H2O AI

**Under the guidance of Professor Srikanth Krishnamurthy**

# What is H2O?



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.

H2O’s core code is written in Java. Inside H2O, a Distributed Key/Value store is used to access and reference data, models, objects, etc., across all nodes and machines. The algorithms are implemented on top of H2O’s distributed Map/Reduce framework and utilize the Java Fork/Join framework for multi-threading. The data is read in parallel and is distributed across the cluster and stored in memory in a columnar format in a compressed way. H2O’s data parser has built-in intelligence to guess the schema of the incoming dataset and supports data ingest from multiple sources in various formats.

H2O’s REST API 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).

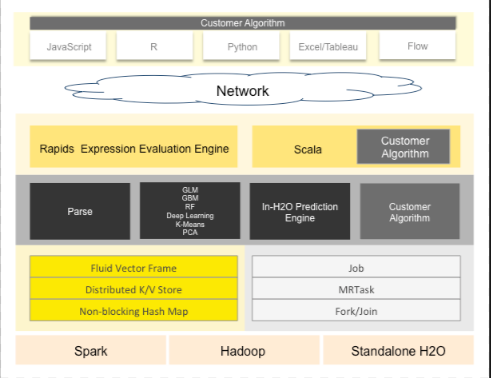
The speed, quality, ease-of-use, and model-deployment for the various cutting edge Supervised and Unsupervised algorithms like Deep Learning, Tree Ensembles, and GLRM make H2O a highly sought after API for big data data science.

# H2O Architecture

H2O does in-memory analytics on clusters with distributed parallelized state-of-the-art Machine Learning algorithms. H2O is based on a number of layers, and is coded to at different layers to best approach different tasks and objectives.

The diagram below shows most of the different components that work together to form the H2O software stack. The diagram is split into a top and bottom section, with the network cloud dividing the two sections. The top section shows some of the different REST API clients that exist for H2O.

The bottom section shows different components that run within an H2O JVM process.



REST API Clients

All REST API clients communicate with H2O over a socket connection.

**JavaScript**

The embedded H2O Web UI is written in JavaScript, and uses the standard REST API.

**R**

R scripts can use the H2O R package [‘library(h2o)’]. Users can write their own R functions that run on H2O with ‘apply’ or ‘ddply’.

**Python**

Python scripts currently must use the REST API directly. An H2O client API for python is planned.

**Excel**

An H2O worksheet for Microsoft Excel is available. It allows you to import big datasets into H2O and run algorithms like GLM directly from Excel.

**Tableau**

Users can pull results from H2O for visualization in Tableau.

**Flow**

H2O Flow is the notebook style Web UI for H2O.

JVM Components

An H2O cloud consists of one or more nodes. Each node is a single JVM process. Each JVM process is split into three layers: language, algorithms, and core infrastructure.

The language layer consists of an expression evaluation engine for R and the Shalala Scala layer. The R evaluation layer is a slave to the R REST client front-end. The Scala layer, however, is a first-class citizen in which you can write native programs and algorithms that use H2O.

The algorithms layer contains those algorithms automatically provided with H2O. These are the parse algorithm used for importing datasets, the math and machine learning algorithms like GLM, and the prediction and scoring engine for model evaluation.

The bottom (core) layer handles resource management. Memory and CPU are managed at this level.

How R (and Python) Interacts with H2O

The H2O package for R allows R users to control an H2O cluster from an R script. The R script is a REST API client of the H2O cluster. The data never flows through R.

Memory Management

**Fluid Vector Frame**

A Frame is an H2O Data Frame, the basic unit of data storage exposed to users. “Fluid Vector” is an internal engineering term that caught on. It refers to the ability to add and update and remove columns in a frame “fluidly” (as opposed to the frame being rigid and immutable). The Frame->Vector->Chunk->Element taxonomy that stores data in memory is described in Javadoc. The Fluid Vector (or fvec) code is the column-compressed store implementation.

**Distributed K/V store**

Atomic and distributed in-memory storage spread across the cluster.

**Non-blocking Hash Map**

Used in the K/V store implementation.

CPU Management

#### **Job**

Jobs are large pieces of work that have progress bars and can be monitored in the Web UI. Model creation is an example of a job.

**MRTask**

MRTask stands for MapReduce Task. This is an H2O in-memory MapReduce Task, not to be confused with a Hadoop MapReduce task.

**Fork/Join**

A modified JSR166y lightweight task execution framework.

# Components of H2O

**H2O Cluster**

* Multi-node cluster with shared memory model
* All computations in memory
* Each node sees only some rows of the data
* No limit on cluster size.

**Distributed Key Value store**

* Objects in the cluster such as data frames, models and results are all referenced by Key
* Any node in the cluster can access any object in the cluster by key.

**H2O Frame**

* Distributed Data Frames (Collection of Vectors)
* Columns are distributed (across nodes) arrays.
* Each node must be able to see entire dataset (achieved using HDFS, S3 or multiple copies of the data if it is a CSV file)

# H2O Flow UI

H2O Flow is an open-source user interface for H2O. It is a web-based interactive environment that allows you to combine code execution, text, mathematics, plots, and rich media in a single document.

With H2O Flow, we can capture, rerun, annotate, present, and share your workflow. H2O Flow allows us to use H2O interactively to import files, build models, and iteratively improve them. Based on our models, we can make predictions and add rich text to create vignettes of our work - all within Flow’s browser-based environment.

H2O Flow sends commands to H2O as a sequence of executable cells. The cells can be modified, rearranged, or saved to a library. Each cell contains an input field that allows we to enter commands, define functions, call other functions, and access other cells or objects on the page. When we execute the cell, the output is a graphical object, which can be inspected to view additional details.

While H2O Flow supports REST API, R scripts, and CoffeeScript, no programming experience is required to run H2O Flow.

