H2O.ai AutoML in KNIME for classification problems

a powerful auto-machine-learning framework (v 1.02)

https://forum.knime.com/u/mlauber71/summary

https://hub.knime.com/mlauber71/spaces/Public/latest/automl/

kn_automl_h2o_classification_python

kn_automl_h2o_classification_r

Online article and discussion:

https://forum.knime.com/t/h2o-ai-automl-in-knime-for-classification-problems/20923?u=mlauber71

It features various models like Random Forest or XGBoost along with Deep Learning. It has wrappers for R and Python but also could be used from KNIME. The results will be written to a folder and the models will be stored in MOJO format to be used in KNIME (as well as on a Big Data cluster via Sparkling Water).

One major parameter to set is the running time the model has to test various models and do some hyper parameter optimization as well. The best model of each round is stored, and some graphics are produced to see the results.

Results are interpreted thru various statistics and model characteristics are stored in and Excel und TXT file as well as in PNG graphics you can easily re-use in presentations and to give your winning models a visual inspection.

Also, you could use the Metanode “Model Quality Classification - Graphics” to evaluate other binary classification models.

Python and R in KNIME
(if you are using the R wrapper you can just skip the Python part)

In order for H2O.ai to work you will have to install Python and the necessary packages:
h2o, pandas, numpy, os, time, datetime, sys
optional: pyarrow

KNIME Python Integration Installation Guide
https://docs.knime.com/latest/python_installation_guide/index.html

Python and Anaconda and KNIME – the short story
https://forum.knime.com/t/problem-with-setting-a-python-deep-learning-environment/19477/2?u=mlauber71

Downloading and installing H2O
http://docs.h2o.ai/h2o/latest-stable/h2o-docs/downloading.html

```
-----------------------------------
# make sure you have the necessary Python packages installed

import numpy as np # linear algebra
import os # accessing directory structure
import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)

print("pandas (pd) version: ", pd.__version__)
print("numpy (np) version", np.__version__)

# http://strftime.org'
import time
import datetime as dt

# conda install -c conda-forge pyarrow=0.15.1
import pyarrow.parquet as pq

# pip install -f http://h2o-release.s3.amazonaws.com/h2o/latest_stable_Py.html h2o
import h2o

from pandas import ExcelWriter
from pandas import ExcelFile

import sys
```

Install R alongside KNIME on Windows and MacOS
https://forum.knime.com/t/install-r-alongside-knime-on-windows-and-macos/13287

R and Rtools
https://forum.knime.com/t/how-to-import-tables-from-docx-documents-via-r-snippet/19284/10

