Machine Learning with H2O.ai

1. Problem statement

The goal of this project is twofold.

1. Evaluate the following different ways of using H2O.ai for Machine learning
   1. H2O directly from Python
   2. H2O directly from R
   3. PySparkling from local standalone Spark cluster
   4. H2O Flow on Desktop mode
2. Demonstrate the use of following H2O algorithms on a lesser known Benchmark dataset used by academics and compare the performance metrics
   1. Deep Learning
   2. Gradient Boosting Machine
   3. Generalized Linear Model
   4. Distributed Random Forest
   5. Ensembles stacking using R
3. Dataset

The Protein Fold recognition is a posed as binary classification with a goal of identify proteins that have similar tertiary structure. The FOLDpro dataset is a comprehensive dataset used to benchmark multiple algorithms for this task.

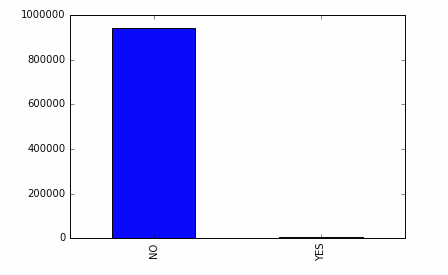
The features corresponding to combinations of 976 proteins among which 555 proteins have at least one positive match with other proteins at the family level, 434 proteins at the super-family level, and 321 proteins at the fold level. The FOLDpro dataset was retrieved from UCI webpage at

<http://mine5.ics.uci.edu:1026/fold_help.html>

The total number of records in the dataset is 951,600 ( i.e 976 x 975 ). Each record consists of 84 features, description of the which may be found in the table below. The file is in LIBSVM format and the uncompressed file size is 1.5GB.

J.Cheng et al have demonstrated that RandomForests to be a suitable algorithm for this task. I shall try and replicate their results using H2O implementation of RandomForest and also evaluate performance of other algorithms on this dataset.

The dataset itself is a highly imbalanced dataset. The class distribution profile is as shown in the chart below

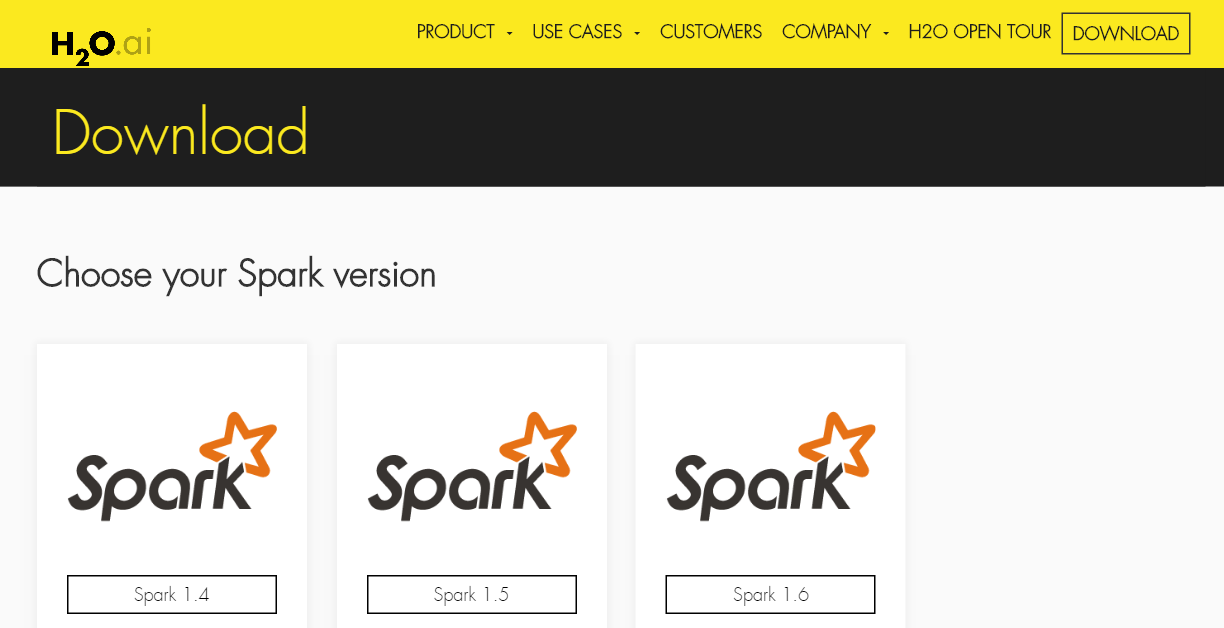


Features & Description

|  |  |
| --- | --- |
| 1 | query length |
| 2 | target length |
| 3 | cosine of composition of monomer of query and template sequences |
| 4 | correlation of composition of monomer of query and template sequences |
| 5 | gaussian function of composition of monomer of query and template sequences |
| 6 | cosine of composition of dimer of query and template sequences |
| 7 | correlation of composition of dimer of query and template sequences |
| 8 | gaussian function of composition of dimer of query and template sequences |
| 9 | cosine of composition of monomer of query and template families |
| 10 | correlation of composition of monomer of query and template families |
| 11 | gaussian function of composition of monomer of query and template families |
| 12 | cosine of composition of dimer of query and template families |
| 13 | correlation of composition of dimer of query and template families |
| 14 | gaussian function of composition of dimer of query and template families |
| 15 | Palign sequence alignment score1 |
| 16 | Palign sequence alignment score2 |
| 17 | Clustalw sequence alignment score |
| 18 | Clustalw profile-profile alignment score |
| 19 | Lobster (coach) profile-profile alignment score |
| 20 | Psiblast profile-sequence alignment score |
| 21 | Psiblast alignment evalue |
| 22 | Psiblast normalized alignment length |
| 23 | Psiblast identity rate of alignment |
| 24 | Psiblast positive rate of alignment |
| 25 | Hmmer(pfam) alignment score |
| 26 | Hmmer(pfam) alignment evalue |
| 27 | Hmmer(search) alignment score |
| 28 | Hmmer(search) alignment evalue |
| 29 | Impala sequence-profile alignment score |
| 30 | Impala alignment evalue |
| 31 | Impala normalized alignment length |
| 32 | Impala alignment identity rate |
| 33 | Impala alignment positive rate |
| 34 | Rpsblast sequence-profile alignment score |
| 35 | Rpsblast alignment evalue |
| 36 | Rpsblast alignment normalized length |
| 37 | Rpsblast alignment indentity rate |
| 38 | Rpsblast alignment positive rate |
| 39 | secondary structure match ratio |
| 40 | relative solvent accessibility match ratio |
| 41 | average contact probability (cmappro, 8 Angstrom) |
| 42 | cosine of residue contact num (cmappro, 8 Angstrom) |
| 43 | correlation of residue contact num (cmappro, 8 Angstrom) |
| 44 | cosine of residue contact order (cmappro, 8 Angstrom) |
| 45 | corr\_of\_residue\_contact\_ord (cmappro, 8 Angstrom) |
| 46 | average contact probability (bmappro, 8 Angstrom) |
| 47 | cosine of residue contact num (bmappro, 8 Angstrom) |
| 48 | correlation of residue contact num (bmappro, 8 Angstrom) |
| 49 | cosine of residue contact order (bmappro, 8 Angstrom) |
| 50 | corr\_of\_residue\_contact\_ord (bmappro, 8 Angstrom) |
| 51 | average contact probability (cmappro, 12 Angstrom) |
| 52 | cosine of residue contact num (cmappro, 12 Angstrom) |
| 53 | correlation of residue contact num (cmappro, 12 Angstrom) |
| 54 | cosine of residue contact order (cmappro, 12 Angstrom) |
| 55 | corr\_of\_residue\_contact\_ord (cmappro, 12 Angstrom) |
| 56 | average contact probability (bmappro, 12 Angstrom) |
| 57 | cosine of residue contact num (bmappro, 12 Angstrom) |
| 58 | correlation of residue contact num (bmappro, 12 Angstrom) |
| 59 | cosine of residue contact order (bmappro, 12 Angstrom) |
| 60 | corr\_of\_residue\_contact\_ord (bmappro, 12 Angstrom) |
| 61 | beta-residue in beta-sheet pairing probability |
| 62 | percentage of helix in query |
| 63 | percentage of beta-strand in query |
| 64 | percentage of coil in query |
| 65 | percentage of exposed residue in query |
| 66 | percentage of buried residue in query |
| 67 | percentage of helix in template |
| 68 | percentage of beta-strand in template |
| 69 | percentage of coil in template |
| 70 | percentage of exposed residue in template |
| 71 | percentage of buried residue in template |
| 72 | consine of SS and SA composition |
| 73 | correlation of SS and SA composition |
| 74 | gaussian function of SS and SSA composition |
| 75 | dot product of SS and SA composition |
| 76 | prc-hmm coemis score (on hmmer profile) |
| 77 | prc-hmm simple score (on hmmer profile) |
| 78 | prc-hmm reverse score (on hmmer profile) |
| 79 | prc-hmm coemis score (on chk profile) |
| 80 | prc-hmm simple score (on chk profile) |
| 81 | prc-hmm reverse score (on chk profile) |
| 82 | HHsearch profile-profile alignment score |
| 83 | Compass alignment score |
| 84 | Compass evalue |

