

## BENCHMARKING RANDOM FOREST IMPLEMENTATIONS

MAY 19, 2015 EDUCATION SZILARD PAFKA 15 COMMENTS 2


I currently have the need for machine learning tools that can deal with observations of the order of 10 millions in the context of binary classification. That kind of data is a few GBs in size and it fits comfortably nowadays in the RAM of a decent single machine. It is a trivial task for linear models, as there are plenty of open source tools that can train a logistic regression with this amount of data on a single machine in a few seconds, even while using only 1 processor core (many of these tools are single-threaded). Linear models are also the **gold standard** of large-scale machine learning that can run on clusters, processing very large distributed datasets.

However, many problems would profit from modeling non-linearities. The state of the art in predictive accuracy are machine learning algorithms such as random forests, boosting, neural networks (and more recently deep learning) and (non-linear) support vector machines. In this post we'll take a look at **random forests**, perhaps the easiest to train of the above (as there are few "knobs" to set) and often one of the most accurate algorithms. In fact, they are regularly used as a starting point in Kaggle competitions. I'm planning to look at the other methods as well, so more posts will follow.

Random forests have several commonly known implementations in R packages, Python scikit-learn, Weka, H2O, Spark MLlib, Mahout, Revo ScaleR, among others. For the purposes of this post, I am 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, I'm analyzing the scalability, speed and accuracy of various random forest implementations.

In terms of scalability, I'm studying if the algorithms are able to complete (in decent time) for the given data sizes with given memory (RAM) constraints. Note that the largest dataset in this study is less than 1GB, so scaling out to multiple machines should not be necessary even if the tool can run in a distributed mode, therefore it is not the focus in this study. Actually, some machine learning algorithms perform relatively poorly in the multi-node setting, where communication is over the network rather than via updating shared memory. While we are certainly living in a "big data" craze, **more and more voices** are recognizing the value of choosing simpler solutions and not just using distributed computing

### RECENT POSTS




**CATBOOST: DISTRIBUTED TRAINING, UNCERTAINTY ESTIMATION AND OTHER NEWS**  
 MARCH 10, 2021  
 MACHINE LEARNING / DATA SCIENCE  
 0



**RECENT DEVELOPMENTS IN LIGHTGBM**  
 DECEMBER 15, 2020  
 MACHINE LEARNING / DATA SCIENCE  
 0



**2 TALKS: GBM BENCHMARK IN-DEPTH | THE STATE OF XGBOOST**  
 OCTOBER 15, 2020  
 MACHINE LEARNING / DATA SCIENCE  
 0



**SATRDAY LA - R CONFERENCE ANNOUNCEMENT**  
 JANUARY 28, 2019  
 EDUCATION  
 2

### CATEGORIES

Categories

### EXTERNAL LINKS

- LA R Meetup
- LA Machine Learning / Data Science Meetup
- DataVis LA Meetup
- useR! 2014 Conference
- R-bloggers

unnecessarily because of the hype. — Speed (in the single node setting) is determined by the computational complexity of the algorithm, but also if the implementation can use multiple processor cores or not. Accuracy is measured by the area under the ROC curve (AUC).

Training datasets of sizes 10K, 100K, 1M, 10M are generated from the well-known [airline dataset](#), using data from years 2005 and 2006. A test set of size 100K is generated from the same dataset using year 2007. The task is to predict whether a flight will be delayed by more than 15 minutes. The tests have been carried out on a Amazon EC2 c3.8xlarge instance (32 cores, 60GB RAM). For some of the models that ran out of memory with the larger data sizes a r3.8xlarge instance (32 cores, 250GB RAM) has been used occasionally.

