From the post (http://datascience.la/benchmarking-random-forest-implementations/) we can see that Tree methods, especially the Random Forest, in Spark consumes more time and gives lower accuracy compare with those couter-parts in H2O and XGBoost. Since XGBoost is written in C++ with better tuned server performance, we'll spend most of our effort comparing Spark with H2O this time. We'll try to figure out that to what extend (speed and accuracy) does H2O better than Spark, why those advantages exist and how to fix Spark.

Random forest is a combination of decision trees. Decision tree, as a basic learner, suffers from relatively lower accuracy due to its high variance. Gradient boosted tree (GBT) reduces the variance, in the meantime it reduces the bias of a single tree. While random forest (RF) only reduces the variance through a method called "combination" or "bagging". Usually a GBT could reach the best performance among GBT, RF, and decision tree. However, the RF is faster and easy to parallelize in comparison with GBT.

In order to answer the question we mentioned above, we'll first to reproduce the experiment result of that post. Note that interestingly, the blog author uses different hyper-parameters in different methods, which leads to uneven results. So after recreating the previous result, we will try to construct a series of experiments bottom-up, with the same hyper-parameters as long as possible.

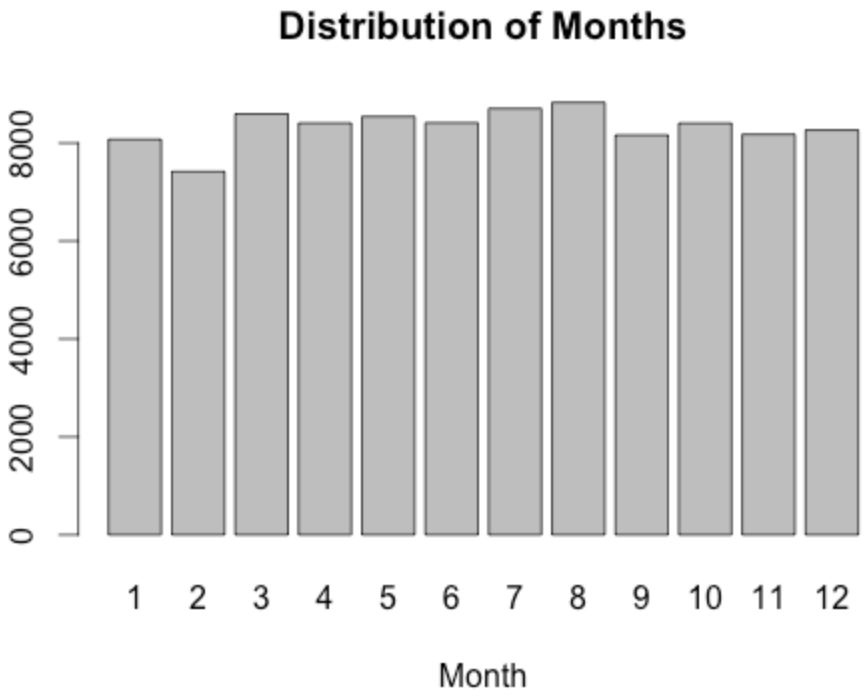
Dataset

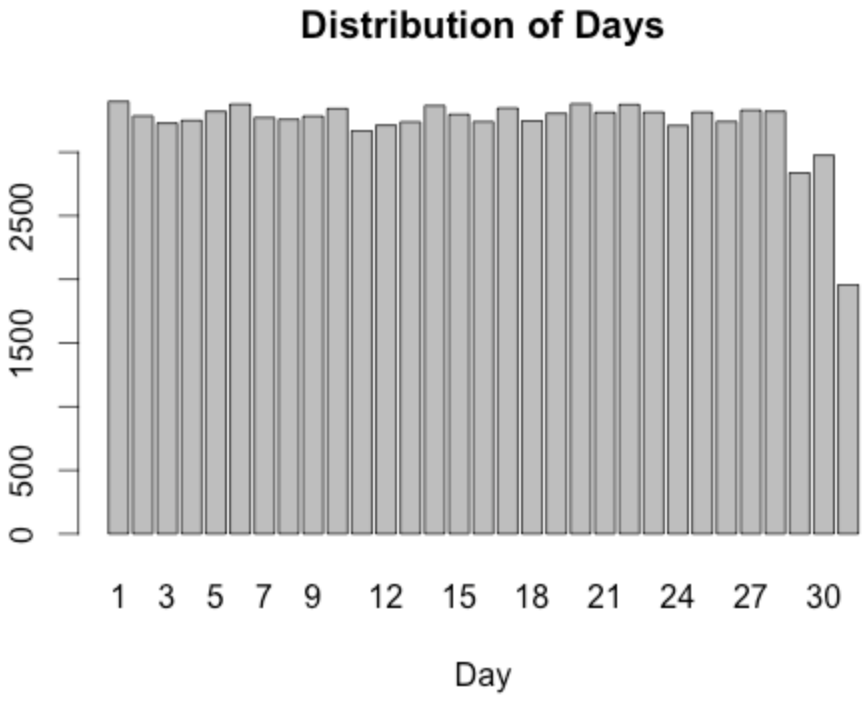
We use the airline on-time performance dataset, which is the same with the dataset used in that blog. Training data is sampled from airline dataset from year 2005 to 2006, while test data is sampled from 2007 airline dataset.

