Machine learning in Finance

Interdisciplinary Project (IDP) - Final Report

Project Undertaken by:
Mohammad Zeeshan
M. Sc. Informatik,
Enrollment No. 03668029
Technical University of Munich

Academic Supervisor:
Patrick Bielstein
Lehrstuhl für Finanzmanagement und Kapitalmärkte
Fakultät für Wirtschaftswissenschaften
Technical University of Munich

Industry Supervisor:
Frank Rosner and Basil Komboz
Global Data & Analytics Team
Allianz SE, Munich

31st December 2017

Preface

I would like to thank Frank Rosner from Allianz Deutschland without whose support the project would not have been possible. I would also like to thank Dr. Gemma, for allowing me to work on this project. Dr. Gemma and Frank work as Data Scientists in the Global Data and Analytics Competence Center of Allianz SE in Munich, Germany.

I would also like to thank Patrick Bielstein for his patient support and collaboration with Chair for Financial and Capital Markets at Technical University of Munich.

Introduction

Today, machine learning algorithms are widely used in data analysis, classification and prediction problems. The Allianz Global Data and Analytics (GD&A) department drives the development of data science and big data related topics inside the company.

To help the GD&A team to keep up with evolving technology in big data and analytics, the project aims to compare state of the art algorithms in local and distributed environments.

Project Goals

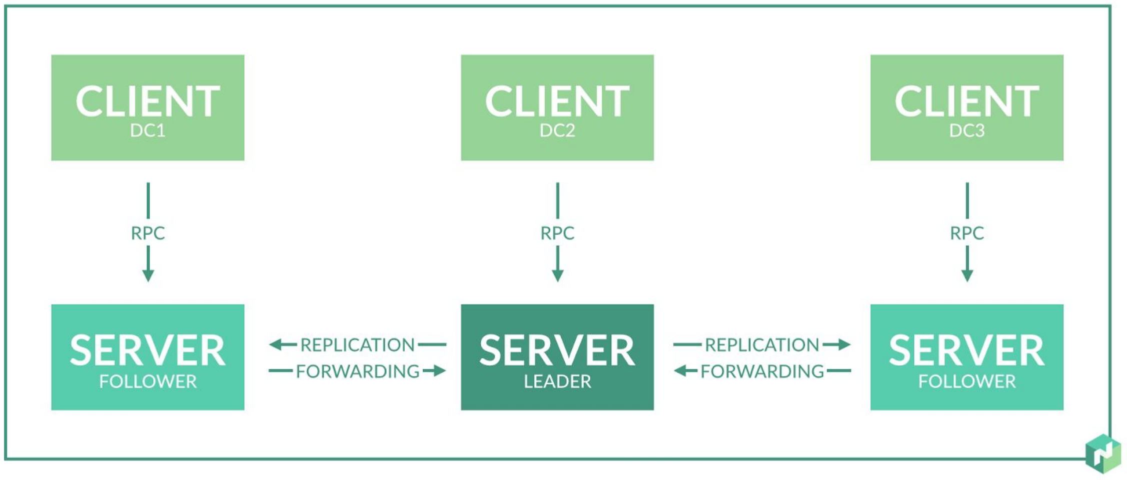
The goal of this interdisciplinary project is to compare the effects of horizontal and vertical scaling on resource consumption and model performance. This allows the selection of the right computational environment depending on the problem and data (size, characteristics, etc.). Additionally, we will obtain experimental results on how varying the computational resources of a system affects the performance of a machine learning algorithm. This gives us a benchmark for comparing different programming tools like H2O vs native R. The machine learning algorithm we use for such a comparison is GBM. We comment later on the choice of this algorithm and datasets used for benchmarking.

Description of Software Stack

We use R as a local execution environment for machine learning algorithms and H2O as a distributed one. The dataset used for evaluation contains several millions of rows and hundreds of columns. We will apply Gradient Boosted Machines (GBM) to solve a classification task and compare the results in both environments with different setups.

Nomad

What is Nomad? Briefly, Nomad is a distributed cluster manager and scheduler. It is designed for microservices and batch workloads. Nomad is Scalable, can scale to thousands of nodes and supports multi-datacenter/regions. The picture below shows the Nomad client-server architecture.



A Nomad job can be defined using a job template. A job template can be defined as follows:

Job Template:	Comments:
`job "rstudio" {`	#Define our simple Rstudio Job
`region = "us"`	
`datacenters = ["us-west-1", "us-east-1"]`	#Run only in US-West-1 data center
`type = "service"`	
`task {`	#Define the rimple RStudio task using Docker
`driver = "docker"`	
`config {`	
`image = "zeecitizen/rstudio"`	#Name of the Docker image to use
`}`	
`resources {`	#Allocating resources for Nomad job
`cpu    = 500 # MHz`	
`memory = 128 # MB`	
`}`	
`}`	
`}`	

```
job "rstudio" {
  region = "us"
  datacenters = ["us-west-1", "us-east-1"]
  type = "service"

  task {
    driver = "docker"
    config {
      image = "zeecitizen/rstudio"
    }
    resources {
      cpu    = 500 # MHz
      memory = 128 # MB
    }
  }
}
```

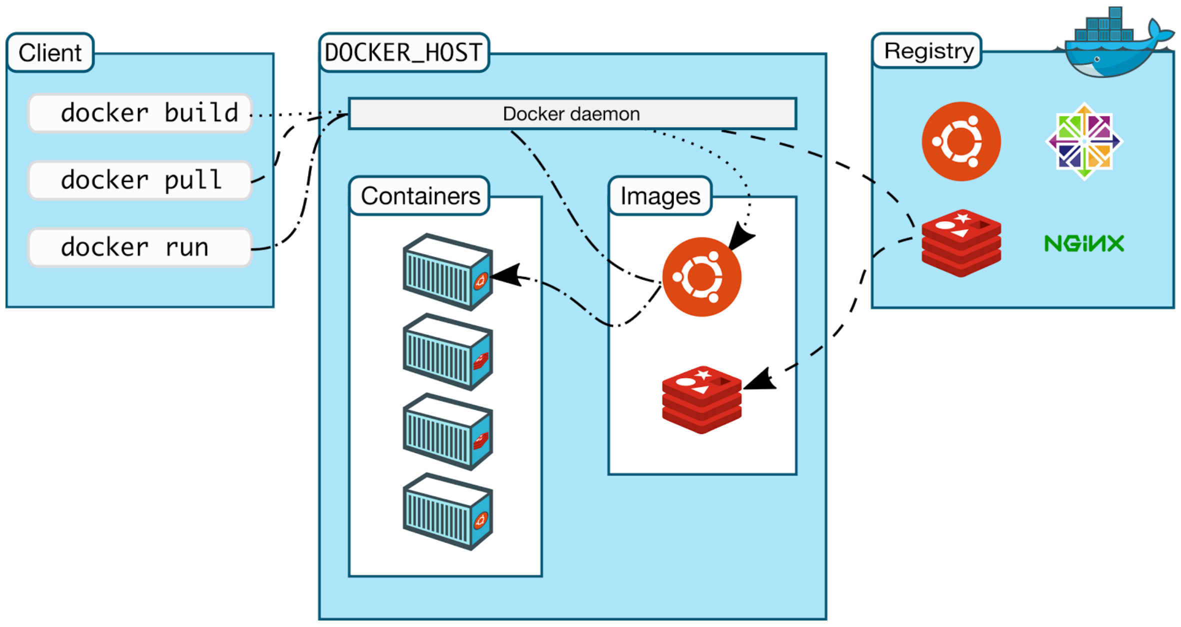
Below is the hierarchy of a Nomad Job.

Hierarchy of a Nomad Job	Comments:
● Job ○ Task Group ■ Task	The general hierarchy for a job is shown on left. Each job file has only a single job, however a job may have multiple groups, and each group may have multiple tasks. Groups contain a set of tasks that are co-located on a machine.