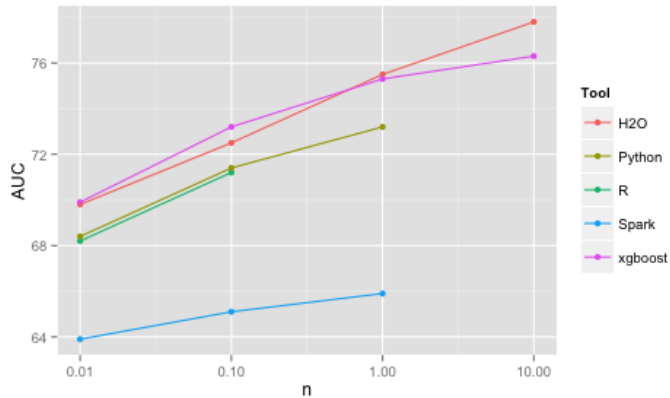
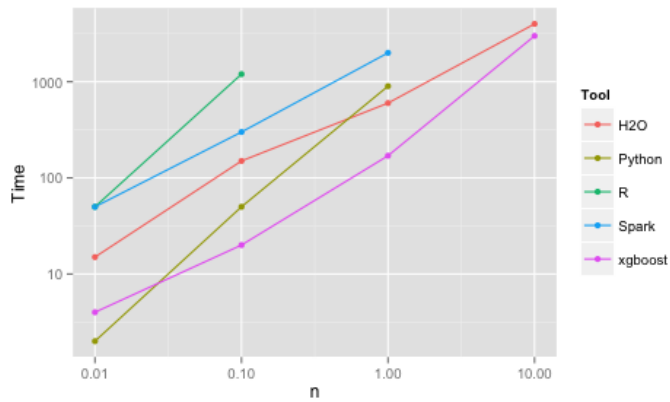
The models have been trained with default values for the (hyper-)parameters. For each algo/tool and each size  $n$  we observe the following: training time, maximum memory usage during training, CPU usage on the cores, and AUC as a measure for predictive accuracy. Times to read the data, pre-process the data, score the test data are also observed but not reported (e.g. they were not the bottleneck).

More details on this setup and the entire study, along with all the code used to get the results, can be found [at this Github repository](#). I'd like to emphasize this is not aimed to be a comprehensive benchmark. It started from my need at work to find a tool that can deal with datasets of similar size and structure. A more comprehensive benchmark would study different datasets of different structure (e.g. dense, sparse, etc.), how results are changing as a function of hyper parameter values, the scaling out to multiple nodes for the distributed systems, larger datasets, and the like.

I also want to emphasize the importance of testing the tools on *your* data: your specific data structure, data size, etc. It is very common lately (especially in a startup environment) to go with the hype and use a tool primarily because of its social media prowess without trying out less flashy alternatives which may better fit your needs. I have seen too many people falling into the “big data” trap and using these “big data” tools on data sets that don't really require such an approach (but this way adding extra costs in resources needed, efficiency or usability). Those who may be falling into this trap are often overheard saying, “We are building a Hadoop cluster. Our data is about 100 GB total. Our largest table is a few GBs.” Therefore, please don't stop your inquiry after reading a few blog posts on the topic or even *this* benchmark. Instead, think about your tasks and try out several tools to find out what works for your problem.

In our experiments, random forests with 500 trees have been trained in each tool with default hyper-parameter values. The training times and AUC as a function of the dataset size are plotted in the figures below (with more details available [on Github](#)).





The R implementation (randomForest package) is slow and inefficient in memory use. It cannot cope by default with a large number of categories, therefore the data had to be one-hot encoded. The implementation uses 1 processor core, but with 2 lines of extra code it is easy to build the trees in parallel using all the cores and combine them at the end. However, it runs out of memory already for  $n = 1M$ . I have to emphasize this has nothing to do with R per se, as it is instead the particular (C and Fortran) RF implementation used by the randomForest package that is inefficient. (Note that I still stand by my argument that R is the best data science platform especially when it comes to data munging and visualization.)

The Python (scikit-learn) implementation is faster, more memory efficient and uses all the cores. Variables needed to be one-hot encoded (which is more involved than for R) and for  $n = 10M$  doing this exhausted all the memory. Even if using a larger machine with 250GB of memory (and 140GB free for RF after transforming all the data) the Python implementation runs out of memory and crashes for this larger size. The algo finished successfully though when run on the larger box with simple integer encoding (which for some datasets/cases might be actually a good approximation/choice).

The H2O implementation is fast, memory efficient and uses all cores. It deals with categorical variables automatically. It is also more accurate than R/Python, which may be because of dealing properly with the categorical variables, i.e. internally in the algo rather than working from a previously 1-hot encoded dataset (where the link between the dummies belonging to the same original variable is lost).

