Classification and automatic recognition of objects using H2o package

Yenumula B Reddy

Department of Computer Science
Grambling State University
Grambling, LA 71245, USA
Email: ybreddy@gram.edu

ABSTRACT

Deep learning (DL) is a process that consists of a set of methods which classifies the raw data to meaningful information that fed into the machine. Deep Convolutional nets composed of various processing layers to learn and represent the data. It has multiple levels of abstraction to process images, video, speech and audio. H2o deep learning architecture has many features that include supervised training protocol, memory efficient Java implementation, adaptive learning, and with related CRAN packages. H2o uses supervised training protocol with a uniform adaptive option which is an optimization based on the size of the network. It can take clusters of computing nodes to train on the entire data set but automatically shuffling the training examples for each iteration locally. The framework supports regularization techniques to prevent overfitting. Further, H2o R has intuitive web interface using localhost and IP address. In this research the computations are performed in the H2o cluster and initiated by REST calls (in highly optimized Java code) from R. Since SPARK is available in R, H2o uses a single R session to communicate H2o Java cluster via REST calls. H2o runs inside the Spark executor JVM. Using these packages in R, we demonstrate the classification and automatic recognition of objects. The research extends the NOAA VIIRS Night fires data to detect the persistent fire activity at a given location around the globe. To perform the classification, we use the H2o deep learning package in R Language.

Keywords: deep Learning, convolutional neural networks, automatic object recognition, H2o Package, SPARK package, classification, and cluster.

1. INTRODUCTION

Deep learning (DL) is a class of machine learning technique uses a set of algorithms that attempt to learn on multiple levels where each level defines from lower levels. The procedure includes many layers of neural networks (supervised or unsupervised) for feature extraction. The process uses pattern recognition and pattern analysis. The method is to make sense of data such as text, sound, and images. The process is about modeling with deep architecture for signal and information technology. DL models consist of multiple layers of nonlinear information processing and method for unsupervised and supervised learning. The popularity of DL is increased general purpose graphical processing unit (GPGPU) chip processing abilities in complex and computational nonlinear functions to learn distributed and hierarchical feature representations.

Deep neural networks (DNN) categorized as unsupervised, supervised, and hybrid. The unsupervised learning does not use any task specific supervision information in the learning process. It generates meaningful samples by sampling from the networks. Examples include recurrent neural networks (used to predict the data sequence in the future using the previous data samples) and sum-product network (a directed acyclic graph with observed variables as leaves and with sum and product operations as internal nodes). The deep supervised learning (DSL) intended to directly provide pattern classification where target label data are always available. In deep supervised networks (DSN) the output of the discriminately learned neural network is treated as part of the observation variable. The Recent approach is the model with many layers using backpropagation learning. Hybrid deep networks use generative and discriminative model components. The generative models trained in unsupervised fashion can provide excellent initialization points in highly nonlinear parameter estimation problems or adequately provide a prior on the set of functions representable by the model.

Training DNN is a challenging task. DL aims at learning features and hierarchies from lower level composition to higher levels. A standard neural network consists of many simple, connected processors called neurons. Each input neuron produces a sequence of real-valued activations. Input neurons get values from sensors or user data. The neurons in hidden layers get activated through weights connected from previous active neurons. Some input neurons get activated from the influence of environment triggering actions. Learning (credit assignment) is the way to make the NN desired behavior through repeated computations. In deep learning, the system takes chains of computational stages, where each stage transforms the aggregate actions of the network. DL is about accurately assigning credit across many stages.

Deep Learning is the hottest subject in machine learning. The popular software for DL including Caffe, Cuda-convert, Deeplearning4j, Pylearn2, Theano, Torch, and other Deep Learning Tools. Theano is a Python library and useful for making algorithms from scratch. Caffe can be used for image classification (not for text or speech) and can process over 60M images per day with a single NVIDIA K40 GPU. The Torch is written in Lua (easy to integrate with C) and can provide an environment like MATLAB for ML algorithms. It is easy to start with and feature extractor trained on the ImageNet dataset with Torch7. CuDNN (Cuda-convnet) supports all the mainstream software such as Caffe, Torch, and Theano and is very easy to enable. DL is Java based, industry focused, commercially supported and designed to use in business environments.

All DL-tools are open source, need training, use CPU and GPU. The core languages are different for different tools. For example, Caffe and Cuda-convert use C++ and have Python bindings. Caffe can also bind with MATLAB. The core language for Theano is Python and Torch7 has Lua. Most of these packages are distributed and do not have pre-trained models (except Caffe). Table I shows the evaluation of DL toolkits.

Table I. Evaluation of Deep Learning Toolkits

Tool	Modeling Capability	Interfaces	Model Development	Performance	Architecture	Ecosystem	Cross-platform
Caffe	• Poor – for recurrent networks and language modeling	• need to define in protobuf for PyCaffe • Poor in interface choice	• C++ based • Best choice	• Average • Building block is layer	• Layerwise design • Extra function to support CPU/GPU	Caffe C++	yes
Tensor Flow	• Poor in modeling flexibility. • Fairly easy specifying a new network	• Supports Python and C++	• Supports C++ • No Windows Support	• Calls CuDNN-Best Performanc e	Well designed and dmodular	Python C++	yes
Theano	• Has implementation for most state-of-the-art networks. • Scan makes implementing RNN easy and efficient	• Python	• Has windows support • Low level interface and inefficienc y of Pyrthon	• Startup problem • Need to compile C/CUDA code to binary	• Hacky • Python Base	Python	Yes
Torch	• Rich set of RNN available • Excellent for Conv nets • Easy to define new layers	• LuaJIT	• Requires LuaJIT • Integration problem at API level	Calls CuDNN-Best Performance	• Well designed and dmodular	Libraries built for it are not as rich as ones built for Python.	not on Window s

DL tools in R language is still relatively new. They are extensible, and users can build blocks using simple math Legos in the core. The process is fast, scalable, open source machine learning and deep learning for smarter applications. The product uses dynamic pricing, insurance fraud detection, healthcare, telecommunications, and retail. H2o R can handle billions of data rows in memory, even with a small cluster. H2o platform interface to R, Python, Scala, Java, JSON, and Caffe. It runs on a stand-alone on Hadoop or within a Spark Cluster. Since we are planning to use H2o interfacing to R for classification of objects, it requires introducing the features of H2o.

2. H2O FEATURES – MACHINE LEARNING IN R LANGUAGE

H2o is an open source CRAN package, distributed, Java machine learning library software, and ease of use via the Web interface. The software was designed with distributed algorithms scale to big data and have an interface to Python, Scala, R, Spark, and Hadoop. The H2o is scale to big data, access data links and use all data without sampling. It is interactive, high-speed, accurate, faster in-memory and distributed. It has cutting-edge machine learning algorithms (linear regression, logistic regression, etc.), Nano-fast scoring engine and deep learning modules (multilayer feed-forward neural network, auto-encoder, anomaly detection, and deep learning features). It has K-means clustering, statistical analysis, dimension reduction (principal component analysis and generalized low-rank models), optimization and data munging (integrated R-environment, slice, and log transform).