1. Installation / Configuration





This section contains following details

1. Installing h2o in python & verifying installation
2. Installing h2o in R & verifying installation
3. Installing Sparkling water ( h2o + spark ) & verifying installation
4. Installing h2oensemble R package
5. Installing and verifying installation of H2O flow
6. Downloading and Installation of executables for benchmarking performance of algorithms as related to the problem statement of this project.

Install h2o in python & verifying installation

1. Prerequisite: Python 2.7 or 3.5

2. Install dependencies (use sudo if needed):

pip install -U requests

pip install -U tabulate

pip install -U future

pip install -U six

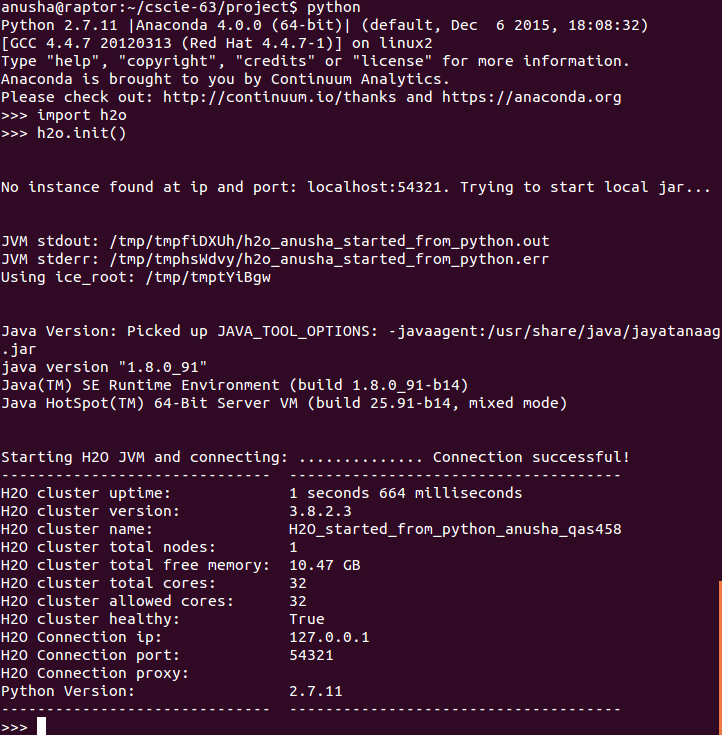
3. Uninstall older version of h2o if it exists in the system

pip uninstall h2o

4. Installing H2O Python module.

pip install http://h2o-release.s3.amazonaws.com/h2o/rel-turchin/3/Python/h2o-3.8.2.3-py2.py3-none-any.whl

5. Verifying installation



Installing h2o in R & verifying installation

Open an R terminal and type in the following

install.packages("h2o", type="source", repos=(c("http://h2o-release.s3.amazonaws.com/h2o/rel-turchin/3/R")))  
library(h2o)  
localH2O = h2o.init(nthreads=-1)



Installing H2OEnsemble Package in R

H2O ensemble methods are implemented in a standalone R package called h2oensemble. This package is an extension of h2o R package which allows the user to train an ensemble in the H2O cluster

The following packages were installed as prerequisites

install.packages(“git2r”)

install.packages(“curl”)

install.packages(“httr”)

install.packages(“devtools”)

The package devtools was loaded to install h2o ensemble package

library(devtools)  
install\_github("h2oai/h2o-3/h2o-r/ensemble/h2oEnsemble-package")

Sparking water Installation

*Step1:* Install spark on the machine if not already present. Verify that SPARK\_HOME points to spark installation folder

*Step2:* Set variable MASTER which contains a configuration of the cluster to be launched

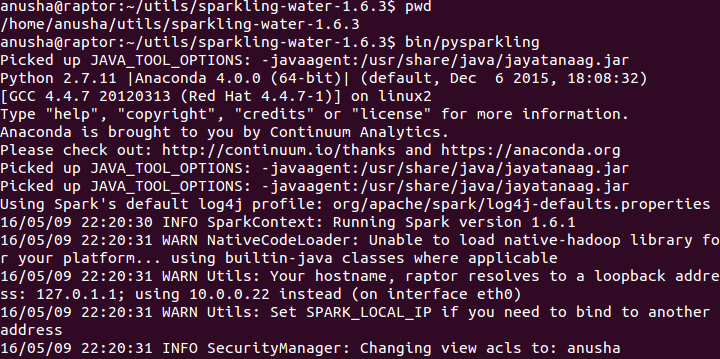
For example, to launch a local Spark cluster with 3 worker nodes with 2 cores and 1g per node, run in a terminal :

export MASTER="local-cluster[3,2,1024]"

*Step3:* Download sparking water software from : <http://www.h2o.ai/download/sparkling-water/spark16> . Unzip contains to a folder and navigate to the folder

*Step4:* Code can be run using either an interactive Python terminal or Ipython notebook at this point. The code to launch the same is as follows