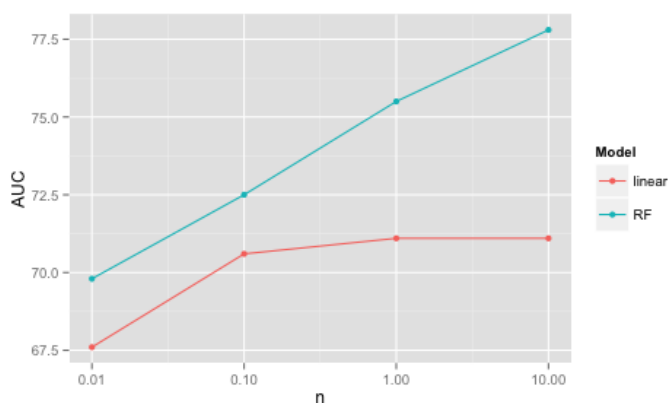
Spark (MLlib) implementation is somewhat slower, provides the lowest accuracy and it crashes already at  $n = 1M$  due to inefficient memory handling. With 250G of RAM it finishes for  $n = 1M$ , but runs out of memory for  $n = 10M$ . However, as Spark can run on a cluster one can throw in even more RAM by using more nodes. Alternatively, on a single machine, it is possible to train random forests with a smaller number of trees (but then accuracy decreases). I also tried to provide the categorical variables encoded simply as integers and passing the categoricalFeaturesInfo parameter, but that made training much slower. A convenience issue, reading the data is more than one line of code and Spark does not provide a one-hot encoder for the categorical data (therefore I used R for that). Note again the low prediction accuracy vs the other methods. One can improve a bit by increasing the maximum depth of trees (but only to Spark's limit of 30), but then training slows down further and AUC is still lower than with the other methods. Finding the reason for the lower AUC would need more investigation (the reason might

be that predict for Spark decision trees returns 0/1 and not probability scores therefore the random forest prediction is based on voting not probability averaging, or different stopping criteria, or just an algorithm that uses some approximations that hurts accuracy). **Update:** See some improvements in the comments [here](#), [here](#) and [here](#).

I also tried xgboost, a popular library for boosting which is capable to build random forests as well. It is fast, memory efficient and of high accuracy. Note the different shapes of the AUC and runtime vs dataset sizes for H2O and xgboost, however.

In addition to the above, several other random forest implementations have been tested (Weka, Revo ScaleR, Rborist R package, Mahout) but all of them proved slow and/or unable to scale to the larger sizes. While not the focus of this study, there are signs that running the distributed random forests implementations (e.g. H2O) on multiple nodes does not provide the speed benefit one would hope for (because of the high cost of shipping the histograms at each split over the network).

Besides random forests, I also trained various implementations of logistic regression (i.e. linear models). Their AUC as a function of dataset size is plotted below. One can see that the linear models' accuracy increases only a little from 100K to 1M and it is virtually the same for 1M and 10M. This is because a simple linear structure can be extracted from a smaller dataset and having more data points will not change the classification boundary significantly. On the other hand, more complex models such as random forests can further improve with increasing data size by further adjusting the classification boundary. However, one needs to pay a price in increased computational time for these more complex models. For example, if using H2O it takes 4000 seconds to train a random forest on the largest size and only 5 seconds to train a linear model.



An interesting point to note is that the AUC for the random forest trained on 100K observations is better than the AUC on a linear model trained on 10M observations. Therefore the answer to the question "more data or better algorithms?" often posed nowadays is that it depends (rather than the often argued "more data usually beats better algorithms").

To summarize, the commonly used R and Python random forest implementations have serious difficulties in dealing with training sets of tens of millions of observations. H2O or xgboost can deal with these datasets on a single machine (using memory and multiple cores efficiently). **Most data scientists are using R or Python** because of their excellent coverage of data munging, visualization and modeling (e.g. machine learning), their high-level APIs and the environments for interactive/exploratory data analysis they provide. (Note: this author argues that R is a better choice between the two). There is still a heated debate though about what to do when your data or computation is beyond what these two tools can handle. In fact, both R and Python can easily handle datasets of 10 million rows for aggregates, joins and other common data munging (i.e. "small/simple analytics") operations taking at most a few seconds (e.g. see [this benchmark](#)), therefore still supporting a nice interactive workflow. For machine learning (i.e. "big/complex analytics") on such datasets, with respect to algorithms such as random forests for example, one can use H2O or xgboost right from within R or Python almost seamlessly. Overall, I cannot help to observe that non-linear machine learning on such data sizes looks more like a high-performance computing task rather than a "big data" one.

Essentially, data scientists should ask themselves if their analytics tasks really need "big data" tools before choosing tools that add extra complexity (sometimes along with efficiency or reliability issues)

and that are still largely missing the multitude of statistical and data manipulation libraries and high-level APIs that make using R or Python so productive. Without taking the time to understand the tradeoffs and make an informed decision, many will spend a large part of their time fighting the tools instead of gaining value and insights from their data.



SZILARD PAFKA

+ SHARE THIS POST

[← Previous Post](#)

[Next Post →](#)

### 15 COMMENTS



Tianqi Chen - May 19, 2015

Comment on distributed learning.