H2o principal components are a cluster, frame, and distributed key-value store (a perfect peer-to-peer distributed hash table). In distributed key-value store each key has a home node and are picked pseudo-randomly. Keys can be cached anywhere, and key's home is solely responsible for breaking ties. There is no name node or central dictionary for keys. Data in H2o is memory bound not CPU bound. Linear access of data is compatible with C or FORTRAN. Data is highly compressed (about 2 to 4 times smaller than gzip). The table length is limited by memory and table width is <1K fast, <10K works, and < 100K slow.

H20 communications require Java virtual machine (JVM) and implements on remote procedure call (RPC). The failed connections may retrieve by reconfiguring the network. H2o supports Map/Reduce to write parallel code. Distributed fork/join and parallel map available within each node. GroupBy operator can handle millions of groups on billions of rows and runs Map/Reduce tasks on the team members. It has overloaded all the necessary data frame manipulation functions in R and Python.

Amazon Elastic Compute Cloud (EC2) cluster is an example of the high-performance computing (HPC). EC2 use cluster computing and optimized for GPU instance types (virtual machines) in R with H2o. HPC allows engineers and scientists to solve complex problems in engineering, science, and business problems. HPC-EC2 use applications that require high bandwidth, enhanced networking and great compute capabilities. Amazon web service (AWS) allows the customers to increase the speed of research by executing their applications with HPC facility in the cloud. The process reduces the cost by providing Cluster Compute or Cluster GPU servers' on-demand without significant capital investments. The customer has access to a high bandwidth network, full-bisection, for tightly-coupled, and IO-intensive workloads, which enables them to scale out across thousands of cores for throughput-oriented applications. The experiment of Eric LeDell [1] shows that H2o with 16 ECT nodes on Amazon has high throughput with numerical compared to numerical and categorical.

H2o uses the model of multi-layer, feedforward neural networks for predictive modeling. H2o uses purely supervised training protocol. The H2o's DL features include supervised training protocol for regression and classification tasks, automatic pre- and post-processing for categorical and numerical data, and deep autoencoders for unsupervised feature learning and anomaly detection.

The reference [2] provides H2o installation (steps to load H2o), R connecting to H2o cluster, and deep learning overview. The authors discussed with examples of DL model with R code, the performance of the trained model, training metrics, validation, cross-validation, and view predictions in R, Web interface, grid search, and random grid search. We tested the code on Windows 10 based Intel Core i7 with NVIDIA GEFORCE GTX 980M computer. Grid search, print out all prediction errors, restart training in R checkpoint model, save the model and retrieve the model was working correctly. Arno et al. [2] updated the document on September 16, 2016, helps many researchers working in deep learning using H2o in R language.

High-performance machine learning in R with H2o [3] published in ISM HPC workshop in 2015 is another important document. The author discussed building H2o on Hadoop and H2o vs. SparkR. Details were not provided to test the model. Arno and Viraj [4] presents the MNIST digit classification and Web interface. The work is close to Amir et al. [2]. MNIST digit classification and achieving world record performance is same as in [2], but Amo et al. discussed more explicitly.

3. REVIEW OF RECENT WORK

H2o is an open source parallel processing software for machine learning (ML) on Big Data (BD). The software has number of impressive statistical and ML algorithms to run on HDFS, S3, SQL, and NoSQL. ML with R and H2o [5] has been available through http://h2o.ai/recources. The documentation provides data preparation, models, manipulation, and running models in R. Bernd et al. [6] presented the ML in R package that facilitates researchers to apply ML algorithms. They introduced the ML package in R provides unified interface to more than 160 primary learners and includes meta-algorithm and model selection techniques to improve and extend functionality. The features include hyper parameter tuning, feature selection, and ensemble construction. In R, parallel high-performance computing is supported.

Machine learning with R [7] provides the tools to apply ML techniques to the research projects. The classification using Naïve Bayes, nearest neighbor, decision trees, neural networks, clustering with K-means, and evaluating model performance. The author discussed the classification prediction data in R and improving the model performance in the book during last few chapters. Michal et al. [8] introduced H2o with capabilities of Spark (Sparkling water) to provide ideal machine learning environment. The tutorial explains the H2o with Spark users ML algorithms of H2o can compete with Scala, R, or Python that use with H2o. Sparkling water can provide deep learning to build a model, make predictions, and use results again in Spark.

R language has high adaptability, and H2o R connects to Oxdata's [9] software (Apache 2.0 license). The document [10] 'Fast Scalable R with H2o' provides H2o launching from R, tools, and methods for data preparation, initialization, data manipulation, and running various models including K-means, regression analysis, and predictions. The authors explained step-by-step to the user to use the models in H2o R environment.

Se-Jeong Park et al [11] presented DL model for data analysis using R. The paper discusses the import and process data and DL example. In the initial phase, they tried to collect data for DL through Web scraping program with H2o. Experiments are underway. Gradient Boosted models [12] discusses the Airline data example. The document is primarily provides loading data, Web interface, Java models, and grid search for model comparison. The report provides some basics. Cliff et al. [13] used Airline data classification using H2o. They explained the predictions with a trial run. The code is same as in [12] except updating the previous document. The code example in [14] helps to lead H2o R package and start and local H2o cluster. The paper used the code to fit a basic logistic regression model. It generates performance metrics from a single model, making class predictions based on a probability cutoff, remove short-term objects from R and the H2O cluster, fit a gradient boosting machine binomial regression model, fit a random forest classifier, and fit a deep learning classifier.

Sara et al. [15] presented the open source tools and their role in machine learning with big data in the Hadoop ecosystem. The document discusses Hadoop ecosystems, data processing engines for Hadoop, fundamentals of MapReduce, H2o, overview and evaluation of ML toolkits. The authors presented Samoa (new version of Mahout) for Spark and MLlib for H2o and their wider selection of algorithms. They suggested the tools for social network and e-commerce. They also studied the comparison of major machine learning frameworks (Mahout, MLLIB, H2o, and SAMOA) and their scalability, speed, coverage, usability, and extensibility.

Chaitra et al. [16] presented H2o ML approach with D3 visualization. The authors used old traditional reports as data for proposed research and demonstrated product recommendation. The tutorial on R and high-performance computing (HPC) discusses the big data and parallel computing with various approaches including Spark, H2o, Dato's SFrame. A simple example was used to explain the process [17]. Drew [18] presented facilities of R, improving R performance, interfacing with other languages, parallelism, and distributed computing. The demo examples demonstrate with the appropriate code. The document helps to develop parallel applications.

The review of recent literature presented above reveals the practical work developed at different organizations and little published in journals or conferences. The research is in infant level for object recognition with H2o. The implementation in various directions using H2o, NVIDIA CUDA tools, and DL approaches. In this study, we will test unstructured data available on Web database and later demonstrate with NOAA VIIRS Night fires data if available to detect the persistent fire activity at a given location around the globe. Initially use R H2o and then use the DL approach.