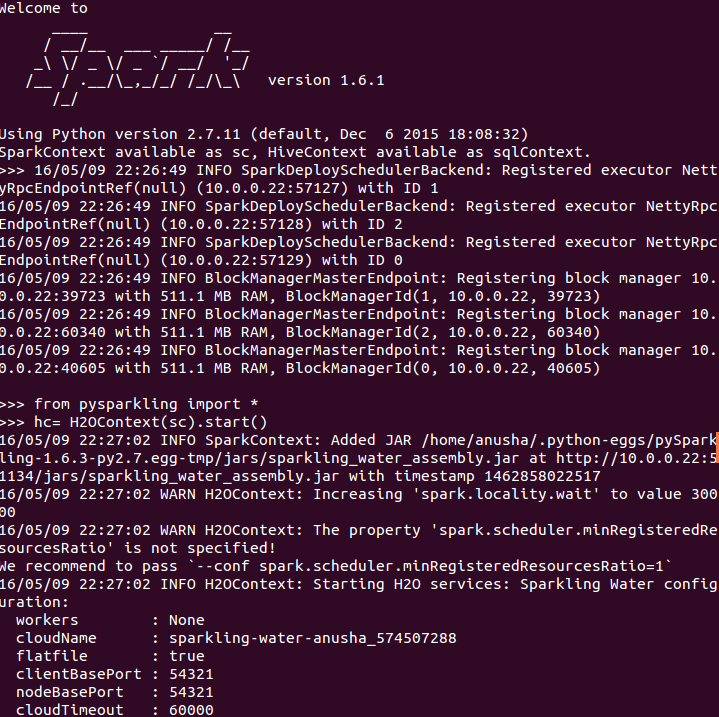
*#interactive Python terminal*  
bin/pysparkling

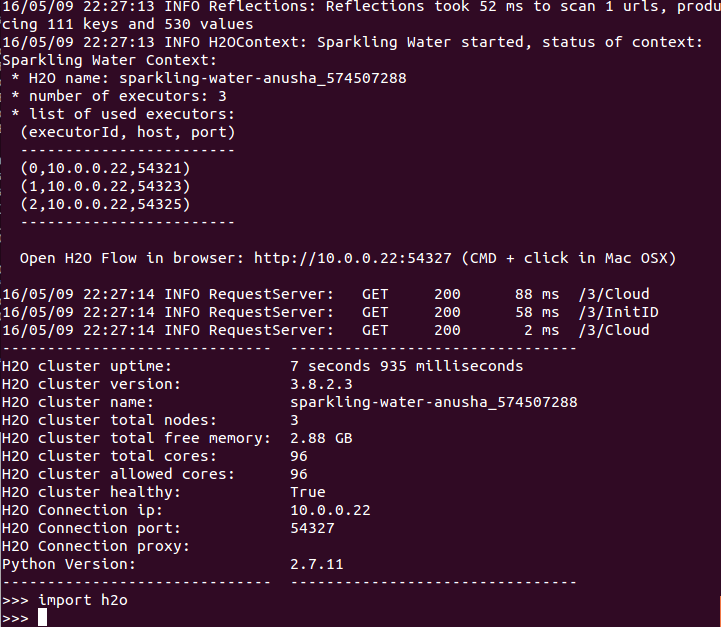


*#ipython notebook*  
IPYTHON\_OPTS="notebook" bin/pysparkling

*Step5:* Create an H2O cluster by running the following

from pysparkling import \*  
hc= H2OContext(sc).start()  
import h2o





H2O Flow Installation

H2O flow is a software provide provided by H2O which makes it easy to do machine learning using a GUI. H2O flow can be run on a Spark mode / Local mode. On Spark mode, the Flow UI can be accessed using the URL which gets printed ( screen shot above ) upon launching SparklingWater.

H2O Flow can also be installed locally using the following steps.

*Step1* : Download h2o software using curl command or using a browser from the website - <http://www.h2o.ai/download/h2o/desktop>

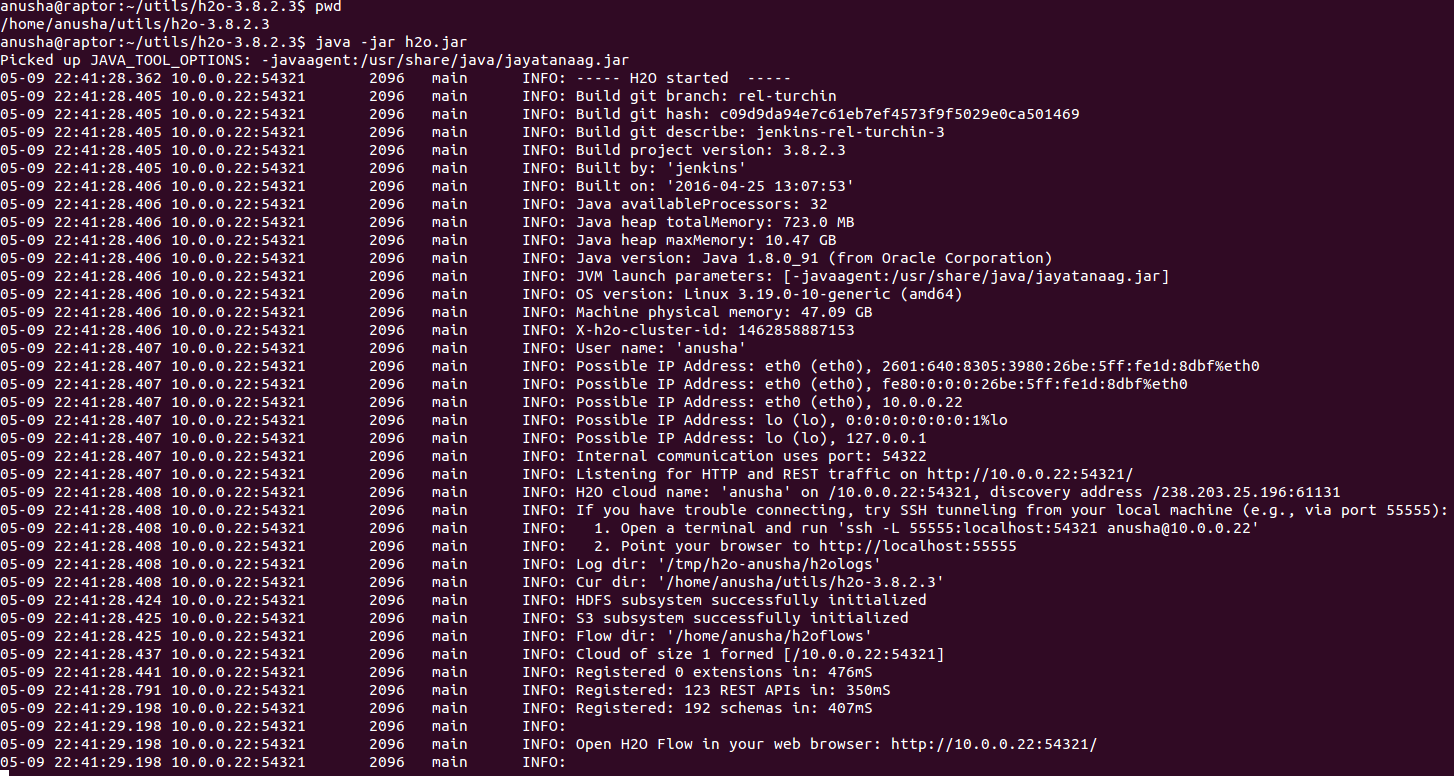
curl -o h2o.zip http://download.h2o.ai/versions/h2o-3.8.2.3.zip

*Step2* Unzip the contents of zip file to a folder

*Step3* Launch H2O locally as follows after navigating to the folder from Step2

java -jar h2o.jar

*Step4* Access Flow UI at [http://localhost:54321](http://localhost:54321/)using a browser



1. Implementation

This project’s implementation consisted of following high-level tasks

1. Data transformation from LIBSVM to CSV format
2. Data exploration & Model Building with H2O Flow in Desktop mode
3. Building classifier models using H2O + Python
4. Run time comparison : PySparkling ( Local PySpark + H2O) vs H2O Python
5. Building ensemble model using H2O + R
6. Calculating Top1/Top5 sensitivity metrics for H2O Algorithms on the benchmark dataset using external tools provided by authors of the original research paper.