When the dataset grows further, either distributed version or external memory version could be used. For example, distributed xgboost on a 4 B instance data with 20 machines in reasonable speed.

I agree that HPC optimization(push the limit of single and multi-core) is crucial, and this also pushes the boundary of what you can do distributely as well

[Reply to comment→](#)



Ken Williams - June 1, 2015

Hi Szilard,

I think it is fair to say that if you are capable of fitting all your data within one machine with a decent amount of RAM (as in your case) then do not expect scalable processing tools (such as Spark and Hadoop) to give you the most efficient solution.

When you run a program on a distributed 'Big Data' framework like these they have an initial 'start-up' overhead associated with them. This usually involves allocating memory for buffers and network connections in preparation for processing data in a distributed manner - none of which is really needed if you are only using a single machine. As such, they are not optimised for single machine processing and therefore - as you rightly point out - other solutions (such as H2O and scikit-learn) might be better for you.

Tools like Spark and Hadoop are most useful in cases where (1) people have too much data to fit on 1 machine, or (2) people are developing an initial 'proof of concept' type system and have the expectation that the solution they develop will be deployed to a production cluster and process larger amounts of data. If neither of these cases apply to you then other tools like H2O and scikit-learn may, as you correctly pointed out, provide a better solution.

Regards,



Ken Williams

Reply to comment→



Szilard - June 1, 2015

Thanks Ken for comments. I would totally agree with you in theory, but the reality of the existing “big data” machine learning tools do now live up to this (for now).

The initial overhead etc would be acceptable is something would take instead of 10ms 100ms or so (as that would diminish when scaling out), but these are computations running for minutes. It does not look to me the time is spend on building network connections etc. but rather in (inefficient) Java computation and memory handling etc. Note that H2O is also a distributed system (and ironically implemented in Java – but without using Java’s standard memory management), but it’s an order of magnitude faster and with an order of magnitude less memory footprint. Even assuming linear scaling you would need to throw in a large Spark/Hadoop cluster to compensate for this.

On the other hand I found it hard to find datasets for supervised learning that do not fit in the RAM of a single beefy server (unless you work for one of the few large internet companies or in adtech). The data initially collected by your systems might be “big” but after creating the data matrix for machine learning by extracting the useful features, the resulting datasets are typically not that big. Indeed, Spark/Hadoop might play a nice role in this feature engineering process (“simple analytics”, usually joins or aggregates) if your data is so large (though in many cases you are actually better served by a analytical/columnar/MPP database). But as far as training supervised models, my guess is that most people can fit their data on one machine: <http://fastml.com/the-emperors-new-clothes-distributed-machine-learning/>

On the other hand it seems to me that currently several algorithms in Spark MLlib provide lower predictive accuracy than its peers. Random forests is an extreme case (see results in the post), but Spark’s simple linear model also has accuracy problems. There is an independent study that also found this problem: <https://github.com/BIDData/BIDMach/wiki/Benchmarks#reuters-data>

Reply to comment→



Ken Williams - June 2, 2015

Hi Szilard – thanks for the reply. I’m not really sure why JVM-based distributed solutions (Hadoop, Spark) have such slow start-up times. I’m guessing that maybe they start-up a generic JVM each time but memory management is not optimised with maybe command line parameters (-Xms -Xmx, etc)? I do not know. Maybe using these parameters might improve start-up times. Maybe slow computation times are caused by poor garbage-collection settings, or inefficient code, or even poor compiler optimizations – without being involved in the build process it is difficult to say.

It would be interesting to know exactly what the H2O people have changed with the java memory management. I thought that memory management was a standard part of the JVM (although it might be slightly different between JSE, JEE and JME editions) which could only be modified with command-line arguments. Have they really changed this ?? How ?

I agree that most people can probably fit their data into one machine, although this may change in the future as more data is generated (IoT, mobile devices, etc). Hadoop and Spark can help in the ‘data cleaning’ process.



The benchmarks figures are interesting. I notice that these are using Spark 1.1 and 1.2, the current release is 1.3 with 1.4 due in July. I'm generally suspicious of benchmarks run in the cloud for a few reasons. They can be run on 'reserved machines' or 'spot available' machines (which means the processes might be stopped and moved to another machine at any moment - which adds to execution time) - obviously distributed cluster systems such as Hadoop and Spark which use multiple machines are particularly susceptible to this. (The text on github does not make this clear). I'm also cautious about benchmarks in the cloud because the machines used might also be 'dedicated' (i.e. solely for your use only) or 'shared' - and you have no idea what other processes people using your machine are doing (cpu-intensive, maybe ???). So generally, I do not trust benchmarks run in the cloud, but consistently slow times for spark would be a concern. I also see that "All the systems that had support for Intel MKL were compiled with MKL support linked in" so this is a \*big\* advantage for those systems.