**Download Flow**

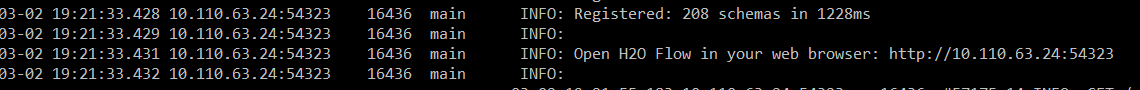
1. [Download H2O](http://www.h2o.ai/download/). This will download a zip file in the Downloads folder that contains everything we need to get started.

Link: <http://h2o-release.s3.amazonaws.com/h2o/rel-turing/10/index.html>

1. Next, in terminal, enter the following command lines one at a time:

cd ~/Downloads  
unzip h2o-3.10.0.10.zip  
cd h2o-3.10.0.10  
java -jar h2o.jar

1. Finally, to start Flow Point your browser to [http://localhost:54321](http://localhost:54321/)

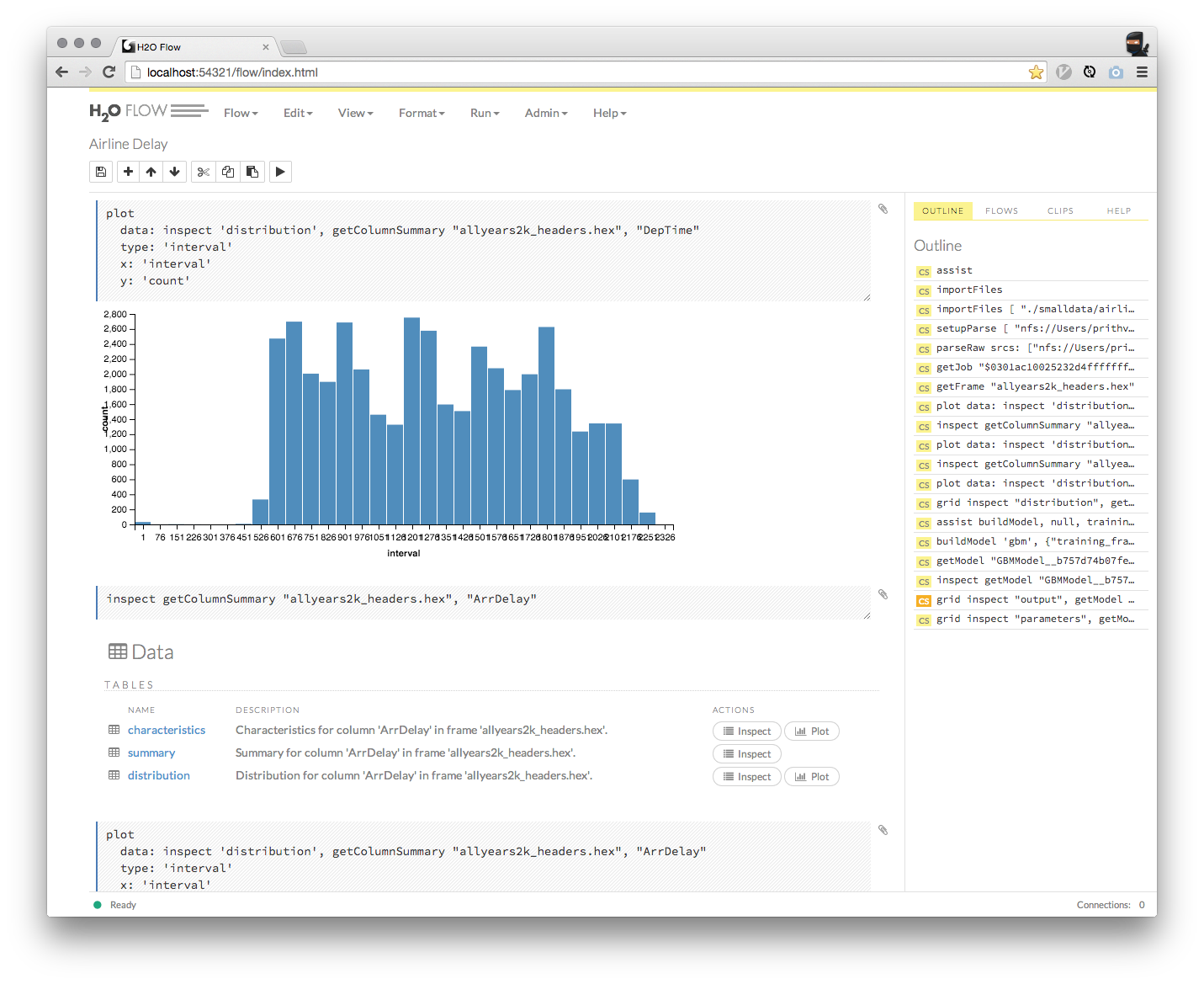
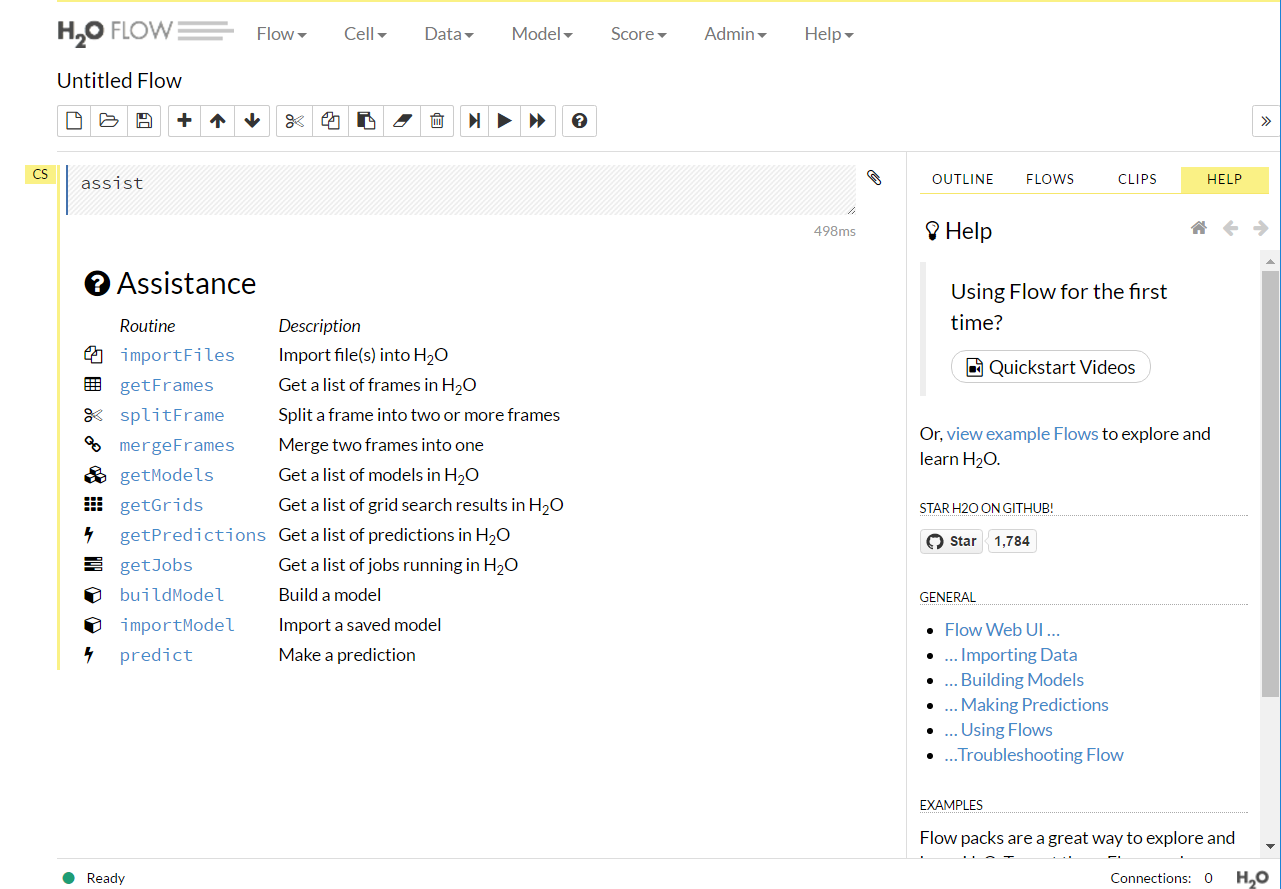


