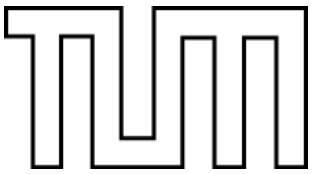


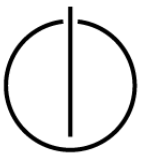
FAKULTÄT FÜR INFORMATIK

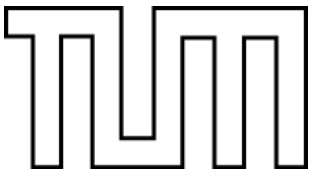
TECHNISCHE UNIVERSITÄT MÜNCHEN

Interdisciplinary Project (IDP) in Informatics

Machine Learning in Finance

Mohammad Zeeshan





FAKULTÄT FÜR INFORMATIK

TECHNISCHE UNIVERSITÄT MÜNCHEN

Interdisciplinary Project (IDP) in Informatics

Machine Learning in Finance

Maschinelles Lernen in der Finanzierung

Author:	Mohammad Zeeshan
Supervisor:	Patrick Bielstein, Lehrstuhl für Finanzmanagement und Kapitalmärkte, TUM
Advisor:	Dr. Gemma Garriga, Frank Rosner, Global Data & Analytics Team, Allianz SE
Submission Date:	July 13, 2017

Contents

1 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.

1.1 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.

1.2 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 hundreds of columns (features) along with millions of rows. We will apply Gradient Boosted Machines (GBM) to solve a classification task and compare the results in both environments with different setups.

1.2.1 Nomad

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

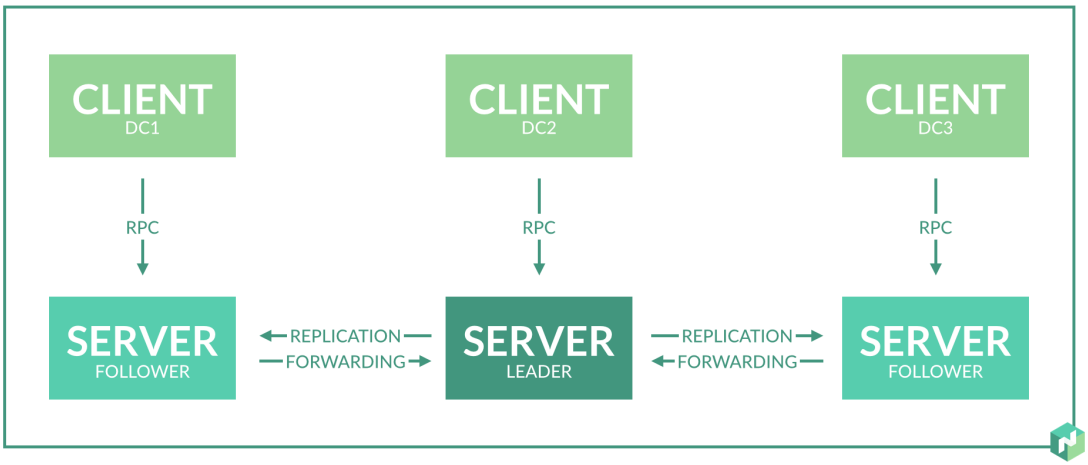


Figure 1: Nomad Client-Server Architecture

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

Job Template	Comments

`job "rstudio" {`
`  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`
`  }`
`  }`
`}`
	`#Define our simple Rstudio Job`
	
	`#Run only in US-West-1 data center`
	
	`#Define the rimple RStudio task using Docker`
	
	`#Name of the Docker image to use`
	
	`#Allocating resources for Nomad job`

Below is the hierarchy of a Nomad Job.

Job Template	Comments

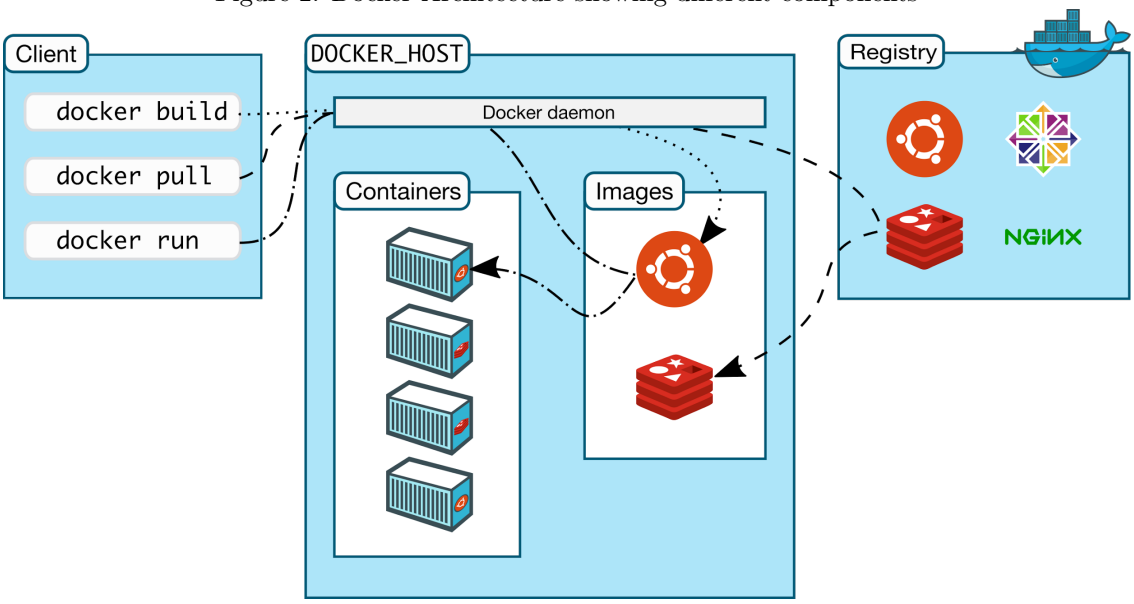
`Hierarchy of a Nomad Job`
`- Job`
`  - Task Group`
`  -Task`
	`Comments:`
	