4. STUDY OF RELATED PROBLEM TO CLASSIFY OBJEVTS

H2o package handles large datasets (billions of data) rows in memory even with a small cluster and works standalone mode on Hadoop. Selection of H2o DL model (H2o uses supervised training protocol) is to represent raw data and exhibit high performance on complex data like objects. Before we design our problem, we will work on two similar examples. The first example is train and test of MNIST data using H2o DL package and the second airline problem. The combination of these two prototypes helps to analyze and predict the persistent fire activity of NOAA VIIRS data at a given location around the globe. The following are the sequence of statements in R to load the train and test data files.

```
> library(h2o)
> h2o.init(nthreads = -1)
> train_file <- "https://h2o-public-test-data.s3.amazonaws.com/bigdata/laptop/mnist/train.csv.gz"
> test_file <- "https://h2o-public-test-data.s3.amazonaws.com/bigdata/laptop/mnist/test.csv.gz"
> train <- h2o.importFile(train_file)
> test <- h2o.importFile(test_file)
> summary(train, exact_quanties=TRUE)
>summary(test, exact_quanties=TRUE)
```

The example uses the MNIST academic data set that consists of 60,000 training images and 10,000 test images. Each image of size 28x28 (pixel greyscale image of a single hand-written image). We adjust the predictor column to 785. Train the model using above commands in R. Table IIa, and IIb shows the summary of training and testing. Column 785 illustrates the number of samples trained or tested for each digit (digit class).

Table IIa. Summary of training

```
> summary(train, exact_quanties=TRUE)
 C1            C2            C3            C4            C5            C6            C7            C8
 Min.   : 0   Min.   : 0   Min.   : 0   Min.   : 0   Min.   : 0   Min.   : 0   Min.   : 0   Min.   : 0
 1st Qu.:NaN  1st Qu.:NaN  1st Qu.:NaN  1st Qu.:NaN  1st Qu.:NaN  1st Qu.:NaN  1st Qu.:NaN  1st Qu.:NaN
 Median :NaN  Median :NaN  Median :NaN  Median :NaN  Median :NaN  Median :NaN  Median :NaN  Median :NaN
 Mean   : 0   Mean   : 0   Mean   : 0   Mean   : 0   Mean   : 0   Mean   : 0   Mean   : 0   Mean   : 0
 3rd Qu.:NaN  3rd Qu.:NaN  3rd Qu.:NaN  3rd Qu.:NaN  3rd Qu.:NaN  3rd Qu.:NaN  3rd Qu.:NaN  3rd Qu.:NaN
 Max.   : 0   Max.   : 0   Max.   : 0   Max.   : 0   Max.   : 0   Max.   : 0   Max.   : 0   Max.   : 0
 C9            C10           C11           C12           C13              C14              C15
 Min.   : 0   Min.   : 0   Min.   : 0   Min.   : 0   Min.   :  0.0000   Min.   :0.000e+00   Min.   :  0.0000
 1st Qu.:NaN  1st Qu.:NaN  1st Qu.:NaN  1st Qu.:NaN  1st Qu.:  0.0000   1st Qu.:0.000e+00   1st Qu.:  0.0000
 Median :NaN  Median :NaN  Median :NaN  Median :NaN  Median :  0.0000   Median :0.000e+00   Median :  0.0000
 Mean   : 0   Mean   : 0   Mean   : 0   Mean   : 0   Mean   :  0.0021   Mean   :7.833e-03   Mean   :  0.0036
 3rd Qu.:NaN  3rd Qu.:NaN  3rd Qu.:NaN  3rd Qu.:NaN  3rd Qu.:  0.0000   3rd Qu.:0.000e+00   3rd Qu.:  0.0000
 Max.   : 0   Max.   : 0   Max.   : 0   Max.   : 0   Max.   :116.0000   Max.   :2.540e+02   Max.   :216.0000

       ------------
       ------------
C775              C776              C777              C778              C779              C780
Min.   :  0.0000  Min.   :  0.00000  Min.   :  0.00000  Min.   :  0.00000  Min.   :  0.00000  Min.   : 0.000
1st Qu.:  0.0000  1st Qu.:  0.00000  1st Qu.:  0.00000  1st Qu.:  0.00000  1st Qu.:  0.00000  1st Qu.: 0.000
Median :  0.0000  Median :  0.00000  Median :  0.00000  Median :  0.00000  Median :  0.00000  Median : 0.000
Mean   :  0.2004  Mean   :  0.08887  Mean   :  0.04563  Mean   :  0.01928  Mean   :  0.01512  Mean   : 0.002
3rd Qu.:  0.0000  3rd Qu.:  0.00000  3rd Qu.:  0.00000  3rd Qu.:  0.00000  3rd Qu.:  0.00000  3rd Qu.: 0.000
Max.   :254.0000  Max.   :254.00000  Max.   :253.00000  Max.   :253.00000  Max.   :254.00000  Max.   :62.000
C781          C782          C783          C784          C785
Min.   : 0   Min.   : 0   Min.   : 0   Min.   : 0   1:6742
1st Qu.:NaN  1st Qu.:NaN  1st Qu.:NaN  1st Qu.:NaN  7:6265
Median :NaN  Median :NaN  Median :NaN  Median :NaN  3:6131
Mean   : 0   Mean   : 0   Mean   : 0   Mean   : 0   2:5958
3rd Qu.:NaN  3rd Qu.:NaN  3rd Qu.:NaN  3rd Qu.:NaN  9:5949
Max.   : 0   Max.   : 0   Max.   : 0   Max.   : 0   0:5923
```