Launch Flow

The next time when we want to launch Flow, change into the directory that contains H2O package and run the JAR file from the command line.

cd ~/Downloads/h2o-{version}

java -jar h2o.jar



# H2O on R and Python

## Installation on R:

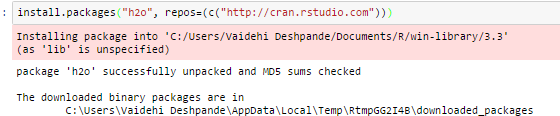
* Requirements: The only requirement to run “h2o” package in R is R>=3.1.0 and Java 7 or later.
* Linux, OS X, Windows
* No Computation is ever performed in R. All Computations are performed (in highly optimized Java code) in the H2O Cluster and initiated by REST calls from R

Follow the below steps to use H2O directly from R:

Copy and paste these commands into R one line at a time.

# 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.  
pkgs <- c("methods","statmod","stats","graphics","RCurl","jsonlite","tools","utils")  
for (pkg in pkgs) {  
if (! (pkg %in% rownames(installed.packages()))) { install.packages(pkg, repos = "http://cran.rstudio.com/") }  
}

We will get below message after H2O has been successfully downloaded:



# Now we download, install and initialize the H2O package for R.  
install.packages("h2o", type="source", repos=(c("http://h2o-release.s3.amazonaws.com/h2o/rel-turing/10/R")))  
library(h2o)  
localH2O = h2o.init(nthreads=-1)

## Installation in Python

* Requirements: Python 2 or 3. Java 7 or later.
* A few Python module dependencies
* Linux, OS X, Windows
* No Computation is ever performed in Python. All Computations are performed (in highly optimized Java code) in the H2O Cluster and initiated by REST calls from Python.

To use H2O directly from Python:

Install dependencies (use sudo if needed)-----

pip install h2o  
  
# This line is needed only if there are import errors when running h2o.  
pip install colorama requests tabulate future --upgrade  
  
# If you have Anaconda use:  
pip install tabulate

At the command line, copy and paste these commands one line at a time:

# The following command removes the H2O module for Python.  
pip uninstall h2o  
  
# Next, use pip to install this version of the H2O Python module.  
pip install http://h2o-release.s3.amazonaws.com/h2o/rel-turing/10/Python/h2o-3.10.0.10-py2.py3-none-any.whl

# Overview of Algorithms in H2O AI

**Supervised Learning:**  The following are the algorithms that’s fall under Supervised Algorithms Class.

* **Statistical Analysis**
  + **Generalized Linear Models with Regularization:** Binomial, Gaussian, Gamma, Poisson and Tweedie
  + **Naïve Bayes**

* **Ensembles**
  + **Distributed Random Forest:** Classification or regression models
  + **Gradient Boosting Machine:** Produces an ensemble of decision trees with increasing refined approximations
* **Deep Neural Networks** 
  + **Deep learning:** Create multi-layer feed forward neural networks starting with an input layer followed by multiple layers of nonlinear transformations

**Unsupervised Learning**

* **Clustering**
  + **K-means:** Partitions observations into k clusters/groups of the same spatial size
* **Dimensionality Reduction**
  + **Principal Component Analysis**: Linearly transforms correlated variables to independent components
  + **Generalized Low Rank Models\*:** extend the idea of PCA to handle arbitrary data consisting of numerical, Boolean, categorical, and missing data
* **Anomaly Detection**
  + **Autoencoders:** Find outliers using a nonlinear dimensionality reduction using deep learning

# Regression & Classification

Dataset: prostate.