Docker:

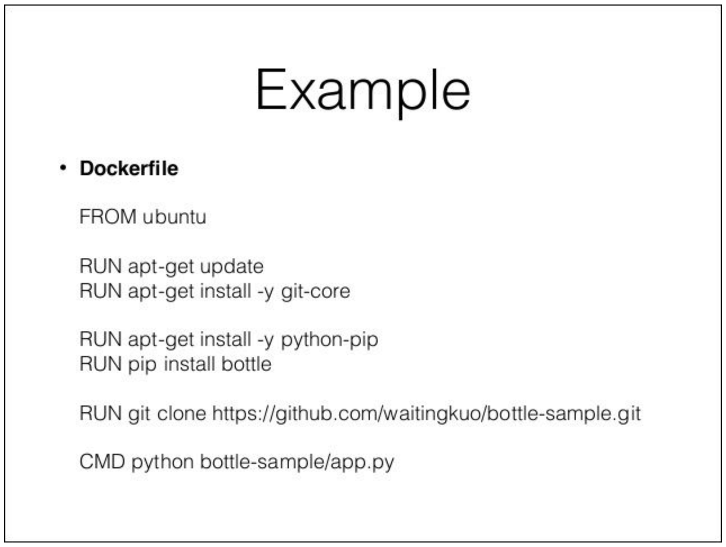
Docker is an open-source project that automates the deployment of applications inside software containers. Docker allows us to package our applications into a standardized unit for software development called a container. Docker containers wrap a piece of software in a complete filesystem that contains everything needed to run: code, runtime, system tools, system libraries – anything that can be installed on a server. This guarantees that the software will always run the same, regardless of its environment.

The Docker architecture provides independence from host environment. Docker containers are reproducible and portable for deployment. It allows us to run different workloads on the same hardware. Docker uses a client-server architecture as shown below:



An *image* is a combination of the filesystem and parameters to use at runtime. A docker image provides necessary libraries for software to run on a designated operating system.

A Docker container is a runnable instance of a Docker image. Docker can build images automatically by reading the instructions from a Dockerfile. A Dockerfile is a text document that contains all the commands a user could call on the command line to assemble an image. Below is an example of a simple Dockerfile.

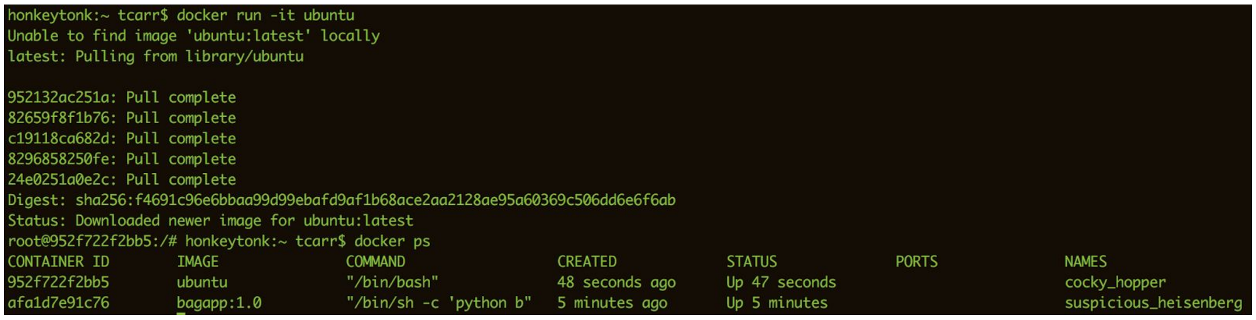


Example

- **Dockerfile**

```
FROM ubuntu

RUN apt-get update
RUN apt-get install -y git-core

RUN apt-get install -y python-pip
RUN pip install bottle

RUN git clone https://github.com/waitingkuo/bottle-sample.git

CMD python bottle-sample/app.py
```

Image below shows a simple docker command being run on a Linux terminal:



```
honkeytonk:~ tcarr$ docker run -it ubuntu
Unable to find image 'ubuntu:latest' locally
latest: Pulling from library/ubuntu

952132ac251a: Pull complete
82659f8f1b76: Pull complete
c19118ca682d: Pull complete
8296858250fe: Pull complete
24e0251a0e2c: Pull complete
Digest: sha256:f4691c96e6bbaa99d99ebafd9af1b68ace2aa2128ae95a60369c506dd6e6f6ab
Status: Downloaded newer image for ubuntu:latest
root@952f722f2bb5:/# honkeytonk:~ tcarr$ docker ps
CONTAINER ID        IMAGE               COMMAND               CREATED             STATUS              PORTS               NAMES
952f722f2bb5        ubuntu              "/bin/bash"           48 seconds ago      Up 47 seconds                           cocky_hopper
afa1d7e91c76        bagapp:1.0          "/bin/sh -c 'python b"   5 minutes ago   Up 5 minutes                            suspicious_heisenberg
```

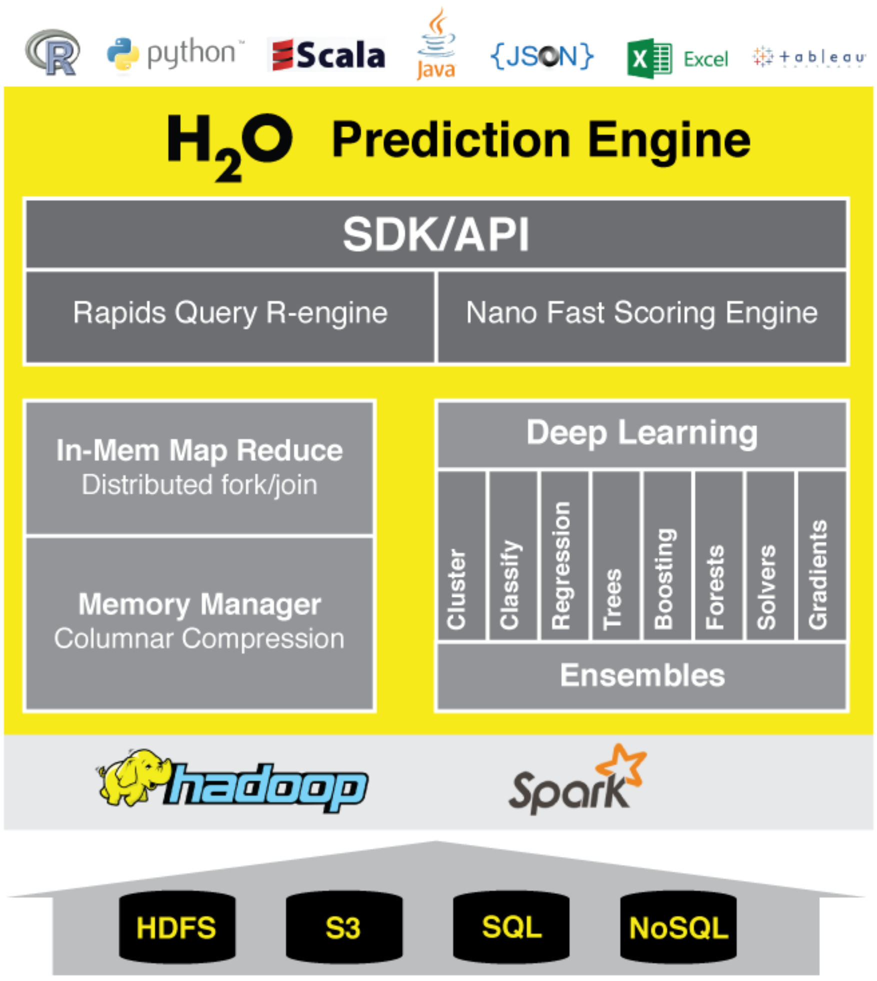
R

R is a language and environment for statistical computing and graphics. R provides a wide variety of statistical (linear and nonlinear modelling, classical statistical tests, time-series analysis, classification, clustering, …) and graphical techniques, and is highly extensible.

H2O

H2O is open-source browser based software for big-data analysis. It is used for exploring and analyzing datasets held in the cloud or distributed systems. H2O uses iterative methods that provide quick answers using all of the client's data. When a client cannot wait for an optimal solution, the client can interrupt the computations and use an approximate solution[8].

H2O is a Java Virtual Machine that is optimized for doing “in memory” processing of distributed, parallel machine learning algorithms on clusters. A “cluster” is a software construct that can be can be fired up on your laptop, on a server, or across the multiple nodes of a cluster of real machines, including computers that form a Hadoop cluster. According to the documentation a cluster’s “memory capacity is the sum across all H2O nodes in the cluster”[9].



Description of Algorithm used - GBM:
Gradient boosting originated in the observation by Leo Breiman[1] that boosting can be interpreted as an optimization algorithm on a suitable cost function. Explicit regression gradient boosting algorithms were subsequently developed by Jerome H. Friedman[2].

Gradient boosting is a machine learning technique for regression and classification problems, which produces a prediction model in the form of an ensemble of weak prediction models, typically decision trees. It builds the model in a stage-wise fashion, and it generalizes them by allowing optimization of an arbitrary differentiable loss function[10].

