Benchmarking Machine Learning Implementations

Performance (time and accuracy)
of machine learning algorithms utilizing
R, Python, H2O, and Azure ML

Stuart Ward
October 14, 2015

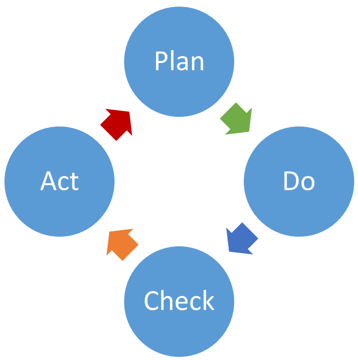
Agenda

- Context and caveats
- Computer and configurations
- Benchmarks and demos
- Discussion topics
- Q & A and Next Steps

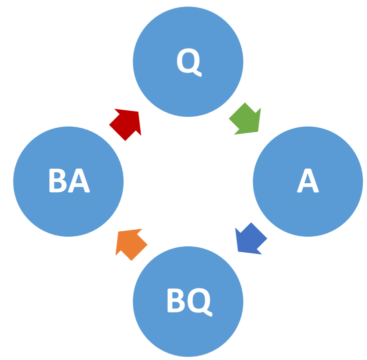
Context and caveats

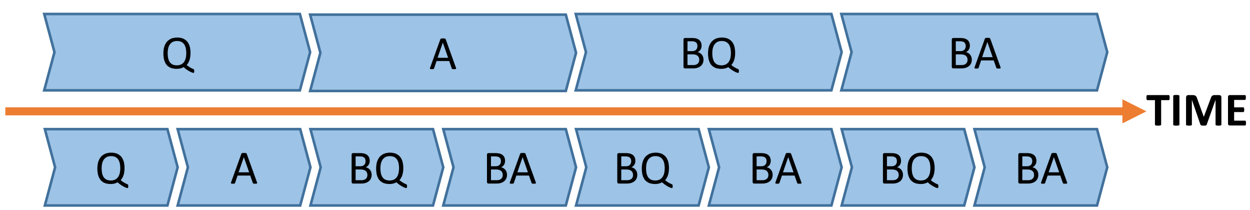
- Motivation: training time for random forest model in R
- Reduce the Question <> Answer cycle time

Plan > Do > Check > Act
Deming Cycle



Question > Answer
Better Question > Better Answer





Context and caveats

- **“No Free Lunch” Theorem**
  - The “No Free Lunch” Theorem (Wolpert 1996) argues that, without having substantive information about the modeling problem, there is no single model that will always do better than any other model. Because of this, a strong case can be made to try a wide variety of techniques, then determine which model to focus on.
    - *Applied Predictive Modeling*, Max Kuhn

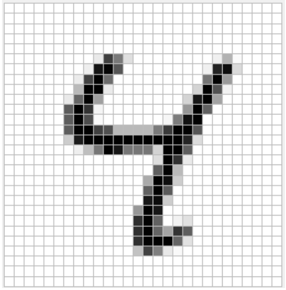
- Not even remotely comprehensive
- Presentation > Conversation > Ongoing Dialogue

Computer and configurations

- Computer
  - Windows 8.1 Pro 64-bit
  - Intel Quad Core i7-3840QM 2.8GHz
  - 16 GB RAM
- R
  - R version 3.2.1 (2015-06-08) – "World-Famous Astronaut" (64-bit)
  - Revolution R Open 3.2.1 (Multithreaded BLAS/LAPACK libraries detected. Using 4 cores for math algorithms.)
- Python
  - Anaconda distribution (Python 2.7.10-1)
  - scikit-learn (0.16.1)
- H2O
  - H2O version 3.0.1.7; Flow version 0.3.52
- Azure ML
  - Standard version

MNIST data

- From http://yann.lecun.com/exdb/mnist/
  - The MNIST ("Modified National Institute of Standards and Technology") database of handwritten digits, has a training set of 60,000 examples, and a test set of 10,000 examples. It is a subset of a larger set available from NIST. The digits have been size-normalized and centered in a fixed-size image
- From Kaggle
  - Each image is 28 pixels in height and 28 pixels in width, for a total of 784 pixels in total. Each pixel has a single pixel-value associated with it, indicating the lightness or darkness of that pixel, with higher numbers meaning darker. This pixel-value is an integer between 0 and 255, inclusive.