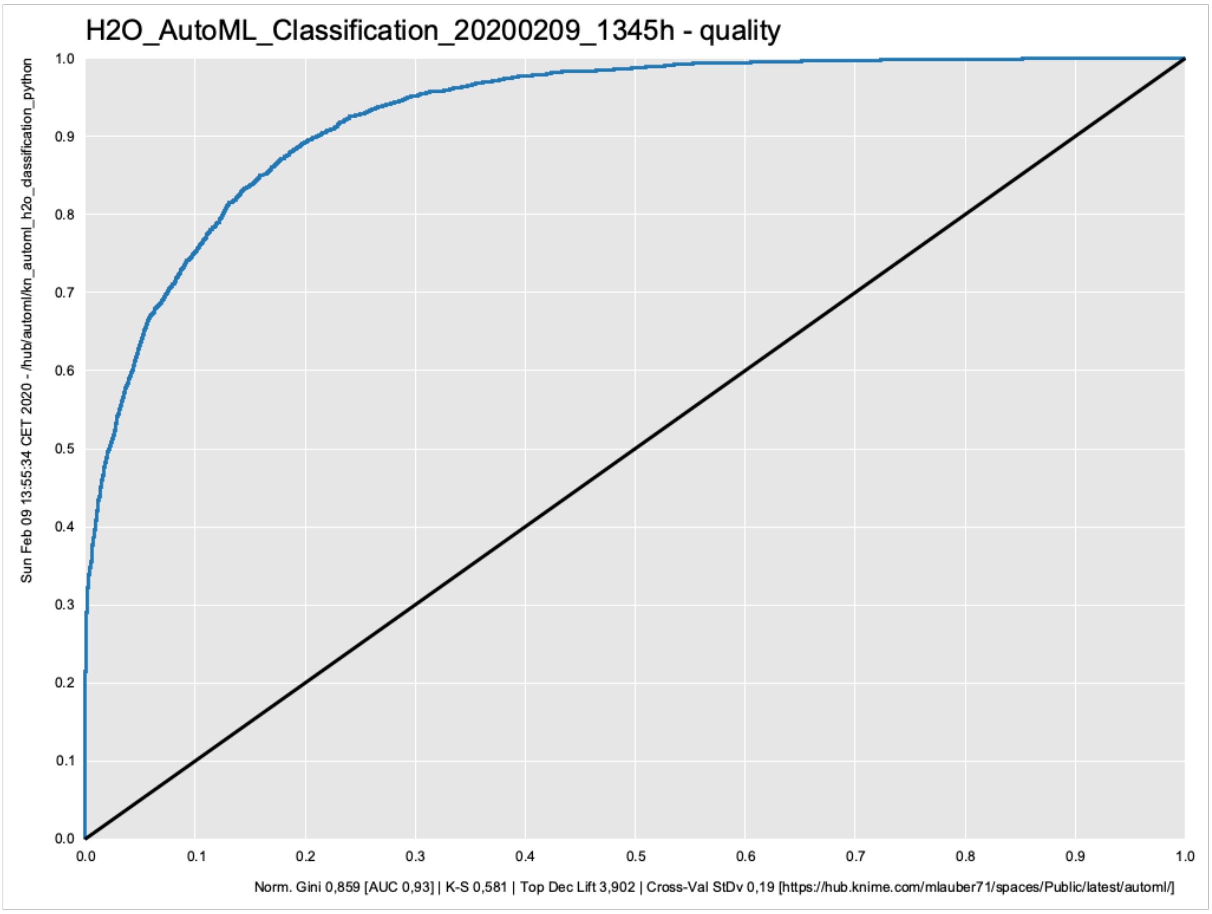
RServe 1.8.6 on MacOSX
https://forum.knime.com/t/installing-rserve-1-8-6-on-macos-10-15-catalina/20909/6?u=mlauber71

R packages needed:
ggplot2, lift, reshape2

If you use the R wrapper you will need the **h2o** package and the arrow package if you plan on using the pure R script in the /script/ subfolder