A weak hypothesis or weak learner is defined as one whose performance is at least slightly better than random chance. Gradient boosting combines weak "learners" into a single strong learner, in an iterative fashion. The idea is to use the weak learning method several times to get a succession of hypotheses, each one refocused on the examples that the previous ones found difficult and misclassified[11].

Gradient Boosting is basically about "boosting" many *weak* predictive models into a *strong* one, in the form of ensemble of weak models. To build the strong model, we need to find a good way to "combine" weak models. Boosting refers to this general problem of producing a very accurate prediction rule by combining rough and moderately inaccurate rules-of-thumb.

The GBM algorithm always trains models using data samples that are "difficult" to learn in previous rounds, which results in an ensemble of models that are good at learning different "parts" of training data. Gradient boosting is typically used with decision trees (especially CART trees) of a fixed size as base learners. In his talk titled “Gradient Boosting Machine Learning” held by H2O.ai, Stanford University professor Trevor Hastie, made the comment that in general gradient boosting performs better than random forest, which in turn performs better than individual decision trees.

Gradient Boosting > Random Forest > Bagging > Single Trees

GBMs are a family of powerful machine-learning techniques that have shown considerable success in a wide range of practical applications.

Gradient boosting involves three elements:

1. A loss function to be optimized.
2. A weak learner to make predictions.
3. An additive model to add weak learners to minimize the loss function.

For the purpose of this document, to reduce technical complexity, we leave further detail to the reader to explore.

Why we chose GBM as algorithm for our experiments?

In machine learning, a loss function or cost function is a function that maps an event or values of one or more variables onto a real number intuitively representing some "cost" associated with the event. An optimization problem seeks to minimize a loss function.

In GBMs, the loss functions applied can be arbitrary. The choice of the loss function is up to the researcher, with both a rich variety of loss functions derived so far and with the possibility of implementing one's own task-specific loss. This high flexibility makes the GBMs highly customizable to any particular data-driven task. It introduces a lot of freedom into the model design thus making the choice of the most appropriate loss function a matter of trial and error[12].

Boosting algorithms are also relatively simple to implement, which allows one to experiment with different model designs. Moreover, the GBMs have shown considerable success in not only practical applications, but also in various machine-learning and data-mining challenges (Bissacco et al.[3], Hutchinson et al.[4], Pittman and Brown[5], Johnson and Zhang[6]).

Project Planning - Phases	
Planned Milestone 1 **Setting up tools**	Automate launching of distributed instances to run the H2O.ai interface. We use Docker for setting up a portable quick-launch environment to run H2O. We then write Nomad jobs for reserving varying computational resources for these Docker containers. A dockerized launchable instance for R is already available.
Planned Milestone 2 **Conducting experiments**	A CSV data set of records with flight arrival and departure details for all commercial flights within the USA is available to us. We use a subset of flight data from October 2005 to April 2007. There are nearly 21 million records in total. The data must first be formatted correctly to input to H2O and R. Then, we will train a classifier. The goal is to consider all factors that will affect a flight and return the probability of a flight being delayed To test the impacts of resource allocation, on performance of running GBM on H2O and R, we run these algorithms as distributed jobs on the cluster. This allows us to compare a parallel environment to traditional R environment answering an essential question for our team on how we can improve the speed of machine learning algorithms with optimal utilization of our current computing cluster. Model evaluation will be performed to compare the estimated classification performance.
Planned Milestone 3 **Documenting results and presentation**	Outcome of this phase is a report containing the main results and findings of the experiments. It should give a detailed overview of the experimental setup and results, showing how different setups influence different factors such as model performance, cost, resource consumption, time, etc.

Goals Achieved

Milestone 1 - Setting up tools

Setting up H2O:

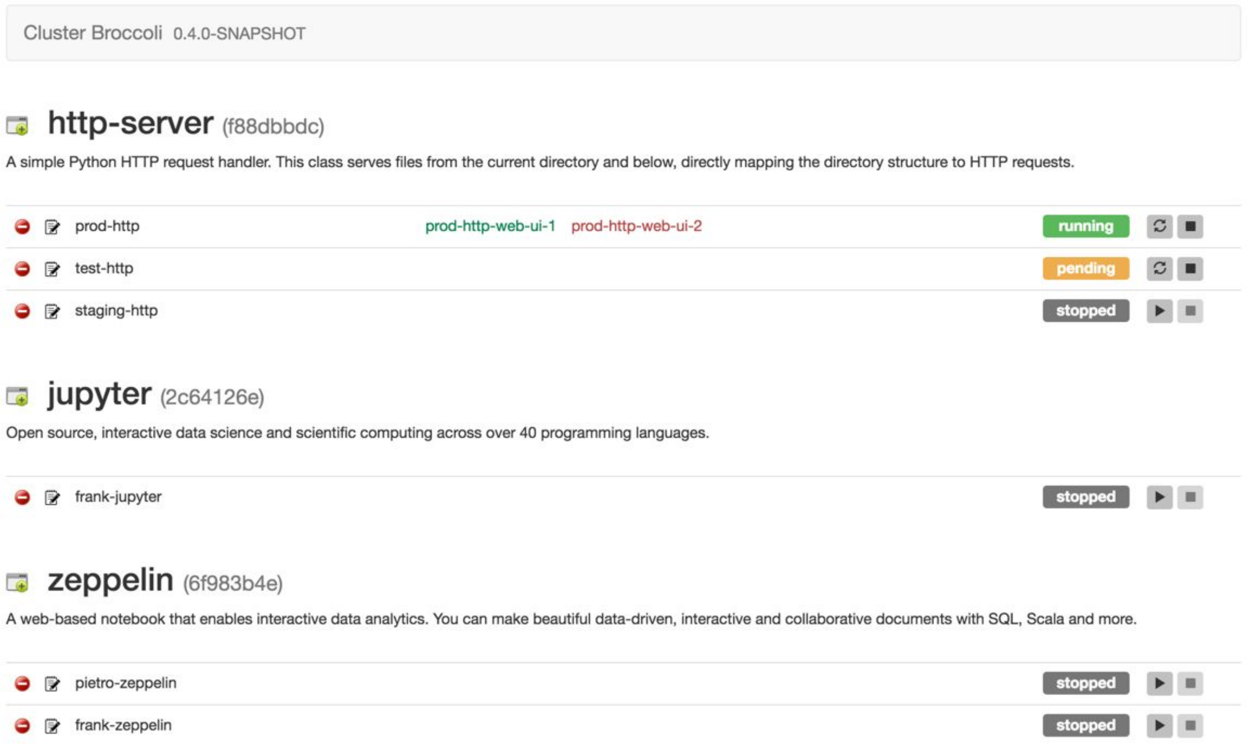
We studied how to make H2O installation process into an executable Docker instance. For this purpose we use a base of Ubuntu:14.04. We then proceeded to downloading the latest stable release of H2O.ai and exposing the necessary ports all within a Dockerfile. Our Dockerfile can now be built/run by the team by following these simple steps:

```
Step 1: Install Docker on your computer

Step 2: Run this command to fetch and run the H2O (latest stable release) docker image:

       docker run -ti zeecitizen/h2o-zeeshan:latest

This command pulls and runs the H2O.ai image saved at this online docker repository
location: 'zeecitizen/h2o-zeeshan'.

The setup is done. The output in the terminal will look like this:

       INFO: Open H2O Flow in your web browser: http://172.17.0.2:54321

Now you can launch the H2O instance in your browser by typing the ip address given in the
output.
```

Setting up Nomad:

After setting up H2O we had to automate launching of distributed instances of H2O. For this purpose we studied Nomad and how it works. During the first month of the project, we were able to code a Nomad job template successfully. The Nomad template allows integration of a new tool H2O into an existing web browser based interface used by the team. Using this interface, the team can start/stop H2O on one click of a button. This saves the time required to set up the right host operating system and libraries for H2O to run. The Dockerfile described in the previous section is used to run H2O in a portable container.