Table IIb. Summary of testing

```
> summary(test, exact_quanties=TRUE)
       C1              C2              C3              C4              C5              C6              C7              C8
 Min.   : 0   Min.   : 0   Min.   : 0   Min.   : 0   Min.   : 0   Min.   : 0   Min.   : 0   Min.   : 0
 1st Qu.:NaN   1st Qu.:NaN   1st Qu.:NaN   1st Qu.:NaN   1st Qu.:NaN   1st Qu.:NaN   1st Qu.:NaN   1st Qu.:NaN
 Median :NaN   Median :NaN   Median :NaN   Median :NaN   Median :NaN   Median :NaN   Median :NaN   Median :NaN
 Mean   : 0   Mean   : 0   Mean   : 0   Mean   : 0   Mean   : 0   Mean   : 0   Mean   : 0   Mean   : 0
 3rd Qu.:NaN   3rd Qu.:NaN   3rd Qu.:NaN   3rd Qu.:NaN   3rd Qu.:NaN   3rd Qu.:NaN   3rd Qu.:NaN   3rd Qu.:NaN
 Max.   : 0   Max.   : 0   Max.   : 0   Max.   : 0   Max.   : 0   Max.   : 0   Max.   : 0   Max.   : 0
       C9             C10             C11             C12             C13             C14             C15             C16
 Min.   : 0   Min.   : 0   Min.   : 0   Min.   : 0   Min.   : 0   Min.   : 0   Min.   : 0   Min.   : 0
 1st Qu.:NaN   1st Qu.:NaN   1st Qu.:NaN   1st Qu.:NaN   1st Qu.:NaN   1st Qu.:NaN   1st Qu.:NaN   1st Qu.:NaN
 Median :NaN   Median :NaN   Median :NaN   Median :NaN   Median :NaN   Median :NaN   Median :NaN   Median :NaN
 Mean   : 0   Mean   : 0   Mean   : 0   Mean   : 0   Mean   : 0   Mean   : 0   Mean   : 0   Mean   : 0
 3rd Qu.:NaN   3rd Qu.:NaN   3rd Qu.:NaN   3rd Qu.:NaN   3rd Qu.:NaN   3rd Qu.:NaN   3rd Qu.:NaN   3rd Qu.:NaN
 Max.   : 0   Max.   : 0   Max.   : 0   Max.   : 0   Max.   : 0   Max.   : 0   Max.   : 0   Max.   : 0
       C17             C18             C19             C20             C21             C22             C23             C24
 Min.   : 0   Min.   : 0   Min.   : 0   Min.   : 0   Min.   : 0   Min.   : 0   Min.   : 0   Min.   : 0
 1st Qu.:NaN   1st Qu.:NaN   1st Qu.:NaN   1st Qu.:NaN   1st Qu.:NaN   1st Qu.:NaN   1st Qu.:NaN   1st Qu.:NaN
 Median :NaN   Median :NaN   Median :NaN   Median :NaN   Median :NaN   Median :NaN   Median :NaN   Median :NaN
 Mean   : 0   Mean   : 0   Mean   : 0   Mean   : 0   Mean   : 0   Mean   : 0   Mean   : 0   Mean   : 0
 3rd Qu.:NaN   3rd Qu.:NaN   3rd Qu.:NaN   3rd Qu.:NaN   3rd Qu.:NaN   3rd Qu.:NaN   3rd Qu.:NaN   3rd Qu.:NaN
 Max.   : 0   Max.   : 0   Max.   : 0   Max.   : 0   Max.   : 0   Max.   : 0   Max.   : 0   Max.   : 0
       C25             C26             C27             C28             C29             C30             C31             C32
 Min.   : 0   Min.   : 0   Min.   : 0   Min.   : 0   Min.   : 0   Min.   : 0   Min.   : 0   Min.   : 0
 1st Qu.:NaN   1st Qu.:NaN   1st Qu.:NaN   1st Qu.:NaN   1st Qu.:NaN   1st Qu.:NaN   1st Qu.:NaN   1st Qu.:NaN
 Median :NaN   Median :NaN   Median :NaN   Median :NaN   Median :NaN   Median :NaN   Median :NaN   Median :NaN
 Mean   : 0   Mean   : 0   Mean   : 0   Mean   : 0   Mean   : 0   Mean   : 0   Mean   : 0   Mean   : 0
 3rd Qu.:NaN   3rd Qu.:NaN   3rd Qu.:NaN   3rd Qu.:NaN   3rd Qu.:NaN   3rd Qu.:NaN   3rd Qu.:NaN   3rd Qu.:NaN
 Max.   : 0   Max.   : 0   Max.   : 0   Max.   : 0   Max.   : 0   Max.   : 0   Max.   : 0   Max.   : 0
       C33             C34               C35               C36               C37               C38
 Min.   : 0   Min.   : 0.0000   Min.   :  0.0000   Min.   :  0.0000   Min.   :  0.0000   Min.   :  0.0000
 1st Qu.:NaN   1st Qu.: 0.0000   1st Qu.:  0.0000   1st Qu.:  0.0000   1st Qu.:  0.0000   1st Qu.:  0.0000
 Median :NaN   Median : 0.0000   Median :  0.0000   Median :  0.0000   Median :  0.0000   Median :  0.0000
 Mean   : 0   Mean   : 0.0038   Mean   :  0.0236   Mean   :  0.0158   Mean   :  0.0403   Mean   :  0.0511
 3rd Qu.:NaN   3rd Qu.: 0.0000   3rd Qu.:  0.0000   3rd Qu.:  0.0000   3rd Qu.:  0.0000   3rd Qu.:  0.0000
 Max.   : 0   Max.   :38.0000   Max.   :236.0000   Max.   :158.0000   Max.   :233.0000   Max.   :198.0000
```

```
      C771              C772              C773              C774              C775              C776
 Min.   :  0.0000   Min.   :  0.0000   Min.   :  0.0000   Min.   :  0.0000   Min.   :  0.0000   Min.   :  0.0000
 1st Qu.:  0.0000   1st Qu.:  0.0000   1st Qu.:  0.0000   1st Qu.:  0.0000   1st Qu.:  0.0000   1st Qu.:  0.0000
 Median :  0.0000   Median :  0.0000   Median :  0.0000   Median :  0.0000   Median :  0.0000   Median :  0.0000
 Mean   :  0.5583   Mean   :  0.5746   Mean   :  0.4582   Mean   :  0.2741   Mean   :  0.1793   Mean   :  0.1636
 3rd Qu.:  0.0000   3rd Qu.:  0.0000   3rd Qu.:  0.0000   3rd Qu.:  0.0000   3rd Qu.:  0.0000   3rd Qu.:  0.0000
 Max.   :255.0000   Max.   :255.0000   Max.   :255.0000   Max.   :254.0000   Max.   :253.0000   Max.   :253.0000
      C777              C778            C779          C780          C781          C782          C783          C784
 Min.   :  0.0000   Min.   :0e+00   Min.   : 0   Min.   : 0   Min.   : 0   Min.   : 0   Min.   : 0   Min.   : 0
 1st Qu.:  0.0000   1st Qu.:0e+00   1st Qu.:NaN   1st Qu.:NaN   1st Qu.:NaN   1st Qu.:NaN   1st Qu.:NaN   1st Qu.:NaN
 Median :  0.0000   Median :0e+00   Median :NaN   Median :NaN   Median :NaN   Median :NaN   Median :NaN   Median :NaN
 Mean   :  0.0526   Mean   :6e-04   Mean   : 0   Mean   : 0   Mean   : 0   Mean   : 0   Mean   : 0   Mean   : 0
 3rd Qu.:  0.0000   3rd Qu.:0e+00   3rd Qu.:NaN   3rd Qu.:NaN   3rd Qu.:NaN   3rd Qu.:NaN   3rd Qu.:NaN   3rd Qu.:NaN
 Max.   :156.0000   Max.   :6e+00   Max.   : 0   Max.   : 0   Max.   : 0   Max.   : 0   Max.   : 0   Max.   : 0
      C785
 1:1135
 2:1032
 7:1028
 3:1010
 9:1009
 4: 982
```

The trial run image size is 28x28 = 784 pixels and column 785 is summary. The trial run uses 10 epochs and the following code works for H2o DL model.

```
> y <- "C785"
> x <- setdiff(names(train),y)
> train[,y] <- as.factor(train[,y])
```

```
> test[,y] <- as.factor(test[,y])
> model <- h2o.deeplearning(x=x,y=y, training_frame = train, validation_frame = test,
+ distribution = "multinomial", activation ="RectifierWithDropout", hidden = c(32,32,32),
+ input_dropout_ratio = 0.2, sparse = TRUE, l1 = 1e-5, epochs = 10)
  |=============================================================================| 100%
Warning message:
In .h2o.startModelJob(algo, params, h2oRestApiVersion) :
Dropping constant columns: [C86, C85, ----------, C84, C83].
>model
```

The sample part of the result is in Table III. The model has 5 layers, 25,418 weights, and 610,989 training samples. The Training set matrices hit-ratio and validation matrices hit ratios are in Table IV.