	`The general hierarchy for a job is shown on left.`
	
	`Each job file has only a single job, however a job may have multiple groups, and each group may have multiple tasks. Groups contain a set of tasks that are co-located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

1.2.2 Docker:

Docker is used for the automation of deployment of application inside software containers. It is an open-source project (Profiles n.d.) (Passage n.d.). 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. Docker is used for the automation of deployment of application inside software containers. It is an open-source project (T. Brown n.d.). Docker uses a client-server architecture as shown below:

Figure 2: Docker Architecture showing different components



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 read instructions from a Dockerfile and build images in an automatic manner. A DockerFile is a text document which contains the scripts (commands) used to build or make an image. Below is an example of a simple Dockerfile.

Figure 3: Image 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
```

1.2.3 R

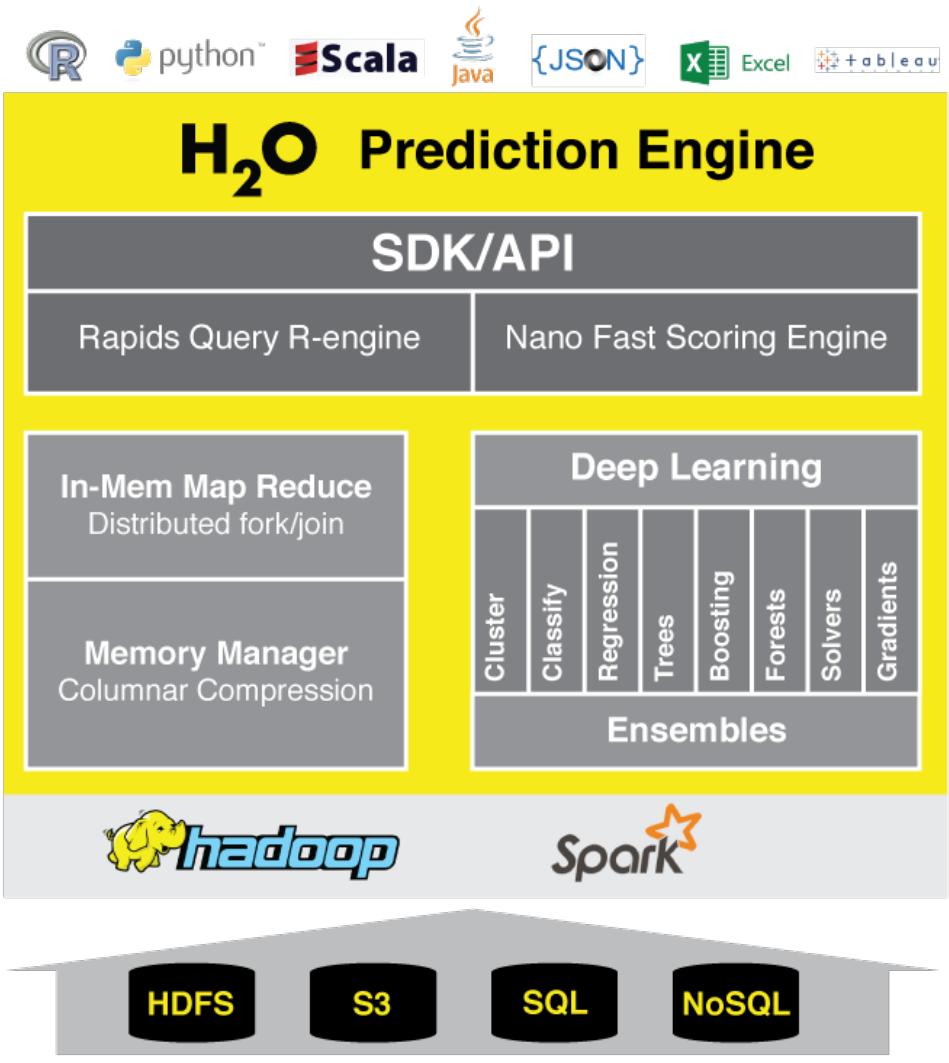
R is a language and environment for statistical computing and graphics (Wikipedia 2017c). R enables a wide range of statistical (classical statistical testing, time-series analysis, clustering, classification, linear and non-linear modelling) and graphical programming operations in a highly extensible manner.

1.2.4 H20

H2O is open-source browser based software for big-data analysis (Wikipedia 2017b). 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.

H20 is a Java Virtual Machine. It has been optimized to perform "in memory" processing of parallel, distributed and machine learning algorithms on clusters. A "cluster" is a type of software construct which can be started from a standalone laptop, server or across different nodes of a cluster of real machines which may include computers that constitute a Hadoop cluster. Documentation states that the "sum of memory capacity across all H20 nodes in a cluster gives us the cluster's total memory capacity" (Rickert n.d.)

Figure 4: The H2O internal architecture



1.2.5 Description of Algorithm used

Gradient boosting technique is a type of machine learning craft developed to tackle classification and regression problems. It generates prediction models in the shape of a group of weak prediction models, for example decision trees. Usually, the model is developed in a stage-wise manner and generalization is done by optimizing arbitrary differentiable loss function (Wikipedia 2017a).

Gradient boosting is basically an optimization algorithm which works with a reasonable cost function. They originated due to observation of Leo Breiman (Breiman 1997). Jerome H. Friedman developed explicit regression gradient boosting algorithms, and later on (Friedman 1999).

A weak learner or weak hypothesis is termed as the one whose conduct is at least somewhat better than a random chance. Gradient boosting combines weak "learners" into a strong single learner through an iterative process. The basic aim is to make use of weaker learning methods repetitively. Running these several times helps to get a cycle of hypotheses, each one retargeted on the inputs that the previous hypothesis determined as misclassified and difficult (Brownlee 2016). Gradient Boosting is simply about "boosting" several weaker predictive models into a strong one, in shape of ensemble of weak models. To build the strong model, we need to find a good way to "combine" weak models. Boosting ascribes to this general problem of generating an accurate prediction rule by joining rough and moderately inaccurate rules-of-thumb, as shown by [Freund and Schapire 1995]. The GBM algorithm always makes use of data samples which are "difficult" to learn in previous rounds, in order to train models. It leads to an ensemble of models that are good at learning different "parts" of training data. As base learners, Gradient boosting is typically employed with fixed size decision trees (especially CART trees), as mentioned by Professor Trevor Hastie in his talk "Gradient Boosting Machine Learning"; which took place at H20.ai, Stanford University. In this talk, Prof. Hastie also pointed out that generally gradient boosting outperforms random forest and random forest performs better than the individual decision trees. Thus the correlation:

Gradient Boosting >Random Forest >Bagging >Single Trees

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

GBMs are a class of commanding machine-learning mechanism that have seen substantial success in a broad range of practical applications.

Gradient boosting revolves around three constituents.

1. A loss function in need of optimization
2. A weak learner to attempt predictions.
3. An additive model to minimize the loss function by adding weak learners. (Chen et al. 2017)

1.2.6 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 (Reviews 2016). An optimization problem seeks to minimize a loss function.

In GBMs, the application of loss functions can be inconsistent. There is vast variety of loss functions derived so far and with the possibility of instrumenting one's own task-specific loss. Overall, the researcher has the option to choose the loss function. The customizability of GBMs to perform data-driven tasks comes from high flexibility. It brings a great degree of freedom into the model design enabling the choice of most suitable loss function a matter of trial and error (Natekin and Knoll 2013).

Implementation of Boosting algorithms is comparably simpler, which allows one to examine different model designs. Moreover, GBMs have demonstrated sizable success in not only practical

applicatins but also in different data-mining and machine-learning challenges (Bissacco et al. 2007)(Hutchinson et al. 2011)(Pittman and K. A. Brown 2011)(Johnson and Zhang 2014). .

Table 1: 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 details from United States local commercial flights is used (RITA n.d.). 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 possible result we want to achieve is an estimate of the probability (chance) of a particular flight to be delayed. In this pursuit, we consider all necessary contributing factors that influence the flight times. 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 Documentation	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.

2 Goals Achieved

2.1 Milestone 1 - Setting up tools

2.1.1 Setting up H20

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:

2.1.2 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:

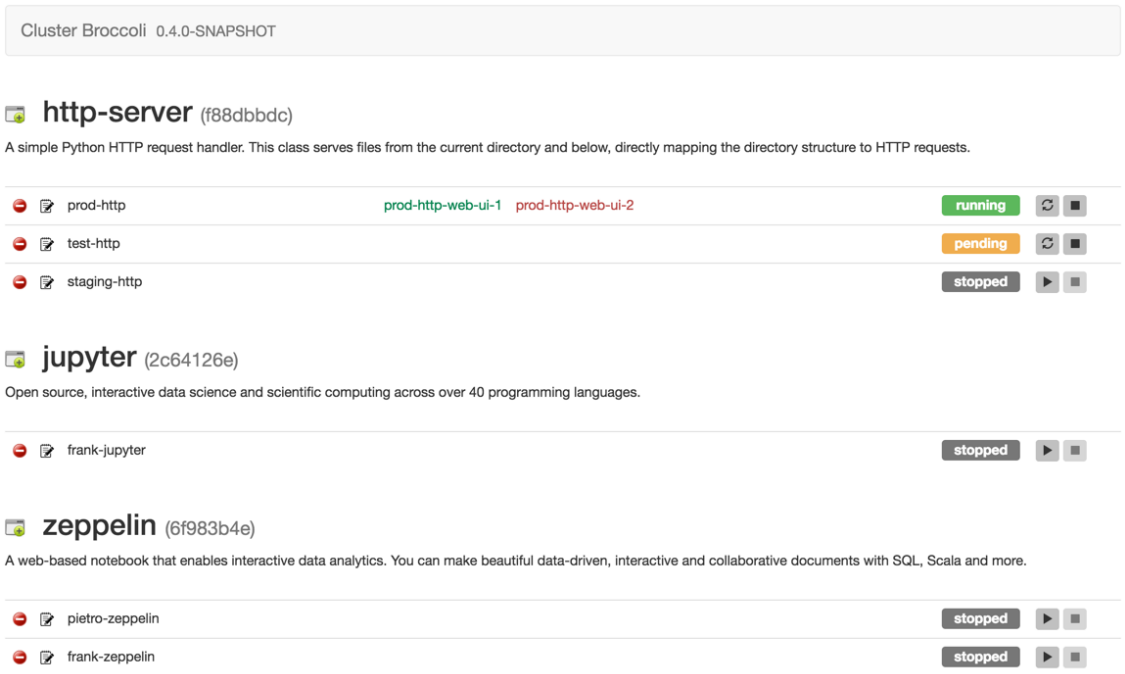
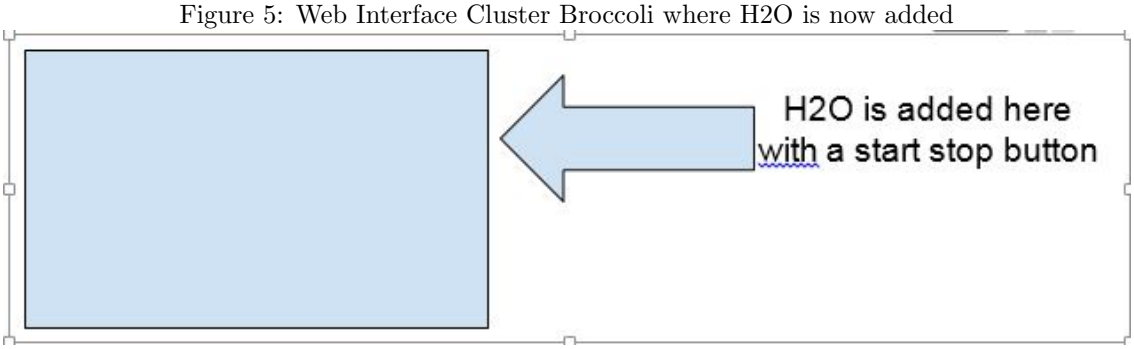


Figure 5: Web Interface Cluster Broccoli where H2O is now added



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",
        "ID": "{{id}}",
        "Datacenters": [
            "dc1"
        ],
        "Constraints": null,
        "TaskGroups": [
            {
                "Name": "h2o",
                "Count": {{nodes}},
                "Constraints": null,
                "Tasks": [
                    {
                        "Name": "h2o",
                        "Driver": "raw_exec",
                        "User": "",
                        "Config": {
                            "command": "nomad-docker-wrapper",
                            "args": ["--entrypoint", "java",
                                     "--net", "host",
                                     "-e", "H2O_NODE_COUNT={{nodes}}",
                                     "zeecitizen/h2o-zeeshan:latest",
                                     "-Xmx4g", "-jar", "/opt/h2o.jar",
                                     "-name", "{{id}}",
                                     "-ip", "${NOMAD_IP_h2o}",
                                     "-port", "${NOMAD_PORT_h2o}"]
                        }
}
```

