where, the nodes represent mathematical operations and the edges represent tensors which are multidimensional arrays.

Tensor has a static type which is an n-dimensional array or list with varying dimensions. A tensor is described by a rank,type and shape.

Rank - refers to the order of degree.

For example,

0 => a = 12 => scalar, 1 => a = [1,2,3] => vector, 2 => a = [[1,2,3],[4,5,6],[7,8,9]] => matrix, n-tensor => a = [[[N1], [N2], [N3]...[Nn], ...]

Type- tensors can have the following data types:

DT_FLOAT - 32 bits floating point.
DT_DOUBLE - 64 bits floating point.
DT_INT8 - 8 bits signed integer.
DT_INT16 - 16 bits signed integer.

DT_INT32 - 32 bits signed integer.
DT_INT64 - 64 bits signed integer.
DT_UINT8 - 8 bits unsigned integer.
DT_STRING - Variable length byte arrays. Each element of a Tensor is a byte array.
DT_BOOL - Boolean.
DT_COMPLEX64 - Complex number made of two 32 bits floating points: real and imaginary parts.
DT_COMPLEX128 - Complex number made of two 64 bits floating points: real and imaginary parts.
DT_QINT8 - 8 bits signed integer used in quantized Ops.
DT_QINT32 - 32 bits signed integer used in quantized Ops.
DT_QUINT8 - 8 bits unsigned integer used in quantized Ops.

Shape - [] => A 0-D tensor, [7] => A 1-D tensor with shape [7], [9, 10] A 2-D tensor with shape [9, 10], [D0, D1, ... Dn-1] => A tensor with shape [D0, D1, ... Dn-1]

- **Tensorflow in R**

The Tensorflow API can be used in R by making use of the tensoflow library as shown below:

```
library(tensorflow)
```

The Tensorflow API has been used for the purpose of image classification in this project.

- **Tensorflow on H2o**

The Tensorflow API can also be accessed within the H2o environment by loading the library once the H2o environment has been setup as shown below:

```
install.packages("h2o")
library(h2o)
library(tm)
demo(h2o.glm)
localH2O = h2o.init()
localH2O <- h2o.init(ip = "localhost", port =54321, startH2O = TRUE)
#Installing and loading packages and libraries for image processing and classification
install.packages("tensorflow")
```

The library slim is used for image classification in this case. Slim acts as a lightweight wrapper which contains pretrained models. In this proof of concept the VGG16 pretrained model is used for image classification which contains 1000 classes.

- **Image classification using Tensorflow and vgg16 on renthop dataset**

Clearing all objects and installing the required packages and libraries

```r
#Clearing all objects
rm(list = ls())

#Installing packages required for image processing and classification
install.packages("rjson")
install.packages("tensorflow")
install.packages("jpeg")
install.packages("readr")
install.packages("h2o")
#___________________________________________________________________________

#loading libraries required for image processing and classification
library(tensorflow)
library(magrittr)
library(jpeg)
library(readr)
library(grid)
library(ggplot2)
library(stringr)
library("rjson")
library(dplyr)
library(jsonlite)
library(h2o)
#___________________________________________________________________________
```

Step1: Start and connect to an H2o instance

```r
localH2Oinstance <- h2o.init(ip = "localhost", port =54321, startH2O = TRUE ,max_mem_size = "25g" )
```

Step 2: Extracting images for each set and preparing the image dataset for classification

```
#loading renthop dataset  from the local machine
json_file<- "C:/Users/tjneh/BAPM/R/train.json"

#the datset obtained is in json format and the logic below is used to convert it into a dataframe
json_str<- paste(readLines(json_file), collapse = "")
json_dat<- fromJSON(json_str)
train = as.data.frame(t(do.call(rbind, json_dat)))


#extracting the colums containing the photos only by using a subset
images = train["photos"]

#as a proof of concept a sample of the data - 100 rows is taken for image classification
#the image for each house is in the form a list - a list containing images of bathroom,kitchen,living room,clostes,foo
#the function extractimage extracts the images in the list and creates rows for a house-image combination
#eg: [row 1 col1] = house1, [row1,col2] = image1, [row 2 col1] = house1, [row2,col2] = image2 and so on for each house
extractImages = function(images)
{
  count = 0
  output = data.frame()
  row_total = nrow(images)

  for(image_list in 1:100)
  {
#unlisting the images
    l = unlist(images[image_list,1])
    l = as.vector(l)
#initialising the count variable to get the house id
    count = count + 1

    for(image in 1:length(l))
    {
#getting the url link to the image/input "Image not found" in case the url was blank
      house = cbind("house",count, ifelse(is.null(l[image]),"Image not found",l[image]))
      output = rbind(output, as.data.frame(house))
    }
```

```
  }
#writing the output to an images file in csv format
  write.csv(output,"C:/Users/tjneh/BAPM/R/images.csv")

}

data = read.csv("C:/Users/tjneh/BAPM/R/images.csv")


#subsetting the data variable to get only the image urls from the column named v3
data = data["V3"]

#
```

A sample images.csv file looks like below – it has the house, house id and the link to the image