Table III. H2o DL model

```
> model
Model Details:
==============

H2OMultinomialModel: deeplearning
Model ID:  DeepLearning_model_R_1482621066782_1593
Status of Neuron Layers: predicting C785, 10-class classification, multinomial distribution,
  CrossEntropy loss, 25,418 weights/biases, 406.4 KB, 610,989 training samples, mini-batch size 1

  layer units             type dropout       l1       l2 mean_rate rate_rms momentum mean_weight weight_rms mean_bias bias_rms
1     1   717            Input 20.00 %
2     2    32 RectifierDropout 50.00 % 0.000010 0.000000  0.033898 0.186224 0.000000   -0.009344   0.065264  0.428685 0.195651
3     3    32 RectifierDropout 50.00 % 0.000010 0.000000  0.000347 0.000176 0.000000   -0.026750   0.196029  0.759296 0.330587
4     4    32 RectifierDropout 50.00 % 0.000010 0.000000  0.000590 0.000296 0.000000   -0.045724   0.211417  0.681243 0.372343
5     5    10          Softmax         0.000010 0.000000  0.002716 0.002334 0.000000   -0.465500   1.039416 -1.926533 0.754041


H2OMultinomialMetrics: deeplearning
** Reported on training data. **
** Metrics reported on temporary training frame with 10006 samples **
```

Table IV. Verification ratios and validation ratios are in

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

MSE: (Extract with `h2o.mse`) 0.09216103
RMSE: (Extract with `h2o.rmse`) 0.3035804
Logloss: (Extract with `h2o.logloss`) 0.3415591
Mean Per-Class Error: 0.0871256
Confusion Matrix: Extract with `h2o.confusionMatrix(<model>,train = TRUE)`)

Hit Ratio Table: Extract with `h2o.hit_ratio_table(<model>,train = TRUE)
=======================================================================
Top-10 Hit Ratios:
    k hit_ratio
1   1  0.912852
2   2  0.954028
3   3  0.970718
4   4  0.979412
5   5  0.988507
6   6  0.994203
7   7  0.996102
8   8  0.998501
9   9  0.999300
10 10  1.000000
```

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

Extract validation frame with `h2o.getFrame("RTMP_sid_9a79_2376")`
MSE: (Extract with `h2o.mse`) 0.09583542
RMSE: (Extract with `h2o.rmse`) 0.309573
Logloss: (Extract with `h2o.logloss`) 0.355917
Mean Per-Class Error: 0.09364633
Confusion Matrix: Extract with `h2o.confusionMatrix(<model>,valid = TRUE)`

Hit Ratio Table: Extract with `h2o.hit_ratio_table(<model>,valid = TRUE)`
=======================================================================
Top-10 Hit Ratios:
    k hit_ratio
1   1  0.906900
2   2  0.951800
3   3  0.971000
4   4  0.980700
5   5  0.987500
6   6  0.992600
7   7  0.995200
8   8  0.998500
9   9  0.999700
10 10  1.000000
```

The cross-validation can be performed with the following code. keep_cross_validation_predictions=TRUE, keyword saves the predicted values. The hit ratio in Table V shows the top 10 hit-ratio and cross-Validation closes to zero at 10^{th} hit. Cross-Validation Metrics Summary is provided in Table VI.

```
> model_cv <- h2o.deeplearning (x=x,y=y, training_frame=train, distribution = "multinomial",
+ activation = "RectifierWithDropout", hidden = c(32,32,32),
```

+ input_dropout_ratio = 0.2, keep_cross_validation_predictions=TRUE, sparse = TRUE, l1=1e-5, epochs=10,nfolds=5)

==========| 100%

Warning message:
In .h2o.startModelJob(algo, params, h2oRestApiVersion) :
Dropping constant columns: [C86, . . . , C84, C83].

Table V. Top 10 hit-ratio and cross-Validation

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

MSE: (Extract with `h2o.mse`) 0.1366648
RMSE: (Extract with `h2o.rmse`) 0.3696821
Logloss: (Extract with `h2o.logloss`) 0.4283022
Mean Per-Class Error: 0.09784974
Confusion Matrix: Extract with
      `h2o.confusionMatrix(<model>,train = TRUE)`)

Hit Ratio Table: Extract with `
==============================
 h2o.hit_ratio_table(<model>,train = TRUE)`
 =======================================
Top-10 Hit Ratios:
    k hit_ratio
1   1  0.903027
2   2  0.958981
3   3  0.975707
4   4  0.985165
5   5  0.990343
6   6  0.993927
7   7  0.996117
8   8  0.998407
9   9  0.999403
10 10  1.000000
```

```
Cross-Validation Set Metrics:
=====================
Extract cross-validation frame with
  `h2o.getFrame("RTMP_sid_9a79_2375")`
MSE: (Extract with `h2o.mse`) 0.1276199
RMSE: (Extract with `h2o.rmse`) 0.3572393
Logloss: (Extract with `h2o.logloss`) 0.4263426
Mean Per-Class Error: 0.1113672
Hit Ratio Table: Extract with
==============================
 `h2o.hit_ratio_table(<model>,xval = TRUE)`
 =======================================

Top-10 Hit Ratios:
    k hit_ratio
1   1  0.889833
2   2  0.954450
3   3  0.972850
4   4  0.982217
5   5  0.988500
6   6  0.992333
7   7  0.995050
8   8  0.997700
9   9  0.999200
10 10  1.000000
```

Table VI Cross-Validation Metrics Summary