2.2 Milestone 2 - Conducting experiments

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

Scripts can be found at this link:
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)
})
```

2.2.1 Description of Data Set used

The objective is to make the machine consider all possible aspects that will influence an airline flight. This helps us compute the chance of whether a flight will be delayed.

Flight details of all the commercial flights within the USA are included in our data set (RITA n.d.). These details include flight departure and arrival timings, from October 2005 to April 2007. This is a subset of original dataset. It takes nearly 1 Gigabyte of memory with approximately 21 million total records, when uncompressed.

Contingency plans can be developed by users if we can help them by predicting flight delays. Flight options can be ranked accordingly and users can be altered in advance by recommendation engines for possible delays. Some businesses may even opt to spend more to guarantee timely arrival of their shipments.

Table 2: Description of our dataset

Name	**Description (Data by RITA n.d.)**
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
TaxOut	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

2.2.2 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.

Linux Bash Script

```
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.

2.2.3 Script 1 for comparison - Running GBM using H2O on RStudio

To run the GBM algorithm on our dataset, we've written this script (given below). Szilard Pafka has also used similar scripts in his work at Pafka n.d.(b). Considering Szilard's work is incomplete and limited, we've written our own scripts suitable to our experimental setup. The script is written using R language which calls the H2O libraries.

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.