  - The training data set has 785 columns.
    The first column, called "label", is the digit that was drawn by the user. The rest of the columns contain the pixel-values of the associated image.

MNIST – Random Forest – R

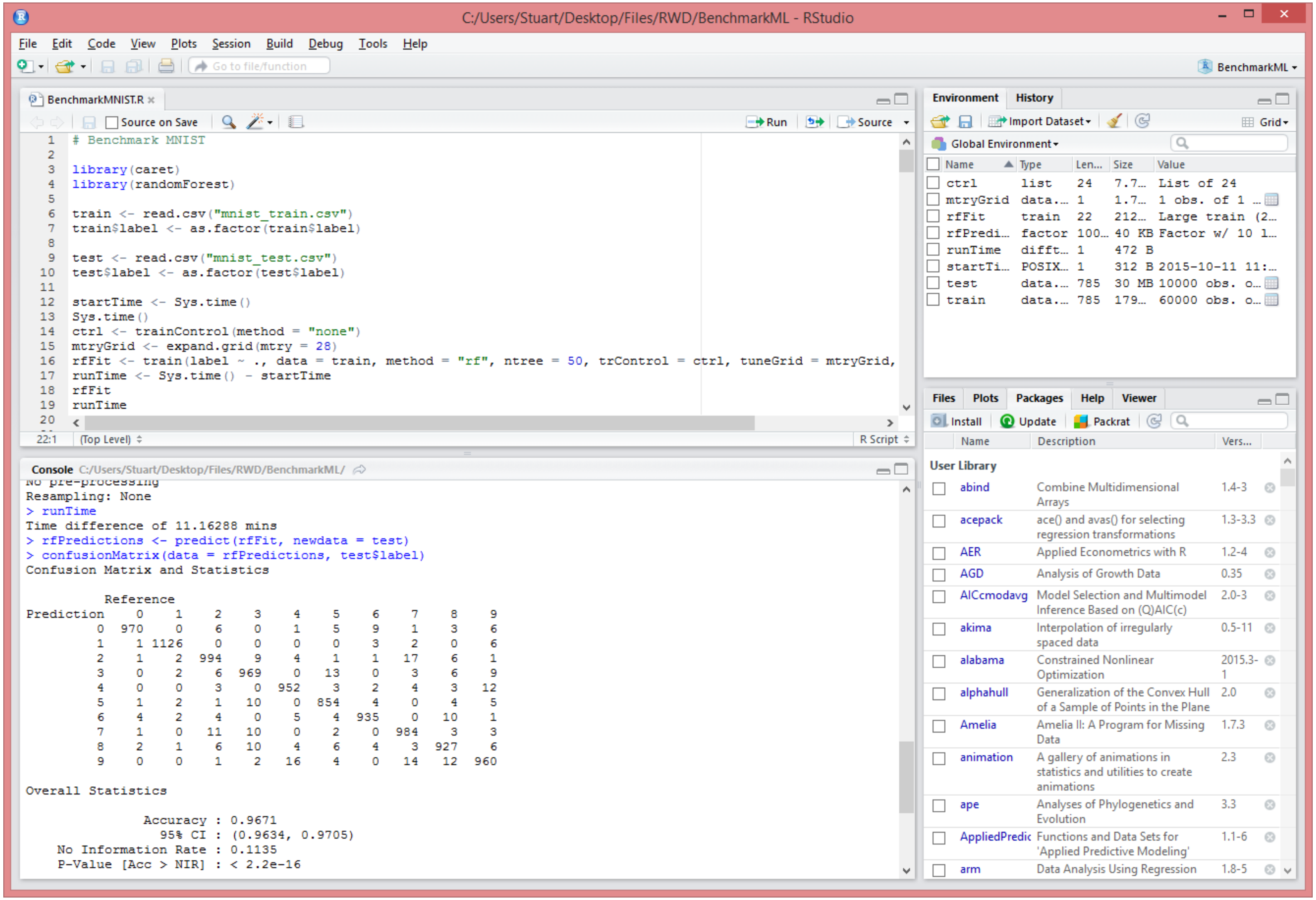
- Package 'randomForest' – Breiman and Cutler
  - "Why is R Slow" – Hadley Wickham, Advanced R
    - "Beyond performance limitations due to design and implementation, it has to be said that a lot of R code is slow simply because it's poorly written. Few R users have any formal training in programming or software development. Fewer still write R code for a living. Most people use R to understand data: it's more important to get an answer quickly than to develop a system that will work in a wide variety of situations."

