PREDICTIVE TOOLS EVALUATION



BY

Kinshuk Srivastava

Advanced Analytics

**SUMMARY:**

This document aims at minimal evaluation for scalability, speed and accuracy of commonly used implementations of a few machine learning algorithms.

The evaluation is based on the binary classification with numeric and categorical variables with no missing data.

The datasets are varied as:

* 10K
* 100K
* 1M
* 10M

The algorithms evaluated are

* linear
* random forest
* boosting

in various common open source implementations like

* R packages
* Python Scikit-Learn
* H2O

We are focusing on which algorithms/implementations can be used to train relatively accurate binary classifiers for data with millions of observations and thousands of features processed on commodity hardware.

**DATA:**

Training datasets of sizes 10K, 100K, 1M, 10M are generated and cleaned in RStudio.

**SETUP:**

For R package and Python Scikit-Learn

Processor Name: Intel Core i7

Processor Speed: 2.5 GHz

Number of Processors: 1

Total Number of Cores: 4

L2 Cache (per Core): 256 KB

L3 Cache: 6 MB

Memory: 16 GB

For H2O

H2O cluster version: 3.10.3.4

H2O cluster version age: 1 month and 13 days

H2O cluster name: H2O\_24919

H2O cluster total nodes: 4

H2O cluster total memory: 9.26 GB

H2O cluster total cores: 32

H2O cluster allowed cores: 32

H2O cluster healthy: TRUE

H2O Connection ip: datanoded01.dev.bigdata.jcpcloud2.net

H2O Connection port: 54321

H2O Connection proxy: NA

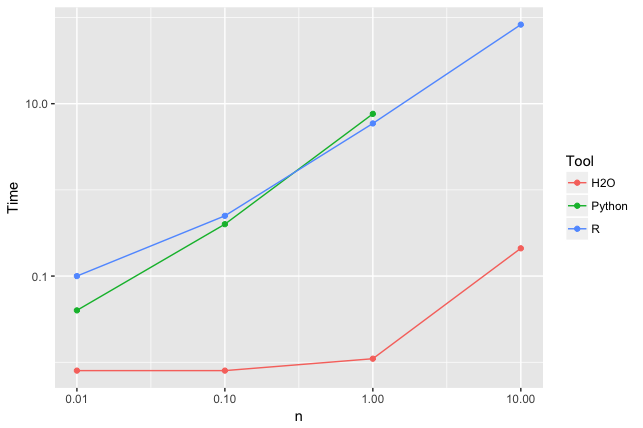
**The Linear Models:**

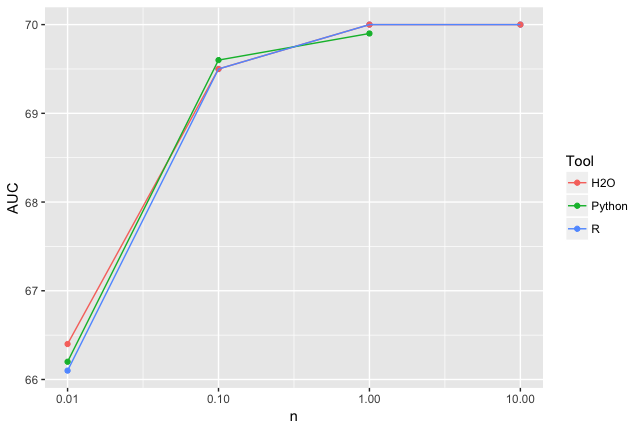
Linear Models are analyzed just to get some sort of baseline. These models are not very accurate and easy to predict.

The R glm function (basic logistic Regression) has been used. For Python/Scikit-Learn library logistic regression has been used. For H2O h2o.glm function has been used.

|  |  |  |  |
| --- | --- | --- | --- |
| **Tool** | **n(million)** | **Time(secs)** | **AUC** |
| R | .01 | 0.1 | 66.1 |
| R | .1 | 0.5 | 69.5 |
| R | 1 | 5.9 | 70 |
| R | 10 | 82.7 | 70 |
| Python | .01 | 0.04 | 66.2 |
| Python | .1 | 0.4 | 69.6 |
| Python | 1 | 7.62 | 69.9 |
| Python | 10 | NA | NA |
| H2O | .01 | .008 | 66.4 |
| H2O | .1 | .008 | 69.5 |
| H2O | 1 | .011 | 70 |
| H2O | 10 | .125 | 70 |

Python crashes for 10M rows.



