H2O for Internet of Things

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

Data Science Milan
Politecnico di Milano
10th October, 2016

Agenda

- First Talk (25 mins)
  - About H2O.ai
  - Demo
    - A Simple Classification Task
    - H2O’s Web Interface
  - Why H2O?
    - Our Community
    - Our Customers
  - What’s Next?
    - New H2O Features
- Second Talk (25 mins)
  - H2O for IoT
    - Predictive Maintenance
    - Anomaly Detection
    - H2O’s R Interface
- Third Talk (25 mins)
  - Deep Water
  - Demo
    - H2O + mxnet on GPU
    - H2O’s Python Interface

Data and Code

- Please go to bit.ly/h2o_milan_1
- subfolders
    - iot_use_case_1
    - iot_use_case_2

Use Case 1

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:

Authors: Michael McCann, Adrian Johnston

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

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 = pass
    - 1 = fail
- Size: 1567 Samples

secom.csv - Excel

Features (Numeric)

ID (excluded from modeling)

	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 0
2	1	3030.93	2564	2187.7333	1411.1265	1.3602	100	97.6133	0.1242	1.5005	0.0162	-0.00
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	[illegible]	[illegible]	[illegible]	[illegible]	1.4436	0.0041	0.00
5	4	2988.72	2479.9	2199.0333	909.7926	[illegible]	[illegible]	[illegible]	[illegible]	1.4882	-0.0124	-0.00
6	5	3032.24	2502.87	2233.3667	1326.52	[illegible]	[illegible]	[illegible]	[illegible]	1.5031	-0.0031	-0.00
7	6	2946.25	2432.84	2233.3667	1326.52	1.5334	100	100.3967	0.1235	1.5287	0.0167	0.00
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
10	9	2967.68	2600.47	2248.9	1004.4692	0.7884	100	106.24	0.1185	1.5358	0.0111	-0.00
11	10	3016.11	2428.37	2248.9	1004.4692	0.7884	100	106.24	0.1185	1.5381	0.0159	0.00
12	11	2994.05	2548.21	2195.1222	1046.1468	1.3204	100	103.34	0.1223	1.5144	-0.019	0.00
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	-0.00
16	15	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	100	104.8489	0.1197	1.4474	0.0144	-0.01
17	16	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	100	104.8489	0.1197	1.5465	0.025	-0.00
18	17	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	100	104.8078	0.1207	1.4368	0.015	-0.00
19	18	3032.73	2517.79	2270.2556	1258.4558	1.395	100	104.8078	0.1207	1.5537	0.022	-0.00
20	19	3040.34	2501.16	2207.3889	962.5317	1.2043	100	104.0311	0.121	1.5481	-0.0367	0.00

secom

Features (Numeric)

Response (Classification)

-1 (Pass) or 1 (Fail)

	VJ	VK	VL	VM	VN	VO	VP	VQ	VR	VS	VT	VU
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					2008-07-19	-1
3	0.006	[illegible]	[illegible]	[illegible]	[illegible]	4.4447	0.0096	0.0201	0.006	208.2045	2008-07-19	-1
4	0.0148	[illegible]	[illegible]	[illegible]	[illegible]	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	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	2008-07-19	-1
12	0.0071	57.8122	0.4925	0.0158	0.0041	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	2008-07-19	1
13	0.0081	75.5077	0.4987	0.0427	0.0092	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	2008-07-19	1
14	0.0034	52.2039	0.495	0.0153	0.0041	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	2008-07-20	-1
15			0.5034	0.0151	0.0038	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	2008-07-21	-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

Use Case 1: Predictive Maintenance

Step 1: R Packages

step_01_install_packages.R

Package ‘h2o’

```r
# ---------------------------------------------------------------------------
# Step 1: Install R Packages for Workshop
# ---------------------------------------------------------------------------

# ---------------------------------------------------------------------------
# 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.
pkgs <- c("methods","statmod","stats","graphics","RCurl","jsonlite","tools","utils")
for (pkg in pkgs) {
  if (! (pkg %in% rownames(installed.packages()))) { install.packages(pkg) }
}

# 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/7/R")))

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

```
> suppressPackageStartupMessages(library(h2o))
> h2o.init(nthreads = -1)
 Connection successful!

R is connected to the H2O cluster:
    H2O cluster uptime:         1 minutes 891 milliseconds
    H2O cluster version:        3.10.0.7
    H2O cluster version age:    6 days
    H2O cluster name:           H2O_started_from_R_jofaichow_ayn543
    H2O cluster total nodes:    1
    H2O cluster total memory:   3.28 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)
```

Use Case 1: Predictive Maintenance

Step 2: Exploratory Analysis

step_02_exploratory_analysis.R

Importing SECOM data

```r
# ---------------------------------------------------------------------------
# Step 2: Data Exploration
# ---------------------------------------------------------------------------

# Start and connect to a local H2O cluster
library(h2o)
h2o.init(nthreads = -1)

# 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")

# (Optional) Demo - Importing files using URLs
secom <- h2o.importFile(
  path = "https://github.com/woobe/H2O_London_Workshop/raw/master/data/secom.csv",
  destination_frame = "secom")

# (Optional) Demo - Converting R data frame into H2O data frame
hdf_iris <- as.h2o(iris)

# (Optional) Turning off progress bar in R
h2o.no_progress()
```

Optional (different ways to import data)

```
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%
```

step_02_exploratory_analysis.R

Basic exploratory analysis