	V1	count	V3
1	house	1	https://photos.renthop.com/2/7170325_3bb5ac84a5a10227b17b273e79bd77b4.jpg
2	house	1	https://photos.renthop.com/2/7170325_a29a17a771ee6af213966699b05c8ea2.jpg
3	house	1	https://photos.renthop.com/2/7170325_149a898e8760cac1cad56e30cfe98baa.jpg
4	house	1	https://photos.renthop.com/2/7170325_f74a43d781bcc3c5588e61dd47de81ba.jpg
5	house	1	https://photos.renthop.com/2/7170325_e677d9d249ac99abe01aa5454c6e9f59.jpg
6	house	1	https://photos.renthop.com/2/7170325_960ea0e180bf2f15467b68b455db6172.jpg
7	house	1	https://photos.renthop.com/2/7170325_cbc1b8437155dbf7f5d63b3a0b5a45a3.jpg
8	house	1	https://photos.renthop.com/2/7170325_9a9f2adc2ce922e1d5394727efdf64bb.jpg
9	house	1	https://photos.renthop.com/2/7170325_aae2a39d536103eebb282775fab1c315.jpg
10	house	1	https://photos.renthop.com/2/7170325_cd290d0051b9f08e3482195dcbf6b5a6.jpg
11	house	1	https://photos.renthop.com/2/7170325_a2b599da7880eea1edd10c4b04250dc1.jpg
12	house	1	https://photos.renthop.com/2/7170325_6b83fa82d662bcb09733ac3a8a107113.jpg
13	house	2	https://photos.renthop.com/2/7092344_7663c19af02c46104bc4c569f7162ae0.jpg
14	house	2	https://photos.renthop.com/2/7092344_8287349abe511d195a7b6129bf24af0e.jpg
15	house	2	https://photos.renthop.com/2/7092344_e9e6a2b7aa95aa7564fe3318cadcf4e7.jpg
16	house	2	https://photos.renthop.com/2/7092344_d51ee4b92fd9246633f93afe6e86d8f0.jpg
17	house	2	https://photos.renthop.com/2/7092344_f0573fa184ca130b1b6000f2fa90511c.jpg
18	house	2	https://photos.renthop.com/2/7092344_b2a62f769a59a317b0a243000db46fd0.jpg
19	house	3	https://photos.renthop.com/2/7158677_c897a134b77dc1c772a663874da69315.jpg
20	house	3	https://photos.renthop.com/2/7158677_cbf67ce22b270aeefe274de0e3767e5f.jpg
21	house	3	https://photos.renthop.com/2/7158677_c10b443e36a92a8db1d3c52b955045f0.jpg
22	house	3	https://photos.renthop.com/2/7158677_96705ca1b7fa414863eacf7d5050c544.jpg
23	house	3	https://photos.renthop.com/2/7158677_b994d4d8ec48f9ec099511883eb5fe9c.jpg
24	house	3	https://photos.renthop.com/2/7158677_d577322b9e7cd1ed16a301917db9cd90.jpg
25	house	4	https://photos.renthop.com/2/7211212_1ed4542ec81621d70d1061aa833e669c.jpg
26	house	4	https://photos.renthop.com/2/7211212_7dfc41dced69245065df83d08eed4a00.jpg
27	house	4	https://photos.renthop.com/2/7211212_c17853c4b869af6f53af08b0f5820b4c.jpg
28	house	4	https://photos.renthop.com/2/7211212_787ad8ea0c089792e7453e2121f8ac89.jpg

Step 3: Image classification using slim library and pretrained vvg16 model

```r
imageClassifier = function(image_link)
{
  #TF-slimmodule is a lightweight library for defining, training and evaluating complex models in TensorFlow.
  #Components of tf-slimmodule can be freely mixed with native tensorflow, as well as other frameworks,
  #such as tf.contrib.learn.
  #Importing tensorflow.contrib.slimmodule as slimmodule
  slimmodule = tf$contrib$slim

  #resetting the default graph
  tf$reset_default_graph()

  #the tensor here has an order of 4 - index 1 holds the image number, 2 - width, 3 - height and 4 - color
  #the value 3 ertains to the 3 color channels - r(red), g(green) and b(blue)
  images = tf$placeholder(tf$float32, shape(NULL, NULL, NULL, 3))

  #the images of varying size are scaled to the same size
  imgs_scaled = tf$image$resize_images(images, shape(224,224))

  #The VGG16 is a convolutional neural network model and the slim library is used to build the network
  #Defining the layers for VGG16 implementation
  # The last layer is the Tensor holding the logits of the classes
  lastlayer = slimmodule$conv2d(imgs_scaled, 64, shape(3,3), scope='vgg_16/conv1/conv1_1') %>%
    slimmodule$conv2d(64, shape(3,3), scope='vgg_16/conv1/conv1_2')  %>%
    slimmodule$max_pool2d( shape(2, 2), scope='vgg_16/pool1')  %>%

    slimmodule$conv2d(128, shape(3,3), scope='vgg_16/conv2/conv2_1')  %>%
    slimmodule$conv2d(128, shape(3,3), scope='vgg_16/conv2/conv2_2')  %>%
    slimmodule$max_pool2d( shape(2, 2), scope='vgg_16/pool2')  %>%

    slimmodule$conv2d(256, shape(3,3), scope='vgg_16/conv3/conv3_1')  %>%
    slimmodule$conv2d(256, shape(3,3), scope='vgg_16/conv3/conv3_2')  %>%
    slimmodule$conv2d(256, shape(3,3), scope='vgg_16/conv3/conv3_3')  %>%
    slimmodule$max_pool2d(shape(2, 2), scope='vgg_16/pool3')  %>%

    slimmodule$conv2d(512, shape(3,3), scope='vgg_16/conv4/conv4_1')  %>%
    slimmodule$conv2d(512, shape(3,3), scope='vgg_16/conv4/conv4_2')  %>%
    slimmodule$conv2d(512, shape(3,3), scope='vgg_16/conv4/conv4_3')  %>%
    slimmodule$max_pool2d(shape(2, 2), scope='vgg_16/pool4')  %>%
```