```
Cross-Validation Metrics Summary:
                            mean           sd  cv_1_valid  cv_2_valid  cv_3_valid cv_4_valid  cv_5_valid
accuracy                0.88994753 0.0108982595  0.88604033  0.90155053  0.89938843 0.86135596   0.9014025
err                    0.110052444 0.0108982595  0.11395964  0.09844948 0.100611545 0.13864404   0.0985975
err_count                   1322.0     138.5655      1378.0      1181.0      1201.0     1683.0      1167.0
logloss                 0.42614755  0.023867084  0.39853942  0.43712854   0.4305506  0.4808368  0.38368243
max_per_class_error     0.31585976   0.10311232  0.36090225  0.16944689   0.2923617  0.5713066  0.18528138
mean_per_class_accuracy 0.88872665  0.011073175  0.88521814  0.90071607  0.89725715 0.85957354   0.9008684
mean_per_class_error    0.11127335  0.011073175  0.11478187 0.099283956  0.10274283 0.14042647   0.0991316
mse                     0.12753591  0.008955994 0.121007994  0.12571135   0.1282542 0.15041211 0.112293884
r2                       0.9847178 0.0010993186  0.98549634  0.98500985   0.9846194   0.981896   0.9865675
rmse                    0.35669574  0.012329829  0.34786204  0.35455796  0.35812595    0.38783   0.3351028
>
```

The world class performance of the model takes 100 minutes with 1 node with 64 epochs. In the current case we completed the model with 10 epochs on Intel core i7 with NVIDIA GEFORCE GTX 980 laptop in (start time 14:25:16 and end time 14:59:57) 34 minutes and 41 seconds. It has 5 layers and layer 1 has 717 units (nodes), layer 2 and 3 with 1024 units, layer 4 has 2018 and layer 5 has 10 units. Layer 1 is input, layers 2, 3, and 4 are dropouts and layer 5 is softmax (10 digits). The code is given below. Figure 1 and procedure to work is from [4].

```
> model <- h2o.deeplearning(x=x,y=y, training_frame=train, validation_frame=test,
+ activation="RectifierWithDropout",hidden=c(1024,1024,2048),epochs=10,
+ input_dropout_ratio=0.2,l1=1e-5,max_w2=10,train_samples_per_iteration=-1,
+ classification_stop=-1,stopping_rounds=0)
```

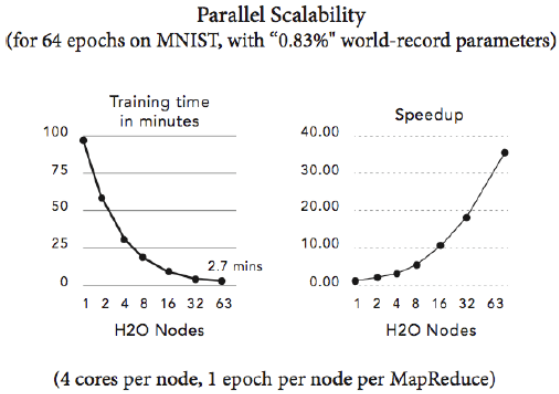

Figure 1. Parallel Scalability

Table VII for 10 epochs on MNIST with validation classification error 0.01990

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

H2OMultinomialModel: deeplearning
Model Key:  DeepLearning_model_R_1482621066782_1596
Status of Neuron Layers: predicting C785, 10-class classification, multinomial distribution, CrossEntropy loss,
3,904,522 weights/biases, 44.8 MB, 600,000 training samples, mini-batch size 1
  layer units             type dropout       l1       l2 mean_rate rate_rms momentum mean_weight weight_rms mean_bias bias_rms
1     1   717            Input 20.00 %
2     2  1024 RectifierDropout 50.00 % 0.000010 0.000000  0.183396 0.288053 0.000000    0.005645   0.046000  0.229504 0.078676
3     3  1024 RectifierDropout 50.00 % 0.000010 0.000000  0.006939 0.006107 0.000000   -0.007604   0.038643  0.958572 0.038793
4     4  2048 RectifierDropout 50.00 % 0.000010 0.000000  0.028045 0.027245 0.000000   -0.004581   0.030031  0.801948 0.078219
5     5    10          Softmax         0.000010 0.000000  0.013975 0.042138 0.000000   -0.052158   0.046882 -1.089825 0.112885

H2OMultinomialMetrics: deeplearning
** Reported on training data. **
** Metrics reported on temporary training frame with 10043 samples **

Scoring History:
            timestamp          duration training_speed   epochs iterations       samples training_rmse training_logloss
1 2016-12-29 14:25:16         0.000 sec                 0.00000          0      0.000000
2 2016-12-29 14:28:53  4 min 19.991 sec    277 obs/sec  1.00000          1  60000.000000       0.18567          0.13463
3 2016-12-29 14:36:17 11 min 44.659 sec    290 obs/sec  3.00000          3 180000.000000       0.14344          0.07713
4 2016-12-29 14:46:03 21 min 30.189 sec    308 obs/sec  6.00000          6 360000.000000       0.11490          0.04816
5 2016-12-29 14:53:06 28 min 32.980 sec    310 obs/sec  8.00000          8 480000.000000       0.10448          0.04043
6 2016-12-29 14:59:57 35 min 29.655 sec    312 obs/sec 10.00000         10 600000.000000       0.09413          0.03233

   training_classification_error validation_rmse validation_logloss validation_classification_error

                         0.04003         0.18792           0.13689                         0.04120
                         0.02539         0.15254           0.09425                         0.02720
                         0.01573         0.13902           0.07637                         0.02270
                         0.01454         0.13144           0.07212                         0.02040
                         0.01125         0.12877           0.06845                         0.01990
```

Example 2:

Initialize the local host and check the cluster information:

```
>h2o.init(ip="localhost",port=54321)
>h2o.clusterInfo()
```

```
R is connected to the H2O cluster:
    H2O cluster uptime:         5 days 15 hours
    H2O cluster version:        3.10.0.8
    H2O cluster version age:    2 months and 19 days
    H2O cluster name:           H2O_started_from_R_Dr._Reddy_jzw210
    H2O cluster total nodes:    1
    H2O cluster total memory:   6.64 GB
    H2O cluster total cores:    8
    H2O cluster allowed cores:  2
    H2O cluster healthy:        TRUE
    H2O Connection ip:          localhost
    H2O Connection port:        54321
    H2O Connection proxy:       NA
    R Version:                  R version 3.3.0 (2016-05-03)
```

Data manipulation can be done in three commands.

- as.data.frame() converts an h2o data frame into R data frame (recommended to take subsets)
- as.h2o() transfers data from R to the h2o instance
- str.H2oFrame() returns the elements of the new object to confirm success

The following statements import the data set and display the summary. From here, we can find the number of flights by airport, number of flights per month, and highest cancellation per month. Once we train the data, we can predict the delay or cancelation depending upon environmental conditions.

```
>airlinesURL="https://s3.amazonaws.com/h2o-airlines-unpacked/allyears2k.csv"
>airlines.hex=h2o.importFile(path=airlinesURL,destination_frame="airlines.hex")
> summary(airlines.hex)
```

To detect the persistent fire activity at a given location is same as anomalies at any given location. The anomalies of heart beats can be simulated to unusual activity in a given location data.