Poor AUC accuracy in spark .... as the notes mention ("Spark accuracy was lower than the other systems after 3 passes over the dataset, perhaps due to Spark's SGD implementation").

Benchmarks are certainly worthwhile things but you need to see them on a 'level-playing field' and investigate all the options as to exactly how they were run before you can draw any firm conclusions from them.

Regards,

Ken Williams

Reply to comment→



Szilard - June 2, 2015

Thanks Ken. Here are some answers inline:

Q: It would be interesting to know exactly what the H2O people have changed with the java memory management. I thought that memory management was a standard part of the JVM (although it might be slightly different between JSE, JEE and JME editions) which could only be modified with command-line arguments. Have they really changed this ?? How ?

A: They basically implement their own storage system and use Java byte arrays only: <http://0xdata.com/blog/2014/03/h2o-architecture/>

Q: I agree that most people can probably fit their data into one machine, although this may change in the future as more data is generated (IoT's, mobile devices, etc). Hadoop and Spark can help in the 'data cleaning' process.

A: Yes, Hadoop/Spark can be useful for ETLing big data into useful data.

Q: The benchmarks figures are interesting. I notice that these are using Spark 1.1 and 1.2, the current release is 1.3 with 1.4 due in July.

A: I've been using 1.3 from the beginning: <https://github.com/szilard/benchmark/blob/master/0-init/1-install.txt>

Q: I'm generally suspicious of benchmarks run the in the cloud for a few reasons. They can be run on 'reserved machines' or 'spot available' machines (which means the processes might be stopped and moved to another machine at any moment - which adds to execution time) - obviously distributed cluster systems such as Hadoop and Spark which use multiple machines are particularly susceptible to this. (The text on github does not make this clear). I'm also cautious about benchmarks in the cloud because the machines used might also be 'dedicated' (i.e. solely for your use only) or 'shared' - and you have no idea what other processes people using your

machine are doing (cpu-intensive, maybe ???). So generally, I do not trust benchmarks run in the cloud, but consistently slow times for spark would be a concern.

A: I've been monitoring "steal time", did not find such issues:

<http://blog.scoutapp.com/articles/2013/07/25/understanding-cpu-steal-time-when-should-you-be-worried> The jobs are CPU only, so I/O does not play any role.

Q: I also see that "All the systems that had support for Intel MKL were compiled with MKL support linked in" so this is a \*big\* advantage for those systems.

A: I'm not sure what you are quoting, I never said anything like that or related to MKL.

Q: Poor AUC accuracy in spark .... as the notes mention ("Spark accuracy was lower than the other systems after 3 passes over the dataset, perhaps due to Spark's SGD implementation").

A: Well, the point is that it's inaccurate. Maybe there is magic option for a workaround, or it's a bug, or it uses a poor approximation. But right now out-of-the-box you lose predictive power.

Q: Benchmarks are certainly worthwhile things but you need to see them on a 'level-playing field' and investigate all the options as to exactly how they were run before you can draw any firm conclusions from them.

A: Sure, I say these:

"Simple/limited/incomplete benchmark for..." - the title of the github repo

"I'd like to emphasize this is not aimed to be a comprehensive benchmark. It started from my need at work to find a tool that can deal with datasets of similar size and structure. A more comprehensive benchmark would..."

"I also want to emphasize the importance of testing the tools on your data: your specific data structure, data size, etc."

As a final note, all code is on github and you are welcome to run it on your system (bare metal if you want) or even your data. If you have interesting results, please let me know (e.g. via github issues).

Reply to comment→



KEN WILLIAMS - June 3, 2015

Hi Szilard,

- Their implementation using Java byte arrays only looks extremely impressive.
- Apologies, I missed that your benchmarks were using Spark 1.3 - so the problem clearly hasn't been fixed then ....
- OK, 'steal time' is not an issue.
- My quote regarding "MKL support" was from here (<https://github.com/BIDData/BIDMach/wiki/Benchmarks#reuters-data>) and would have affected these results. I see you did not use MKL support with your benchmarks, so this does not affect your results.
- Poor AUC accuracy in Spark does not sound good. The only way to fix a poor implementation is with a better re-implementation. Someone will need to fix the code.