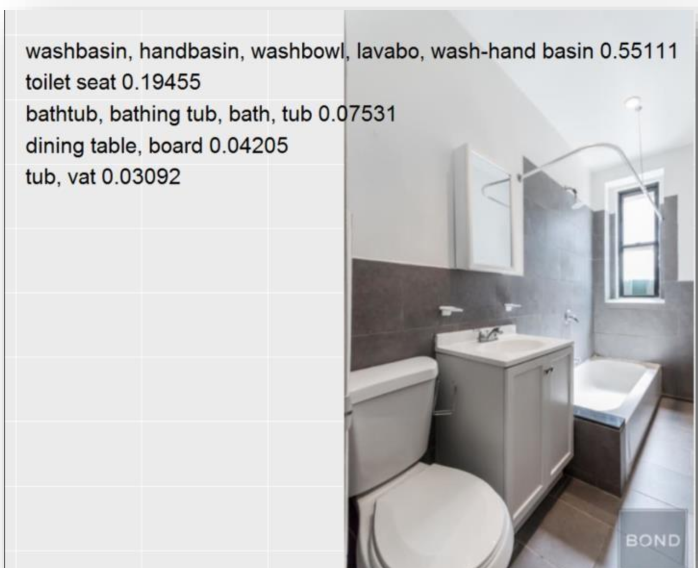
```
  slimmodule$conv2d(512, shape(3,3), scope='vgg_16/conv5/conv5_1')  %>%
  slimmodule$conv2d(512, shape(3,3), scope='vgg_16/conv5/conv5_2')  %>%
  slimmodule$conv2d(512, shape(3,3), scope='vgg_16/conv5/conv5_3')  %>%
  slimmodule$max_pool2d(shape(2, 2), scope='vgg_16/pool5')  %>%

  slimmodule$conv2d(4096, shape(7, 7), padding='VALID', scope='vgg_16/fc6')  %>%
  slimmodule$conv2d(4096, shape(1, 1), scope='vgg_16/fc7') %>%

  slimmodule$conv2d(1000, shape(1, 1), scope='vgg_16/fc8')  %>%
  tf$squeeze(shape(1, 2), name='vgg_16/fc8/squeezed')


#starting a session and extracting the model weights from the vgg_16.ckpt file
restore = tf$train$Saver()
new_session = tf$Session()
restore$restore(new_session, 'C:/Users/tjneh/BAPM/R/vgg_16.ckpt')


#Note - the image is loaded from the url and not downloaded so that the latest/updated images are obtained
#load the image as url and convert into a jpeg format for processing
image_feed = as.character(image_link)
z <- tempfile()
download.file(image_feed,z,mode="wb")
new_image <- readJPEG(z)

#retrieving the dim attribute
image_dim = dim(new_image)

#ensuring the images are in the range of 0-255 (rgb color scale)
image_array = array(255*new_image, dim = c(1, image_dim[1], image_dim[2], image_dim[3]))

#the prediction is performed by inputing image_array and loading the last_layer having the correct weights
lastlayer_values = new_session$run(lastlayer, dict(images = image_array))

#sorting the classes with the highest probabilities and loading the imagenet_classes.txt file which contains the desc
index = sort.int(lastlayer_values, index.return = TRUE, decreasing = TRUE)$ix[1:5]
probability = exp(lastlayer_values)/sum(exp(lastlayer_values))
classes = read_delim("C:/Users/tjneh/BAPM/R/imagenet_classes.txt", "\t", escape_double = FALSE, trim_ws = TRUE,col_na
```

```
#initialising the class_name variable to contain the name and probabilities of the classification
class_name = ""
for (i in index) {
  class_name = paste0(class_name, classes[i,][[1]], " ", round(probability[i],5), "\n")
}

#read the values in the class_name string and split it at a new line into different columns
x = read.table(text = class_name, sep = "\n", colClasses = "character")

#transposing the colums into rows and converting it into a vector
transpose_x = t(x)
vect = as.vector(transpose_x)
vect
length(vect)
cha <- NULL
num <- NULL
vals <- vector()
#getting the top 3 classes and probabilities
  for( i in 1:3)
{
#the value is in a string format and the data is split into a column containing class and a column containing probab
  cha[i] = gsub("[[:digit:]]","",vect[i])
  num[i] <- as.numeric(str_extract(vect[i], "[0-9]+.[0-9]+"))
  vals = c(vals,cha[i],num[i])
  t(vals)
}

dframe = as.data.frame(t(vals))

#creating a dataframe with the image link and its classifications
op_dframe = c(image_feed,dframe)
op_dframe = as.data.frame(op_dframe)

#adding column names
colnames(op_dframe) = c("url","class1","pobability1","class2","probability2","class3","probability3")

#output the dataframe
return(op_dframe)
```

At this point a ggplot can be obtained for an image imposed with the classes it was classified into along with the probabilities using the below code:

```
library(grid)
graph = rasterGrob(new_image, interpolate=TRUE)
class_name = ""
for (i in index) {
  text = paste0(class_name, classes[i,][[1]], " ", round(probs[i],5), "\n")
}

library(ggplot2)
ggplot(data.frame(d=1:3)) + annotation_custom(graph) + annotate('text',x=0.01,y=0.65,label=text,
size=8, hjust = 0, vjust=0, color='black') + xlim(0,1) + ylim(0,1)
```

Ggplot of an image that was classified



Step 4: Writing the classes and the probabilities of each image to an output file

```
#writing the classes and the probabilities of each image to an output file

#converting datatype of data to vector for processing
as.vector(data)

#creating a temporary dataframe variable
new_frame = data.frame(url = character(0),c1 = character(0),p1 = numeric(0),c2 = character(0),p2= numeric(0),c3 = chara
loopit <- function(data)
{
  for (i in 1:nrow(data))
  {
    #process for rows which have a url only
    if(data[i,1] != 'Image not found')
    {
      op_dframe = imageClassifier(image_link = data[i,1])

      new_frame = rbind.data.frame(new_frame,op_dframe)
    }
  }
  return(new_frame)
}

#extract the classification and write it to a csv file
classified_images = loopit(data)
write.csv(classified_images,"C:/Users/tjneh/BAPM/R/image_classification.csv")
```

