

H2O Machine Learning & Deep Learning London Workshop



Jo-fai (Joe) Chow
Data Scientist
joe@h2o.ai
@matlabulous

Data Science for IoT Meetup
Barclays Eagle Venture Labs
21st & 24th November, 2016

Download Data & Code for Workshop

- Please go to

[bit.ly/h2o_iot_workshop1](https://github.com/woobe/H2O_London_Workshop)

The screenshot shows a GitHub repository page for 'H2O_London_Workshop'. The repository has 30 commits, 1 branch, 0 releases, 1 contributor, and is licensed under Apache-2.0. A black arrow points to the 'Clone or download' button in the top right corner of the main content area.

File	Description	Time
code	new version for Nov 2016 workshops	3 hours ago
data	data for workshop	2 months ago
slides/2016_09_First_Round	first round slides	3 hours ago
.gitignore	added .Rproj	7 hours ago
LICENSE	Initial commit	2 months ago
README.md	Create README.md	2 months ago

Example 1: Classification

(Use Case: Predictive Maintenance)



Data for Use Case 1: SECOM

UCI 

Machine Learning Repository
Center for Machine Learning and Intelligent Systems

SECOM Data Set

Download: [Data Folder](#), [Data Set Description](#)

Abstract: Data from a semi-conductor manufacturing process



Data Set Characteristics:	Multivariate	Number of Instances:	1567	Area:	Computer
Attribute Characteristics:	Real	Number of Attributes:	591	Date Donated	2008-11-19
Associated Tasks:	Classification, Causal-Discovery	Missing Values?	Yes	Number of Web Hits:	37895

Source:

<https://archive.ics.uci.edu/ml/datasets/SECOM>

Authors: Michael McCann, Adrian Johnston

Data Set Information:

A complex modern semi-conductor manufacturing process is normally under consistent surveillance via the monitoring of signals/variables collected from sensors and or process measurement points. However, not all of these signals are equally valuable in a specific monitoring system. The measured signals contain a combination of useful information, irrelevant information as well as noise. It is often the case that useful information is buried in the latter two. Engineers typically have a much larger number of signals than are actually required. If we consider each type of signal as a feature, then feature selection may be applied to identify the most relevant signals. The Process Engineers may then use these signals to determine key factors contributing to yield excursions downstream in the process. This will enable an increase in process throughput, decreased time to learning and reduce the per unit production costs.

To enhance current business improvement techniques the application of feature selection as an intelligent systems technique is being investigated.

The dataset presented in this case represents a selection of such features where each example represents a single production entity with associated measured features and the labels represent a simple pass/fail yield for in house line testing, figure 2, and associated date time stamp. Where -1 corresponds to a pass and 1 corresponds to a fail and the data time stamp is for that specific test point.

Using feature selection techniques it is desired to rank features according to their impact on the overall yield for the product, causal relationships may also be considered with a view to identifying the key features.

Results may be submitted in terms of feature relevance for predictability using error rates as our evaluation metrics. It is suggested that cross validation be applied to generate these results. Some baseline results are shown below for basic feature selection techniques using a simple kernel ridge classifier and 10 fold cross validation.

Baseline Results: Pre-processing objects were applied to the dataset simply to standardize the data and remove the constant features and then a number of different feature selection objects selecting 40 highest ranked features were applied with a simple classifier to achieve some initial results. 10 fold cross validation was used and the balanced error rate (*BER) generated as our initial performance metric to help investigate this dataset.

SECOM Dataset: 1567 examples 591 features, 104 fails

We want to predict fails in the future.

FSmethod (40 features) BER % True + % True - %
S2N (signal to noise) 34.5 +-2.6 57.8 +-5.3 73.1 +-2.1
Ttest 33.7 +-2.1 59.6 +-4.7 73.0 +-1.8
Relief 40.1 +-2.8 48.3 +-5.9 71.6 +-3.2
Pearson 34.1 +-2.0 57.4 +-4.3 74.4 +-4.9
Ftest 33.5 +-2.2 59.1 +-4.8 73.8 +-1.8
Gram Schmidt 35.6 +-2.4 51.2 +-11.8 77.5 +-2.3

The ML Problem – Pass/Fail

- Inputs
 - 591 features
- Output
 - Classification
 - $-1 = \text{pass}$
 - $1 = \text{fail}$
- Size: 1567 Samples



	A	B	C	D	E	F	G	H	I	J	K	L
1	ID	Feature 001	Feature 002	Feature 003	Feature 004	Feature 005	Feature 006	Feature 007	Feature 008	Feature 009	Feature 010	Feature 011
2	1	3030.93	2564	2187.7333	1411.1265	1.3602	100	97.6133	0.1242	1.5005	0.0162	-0.001
3	2	3095.78	2465.14	2230.4222	1463.6606	0.8294	100	102.3433	0.1247	1.4966	-5.00E-04	-0.01
4	3	2932.61	2559.94	2186.4111	1698.0172	1.5102	100	95.4878	0.1241	1.4436	0.0041	0.001
5	4	2988.72	2479.9	2199.0333	909.7926	1.3204	100	100.3967	0.1235	1.4882	-0.0124	-0.001
6	5	3032.24	2502.87	2233.3667	1326.52	1.5334	100	100.3967	0.1235	1.5031	-0.0031	-0.001
7	6	2946.25	2432.84	2233.3667	1326.52	1.5334	100	100.3967	0.1235	1.5287	0.0167	0.001
8	7	3030.27	2430.12	2230.4222	1463.6606	0.8294	100	102.3433	0.1247	1.5816	-0.027	0.01
9	8	3058.88	2690.15	2248.9	1004.4692	0.7884	100	106.24	0.1185	1.5153	0.0157	7.00E-01
10	9	2967.68	2600.47	2248.9	1004.4692	0.7884	100	106.24	0.1185	1.5358	0.0111	-0.001
11	10	3016.11	2428.37	2248.9	1004.4692	0.7884	100	106.24	0.1185	1.5381	0.0159	0.001
12	11	2994.05	2548.21	2195.1222	1046.1468	1.3204	100	103.34	0.1223	1.5144	-0.019	0.001
13	12	2928.84	2479.4	2196.2111	1605.7578	0.9959	100	97.9156	0.1257	1.469	0.017	-0.01
14	13	2920.07	2507.4	2195.1222	1046.1468	1.3204	100	103.34	0.1223	1.531	-0.0259	0.02
15	14	3051.44	2529.27	2184.4333	877.6266	1.4668	100	107.8711	0.124	1.5236	-0.0209	-0.001
16	15	2063.97	2629.18	2224.6222	947.7730	1.2924	100	104.8489	0.1197	1.4474	0.0144	-0.01
17	16	ID (excluded from modeling)										
18	17	3032.73	2517.79	2270.2556	1258.4558	1.395	100	104.8078	0.1207	1.5537	0.022	-0.001
19	18	3040.34	2501.16	2207.3889	962.5317	1.2043	100	104.0311	0.121	1.5481	-0.0367	0.001
20	19											

File Home Insert Draw Page Layout Formulas Data Review View Tell me what you want to do secom.csv - Excel Chow, Jo Share

VU1 Classification

	VJ	VK	VL	VM	VN	VO	VP	VQ	VR	VS	VT	VU	Classification
1	Feature 581	Feature 582	Feature 583	Feature 584	Feature 585	Feature 586	Feature 587	Feature 588	Feature 589	Feature 590	Date		
2			0.5005	0.0118	0.0035	2.363					2008-07-19		-1
3	0.006					4.4447	0.0096	0.0201	0.006	208.2045	2008-07-19		-1
4	0.0148	Features (Numeric)				3.1745	0.0584	0.0484	0.0148	82.8602	2008-07-19		1
5	0.0044	73.8432	0.499	0.0103	0.0025	2.0544	0.0202	0.0149	0.0044	73.8432	2008-07-19		-1
6			0.48	0.4766	0.1045	99.3032	0.0202	0.0149	0.0044	73.8432	2008-07-19		-1
7	0.0052	44.0077	0.4949	0.0189	0.0044	3.8276	0.0342	0.0151	0.0052	44.0077	2008-07-19		-1
8			0.501	0.0143	0.0042	2.8515	0.0342	0.0151	0.0052	44.0077	2008-07-19		-1
9	0.0063	95.031	0.4984	0.0106	0.0034	2.1261	0.0204	0.0194	0.0063	95.031	2008-07-19		-1
10	0.0045	111.6525	0.4993	0.0172	0.0046	3.4456	0.0111	0.0124	0.0045	111.6525	2008-07-19		-1
11	0.0073	90.2294	0.4967	0.0152	0.0038						2008-07-19		-1
12	0.0071	57.8122	0.4925	0.0158	0.0041	2.1261	0.0204	0.0194	0.0063	95.031	2008-07-19		1
13	0.0081	75.5077	0.4987	0.0427	0.0092						2008-07-19		1
14	0.0034	52.2039	0.495	0.0153	0.0041	3.0	0.0111	0.0124	0.0045	111.6525	2008-07-19		-1
15			0.5034	0.0151	0.0038	3.0	0.0111	0.0124	0.0045	111.6525	2008-07-19		-1
16	0.0084	142.908	0.5077	0.0094	0.0026	1.8483	0.0202	0.0289	0.0084	142.908	2008-07-21		1
17	0.0045	100.2745	0.5058	0.0078	0.0021	1.5352	0.0174	0.0174	0.0045	100.2745	2008-07-22		-1
18	0.0042	82.0989	0.5005	0.0108	0.0034	2.1574	0.0184	0.0151	0.0042	82.0989	2008-07-22		-1
19			0.5015	0.0105	0.0027	2.0979	0.0184	0.0151	0.0042	82.0989	2008-07-22		-1
20			0.4948	0.0117	0.0034	2.3737	0.0184	0.0151	0.0042	82.0989	2008-07-22		-1

secom

Response (Classification)
-1 (Pass) or 1 (Fail)

Ready

File Home Insert Draw Page Layout Formulas Data Review View Tell me what you want to do secom.csv - Excel Chow, Jo Share

VU1 Classification

VJ VK VL VM VN VO VP VQ VR VS VT VU Classification

1 Feature 581 Feature 582 Feature 583 Feature 584 Feature 585 Feature 586 Feature 587 Feature 588 Feature 589 Feature 590 Date Classification