This is cancer research dataset available when we install H2O package that depicts some factors and analyze how those factors influence the treatment to be given

Following are the columns:

|  |
| --- |
| CAPSULE |
| AGE |
| RACE |
| DPROS |
| DCAPS |
| PSA |
| VOL |
| GLEASON |

We are taking CAPSULE as our outcome variable and selected some of the variables which may predict the outcome variable.

## R

### Linear Regression

Command to load dataset using H2O:

prostate.hex = h2o.uploadFile(path = system.file("extdata", "prostate.csv", package="h2o"), destination\_frame = "prostate.hex")

**Arguments**

**Object**: An H2OClient object containing the IP address and port of the server running H2O.

Path: The complete URL or normalized file path of the file to be imported. Each row of data appears as one line of the file.

**Key/ destination\_frame**: (Optional) The unique hex key assigned to the imported file. If none is given, a key will automatically be generated based on the URL path.

**Parse**: (Optional) A logical value indicating whether the file should be parsed after import.

**Header**: (Optional) A logical value indicating whether the first line of the file contains column headers. If left empty, the parser will try to automatically detect this.

**header\_with\_hash:** (Optional) A logical value indicating whether the first line of the file contain a column header that begins with a hash character. If left empty, the parser will try to automatically detect this.

**Sep:** (Optional) The field separator character. Values on each line of the file are separated by this character. If sep = "", the parser will automatically detect the separator.

**col.names:** (Optional) A H2OParsedData object containing a single delimited line with the column names for the file.

**Silent**:(Optional) A logical value indicating whether or not to display an upload progress bar.

**parser\_type**: (Optional) Specify the type of data to be parsed. parser\_type = "AUTO" is the default, other acceptable values are "SVMLight", "XLS", and "CSV".

**Summary function:** Using summary function on each column we can find out min, max, mean, mode etc for each column in the dataset:

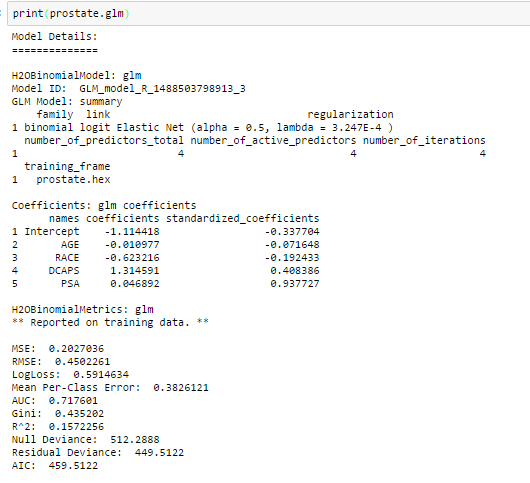
summary(prostate.hex)

**GLM function to determine Linear Regression:**

Here we are trying to determine appropriate capsule based on features like: age, race, psa, dcaps

prostate.glm = h2o.glm(x = c("AGE","RACE","PSA","DCAPS"), y = "CAPSULE", training\_frame = prostate.hex, family = "binomial", alpha = 0.5)

**Print(prostate.glm):**



Use following commands to plot the graph for linear regression:

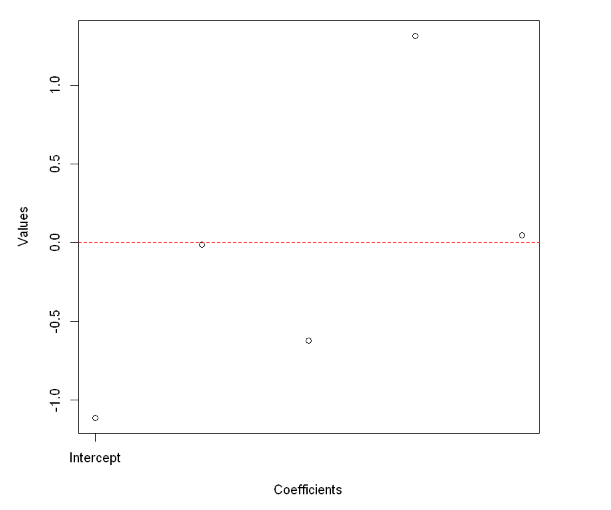
myLabels = c(prostate.glm@model$x, "Intercept")

plot(prostate.glm@model$coefficients, xaxt = "n", xlab = "Coefficients", ylab = "Values")

axis(1, at = 1:length(myLabels), labels = myLabels)

abline(h = 0, col = 2, lty = 2)

Below is the output graph:



Random Forest algorithm: It is a decision tree algorithm also known as ensemble algorithm.

H2O command to run Random Forest algorithm on our dataset:

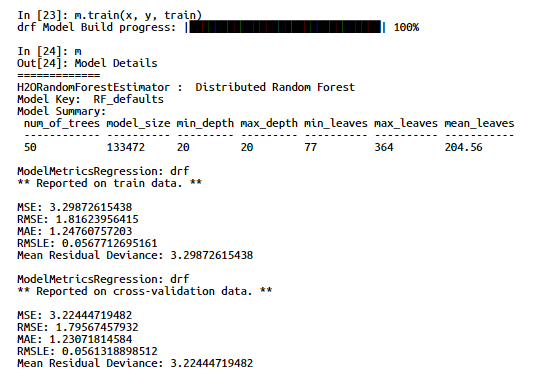
h2o.randomForest(x, y, train, nfolds = 10, model\_id = "RF\_defaults")

**Arguments**:

**Ntrees:** How many trees in your forest.

**max\_depth:** How deep a tree is allowed to grow. In other words, how complex each tree is allowed to be.

Following is the input of Random Forest algorithm:



## Python

We have performed Linear Regression and Deep Learning Algorithm.