We committed the coded Nomad job template to the central source code repository of Allianz in Munich. Here is a screenshot of the web interface (called Broccolli) where we integrated our Nomad template for use:



Cluster Broccoli 0.4.0-SNAPSHOT

http-server (f88dbbdc)

A simple Python HTTP request handler. This class serves files from the current directory and below, directly mapping the directory structure to HTTP requests.

prod-http	prod-http-web-ui-1 prod-http-web-ui-2	running
test-http		pending
staging-http		stopped

jupyter (2c64126e)

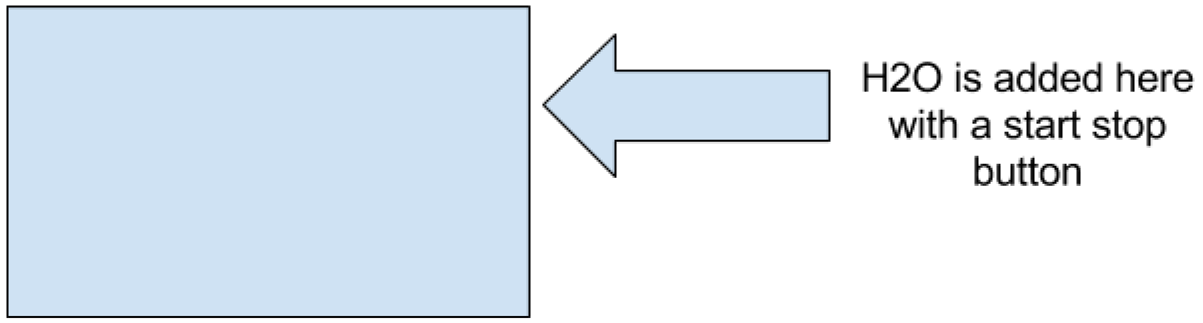
Open source, interactive data science and scientific computing across over 40 programming languages.

frank-jupyter	stopped

zeppelin (6f983b4e)

A web-based notebook that enables interactive data analytics. You can make beautiful data-driven, interactive and collaborative documents with SQL, Scala and more.

pietro-zeppelin	stopped
frank-zeppelin	stopped



H2O is now a tool available for use by the Allianz teams through their existing web interface. This has been a definite success of the first milestone of this project.

The web interface where we integrated our Nomad template is called Cluster Broccoli. The Nomad template that we were able to write for running H2O can be found at this link:

```
https://github.com/zeecitizen/cluster-broccoli/blob/master/templates/h2o/template.json
```

Following are few lines from the Nomad Template for H2O:

```
{
    "Job": {
        "Region": "global",        Comment:
        "ID": "{{id}}",            Define our simple H2O Job and
        "Datacenters": [           Run only in 'dc1' data center
            "dc1"
        ],
        "Constraints": null,
        "TaskGroups": [
            {
                "Name": "h2o",             Comment: Pulling number of 'nodes'
                "Count": {{nodes}},        as specified in web interface
                "Constraints":                                          null,
                "Tasks": [
                    {
                        "Name": "h2o",
                        "Driver": "raw_exec",
                        "User": "",
                        "Config": {
                            "command": "nomad-docker-wrapper",
                            "args": ["--entrypoint", "java",          Comment: Pulling Docker
                                  "--net", "host",                     Image from repository
                                     "-e",                             and launching a 4 GB
"H2O_NODE_COUNT={{nodes}}",                                            JVM to run H2O on this
                                     "zeecitizen/h2o-zeeshan:latest",  node
                                  "-Xmx4g", "-jar", "/opt/h2o.jar",
                                    "-name", "{{id}}",
                                    "-ip", "${NOMAD_IP_h2o}",
                                    "-port", "${NOMAD_PORT_h2o}"]
                        }
}
```

Milestone 2 - Conducting experiments

The scripts used for experiments with running GBM algorithm on RStudio with different configurations is published at this github repository

```
https://github.com/zeecitizen/H2O-vs-R-GBM
```

```
A small excerpt from the code for running gbm within h2O

system.time({
  md <- h2o.gbm(x = Xnames, y = "dep_delayed_15min", training_frame = dx_train, distribution
= "bernoulli",
          ntrees = 1000,
          max_depth = 16, learn_rate = 0.01, min_rows = 1,
          nbins = 100)
})
```

Description of Data Set used

The goal is to have the machine consider all possible factors that will affect a flight and return the probability of a flight being delayed.

The data set we are using consists of flight arrival and departure details for all commercial flights within the USA[7], from October 2005 to April 2007. This is a subset of the original dataset: there are nearly 21 million records in total, and takes 1 gigabytes of space when uncompressed.

By predicting potential flight delays we can help users make contingency plans. Recommendation engines can forewarn flyers of possible delays and rank flight options accordingly. Some businesses might even pay more for a flight to ensure their shipments arrive on time.

Description of variables in Data Set:

Name:	**Description**
Year	1987-2008
Month	1-12
DayofMonth	1-31
DayOfWeek	1 (Monday) - 7 (Sunday)
DepTime	actual departure time (local, hhmm)

CRSDepTime	scheduled departure time (local, hhmm)
ArrTime	actual arrival time (local, hhmm)
CRSArrTime	scheduled arrival time (local, hhmm)
UniqueCarrier	unique carrier code
FlightNum	flight number
TailNum	plane tail number
ActualElapsedTime	in minutes
CRSElapsedTime	in minutes
AirTime	in minutes
ArrDelay	arrival delay, in minutes
DepDelay	departure delay, in minutes
Origin	origin IATA airport code
Dest	destination IATA airport code
Distance	in miles
TaxiIn	taxi in time, in minutes
TaxiOut	taxi out time in minutes
Cancelled	was the flight cancelled?
CancellationCode	reason for cancellation (A = carrier, B = weather, C = NAS)
Diverted	1 = yes, 0 = no
CarrierDelay	in minutes
WeatherDelay	in minutes
NASDelay	in minutes
SecurityDelay	in minutes
LateAircraftDelay	in minutes

```
More details on Dataset here: http://stat-computing.org/dataexpo/2009/the-data.html
```

Bash Script used for Generating the data set:
The following linux bash script brings CSV data files for year 2005, 2006 and 2007 containing data which is described in the previous section.

```
for yr in 2005 2006 2007; do
  wget http://stat-computing.org/dataexpo/2009/$yr.csv.bz2
  bunzip2 $yr.csv.bz2
done
```

As a next step we loaded this data into H2O for performance comparisons.

Script 1 for comparison - Running GBM using H2O on RStudio

To run the GBM algorithm on our dataset, we've extended this script (given below) from Szilard Pafka's work. The script is written using the **H2O toolkit for R**. The author's github repository can be found here:

```
https://github.com/szilard/benchm-ml
```

We vary the dataset from 1 million rows to ten million rows and record the time it takes for it to train a model and predict the result. Finally, we print out the accuracy.

```
library(h2o)

h2o.init(max_mem_size="60g", nthreads=-1)

dx_train <- h2o.importFile(path = "train-1m.csv")
dx_test <- h2o.importFile(path = "test.csv")


Xnames <- names(dx_train)[which(names(dx_train)!="dep_delayed_15min")]


system.time({
  md <- h2o.gbm(x = Xnames, y = "dep_delayed_15min", training_frame = dx_train, distribution
= "bernoulli",
          ntrees = 1000,
          max_depth = 16, learn_rate = 0.01, min_rows = 1,
          nbins = 100)
})


system.time({
  print(h2o.auc(h2o.performance(md, dx_test)))
})
```

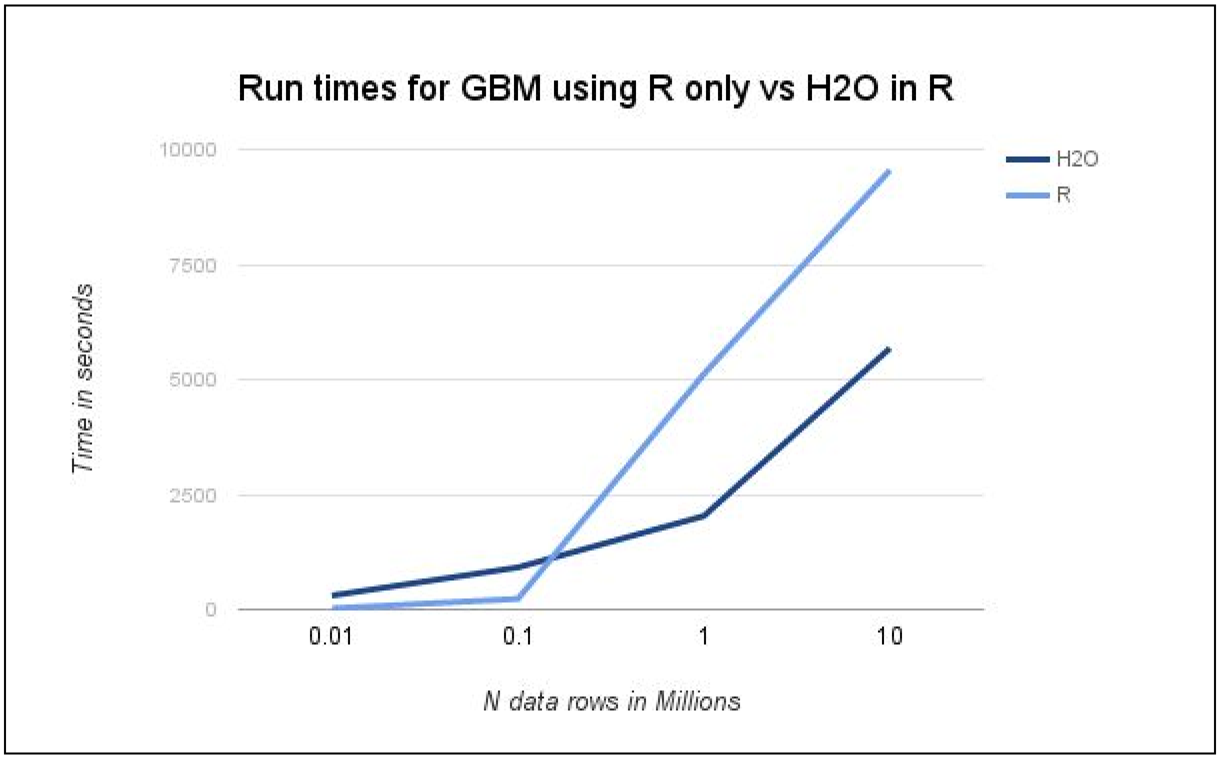
Script 2 for comparison - Running GBM using R only