R Script 1

```
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)))
})
```

2.2.4 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

R Script 1

```
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:

Table 3: 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	N/A		
H2O	10K	302.662	52.78	0.6613423
	100K	914.307	52.78	0.7153
	1M	2032	52.78	0.7565
	10M	N/A		

Figure 6: Run time for GBM using R only vs H2O in R

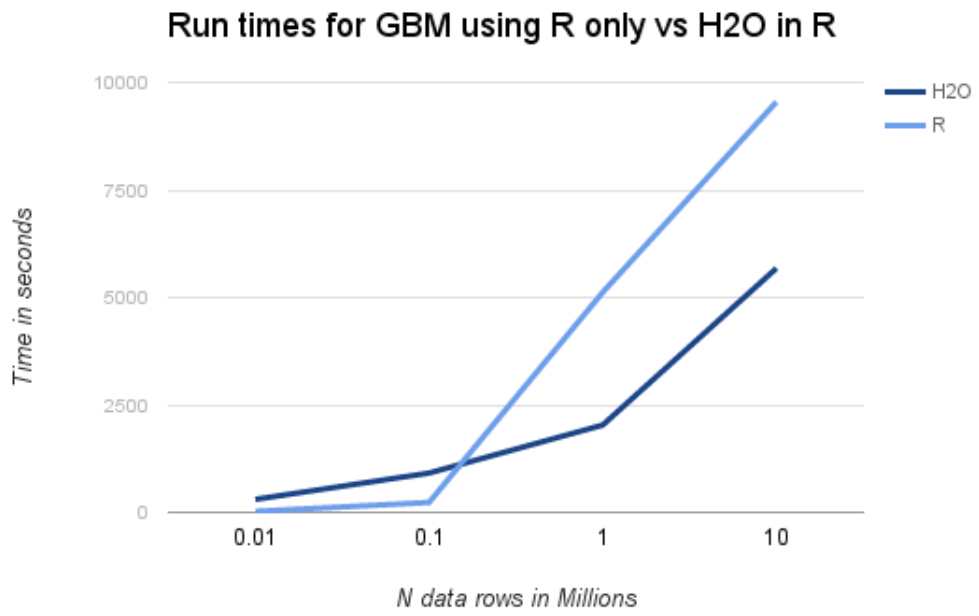
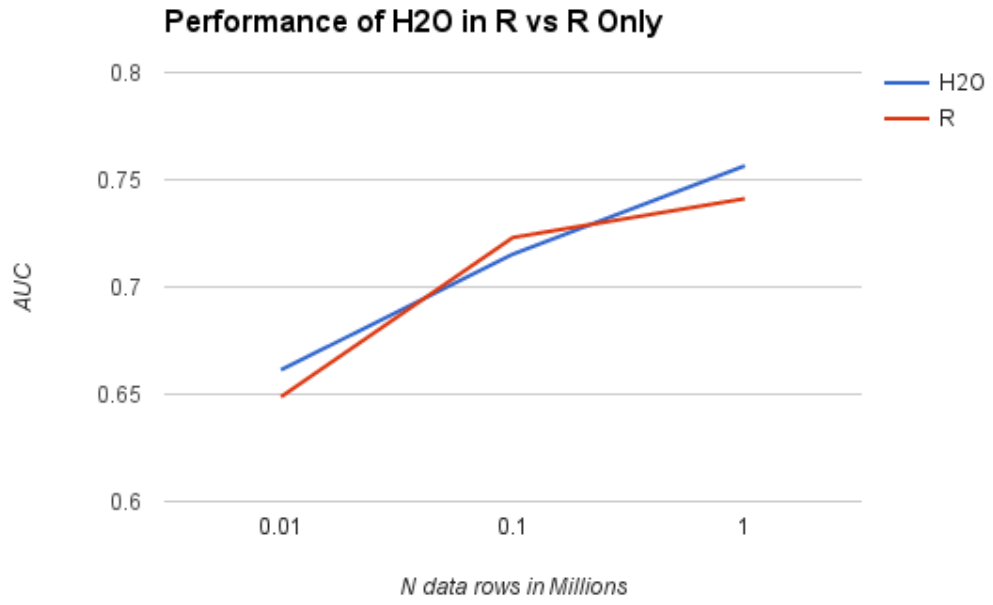


Figure 7: Performance of H2O in R vs R Only



Note:

Gradient boosting is typically used with decision trees of a fixed size as base learners, as mentioned by (Xu et al. 2016). We keep the number of trees (M) to be 1000 in our runs. Friedman commented on the trade-off between the learning rate (v) and the number of trees (M) in his first paper regarding gradient boosting. He recommended a small value for the learning rate ¡0.1 (Configure the Gradient Boosting Algorithm n.d.).

Larger M values result from smaller v values for identical risk in training. A balance of tradeoffs between both the values is achieved. This was also commented upon by Friedman 1999. Setting v to very small (v¡ 0.1) and then choosing M by early stopping seems to be the best strategy. We keep the learning rate to a low of 0.01 in our runs.

2.3 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.