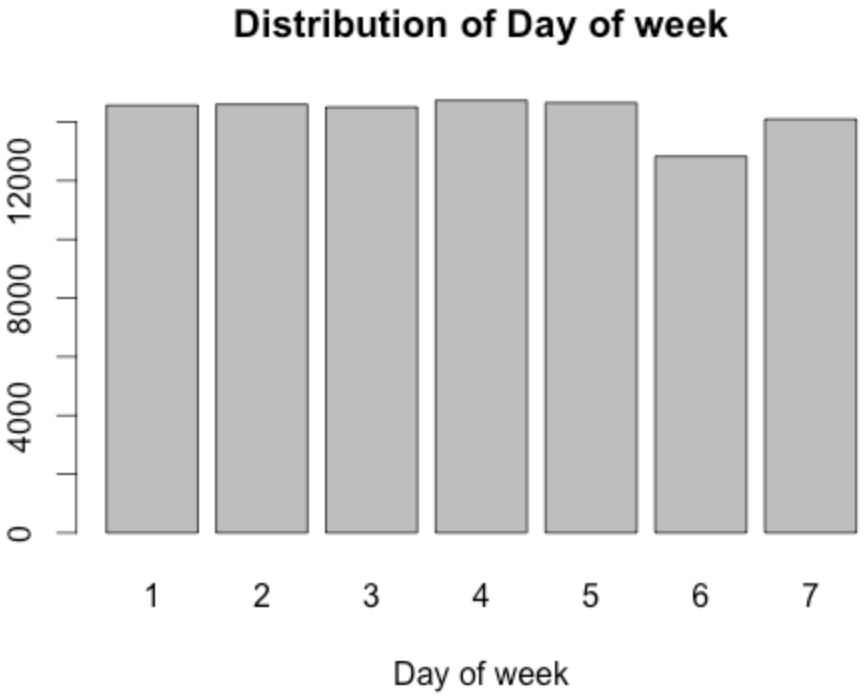
Columns of the data:

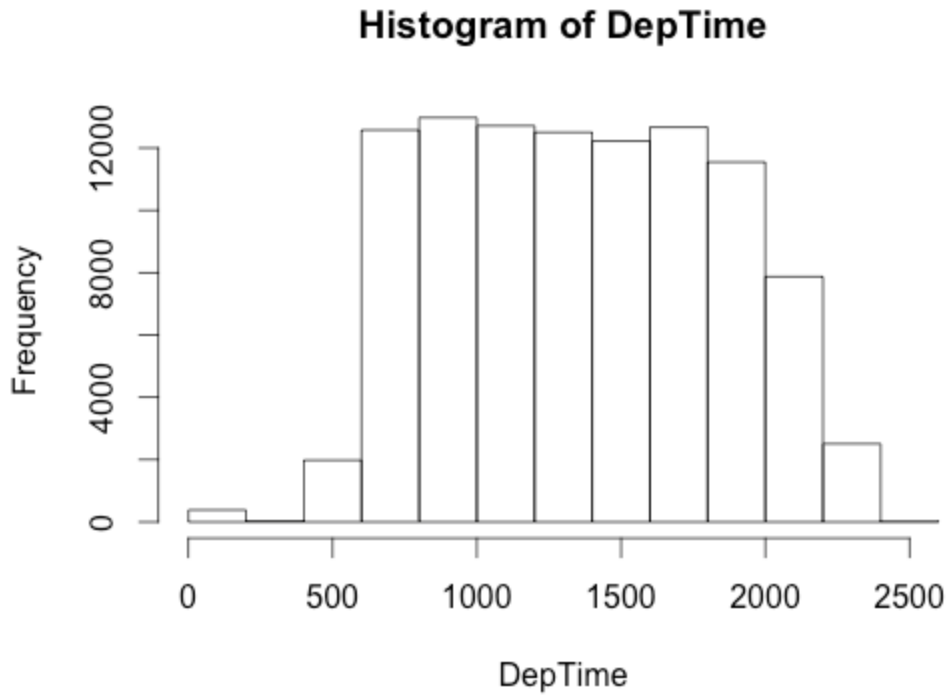
Label	Feature 1	Feature 2	Feature 3	Feature 4	Feature 5	Feature 6	Feature 7	Feature 8
dep_delayed_15min	Month	DayofMonth	DayOfWeek	DepTime	UniqueCarrier	Origin	Dest	Distance