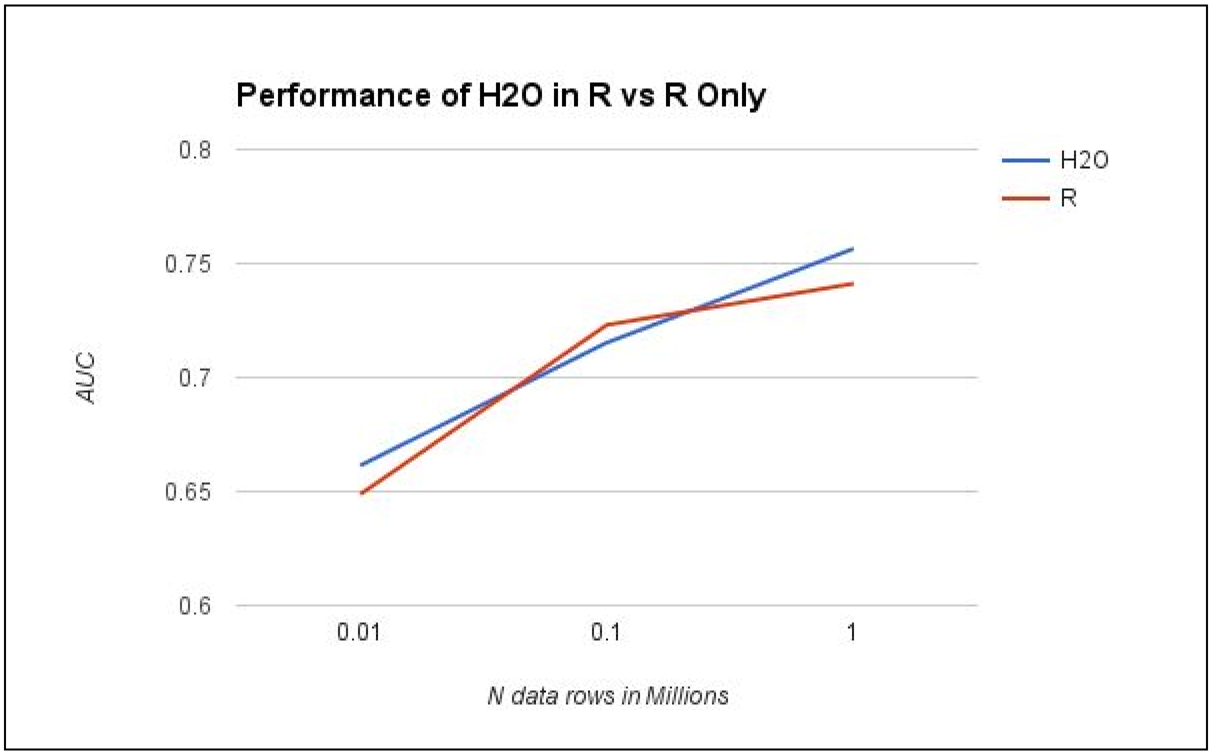
We've written this script (given below) to run the GBM algorithm in RStudio on our dataset **using simple R without H2O** (for performance comparison reasons). We vary the dataset from 1 million rows to ten million rows and record the time it takes for it to train a model and predict the result. Finally, we print out the accuracy.

```
library(ROCR)
library(gbm)

set.seed(123)

d_train <- read.csv("train-1m.csv")
d_test <- read.csv("test.csv")
d_train$dep_delayed_15min <- ifelse(d_train$dep_delayed_15min=="Y",1,0)
d_test$dep_delayed_15min <- ifelse(d_test$dep_delayed_15min=="Y",1,0)

facCols <- c("UniqueCarrier", "Origin","Dest", "Month", "DayofMonth", "DayOfWeek")
numCols <- c("DepTime","Distance")

for (k in facCols) {
  d_train[[k]] <- as.factor(d_train[[k]])
  d_test[[k]] <- as.factor(d_test[[k]])
}

for (k in numCols) {
  d_train[[k]] <- as.numeric(d_train[[k]])
  d_test[[k]] <- as.numeric(d_test[[k]])
}

system.time({
  md <- gbm(dep_delayed_15min ~ ., data = d_train, distribution = "bernoulli",
            n.trees = 1000,
            interaction.depth = 16, shrinkage = 0.01, n.minobsinnode = 1,
            bag.fraction = 0.5, n.cores = 32)
})


phat <- predict(md, newdata = d_test, n.trees = md$n.trees, type = "response")
rocr_pred <- prediction(phat, d_test$dep_delayed_15min)
performance(rocr_pred, "auc")@y.values[[1]]
```

Recorded results from single-node experimental setup:

Algorithm under test: GBM				
Method: Comparing run times for simple R and H2O in R				
Tool Used	***N data rows***	**Time (sec)**	**RAM (GB)**	**AUC**
R	10K	22.957	52.78	0.64879
	100K	230.425	52.78	0.72306
	1M	5121.748	52.78	0.7411664
	10M			
H2O	10K	302.662	52.78	0.6613423
	100K	914.307	52.78	0.7153
	1M	2032	53.33	0.7565
	10M			





Note:

Gradient boosting is typically used with decision trees of a fixed size as base learners. We keep the number of trees (M) to be 1000 in our runs. Friedman's first gradient boosting paper, the author's comment on the trade-off between the number of trees (M) and the learning rate (v) and recommend a small value for the learning rate < 0.1.

Smaller values of v lead to larger values of M for the same training risk, so that there is a tradeoff between them. The best strategy appears to be to set v to be very small (v < 0.1) and then choose M by early stopping. We keep the learning rate to a low of 0.01 in our runs.