2.3.1 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.

Figure 8: Successful creation of a cluster with 7 nodes.

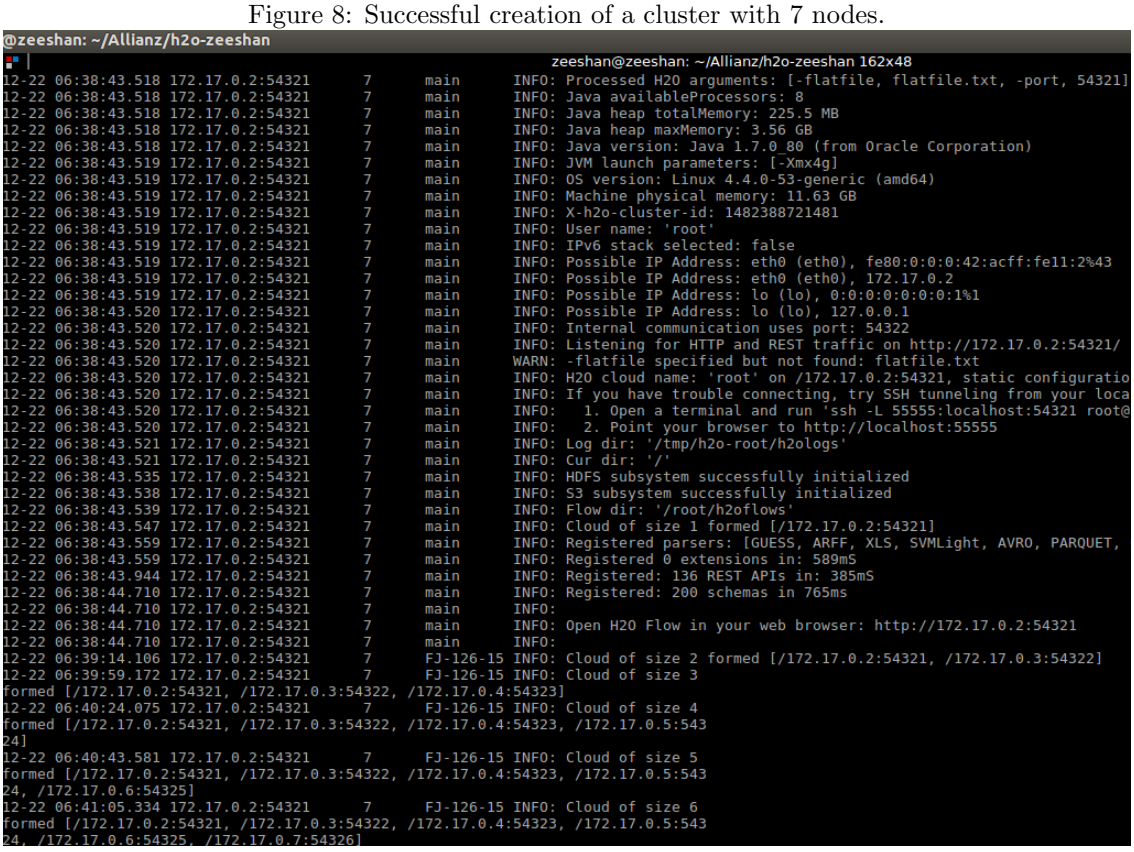
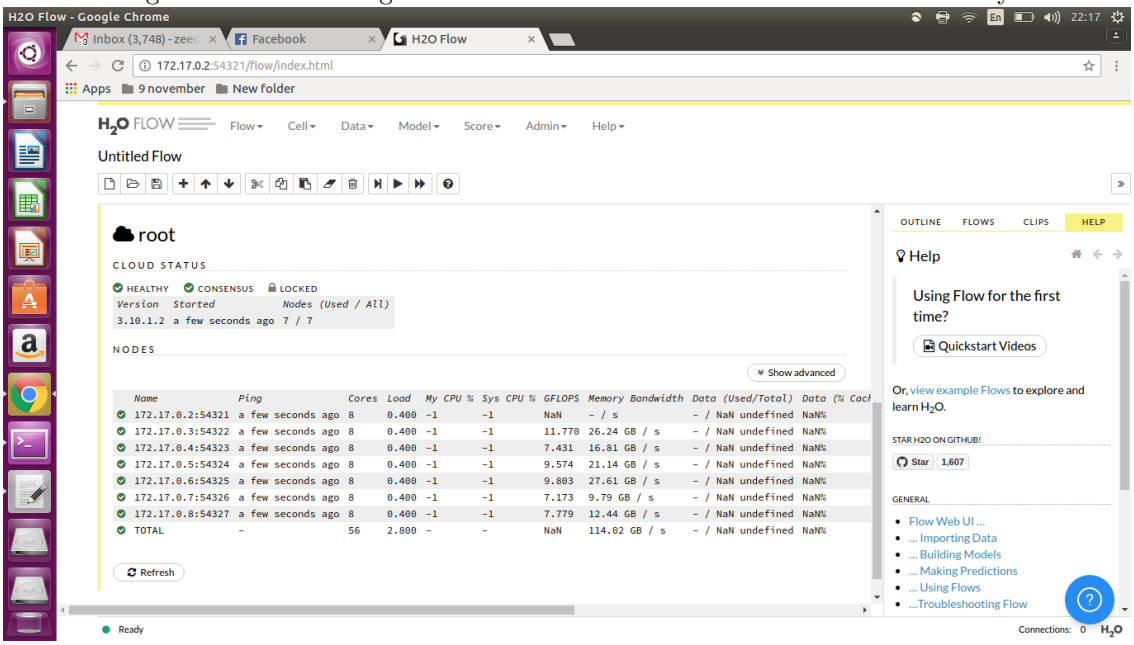


Figure 9: H2O showing nodes added to cloud based on total cluster memory



Dataset for multi-node H2O experiment:

We keep the problem set the same as in our single node experiment. That is: using publicly available airlines dataset to correctly predict potential future flight delays (Hill n.d.). However, we use a slightly different data set. The dataset utilized is a sample of flight data spanning over two decades. This has been done to ensure the quick download and import process (A. Wang n.d.

We had to download a total of 14.5GB data containing more than 150 million rows from year 1987-2013. This dataset contains 26 years of flight information (Bureau of Transportation Statistics n.d.). It will be used to predict cancellations and delays (RITA n.d.).

We can find the Dataset at the following link:
`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.

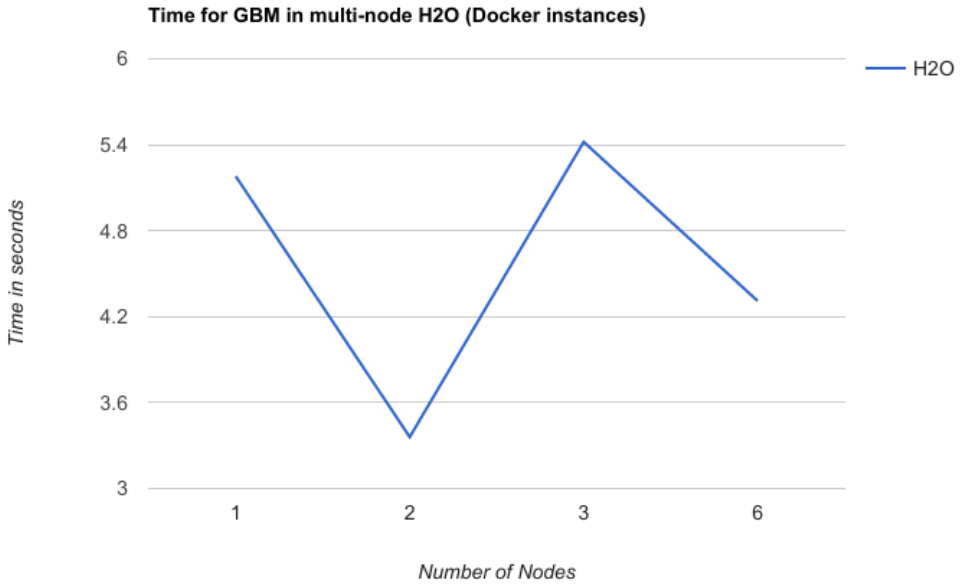
2.3.2 Recorded results from multi-node experimental setup:

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:

Command for launching H2O instance in Dockerized container:
```
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.