A sample output of the above implementation in csv format is as below

	url	class1	proabaility1	class2	probability2	class3	probability:
1	https://photos.renthop.com/2/7170325_3bb5ac84a5a10227b17b273e79bd77b4.jpg	home theater, home theatre .	0.33222	dining table, board .	0.23319	turnstile .	0.04314
2	https://photos.renthop.com/2/7170325_a29a17a771ee6af213966699b05c8ea2.jpg	microwave, microwave oven .	0.9033	refrigerator, icebox .	0.03774	turnstile .	0.01275
3	https://photos.renthop.com/2/7170325_149a898e8760cac1cad56e30cfe98baa.jpg	dining table, board .	0.40334	bannister, banister, ba	0.34082	sliding door .	0.05365
4	https://photos.renthop.com/2/7170325_f74a43d781bcc3c5588e61dd47de81ba.jpg	microwave, microwave oven .	0.56385	file, file cabinet, filing	0.0945	refrigerator, icebox .	0.08312
5	https://photos.renthop.com/2/7170325_e677d9d249ac99abe01aa5454c6e9f59.jpg	dining table, board .	0.10699	file, file cabinet, filing	0.10499	desk .	0.10113
6	https://photos.renthop.com/2/7170325_960ea0e180bf2f15467b68b455db6172.jpg	sliding door .	0.35295	wardrobe, closet, pres	0.34358	medicine chest, medicine cabi	0.04338
7	https://photos.renthop.com/2/7170325_cbc1b8437155dbf7f5d63b3a0b5a45a3.jpg	dining table, board .	0.22399	sliding door .	0.20988	wardrobe, closet, press .	0.1454
8	https://photos.renthop.com/2/7170325_9a9f2adc2ce922e1d5394727efdf64bb.jpg	tub, vat .	0.30861	bathtub, bathing tub, l	0.29936	medicine chest, medicine cabi	0.22098
9	https://photos.renthop.com/2/7170325_aae2a39d536103eebb282775fab1c315.jpg	washbasin, handbasin, washbo	0.31853	medicine chest, medic	0.16018	bathtub, bathing tub, bath, tul	0.14988
10	https://photos.renthop.com/2/7170325_cd290d0051b9f08e3482195dcbf6b5a6.jpg	sliding door .	0.32214	home theater, home t	0.14415	dining table, board .	0.12544
11	https://photos.renthop.com/2/7170325_a2b599da7880eea1edd10c4b04250dc1.jpg	sliding door .	0.24371	wardrobe, closet, pres	0.15133	shoji .	0.08844
12	https://photos.renthop.com/2/7170325_6b83fa82d662bcb09733ac3a8a107113.jpg	wardrobe, closet, press .	0.34667	four-poster .	0.25374	sliding door .	0.18489
13	https://photos.renthop.com/2/7092344_7663c19af02c46104bc4c569f7162ae0.jpg	crib, cot .	0.17285	microwave, microwav	0.14091	bannister, banister, balustrade	0.07089
14	https://photos.renthop.com/2/7092344_8287349abe511d195a7b6129bf24af0e.jpg	medicine chest, medicine cabir	0.38236	refrigerator, icebox .	0.26866	microwave, microwave oven	0.22945
15	https://photos.renthop.com/2/7092344_e9e6a2b7aa95aa7564fe3318cadcf4e7.jpg	medicine chest, medicine cabir	0.50721	wardrobe, closet, pres	0.13267	sliding door .	0.08657
16	https://photos.renthop.com/2/7092344_d51ee4b92fd9246633f93afe6e86d8f0.jpg	medicine chest, medicine cabir	0.61491	washbasin, handbasin	0.16043	toilet seat .	0.09388
17	https://photos.renthop.com/2/7092344_f0573fa184ca130b1b6000f2fa90511c.jpg	wardrobe, closet, press .	0.48046	safe .	0.12975	microwave, microwave oven	0.07476
18	https://photos.renthop.com/2/7092344_b2a62f769a59a317b0a243000db46fd0.jpg	web site, website, internet site	0.79668	envelope .	0.07651	binder, ring-binder .	0.01359
19	https://photos.renthop.com/2/7158677_c897a134b77dc1c772a663874da69315.jpg	cash machine, cash dispenser,	0.93434	safe .	0.02351	photocopier .	0.00486
20	https://photos.renthop.com/2/7158677_cbf67ce22b270aeefe274de0e3767e5f.jpg	file, file cabinet, filing cabinet .	0.25431	wardrobe, closet, pres	0.19299	safe .	0.16134
21	https://photos.renthop.com/2/7158677_c10b443e36a92a8db1d3c52b955045f0.jpg	refrigerator, icebox .	0.76301	stove .	0.03567	file, file cabinet, filing cabinet	0.03456
22	https://photos.renthop.com/2/7158677_96705ca1b7fa414863eacf7d5050c544.jpg	file, file cabinet, filing cabinet .	0.23485	safe .	0.13186	bookcase .	0.08798
23	https://photos.renthop.com/2/7158677_b994d4d8ec48f9ec099511883eb5fe9c.jpg	safe .	0.46875	cash machine, cash di:	0.38855	dining table, board .	0.04683
24	https://photos.renthop.com/2/7158677_d577322b9e7cd1ed16a301917db9cd90.jpg	safe .	0.5618	wardrobe, closet, pres	0.17458	medicine chest, medicine cabi	0.07261
25	https://photos.renthop.com/2/7211212_1ed4542ec81621d70d1061aa833e669c.jpg	prison, prison house .	0.42596	sliding door .	0.15144	safe .	0.11175
26	https://photos.renthop.com/2/7211212_7dfc41dced69245065df83d08eed4a00.jpg	sliding door .	0.34253	medicine chest, medic	0.11391	window shade .	0.07439
27	https://photos.renthop.com/2/7211212_c17853c4b869af6f53af08b0f5820b4c.jpg	wardrobe, closet, press .	0.68121	safe .	0.05478	medicine chest, medicine cabi	0.04379
28	https://photos.renthop.com/2/7211212_787ad8ea0c089792e7453e2121f8ac89.jpg	sliding door .	0.20405	medicine chest, medic	0.18452	safe .	0.14277

There could be several use cases for this classes and one use case identified is the classes of the images can be used to see if there is a correlation between the classes and the interest level (target variable). Based on the correlation if any the classes of images which generated maximum interest level can be uploaded on renthop to improve the sales.

5. Getting the Predictions

Based on the models that were run in the previous steps, we could use the hold out samples to fit the built models. This can be done using the h2o.predict() function. Below is the snap shot for the GBM model.

```
gbm2 <- h2o.gbm(x= varnames, y="interest_level", training_frame = train , validation_frame = valid,
                ntrees = 1000, learn_rate = 0.01, stopping_metric = "misclassification")

TestModel<- h2o.predict(gbm2, newdata = test )
summary(gbm2)
plot(gbm2)
summary(TestModel)

> summary(TestModel)
 predict       high                low                medium
 low   :2899   Min.   :0.0008973   Min.   :0.009031   Min.   :0.002209
 medium: 523   1st Qu.:0.0153385   1st Qu.:0.518768   1st Qu.:0.066650
 high  : 108   Median :0.0394070   Median :0.750422   Median :0.194660
               Mean   :0.0798549   Mean   :0.693275   Mean   :0.226870
               3rd Qu.:0.0894697   3rd Qu.:0.910950   3rd Qu.:0.354021
               Max.   :0.9636408   Max.   :0.996894   Max.   :0.873030
```

6. Shutting Down H2O

Once all the required data analysis is performed on H2O, it can be shutdown using the h2O.shutdown() function.