```r
# Basic exploratory analysis
print(dim(secom)) # 1567 x 599
print(summary(secom$Classification))
# alternatively, use H2O flow to look at data (localhost:54321)

# Convert Classification to factor
secom$Classification <- as.factor(secom$Classification)
print(summary(secom$Classification))
```

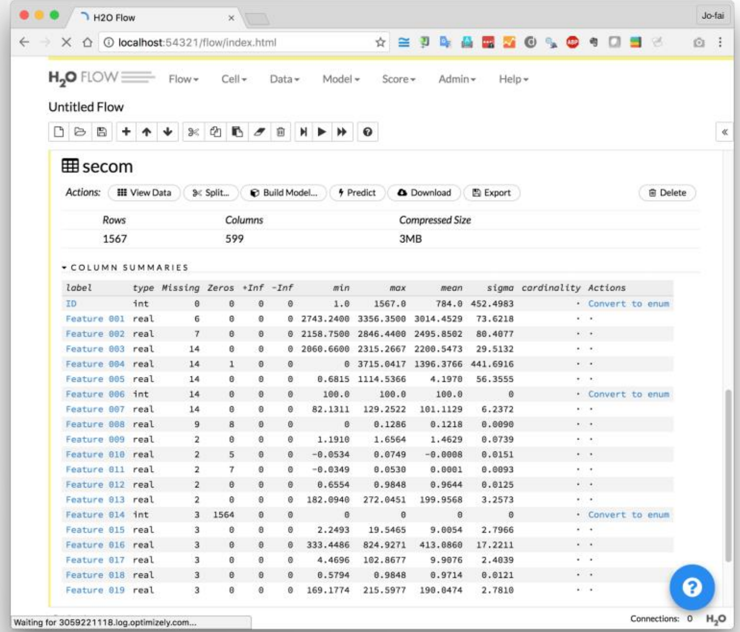
Convert -1 and 1 to categorical value

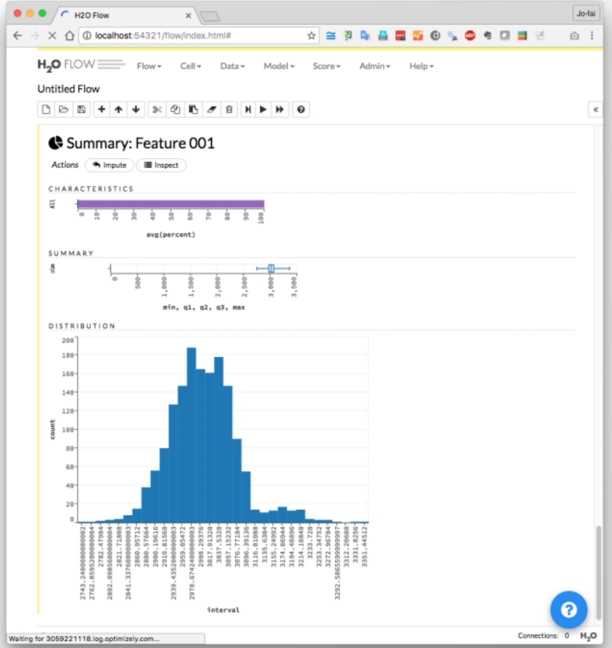
```
> # Basic exploratory analysis
> print(dim(secom)) # 1567 x 599
[1] 1567  599
> print(summary(secom$Classification))
 Classification
 Min.   :-1.0000
 1st Qu.:-1.0000
 Median :-1.0000
 Mean   :-0.8673
 3rd Qu.:-1.0000
 Max.   : 1.0000
```

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

Note: Imbalance dataset (only 104 fails)

Use H2O Flow (localhost:54321)





Use Case 1: Predictive Maintenance

Step 3: Building & Evaluating Models

step_03_basic_models.R

Define features & target

```r
# ---------------------------------------------------------------------------
# Loading data (same as previous steps)
# ---------------------------------------------------------------------------

# Start and connect to a local H2O cluster
library(h2o)
h2o.init(nthreads = -1)

# 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")

# Convert Classification to factor
secom$Classification <- as.factor(secom$Classification)


# ---------------------------------------------------------------------------
# Define Targets and Features
# ---------------------------------------------------------------------------

target <- "Classification"
features <- setdiff(colnames(secom), c("ID", "Classification"))