Figure 10: Time for GBM runs in multi-node H2O



H2O Script (Flow) used for running GBM on this dataset:
https://github.com/zeecitizen/H2O-vs-R-GBM/blob/master/GBM_Airlines_Classification.flow

Results from Figure 10 are explained in the table below.

Table 4: Method: Comparing run time for H2O in R by varying # of Nodes. #Data Rows: 38 Million

Time (sec)	#Nodes	Cluster Memory	AUC	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.

3 Evaluating Results

This section evaluates 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.

4 Conclusion

An analysis of R and H20 implementations of basic GBM reveals R implementation's memory use inefficiencies. The data has to be one-hot encoded due to the fact that R implementation can't deal with large number of columns by default. One possible solution is to proceed with feature selection scheme to retain some of the features (Pafka n.d.(a). On the other hand, distributed nature of H20's algorithm implementations make them highly scalable to very large datasets, which may not even size into RAM on a single machine. The H20 implementation is memory efficient and fast as it utilizes all the available cores (Pafka n.d.(a)). Categorical variables are handled automatically. Compared with GBM R package, H20's implementation is more accurate; one reason could be that it properly deals with categorical variables.

By default, R has matrix dynamic libraries that are meant to utilize one CPU core only. Revolution R community edition comes with Intel Match Kernel Library. This enables some extent of matrix computations in a parallel order but definitely it is still less efficient than the H20 (Data Science n.d.). The R implementation uses single processor core. For example, in usual cases, At n = one million, R can be seen as running out of memory. In general, features of the R language are well suited for data wrangling or visualization tasks. However, inefficiency has been observed only with the implementation used by the GBM package. Even with the limited amount of data we used in our experimental setup, R seems to slow down and sometimes fails to generate a model in reasonable time unless we add extensive memory. Same was also found by KDDCup98 competition winners ((Staff) n.d.). H20 can deal with greater amounts of data really quick. For example, for one million rows, execution of the model takes less than 2 minutes, which is quiet fast. This efficiency is also observed by (Vries n.d.).

We can interface R with H2O using their R API. The advantage of blending together R and H20 is that the H20 implementation is superb at maneuvering performance out of multi-cores or clusters with little effort from the user. It is much more difficult to reach the same levels of efficiency with R alone. Better data indexing and exploitation of parallelism to the fullest extent are some of the additional reasons for H20's faster performance.

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. H20 beats R in memory management especially when it comes to developing large scale machine learning models. 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 awesome when it comes to the automation of modeling flows. If we want to re-run our model several times, the web interface of H2O may be inconvenient as the user has to re-enter the selections he/she has 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 also, open source. But same is the case with H20 which has been kept on the cutting edge of Machine Learning by the three of its Stanford professors (Peck n.d.).

5 Appendix

List of Figures

List of Tables

References

[1] Alessandro Bissacco, Ming-Hsuan Yang, and Stefano Soatto. "Fast human pose estimation using appearance and motion via multi-dimensional boosting regression". In: *Computer Vision and Pattern Recognition, 2007. CVPR'07. IEEE Conference on*. IEEE. 2007, pp. 1–8.

[2] Leo Breiman. *Arcing the edge*. Tech. rep. Technical Report 486, Statistics Department, University of California at Berkeley, 1997.

[3] Taylor Brown. *Containers - Bringing Docker To Windows Developers with Windows Server Containers*. URL: https://msdn.microsoft.com/en-%20us/magazine/mt797649.aspx.