http://docs.h2o.ai/h2o/latest-stable/h2o-docs/downloading.html

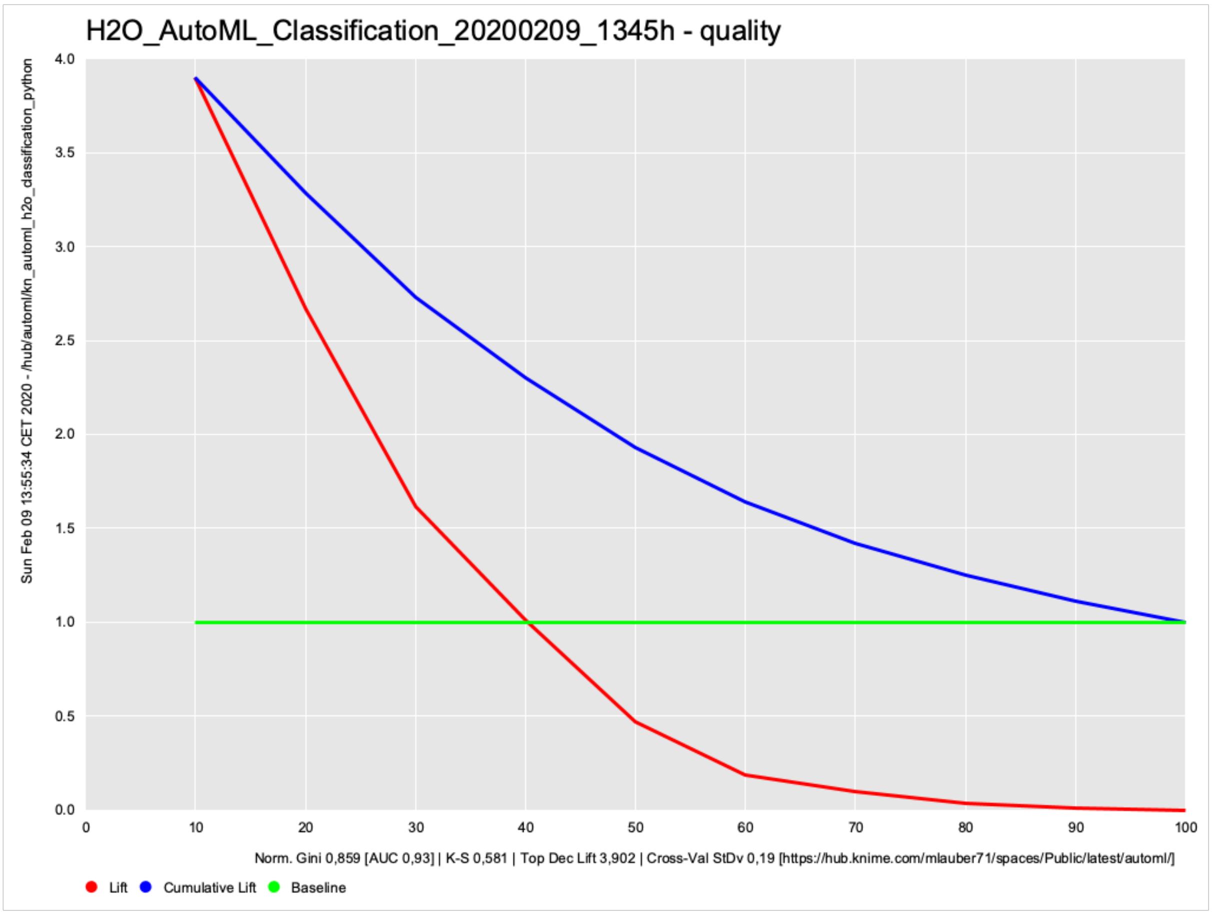
ROC Curve and Gini coefficient



A classic ROC (receiver operating characteristic) curve with statistics like Gini coefficient measuring the 'un-equality' – which is what we want to maximize in this case