VI. Further exploration using H2O

Other than the models that we have built on our dataset, H2O can also be used to perform various other data analysis techniques like (these techniques haven't been implemented in the code).

- **Principal component analysis**

```
sample_pca = h2o.prcomp(data = h2o_dataframe, standardize = TRUE)
print(sample_pca)
summary(sample_pca)
```

- **K-means**

```
sample_k = h2o.kmeans(data = h2o_dataframe, centers = 4, cols = c("x1", "x2", "x3"))
print(sample_k)
```

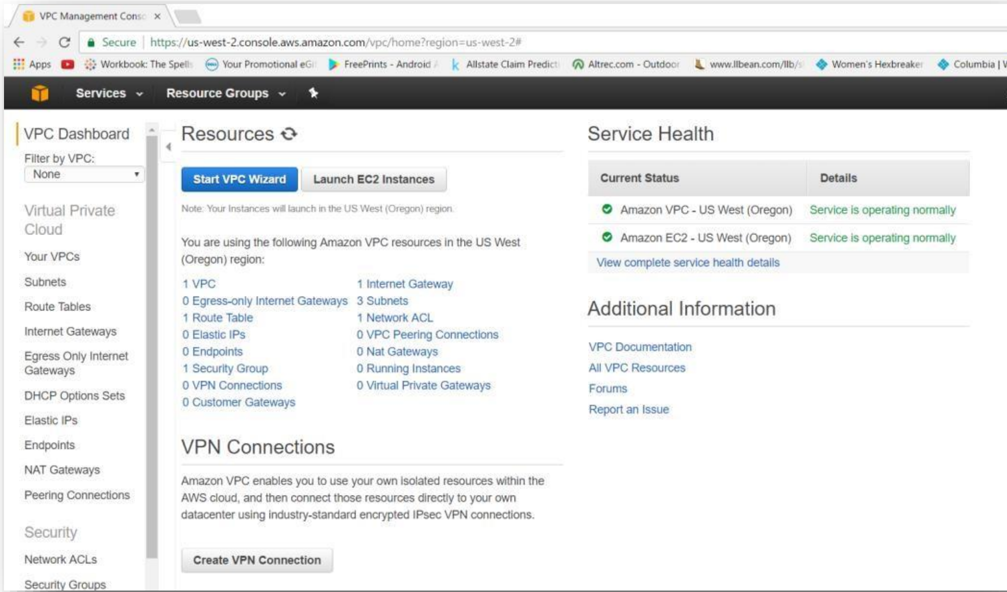
- **Generalized Linear Models**

```
h2o.glm(y = "y1", x = c("x1","x2","x3"), data =h2o_dataframe, family = "binomial",
nfolds = 4, alpha = 0.5)
```

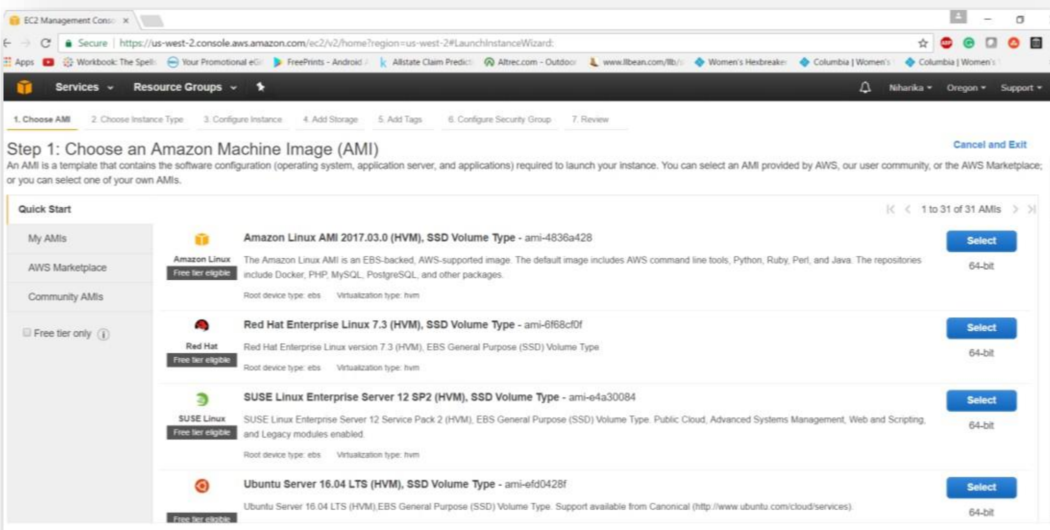
- **Running H2O on Rstudio server on AWS environment**

Also, other than exploring these techniques, one of the best ways of using H2O to its limits is by using it on RStudio Server on AWS. The basic tier allows the user memory upto 30GB without any extra charges. The implementation for the same is as follows:

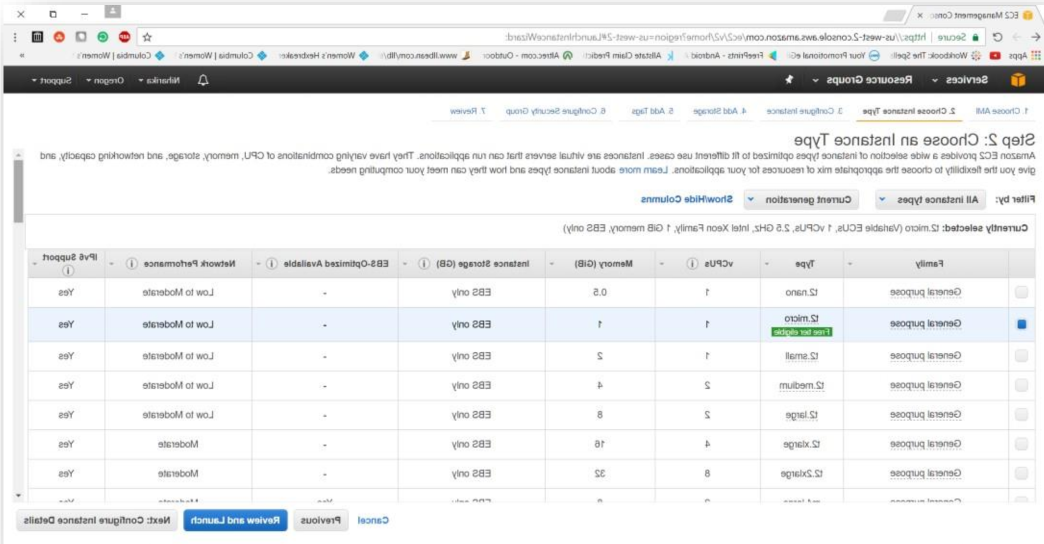
- Set-up a free account on AWS
- Open VPC console dashboard (http s:/ /con sole.aws.ama zon .com / vp c/)