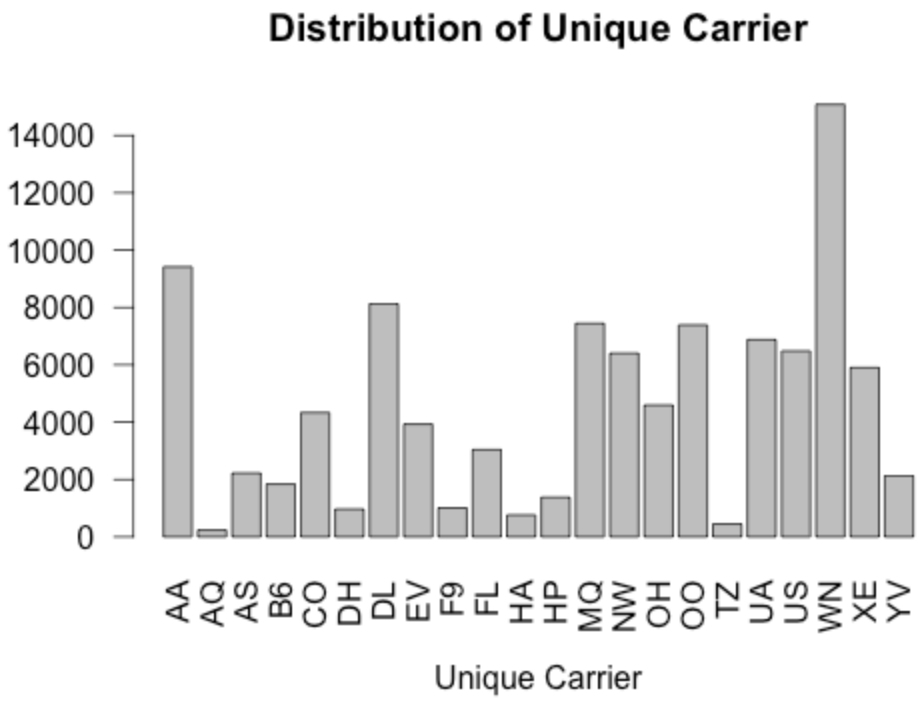
Here are some statistics results of train-0.1m.csv. According to the training set statistics, it is a highly biased dataset with 80956 "N" and 19044 "Y" for the target column "dep_delayed_15min". Our test set has similar distribution with 78383 "N" and 21617 "Y".