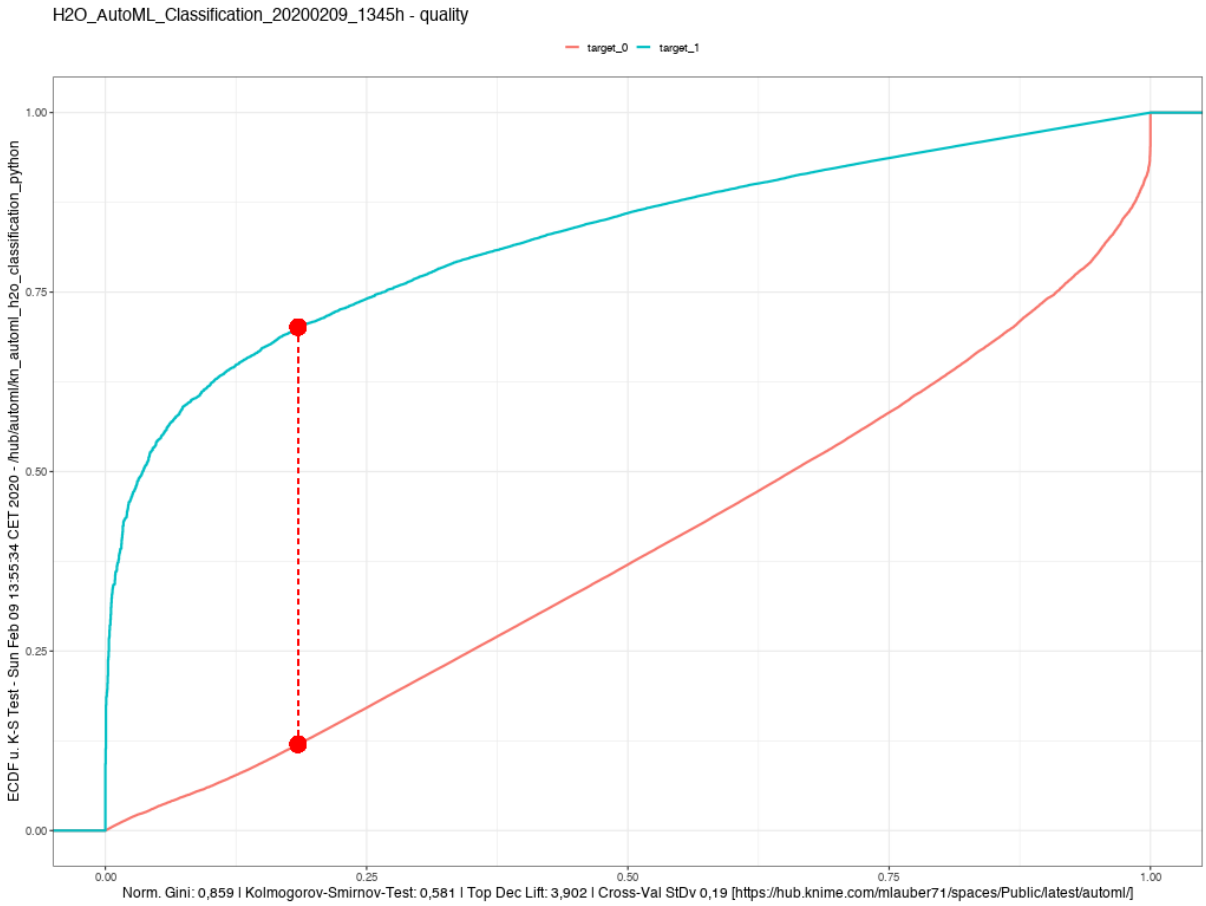
https://en.wikipedia.org/wiki/Receiver_operating_characteristic