- Package 'caret' – Kuhn
  - The caret package (short for ***C***lassification ***A***nd ***RE***gression **T**raining) is a set of functions that attempt to streamline the process for creating predictive models. The package contains tools for:
    - data splitting
    - pre-processing
    - feature selection
    - model tuning using resampling
    - variable importance estimation
    - http://topepo.github.io/caret/
  - "I built the function for ease of use rather than for speed or efficiency."
    - Max Kuhn, stack overflow post

MNIST – Random Forest – R

- Parameters
  - Training data = 60,000 rows, 785 columns (107 MB CSV file)
  - Testing data = 10,000 rows (18 MB CSV file)
  - ntree = 50; mtry = 28 (number of variables randomly sampled as candidates at each split)
- Results
  - Time to train the model
    - 743 seconds
  - RAM utilized during training
    - 2.5 GB
  - Prediction results on Test data
    - 96.64% accuracy; 3.36% error rate

MNIST – Random Forest – R



MNIST – Random Forest – Azure ML

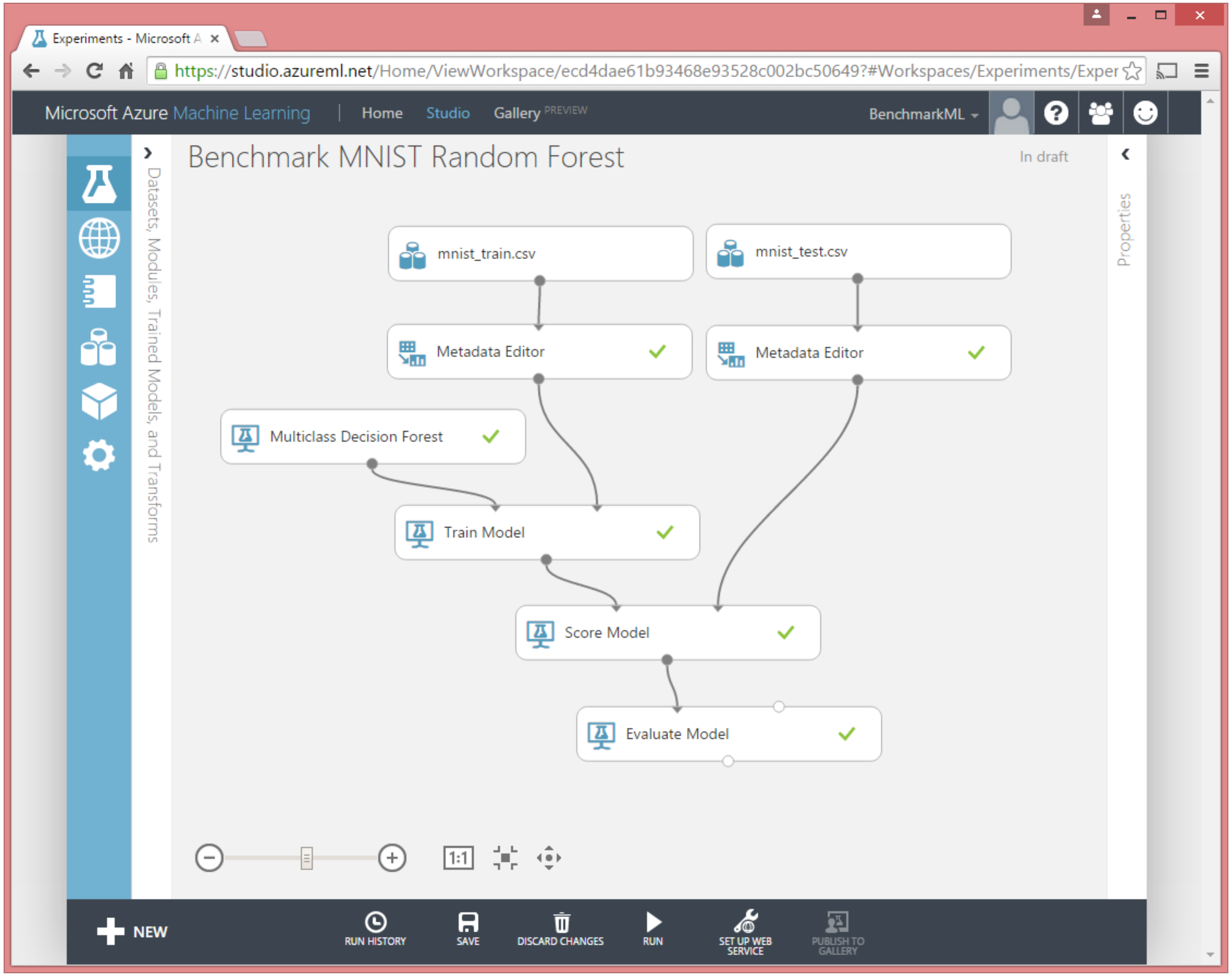
- Microsoft Azure Machine Learning
- “Powerful cloud-based predictive analytics”
  - Model your way
  - Deploy in minutes
  - Expand your reach
- Integration with R
- Integration with Python
- Jupyter Notebooks
- Pricing tiers
  - Free (single node; 100 modules and 1 hour duration per experiment)
  - Standard (multiple nodes; no limit on modules or duration)