2 0.5005 0.0118 0.0035 2.363 4.4447 0.0096 0.0201 0.006 208.2045 2008-07-19 -1

3 0.006 4.4447 0.0096 0.0201 0.006 208.2045 2008-07-19 -1

4 0.0148 3.1745 0.0584 0.0484 0.0148 82.8602 2008-07-19 1

5 0.0044 2.0544 0.0202 0.0149 0.0044 73.8432 2008-07-19 -1

6 0.0044 99.3032 0.0202 0.0149 0.0044 73.8432 2008-07-19 -1

7 0.0052 3.8276 0.0342 0.0151 0.0052 44.0077 2008-07-19 -1

8 0.0052 2.8515 0.0342 0.0151 0.0052 44.0077 2008-07-19 -1

9 0.0063 2.1261 0.0204 0.0194 0.0063 95.031 2008-07-19 -1

10 0.0045 3.4456 0.0111 0.0124 0.0045 111.6525 2008-07-19 -1

11 0.0073 2.1261 0.0204 0.0194 0.0063 95.031 2008-07-19 -1

12 0.0071 2.1261 0.0204 0.0194 0.0063 95.031 2008-07-19 1

13 0.0081 3.0 0.0111 0.0124 0.0045 111.6525 2008-07-19 1

14 0.0034 3.0 0.0111 0.0124 0.0045 111.6525 2008-07-19 -1

15 0.0034 3.0 0.0111 0.0124 0.0045 111.6525 2008-07-19 -1

16 0.0084 1.8483 0.0202 0.0289 0.0084 142.908 2008-07-21 1

17 0.0045 1.5352 0.0174 0.0174 0.0045 100.2745 2008-07-22 -1

18 0.0042 2.1574 0.0184 0.0151 0.0042 82.0989 2008-07-22 -1

19 0.0042 2.0979 0.0184 0.0151 0.0042 82.0989 2008-07-22 -1

20 0.0042 2.3737 0.0184 0.0151 0.0042 82.0989 2008-07-22 -1

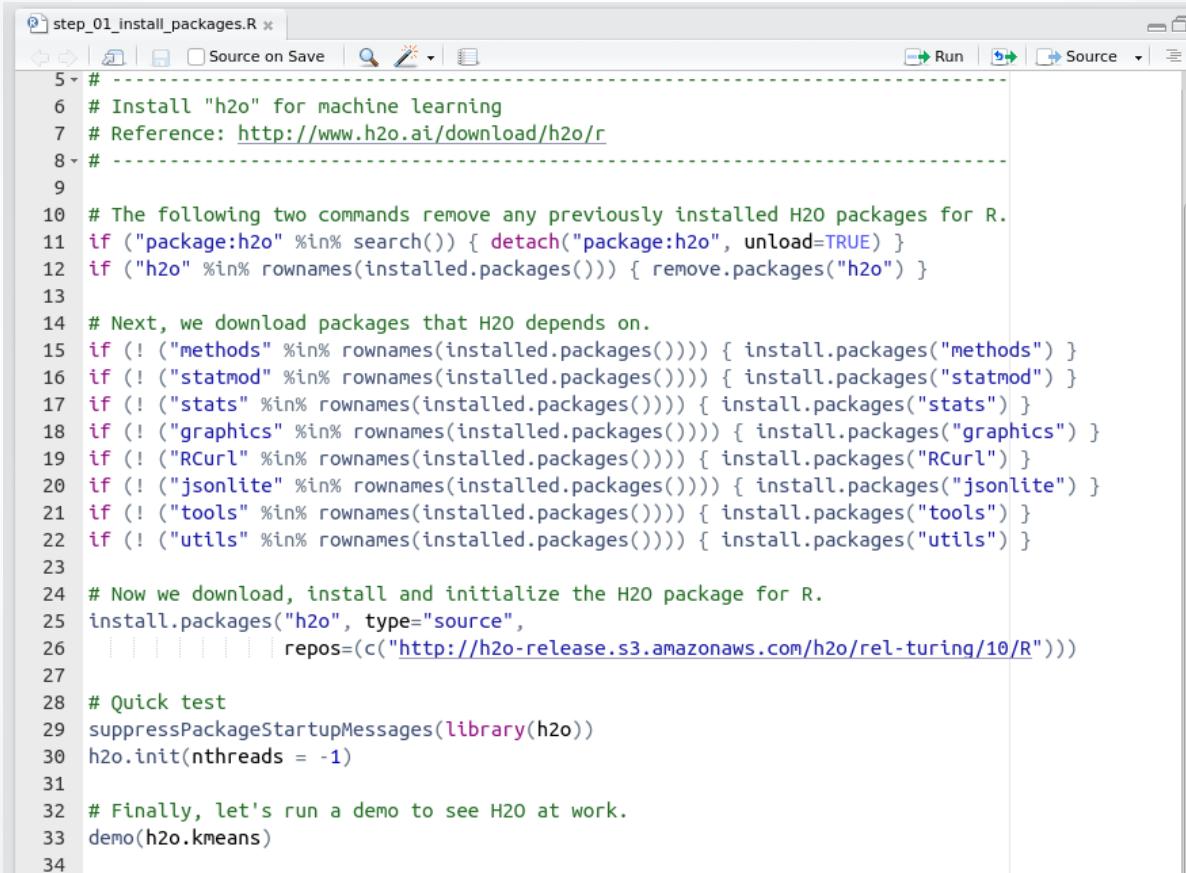
Workshop #1

Example 1: Classification

Step 1: R Packages



step_01_install_packages.R



The screenshot shows the RStudio interface with the script file "step_01_install_packages.R" open. The code is a script for installing the H2O package and its dependencies in R. It includes comments explaining the steps and uses several if statements to handle package removal and installation.

```
# -----
# Install "h2o" for machine learning
# Reference: http://www.h2o.ai/download/h2o/r
# -----
#
# The following two commands remove any previously installed H2O packages for R.
if ("package:h2o" %in% search()) { detach("package:h2o", unload=TRUE) }
if ("h2o" %in% rownames(installed.packages())) { remove.packages("h2o") }

# Next, we download packages that H2O depends on.
if (! ("methods" %in% rownames(installed.packages()))) { install.packages("methods") }
if (! ("statmod" %in% rownames(installed.packages()))) { install.packages("statmod") }
if (! ("stats" %in% rownames(installed.packages()))) { install.packages("stats") }
if (! ("graphics" %in% rownames(installed.packages()))) { install.packages("graphics") }
if (! ("RCurl" %in% rownames(installed.packages()))) { install.packages("RCurl") }
if (! ("jsonlite" %in% rownames(installed.packages()))) { install.packages("jsonlite") }
if (! ("tools" %in% rownames(installed.packages()))) { install.packages("tools") }
if (! ("utils" %in% rownames(installed.packages()))) { install.packages("utils") }

# Now we download, install and initialize the H2O package for R.
install.packages("h2o", type="source",
repos=c("http://h2o-release.s3.amazonaws.com/h2o/rel-turing/10/R"))

# Quick test
suppressPackageStartupMessages(library(h2o))
h2o.init(nthreads = -1)

# Finally, let's run a demo to see H2O at work.
demo(h2o.kmeans)
```

step_01_install_packages.R

Package ‘h2oEnsemble’

```
30 # -----
31 # Install "h2oEnsemble" for model stacking
32 # Reference: https://github.com/h2oai/h2o-3/tree/master/h2o-r/ensemble
33 #
34
35 # Install stable version (1.8)
36 install.packages(
37   "https://h2o-release.s3.amazonaws.com/h2o-ensemble/R/h2oEnsemble_0.1.8.tar.gz",
38   repos = NULL)
39
40 # Quick test
41 library(h2oEnsemble)
```

```
> library(h2oEnsemble)
h2oEnsemble (beta) for H2O >=3.0
Version: 0.1.8
Package created on 2016-03-29
```

Example 1: Classification

Step 2: Exploratory Analysis

step_02_exploratory_analysis.R

Importing SECOM data

```
1 - # -----  
2 # Step 2: Data Exploration  
3 # -----  
4  
5 # Start and connect to a local H2O cluster  
6 library(h2o)  
7 h2o.init(nthreads = -1)  
8  
9 # Import data from a local CSV file  
10 # Source: https://archive.ics.uci.edu/ml/machine-learning-databases/secom/  
11 secom <- h2o.importFile(path = "./data/secom.csv", destination_frame = "secom")  
12  
13 # (Optional) Demo - Importing files using URLs  
14 secom <- h2o.importFile(  
15   path = "https://github.com/woobie/H2O_London_Workshop/raw/master/data/secom.csv",  
16   destination_frame = "secom")  
17  
18 # (Optional) Demo - Converting R data frame into H2O data frame  
19 hdf_iris <- as.h2o(iris)  
20  
21 # (Optional) Turning off progress bar in R  
22 h2o.no_progress()
```

```
java version "1.8.0_72"  
Java(TM) SE Runtime Environment (build 1.8.0_72-b15)  
Java HotSpot(TM) 64-Bit Server VM (build 25.72-b15, mixed mode)  
  
Starting H2O JVM and connecting: ... Connection successful!  
  
R is connected to the H2O cluster:  
  H2O cluster uptime:      2 seconds 721 milliseconds  
  H2O cluster version:    3.10.0.7  
  H2O cluster version age: 7 days, 10 hours and 4 minutes  
  H2O cluster name:       H2O_started_from_R_jofaichow_cow128  
  H2O cluster total nodes: 1  
  H2O cluster total memory: 3.56 GB  
  H2O cluster total cores: 8  
  H2O cluster allowed cores: 8  
  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)  
  
>  
> # Import data from a local CSV file  
> # Source: https://archive.ics.uci.edu/ml/machine-learning-databases/secom/  
> secom <- h2o.importFile(path = "./data/secom.csv", destination_frame = "secom")  
|=====| 100%
```

Optional (different ways to import data)

step_02_exploratory_analysis.R

Basic exploratory analysis

```
25 # Basic exploratory analysis
26 print(dim(secom)) # 1567 x 599
27 print(summary(secom$Classification))
28 # alternatively, use H2O flow to look at data (localhost:54321)
29
30 # Convert Classification to factor
31 secom$Classification <- as.factor(secom$Classification)
32 print(summary(secom$Classification))
```