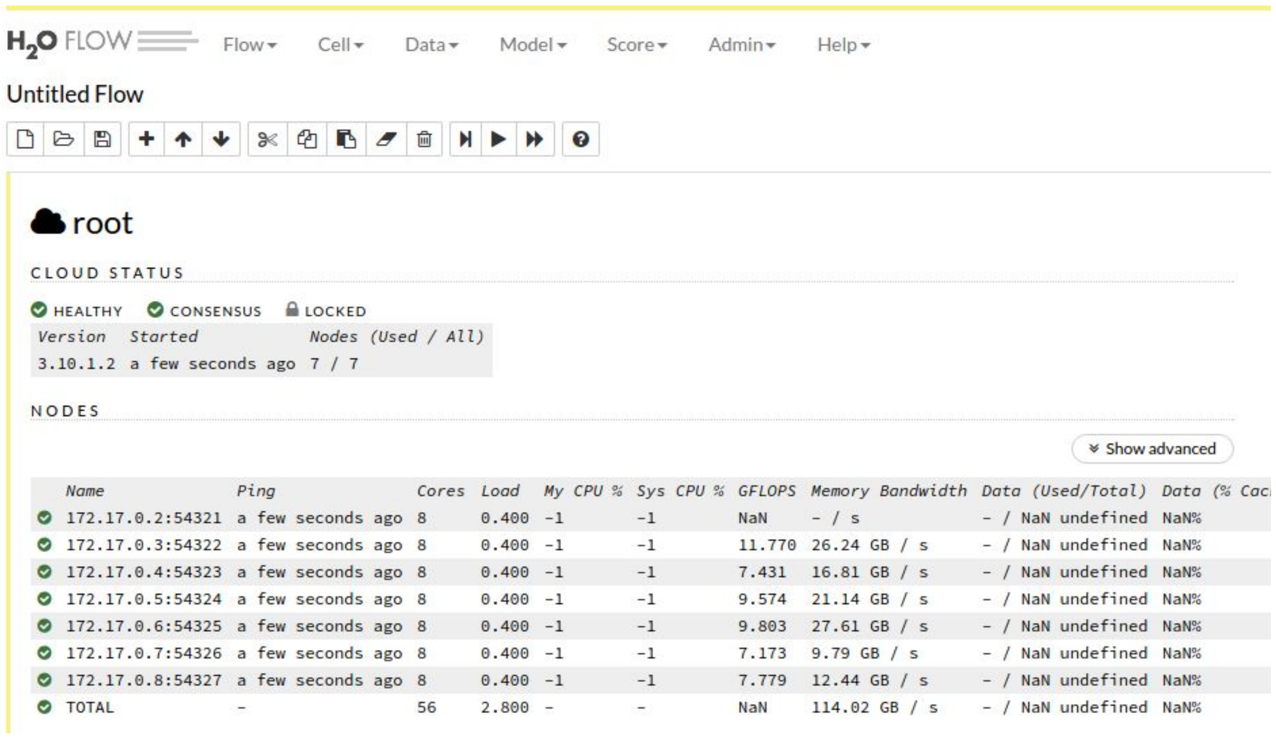
H2O Multi-Node Experiment:

H2O allows to run distributed implementations of different algorithms like GBM, Random Forest and Deep Neural Nets. H2O can be setup to speed up machine learning problems on a laptop as a local multi-core cluster, or it can be used in a multi-node cluster setting for example, on Amazon EC2. The purpose of our experiment is to see how scaling the number of nodes in an H2O cluster effect the running time of a particular model for classification.

Cluster Setup for H2O multi-node experiment

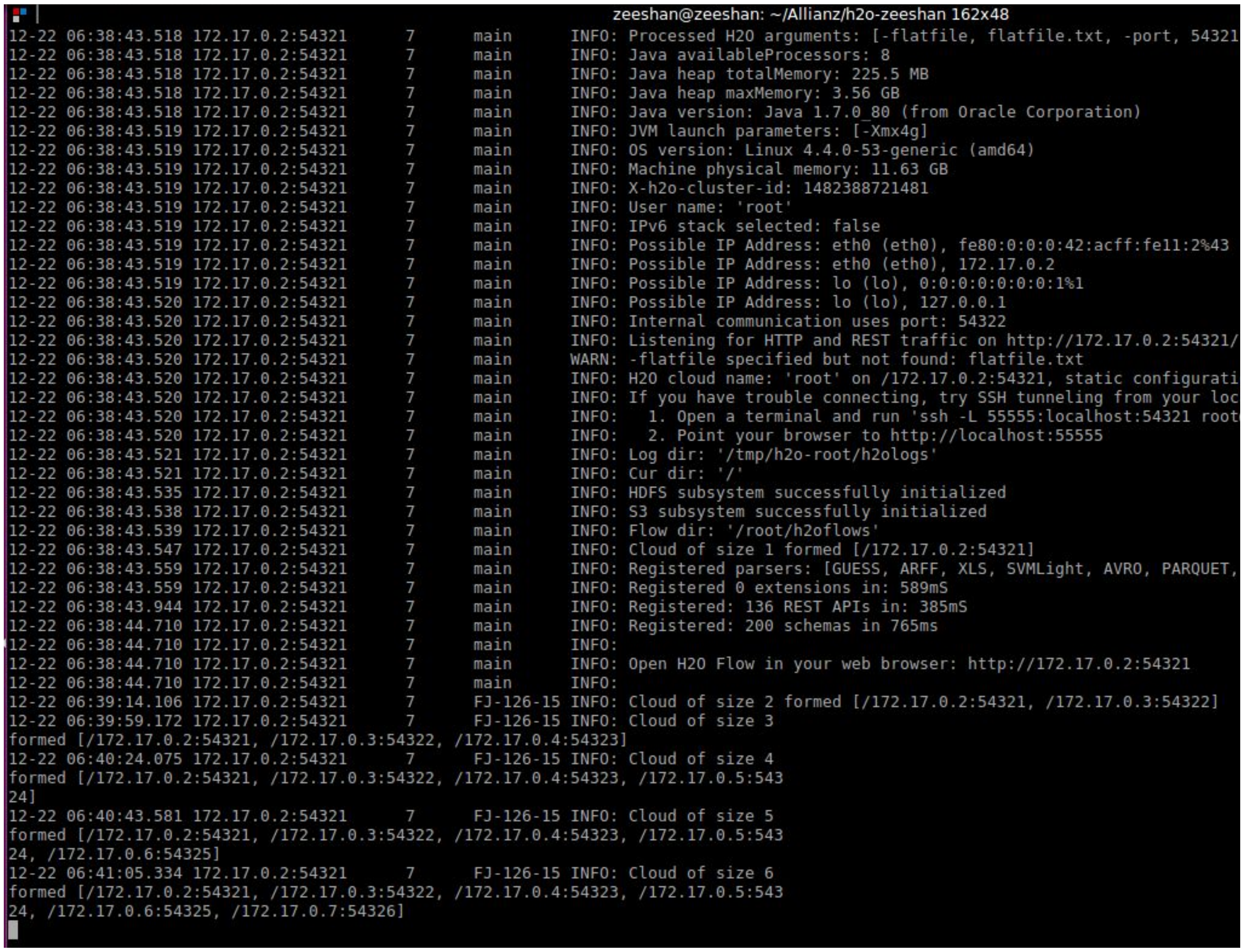
For the purpose of our multi-node experiment, we set up an H2O cloud which can consist of several nodes. We use a text file called 'flat file' to describe the topology of an H2O cluster. The flat file contains IP addresses of each node on the cluster and it needs to be passed to each node in the cluster so that they may connect to form a cluster. New H2O nodes can only join in at the time of launch.

Below is a screenshot of how our H2O server looks like with 7 nodes added to the H2O cloud during our experiments. Afterwards we describe our dataset.