Deep Learning is a new area of Machine Learning research, which has been introduced with the objective of moving Machine Learning closer to one of its original goals: Artificial Intelligence. It Comes under the Supervised algorithms category of H2O AI.

### Deep Learning:

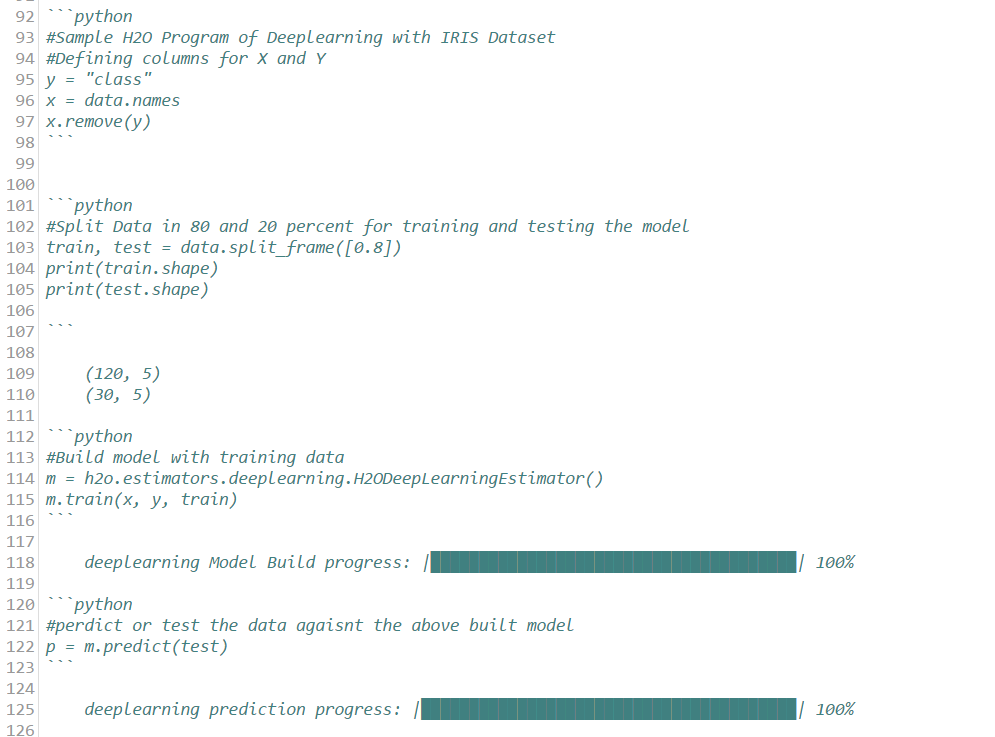
Dataset: IRIS

Import the h2o package and initialize it for Python.

Then, import the IRIS dataset.

#Import IRIS Data Set

data = h2o.import\_file(datasets + "iris\_wheader.csv")



### Random Forest

Dataset: Building Energy Efficiency

Following is some description:

X1: Relative Compactness

X2: Surface Area

X3: Wall Area

X4: Roof Area

X5: Overall Height

X6: Orientation

X7: Glazing area

X8: Glazing area distribution

Y1: Heating Load

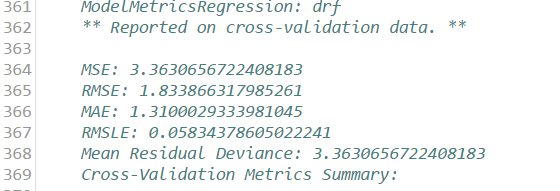
Y2: Cooling Load

First, import the dataset:

data = h2o.import\_file(datasets + "ENB2012\_data.csv")

Then, we create some Factorlist of X6 and X8 variables, split the dataset into train and test and perform random forest function.





### 

### Linear Regression

Dataset: Building Energy Efficiency

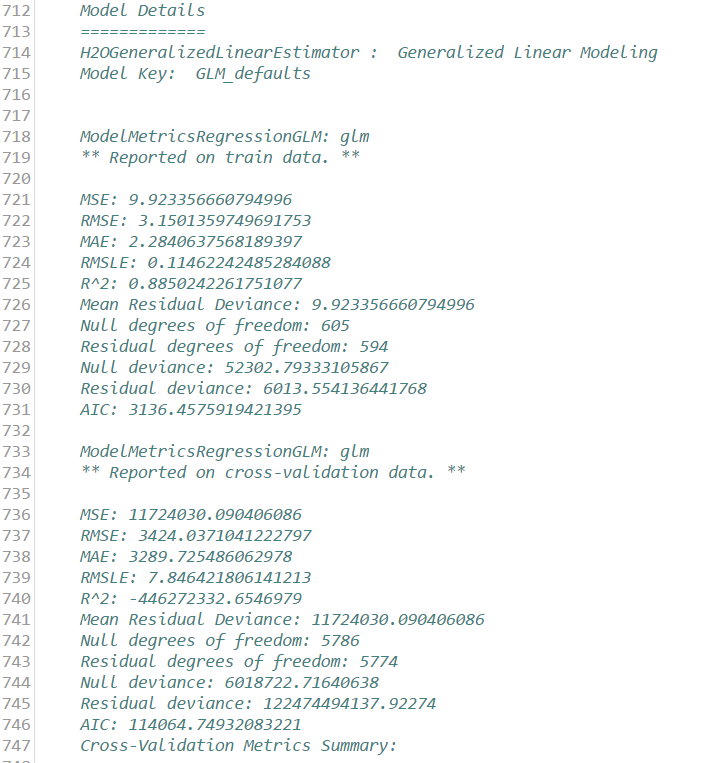
Import the dataset

*#Gradient Linear Model on Building Energy Efficency Dataset*

*data = h2o.import\_file(datasets + "ENB2012\_data.csv")*

We will split the dataset into train and test and peform the H2OGeneralizedLinearEstimator function.