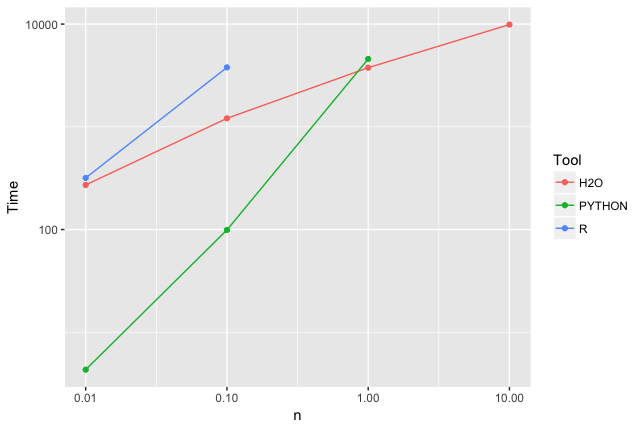


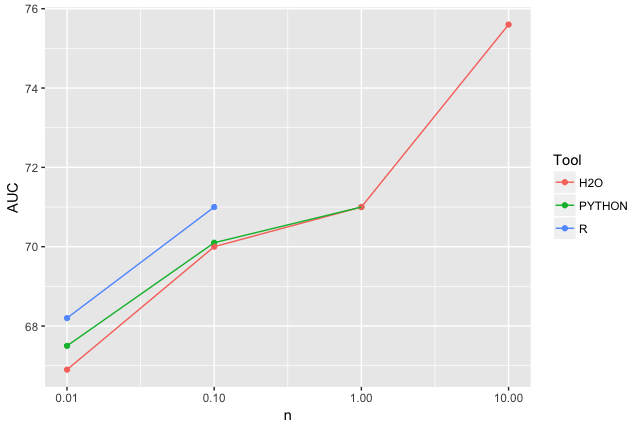
The main conclusion here is that it is easy to train linear models even on 10M rows in any of the above mentioned tools. H2O seems efficient from the above graphs but again the difference is the capacity. The fair test will be to give all the tools same capacity and even more larger datasets to perform on.

**Non Linear (Random Forest):**

Random forests have several commonly known implementations in R packages, Python Scikit-Learn and H2O. We are interested in which tools can deal with 10 million observations and train a random forest in a reasonable time (i.e. a few hours at most). Therefore, we are analyzing the scalability, speed and accuracy of various random forest implementations.

|  |  |  |  |
| --- | --- | --- | --- |
| **Tool** | **n(million)** | **Time(secs)** | **AUC** |
| R | .01 | 317.8 | 68.2 |
| R | .1 | 3789.6 | 71 |
| R | 1 | NA | NA |
| R | 10 | NA | NA |
| Python | .01 | 4.32 | 67.5 |
| Python | .1 | 99 | 70.1 |
| Python | 1 | 4578 | 71 |
| Python | 10 | NA | NA |
| H2O | .01 | 271.6 | 66.9 |
| H2O | .1 | 1209 | 70 |
| H2O | 1 | 3765 | 71 |
| H2O | 10 | 9896.89 | 75.6 |



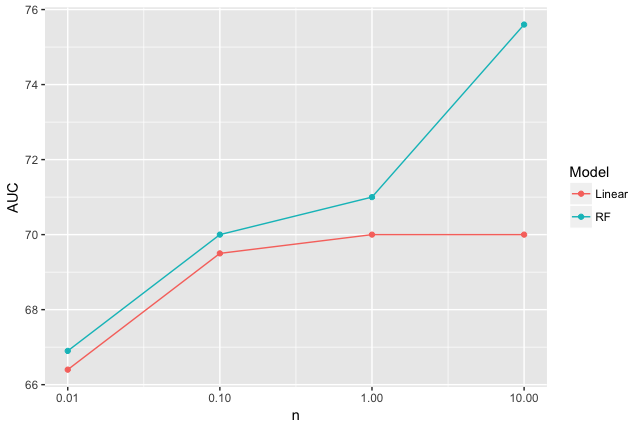


The R implementation (random Forest package) is slow and inefficient in memory use. It cannot cope by default with a large number of categories, therefore with larger dataset it crashes.