print(target)
print(features)
```

```
> print(target)
[1] "Classification"
> print(features)
  [1] "Feature 001"      "Feature 002"      "Feature 003"      "Feature 004"
  [5] "Feature 005"      "Feature 006"      "Feature 007"      "Feature 008"
  [9] "Feature 009"      "Feature 010"      "Feature 011"      "Feature 012"
 [13] "Feature 013"      "Feature 014"      "Feature 015"      "Feature 016"
 [17] "Feature 017"      "Feature 018"      "Feature 019"      "Feature 020"
 [21] "Feature 021"      "Feature 022"      "Feature 023"      "Feature 024"
 [25] "Feature 025"      "Feature 026"      "Feature 027"      "Feature 028"
 [29] "Feature 029"      "Feature 030"      "Feature 031"      "Feature 032"
 [33] "Feature 033"      "Feature 034"      "Feature 035"      "Feature 036"
 [37] "Feature 037"      "Feature 038"      "Feature 039"      "Feature 040"
 [41] "Feature 041"      "Feature 042"      "Feature 043"      "Feature 044"
 [45] "Feature 045"      "Feature 046"      "Feature 047"      "Feature 048"
 [49] "Feature 049"      "Feature 050"      "Feature 051"      "Feature 052"
 [53] "Feature 053"      "Feature 054"      "Feature 055"      "Feature 056"
 [57] "Feature 057"      "Feature 058"      "Feature 059"      "Feature 060"
 [61] "Feature 061"      "Feature 062"      "Feature 063"      "Feature 064"
 [65] "Feature 065"      "Feature 066"      "Feature 067"      "Feature 068"
 [69] "Feature 069"      "Feature 070"      "Feature 071"      "Feature 072"
 [73] "Feature 073"      "Feature 074"      "Feature 075"      "Feature 076"
 [77] "Feature 077"      "Feature 078"      "Feature 079"      "Feature 080"
 [81] "Feature 081"      "Feature 082"      "Feature 083"      "Feature 084"
 [85] "Feature 085"      "Feature 086"      "Feature 087"      "Feature 088"
 [89] "Feature 089"      "Feature 090"      "Feature 091"      "Feature 092"
 [93] "Feature 093"      "Feature 094"      "Feature 095"      "Feature 096"
 [97] "Feature 097"      "Feature 098"      "Feature 099"      "Feature 100"
[101] "Feature 101"      "Feature 102"      "Feature 103"      "Feature 104"
[105] "Feature 105"      "Feature 106"      "Feature 107"      "Feature 108"
```

step_03_basic_models.R

Split data with a random seed

```r
# ---------------------------------------------------------------------------
# Split data into training / test
# ---------------------------------------------------------------------------

# Split
# i.e. using 60% of data for training and 40% for test
secom_splits <- h2o.splitFrame(data = secom, ratios = 0.6, seed = 1234)
secom_train <- secom_splits[[1]]
secom_test  <- secom_splits[[2]]

# Check
summary(secom_train$Classification) # 882 : 62 ... % of 1 = 0.07029478
summary(secom_test$Classification) # 581 : 42 ... % of 1 = 0.07228916
```

```
> summary(secom_train$Classification)
 Classification
 -1:882
 1 : 62
```

```
> summary(secom_test$Classification)
 Classification
 -1:581
 1 : 42
```

Classification 1 samples ≈ 7%

step_03_basic_models.R

Train H2O models with default values

```r
# ---------------------------------------------------------------------------
# Train H2O models with default value
# ---------------------------------------------------------------------------

# Turn off progress bar (if you want to ...)
# h2o.no_progress()

# GBM
model_gbm <- h2o.gbm(x = features, y = target,
                     training_frame = secom_train)

# Random Forest
model_drf <- h2o.randomForest(x = features, y = target,
                              training_frame = secom_train)

# Deep Neural Network
model_dnn <- h2o.deeplearning(x = features, y = target,
                              training_frame = secom_train)
```

```
> # GBM
> model_gbm <- h2o.gbm(x = features, y = target,
+                      training_frame = secom_train)
  |==========================================================================| 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 479, Feature 515, Feature 514, Feature 193, Featur
e 192, Feature 195, Feature 194, Feature 075, Feature 230, Feature 232, Feature 231, Feature 529, Feat
ure 244, Feature 365, Feature 401, Feature 400, Feature 006, Feature 403, Feature 402, Feature 405, Fe
ature 404, Feature 241, Feature 482, Feature 243, Feature 242, Feature 180, Feature 179, Feature 459,
Feature 050, Feature 053, Feature 450, Feature 210, Feature 331, Feature 452, Feature 330, Feature 451
, Feature 191, Feature 070, Feature 190, Feature 506, Feature 505, Feature 508, Feature 507, Feature 5
09, Feature 465, Feature 343, Feature 464, Feature 467, Feature 466, Feature 227, Feature 348, Feature
502, Feature 504, Feature 503, Feature 463, Feature 187, Feature 462, Feature 399, Feature 277, Featur
e 398, Feature 315, Feature 314, Feature 316, Feature 150, Feature 395, Feature 39 [... truncated]
```

H2O automatically ignores
Columns with constant values

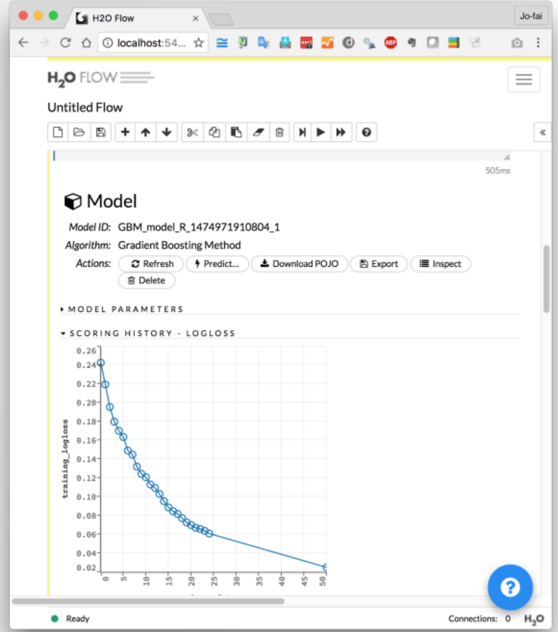
step_03_basic_models.R

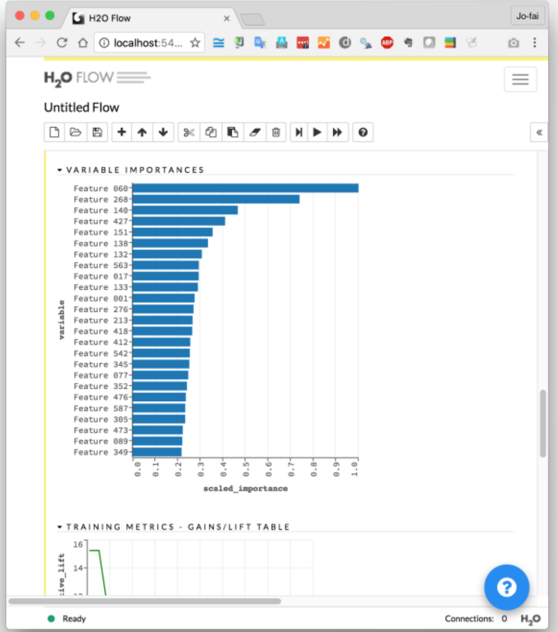
summary(model_xxx)