Version	Started	Nodes (Used / All)
3.10.1.2	a few seconds ago	7 / 7

	Name	Ping	Cores	Load	My CPU %	Sys CPU %	GFLOPS	Memory Bandwidth	Data (Used/Total)	Data (% Cac
✓	172.17.0.2:54321	a few seconds ago	8	0.400	-1	-1	NaN	- / s	- / NaN undefined	NaN%
✓	172.17.0.3:54322	a few seconds ago	8	0.400	-1	-1	11.770	26.24 GB / s	- / NaN undefined	NaN%
✓	172.17.0.4:54323	a few seconds ago	8	0.400	-1	-1	7.431	16.81 GB / s	- / NaN undefined	NaN%
✓	172.17.0.5:54324	a few seconds ago	8	0.400	-1	-1	9.574	21.14 GB / s	- / NaN undefined	NaN%
✓	172.17.0.6:54325	a few seconds ago	8	0.400	-1	-1	9.803	27.61 GB / s	- / NaN undefined	NaN%
✓	172.17.0.7:54326	a few seconds ago	8	0.400	-1	-1	7.173	9.79 GB / s	- / NaN undefined	NaN%
✓	172.17.0.8:54327	a few seconds ago	8	0.400	-1	-1	7.779	12.44 GB / s	- / NaN undefined	NaN%
✓	TOTAL	-	56	2.800	-	-	NaN	114.02 GB / s	- / NaN undefined	NaN%



```
zeeshan@zeeshan: ~/Allianz/h2o-zeeshan 162x48
12-22 06:38:43.518 172.17.0.2:54321      7      main      INFO: Processed H2O arguments: [-flatfile, flatfile.txt, -port, 54321
12-22 06:38:43.518 172.17.0.2:54321      7      main      INFO: Java availableProcessors: 8
12-22 06:38:43.518 172.17.0.2:54321      7      main      INFO: Java heap totalMemory: 225.5 MB
12-22 06:38:43.518 172.17.0.2:54321      7      main      INFO: Java heap maxMemory: 3.56 GB
12-22 06:38:43.518 172.17.0.2:54321      7      main      INFO: Java version: Java 1.7.0_80 (from Oracle Corporation)
12-22 06:38:43.519 172.17.0.2:54321      7      main      INFO: JVM launch parameters: [-Xmx4g]
12-22 06:38:43.519 172.17.0.2:54321      7      main      INFO: OS version: Linux 4.4.0-53-generic (amd64)
12-22 06:38:43.519 172.17.0.2:54321      7      main      INFO: Machine physical memory: 11.63 GB
12-22 06:38:43.519 172.17.0.2:54321      7      main      INFO: X-h2o-cluster-id: 1482388721481
12-22 06:38:43.519 172.17.0.2:54321      7      main      INFO: User name: 'root'
12-22 06:38:43.519 172.17.0.2:54321      7      main      INFO: IPv6 stack selected: false
12-22 06:38:43.519 172.17.0.2:54321      7      main      INFO: Possible IP Address: eth0 (eth0), fe80:0:0:0:42:acff:fe11:2%43
12-22 06:38:43.519 172.17.0.2:54321      7      main      INFO: Possible IP Address: eth0 (eth0), 172.17.0.2
12-22 06:38:43.519 172.17.0.2:54321      7      main      INFO: Possible IP Address: lo (lo), 0:0:0:0:0:0:0:1%1
12-22 06:38:43.520 172.17.0.2:54321      7      main      INFO: Possible IP Address: lo (lo), 127.0.0.1
12-22 06:38:43.520 172.17.0.2:54321      7      main      INFO: Internal communication uses port: 54322
12-22 06:38:43.520 172.17.0.2:54321      7      main      INFO: Listening for HTTP and REST traffic on http://172.17.0.2:54321/
12-22 06:38:43.520 172.17.0.2:54321      7      main      WARN: -flatfile specified but not found: flatfile.txt
12-22 06:38:43.520 172.17.0.2:54321      7      main      INFO: H2O cloud name: 'root' on /172.17.0.2:54321, static configurati
12-22 06:38:43.520 172.17.0.2:54321      7      main      INFO: If you have trouble connecting, try SSH tunneling from your loc
12-22 06:38:43.520 172.17.0.2:54321      7      main      INFO:   1. Open a terminal and run 'ssh -L 55555:localhost:54321 root
12-22 06:38:43.520 172.17.0.2:54321      7      main      INFO:   2. Point your browser to http://localhost:55555
12-22 06:38:43.521 172.17.0.2:54321      7      main      INFO: Log dir: '/tmp/h2o-root/h2ologs'
12-22 06:38:43.521 172.17.0.2:54321      7      main      INFO: Cur dir: '/'
12-22 06:38:43.535 172.17.0.2:54321      7      main      INFO: HDFS subsystem successfully initialized
12-22 06:38:43.538 172.17.0.2:54321      7      main      INFO: S3 subsystem successfully initialized
12-22 06:38:43.539 172.17.0.2:54321      7      main      INFO: Flow dir: '/root/h2oflows'
12-22 06:38:43.547 172.17.0.2:54321      7      main      INFO: Cloud of size 1 formed [/172.17.0.2:54321]
12-22 06:38:43.559 172.17.0.2:54321      7      main      INFO: Registered parsers: [GUESS, ARFF, XLS, SVMLight, AVRO, PARQUET,
12-22 06:38:43.559 172.17.0.2:54321      7      main      INFO: Registered 0 extensions in: 589mS
12-22 06:38:43.944 172.17.0.2:54321      7      main      INFO: Registered: 136 REST APIs in: 385mS
12-22 06:38:44.710 172.17.0.2:54321      7      main      INFO: Registered: 200 schemas in 765ms
12-22 06:38:44.710 172.17.0.2:54321      7      main      INFO:
12-22 06:38:44.710 172.17.0.2:54321      7      main      INFO: Open H2O Flow in your web browser: http://172.17.0.2:54321
12-22 06:38:44.710 172.17.0.2:54321      7      main      INFO:
12-22 06:39:14.106 172.17.0.2:54321      7      FJ-126-15 INFO: Cloud of size 2 formed [/172.17.0.2:54321, /172.17.0.3:54322]
12-22 06:39:59.172 172.17.0.2:54321      7      FJ-126-15 INFO: Cloud of size 3
formed [/172.17.0.2:54321, /172.17.0.3:54322, /172.17.0.4:54323]
12-22 06:40:24.075 172.17.0.2:54321      7      FJ-126-15 INFO: Cloud of size 4
formed [/172.17.0.2:54321, /172.17.0.3:54322, /172.17.0.4:54323, /172.17.0.5:543
24]
12-22 06:40:43.581 172.17.0.2:54321      7      FJ-126-15 INFO: Cloud of size 5
formed [/172.17.0.2:54321, /172.17.0.3:54322, /172.17.0.4:54323, /172.17.0.5:543
24, /172.17.0.6:54325]
12-22 06:41:05.334 172.17.0.2:54321      7      FJ-126-15 INFO: Cloud of size 6
formed [/172.17.0.2:54321, /172.17.0.3:54322, /172.17.0.4:54323, /172.17.0.5:543
24, /172.17.0.6:54325, /172.17.0.7:54326]
```

Dataset for multi-node H2O experiment:

We keep the problem set the same as in our single node experiment. That is: predicting potential flight delays using a publicly available airlines dataset. However, we use a slightly different data set. The dataset used is a sample of what is more than two decades worth of flight data in order to ensure the download and import process would not take more than a minute or two.

The data comes originally from RITA where it is described in detail at (http://www.transtats.bts.gov/OT_Delay/OT_DelayCause1.asp). To use the entire 26 years worth of flight information to more accurately predict delays and cancellation we downloaded the total of data containing 152 Million Rows (Years: 1987-2013) - 14.5GB rows:

```
https://s3.amazonaws.com/h2o-airlines-unpacked/allyears.1987.2013.csv
```

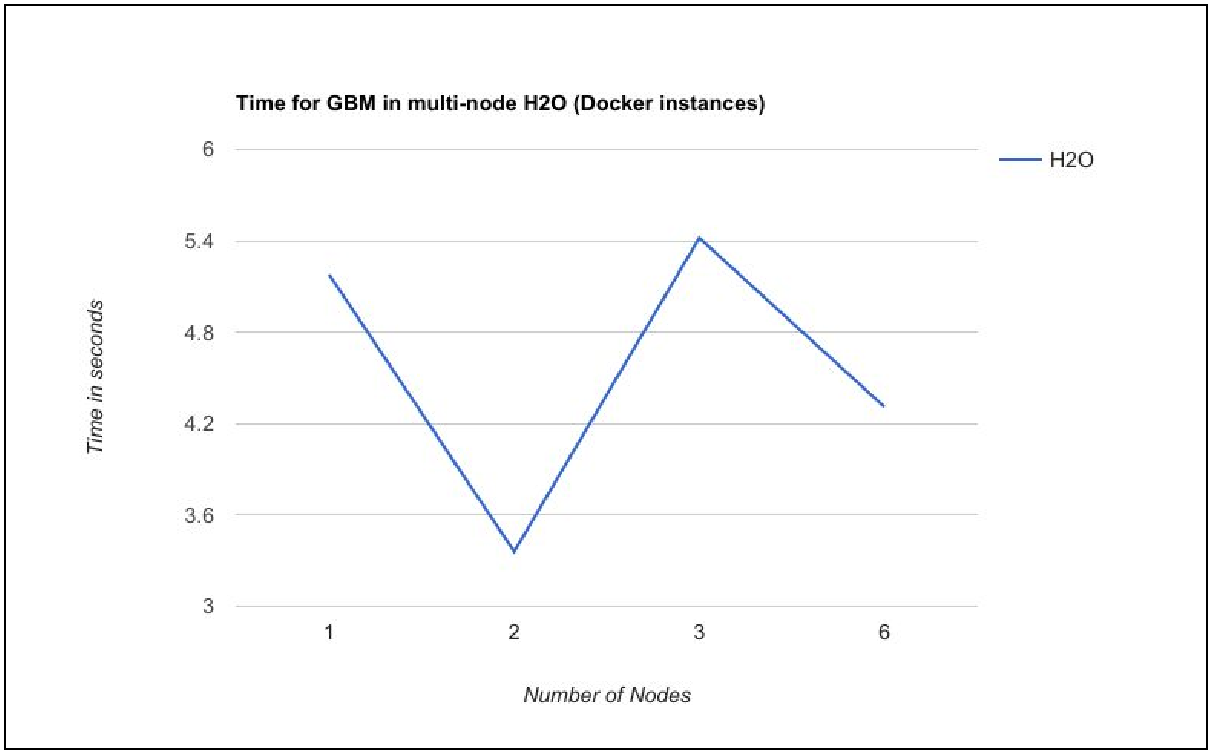
Since the available memory on the cluster for our experiment is 12GB, we split this dataset and used a 4 GB CSV file with 38 million rows.