The Python (Scikit-Learn) implementation is faster, more memory efficient. Since I ran it on single machine it crashed for 10M rows.

The H2O implementation is fast, memory efficient and produced more accurate result than R/Python packages.

**H2O Linear VS Non Linear:**

****

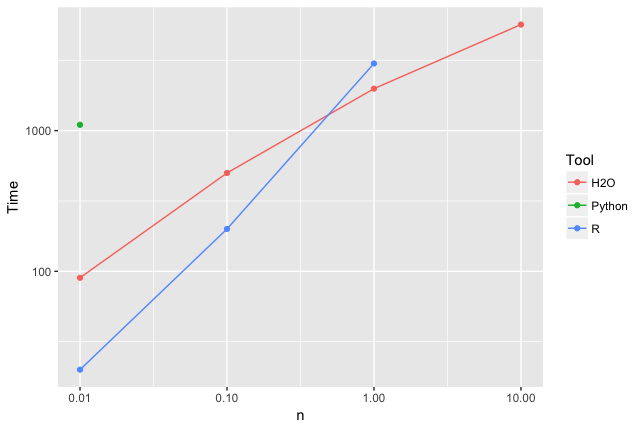
Random Forest model is more accurate than the linear one for any size. The random forest model on 1% of the data (100K records) beats the linear model on all the data (10M records).

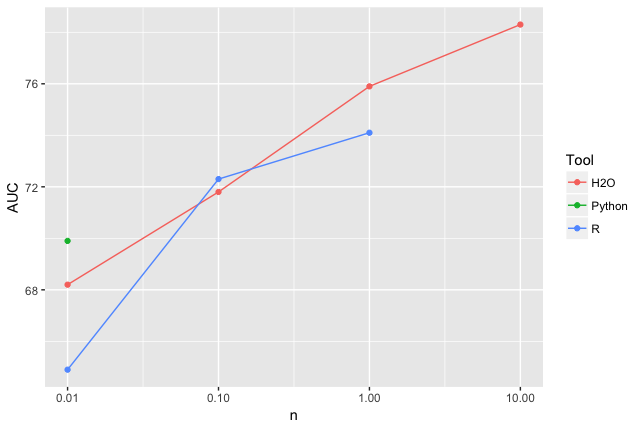
**Boosting (Gradient Boosted Trees/Gradient Boosting Machines):**

Boosting models are more complex in relationship and accurate in comparison to the random forest model. But if not trained properly can result in overfitting thus decreasing the accuracy.

|  |  |  |  |
| --- | --- | --- | --- |
| **Tool** | **n(million)** | **Time(secs)** | **AUC** |
| R | .01 | 20 | 64.9 |
| R | .1 | 200 | 72.3 |
| R | 1 | 3000 | 74.1 |
| R | 10 | NA | NA |
| Python | .01 | 1100 | 69.9 |
| Python | .1 | NA | NA |
| Python | 1 | NA | NA |
| Python | 10 | NA | NA |
| H2O | .01 | 90 | 68.2 |
| H2O | .1 | 500 | 71.8 |
| H2O | 1 | 1987 | 75.9 |
| H2O | 10 | 5673 | 78.3 |

Hard to implement, requires more tuning of the parameters but gives more accuracy.





**Conclusion**

* R implementation is easy and have rich packages to implement parameterized, non-parameterized, probabilistic models. But requires huge memory to run non-linear algorithms.
* Python Scikit-Learn implementation is easy and have rich packages to implement parameterized, non-parameterized, probabilistic models. Able to work on large datasets and is memory efficient.
* H2O implementation is fast and efficient but has a learning curve on data wrangling side because of the change in coding convention.
* H2O machine learning algorithms coding convention is same for both R and Python.
* H2O lacks on various packages which are present in R/Python for doing probabilistic modelling like Association rules.