TOP Decile Lift



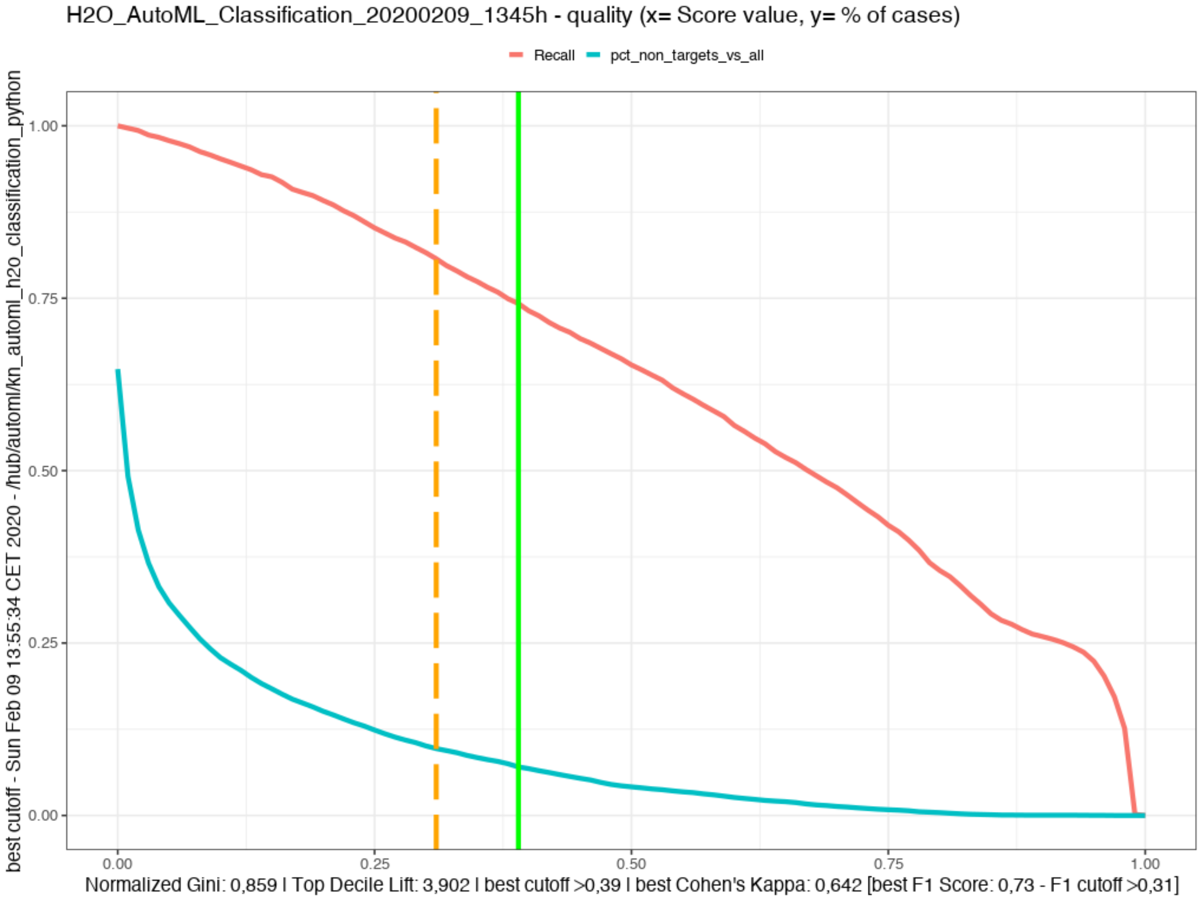
A classic lift curve with statistics. Illustrating how the TOP 10% of your score are doing compared to the rest. You have the cumulative lift that ends in 1.0 (the green line ^= the average % of Targets in your population) and the Lift for each 10% step. This graphic and statistics are useful if you want to put emphasis on the Top group.

Kolmogorov-Smirnov Goodness-of-Fit Test



two curves illustrating the Kolmogorov-Smirnov Goodness-of-Fit Test. An indication about how good the two groups have been separated. The higher the better. Also inspect the curves visually.

Find the best cut-off point for your model



Gives you the idea where the best cutoff might be by consulting two measures

- > 0.39 score if you follow Cohen's Kappa
- > 0.31 if you follow the best F1 score

There is always a price to pay. The blue curve gives you the % of non-targets with regards to all cases that you would have to carry with you if you choose this specific cutoff

If you choose >0.39 you will capture 74% of all your targets. You will have to 'carry' 7% off *all* your cases that are non-Targets which overall make 67% of your population.

If you choose a cutoff of >0.68 you get nearly 50% of your Targets with only about 2% of the population as non-Targets. If this is good or bad for your business case you would have to decide. For more details see the Excel file.

https://en.wikipedia.org/wiki/Precision_and_recall

The accompanying Excel file also holds some interesting information

The Leaderboard from the set of models run

	model_id	auc	logloss	aucpr	_per_class_	rmse	mse
0	GBM_1_AutoML_20200209_134514	0,925952	0,281633	0,806913	0,179058	0,298255	0,088956
1	DRF_1_AutoML_20200209_134514	0,913631	0,315944	0,764041	0,184507	0,309066	0,095522
2	GLM_1_AutoML_20200209_134514	0,905259	0,320347	0,738104	0,197592	0,319713	0,102216