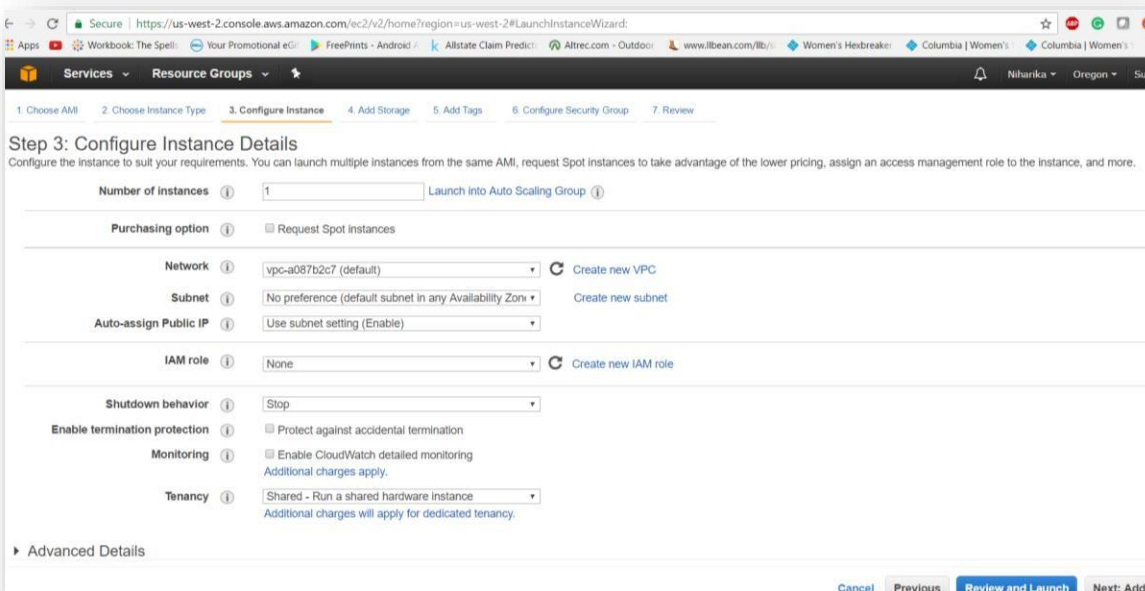
- Click on launch EC2 instance and select the Amazon Linux AMI machine



- Then choose the instance that qualifies for the free tier



- In the next tab, configure the instances as required

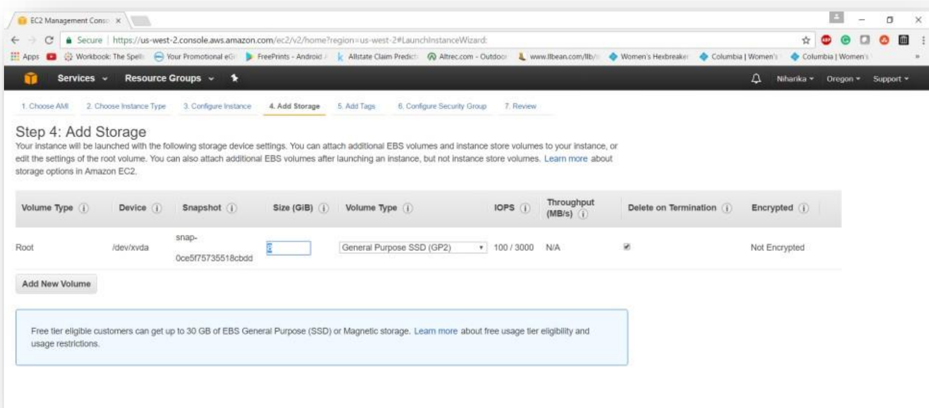


- And under advanced details, under text add the following to install RStudio server on the instance (Based on AWS blog -https://aws.amazon.com/blogs/big-data/running-r-on-aws/)

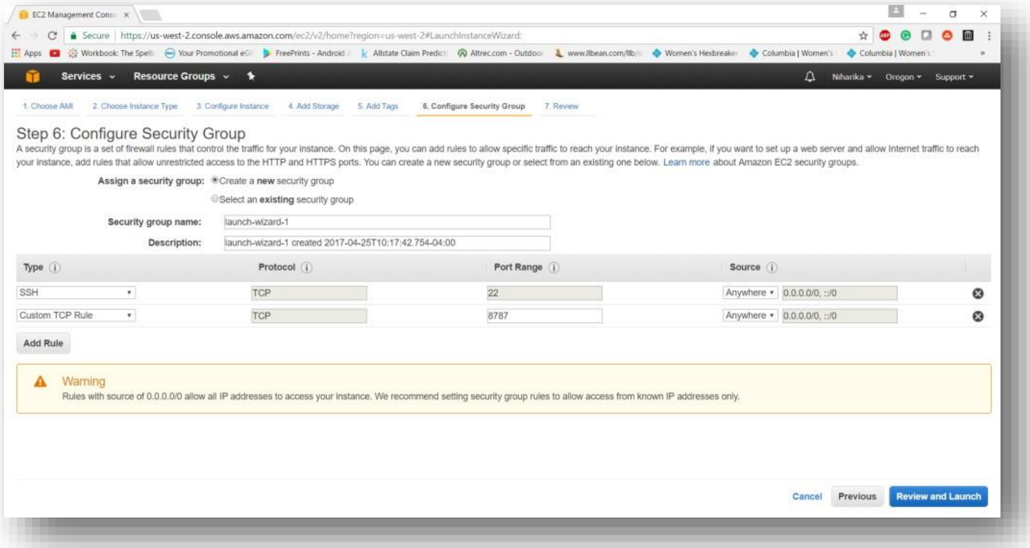
```
#!/bin/bash
# install R
yum install -y R
# install RStudio-Server
wget https://download2.rstudio.org/rstudio-server-rhel-1.0.143-x86_64.rpm
yum install -y --nogpgcheck rstudio-server-rhel-1.0.143-x86_64.rpm
yum install -y curl-devel
# add user
useradd niharika
echo niharika:testing | chpasswd
```

