Installation and run example:

1. Download H2O-RE

2. In .zshrc file, put:
   export REGO_HOME=/home/xiaoydai/H2OREgo (your direction of H2OREgo)

3. Make sure R (RStudio)and rPython package are installed on the machine. (The DSE machine I used is good.)

4. Download H2O for Hadoop:
   wget http://download.h2o.ai/download/h2o-3.10.0.3-hdp2.4?id=61d4da62-bd4b-4c0e-aaa8-be8ef441b331 -O h2o-3.10.0.3-hdp2.4.zip (version may be updated later)

5. Then you can find a .tar.gz file for the R H2O package in:
   /home/xiaoydai/h2o-3.10.0.3-hdp2.4/R/h2o_3.10.0.3.tar.gz

   In R or RStudio, install the H2O package with corresponding version:

```
# The following two commands remove any previously installed H2O packages for R.
if ("package:h2o" %in% search()) { detach("package:h2o", unload=TRUE) }
if ("h2o" %in% rownames(installed.packages())) { remove.packages("h2o") }

# Next, we download packages that H2O depends on.
if (! ("methods" %in% rownames(installed.packages()))) { install.packages("methods") }
if (! ("statmod" %in% rownames(installed.packages()))) { install.packages("statmod") }
if (! ("stats" %in% rownames(installed.packages()))) { install.packages("stats") }
if (! ("graphics" %in% rownames(installed.packages()))) { install.packages("graphics") }
if (! ("RCurl" %in% rownames(installed.packages()))) { install.packages("RCurl") }
if (! ("rjson" %in% rownames(installed.packages()))) { install.packages("rjson") }
if (! ("tools" %in% rownames(installed.packages()))) { install.packages("tools") }
if (! ("utils" %in% rownames(installed.packages()))) { install.packages("utils") }

# Now we download, install and initialize the H2O package for R.
install.packages('/home/xiaoydai/h2o-3.10.0.3-hdp2.4/R/h2o_3.10.0.3.tar.gz', repos = NULL, type="source")
```

6. Launch H2O:
   3p hadoop jar /home/xiaoydai/h2o-3.10.0.3-hdp2.4/h2odriver.jar -nodes 8 -mapperXmx 20g -output /home/xiaoydai/new1/

   here we claim 8 nodes with each has 20g memory.
   In the console, you will see the corresponding ip and port on the bottom.

7. SSH again to the DSE machine, in /home/xiaoydai/H2OREgo/examples/, there are 2 examples when the train and test data are local csv files, you could check their configuration files and run it through the command line stored in the readme.txt file.

   Also in /home/xiaoydai/H2OREgo/examples/hdfs/, there is another example showing how to run Rule Ensemble taking train and test data directly from HDFS, which I think is what you guys are most interested in:

   First follow Stathis' note to convert HIVE table to HDFS:

   sps 100 --packages com.databricks:spark-csv_2.10:1.4.0
   val df=sqlContext.sql("""select… from ….""")
   df.repartition(64).write.mode("overwrite").format("com.databricks.spark.csv").option("header","true").option("codec", "org.apache.hadoop.io.compress.GzipCodec").option("nullValue", "NA").save("data.csv")

   Then you can run model through the command line in readme.txt file:
   $REGO_HOME/bin/trainModel.sh --d=/home/xiaoydai/H2OREgo/examples/hdfs/data.conf --m=/home/xiaoydai/H2OREgo/examples/hdfs/model.conf

   Also you should check and modify the four files there when you run model on your own data:
   col2stkip.txt, colTypes.txt, data.conf, model.conf

   Most of the configuration parameters are self-explaining when you look into them, and most of them are consistent with prof. Friedman's code, which you can refer to the documentation at:
   https://github.com/intuit/rego/tree/master/doc

   I will highlight a few important parts about the configuration files below.

8. You can also test the model performance on your test data:
   $REGO_HOME/bin/runModel.sh --m=/home/xiaoydai/H2OREgo/examples/hdfs/output/export/ --d=/home/xiaoydai/H2OREgo/examples/hdfs/test.conf

   then the model evaluation metrics will pop out on the console.

About data.conf and test.conf:

1. hdfs.sever should remain unchanged:
   you only need to change hdfs.fname to your own file direction

2. Every time you start a new H2O Hadoop session, make sure the ip and port are consistent with those in step 6 above.

About model.conf:

1. num_trees and max_depth parsed into H2O.GBM are determind by tree.size and model.max.rules as following:

   rules_per_tree = 2 * (tree.size - 1)
   num_trees = int(model.max.rules / rules_per_tree) + 1
   max_depth = int(log_2(tree.size)) + 1

2. model.max.rules is the key factor to affect the speed.

3. model.max.terms are not implemented yet. Will talk this later.

Comments:

1. Missing values are handled by H2O's default setting, you can refer to: http://h2o-release.s3.amazonaws.com/h2o/rel-slater/5/docs-website/h2o-docs/index.html#Data%20Science%20Algorithms-DRF-FAQ
   and search "missing"
   e.g. Missing values affect tree split points. NAs always "go left", Missing values are automatically imputed by the column mean, etc.

2. For the lambda selection (i.e. regularization penalty weight), you can specify lambda or it will be handled by H2O's default.
   You can refer to 5.5.2 Lambda Search at:
   https://h2o-release.s3.amazonaws.com/h2o/rel-slater/9/docs-website/h2o-docs/booklets/GLM_Vignette.pdf
   But some of these parameters seem not working to me….

3. If you can't read your data from HDFS correctly, increase your partition when you convert HIVE table to HDFS.

4. Winsorizing linear features is slow, so I suppress this step for now. I will talke with Giovanni to see if we can figure this out.

5. Make sure you have access to /tmp on the DSE machine or whichever machine you put your H2OREgo on, because the python function POJO_extractor will put a temporary json file there.

6. H2O.saveModel behaves odd on the H2O Hadoop cluster, that I can only specify its path as /tmp/…, but I don't where the model is being exported. It's not the /tmp file in any of the DSE machine, getway machine or the hdfs cluster. But it is there and I can load the model again from there. All other things like H2O.download_POJO and the POJO_extractor work fine.