MNIST – Random Forest – Azure ML

- Multiclass Decision Forest
  - Number of random splits per node (mtry equivalent): 128
  - Maximum depth of trees: 20

- Results (Standard tier)
  - Time to train the model
    - 185 seconds
  - Prediction results on Test data
    - 96.66% accuracy; 3.34% error rate

MNIST – Random Forest – Azure ML



MNIST – Random Forest – H2O

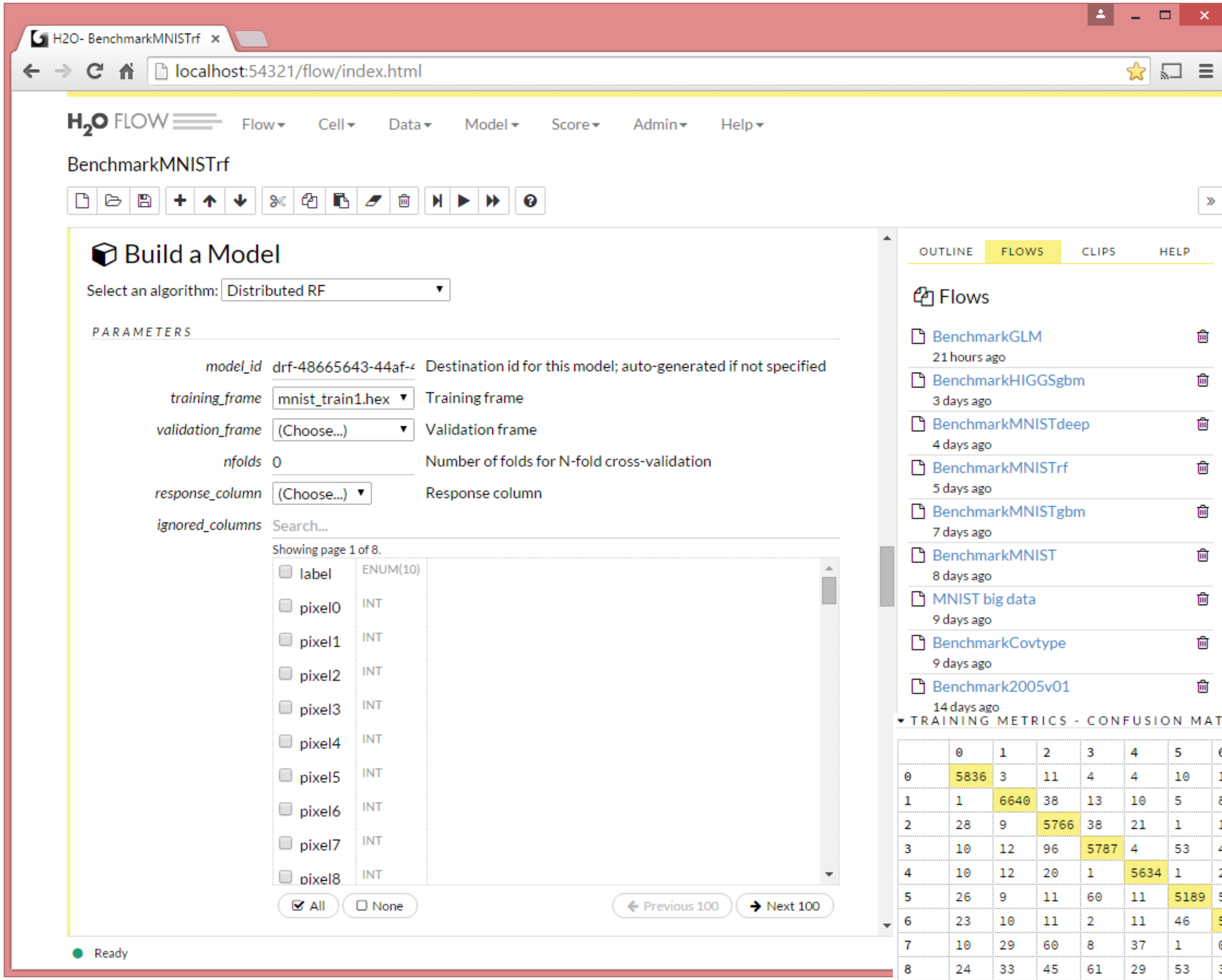
- H2O – "Fast Scalable Machine Learning"
  - "H2O is for data scientists and application developers who need fast, in-memory scalable machine learning for smarter applications. H2O is an open source parallel processing engine for machine learning. Unlike traditional analytics tools, H2O provides a combination of extraordinary math, a high performance parallel architecture, and unrivaled ease of use."
- Prerequisite: 64-bit Java version 1.6 or later
- User interfaces to H2O cluster(s)
  - H2O Flow – web-based user interface
  - R package ('h2o') – operate an H2O cluster from within R
  - Python module ('h2o') – operate an H2O cluster from within Python
- Sparkling Water – integration of H2O and Spark
- Tableau >> R >> H2O
- Excel >> H2O

MNIST – Random Forest – H2O