  Also before running submitting the above code check for the latest version of RStudio server on the following link http s:/ / www.rst u d io.com / p rod u ct s/ rst ud io/ down loa d - ser ver / under the RedHat/CentOS tab

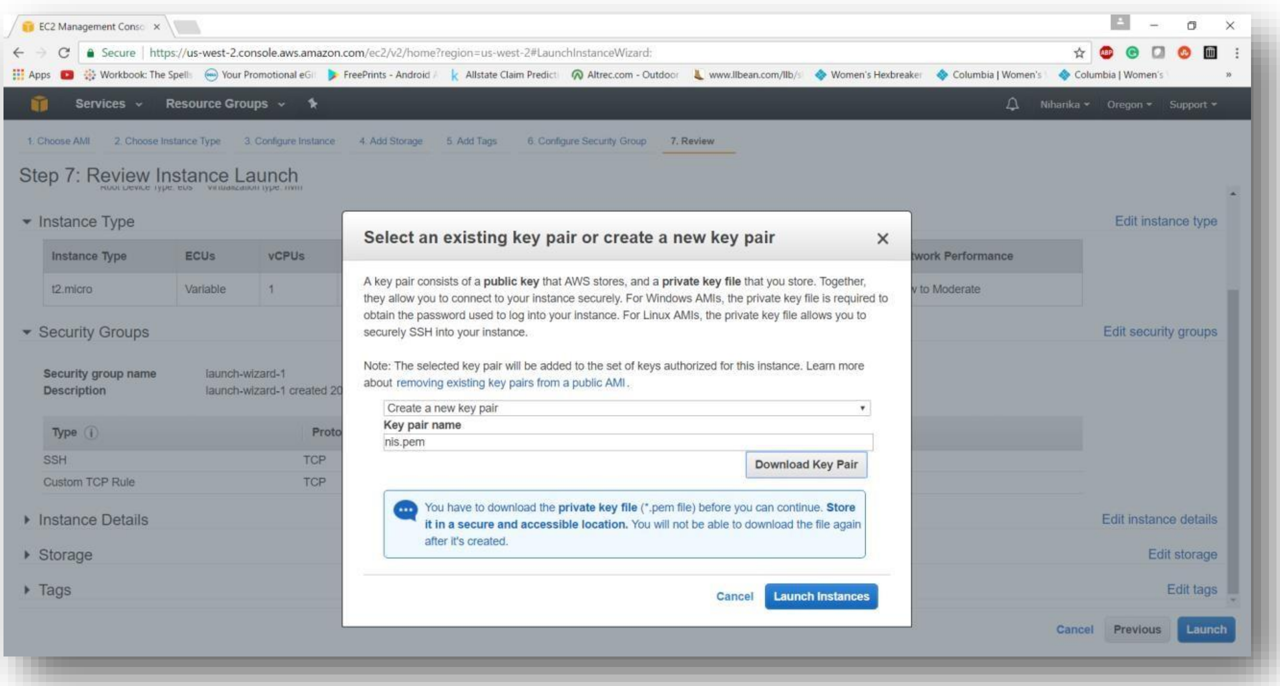
- Next add storage to the AWS environment, where we can have upto 30GB of storage for free



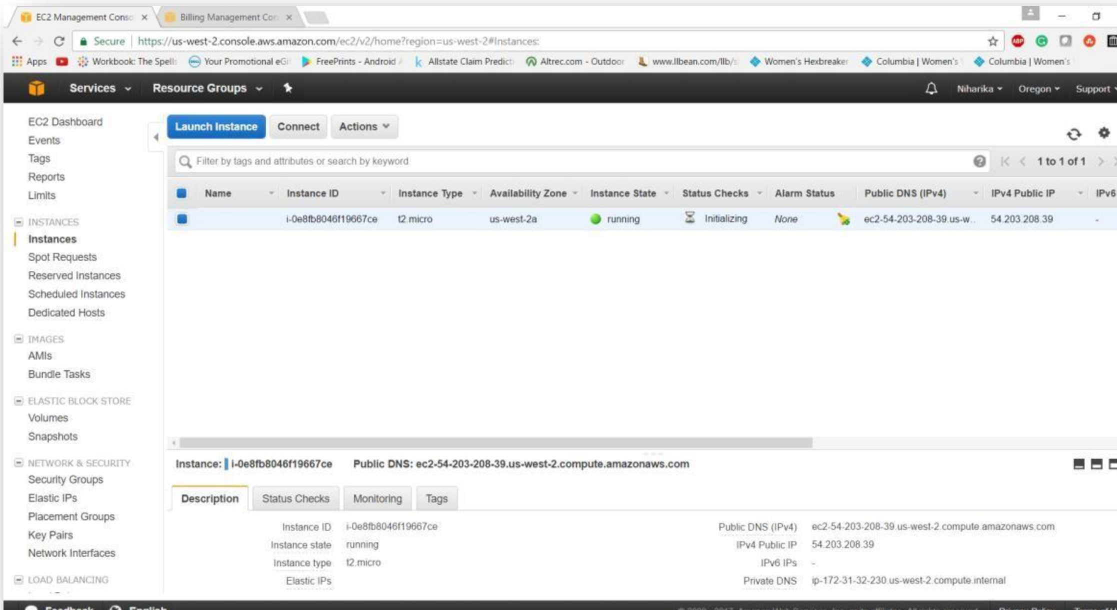
- Post this click on submit and configure the security groups. Be careful while setting the source as setting it as Anywhere would allow the instance to be publicly available (Set it to My IP). Also add another rule with port range set to 8787 (will be used for accessing RStudio Server)