Thank you for sharing your code on github. I have downloaded a copy and will happily share if I find any interesting results. Also, I very much agree with your main point that people should first consider choosing simpler solutions rather than just using "big data" tools all the time.



Thanks again and Regards,

Ken Williams

[Reply to comment](#)→



Szilard - June 3, 2015

Thanks Ken. Independent verification is always good/welcome. Let me know if you have any results.

[Reply to comment](#)→



Joseph - July 17, 2015

Hi Szilard,

I work on Spark and wanted to add my thoughts about the MLlib benchmarks. First, I want to say thanks \*very much\* for including careful documentation of what you did. Too many benchmarks neglect that. A few thoughts:

**AUC/accuracy:** The AUC issue appears to be caused by MLlib tree ensembles aggregating votes, rather than class probabilities, as you suggested. I re-ran your test using class probabilities (which can be aggregated by hand), and then got the same AUC as other libraries. We're planning on including this fix in Spark 1.5 (and thanks for providing some evidence of its importance!).

\* Relatedly, from what I understand, that independent BIDMach study showing problems with Spark's accuracy was actually showing accuracy on training data, not test data; the only conclusion to be made from it is about the speed of convergence of the optimization algorithm. (They used SGD, but there are better ones available in Spark.)

**One-hot encoder:** Spark 1.4 includes this, plus a lot more feature transformers. Preprocessing should become ever-easier, especially using DataFrames (Spark 1.3+).

\* Btw, I noticed some of your tests use different one-hot encodings. (E.g., the Spark test uses dropFirst since R does that by default, but the sklearn one does not.) That could make a difference for trees, where you would not want to use dropFirst. However, in my tests on your benchmark, it did not actually matter much.

**Settings:** In your tests, I also noticed that some libraries use Gini and some Entropy for the splitting criterion. Interestingly, Entropy produces much smaller trees (1-2 orders of magnitude in some cases), though that did not actually affect results that much.

**Timing:** I didn't try to reproduce your results yet, but have a few thoughts. The main issue with MLlib's tree implementation is that it is optimized for training shallow trees, following the PLANET project. We're working on an alternative implementation geared towards training deep trees, hopefully aimed at Spark 1.5 or 1.6. One big benefit of running on top of Spark is that there is constant work on improving the underlying system, which MLlib will benefit from. In particular, the JVM memory management issues will improve as project Tungsten (which can be Googled) progresses.

**Local vs. distributed ML:** Tungsten and other efforts will help decrease the data size threshold above which distributed computing becomes faster. However, we're also working on providing better integration



between Spark and local ML libraries. E.g., you might want to do ETL in Spark, and then use Spark to distribute the task of model selection using a local ML library.

ETL vs. ML: As you discussed, ETL is a strong argument for using Spark. For academics, it may not be that important. But for industry, I think there's a big benefit in moving away from the traditional divide between engineers handling data management and ETL vs. data scientists handling ML on prepared data. Supporting both within the same platform makes for much smoother development.

Phew, that was longer than I meant, but hopefully it's helpful/interesting.

Joseph Bradley

[Reply to comment](#)→



Szilard - July 23, 2015

Thanks Joseph for extensive comments.

Since that post I created an "Absolute Minimal Benchmark" for random forests for different tools designed to be done with minimal amount of work – let's try to see the issues you addressed below on this specific data/size/setup. I also opened a github issue for Spark random forests, so we can easily collaborate on this: <https://github.com/szilard/benchmark-ml/issues/19>

Q: One-hot encoder: Spark 1.4 includes this, plus a lot more feature transformers. Preprocessing should become ever-easier, especially using DataFrames (Spark 1.3+).  
\* Btw, I noticed some of your tests use different one-hot encodings. (E.g., the Spark test uses dropFirst since R does that by default, but the sklearn one does not.) That could make a difference for trees, where you would not want to use dropFirst. However, in my tests on your benchmark, it did not actually matter much.

A: Yes, indeed. Can you please provide code (here on github: <https://github.com/szilard/benchmark-ml/issues/19>) that reads in the original dataset (pre- 1-hot encoding) and does the 1-hot encoding in Spark. Also, if random forest 1.4 API can use data frames, I guess we should use that for the training. Can you please provide code here <https://github.com/szilard/benchmark-ml/issues/19>.

Q: AUC/accuracy: The AUC issue appears to be caused by MLlib tree ensembles aggregating votes, rather than class probabilities, as you suggested. I re-ran your test using class probabilities (which can be aggregated by hand), and then got the same AUC as other libraries. We're planning on including this fix in Spark 1.5 (and thanks for providing some evidence of its importance!).