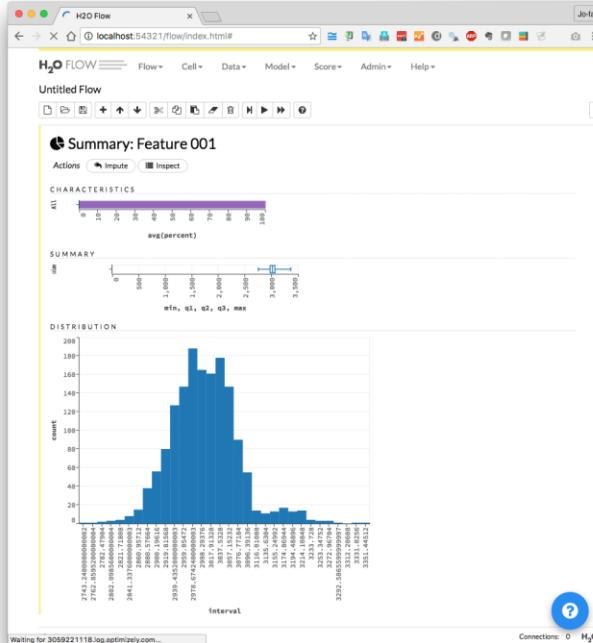
Convert -1 and 1 to categorical value

```
> # Basic exploratory analysis
> print(dim(secom)) # 1567 x 599
[1] 1567 599
> print(summary(secom$Classification))
Classification
Min. :-1.0000
1st Qu.:-1.0000
Median :-1.0000
Mean   :-0.8673
3rd Qu.:-1.0000
Max.   : 1.0000
```

```
> # Convert Classification to factor
> secom$Classification <- as.factor(secom$Classification)
> print(summary(secom$Classification))
Classification
-1:1463
1 : 104
```

Use H2O Flow (localhost:54321)

The screenshot shows the H2O Flow interface with the 'secom' dataset loaded. The top navigation bar includes 'Flow', 'Cell', 'Data', 'Model', 'Score', 'Admin', and 'Help'. Below the title 'Untitled Flow' are buttons for 'View Data', 'Split...', 'Build Model...', 'Predict', 'Download', and 'Export'. The dataset summary shows 1567 rows and 599 columns, with a compressed size of 3MB. A detailed table of column summaries follows, listing various features with their data types, missing values, and statistical properties like min, max, mean, and standard deviation. A 'Actions' column contains links such as 'Convert to enum'. At the bottom, a status bar indicates 'Waiting for 3059221118.log.optimizely.com...' and 'Connections: 0'.



Example 1: Classification

Step 3: Building Models



```

step_03_basic_models.R *
19
20
21 # -----
22 # Define Targets and Features
23 # -----
24
25 target <- "Classification"
26 features <- setdiff(colnames(secom), c("ID", "Classification"))
27
28 print(target)
29 print(features)
30
31
32 # -----
33 # Train H2O models with default value
34 # -----
35
36 # Turn off progress bar (if you want to ...)
37 # h2o.no_progress()
38
39 # GBM
40 model_gbm <- h2o.gbm(x = features, y = target,
41                         training_frame = secom)
42
43 # Random Forest
44 model_drf <- h2o.randomForest(x = features, y = target,
45                                training_frame = secom)
46
47 # Deep Neural Network
48 model_dnn <- h2o.deeplearning(x = features, y = target,
49                                 training_frame = secom)
50
51 # Use R / Flow to look at models
52 print(summary(model_gbm))
53 print(summary(model_drf))
54 print(summary(model_dnn))
55
56 # Look at variable importance
57 print(h2o.varimp(model_gbm))
58 h2o.varimp_plot(model_gbm)
59
60
61
22:1 (Untitled) □ R Script □

```

```

Console /media/SUPPORT/Repo/H2O_London_Workshop/
> print(target)
[1] "Classification"
>
> print(features)
[1] "Feature 001"      "Feature 002"      "Feature 003"      "Feature 004"      "Feature 005"
[6] "Feature 006"      "Feature 007"      "Feature 008"      "Feature 009"      "Feature 010"
[11] "Feature 011"      "Feature 012"      "Feature 013"      "Feature 014"      "Feature 015"
[16] "Feature 016"      "Feature 017"      "Feature 018"      "Feature 019"      "Feature 020"
[21] "Feature 021"      "Feature 022"      "Feature 023"      "Feature 024"      "Feature 025"
[26] "Feature 026"      "Feature 027"      "Feature 028"      "Feature 029"      "Feature 030"
[31] "Feature 031"      "Feature 032"      "Feature 033"      "Feature 034"      "Feature 035"
[36] "Feature 036"      "Feature 037"      "Feature 038"      "Feature 039"      "Feature 040"
[41] "Feature 041"      "Feature 042"      "Feature 043"      "Feature 044"      "Feature 045"
[46] "Feature 046"      "Feature 047"      "Feature 048"      "Feature 049"      "Feature 050"
[51] "Feature 051"      "Feature 052"      "Feature 053"      "Feature 054"      "Feature 055"
[56] "Feature 056"      "Feature 057"      "Feature 058"      "Feature 059"      "Feature 060"
[61] "Feature 061"      "Feature 062"      "Feature 063"      "Feature 064"      "Feature 065"
[66] "Feature 066"      "Feature 067"      "Feature 068"      "Feature 069"      "Feature 070"
[71] "Feature 071"      "Feature 072"      "Feature 073"      "Feature 074"      "Feature 075"
[76] "Feature 076"      "Feature 077"      "Feature 078"      "Feature 079"      "Feature 080"
[81] "Feature 081"      "Feature 082"      "Feature 083"      "Feature 084"      "Feature 085"
[86] "Feature 086"      "Feature 087"      "Feature 088"      "Feature 089"      "Feature 090"
[91] "Feature 091"      "Feature 092"      "Feature 093"      "Feature 094"      "Feature 095"
[96] "Feature 096"      "Feature 097"      "Feature 098"      "Feature 099"      "Feature 100"
[101] "Feature 101"      "Feature 102"      "Feature 103"      "Feature 104"      "Feature 105"
[106] "Feature 106"      "Feature 107"      "Feature 108"      "Feature 109"      "Feature 110"
[111] "Feature 111"      "Feature 112"      "Feature 113"      "Feature 114"      "Feature 115"
[116] "Feature 116"      "Feature 117"      "Feature 118"      "Feature 119"      "Feature 120"
[121] "Feature 121"      "Feature 122"      "Feature 123"      "Feature 124"      "Feature 125"
[126] "Feature 126"      "Feature 127"      "Feature 128"      "Feature 129"      "Feature 130"
[131] "Feature 131"      "Feature 132"      "Feature 133"      "Feature 134"      "Feature 135"
[136] "Feature 136"      "Feature 137"      "Feature 138"      "Feature 139"      "Feature 140"
[141] "Feature 141"      "Feature 142"      "Feature 143"      "Feature 144"      "Feature 145"
[146] "Feature 146"      "Feature 147"      "Feature 148"      "Feature 149"      "Feature 150"
[151] "Feature 151"      "Feature 152"      "Feature 153"      "Feature 154"      "Feature 155"
[156] "Feature 156"      "Feature 157"      "Feature 158"      "Feature 159"      "Feature 160"
[161] "Feature 161"      "Feature 162"      "Feature 163"      "Feature 164"      "Feature 165"
[166] "Feature 166"      "Feature 167"      "Feature 168"      "Feature 169"      "Feature 170"
[171] "Feature 171"      "Feature 172"      "Feature 173"      "Feature 174"      "Feature 175"
[176] "Feature 176"      "Feature 177"      "Feature 178"      "Feature 179"      "Feature 180"
[181] "Feature 181"      "Feature 182"      "Feature 183"      "Feature 184"      "Feature 185"
[186] "Feature 186"      "Feature 187"      "Feature 188"      "Feature 189"      "Feature 190"
[191] "Feature 191"      "Feature 192"      "Feature 193"      "Feature 194"      "Feature 195"
[196] "Feature 196"      "Feature 197"      "Feature 198"      "Feature 199"      "Feature 200"
[201] "Feature 201"      "Feature 202"      "Feature 203"      "Feature 204"      "Feature 205"

```

```
step_03_basic_models.R x
Source on Save Run
19
20
21 # -----
22 # Define Targets and Features
23 #
24
25 target <- "Classification"
26 features <- setdiff(colnames(secom), c("ID", "Classification"))
27
28 print(target)
29 print(features)
30
31
32 # -----
33 # Train H2O models with default value
34 #
35
36 # Turn off progress bar (if you want to ...)
37 # h2o.no_progress()
38
39 # GBM
40 model_gbm <- h2o.gbm(x = features, y = target,
41                         training_frame = secom)
42
43 # Random Forest
44 model_drf <- h2o.randomForest(x = features, y = target,
45                                training_frame = secom)
46
47 # Deep Neural Network
48 model_dnn <- h2o.deeplearning(x = features, y = target,
49                                training_frame = secom)
50
51 # Use R / Flow to look at models
52 print(summary(model_gbm))
53 print(summary(model_drf))
54 print(summary(model_dnn))
55
56 # Look at variable importance
57 print(h2o.varimp(model_gbm))
58 h2o.varimp_plot(model_gbm)
59
60
61
```

Console /media/SUPPORT/Bento/H2O_London_Workshop/

```
> # GBM
> model_gbm <- h2o.gbm(x = features, y = target,
+                         training_frame = secom)
|=====| 100%
Warning message:
In .h2o.startModelJob(algo, params, h2oRestApiVersion) :
  Dropping constant columns: [Feature 516, Feature 234, Feature 233, Feature 236, Feature 235, Feature 510, Feature 238, Feature 513, Feature 237, Feature 515, Feature 514, Feature 193, Feature 192, Feature 195, Feature 194, Feature 230, Feature 232, Feature 231, Feature 529, Feature 244, Feature 365, Feature 401, Feature 400, Feature 006, Feature 403, Feature 402, Feature 405, Feature 404, Feature 241, Feature 482, Feature 243, Feature 242, Feature 180, Feature 179, Feature 459, Feature 050, Feature 053, Feature 450, Feature 331, Feature 452, Feature 330, Feature 451, Feature 191, Feature 070, Feature 190, Feature 506, Feature 505, Feature 508, Feature 507, Feature 509, Feature 465, Feature 464, Feature 467, Feature 466, Feature 227, Feature 502, Feature 504, Feature 503, Feature 463, Feature 187, Feature 462, Feature 399, Feature 277, Feature 398, Feature 315, Feature 314, Feature 316, Feature 150, Feature 395, Feature 397, Feature 396, Feature 329, Date.year, Feature 323, Feature 326... <truncated>
```

```

step_03_basic_models.R *
19
20
21 # -----
22 # Define Targets and Features
23 # -----
24
25 target <- "Classification"
26 features <- setdiff(colnames(secom), c("ID", "Classification"))
27
28 print(target)
29 print(features)
30
31
32 # -----
33 # Train H2O models with default value
34 # -----
35
36 # Turn off progress bar (if you want to ...)
37 # h2o.no_progress()
38
39 # GBM
40 model_gbm <- h2o.gbm(x = features, y = target,
41                         training_frame = secom)
42
43 # Random Forest
44 model_drf <- h2o.randomForest(x = features, y = target,
45                                training_frame = secom)
46
47 # Deep Neural Network
48 model_dnn <- h2o.deeplearning(x = features, y = target,
49                                 training_frame = secom)
50
51 # Use R / Flow to look at models
52 print(summary(model_gbm))
53 print(summary(model_drf))
54 print(summary(model_dnn))
55
56 # Look at variable importance
57 print(h2o.varimp(model_gbm))
58 h2o.varimp_plot(model_gbm)
59
60
61