```
> print(summary(model_gbm))
Model Details:
==============

H2OBinomialModel: gbm
Model Key:  GBM_model_R_1474971910804_1
Model Summary:
  number_of_trees number_of_internal_trees model_size_in_bytes min_depth max_depth mean_depth
1              50                       50               10880         5         5    5.00000
  min_leaves max_leaves mean_leaves
1          6         21    12.38000

H2OBinomialMetrics: gbm
** Reported on training data. **

MSE:  0.002598174
RMSE:  0.05097229
LogLoss:  0.0248643
Mean Per-Class Error:  0
AUC:  1
Gini:  1

Confusion Matrix for F1-optimal threshold:
        -1  1    Error    Rate
-1     882  0 0.000000  =0/882
1        0 62 0.000000   =0/62
Totals 882 62 0.000000  =0/944

Maximum Metrics: Maximum metrics at their respective thresholds
                        metric threshold    value idx
1                       max f1  0.628627 1.000000  61
2                       max f2  0.628627 1.000000  61
3                 max f0point5  0.628627 1.000000  61
4                 max accuracy  0.628627 1.000000  61
5                max precision  0.956650 1.000000   0
6                   max recall  0.628627 1.000000  61
7              max specificity  0.956650 1.000000   0
8             max absolute_mcc  0.628627 1.000000  61
9   max min_per_class_accuracy  0.628627 1.000000  61
10 max mean_per_class_accuracy  0.628627 1.000000  61
```

```
Scoring History:
            timestamp   duration number_of_trees training_rmse training_logloss training_auc
1 2016-09-27 11:25:31  0.021 sec               0       0.24772          0.24231      0.50000
2 2016-09-27 11:25:32  0.722 sec               1       0.24010          0.21926      0.83663
3 2016-09-27 11:25:32  1.001 sec               2       0.22980          0.19523      0.92679
4 2016-09-27 11:25:32  1.178 sec               3       0.22041          0.17966      0.94964
5 2016-09-27 11:25:32  1.320 sec               4       0.21468          0.16997      0.96445
  training_lift training_classification_error
1       1.00000                       0.93432
2       9.13548                       0.08581
3      12.18065                       0.04449
4      13.70323                       0.03496
5      15.22581                       0.03708

---
             timestamp   duration number_of_trees training_rmse training_logloss training_auc
21 2016-09-27 11:25:34  3.500 sec              20       0.12174          0.06986      0.99996
22 2016-09-27 11:25:34  3.631 sec              21       0.11797          0.06696      1.00000
23 2016-09-27 11:25:35  3.726 sec              22       0.11645          0.06571      1.00000
24 2016-09-27 11:25:35  3.841 sec              23       0.11373          0.06369      1.00000
25 2016-09-27 11:25:35  3.967 sec              24       0.10979          0.06082      1.00000
26 2016-09-27 11:25:37  6.596 sec              50       0.05097          0.02486      1.00000
   training_lift training_classification_error
21      15.22581                       0.00106
22      15.22581                       0.00000
23      15.22581                       0.00000
24      15.22581                       0.00000
25      15.22581                       0.00000
26      15.22581                       0.00000

Variable Importances: (Extract with `h2o.varimp`)
=================================================

Variable Importances:
     variable relative_importance scaled_importance percentage
1 Feature 060            9.594806          1.000000   0.050880
2 Feature 268            7.089089          0.738847   0.037592
3 Feature 140            4.466067          0.465467   0.023683
4 Feature 427            3.924058          0.408977   0.020809
5 Feature 151            3.397748          0.354124   0.018018
```

Use H2O Flow (localhost:54321)





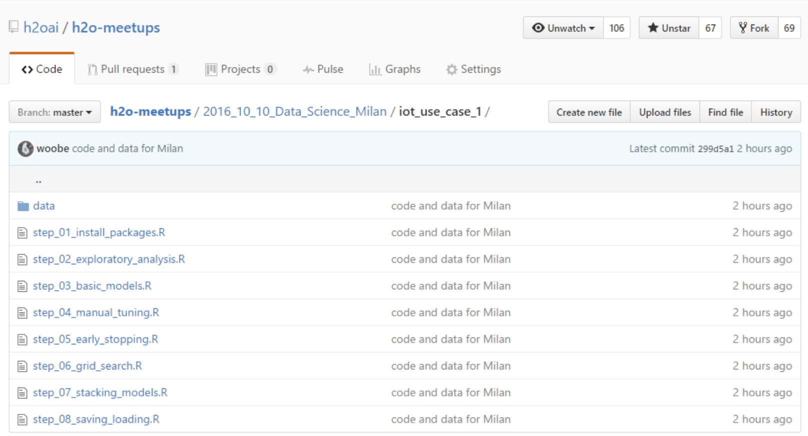
step_03_basic_models.R

Evaluate models with test data