It gives you an idea

- Which types of models were considered?
- Also, the stretch of the AUC could be quite wide. Since all the models only trained 2.5 minutes it would be possible that further training time might result in better models
- In between there as some other models besides GBM if they would appear more often you might also investigate that further

If you are into tweaking, you models further the model summary also gives you the parameters used.

		number_of_trees	number_of_internal_trees	model_size_in_bytes	min_depth	max_depth	mean_depth	min_leaves	max_leaves	mean_leaves
0		100	100	82263	6	6	6	42	64	57,55

Further information will be stored in the print of the whole model with all parameters, also about the cross-validations done.

H2O_AutoML_Classification_20200209_1345h.txt

```
MSE: 0.07594983710496706
RMSE: 0.27558998005182817
LogLoss: 0.2439220731412011
Mean Per-Class Error: 0.1303064492922299
AUC: 0.9476788592324248
AUCPR: 0.8561080375372859
Gini: 0.8953577184648496

Confusion Matrix (Act/Pred) for max f1 @ threshold = 0.3919679260528399:
       0      1     Error    Rate
-----  -----  ----  -------  ----------------
0      24002  2027  0.0779   (2027.0/26029.0)
1      1773   6387  0.2173   (1773.0/8160.0)
Total  25775  8414  0.1111   (3800.0/34189.0)

Maximum Metrics: Maximum metrics at their respective thresholds
metric                       threshold    value     idx
---------------------------  -----------  --------  -----
max f1                       0.391968     0.770725  200
max f2                       0.193525     0.836376  277
max f0point5                 0.588931     0.809477  134
max accuracy                 0.466213     0.891954  173
max precision                0.994328     1         0
max recall                   0.00324808   1         397
max specificity              0.994328     1         0
max absolute_mcc             0.404031     0.697877  196
max min_per_class_accuracy   0.283703     0.86807   239
max mean_per_class_accuracy  0.24332      0.869694  255
max tns                      0.994328     26029     0
max fns                      0.994328     8043      0
max fps                      0.0021123    26029     399
max tps                      0.00324808   8160      397
max tnr                      0.994328     1         0
max fnr                      0.994328     0.985662  0
max fpr                      0.0021123    1         399
max tpr                      0.00324808   1         397

Gains/Lift Table: Avg response rate: 23,87 %, avg score: 23,88 %
    group    cumulative_data_fraction    lower_threshold    lift        cumulative_lift    response_rate    score      cumulative_response_rate
cumulative_score    capture_rate    cumulative_capture_rate    gain      cumulative_gain
--  -------  --------------------------  -----------------  ----------  -----------------  ---------------  ---------  --------------------------
-----------------  --------------  -------------------------  --------  -----------------
    1        0.0100032                   0.992006           4.18983     4.18983            1                0.993241   1
0.993241            0.0419118       0.0419118                  318.983   318.983
    2        0.0200064                   0.989704           4.18983     4.18983            1                0.990936   1
0.992088            0.0419118       0.0838235                  318.983   318.983
```

Variable Importance is very important

Then there is the variable importance list. You should study that list carefully. If one variable captures all the importance you might have a leak. And the variables also should make sense.

If you have a very large list and further down, they stop making sense you could cut them off (besides all the data preparation magic you could do with vtreat, featuretools, tsfresh, label encoding and so on). And also, H2O does some modifications.

	variable	relative_importance	scaled_importance	percentage
0	relationship	4.123,79	1,00	25%
1	capital-gain	2.997,02	0,73	18%
2	marital-status	2.182,30	0,53	13%
3	occupation	1.659,41	0,40	10%
4	education	1.443,28	0,35	9%
5	education-num	1.040,61	0,25	6%
6	age	903,85	0,22	5%
7	capital-loss	895,13	0,22	5%
8	hours-per-week	525,65	0,13	3%
9	native-country	390,08	0,09	2%
10	workclass	261,75	0,06	2%
11	fnlwgt	186,25	0,05	1%
12	sex	38,84	0,01	0%
13	race	37,13	0,01	0%

You could use that list to shrink your y variables and re-run the model. The list of variables is also stored in the overall list.