```

Console /media/SUPPORT/Repo/H2O_London_Workshop/

> print(summary(model_gbm))

Model Details:

=====

H2OBinomialModel: gbm

Model Key: GBM_model_R_1479672080021_139

Model Summary:

	number_of_trees	number_of_internal_trees	model_size_in_bytes	min_depth	max_depth	mean_depth	min_leaves
1	50	50	11781	5	5	5.00000	7
				max_leaves	mean_leaves		
1				20	13.82000		

H2OBinomialMetrics: gbm

** Reported on training data. **

MSE: 0.005638814

RMSE: 0.07509204

LogLoss: 0.03756363

Mean Per-Class Error: 0

AUC: 1

Gini: 1

Confusion Matrix for F1-optimal threshold:

	-1	1	Error Rate
-1	1463	0	0.000000 =0/1463
1	0	104	0.000000 =0/104
Totals	1463	104	0.000000 =0/1567

Maximum Metrics: Maximum metrics at their respective thresholds

	metric	threshold	value	idx
1	max f1	0.399934	1.000000	98
2	max f2	0.399934	1.000000	98
3	max f0points	0.399934	1.000000	98
4	max accuracy	0.399934	1.000000	98
5	max precision	0.960024	1.000000	0
6	max recall	0.399934	1.000000	98
7	max specificity	0.960024	1.000000	0
8	max absolute_mcc	0.399934	1.000000	98
9	max min_per_class_accuracy	0.399934	1.000000	98
10	max mean_per_class_accuracy	0.399934	1.000000	98

Gains/Lift Table: Extract with `h2o.gainsLift(<model>, <data>)` or `h2o.gainsLift(<model>, valid=<T/F>, xval=<T/F>)`

```
step_03_basic_models.R * 
19
20
21 # -----
22 # Define Targets and Features
23 # -----
24
25 target <- "Classification"
26 features <- setdiff(colnames(secom), c("ID", "Classification"))
27
28 print(target)
29 print(features)
30
31
32 # -----
33 # Train H2O models with default value
34 # -----
35
36 # Turn off progress bar (if you want to ...)
37 # h2o.no_progress()
38
39 # GBM
40 model_gbm <- h2o.gbm(x = features, y = target,
41                         | training_frame = secom)
42
43 # Random Forest
44 model_drf <- h2o.randomForest(x = features, y = target,
45                                | training_frame = secom)
46
47 # Deep Neural Network
48 model_dnn <- h2o.deeplearning(x = features, y = target,
49                                | training_frame = secom)
50
51 # Use R / Flow to look at models
52 print(summary(model_gbm))
53 print(summary(model_drf))
54 print(summary(model_dnn))
55
56 # Look at variable importance
57 print(h2o.varimp(model_gbm))
58 h2o.varimp_plot(model_gbm)
59
60
61
```

Console /media/SUPPORT/Bapo/H2O_London_Workshop/

> **betaht(h28.VarImp)**

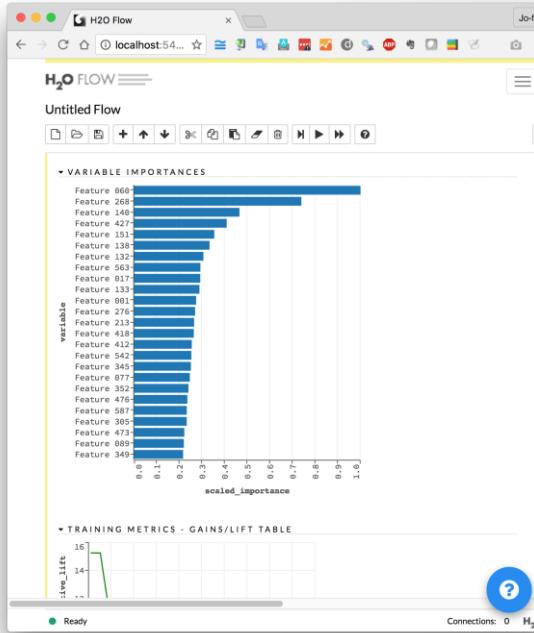
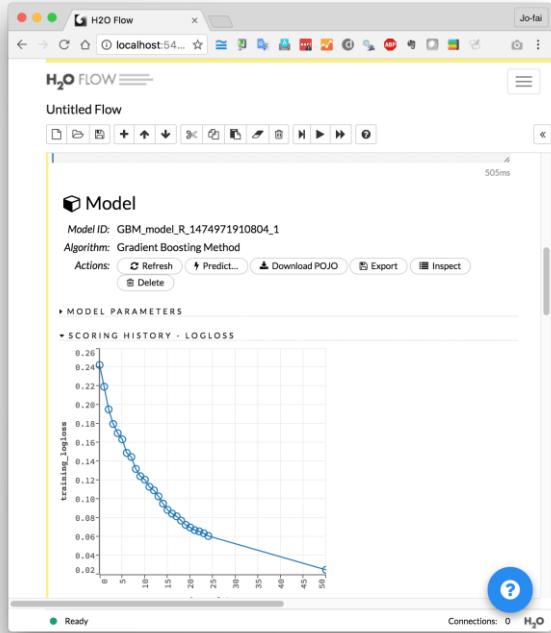
	variable	relative_importance	scaled_importance	percentage
1	Feature_060	21.270756	1.000000	0.067999
2	Feature_065	10.369788	0.487514	0.033140
3	Feature_563	9.856809	0.463397	0.031500
4	Feature_349	9.147681	0.430059	0.029244
5	Feature_030	8.751948	0.411455	0.027975

-

	variable	relative_importance	scaled_importance	percentage
474	Feature 582	0.000000	0.000000	0.000000
475	Feature 587	0.000000	0.000000	0.000000
476	Feature 590	0.000000	0.000000	0.000000
477	Date.month	0.000000	0.000000	0.000000
478	Date.day-of-week	0.000000	0.000000	0.000000
479	Date.hour	0.000000	0.000000	0.000000

7

Use H2O Flow (localhost:54321)



Example 1: Classification

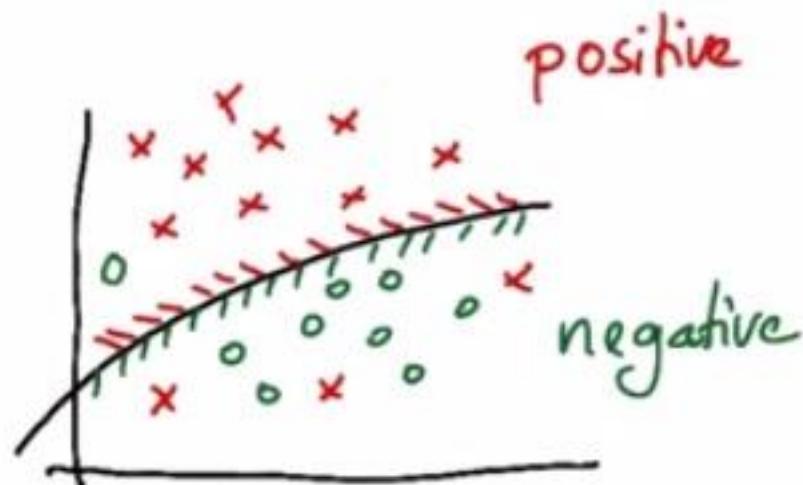
Step 4: Evaluating Models



Confusion Matrix

CONFUSION MATRIX

actual class	
positive	negative
predicted class	positive
positive	9
negative	1
positive	3
negative	8



The Confusion Matrix

		ACTUAL	
		POSITIVE	NEGATIVE
PREDICTED	Positive	TRUE POSITIVE	FALSE POSITIVE Type I Error
	Negative	FALSE NEGATIVE Type II Error	TRUE NEGATIVE



step_04_evaluate_models.R x

```

18
19 # Define Targets and Features
20 target <- "Classification"
21 features <- setdiff(colnames(secom), c("ID", "Classification"))
22
23
24 # -----
25 # Method 1: Split data into training / test
26 #
27
28 # Split
29 # i.e. using 60% of data for training and 40% for test
30 secom_splits <- h2o.splitFrame(data = secom, ratios = 0.6, seed = 1234)
31 secom_train <- secom_splits[[1]]    # optional step
32 secom_test <- secom_splits[[2]]    # optional step
33
34 # Check ratio
35 summary(secom_train$Classification) # 882 : 62 ... % of 1 = 0.07029478
36 summary(secom_test$Classification) # 581 : 42 ... % of 1 = 0.07228916
37
38 # Train a simple Deep Learning model using 60% of data
39 model_dnn1 <- h2o.deeplearning(x = features, y = target,
40                                training_frame = secom_train)
41
42 # Evaluate model performance on unseen (40%) data
43 h2o.performance(model_dnn1, newdata = secom_test)
44
45
46 # -----
47 # Method 2: K-fold Cross-Validation
48 #
49
50 # Train a simple Deep Learning model using 100% of data with n-fold CV
51 model_dnn2 <- h2o.deeplearning(x = features, y = target,
52                                training_frame = secom,
53                                nfolds = 5,
54                                seed = 1234,
55                                fold_assignment = "Stratified")
56
57 # Look at the evaluation results on n-fold CV
58 print(model_dnn2)
59
60
44:1 (Untitled) R Script