- Distributed Random Forests
  - The H2O implementation uses sampling without replacement and a user-specified sampling rate, but does not currently support traditional bootstrapping (sampling with replacement).
  - Maximum depth of trees: 20

- Results (multi-threaded)
  - Compressed file size
    - 28 MB
  - Time to train the model
    - 120 seconds
  - RAM utilized during training
    - 4.1 GB
  - Prediction results on Test data
    - 96.36% accuracy; 3.64% error rate

MNIST – Random Forest – H2O



TRAINING METRICS - CONFUSION MATRIX

	0	1	2	3	4	5	6	7	8	9	Error	Rate
0	5836	3	11	4	4	10	16	2	34	3	0.0147	87 / 5,923
1	1	6640	38	13	10	5	8	14	7	6	0.0151	102 / 6,742
2	28	9	5766	38	21	1	15	24	48	8	0.0322	192 / 5,958
3	10	12	96	5787	4	53	4	56	69	40	0.0561	344 / 6,131
4	10	12	20	1	5634	1	27	14	17	106	0.0356	208 / 5,842
5	26	9	11	60	11	5189	57	9	29	20	0.0428	232 / 5,421
6	23	10	11	2	11	46	5791	0	24	0	0.0215	127 / 5,918
7	10	29	60	8	37	1	0	6041	8	71	0.0358	224 / 6,265
8	24	33	45	61	29	53	38	12	5504	52	0.0593	347 / 5,851
9	18	12	21	85	70	14	3	68	32	5626	0.0543	323 / 5,949
Total	5986	6769	6079	6059	5831	5373	5959	6240	5772	5932	0.0364	2,186 / 60,000