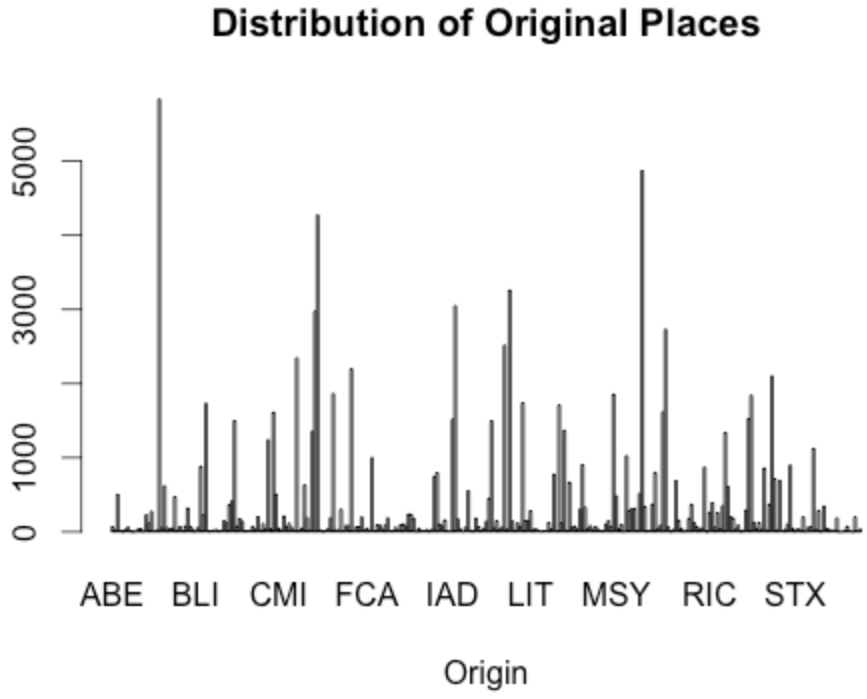

Distribution of Months
0
2000
4000
6000
8000
1
2
3
4
5
6
7
8
9
10
11
12
Month

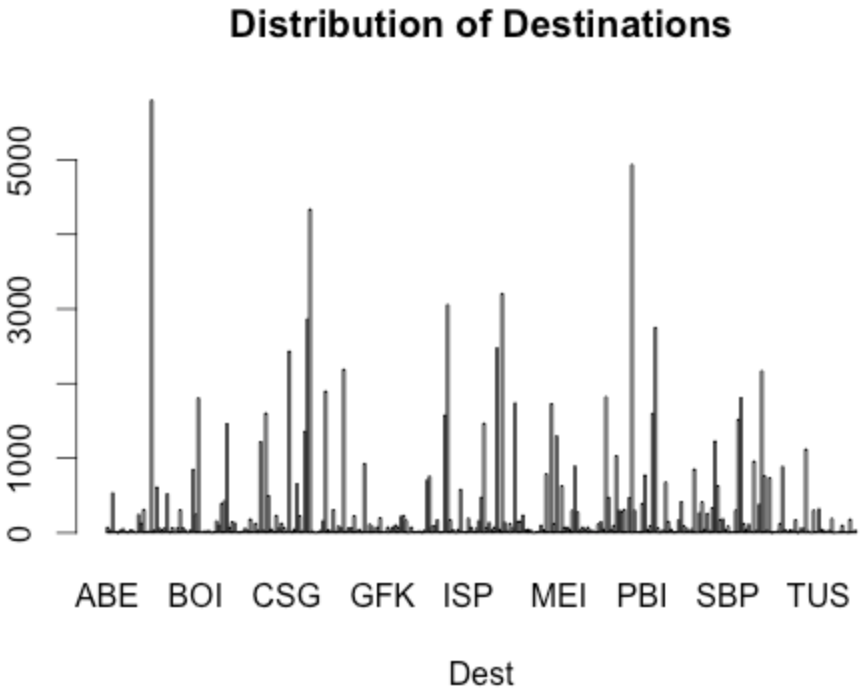

Distribution of Days
0
500
1500
2500
1
3
5
7
9
12
15
18
21
24
27
30
Day

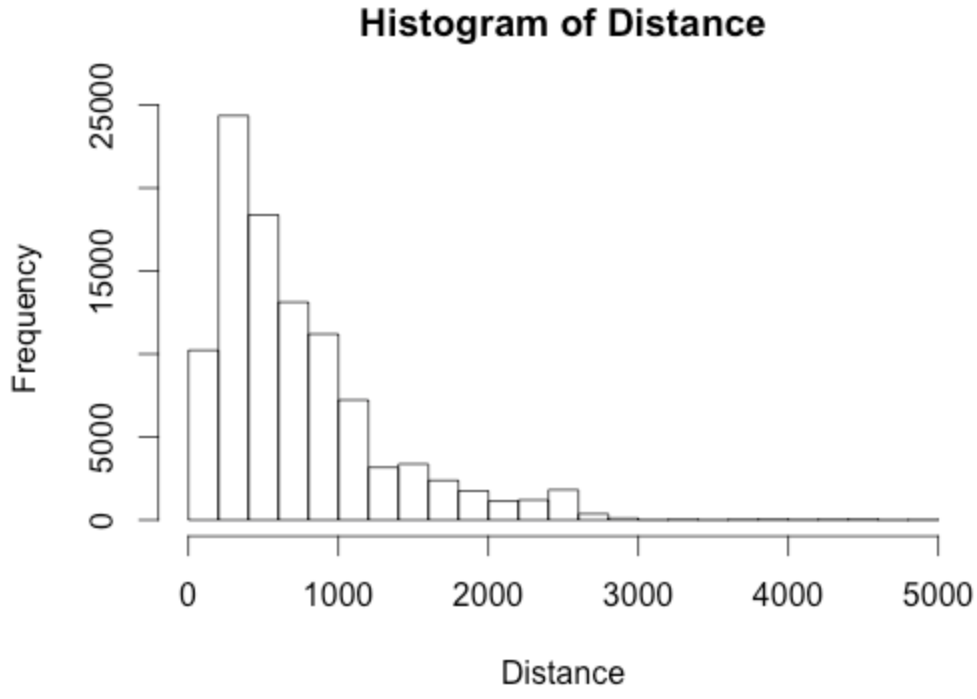

Distribution of Day of week
0
4000
8000
12000
1
2
3
4
5
6
7
Day of week


Histogram of DepTime
Frequency
0
4000
8000
12000
0
500
1000
1500
2000
2500
DepTime