```

Console /media/SUPPORT/Repo/H2O_London_Workshop/

```

> h2o.performance(model_dnn1, newdata = secom_test)
H2OBinomialMetrics: deeplearning

MSE:  0.07466812
RMSE:  0.2732547
LogLoss:  0.8965152
Mean Per-Class Error:  0.3538439
AUC:  0.6748217
Gini:  0.3496435

Confusion Matrix for F1-optimal threshold:
      -1   1   Error   Rate
-1   405 176 0.302926  =176/581
 1    17  25 0.404762  =17/42
Totals 422 201 0.309791  =193/623

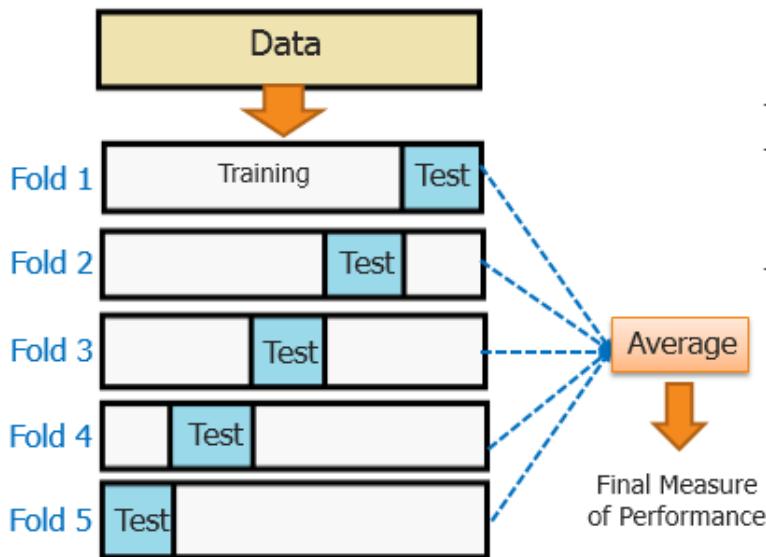
Maximum Metrics: Maximum metrics at their respective thresholds
               metric threshold   value idx
1       max f1  0.000002  0.205761 200
2       max f2  0.000001  0.339426 214
3       max fpoints 0.002693  0.187970 55
4       max accuracy 0.996206  0.930979  0
5       max precision 0.037511  0.190476 20
6       max recall  0.000000  1.000000 399
7       max specificity 0.996206  0.998279  0
8       max absolute_mcc 0.000002  0.156785 200
9       max min_per_class_accuracy 0.000001  0.619048 214
10      max mean_per_class_accuracy 0.000001  0.646873 214

Gains/Lift Table: Extract with `h2o.gainsLift(<model>, <data>)` or `h2o.gainsLift(<model>, valid=<T/F>, xval=<T/F>)`
```

Cross Validation

Cross-Validation (CV)

edureka!



- Technique to validate models/classifiers
- Method to estimate how accurately the model generalizes to unseen data i.e., how well it performs/predicts
- K-fold CV
 - » Most popular
 - » k is typically set to 10
 - » Every sample/record is used both in training and test sets

```

18
19 # Define Targets and Features
20 target <- "Classification"
21 features <- setdiff(colnames(secom), c("ID", "Classification"))
22
23
24 # -----
25 # Method 1: Split data into training / test
26 # -----
27
28 # Split
29 # i.e. using 60% of data for training and 40% for test
30 secom_splits <- h2o.splitFrame(data = secom, ratios = 0.6, seed = 1234)
31 secom_train <- secom_splits[[1]]    # optional step
32 secom_test <- secom_splits[[2]]    # optional step
33
34 # Check ratio
35 summary(secom_train$Classification) # 882 : 62 ... % of 1 = 0.07029478
36 summary(secom_test$Classification) # 581 : 42 ... % of 1 = 0.07228916
37
38 # Train a simple Deep Learning model using 60% of data
39 model_dnn1 <- h2o.deeplearning(x = features, y = target,
40                                training_frame = secom_train)
41
42 # Evaluate model performance on unseen (40%) data
43 h2o.performance(model_dnn1, newdata = secom_test)
44
45
46 # -----
47 # Method 2: K-fold Cross-Validation
48 # -----
49
50 # Train a simple Deep Learning model using 100% of data with n-fold CV
51 model_dnn2 <- h2o.deeplearning(x = features, y = target,
52                                training_frame = secom,
53                                nfolds = 5,
54                                seed = 1234,
55                                fold_assignment = "Stratified")
56
57 # Look at the evaluation results on n-fold CV
58 print(model_dnn2)
59
60
60.1 (Untitled) □ R Script □

```

Console /media/SUPPORT/Repo/H2O_London_Workshop/

H2OBinomialMetrics: deeplearning
 ** Reported on cross-validation data. **
 ** 5-fold cross-validation on training data (Metrics computed for combined holdout predictions) **

MSE: 0.08972877
 RMSE: 0.2995476
 LogLoss: 0.6970678
 Mean Per-Class Error: 0.4119072
 AUC: 0.6058744
 Gini: 0.2117488

Confusion Matrix for F1-optimal threshold:

	-1	1	Error Rate
-1	1144	319	0.218045 = 319/1463
1	63	41	0.605769 = 63/104
Totals	1207	360	0.243778 = 382/1567

Maximum Metrics: Maximum metrics at their respective thresholds

	metric	threshold	value	idx
1	max f1	0.002525	0.176724	317
2	max f2	0.000044	0.295883	391
3	max f0points5	0.020865	0.137755	213
4	max accuracy	0.999999	0.932355	0
5	max precision	0.020865	0.123288	213
6	max recall	0.000000	1.000000	399
7	max specificity	0.999999	0.998633	0
8	max absolute_mcc	0.002525	0.104257	317
9	max min_per_class_accuracy	0.000086	0.574846	387
10	max mean_per_class_accuracy	0.000044	0.598990	391

Gains/Lift Table: Extract with `h2o.gainsLift(<model>, <data>)` or `h2o.gainsLift(<model>, valid=<T/F>, xval=<T/F>)`
 Cross-Validation Metrics Summary:

	mean	sd	cv_1_valid	cv_2_valid	cv_3_valid	cv_4_valid	cv_5_valid
accuracy	0.7087468	0.14202245	0.88782054	0.79591835	0.4781145	0.9235474	0.45833334
auc	0.6190499	0.026494274	0.62117493	0.66482586	0.62324876	0.63509035	0.5509008
err	0.29125318	0.14202245	0.11217949	0.20408164	0.5218855	0.0764526	0.5416667
err_count	88.2	40.267605	35.0	70.0	155.0	25.0	156.0
f0points5	0.19044691	0.063521884	0.21367522	0.17006803	0.12160229	0.3508772	0.09601182
f1	0.201924	0.025374223	0.22222222	0.22222222	0.17989418	0.24242423	0.14285715
f2	0.2723356	0.04121469	0.23148148	0.32051283	0.34552845	0.18518518	0.27896994
lift_top_group	1.308	1.8497913	0.0	0.0	0.0	6.54	0.0
logloss	0.69674987	0.04815367	0.6732423	0.636079	0.641785	0.822235	0.71040815
max_per_class_error	0.65172094	0.08798135	0.7619048	0.54545456	0.5503597	0.84	0.5608856
MCC	0.17012891	0.034879416	0.16248225	0.16829523	0.17033629	0.25242126	0.097109556

Example 1: Classification

Step 5: Tuning Models



step_05_manual_tuning.R

```

23
24 # -----
25 # Train H2O models with default / manual settings
26 #
27
28 # Check out all parameters
29 # ?h2o.gbm
30 # ?h2o.deeplearning
31 # ?h2o.randomForest
32
33 # Deep Learning model with CV and default value
34 model_dnn1 <- h2o.deeplearning(x = features,
35                               y = target,
36                               training_frame = secom,
37                               nfolds = 5,
38                               seed = 1234,
39                               fold_assignment = "Stratified")
40 print(model_dnn1)
41
42 # Deep Learning model with manual settings
43 # ?h2o.deeplearning
44 model_dnn2 <- h2o.deeplearning(x = features,
45                               y = target,
46                               training_frame = secom,
47                               nfolds = 5,
48                               seed = 1234,
49                               fold_assignment = "Stratified",
50                               score_duty_cycle = 1,
51
52                               # Manual tweaks
53                               activation = "RectifierWithDropout",
54                               balance_classes = TRUE,
55                               hidden = c(100, 100, 100),
56                               epochs = 100)
57 print(model_dnn2)
58
59 # Use R / Flow to look at models
60 print(model_dnn1)
61 print(model_dnn2)
62
63 # -----
64 # Making predictions
65 #

```

Console /media/SUPPORT/Repo/H2O_London_Workshop/

H2OBinomialMetrics: deeplearning
** Reported on cross-validation data. **
** 5-fold cross-validation on training data (Metrics computed for combined holdout predictions) **

MSE: 0.08972877
RMSE: 0.2995476
LogLoss: 0.6970678
Mean Per-Class Error: 0.4119072
AUC: 0.6058744
Gini: 0.2117488

Confusion Matrix for F1-optimal threshold:

	-1	1	Error Rate
-1	1144	319	0.218045 =319/1463
1	63	41	0.605769 =63/104
Totals	1207	360	0.243778 =382/1567

Maximum Metrics: Maximum metrics at their respective thresholds

	metric	threshold	value	idx
1	max f1	0.002525	0.176724	317
2	max f2	0.000044	0.295883	391
3	max f0points5	0.020865	0.137755	213
4	max accuracy	0.999999	0.932355	0
5	max precision	0.020865	0.123288	213
6	max recall	0.000000	1.000000	399
7	max specificity	0.999999	0.998633	0
8	max absolute_mcc	0.002525	0.104257	317
9	max min_per_class_accuracy	0.000086	0.574846	387
10	max mean_per_class_accuracy	0.000044	0.598990	391