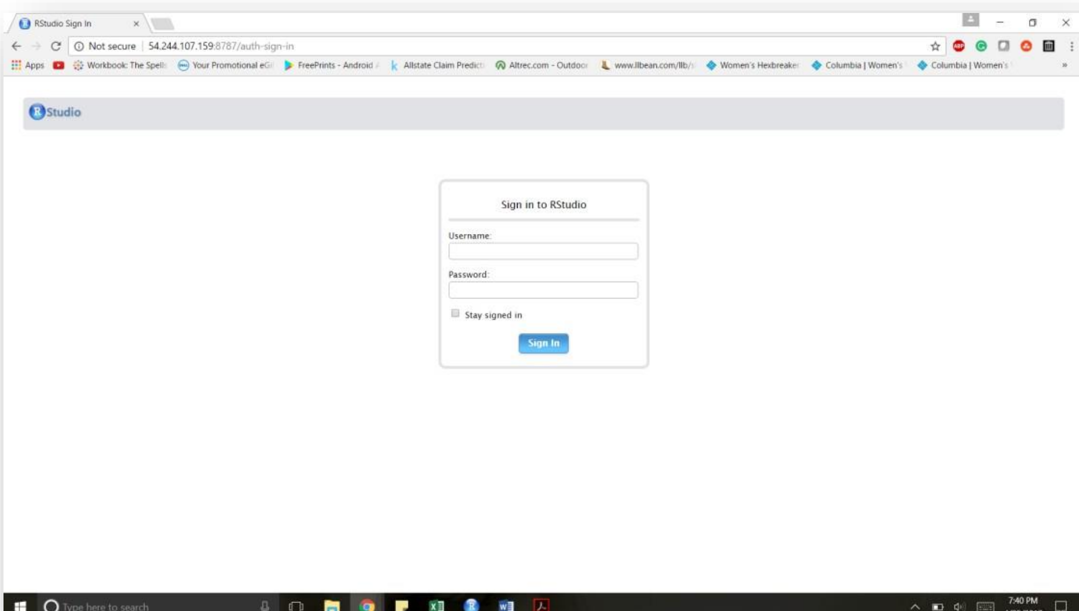
- Review the instance and click on launch which will prompt to select an existing key or create a new one. Incase it's the first-time usage create a new key and download it. Make sure to keep the key safe as a duplicate can't be generated for the same and it won't bepossible to access the instance without the key



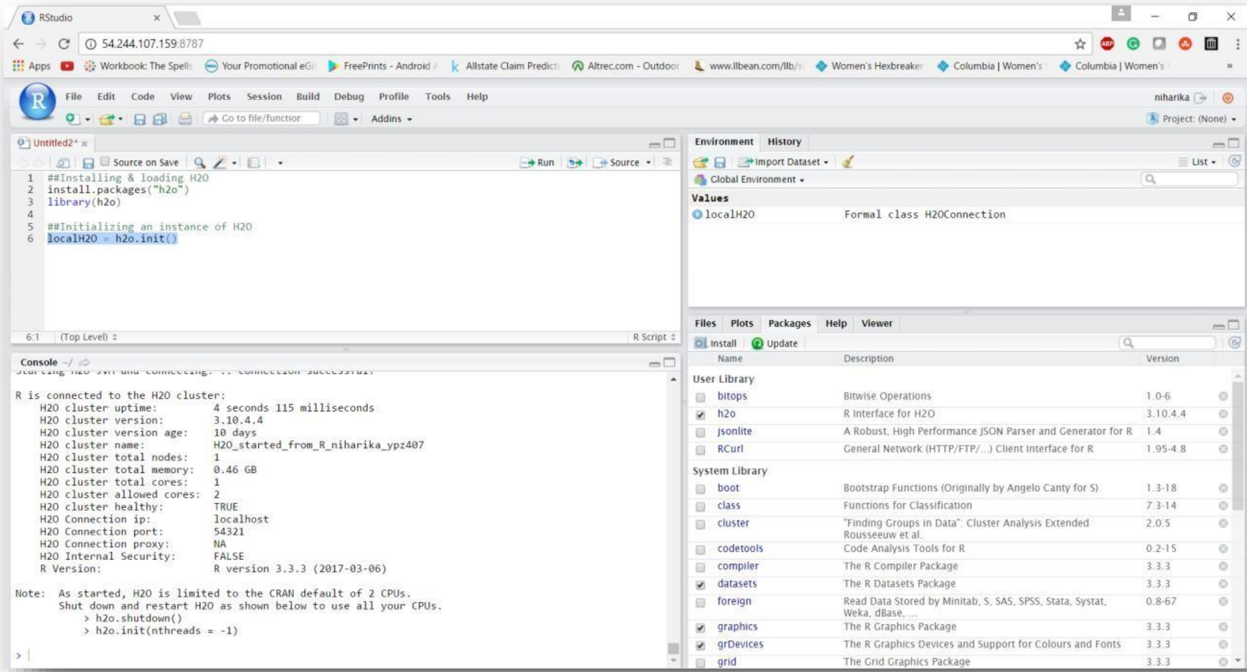
- Finally, the active instance can be viewed on the EC2 dashboard. (It might take a few minutes to activate)



- Finally start the RStudio server by entering the IPv4 Public IP (present on EC2 dashboard under the instance information) followed by :8787 and press enter. On the sign in window enter the username and password given along with the set-up details.



- Below is the screenshot of an H2O instance running on the RStudio on AWS server



REFERENCES

http: / / do cs.h2o .ai

http: / / do cs.h2o .ai/ h2 o/ latest-stable/ h2o -do cs/architect ure .htm l

https: / /en.wikipedia.o rg /wiki / Paging

https: // ww w.h2o .ai/ h2o /

https: // rstu dio . github.io/ tenso rflow/

https: // research. goo gleblog .co m/ 20 16 /03 / train -yo ur-o wn -im age -classifie r-with.htm l

https: // ww w.cs.toro nto .edu/ ~ fro ssard/ po st/ vg g1 6/

https: / /aws.am azo n.com / blo gs / big-data/ running-r-on-aws/)

http: / / do cs.h2o .ai/ h2 o/ latest-stable/ h2o -r/ h2o _package.pdf

https: // ww w.tenso rflo w.o rg / pro gramm ers_guide/ dims_t y pes