Distribution of Unique Carrier
14000
12000
10000
8000
6000
4000
2000
0
AA
AQ
AS
B6
CO
DH
DL
EV
F9
FL
HA
HP
MQ
NW
OH
OO
TZ
UA
US
WN
XE
YV
Unique Carrier


Distribution of Original Places
0
1000
3000
5000
ABE
BLI
CMI
FCA
IAD
LIT
MSY
RIC
STX
Origin


Distribution of Destinations
0
1000
3000
5000
ABE
BOI
CSG
GFK
ISP
MEI
PBI
SBP
TUS
Dest


Histogram of Distance
Frequency
0
5000
15000
25000
0
1000
2000
3000
4000
5000
Distance

How to handle discrete values

In this dataset, all columns except DepTime and Distance are all categorical features. The easiest way to transform the data into LabeledPoint style is RFormula. However, RFormula is not suitable here because it produces different shapes of dataset in comparison with the original one. RFormula uses One-hot encoder so it expands the original dataset into thousands of columns. It brings two drawbacks:

1. The volume of dataset is expanded, which may hurt the performance.
2. One-hot encoder splits one column into cardinality size of new columns, while Random Forest cannot take groups of features into consideration, so it may hurt the accuracy.

The RFormula also recognizes DepTime and Distance into categorical features, so it brings more unnecessary new columns and reduces the accuracy a step further. Because DepTime and Distance are the two most important features for this task.

On the contrary, H2O uses the original dataset, without further preprocessing. As a result, I re-processed the data with the StringIndexer for each categorical columns.

Some results

Parameters for Spark

impurity	List(gini)
maxDepth	List(5)
maxBins	List(400)
minInstancesPerNode	List(1)
numTrees	ArraySeq(1)
featureSubsetStrategy	List(auto)
subsamplingRate	List(1.0)
maxMemoryInMB	32
cacheNodeIds	false
checkpointDir	None
checkpointInterval	10

Spark

dataset	base	maxDepth =20	ntrees=10	ntrees=10 ,	ntrees=10 0	100, 20	500	500, 20

				maxDepth=20				
0.01	0.651820952	0.556367746	0.663224047	0.633448682	0.678784341	0.669790915	0.684695681	NA
0.1	0.69303609	0.586464084	0.702090021	0.674857269	0.706579952	0.72267043	0.707556013	NA
1	0.701479433	0.62499902	0.705408545	0.744321364	0.710382107	NA	NA	NA
10	0.698504332	0.688066209	0.705764649	NA	NA	NA	NA	NA

H2O + R

dataset	base	maxDepth=20	ntrees=10	ntrees=10, maxDepth=20	ntrees=100	100, 20	500	500, 20
0.01	0.6185947	0.544936	0.6254092	0.6155168	0.6601971	0.6598323	0.6637433	0.661779
0.1	0.6883683	0.5695169	0.6948827	0.6646325	0.7088882	0.7078177	0.7092944	0.7147583
1	0.6426786	0.6029741	0.7109577	0.695598	0.7130144	0.735657	0.712805	0.7414554
10	0.6775318	0.6452127	0.701887	0.7077832	0.7133782	0.7471541	0.713035	0.7529335

Conclusions for now

OOM caused by new array

Repeatedly generate new arrays Endless OOM for heap
Wrong computation of working tasks in the same time
Parameters not good, like maxBins, 只能对所有的列统一设定.

调整一下base learner，看看不同数据集的情况下base learner能达到的最好水平，然后看看H2O和Spark哪个可以通过ensemble提升的更多。现在看来是H2O更多。所以Spark ensemble的方面可能存在问题。

1. Parameters differences (different params between H2O and Spark)

2. Data ingestion differences (categorical/continuous/OHE/input->treePoint->baggedInput->nodeIdCache)
3. Base learner differences (sampling bins, order/unorder categorical features, tree pruning)
4. Ensembling differences (weighted point, subsampling, feature subset, voting)
5. Design and implementation differences (histogram collecting/OOM, reduce problem, ser/de problem)