MNIST – Random Forest – Python

- scikit-learn "Machine Learning in Python"
  - Simple and efficient tools for data mining and data analysis
  - Accessible to everybody, and reusable in various contexts
  - Built on NumPy, SciPy, and matplotlib
  - Open source, commercially usable - BSD license
  - Included in Anaconda distribution of Python

- Results (single-threaded)
  - Time to train the model
    - 21 seconds
  - RAM utilized during training
    - 0.7 GB
  - Prediction results on Test data
    - 96.82% accuracy; 3.18% error rate

- Results (multi-threaded)
  - Time to train the model
    - 5 seconds
  - RAM utilized during training
    - 0.7 GB
  - Prediction results on Test data
    - 96.74% accuracy; 3.26% error rate

MNIST – Random Forest – Python

- Release dates for scikit-learn
  - 0.14 – August 2013
  - 0.15 – July 2014
  - 0.16 – April 2015
- Performance improvements in the RF implementation
  - Big speedup for training Random Forests in scikit-learn 0.15

Dataset	wiseRF 1.5.11	scikit-learn 0.14	scikit-learn 0.15	CudaTree 0.6
ImageNet subset	23s	50s	13s	25s
CIFAR-100 (raw)	160s	502s	181s	197s
covertype	107s	463s	73s	67s
poker	117s	415s	99s	59s
PAMAP2	1,066s	7,630s	1,683s	934s
intrusion	667s	1,528s	241s	199s

MNIST – Random Forest – Python

Python/Notebooks/ × BenchmarkMNIST ×

localhost:8888/notebooks/Desktop/Files/Python/Notebooks/BenchmarkMNIST.ipynb

jupyter BenchmarkMNIST Last Checkpoint: Last Tuesday at 1:59 AM (autosaved)

File Edit View Insert Cell Kernel Help | Python 2

Code | Cell Toolbar: None

```
In [1]: import pandas as pd
        from sklearn.ensemble import RandomForestClassifier
        from sklearn.metrics import confusion_matrix
        from sklearn.metrics import accuracy_score
        from datetime import datetime

In [2]: train = pd.read_csv('mnist_train.csv')
        test = pd.read_csv('mnist_test.csv')

In [3]: X_train = train.ix[:,1:785]
        y_train = train.ix[:,0]
        X_test = test.ix[:,1:785]
        y_test = test.ix[:,0]

In [16]: rf = RandomForestClassifier(n_estimators = 50, max_features = 28, n_jobs = -1)
         startTime = datetime.now()
         rf.fit(X_train, y_train)
         print "Run time ", datetime.now() - startTime

         Run time  0:00:05.181000

In [8]: rf.fit

Out[8]: <bound method RandomForestClassifier.fit of RandomForestClassifier(bootstrap=True, class_weight=None,
        criterion='gini',
                    max_depth=None, max_features=28, max_leaf_nodes=None,
                    min_samples_leaf=1, min_samples_split=2,
                    min_weight_fraction_leaf=0.0, n_estimators=50, n_jobs=-1,
                    oob_score=False, random_state=None, verbose=0,
                    warm_start=False)>

In [17]: preds = rf.predict(X_test)
         print accuracy_score(y_test,preds)
         print confusion_matrix(y_test,preds)

         0.9674
         [[ 969    0    0    0    0    3    3    2    3    0]
          [    0 1125    1    3    0    2    2    0    1    1]
          [    7    0  999    3    4    0    2   10    7    0]
          [    1    0   13  965    0   12    0    9    7    3]
          [    0    0    2    0  955    0    6    0    5   14]
          [    3    2    1   15    3  851    7    3    4    3]
          [    6    3    0    0    3    4  938    0    4    0]
          [    1    6   21    2    0    0    0  985    3   10]
          [    7    0    2    8    4    5    7    4  927   10]
          [    5    6    3   10   12    2    2    3    6  960]]
```