```r
# ----------------------------------------------------
# Evaluate model performance on unseen data
# ----------------------------------------------------

h2o.performance(model_gbm, newdata = secom_test)
h2o.performance(model_drf, newdata = secom_test)
h2o.performance(model_dnn, newdata = secom_test)
```

```
> h2o.performance(model_gbm, newdata = secom_test)
H2OBinomialMetrics: gbm

MSE:  0.06408139
RMSE:  0.253143
LogLoss:  0.2678549
Mean Per-Class Error:  0.2928858
AUC:  0.7041841
Gini:  0.4083682

Confusion Matrix for F1-optimal threshold:
        -1   1    Error     Rate
-1     379 202 0.347676  =202/581
1       10  32 0.238095   =10/42
Totals 389 234 0.340289  =212/623

Maximum Metrics: Maximum metrics at their respective thresholds
                        metric threshold    value idx
1                       max f1  0.016032 0.231884 201
2                       max f2  0.016032 0.398010 201
3                 max f0point5  0.053221 0.196078  63
4                 max accuracy  0.492634 0.930979   0
5                max precision  0.244674 0.333333   5
6                   max recall  0.005088 1.000000 378
7              max specificity  0.492634 0.998279   0
8             max absolute_mcc  0.016032 0.214472 201
9   max min_per_class_accuracy  0.016481 0.659208 196
10 max mean_per_class_accuracy  0.016032 0.707114 201
```

Advanced Procedures

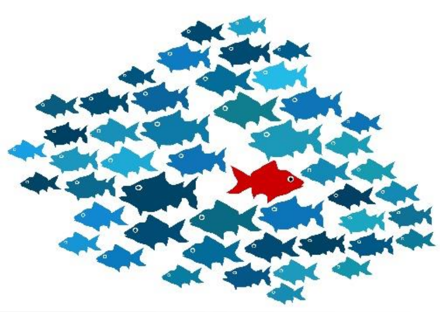


- Step 4 – Manual Tuning
- Step 5 – Early Stopping
- Step 6 – Grid Search
- Step 7 – Stacking Models (“h2oEnsemble”)
- Step 8 – Saving/Loading Models
- Please try them out later (bit.ly/h2o_milan_1)

Use Case 2:
Anomaly Detection

Anomaly (Outlier) Detection

- Definition
  - Identification of items, events or observations which do not conform to an expected pattern or other items in a dataset.



- Use Cases
  - Bank Fraud
  - Monitoring Manufacturing Lines
  - Machine Learning
    - Separate dataset and build different models

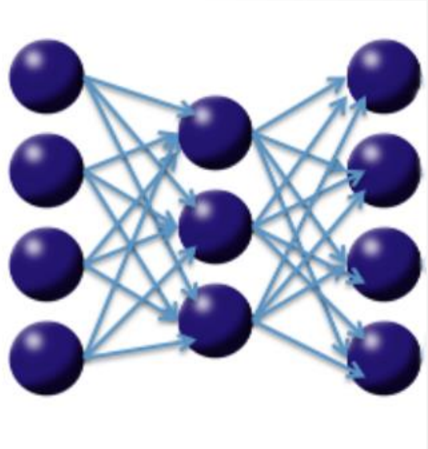