Recorded results from multi-node experimental setup:

For our second experiment, we deployed multi-node H2O on a computing cluster created using Docker on a Core i7 machine with 12GB of available RAM. We set up multiple docker containers running Ubuntu 14.04 to act as nodes. Each node is initialized with specific memory. Each docker container (node) runs an H2O instance using this command given in the Dockerfile:

```
java -Xmx4g -jar /opt/h2o.jar -flatfile flatfile.txt -port 54321
```

The port number (54321 in this case) is changed to add more H2O instances to the cluster. The second parameter `-Xmx4g` is changed to vary the memory available to the node. The IPs and port numbers are listed in a text file called the 'flat file' copies of which are kept on each node. H2O reads the flatfile and creates an H2O Cloud for us.



Results from this chart are explained in the table below.

Algorithm under test: GBM
H2O Script (Flow) used for running GBM on this dataset:

`https://github.com/zeecitizen/H2O-vs-R-GBM/blob/master/GBM_Airlines_Classification.flow`	
Method: Comparing run times for H2O in R by varying number of nodes	
Tool Used: **H2O**	Number of Data Rows: **38 Million**
Results shown in table below	

Time (sec)	**No. of Nodes**	**Total Memory Available to Cluster**	**AUC** (Area under ROC curve)	**Observations**
00:09:27	**2**	4 GB 2g per node	0.681990	4GB of memory is not sufficient to contain dataset and JVM together. Thus, a slow down due to swapping.
00:05:18	**1**	6 GB 6g per node	0.681991	The available memory on single node is enough to process the whole dataset. It takes 5 minutes.
00:03:36	**2**	6 GB 3g per node	0.681990	Distributing the task to 2 Nodes while keeping the overall available memory to cluster same, we see a reduction in the time it takes for the algorithm to run.
00:05:42	**3**	6 GB 2g per node	0.681991	Increasing the number of nodes to three has negligible impact as the dataset easily fits in the two nodes and does not need a third.
00:04:31	**6**	6 GB 1g per node	0.681991	Having six nodes of 1gb each means adding maintenance overhead. Still performs better than a single node with 6gb available memory.

Comments on our result of multi-node H2O experiment:
By increasing the number of nodes available to H2O we saw that H2O actually distributes the job to the cluster, ***successfully reducing the time taken for computation***.

The results also show that if the dataset is small and can fit on a single node, adding more nodes do not influence the ‘time of run’ but rather adds computational overhead.

For best performance, H2O recommends on their website to size the cluster to be about four times the size of data (but to avoid swapping, memory allocation (Xmx) must not be larger than physical memory on any given node). Giving all nodes the same amount of memory is strongly recommended (H2O works best with symmetric nodes). Larger datasets slow down H2O as the underlying model becomes complex. Increasing number of nodes does not affect AUC much.

Conclusion

The R implementation of basic GBM package is inefficient in memory use when compared with H2O. It cannot cope by default with a large number of columns, therefore the data has to be one-hot encoded or we can go for feature selection to only retain some of the features. Since H2O's algorithm implementations are distributed, this allows H2O to scale to very large datasets that may not fit into RAM on a single machine.The H2O implementation is fast, memory efficient and uses all available cores. It deals with categorical variables automatically. It is also more accurate than the studied GBM R package, which may be because of dealing properly with the categorical variables.

R with the default matrix dynamic libraries can only use one CPU core. Revolution R community edition ships with the Intel Math Kernel Library. This allows for some matrix computations in parallel but definitely not as efficient as H2O. The R implementation using 1 processor core, runs out of memory already for n = 1M. In general, features of the R language are well suited for data wrangling or visualization tasks. It is just this particular implementation used by the GBM package that is inefficient. In our experiments we found that even with moderate amount of data, R may fail to build a model in limited time. H2O can handle larger amount of data really fast: the model runs in less than 2 minutes with all the data (1M rows)[13].

We can interface R with H2O using their R API. The benefit of combining R and H2O is that H2O is very good at exploiting multi-cores or clusters with minimal effort of the user. It is much harder to achieve the same efficiency in R alone. One more reason why H2O is much faster is that they have a very good indexing of their data and their algorithms are written such that they exploit parallelism to the fullest.

R is intended for use on data that fits into memory on one machine. So, R can very quickly consume all available memory. It is not intended for use with streaming data, big data or working across multiple machines. While building large scale machine learning models, H2O beats R in memory management. However, powerful statistical and graphical capabilities of R still make it a good fit for data processing tasks which don’t require much scaling. For example, the evaluation step in machine learning can be done in R. R is also great for automating the modeling flow. If we want to re-run our model several times, the web interface of H2O may be cumbersome as you have to re-enter the selections you made before.

Any new research in machine learning likely has an accompanying R package to go with it. So, in this respect, R stays at the cutting edge. R is free and open source. But the same can also be said about H2O, as its three Stanford professor advisors help keep it on the cutting edge of ML.

Achieved Goals:

- Successfully researched and learnt tools. Tools learnt are:
    - Docker
    - Nomad
    - H2O
    - R
- Built a Docker container for running H2O in R
- Built a distributed Nomad Job for running H2O
- Connected with Broccoli Web Interface
- Added H2O to toolkit for Allianz team in France
- Compare R vs H2O - Run Benchmark tests
- Scale H2O from Single to Multi-Node
- Configure Networking on Cluster at Allianz for Multi-Node support in H2O (still in progress)

- Documenting results

IDP Logistics Details:

Timeline:
Commencement of work: September 1st, 2016
Final Presentation of work: December 1st, 2016
Final report submission: March 14th, 2017

IDP Complementary Course:
Lecture course taken as part of IDP: Machine Learning in Robotics EI7419 - 5 ECTS.
Chair: Assistant Professorship of Dynamic Human-Robot-Interaction for Automation Systems (Prof. Lee), Summer Semester 2016.

Relevance of Lecture:
The lecture teaches the following topics to the student:

Applications of Machine Learning for Robots, Probability and Statistics, Density Estimation, linear regression, Pattern Classifiers, Probabilistic Methods for Classification, Dimensionality Reduction, PCA, Feature Selection, Statistical Clustering, Unsupervised Learning, EM algorithm, Validation, Support Vector Machines, Markov process, Hidden Markov Models, Dynamic Time Warping and Gaussian Mixture Models.

These topics have provided the student with a thorough insight on machine learning algorithms and their underlying mathematical models, necessary for the project.

References

1. Breiman, L. "Arcing The Edge" (June 1997)
2. Friedman, J. H. "Greedy Function Approximation: A Gradient Boosting Machine." (February 1999)
3. Bissacco A., Yang M.-H., Soatto S., Fast human pose estimation using appearance and motion via multi-dimensional boosting regression, in IEEE Conference on Computer Vision and Pattern Recognition, (2007)
4. Hutchinson R. A., Liu L.-P., Dietterich T. G., Incorporating boosted regression trees into ecological latent variable models, in AAAI'11, (San Francisco, CA), (2011), Available online at: http://www.aaai.org/ocs/index.php/AAAI/AAAI11/paper/view/3711
5. Pittman S. J., Brown K. A., Multi-scale approach for predicting fish species distributions across coral reef seascapes. (2011)
6. Johnson R., Zhang T., Learning Nonlinear Functions Using Regularized Greedy Forest. Technical Report. (2012)
7. More details on Dataset here: http://stat-computing.org/dataexpo/2009/the-data.html
8. (n.p.). (n.d.). In Wikipedia. Retrieved March 10, 2017, from https://en.wikipedia.org/wiki/H2O_(software)
9. (n.p.). (n.d.). In blog post. Retrieved March 10, 2017, from http://blog.revolutionanalytics.com/2014/04/a-dive-into-h2o.html
10. (n.p.). (n.d.). In Wikipedia. Retrieved March 10, 2017, from https://en.wikipedia.org/wiki/Gradient_boosting
11. Jason Brownlee (n.d.). In blog post. Retrieved March 10, 2017, from http://machinelearningmastery.com/gentle-introduction-gradient-boosting-algorithm-machine-learning/
12. Natekin, Alexey, and Alois Knoll. “Gradient Boosting Machines, a Tutorial.” Frontiers in Neurorobotics. (2013)
13. Andrie de Vries (n.d.). In blog post. Retrieved March 10, 2017, from http://blog.revolutionanalytics.com/2014/10/revolution-r-open-mkl.html