Gains/Lift Table: Extract with `h2o.gainsLift(<model>, <data>)` or `h2o.liftTable(<model>, valid=<T/F>, xval=<T/F>)`
Cross-Validation Metrics Summary:

	mean	sd	cv_1_valid	cv_2_valid	cv_3_valid	cv_4_valid	cv_5_valid
accuracy	0.7087468	0.14202245	0.88782054	0.79591835	0.4781145	0.9235474	0.45833334
auc	0.6190499	0.026494274	0.62117493	0.66482586	0.62324876	0.63509035	0.5509008
err	0.29125318	0.14202245	0.11217949	0.20408164	0.5218855	0.0764526	0.5416667
err_count	88.2	40.267605	35.0	70.0	155.0	25.0	156.0
f0points5	0.19044691	0.063521884	0.21367522	0.17006803	0.12160229	0.3508772	0.09601182
f1	0.201924	0.025374223	0.22222222	0.22222222	0.17989418	0.24242423	0.14285715
f2	0.2723356	0.04121469	0.23148148	0.32051283	0.34552845	0.18518518	0.27896994
lift_top_group	1.308	1.8497913	0.0	0.0	0.0	6.54	0.0
logloss	0.69674987	0.04815367	0.6732423	0.636079	0.641785	0.822235	0.71040815
max_per_class_error	0.65172094	0.08798135	0.7619048	0.54545456	0.5503597	0.84	0.5608856
MCC	0.17012891	0.034879416	0.16248225	0.16829523	0.17033629	0.25242126	0.097109556

```

step_05_manual_tuning.R* *
27
28 # Check out all parameters
29 # ?h2o.gbm
30 # ?h2o.deeplearning
31 # ?h2o.randomForest
32
33 # Deep Learning model with CV and default value
34 model_dnn1 <- h2o.deeplearning(x = features,
35                               y = target,
36                               training_frame = secom,
37                               nfolds = 5,
38                               seed = 1234,
39                               fold_assignment = "Stratified")
40 print(model_dnn1)
41
42 # Deep Learning model with manual settings
43 # ?h2o.deeplearning
44 model_dnn2 <- h2o.deeplearning(x = features,
45                               y = target,
46                               training_frame = secom,
47                               nfolds = 5,
48                               seed = 1234,
49                               fold_assignment = "Stratified",
50
51                               # Manual tweaks
52                               activation = "RectifierWithDropout",
53                               balance_classes = TRUE,
54                               hidden = c(50, 50, 50),
55                               epochs = 100)
56 print(model_dnn2)
57
58 # Use R / Flow to look at models
59 print(model_dnn1)
60 print(model_dnn2)
61
62 # -----
63 # Making predictions
64 # -----
65
66 yhat <- h2o.predict(model_dnn2, secom)
67 print(head(yhat))
68 print(summary(yhat))
69

```

Console /media/SUPPORT/Repo/H2O_London_Workshop/ ↗

H2OBinomialMetrics: deeplearning
 ** Reported on cross-validation data. **
 ** 5-fold cross-validation on training data (Metrics computed for combined holdout predictions) **

MSE: 0.07248065
 RMSE: 0.2692223
 LogLoss: 0.4506603
 Mean Per-Class Error: 0.3919272
 AUC: 0.6879634
 Gini: 0.3759267

Confusion Matrix for F1-optimal threshold:

	-1	1	Error Rate
-1	1315	148	0.101162 = 148/1463
1	71	33	0.682692 = 71/104
Totals	1386	181	0.139757 = 219/1567

Maximum Metrics: Maximum metrics at their respective thresholds

	metric	threshold	value	idx
1	max f1	0.033625	0.231579	172
2	max f2	0.003561	0.342298	315
3	max f0points5	0.033625	0.199275	172
4	max accuracy	1.000000	0.932993	0
5	max precision	0.931569	0.357143	13
6	max recall	0.000002	1.000000	399
7	max specificity	1.000000	0.999316	0
8	max absolute_mcc	0.003561	0.172113	315
9	max min_per_class_accuracy	0.001171	0.644231	354
10	max mean_per_class_accuracy	0.001171	0.562259	354

Gains/Lift Table: Extract with `h2o.gainsLift(<model>, <data>)` or `h2o.gainsLift(<model>, valid=<T/F>, xval=<T/F>)`

Cross-Validation Metrics Summary:

	mean	sd	cv_1_valid	cv_2_valid	cv_3_valid	cv_4_valid	cv_5_valid
accuracy	0.82608616	0.040651035	0.9230769	0.8600583	0.7744108	0.795107	0.7777778
auc	0.6986671	0.013941814	0.69906723	0.69371283	0.73040515	0.6860926	0.7015411
err	0.17391384	0.040651035	0.07692308	0.13994169	0.22558923	0.20489296	0.2222222
err_count	54.0	11.721775	24.0	48.0	67.0	67.0	64.0
f0points5	0.23526171	0.04821312	0.37037036	0.1923077	0.19543974	0.21885522	0.19933555
f1	0.27503464	0.024445241	0.33333334	0.22580644	0.26373628	0.27795699	0.27272728
f2	0.36008653	0.043195467	0.3030303	0.2734375	0.4054054	0.38690478	0.43165466
lift_top_group	3.269126	2.105467	7.428571	0.0	0.0	3.27	5.647059
logloss	0.44672656	0.05261716	0.42377242	0.50482506	0.35533702	0.55806434	0.39163405
max_per_class_error	0.5077285	0.11780413	0.71428573	0.6818182	0.36842105	0.48	0.29411766
MCC	0.2376755	0.031912867	0.29840672	0.16440463	0.23739722	0.2212242	0.2669447

Comparison: Default vs. Manual

Default Settings

```
H2OBinomialMetrics: deeplearning  
** Reported on cross-validation data. **  
** 5-fold cross-validation on training data
```

```
MSE: 0.08972877  
RMSE: 0.2995476  
LogLoss: 0.6970678  
Mean Per-Class Error: 0.4119072  
AUC: 0.6058744  
Gini: 0.2117488
```

```
Confusion Matrix for F1-optimal threshold:  
-1 1 Error Rate  
-1 1144 319 0.218045 =319/1463  
1 63 41 0.605769 =63/104  
Totals 1207 360 0.243778 =382/1567
```

Manual Tweaks

```
H2OBinomialMetrics: deeplearning  
** Reported on cross-validation data. **  
** 5-fold cross-validation on training data
```

```
MSE: 0.07248065  
RMSE: 0.2692223  
LogLoss: 0.4506603  
Mean Per-Class Error: 0.3919272  
AUC: 0.6879634  
Gini: 0.3759267
```

```
Confusion Matrix for F1-optimal threshold:  
-1 1 Error Rate  
-1 1315 148 0.101162 =148/1463  
1 71 33 0.682692 =71/104  
Totals 1386 181 0.139757 =219/1567
```

Smaller Errors

Fewer false positives

File Edit Code View Plots Session Build Debug Profile Tools Help

/media/SUPPORT/Repo/H2O_London_Workshop - master - RStudio

H2O_London_Workshop

step_05_manual_tuning.R*

```
28 # Check out all parameters
29 # ?h2o.gbm
30 # ?h2o.deeplearning
31 # ?h2o.randomForest
32
33 # Deep Learning model with CV and default value
34 model_dnn1 <- h2o.deeplearning(x = features,
35                                y = target,
36                                training_frame = secom,
37                                nfolds = 5,
38                                seed = 1234,
39                                fold_assignment = "Stratified")
40 print(model_dnn1)
41
42 # Deep Learning model with manual settings
43 # ?h2o.deeplearning
44 model_dnn2 <- h2o.deeplearning(x = features,
45                                y = target,
46                                training_frame = secom,
47                                nfolds = 5,
48                                seed = 1234,
49                                fold_assignment = "Stratified",
50
51                                # Manual tweaks
52                                activation = "RectifierWithDropout",
53                                balance_classes = TRUE,
54                                hidden = c(50, 50, 50),
55                                epochs = 100)
56 print(model_dnn2)
57
58 # Use R / Flow to look at models
59 print(model_dnn1)
60 print(model_dnn2)
61
62 # -----
63 # Making predictions
64 #
65
66 yhat <- h2o.predict(model_dnn2, secom)
67 print(head(yhat, 40))
68 print(summary(yhat))
69
70
```

Console /media/SUPPORT/Repo/H2O_London_Workshop/

```
> print(head(yhat, 40))
   predict      p-1      p1
1 -1 1.000000e+00 1.152322e-15
2 -1 1.000000e+00 2.203204e-16
3  1 1.197242e-05 9.999880e-01
4 -1 1.000000e+00 6.160491e-16
5 -1 1.000000e+00 5.075575e-13
6 -1 1.000000e+00 3.021475e-08
7 -1 1.000000e+00 1.530499e-16
8 -1 1.000000e+00 6.463078e-14
9 -1 1.000000e+00 5.099715e-11
10 -1 1.000000e+00 1.501379e-15
11  1 4.556583e-05 9.999544e-01
12  1 9.744068e-05 9.999026e-01
13 -1 1.000000e+00 7.235294e-17
14 -1 1.000000e+00 1.117404e-26
15  1 4.242277e-05 9.999576e-01
16 -1 1.000000e+00 4.651994e-09
17 -1 1.000000e+00 3.355911e-23
18 -1 1.000000e+00 5.910014e-19
19 -1 1.000000e+00 2.524364e-16
20 -1 1.000000e+00 6.455791e-13
21 -1 1.000000e+00 1.634878e-13
22 -1 1.000000e+00 2.045357e-16
23 -1 1.000000e+00 6.730709e-12
24  1 1.139797e-04 9.998860e-01
25 -1 1.000000e+00 3.814657e-09
26 -1 1.000000e+00 1.002225e-12
27 -1 1.000000e+00 1.170781e-11
28 -1 1.000000e+00 4.475900e-09
29 -1 1.000000e+00 9.521416e-15
30 -1 1.000000e+00 3.137428e-13
31 -1 1.000000e+00 1.220719e-08
32 -1 1.000000e+00 1.262628e-12
33 -1 1.000000e+00 2.133081e-13
34 -1 1.000000e+00 2.116694e-12
35 -1 1.000000e+00 6.076701e-12
36 -1 1.000000e+00 5.224696e-12
37 -1 1.000000e+00 3.816010e-08
38 -1 1.000000e+00 5.701559e-13
39  1 1.443873e-06 9.999986e-01
40 -1 1.000000e+00 1.531729e-10
```

Making Predictions

History Environment Files Plots Spark Packages Help Git Viewer

(Untitled)

Pictures - File Manager

Sun, Nov 20, 2016 8:24 pm

End of Workshop #1

(to be continued ...)