4.1 Data transformation from LIBSVM to CSV

As per h2o.ai website ( <https://github.com/h2oai/h2o-2/wiki/Parser-Specification> ), LIBSVM as input file format is supported. Some of the known issues with format is also listed. If LIBSVM could not be used as is, the alternative is to first transform the file to ubiquitous CSV format.

Before proceeding to read the file into H2O Cluster memory for processing, a quick overview of LIBSVM format itself would be useful

LIBSVM format is explained using the following legend for color coding.

Informative comment

Label

FeatureIndex

FeatureValue

The first few lines of file are as follows:

#1aca-d1aca 1abra-d1abra

-1 1:0.86 2:2.51 3:0.698011213320252 4:0.0129149079484725 5:0.809199735874497 6:0.27097496051133 7:0.0427896376978204 8:0.87636911684164 9:0.868627475071523 10:0.452406748572609 11:0.876725945830074 12:0.600177632148831 13:0.263828426616195 14:0.935274598806304 15:0.439352251735332 16:0.339785228557381 17:0.0930232558139535 18:1.75581395348837 19:-2.19767441860465 20:0.169767441860465 21:-0.430782916092454 22:0.13953488372093 23:0.33 24:0.33 25:-2.61511627906977 26:0 27:-0.873255813953488 28:-0.198450938723838 29:0.136046511627907 30:1.50407739677627 31:0.23255813 9534884 32:0.1 33:0.35 34:0.118604651162791 35:2.63905732961526 36:0.0697674418604651 37:0 38:0.5 39:0.22093023255814 40:0.581395348837209 41:0.0229472686904762 42:0.526592567947394 43:0.191639038817938 44:0.537929373295353 45:0.301708457175711 46:0.0139027040702381 47:0.553370683027986 48:0.262824274759729 49:0.503738286626138 50:0.26289871644687 51:0.28208439942029 52:0.763290997965601 53:0.427663172777061 54:0.761204462628882 55:0.578318193392042 56:0.241353475057971 57:0.732093811350112 58:0.309420190767253 59:0.71073499382554 60:0.470352794231018 61:0 62:0.627906976744186 63:0 64:0.372093023255814 65:0.546511627906977 66:0.453488372093023 67:0.374501992031873 68:0.223107569721116 69:0.402390438247012 70:0.446215139442231 71:0.553784860557769 72:0.932923972258857 73:0.682791834197459 74:0.692490152514236 75:0.879875845455388 76:0.15 77:0.122093023255814 78:-0.00930232558139535 79:0.154651162790698 80:0.0941860465116279 81:-0.0313953488372093 82:0.077906976744186 83:0.395348837209302 84:0.0844290831908747

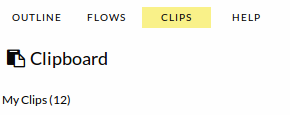
#1aca-d1aca 1abrb-d1abrb1

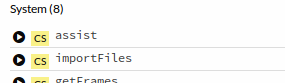
-1 1:0.86 2:1.4 3:0.762213800753194 4:0.230031710821363 5:0.828307292174974 6:0.18845079720878 7:-0.00435811988433937 8:0.862555747226933 9:0.833398050038346 10:0.266497645663691 11:0.86313715880058 12:0.487572776540934 13:0.0939487737732833 14:0.9257515642487 15:0.572883644359854 16:0.76 17:0.0348837209302326 18:1.54651162790698 19:-0.906976744186046 20:0.16046511627907 21:-0.462035459596559 22:0.0697674418604651 23:0.33 24:0.66 25:-1.19651162790698 26:0 27:-0.876744186046512 28:-0.186329578191493 29:0.158139534883721 30:-0.478035800943 31:0.0581395348837209 32:0.4 33:0.6 34:0.126744186046512 35:1.56861591791385 36:0.0465116279069767 37:0.25 38:0.5 39:0.197674418604651 40:0.569767441860465 41:0.0261394677192983 42:0.566081564137592 43:0.33810536939155 44:0.559160939866412 45:0.386797534671188 46:0.00987395312105263 47:0.462420728566599 48:0.169384728012635 49:0.540077513130966 50:0.362099265870245 51:0.16930419326087 52:0.767769058903899 53:0.447522039884289 54:0.721561605359976 55:0.474078003176834 56:0.124980963252174 57:0.761362542453385 58:0.406607138393274 59:0.746894880611322 60:0.501056593104201 61:0 62:0.627906976744186 63:0 64:0.372093023255814 65:0.546511627906977 66:0.453488372093023 67:0.107142857142857 68:0.328571428571429 69:0.564285714285714 70:0.557142857142857 71:0.442857142857143 72:0.79075547682617 73:-0.0942779967636803 74:0.524543194248538 75:0.782558139534884 76:0.133720930232558 77:0.0802325581395349 78:-0.0348837209302326 79:0.145348837209302 80:0.0848837209302326 81:0.00232558139534884 82:0.0453488372093023 83:0.290697674418605 84:1.04640168760164

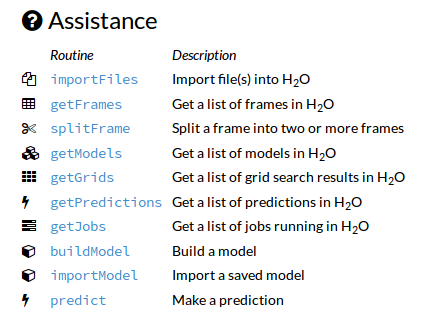
The informative comments here is of the format # RecordId.The RecordId needs to be retained for a later step towards matching labels for algorithm benchmarking purposes. However, as shown in next topic, the presence of these comments made reading the file a difficult task.