Photo credit: www.dbta.com

Deep Autoencoder for Anomaly Detection

- Consider the following three-layer neural network with one hidden layer and the same number of input neurons (features) as output neurons.

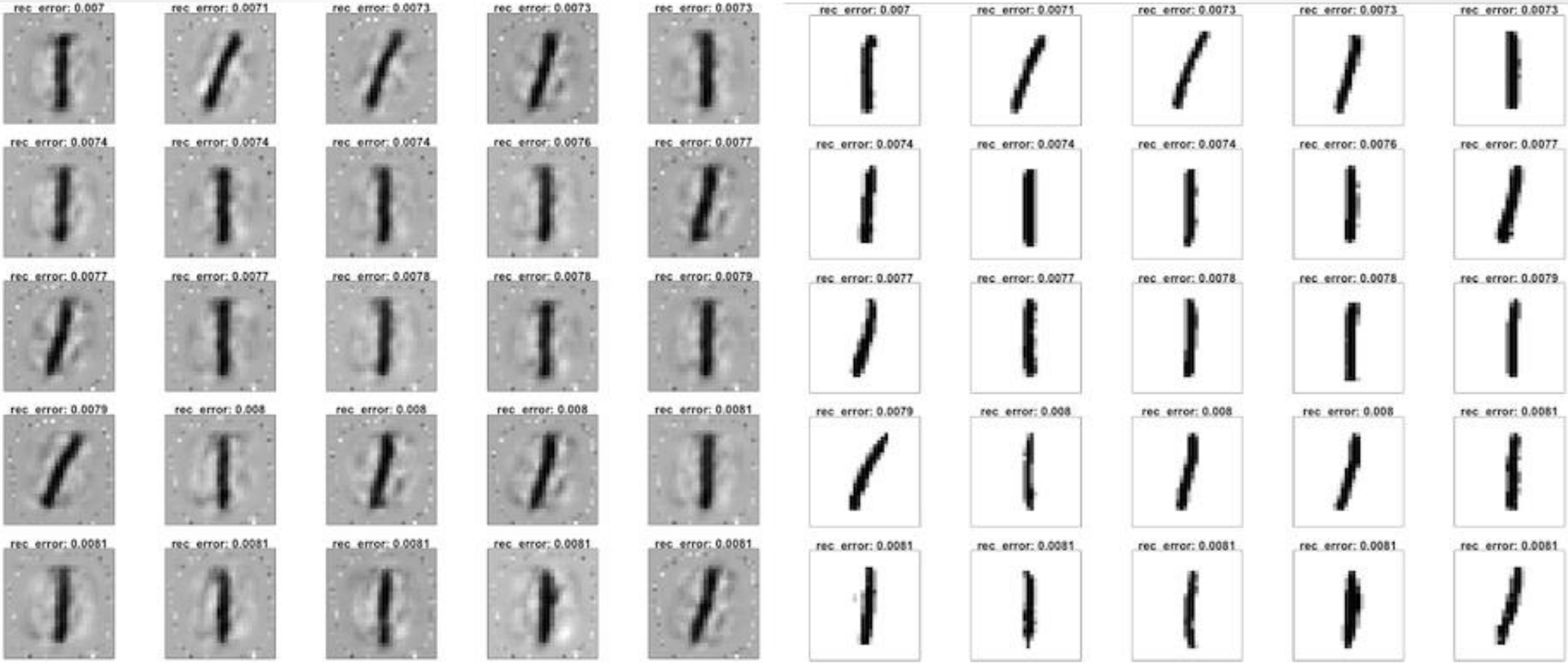
- The loss function is the mean squared error (MSE) between the input and the output. Hence, the network is forced to learn the identity via a nonlinear, reduced representation of the original data.
  - e.g. High MSE = potential outliers

- Such an algorithm is called a **deep autoencoder**.



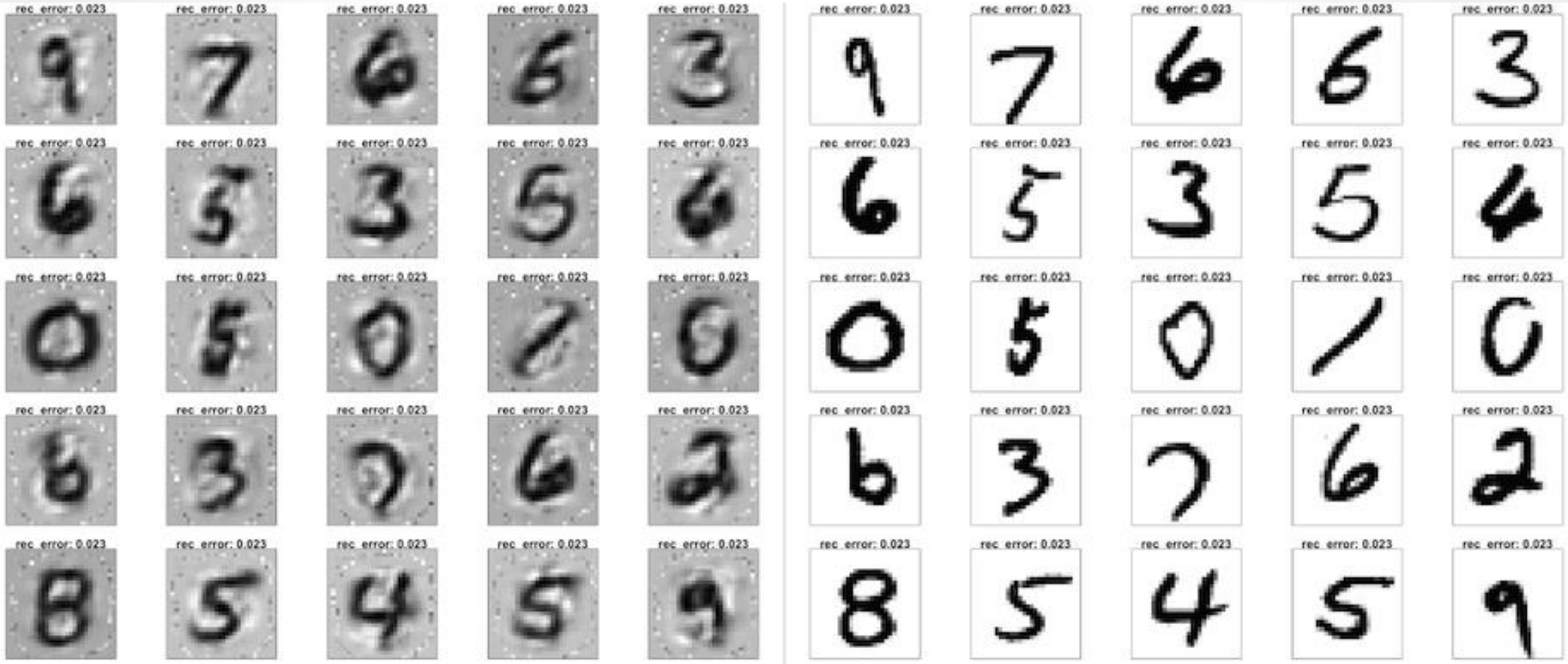
https://github.com/h2oai/h2o-training-book/blob/master/hands-on_training/anomaly_detection.md

MNIST Example – The Good Ones



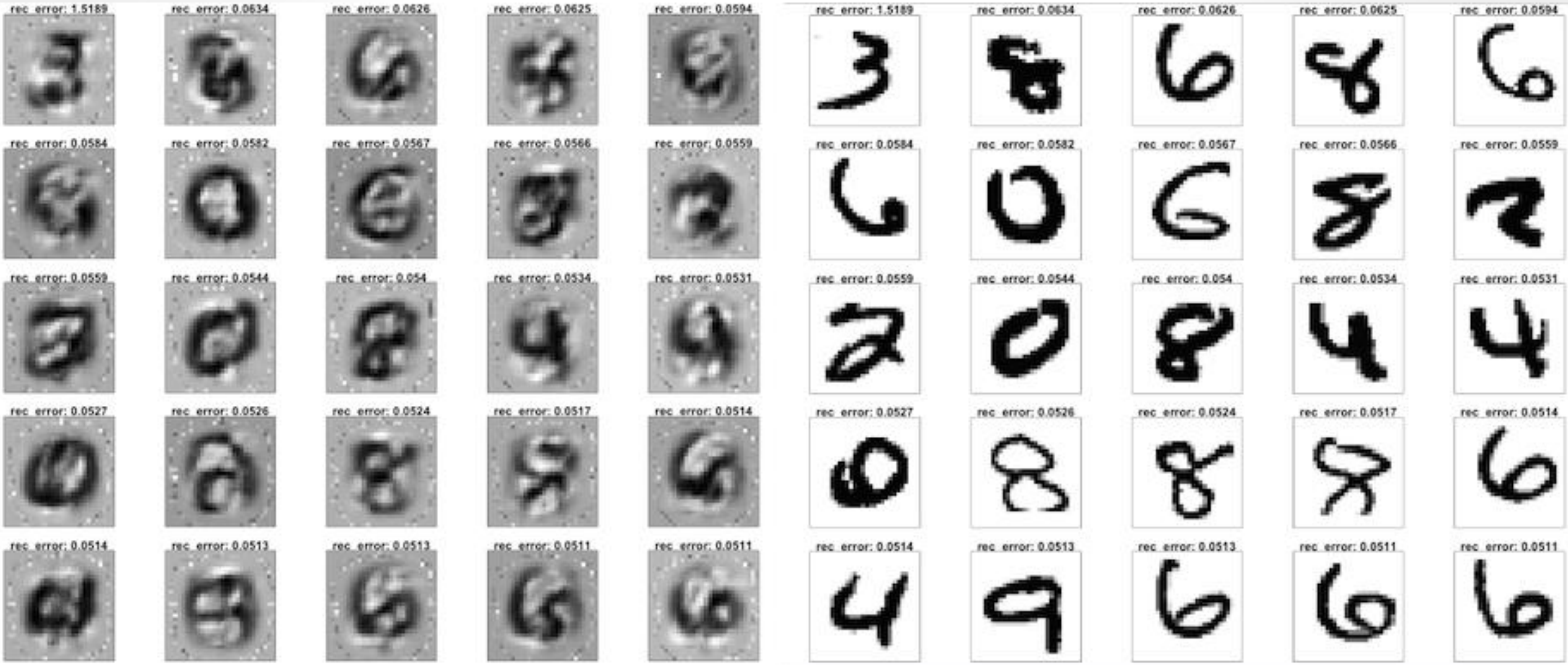
Samples with Low Mean Squared Error (MSE)

MNIST Example – The Bad Ones



Samples with High Mean Squared Error (MSE)

MNIST Example – The Ugly Ones



Samples with Highest Mean Squared Error (MSE)

use_case_2_anomaly_detection.R