**



# Clustering: K-means

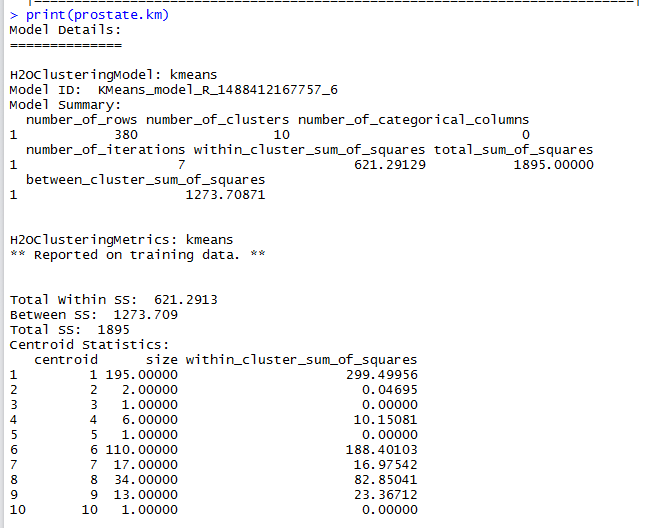
## R

The idea behind k-means is to divide your data up into k groups (you have to specify k) such that each data item is closer to the center of its cluster than to the center of any other cluster.

We will run this algorithm on our already imported prostate dataset.

prostate.km = h2o.kmeans(prostate.hex, k = 10, x = c("AGE","RACE","GLEASON","CAPSULE","DCAPS"))

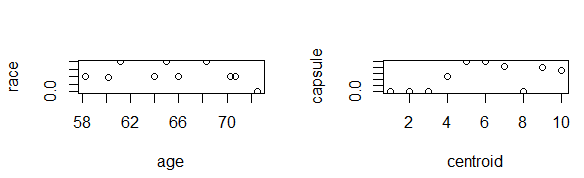
We get following output:



1. We can plot the k-means cluster plot using following commands:



And we get following plots:



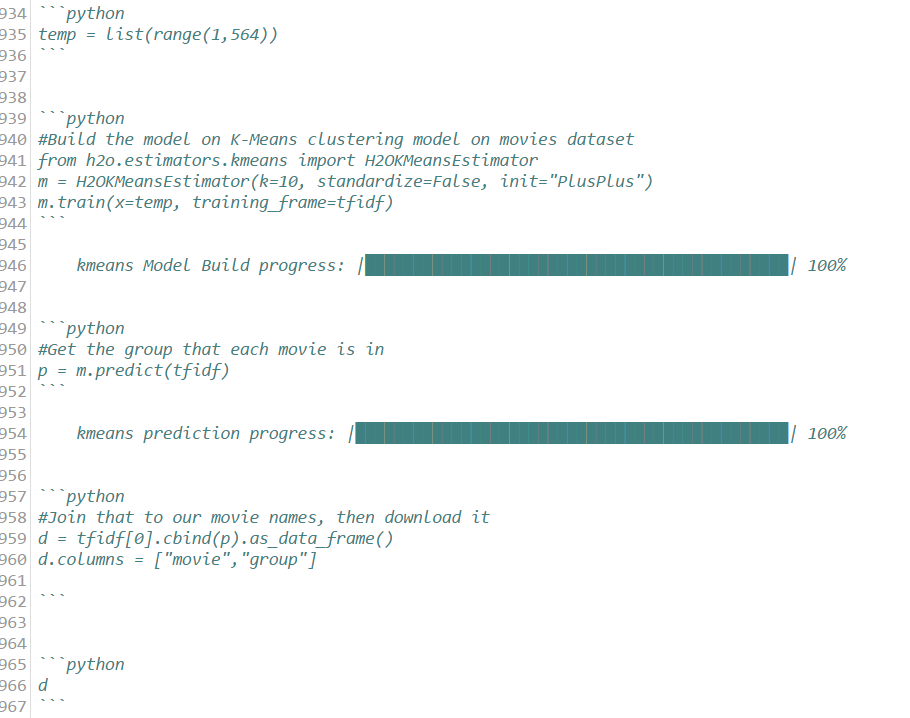
## Python

Dataset: MovieLens Dataset

Import the dataset

tfidf = h2o.import\_file(datasets+ "movie.tfidf.csv")

Then we will build k-means clustering model on movielens dataset. We will use *H2OKMeansEstimator*



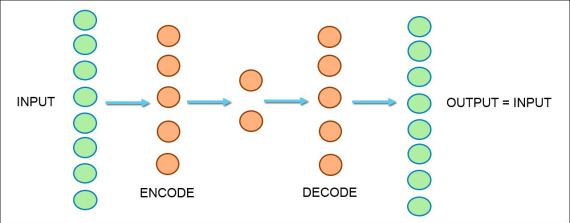
# Anomaly Detection

*Anomaly detection (or outlier detection) is the identification of items, events or observations which do not conform to an expected pattern or other items in a dataset*

H2o offers and easy to use, unsupervised and non-linear auto-encoder as part of its deep-learning model.

The idea is to use the input data to predict the input data by means of a “bottle-neck” network.