Attempting to read the file as in H2O as in without converting to CSV

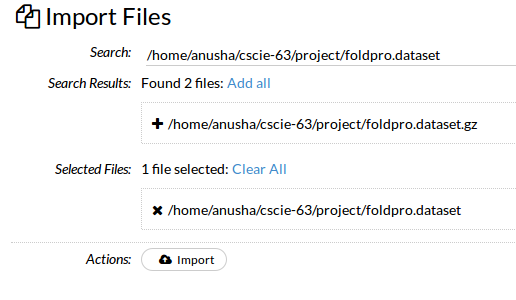
Upon starting H2O Flow notebook, following assist prompt is displayed. If not already displayed it could be selected from the clips clipboard, system tab



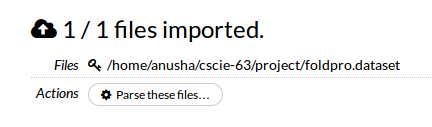


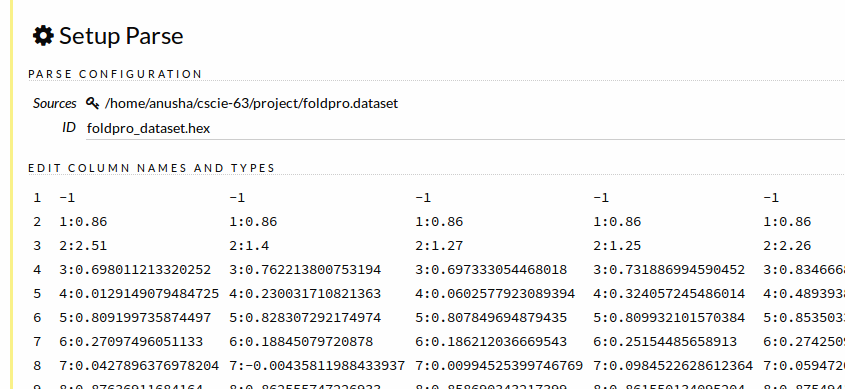


Click on importFiles , fill in details of the select input file and click import

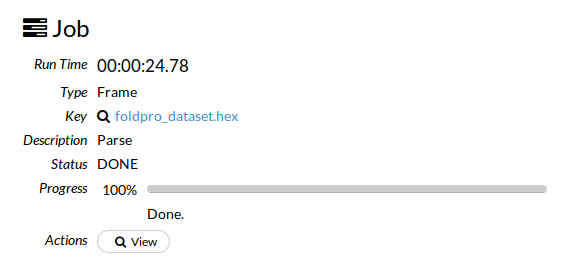


After file is import following cell gets displayed. Click on parse files

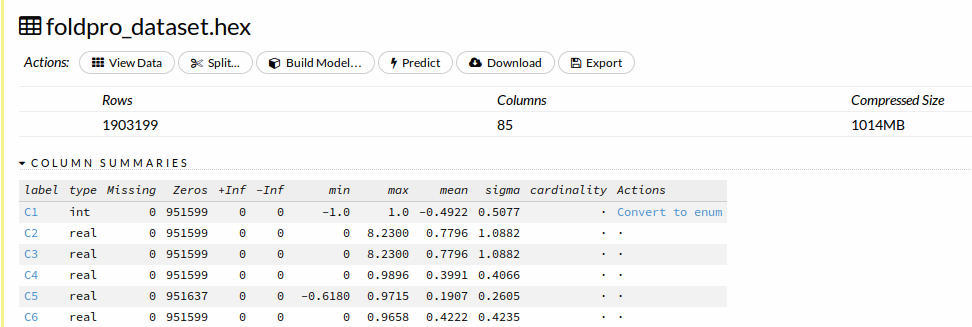


Select “SVMLight” from the drop down list for Parser and click parse  


At this point a data import Job starts to run



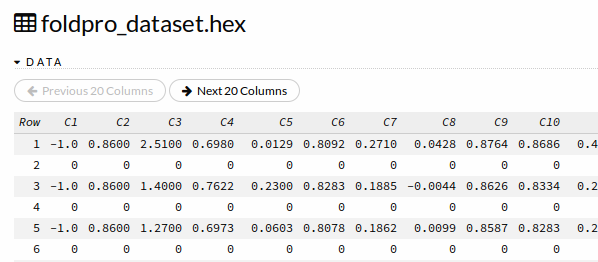
Once job is complete, Click on Actions: View button who got displayed



The above screenshot shows that twice the number of records is getting imported ( 1,903,199 instead of 951,600 ). Clearly SVMLight Parser is not working correctly.

Click on View Data.

From the screenshot below, it seems that instead of skipping Informative comment line altogether, it is reading in the same as zero values even though Parser preview earlier skipped these lines in agreement with LIBSVM format.



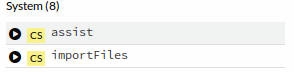
After trying a few different settings to read the file, it made sense to convert the file to CSV format. Since the details of this conversion and appending header is not integral to discussion on H2O this section has been moved to Appendix-I

Appendix-I details the process by which the final file with appropriate column headers was created using open source file conversion software and Unix utils.

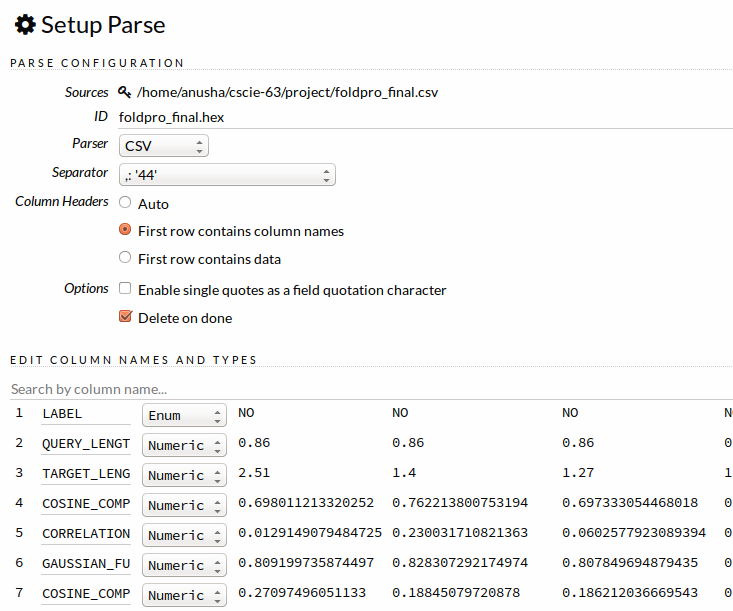
The CSV file thus created using methods in Appendix-I containing modified header is *foldpro\_final.csv.* The Id values are saved in file *IdFile.txt* for later use in evaluating the predictions later on.

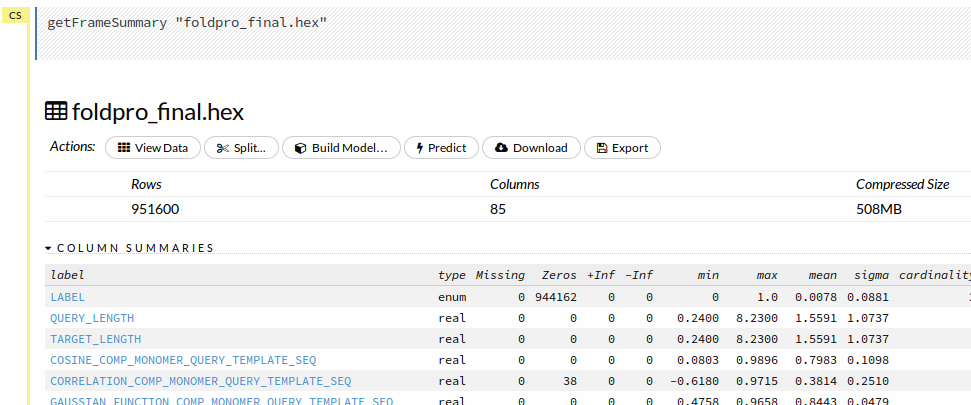
4.2 Data exploration & Model Building with H2O Flow in Desktop mode

Import the new file into H2O flow using Clips -> System -> ImportFiles



Parse using CSV parser



The file is imported and the contents examined like earlier. 