Importing train and test ECG data into h2o cluster:

```
> train_ecg <- h2o.importFile(path="http://h2o-public-test-data.s3.amazonaws.com/smalldata/anomaly/ecg_discord_train.csv",
+ header=FALSE, sep =",")
> test_ecg <- h2o.importFile(path="http://h2o-public-test-data.s3.amazonaws.com/smalldata/anomaly/ecg_discord_test.csv",
+ header=FALSE, sep =",")
> anomaly_model <- h2o.deeplearning(x=names(train_ecg), training_frame=train_ecg, activation="Tanh",
+ autoencoder = TRUE, hidden = c(50,50,50), sparse=TRUE, l1=1e-4,epochs=100)
```

Table VIII. Train deep autoencoder learning model

```
> anomaly_model
Model Details:
==============
H2OAutoEncoderModel: deeplearning
Model ID:  DeepLearning_model_R_1482621066782_1602
Status of Neuron Layers: auto-encoder, gaussian distribution, Quadratic loss, 26,360 weights/biases,
                336.6 KB, 2,000 training samples, mini-batch size 1
  layer units  type dropout       l1       l2 mean_rate rate_rms momentum mean_weight weight_rms mean_bias bias_rms
1     1   210 Input  0.00 %
2     2    50  Tanh  0.00 % 0.000100 0.000000  1.004224 0.000000 0.000000    0.000268   0.091320  0.000000 0.000000
3     3    50  Tanh  0.00 % 0.000100 0.000000  1.004224 0.000000 0.000000   -0.001911   0.138684  0.000000 0.000000
4     4    50  Tanh  0.00 % 0.000100 0.000000  1.004224 0.000000 0.000000   -0.006143   0.138744  0.000000 0.000000
5     5   210  Tanh         0.000100 0.000000  1.004224 0.000000 0.000000    0.000872   0.090417  0.000000 0.000000


H2OAutoEncoderMetrics: deeplearning
** Reported on training data. **

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

MSE: (Extract with `h2o.mse`) 0.1258968
RMSE: (Extract with `h2o.rmse`) 0.3548194
```

5. FRAMING THE PROPOSED PROBLEM

The unusual activity at a given place on the earth is the main focus of the problem. The particular activity may be fire, population variation, in lighting variation, weather, and environmental changes. Brightness lights on earth at night digitally recorded by DMSP satellite [19]. The report helped for future researchers to study the light effects and analysis of images. Christopher presented the Night-time fire or light and how they relate to population globally [20]. The results present for minimum values for population detection may not apply in the continuous growing places. Huang et al. [21] did meta-analysis DMSP/OLS nighttime light images, literature review and summarized standard methods and difficulties. Christopher et al. [22] presented global satellite map of nighttime lights linked to population changes and economic development. NOAA technical report NESDIS 142 [23] provide the guide for visible, infrared imaging radiometer suite (VIIRS).

We We downloaded the climate data for 2016 of Baton Rouge, LA, from NOAA (National Centers for Environmental information). The data include many parameters including wind, snow, and temperature. The following settings are used to calculate the summary.

BatonRouge.hex=h2o.importFile(path="summary.csv",destination_frame=" BatonRouge.hex ")
summary(BatonRouge.hex)

```
> summary(BatonRouge.hex)
 STATION                 STATION_NAME                        ELEVATION       LATITUDE        LONGITUDE        DATE                MDPR
 GHCND:USC00160548:364  BATON ROUGE CONCORD LA US     :364  Min.   : 4.60   Min.   :30.32   Min.   :-91.27   Min.   :20160101   Min.   :-9999.00
 GHCND:USC00160558:364  BATON ROUGE SHERWOOD LA US     :364  1st Qu.:10.68   1st Qu.:30.40   1st Qu.:-91.16   1st Qu.:20160325   1st Qu.:-9999.00
 GHCND:US1LAEB0023:363  BROWNFIELDS 4.0 E LA US        :363  Median :16.80   Median :30.44   Median :-91.13   Median :20160617   Median :-9999.00
 GHCND:USW00013970:362  BATON ROUGE RYAN AIRPORT LA US:362  Mean   :16.37   Mean   :30.45   Mean   :-91.10   Mean   :20160639   Mean   :-9973.85
 GHCND:USC00160555:359  BATON ROUGE LA US              :359  3rd Qu.:21.89   3rd Qu.:30.52   3rd Qu.:-91.05   3rd Qu.:20160920   3rd Qu.:-9999.00
 GHCND:US1LAEB0002:358  BATON ROUGE 2.7 SW LA US       :358  Max.   :32.90   Max.   :30.61   Max.   :-90.91   Max.   :20161229   Max.   :    9.41

 DAPR            PRCP               SNWD            SNOW            TAVG
 Min.   :-9999   Min.   :-9999.000   Min.   :-9999   Min.   :-9999   Min.   :-9999
 1st Qu.:-9999   1st Qu.:   -1.575   1st Qu.:-9999   1st Qu.:-9999   1st Qu.:  NaN
 Median :-9999   Median :   -1.575   Median :-9999   Median :-9999   Median :  NaN
 Mean   :-9974   Mean   : -454.703   Mean   :-9567   Mean   :-6003   Mean   :-9999
 3rd Qu.:-9999   3rd Qu.:   -1.575   3rd Qu.:-9999   3rd Qu.:   -9   3rd Qu.:  NaN
 Max.   :    6   Max.   :   18.460   Max.   :    0   Max.   :    0   Max.   :-9999

 TMAX            TMIN            TOBS            AWND            WDF2            WDF5            WSF2
 Min.   :-9999   Min.   :-9999   Min.   :-9999   Min.   :-9999.0   Min.   :-9999   Min.   :-9999   Min.   :-9999.0
 1st Qu.:-9999   1st Qu.:-9999   1st Qu.:-9999   1st Qu.:-9999.0   1st Qu.:-9999   1st Qu.:-9999   1st Qu.:-9999.0
 Median :-9999   Median :-9999   Median :-9999   Median :-9999.0   Median :-9999   Median :-9999   Median :-9999.0
 Mean   :-8709   Mean   :-8711   Mean   :-9147   Mean   :-9565.4   Mean   :-9559   Mean   :-9561   Mean   :-9564.9
 3rd Qu.:-9999   3rd Qu.:-9999   3rd Qu.:-9999   3rd Qu.:-9999.0   3rd Qu.:-9999   3rd Qu.:-9999   3rd Qu.:-9999.0
 Max.   :   98   Max.   :   82   Max.   :   83   Max.   :   15.9   Max.   :  360   Max.   :  360   Max.   :   33.1

WSF5            WT01            WT02            WT08            WT03
Min.   :-9999   Min.   :-9999   Min.   :-9999   Min.   :-9999   Min.   :-9999
1st Qu.:-9999   1st Qu.:-9999   1st Qu.:-9999   1st Qu.:-9999   1st Qu.:-9999
Median :-9999   Median :-9999   Median :-9999   Median :-9999   Median :-9999
Mean   :-9567   Mean   :-9766   Mean   :-9961   Mean   :-9949   Mean   :-9786
3rd Qu.:-9999   3rd Qu.:-9999   3rd Qu.:-9999   3rd Qu.:-9999   3rd Qu.:-9999
Max.   :   45   Max.   :    1   Max.   :    1   Max.   :    1   Max.   :    1

Warning message:
In summary.H2OFrame (BatonRouge.hex) :
  Approximated quantiles computed! If you are interested in exact quantiles, please pass the `exact_quantiles=TRUE` parameter.
>
```

The NOAA data temperatures for one year (Figure 2) shows the maximum temperature observed are days 169 to 274 of the year and minimum is first 60 days and last 30days of the year (maximum and verify from .