Workshop #1 Recap



Workshop #1 Recap

- Introduction to H2O
 - H2O's Interface
 - Flow (Web) and R
 - MNIST Example
 - Hand-written Digits
 - 28 x 28 Pixels
 - Labels: 0 to 9
- Example 1
 - SECOM Data
 - 591 Features (sensors)
 - One Output
 - Pass = -1
 - Fail = 1
 - Binary Classification

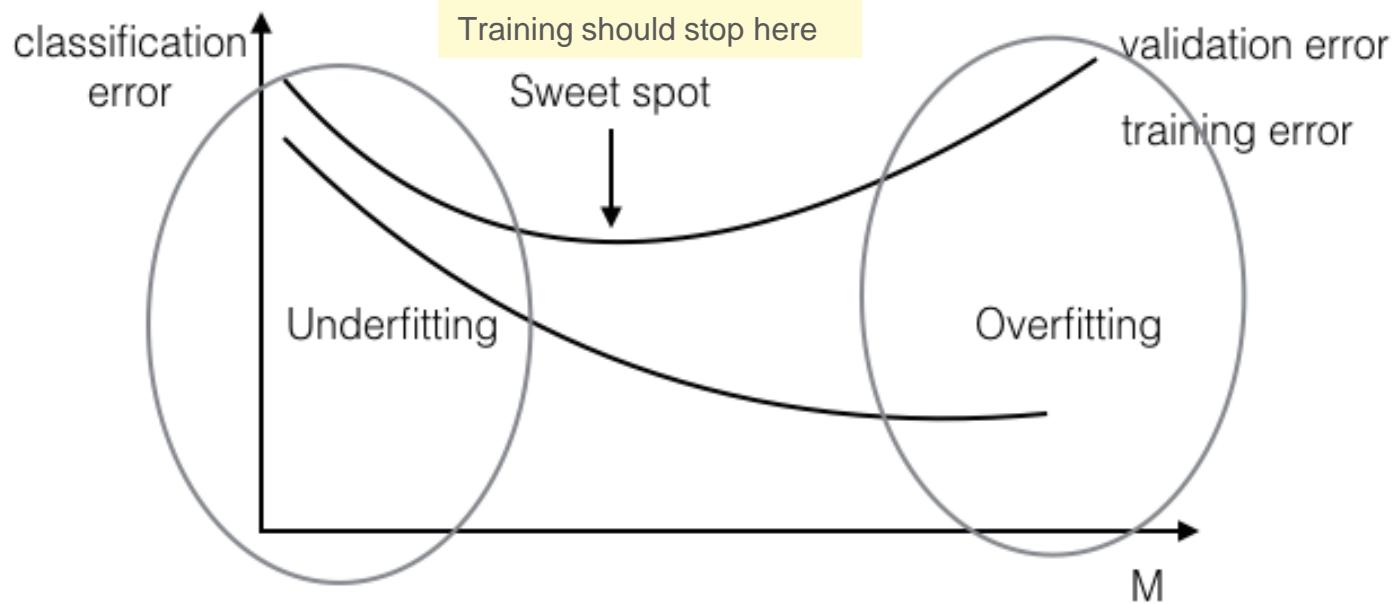
Workshop #2 Agenda

- Example 1 (cont'd)
 - 6. Full/Random Grid Search
 - 7. Model Stacking
 - 8. Saving/Loading Models
 - 9. Deploying Models with H2O Steam
- Example 2
 - Anomaly Detection using Deep Auto Encoder
- Deep Water
 - Motivation
 - Demo

Early Stopping and Grid Search



Underfitting and Overfitting



Grid Search

- Grid Search Example
 - Parameter 1
 - 5, 6 and 7
 - Parameter 2
 - 0.8, 0.9 and 1.0
 - Full Grid Search
 - $3 \times 3 = 9$ Combinations
 - Random Grid Search
 - Randomly picks n out of 9

	Parameter 1	Parameter 2
Model 1	5	0.8
Model 2	5	0.9
Model 3	5	1.0
Model 4	6	0.8
Model 5	6	0.9
Model 6	6	1.0
Model 7	7	0.8
Model 8	7	0.9
Model 9	7	1.0

step_06_full_and_random_grid_search.R

```
# -----  
# Train H2O models with FULL grid search  
# -----  
  
# Define parameters for grid search  
param_dnn <- list(  
  activation = c("Tanh", "Rectifier"),  
  hidden = list(c(50,50), c(50,50,50), c(100,100)),  
  balance_classes = c(TRUE, FALSE)  
)  
  
# DNN with early stopping and FULL grid search  
full_grid_dnn <- h2o.grid(  
  # Core parameters for model training  
  x = features,  
  y = target,  
  training_frame = secom_train,  
  validation_frame = secom_valid,  
  epochs = 100,  
  
  # Parameters for grid search  
  grid_id = "full_grid_dnn",  
  hyper_params = param_dnn,  
  algorithm = "deeplearning",  
  
  # Parameters for early stopping  
  stopping_metric = "logloss",  
  stopping_rounds = 10  
)
```

Define search range

H2O Grid Search Function
(Default = Full Grid Search)

Core parameters

Define H2O algorithm for gird search

Settings for early stopping

step_06_full_and_random_grid_search.R

```
> # Sort models by metric "logloss"
> full_grid_sort <- h2o.getGrid("full_grid_dnn", sort_by = "logloss", decreasing = FALSE)
> print(full_grid_sort)
H2O Grid Details
=====
```

Grid ID: full_grid_dnn

Used hyper parameters:

- activation
- balance_classes
- hidden

Number of models: 12

Number of failed models: 0

Hyper-Parameter Search Summary: ordered by increasing logloss

	activation	balance_classes	hidden	model_ids	logloss
1	Tanh	false	[50, 50]	full_grid_dnn_model_2	0.40711800843533374
2	Rectifier	false	[50, 50, 50]	full_grid_dnn_model_7	0.48874743693102923
3	Rectifier	false	[50, 50]	full_grid_dnn_model_3	0.4982381350341734
4	Tanh	false	[100, 100]	full_grid_dnn_model_10	0.5086143941843354
5	Tanh	false	[50, 50, 50]	full_grid_dnn_model_6	0.6951631654420898
6	Rectifier	false	[100, 100]	full_grid_dnn_model_11	0.7255411856345131
7	Tanh	true	[50, 50]	full_grid_dnn_model_0	0.8135149329182292
8	Rectifier	true	[50, 50]	full_grid_dnn_model_1	0.8196504509768745
9	Rectifier	true	[50, 50, 50]	full_grid_dnn_model_5	0.8967032178246215
10	Tanh	true	[100, 100]	full_grid_dnn_model_8	0.9060965911951484
11	Rectifier	true	[100, 100]	full_grid_dnn_model_9	0.9210162435224971
12	Tanh	true	[50, 50, 50]	full_grid_dnn_model_4	1.1898727562847378

Full Grid Search
Comparing logloss from all 12
models



step_06_full_and_random_grid_search.R

```
# -----  
# Train H2O models with early stopping and RANDOM grid search  
# -----  
  
# Define criteria for random grid search  
search_criteria <- list(  
  strategy = "RandomDiscrete",    # Ask H2O to run random grid search  
  max_models = 5,                 # e.g. only want to test 5 random models  
  max_runtime_secs = 600,          # e.g. only have 600 sec to run this  
  seed = 1234                     # reproducible random parameters combinations  
)  
  
# DNN with early stopping and RANDOM grid search  
random_grid_dnn <- h2o.grid(  
  
  # Core parameters for model training  
  x = features,  
  y = target,  
  training_frame = secom_train,  
  validation_frame = secom_valid,  
  epochs = 100,  
  
  # Parameters for grid search  
  grid_id = "random_grid_dnn",  
  hyper_params = param_dnn,  
  algorithm = "deeplearning",  
  search_criteria = search_criteria, # <- added this for Random Grid Search  
  
  # Parameters for early stopping  
  stopping_metric = "logloss",  
  stopping_rounds = 10  
)
```

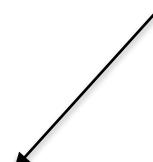
Define search criteria for RANDOM grid search

step_06_full_and_random_grid_search.R

```
> # Sort models by metric "logloss"
> random_grid_sort <- h2o.getGrid("random_grid_dnn",
+                               sort_by = "logloss", decreasing = FALSE)
> print(random_grid_sort)
H2O Grid Details
=====
Grid ID: random_grid_dnn
Used hyper parameters:
- activation
- balance_classes
- hidden
Number of models: 5
Number of failed models: 0

Hyper-Parameter Search Summary: ordered by increasing logloss
  activation balance_classes      hidden          model_ids      logloss
1      Tanh        false    [50, 50] random_grid_dnn_model_0 0.44293730219504907
2 Rectifier      false    [50, 50] random_grid_dnn_model_3 0.5668445883973072
3      Tanh      false [50, 50, 50] random_grid_dnn_model_1 0.5810266574346342
4      Tanh        true   [100, 100] random_grid_dnn_model_4 0.8644619887482393
5 Rectifier      true [50, 50, 50] random_grid_dnn_model_2 0.9745825958041073
> |
```

Only searched five models with
RANDOM grid search



h2oEnsemble

Model Stacking



Common Types of Ensemble Methods

Bagging

- Reduces variance and increases accuracy
 - Robust against outliers or noisy data
 - Often used with Decision Trees (i.e. Random Forest)
-

Boosting

- Also reduces variance and increases accuracy
 - Not robust against outliers or noisy data
 - Flexible – can be used with any loss function
-

Stacking

- Used to ensemble a diverse group of strong learners
- Involves training a second-level machine learning algorithm called a “metalearner” to learn the optimal combination of the base learners

H2O Overview

- H2O Ensemble is a scalable implementation of the Super Learner algorithm for H2O.
- H2O is an open-source, distributed machine learning library written in Java with APIs in R, Python, Scala and REST/JSON.
- Produced by H2O.ai in Mountain View, CA.
- H2O.ai advisers are Trevor Hastie, Rob Tibshirani and Stephen Boyd from Stanford.



step_07_stacking_models.R

```
# -----  
# Train multiple H2O models  
# -----  
  
# Train a Gradient Boosting Machine model  
model_gbm <- h2o.gbm(x = features,  
                      y = target,  
                      training_frame = secom_train,  
                      model_id = "gradient_boosting_machine",  
                      nfolds = 5,  
                      fold_assignment = "Modulo",  
                      keep_cross_validation_predictions = TRUE)  
  
# Train a Distributed Random Forest model  
model_drf <- h2o.randomForest(x = features,  
                               y = target,  
                               training_frame = secom_train,  
                               model_id = "random_forest",  
                               nfolds = 5,  
                               fold_assignment = "Modulo",  
                               keep_cross_validation_predictions = TRUE)  
  
# Train a Deep Learning model  
model_dnn <- h2o.deeplearning(x = features,  
                               y = target,  
                               training_frame = secom_train,  
                               model_id = "deep_learning",  
                               nfolds = 5,  
                               fold_assignment = "Modulo",  
                               keep_cross_validation_predictions = TRUE)
```