Fun fact in this case: your relationship and marital status is more important to determine whether you will earn more than $50,000 then your education …

Get an overview how your model is doing in Bins and numbers

submission	solution_1	solution_0	sum_overall	percent_target_1	col_percent_1	col_percent_overall	information
0	71	6.459	6.530	1%	2%	45%	
0,1	185	1.930	2.115	9%	5%	14%	
0,2	251	877	1.128	22%	7%	8%	
0,3	281	610	891	32%	8%	6%	
0,4	289	443	732	39%	8%	5%	
0,5	280	293	573	49%	8%	4%	
0,6	324	213	537	60%	9%	4%	
0,7	344	174	518	66%	10%	4%	
0,8	449	108	557	81%	13%	4%	
0,9	238	15	253	94%	7%	2%	
1	815	4	819	100%	23%	6%	an excellent group, nearly all Target=1 and representing 23% of all your Targets
							H2O_AutoML_Classification_20200209_1345h - quality
							Sun Feb 09 13:55:34 CET 2020 - /hub/automl/kn_automl_h2o_classification_python
							Norm. Gini 0,859 [AUC 0,93] | K-S 0,581 | Top Dec Lift 3,902 | Cross-Val StDv 0,19 [htt

I like this sort of table since it gives you an idea about what a cut-off at a certain score ("submission") would mean.

All numbers are taken from the test/validation group (30% of your population in this case) – you might have to think about your overall population to get the exact proportion.

0,7	344	174	518	66%	10%		4%	
0,8	449	108	557	81%	13%		4%	
0,9	238	15	253	94%	7%		2%	
1	**815**	**4**	**819**	**100%**	**23%**		**6%**	a
								H
				of all Target=1	43%			S
				no Target=1	1.502	92%		N
				no Target=0	127			

If you choose a cutoff at 0.8 you would get 92% precision and 43% of all your desired targets. In marketing/cross-selling that would be an excellent result. In credit scoring you might not want to live with 8% of people not paying back their loan. So again, the cut-off and value of you model very much depends on your business question.

A word about cross-validation

Another aspect of your model quality and stability could be judged by looking at a cross-validation. Although H2O for example does a lot of that by default in order to avoid overfitting you might want to do some checks of your own.

The basic idea is: if your model is really catching a general trend and has good rules they should work on all (random) sub-populations and you would expect the model to be quite stable.

k_fold	colStdevs(final_result)	names	NormalizedGini_oneout	TopDecileLift_oneout	NormalizedGini_subsample	TopDecileLift_subsample	ir
0	0,004149659	NormalizedGini_oneout					
0	0,029743907	TopDecileLift_oneout					
0	0,016393696	NormalizedGini_subsample					
0	0,139740116	TopDecileLift_subsample					
1			0,858393581	3,888	0,863299665	3,946	
2			0,854403843	3,949	0,878223246	3,661	
3			0,856796462	3,895	0,869162145	3,916	
4			0,862399609	3,871	0,847015003	4,033	
5			0,864611463	3,888	0,838019366	3,931	
-1	0,190027378	<= cumulative stdv cross validation; 0 = most stable model					

Several tests are run. In the end we look at a combined standard deviation. 0 would represent a perfect match between all subgroups (sub-sampling and leaving one out techniques). So if you would have to choose between several excellent model you might want to consider the one with the least deviation.

Jupyter notebook

Enclosed in the workflow in the subfolder
/script/ kn_automl_h2o_classification_python.ipynb

there is a walkthrough of Automl in a Jupyter notebook to explore the functions further and if you do not wish to use the wrapper with KNIME