A: Fantastic. Can you please share code that does that already? <https://github.com/szilard/benchmark-ml/issues/19> I would be happy to check it out.

Q: Relatedly, from what I understand, that independent BIDMach study showing problems with Spark's accuracy was



actually showing accuracy on training data, not test data; the only conclusion to be made from it is about the speed of convergence of the optimization algorithm. (They used SGD, but there are better ones available in Spark.)

A: That was actually for logistic regression only (not random forests), and it seems like that problem has been fixed now in Spark 1.4, see this thread: <https://github.com/szilard/benchmark/issues/17>

Q: Settings: In your tests, I also noticed that some libraries use Gini and some Entropy for the splitting criterion. Interestingly, Entropy produces much smaller trees (1-2 orders of magnitude in some cases), though that did not actually affect results that much.

A: Yes, I played around a bit in H2O with that, but it did not seem to affect results too much.

Q: Timing: I didn't try to reproduce your results yet, but have a few thoughts. The main issue with MLib's tree implementation is that it is optimized for training shallow trees, following the PLANET project. We're working on an alternative implementation geared towards training deep trees, hopefully aimed at Spark 1.5 or 1.6. One big benefit of running on top of Spark is that there is constant work on improving the underlying system, which MLib will benefit from. In particular, the JVM memory management issues will improve as project Tungsten (which can be Googled) progresses.

A: Yes, from what I've been reading on Tungsten, that should give a big performance boost. For the timing I suggest you compare with <https://github.com/szilard/benchmark/tree/master/z-other-tools> Once you provide some code with 1-hot encoding, data frame RF API etc. I'd be happy to rerun myself and update the results.

Q: Local vs. distributed ML: Tungsten and other efforts will help decrease the data size threshold above which distributed computing becomes faster. However, we're also working on providing better integration between Spark and local ML libraries. E.g., you might want to do ETL in Spark, and then use Spark to distribute the task of model selection using a local ML library.

A: Yes, that's pretty much the trend I can see myself, most people I know of are using Spark for ETL and some for logistic regression (ML lib).

Q: ETL vs. ML: As you discussed, ETL is a strong argument for using Spark. For academics, it may not be that important. But for industry, I think there's a big benefit in moving away from the traditional divide between engineers handling data management and ETL vs. data scientists handling ML on prepared data. Supporting both within the same platform makes for much smoother development.

A: As someone working in industry I spend a lot of time with ETL before I get into ML. Of course there is a tradeoff between using specialized tools or a tool that tries to solve it all. I have a lot of structured data (and not super-big), so for me a fast MPP database is the way to go and then a high-performant ML



library. Moving the data between the 2 systems is not a big deal in my case, but I can see that this might be different for others.

Anyway, thanks a lot for your comments and I'm looking forward to working with you on some of these issues here on github: <https://github.com/szilard/benchm-ml/issues/19>

Reply to comment→



Szilard - September 17, 2015

For anyone interested, Joseph Bradley of Databricks has done some great work with the issues above, which I'll summarize below (see <https://github.com/szilard/benchm-ml/issues/19> for more details):

1. Joseph has provided code for 1-hot encoding directly in Spark:

<https://github.com/szilard/benchm-ml/issues/19#issuecomment-138734985> Thanks 😊

2. Joseph has provided code for scoring based on aggregating individual tree probabilities rather than tree votes <https://github.com/szilard/benchm-ml/issues/19#issuecomment-138734985> This fixes some of the accuracy issues, though there is still work to be done to resolve some strange behavior for n=1M (see #4 here

<https://github.com/szilard/benchm-ml/issues/19#issuecomment-139074455> )

3. I rerun the benchmark with the code provided above and with Spark 1.5 and I updated the github README. The new accuracy graph can be seen here [https://raw.githubusercontent.com/szilard/benchm-ml/ea4cb32380461156a7a05b8f39d33f330ce6fb10/2-  
rf/x-plot-auc.png](https://raw.githubusercontent.com/szilard/benchm-ml/ea4cb32380461156a7a05b8f39d33f330ce6fb10/2-rf/x-plot-auc.png) while the training times and memory footprint did not change significantly.

4. Joseph is working on the remaining AUC issue (n=1M) mentioned in #3 above

<https://github.com/szilard/benchm-ml/issues/19#issuecomment-141172953>

Reply to comment→



manikandan t v - August 5, 2015

Hi,

I am trying to run random forest in H2O. My data has 4000 records and 5000 features. the target is multiclass. I specified 500 trees with max depth 20. The run time is huge. It completed only 5% after an hour or so. Is this expected? My platform is windows 8.1 with dual core. Please clarify.