Build three different H2O models

step_07_stacking_models.R

```
# Load h2oEnsemble
library(h2oEnsemble)

# Define a list of all models
models <- list(model_gbm, model_drf, model_dnn)

# Define the metalearner
custom_dnn_metalearner <- function(...,
                                      hidden = c(400, 400, 400),
                                      epochs = 1000,
                                      activation = "RectifierWithDropout",
                                      input_dropout_ratio = 0.2,
                                      l1 = 1e-7,
                                      l2 = 1e-7
) {
  h2o.deeplearning.wrapper(...,
                           hidden = hidden,
                           epochs = epochs,
                           activation = activation,
                           input_dropout_ratio = input_dropout_ratio,
                           l1 = l1, l2 = l2
  )
}
metalearner <- "custom_dnn_metalearner"

# Use h2oEnsemble::h2o.stack for model stacking
model_stack <- h2o.stack(models = models,
                         metalearners = metalearner,
                         response_frame = secom_train[, target])

# Evaluate ensemble performance on test data
print(h2o.ensemble_performance(model_stack, secom_test))
```

Using h2o.stack(...) function to
create ensemble

step_07_stacking_models.R

```
> # Evaluate ensemble performance on test data
> print(h2o.ensemble_performance(model_stack, secom_test))
|=====
|=====
|===== 100%
|===== 100%
|===== 100%  
  
Base learner performance, sorted by specified metric:  
      learner      AUC  
3      deep_learning 0.6439226  
2      random_forest 0.6966232  
1 gradient_boosting_machine 0.7041841  
  
H2O Ensemble Performance on <newdata>:  
-----  
Family: binomial  
  
Ensemble performance (AUC): 0.720781083517744
```

Ensemble has better performance
(e.g. higher AUC) when compared to
individual models



Saving / Loading H2O Models



step_08_saving_loading.R

```
# -----  
# Train a H2O Model  
# -----  
  
# Train a GBM  
model_gbm <- h2o.gbm(x = features,  
                      y = target,  
                      model_id = "default_gbm",  
                      training_frame = secom_train)  
  
# -----  
# Saving / Loading H2O Model  
# -----  
  
# Save model to disk  
h2o.saveModel(model_gbm, path = "/models/")  
  
# Load model from disk  
model_from_disk <- h2o.loadModel(path = "./models/default_gbm/")  
print(model_from_disk)
```

Deploying Models with H2O Steam



step_09_deploy_model_with_steam.R

```
# Import data from a R data frame
data(iris)
d_iris <- as.h2o(iris)

# Quick look
head(d_iris)
summary(d_iris)

# Define Targets and Features
target <- "Species"
features <- setdiff(colnames(d_iris), c("Species"))

# -----
# Train a H2O Model
# -----

# Train three basic H2O models
model_dnn <- h2o.deeplearning(x = features,
                                y = target,
                                model_id = "iris_deep_learning",
                                training_frame = d_iris)

model_drf <- h2o.randomForest(x = features,
                               y = target,
                               model_id = "iris_random_forest",
                               training_frame = d_iris)

model_gbm <- h2o.gbm(x = features,
                      y = target,
                      model_id = "iris_gbm",
                      training_frame = d_iris)
```

Build three different models with the classic Iris dataset

STEAM

WELCOME TO H₂O STEAM

Fast, Distributed Data Science For Teams

[Start A New Project](#)

Projects

Services

Clusters

Users

Support

Logout

1. Select H2O Cluster

Select an H2O cluster to import models and datasets from.

CLUSTER	DATASETS	MODELS	
joe	N/A	N/A	Connect

... Or Connect To A New H2O Cluster

Connect to a H2O cluster where your existing models and data sets are located.

localhost [Connect](#)

1. Select H2O Cluster



H2O_started_from_R_joe_eon283
localhost:54321
[use a different cluster](#)

2. Select Dataframe

3. Select Model Category

4. Pick Models To Import

Models in a project must share the same feature set and response column to enable comparison. By default, Steam picks the most optimized model format for you to import. Advanced users can choose your own model type [here](#).

MODEL	RESPONSE COLUMN	CATEGORICAL	
iris_deep_learning	Species	Multinomial	<input checked="" type="checkbox"/> Select for Import
iris_random_forest	Species	Multinomial	<input checked="" type="checkbox"/> Select for Import
iris_gbm	Species	Multinomial	<input checked="" type="checkbox"/> Select for Import

5. Name Project

Create Project

Projects

Services

Clusters

Users

Support

Logout



STEAM

[Home](#) > Projects

PROJECTS

[CREATE NEW PROJECT](#)

All Projects

Steam Iris Demo
Multinomial
2016-11-23 23:04

 [Projects](#) [Services](#) [Clusters](#) [Users](#) [Support](#) [Logout](#)

STEAM < Projects

Steam ...

- Models
- Deployment
- Configurations
- Collaborators
- ?

Home > Projects > 5 > Models

MODELS

IMPORT MODELS

filter models

F	MODEL	MSE	LOGLOSS	R ²	ACTIONS
	iris_random_forest Created at: 2016-11-23 11:04:38 Num of Observations: 150 Cluster: H2O_started_from_R_joe_eon283	0.034834	0.122869	0.947749	view model details (click) label as deploy model delete model
	iris_gbm Created at: 2016-11-23 11:04:40 Num of Observations: 150 Cluster: H2O_started_from_R_joe_eon283	0.002838	0.018819	0.995744	view model details label as deploy model delete model
	iris_deep_learning Created at: 2016-11-23 11:04:37 Num of Observations: 150 Cluster: H2O_started_from_R_joe_eon283	0.127301	0.577885	0.809048	view model details label as deploy model delete model

1 - 3 of 3 models

MODELS

IMPORT MODELS

filter models

MODEL

iris_random_forest

Created at: 2016-11-23
Num of Observations: 1
Cluster: H2O_started_fr

iris_gbm

Created at: 2016-11-23
Num of Observations: 1
Cluster: H2O_started_fr

iris_deep_learning

Created at: 2016-11-23
Num of Observations: 1
Cluster: H2O_started_fr

1 - 3 of 3 models

DEPLOY IRIS_GBM

CONFIGURE SERVICE

Steam automatically selects a port that's not in use based on the port range set by your admin.

Service name

steam_iris

Preprocessing Script

None (Default)

Deploy

Cancel

STEAM < Projects

Steam ...

- Models
- Deployment
- Configurations
- Collaborators

?

192.168.1.80:41788

localhost:9000/#/projects/5/deployment?_k=0nrwx3

Home > Projects > 5 > Deployment

DEPLOYMENT

UPLOAD NEW PACKAGE

DEPLOYED SERVICES PACKAGING

steam_iris @ 192.168.1.80:41788 started	✖ Stop Service
Model 7	
Status OK	

localhost:9001#/project x Steam :: Prediction Service x

192.168.1.80:41788

Prediction Service Steam

Select input parameters, OR enter your own custom query string to predict

MODEL INPUT PARAMETERS

Parameters

1. Sepal.Length	1.2	
2. Sepal.Width	0.6	
3. Petal.Length	0.8	
4. Petal.Width	1.1	

Query String

The parameters above gets automatically built into a REST API query string. You can also input your own string if that's easier for you.

`http://192.168.1.80:41788/predict? Sepal.Length=1.2&Sepal.Width=0.6&Petal.Length=0.8&Petal.Width=1.1`

PREDICTION RESULTS

Model Predictions

Predicting setosa

Index	Labels	Probability
0	setosa	0.7998
1	versicolor	0.1513
2	virginica	0.0489

Model Runtime Stats

Service started	2016-11-23 23:10:20 UTC
Uptime	28 s

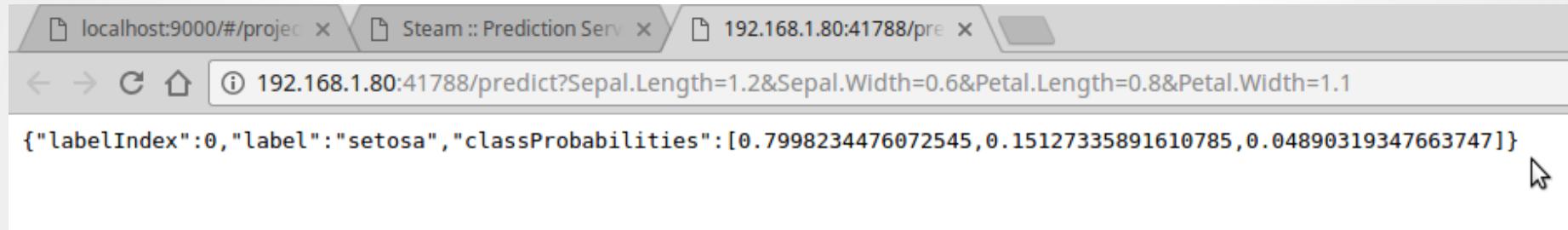
BATCH PREDICTION *OPTIONAL

Select a Batch JSON file

PREDICT **CLEAR**

MORE STATS

Wed, Nov 23, 2016 11:11 pm



A screenshot of a web browser window. The address bar shows the URL `192.168.1.80:41788/predict?Sepal.Length=1.2&Sepal.Width=0.6&Petal.Length=0.8&Petal.Width=1.1`. The main content area displays a JSON response:

```
{"labelIndex":0,"label":"setosa","classProbabilities":[0.7998234476072545,0.15127335891610785,0.04890319347663747]}
```

The Classic REST API Service

Thanks!

- Organisers & Contributors
 - Ajit Jaokar
 - Sibanjan Das
- Slides & Code
 - [bit.ly/
h2o_iot_workshop1](https://bit.ly/h2o_iot_workshop1)
- Key Resources
 - docs.h2o.ai
 - github.com/h2oai/h2o-meetups
- Contact
 - joe@h2o.ai
 - [@matlabulous](https://twitter.com/matlabulous)
 - github.com/woobe