The correct row count is now displayed along with the modifier header that was put into the file.

This window also provides meaningful stats like min, max, mean, cardinality for enum columns etc.

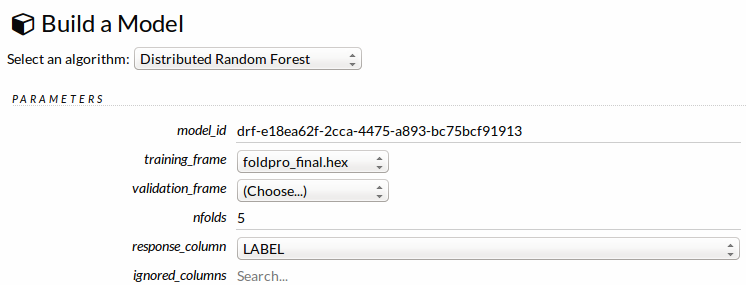
IdFile.txt

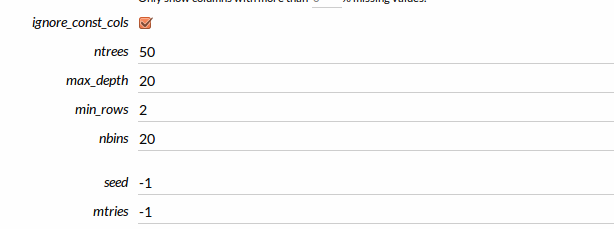
Building a random forest model using the Flow tool:

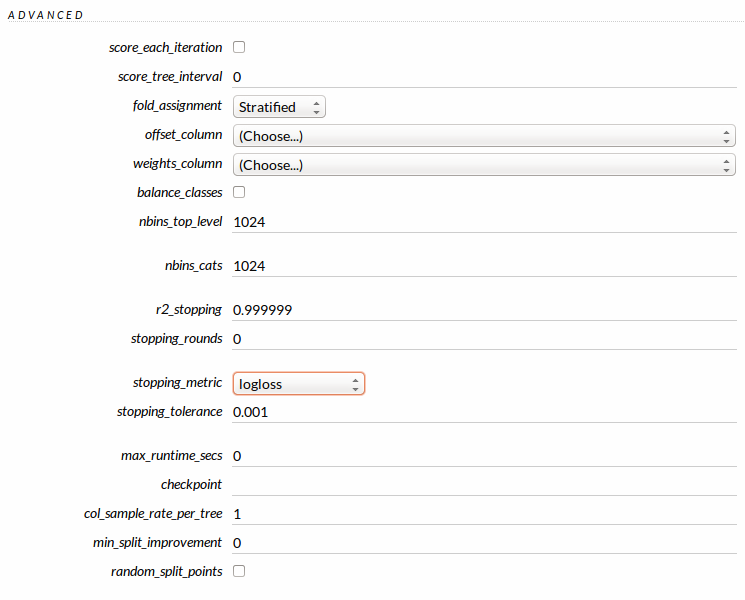
Select clips-> system -> build model

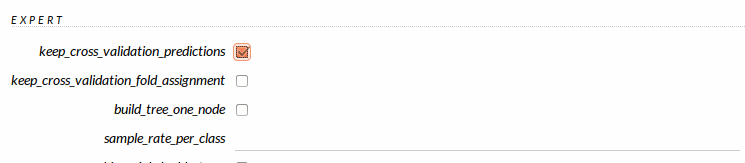


Selected “Distributed Random Forest” from the algorithm drop down list and enter the appropriate algorithm hyper parameters







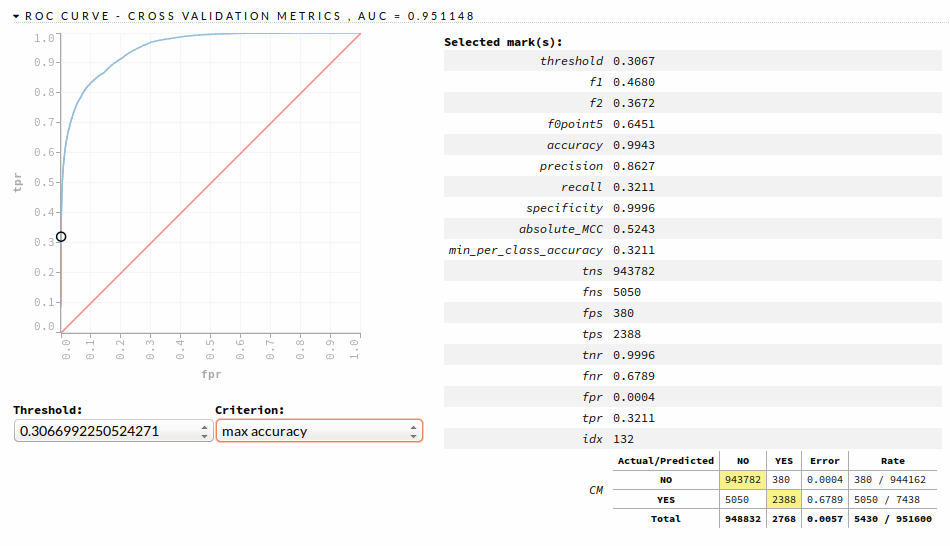


Click on “Build model”

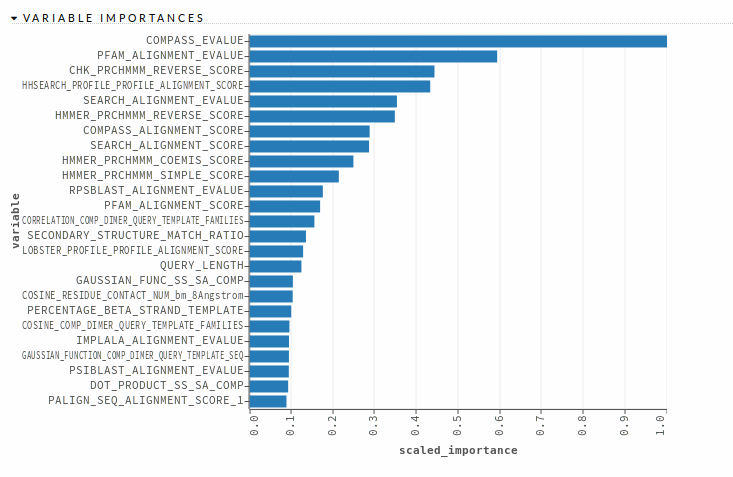
Upon “Job completion” click “View Model”

A detailed summary of the various performance metrics of the model, graphs to select appropriate thresholds and feature relative relevance charts now get printed. Screenshots of some of the report elements can be found below.