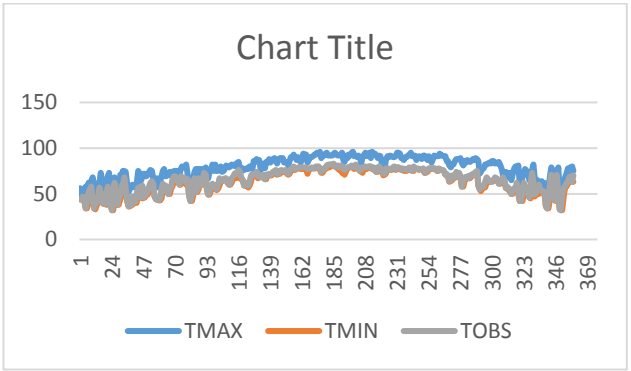

Figure 2. Temperature Variance

we demonstrate the classification and automatic recognition of objects. Further, we use the h2o deep learning package in R Language to classify the NOAA VIIRS Night fires data to detect the persistent fire activity at a given location around the globe.

6. DISCUSSION OF RESULTS

7. CONCLUSIONS AND FUTURE RESERACH

ACKNOWLEDGEMENTS

The research work was supported by the AFRL Collaboration Program – Sensors Research, Air Force Contract FA8650-13-C-5800, through subcontract number GRAM 13-S7700-02-C2. The author wishes to express appreciation to Dr. Connie Walton, Director of Sponsored Programs Grambling State University for her continuous support in research.

REFERENCES

[1] E. LeDell., "High Performance Machine Learning in R with H2o", ISM HPC on R workshop, Tokyo, Japan, 2015.

[2] Amir Arno Candel, Viraj Parmar, Erin Ledell, and Anisha Arora., "Deep Learning with H2o", http://h2o-release.s3.amazonaws.com/h2o/master/3420/docs-website/h2o-docs/booklets/DeepLearning_Vignette.pdf, 2016

[3] Erin LeDell., "High Performance Machine Learning in R with H2o", October 2015, Tokyo, Japan, http://www.stat.berkeley.edu/~ledell/docs/h2o_hpccon_oct2015.pdf

[4] Arno Candel and Viraj Parmar., "Deep Learning with H2o", http://h2o.ai/resources/, February 2015

[5] Spencer Aiello, Eric Eckstrand, Anqi Fu, Mark Landry, and Patrick Aboyoun (edited by Jessica Lanford), "Machine Learning with R and H2o", http://docs.h2o.ai/h2o/latest-stable/h2o-docs/booklets/RBooklet.pdf,

December 2016.
[6] Bernd Bischl, etc., “mlr: Machine Learning in R”, Jr. of Machine Learning Research, 17, 2016, pp. 1-5.
[7] Brett Lantz., “Machine Learning with R”, 2nd edition, Packt Publishing Ltd., 2015
[8] Michal Malohlava, Alex Tellez, and Jessica Lanford., “Machine Learning with Sparkling Water: H2o + Spark”, http://h2o-release.s3.amazonaws.com/h2o/master/3293/docs-website/h2o-docs/booklets/SparklingWaterVignette.pdf
[9] http://docs.h2o.ai/h2o/latest-stable/index.html
[10] Patrick Aboyoun, Spencer Aiello, Anqi Fu and Jessica Lanford., “Fast Scalable R with H2o”, published by H2o.ai, Inc., 2015.
[11] Se-Jeong Park, Kook-Hyun Choi, Jeawon Park, and Jong-Bae Kim., “A Study of Spatial Analysis Using R-Based Deep Learning”, Int. J. Software Engineering and its Applications, Vol. 10, No. 5 (2016), pp. 87-94.
[12] Cliff Click, Jessica Lanford, Michal Malohlava, Hank Roark, and Viraj Parmar., “Gradient Boosted models with H2o’s R Package”, http://h2o.ai/resources/, Sept 2015.
[13] Cliff Click, Michal Malohlava, Arno Candel, Hank Roark, and Viraj Parmar (edited by Jessica Lanford)., “Gradient Boosting Machine with H2o”, http://h2o.ai/resources/, Oct 2016.
[14] Joel Horwitz, Classification using Generalized Linear Models, Gradient Boosting Machines, Random Forests and Deep Learning in H2O”, http://h2o.ai/resources/, 2014
[15] Sara Landset, Tagi M. Khoshgoftaar, Aaron N. Richter and Tawfiq Hasanin., “A survey of open source tools for machine learning with big data in the Hadoop ecosystem”, J. of Big Data, a Springer open Journal, 2015, pp. 2-24.
[16] Chaitra D.B., and Bindiya, M.K., “Approaching Machine Learning Using H2o and Inspecting it on Apparel Industry Using D3 Visualization”, Int. J. of Advanced Networking & Applications (IJANA), ICICN 2016, CSE, RRCE, pp. 526-531.
[17] Dirk Eddelbuettel., “R and High-Performance Computing”, Creative Commons Attribution-ShareAlike 4.0 International License, October 2015.
[18] Drew Schmidt., “High Performance Computing with R”, nimbios.org/tutorials/TT_RforHPC, February 2015.
[19] Thomas A Croft., “The Brightness of Lights on Earth at Night, Digitally Recorded by DMSP Satellite”, Prepared by U.S. Geological Survey, August 1979.
[20] Christopher N. H. Doll., “Population Detection Profiles of DMSP-OLS night-time imagery by region of the world”, Proceedings of the 30th Asia-Pacific Advanced Network Meeting, August 2010, Hanoi, Viatnam.
[21] Qingxu Huang, Xi Yang, Bin Gao, Yang Yang, and Yuanyuan Zhao., “Application of DMSP/OLS Nighttime Light Images: A Meta-Analysis and a Systematic Literature Review”, Remote Sensing, 2014, pp. 6844-6866.
[22] Christopher D. Elvidge, Feng-Chi Hsu, Kimberly E. Baugh, and Tilottama Ghosh., “National Trends in Satellite Observed Lighting: 1992-2012”, Chapter in “Global Urban Monitoring and Assessment Through Earth Observation” Editor Qihao Weng. CRC Press. 2013.
[23] Changyong Cao, et al. “Visible Infrared Imaging Radiometer Suite (VIIRS) Sensor Data Record (SDR) User’s Guide”, NOAA Technical Report, NESDIS 142, September 2013.