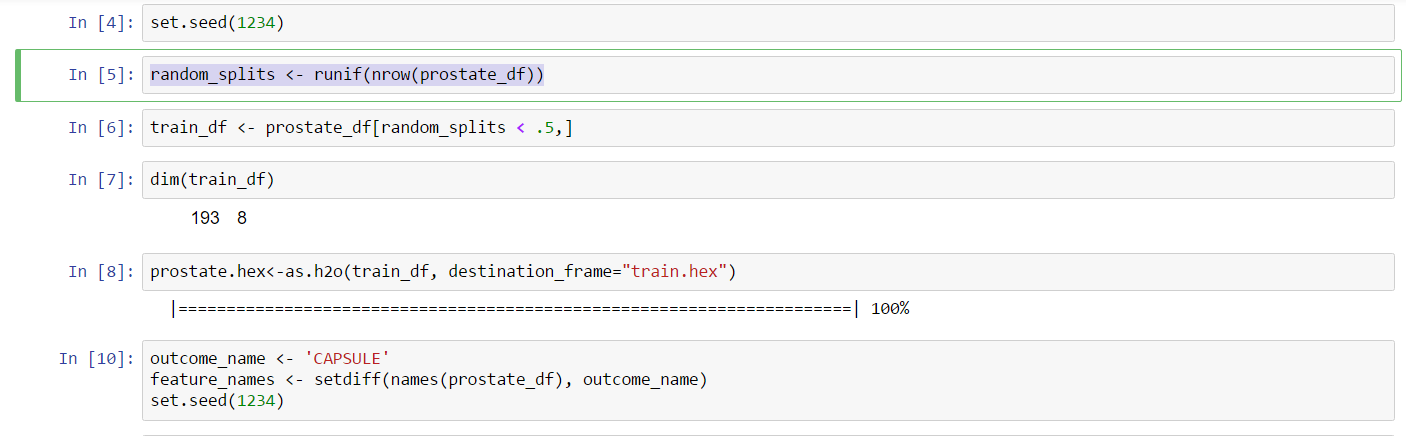
The middle layer can be regarded as a *compressed representation*of the input.



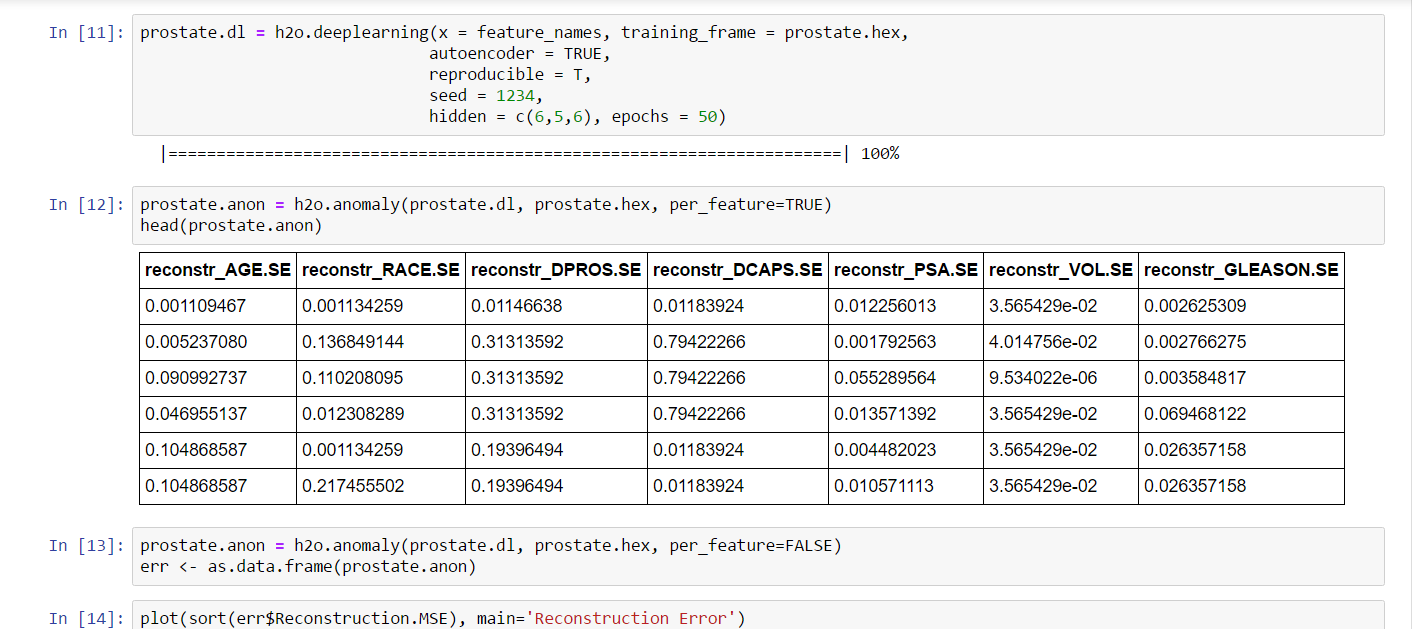
**Dataset; prostate**

So, after importing the dataset, and divide it to get training data.

our outcome variable is CAPSULE



Perform the autoencoder with deep learning function:



Arguments:

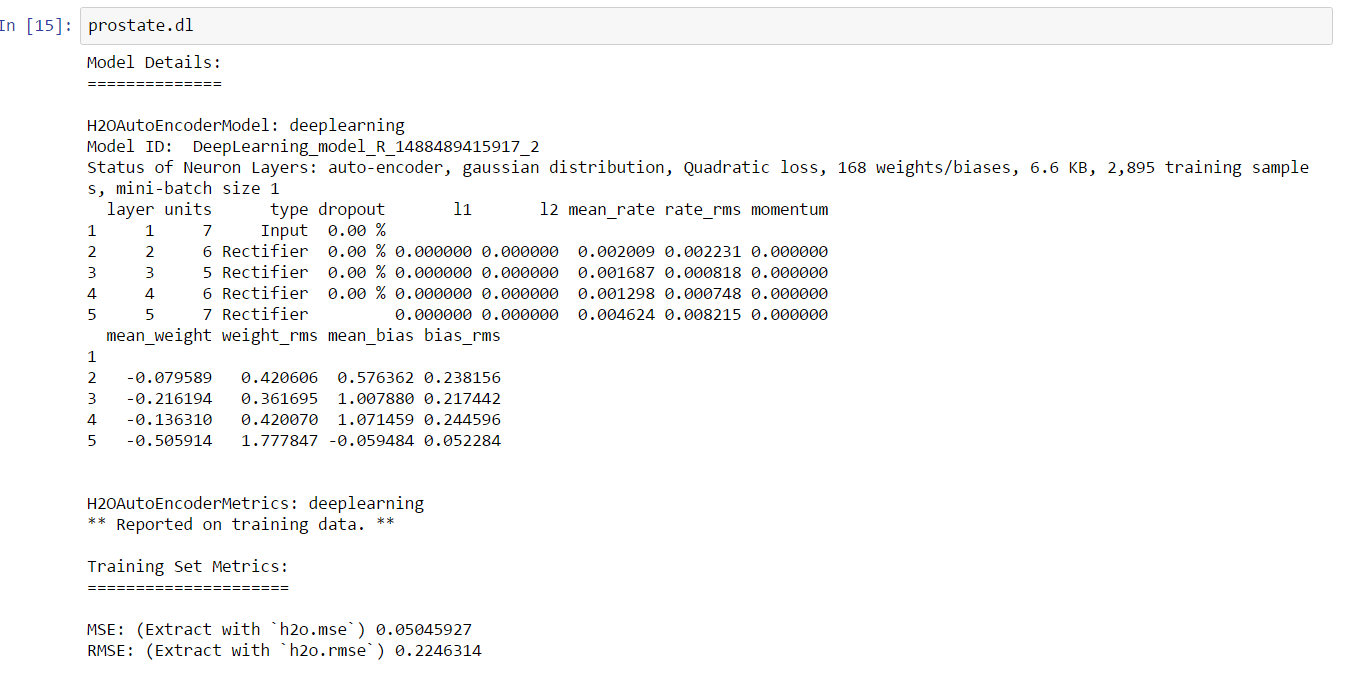
x: all the variables which are useful to predict the outcome variable

training\_frame: training dataset

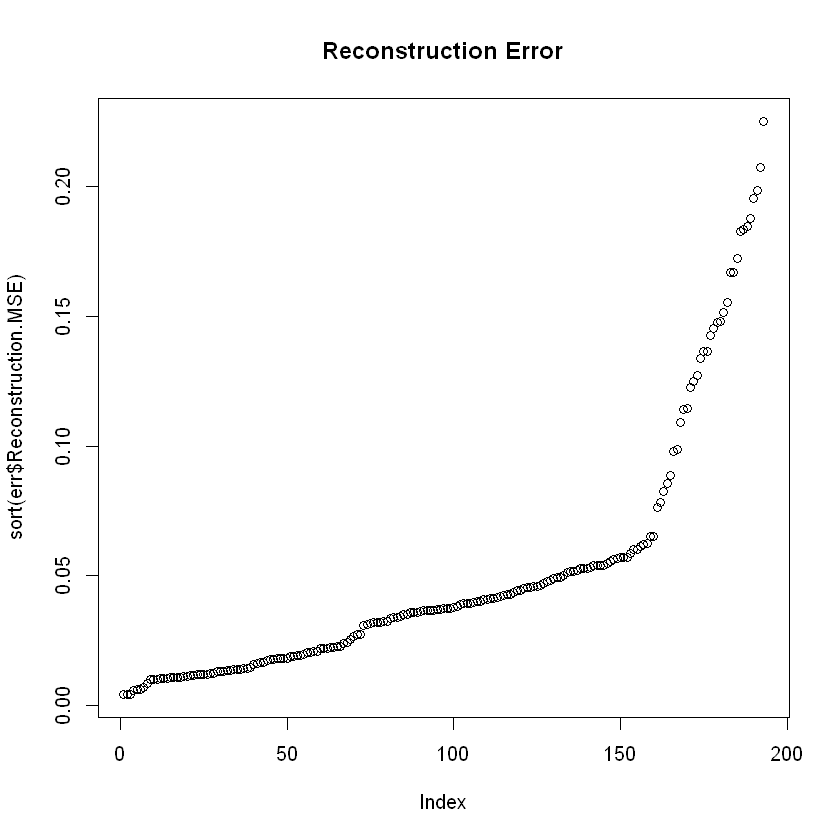
autoencoder: TRUE

reproducible = T

hidden = we are defining layers of how we are going to reduce our data for analysis.



We now call the h2o.anomaly function to reconstruct the original data set using the reduced set of features and calculate a means squared error between both. Here we set per\_feature parameter to FALSE in the h2o.anomaly function call as we want a reconstruction mean error based on observations, not individual features (but you should definitely play around feature-level scores as it could lead to important insights into your data).



# References

* <http://docs.h2o.ai/h2o/latest-stable/index.html>
* <http://proquest.safaribooksonline.com.ezproxy.neu.edu/book/programming/machine-learning/9781491964590/1dot-installation-and-quick-start/chapter_introduction_html>
* [http://amunategui.github.io/anomaly-detection-h2o/#autoencoder](http://amunategui.github.io/anomaly-detection-h2o/)
* <https://github.com/h2oai/h2o-meetups>
* <https://www.rdocumentation.org/packages/h2o/versions/3.10.3.3/topics/h2o.glm>
* <http://h2o-release.s3.amazonaws.com/h2o/rel-turing/10/index.html>
* <https://github.com/h2oai/h2o-tutorials/>