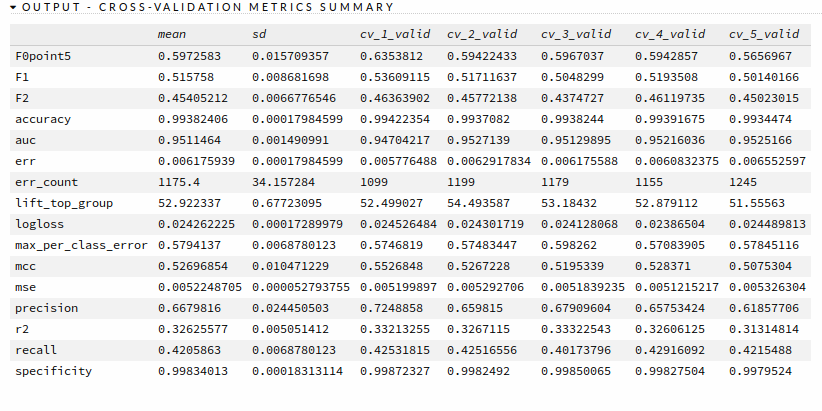
Chart for evaluation of performance of model at a particular threshold on the ROC curve



Variable Importance Chart



Cross-validation metrics summary



Using another data import process another test data file could be imported and scored using the predict function. I’m not exploring the predict function however, because I’m primarily interested in using Cross validation metrics, which is in alignment with the problem statement.

Thus demonstration of GUI mode of Model building & Model exploration using H2O Flow is now complete.

4.3 Building classifier models using H2O + Python

import h2o

import pandas as pd

import os

import tabulate

import operator

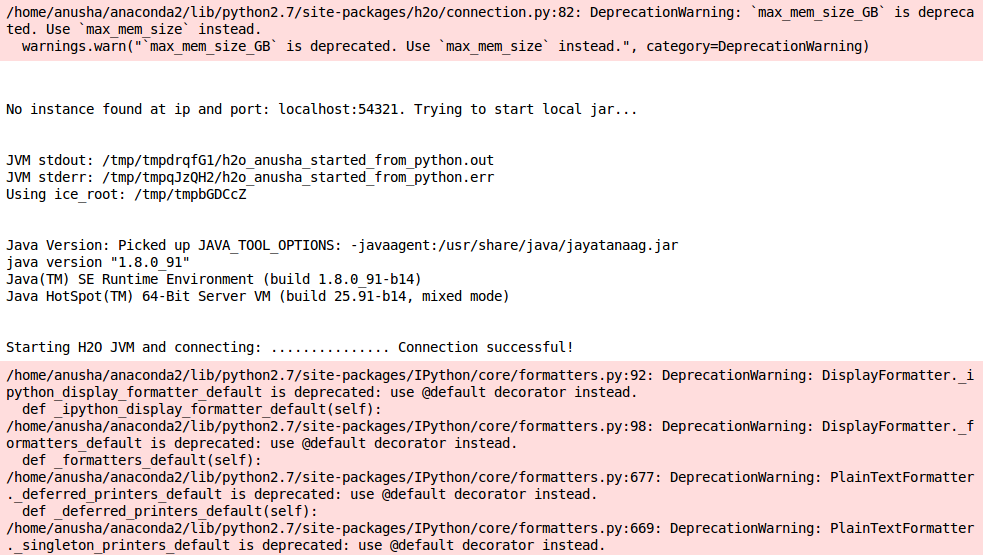
from h2o.estimators.gbm import H2OGradientBoostingEstimator

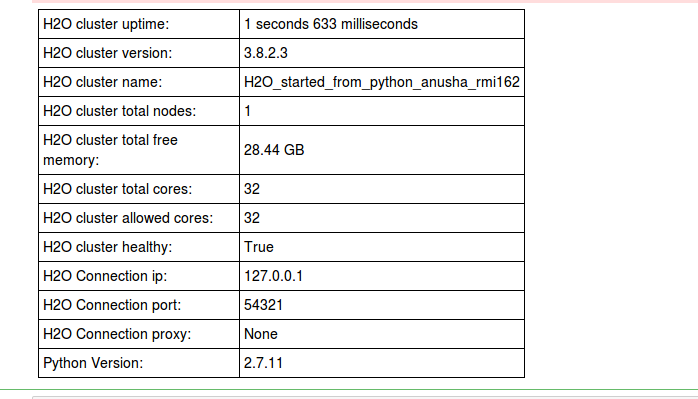
from h2o.estimators.glm import H2OGeneralizedLinearEstimator

from h2o.estimators import H2ORandomForestEstimator

from h2o.estimators import H2ODeepLearningEstimator

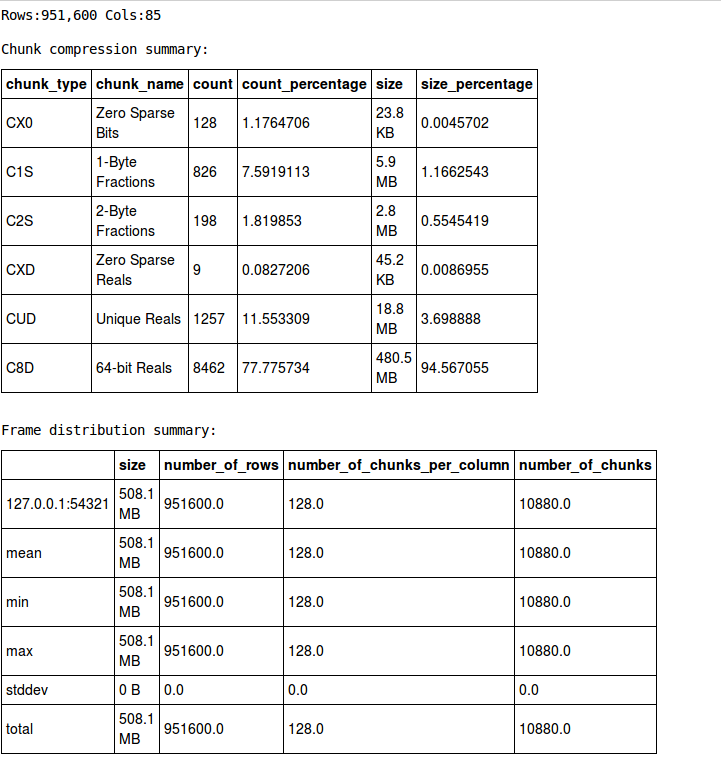
h2o.init(nthreads = -1 , max\_mem\_size\_GB = 32)





foldpro\_df = h2o.import\_file("/home/anusha/cscie-63/project/foldpro\_final.csv")

foldpro\_df.describe()