[4] Jason Brownlee. *A Gentle Introduction to the Gradient Boosting Algorithm for Machine Learning*. [Online; accessed 11-July-2017]. 2016. URL: http://machinelearningmastery.com/gentle-%20introduction-gradient-boosting-algorithm-machine-%20learning/.

[5] United States Department of Transportation Bureau of Transportation Statistics. *Airline On-Time Statistics and Delay Causes*. URL: https://www.transtats.bts.gov/OT_Delay/OT_DelayCaus%20e1.asp.

[6] Chu-Song Chen, Jiwen Lu, and Kai-Kuang Ma. *Computer Vision–ACCV 2016 Workshops: ACCV 2016 International Workshops, Taipei, Taiwan, November 20-24, 2016, Revised Selected Papers*. Vol. 10118. Springer, 2017.

[7] How to Configure the Gradient Boosting Algorithm. *http://abunchofdata.com/how-to-configure-the-gradient-boosting-algorithm/*. URL: http://www.300166.net/language/go/40511115703940971550.htm%20l.

[8] Society of Data Science. *How is H2O faster than R or SAS?* URL: https://datascience.stackexchange.com/questions/6884/how-is-%20h2o-faster-than-r-or-sas/6896.

[9] Yoav Freund and Robert E Schapire. "A desicion-theoretic generalization of on-line learning and an application to boosting". In: *European conference on computational learning theory*. Springer. 1995, pp. 23–37.

[10] J Friedman. "Greedy Function Approximation: A Gradient Boosting Machine http://www.salford-systems. com/doc". In: *GreedyFuncApproxSS. pdf* (1999).

[11] Brandon Hill. *intro to python*. URL: https://github.com/h2oai/h2o-tutorials/blob/master/tutorials/intro-to-r-python-flow/intro-to-python.py.

[12] Rebecca A Hutchinson, Li-Ping Liu, and Thomas G Dietterich. "Incorporating Boosted Regression Trees into Ecological Latent Variable Models." In: *AAAI*. Vol. 11. 2011, pp. 1343–1348.

[13] Rie Johnson and Tong Zhang. "Learning nonlinear functions using regularized greedy forest". In: *IEEE transactions on pattern analysis and machine intelligence* 36.5 (2014), pp. 942–954.

[14] Alexey Natekin and Alois Knoll. "Gradient boosting machines, a tutorial". In: *Frontiers in neurorobotics* 7 (2013).

[15] Szilard Pafka. *Benchmarking Random Forest Implementations*. URL: http://datascience.la/benchmarking-random-forest-%20implementations/.

[16] Szilard Pafka. *Incomplete benchmark for .. machine learning libraries for classification*. URL: https://github.com/szilard/benchm-ml.

[17] Cloud Passage. *CloudPassage Announces Support for Docker*. URL: https://www.cloudpassage.com/company/press-%20releases/cloudpassage-announces-support-for-docker/.

[18] Raymund Peck. *What does H2O.ai (formerly 0xdata) do?* URL: https://www.quora.com/What-does-H2O-ai-formerly-0xdata-do.

[19] Simon J Pittman and Kerry A Brown. "Multi-scale approach for predicting fish species distributions across coral reef seascapes". In: *PloS one* 6.5 (2011), e20583.

[20] VB Profiles. *Docker Landscape*. URL: https://www.vbprofiles.com/markets/docker-%20landscape-56728f6e1493f72993003d24.

[21] CTI Reviews. *Introductory Statistics*. Cram101, 2016. ISBN: 9781497014794. URL: https://books.google.de/books?id=LT7aAwAAQBAJ.

[22] Joseph Rickert. *Diving into H20*. URL: http://blog.revolutionanalytics.com/2014/04/a-%20dive-into-h2o.html.

[23] RITA. *Get the data*. URL: http://stat-%20computing.org/dataexpo/2009/the-data.html.

[24] WP Engine (Staff). *H2O vs R – Winning KDDCup98 in 10 minutes with H2O*. URL: https://blog.h2o.ai/2014/12/winning-kdd/.

[25] Andrie de Vries. *How the MKL speeds up Revolution R Open*. URL: http://blog.revolutionanalytics.com/2014/10/revolution-r-open-mkl.html.

[26] Amy Wang. *Airlines Delay*. URL: https://github.com/h2oai/h2o-3/blob/master/h2o-docs/src/product/flow/packs/examples/Airlines_Delay.flow.

[27] Xu Wang. *Create A Nomad Cluster On GCP with Terraform*. URL: https://github.com/xuwang/gcp-nomad.

[28] Wikipedia. *Gradient boosting — Wikipedia, The Free Encyclopedia*. [Online; accessed 11-July-2017]. 2017. URL: https://en.wikipedia.org/w/index.php?title=Gradient_boosting&oldid=790302586.

[29] Wikipedia. *H2O (software) — Wikipedia, The Free Encyclopedia*. [Online; accessed 11-July-2017]. 2017. URL: https://en.wikipedia.org/w/index.php?title=H2O_(software)&oldid=769723012.

[30] Wikipedia. *R (programming language) — Wikipedia, The Free Encyclopedia*. [Online; accessed 11-July-2017]. 2017. URL: https://en.wikipedia.org/w/index.php?title=R_(programming_language)&oldid=790313533.

[31] Mengwen Xu et al. "Demand driven store site selection via multiple spatial-temporal data". In: *Proceedings of the 24th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems*. ACM. 2016, p. 40.