-Mani

Reply to comment→





Szilard - August 5, 2015

That's hard to tell without more details. Btw you can ask the question on the H2O's forum here:

<https://groups.google.com/forum/#!forum/h2ostream>

[Reply to comment](#)→



Ian Clarke - August 12, 2015

Would be interesting to include <http://quickml.org/> in this benchmark

[Reply to comment](#)→



Szilard - August 13, 2015

Sure, go for it! 😊 You can start with this

<https://github.com/szilard/benchm-ml/tree/master/z-other-tools> it should be very easy. If you have results, please post them on github as "issues" <https://github.com/szilard/benchm-ml/issues>

[Reply to comment](#)→



Stephen Boesch - October 23, 2017

Using "default" parameters represents an unconvincing approach: they will only reflect reasonably against the platforms almost by pure luck. A grid search to get at least "reasonable" hyperparameter values – and then using the optimum found – would seem a better start. The numbers for spark stand out as essentially without meaning/value here. While I would not be surprised to discover the statistical performance lagging – just using "whatever is out there" for the hyperparameters does not provide useful insights.

[Reply to comment](#)→

### LEAVE A REPLY

say something nice...

Name \*

Email \*

Website

Save my name, email, and website in this browser for the next time I comment.



### MORE POSTS



#### WELCOME TO DATA SCIENCE LA!

Welcome! We are launching DataScience.LA to provide a home for all the content (slides, code,...

AUGUST 2, 2014 0 COMMENTS 2



#### USER! 2014 CONFERENCE VIDEOS AND RELEASE PLAN

Many of us here in the DataScience.LA Team were also involved in organizing the useR! 2014...

AUGUST 4, 2014 0 COMMENTS 2

*I think it's interesting that R is an extremely fixable language.*

*Hadley Wickham*

*- Hadley on R -*



#### INTRODUCTION TO DATA SCIENCE FOR HIGH SCHOOL STUDENTS

Another exciting development in data science coming from our department at UCLA is a high...

AUGUST 20, 2014 1 COMMENT 2



#### DATA SCIENCE GOES TO COLLEGE WITH DATAFEST

Below is the first of several exciting data science developments for the younger generation, happening...

AUGUST 6, 2014 0 COMMENTS 2




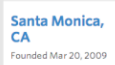
#### A CONVERSATION WITH HADLEY WICKHAM - THE USER! 2014 INTERVIEW

Hadley Wickham is famous. He's not Kardashian famous, but walking around useR! and seeing the...

AUGUST 6, 2014 13 COMMENTS 4

**LA R users group**

 May 6 - 6:30 PM  
New developments in knitr and R Markd  
95 R users | ★★★★★

 Santa Monica, CA  
Founded Mar 20, 2009  
R users: 1,298  
Group reviews: 33  
Past Meetups: 48

March 6 - 6:30 PM  
Statistics and Big Data at Google  
130 R users | ★★★★★ | 3 Photos

#### MEETUP GROUPS: LA R MEETUP

R is an open source software and programming language for data analysis, statistical modeling, visualization...

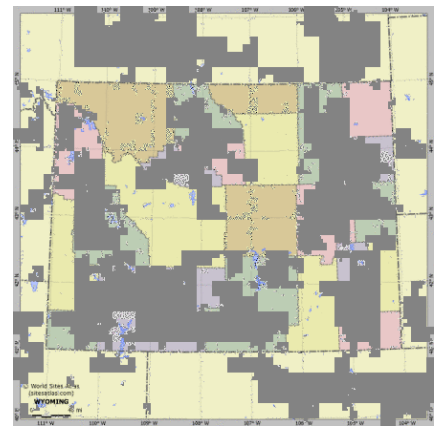
AUGUST 4, 2014 0 COMMENTS 0



#### STATISTICS GRAD SCHOOL- WHAT IS IT REALLY LIKE?

As a PhD graduate student in statistics here at UCLA, I'd like to talk a...

AUGUST 6, 2014 7 COMMENTS 0



#### MAKING MAPS WITH A PUNCHLINE

I've had a lifelong fascination with maps, and working with R definitely enables my map...

AUGUST 6, 2014 4 COMMENTS 0



#### PART 2 OF STATISTICS GRAD SCHOOL - WHAT IS IT REALLY LIKE?

(Note: This is my second installment on grad school, you can read the first part...

AUGUST 27, 2014 0 COMMENTS 2

© 2021 DataScience.LA