MNIST – Random Forest – Summary

- Summary of Random Forest benchmarks

Platform	Time (sec)	RAM (GB)	Accuracy (%)
R	743	2.5	96.64
Azure ML	185	-	96.66
H2O	120	4.1	96.36
Python (1t)	21	0.7	96.82
Python (8t)	5	0.7	96.74

MNIST – Random Forest – ‘Bigger’ data

- One-off test to push the limits of available RAM

- Scale up MNIST data
  - 1 million rows; 785 columns
  - 2 GB CSV file

- Results
  - R – loading data exceeded 90% of available RAM; stopped process
  - Azure ML – initial data load (11 minutes); RF with 10 trees (169 seconds; training time only)
  - H2O – data loaded (4 GB RAM); RF with 10 trees (346 seconds)
  - Python – data loaded (7 GB RAM); RF with 10 trees (64 seconds)

MNIST – Neural Networks – Summary

- "There is quite an art in training neural networks. The model is generally over parametrized, and the optimization problem is nonconvex and unstable unless certain guidelines are followed."

  The Elements of Statistical Learning

Platform	Time (sec)	RAM (GB)	Accuracy (%)	(Layers, Units) and Iterations
R (RSNNS)	3,525	2.6	97.47	(100) and 100
R (RSNNS)	3,268	3.0	97.96	(200, 200) and 30
Azure ML	276	-	97.73	(100) and 100
Azure ML (sweep)	819	-	98.03	(100) and 160
H2O	591	2.0	97.50	(200, 200) and 30

HIGGS data set

- Data set from UCI
  - “Abstract: This is a classification problem to distinguish between a signal process which produces Higgs bosons and a background process which does not.”
- 11,000,000 rows
- 28 attributes
- The first 21 features (columns 2-22) are kinematic properties measured by the particle detectors in the accelerator. The last seven features are functions of the first 21 features; these are high-level features derived by physicists to help discriminate between the two classes.
- Data reduced to 3,000,000 rows (1.3 GB CSV file)

HIGGS – Gradient Boosting – Summary

- Summary of Gradient Boosting benchmarks
- Configuration: trees = 50, depth = 5, distribution = Bernoulli

Platform	Time (sec)	RAM (GB)
R (1t)	1,694	3.8
Azure ML	210	-
H2O (8t)	78	5.3
Python (1t)	2,149	2.2

Synthetic data for logistic regression

- Simulate data for logistic regression

```
x1 = rnorm(1000)            # some continuous variables
x2 = rnorm(1000)
z = 1 + 2*x1 + 3*x2         # linear combination with a bias
pr = 1/(1+exp(-z))          # pass through an inv-logit function
y = rbinom(1000,1,pr)       # bernoulli response variable

df = data.frame(y=y,x1=x1,x2=x2)
glm( y~x1+x2,data=df,family="binomial")
```

- Increase the size of data from 1,000 to 3,000,000 rows
  - 116 MB CSV file

Synthetic data Logistic Regression (w/wo CV) – Summary

Logistic Regression without CV

Platform	Time (sec)	RAM (GB)
R	23	1.9
Azure ML	34	-
H2O	3	0.5
Python (4t)	4	0.7

Logistic Regression with 10-fold CV

Platform	Time (sec)	RAM (GB)
R	197	4.0
Azure ML	149	-
H2O	39	0.5
Python (4t)	132	0.7