```r
# ---------------------------------------------------------
# Step 8: Using Deep Learning for Anomaly Detection
# ---------------------------------------------------------

# Start and connect to a local H2O cluster
library(h2o)
h2o.init(nthreads = -1)

# Import data from a local CSV file
mtcar <- read.csv("./data/auto_design.csv")
mtcar$gear <- as.factor(mtcar$gear)
mtcar$carb <- as.factor(mtcar$carb)
mtcar$cyl  <- as.factor(mtcar$cyl)
mtcar$vs   <- as.factor(mtcar$vs)
mtcar$am   <- as.factor(mtcar$am)
mtcar$ID   <- 1:nrow(mtcar)

# Print it out
print(mtcar)

# Convert R data frame into H2O data frame
h2o_mtcar  <- as.h2o(mtcar)
```

```
> print(mtcar)
```

	X	mpg	cyl	disp	hp	drat	wt	qsec	vs	am	gear	carb	ID
1	Mazda RX4	21.0	6	160.0	110	3.90	2.620	16.46	0	1	4	4	1
2	Mazda RX4 Wag	21.0	6	160.0	110	3.90	2.875	17.02	0	1	4	4	2
3	Datsun 710	22.8	4	108.0	93	3.85	2.320	18.61	1	1	4	1	3
4	Hornet 4 Drive	21.4	6	258.0	110	3.08	3.215	19.44	1	0	3	1	4
5	Hornet Sportabout	18.7	8	360.0	175	3.15	3.440	17.02	0	0	3	2	5
6	Valiant	18.1	6	225.0	105	2.76	3.460	20.22	1	0	3	1	6
7	Duster 360	14.3	8	360.0	245	3.21	3.570	15.84	0	0	3	4	7
8	Merc 240D	24.4	4	146.7	62	3.69	3.190	20.00	1	0	4	2	8
9	Merc 230	22.8	4	140.8	95	3.92	3.150	22.90	1	0	4	2	9
10	Merc 280	19.2	6	167.6	123	3.92	3.440	18.30	1	0	4	4	10
11	Merc 280C	17.8	6	167.6	210	800.00	900.000	1000.00	1	0	4	4	11
12	Merc 450SE	16.4	8	275.8	180	3.07	4.070	17.40	0	0	3	3	12
13	Merc 450SL	17.3	8	275.8	180	3.07	3.730	17.60	0	0	3	3	13
14	Merc 450SLC	15.2	8	275.8	180	3.07	3.780	18.00	0	0	3	3	14
15	Cadillac Fleetwood	10.4	8	472.0	205	2.93	5.250	17.98	0	0	3	4	15
16	Lincoln Continental	10.4	8	460.0	215	3.00	5.424	17.82	0	0	3	4	16
17	Chrysler Imperial	14.7	8	440.0	230	3.23	5.345	17.42	0	0	3	4	17
18	Fiat 128	32.4	4	780.0	2100	400.00	200.000	700.00	1	1	4	1	18
19	Honda Civic	80.4	10	75.7	100	4.93	1.615	150.52	1	1	4	2	19
20	Toyota Corolla	33.9	4	71.1	65	4.22	1.835	19.90	1	1	4	1	20
21	Toyota Corona	21.5	4	120.1	97	3.70	2.465	20.01	1	0	3	1	21
22	Dodge Challenger	15.5	8	318.0	150	2.76	3.520	16.87	0	0	3	2	22
23	AMC Javelin	15.2	8	304.0	150	3.15	3.435	17.30	0	0	3	2	23
24	Camaro Z28	13.3	8	350.0	245	3.73	3.840	15.41	0	0	3	4	24
25	Pontiac Firebird	19.2	8	400.0	175	3.08	3.845	17.05	0	0	3	2	25
26	Fiat X1-9	27.3	4	79.0	66	4.08	1.935	18.90	1	1	4	1	26
27	Porsche 914-2	26.0	4	120.3	91	4.43	2.140	16.70	0	1	5	2	27
28	Lotus Europa	30.4	4	95.1	113	3.77	1.513	16.90	1	1	5	2	28
29	Ford Pantera L	15.8	8	351.0	264	4.22	3.170	14.50	0	1	5	4	29
30	Ferrari Dino	19.7	6	900.0	700	3.62	200.770	150.50	0	1	5	6	30
31	Maserati Bora	15.0	8	301.0	335	3.54	3.570	14.60	0	1	5	8	31
32	Volvo 142E	21.4	4	121.0	109	4.11	2.780	18.60	1	1	4	2	32

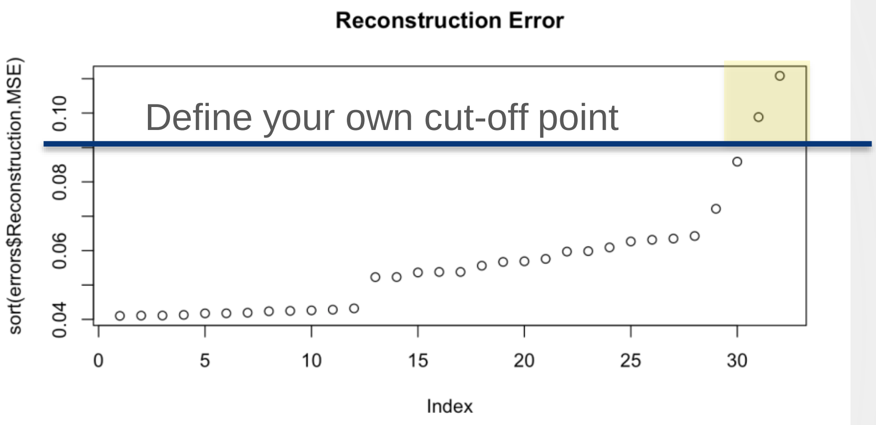
use_case_2_anomaly_detection.R

Build a Deep Autoencoder

```r
# ---------------------------------------------------------------------
# Training an unsupervised deep neural network with autoencoder
# ---------------------------------------------------------------------

# Use a bigger DNN
model <- h2o.deeplearning(x = 1:10,
                          training_frame = h2o_mtcar,
                          autoencoder = TRUE,
                          activation = "RectifierWithDropout",
                          hidden = c(50, 50, 50),
                          epochs = 100)

# Calculate reconstruction errors (MSE)
errors <- h2o.anomaly(model, h2o_mtcar, per_feature = FALSE)
print(errors)
errors <- as.data.frame(errors)

# Plot
plot(sort(errors$Reconstruction.MSE), main = "Reconstruction Error")

# Outliers (define 0.09 as the cut-off point)
row_outliers <- which(errors > 0.09) # based on plot above
mtcar[row_outliers,]
```

Look at the MSE

Define cut-off




```
> row_outliers <- which(errors > 0.09) # based on plot above
> mtcar[row_outliers,]
            X  mpg cyl  disp   hp drat  wt qsec vs am gear carb ID
11 Merc 280C 17.8   6 167.6  210  800 900 1000  1  0    4    4 11
18  Fiat 128 32.4   4 780.0 2100  400 200  700  1  1    4    1 18
```

Outliers identified

End of Second Talk – Thanks!

- Data Science Milan
- Gianmario Spacagna
- Politecnico di Milano

- Resources
  - bit.ly/h2o_milan_1
  - www.h2o.ai
  - docs.h2o.ai
- Contact
  - joe@h2o.ai
  - @matlabulous
  - github.com/woobe