‘Random Forest’ of thoughts

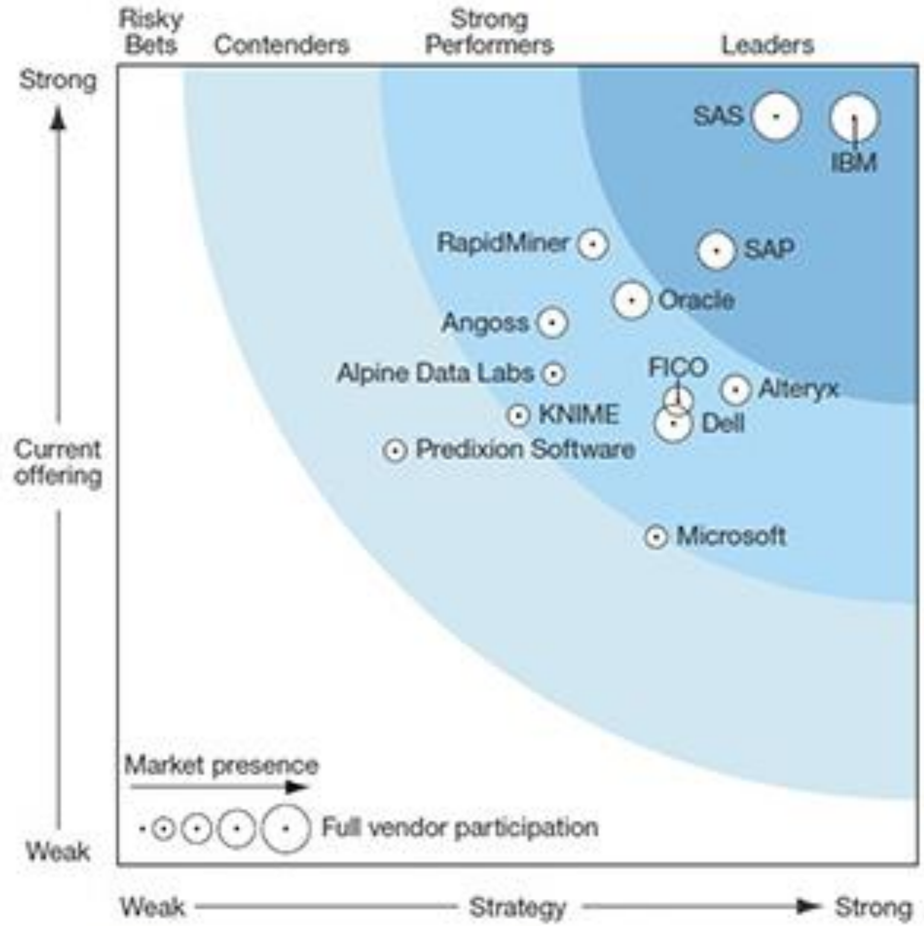
- What about…
  - Gartner 2015 Magic Quadrant Advanced Analytics Platforms
- SAS
- IBM SPSS
- KNIME
- RapidMiner
- Revolution R Enterprise
- Alteryx

Source: Gartner (February 2015)

- Stata
  - [Nate Silver Q&A]:
    - Q. What software do you use to analyze your data?
    - A. I use Stata for anything hardcore and Excel for the rest

‘Random Forest’ of thoughts

- What about…
  - Forrester Wave Q2 ’15
    Big Data Predictive Analytics
- SAS
- IBM SPSS
- KNIME
- RapidMiner
- Revolution R Enterprise
- Alteryx



- Stata
  - [Nate Silver Q&A]:
    - Q. What software do you use to analyze your data?
    - A. I use Stata for anything hardcore and Excel for the rest

‘Random Forest’ of thoughts

- Random Forests: R vs Python by Linda Uruchurtu
- Azure ML multiple models and sweep parameters
- GPU support
- To CV or not to CV
- Scaling up (time and RAM)
- Predict vs Production
- Sampling big data vs. utilizing all the data; does more data beat better algorithms; can you have both?
- No free lunch theorem; no free lunch platform
- Anonymize data; cloud ML
- Class imbalance
- No one ever got fired for running Hadoop on a cluster

‘Random Forest’ of thoughts

- [Classifier Technology and the Illusion of Progress]
  - David J. Hand (2006)

  - A great many tools have been developed for supervised classification, ranging from early methods such as linear discriminant analysis through to modern developments such as neural networks and support vector machines.
  - A large number of comparative studies have been conducted in attempts to establish the relative superiority of these methods.
  - This paper argues that these comparisons often fail to take into account important aspects of real problems, so that the apparent superiority of more sophisticated methods may be something of an illusion.
  - In particular, simple methods typically yield performance almost as good as more sophisticated methods, to the extent that the difference in performance may be swamped by other sources of uncertainty that generally are not considered in the classical supervised classification paradigm.

Q&A and Next Steps