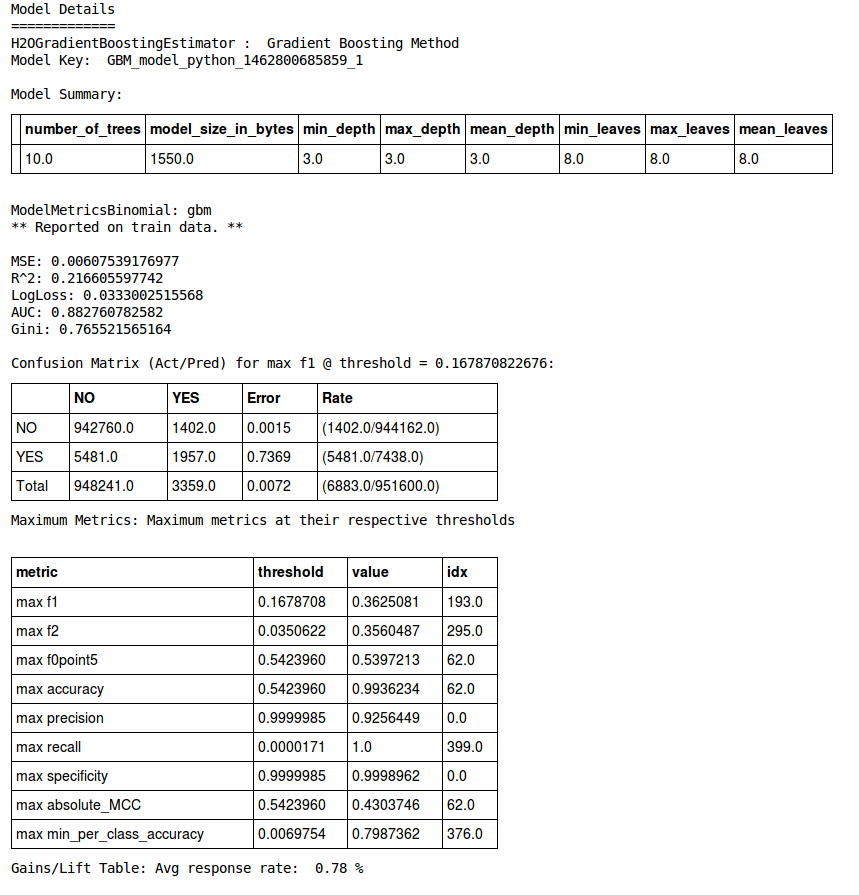
type(foldpro\_df)

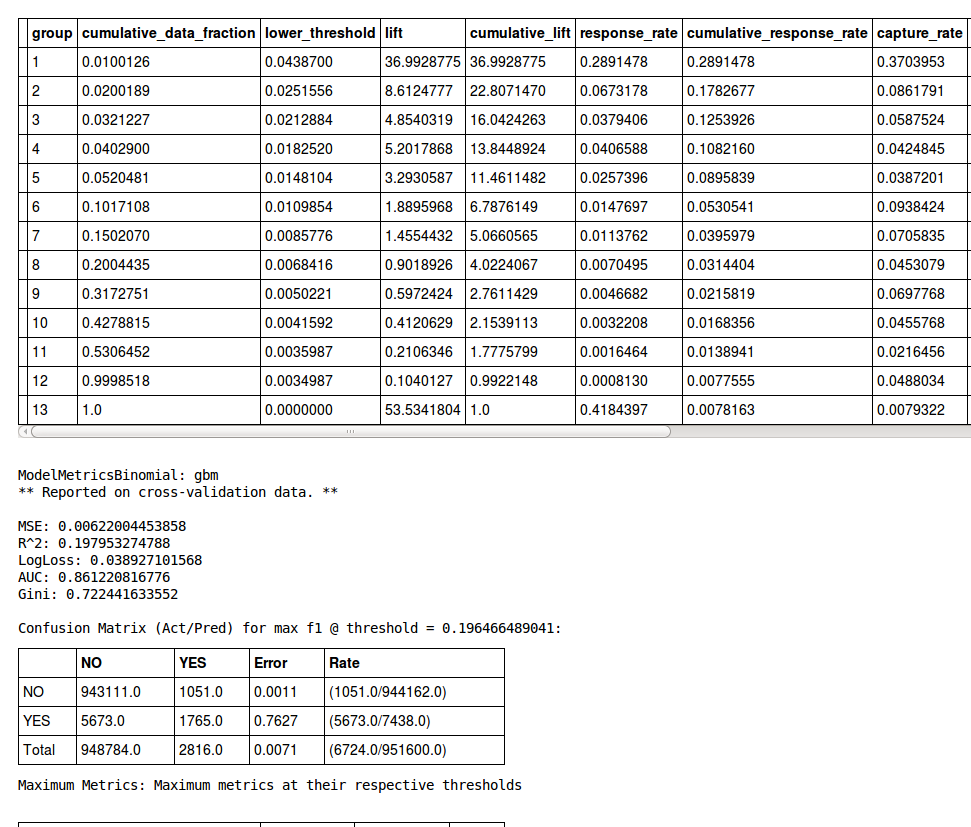
h2o.frame.H2OFrame

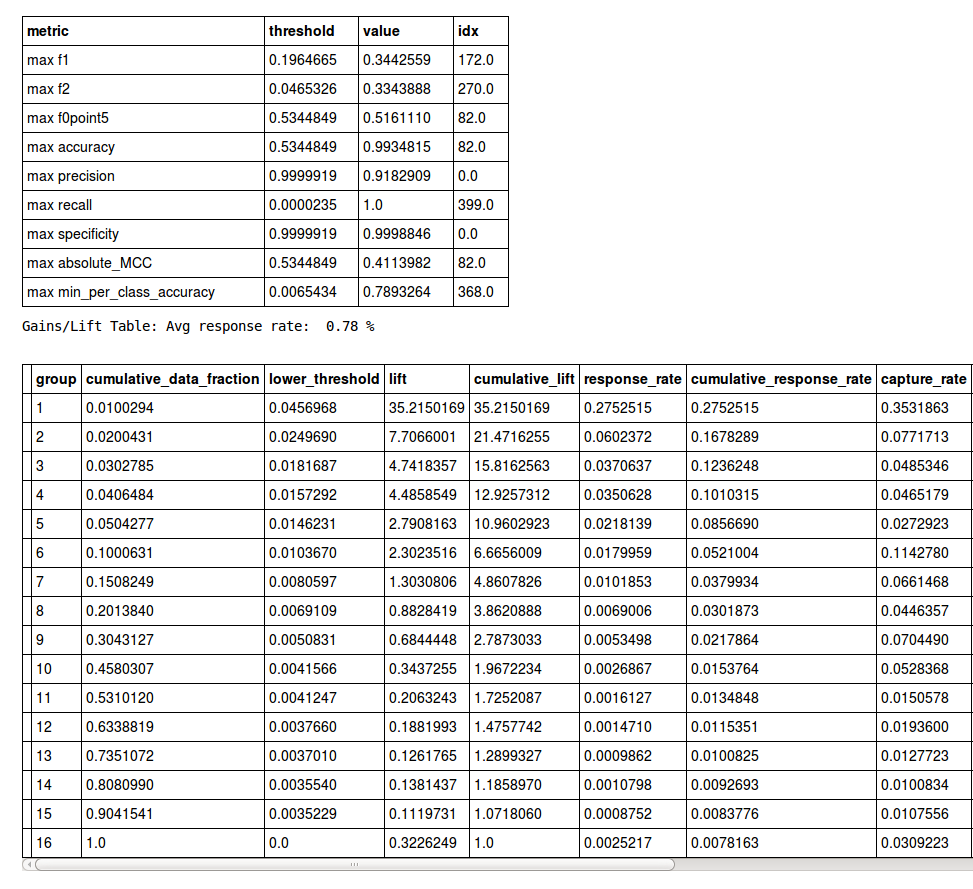
Training a single prediction model which does 10-Fold cross validation  
gbm\_classifier = H2OGradientBoostingEstimator(  
 distribution="multinomial",  
 ntrees=10,   
 max\_depth=3,   
 min\_rows=2,   
 learn\_rate="0.2" ,  
 nfolds = 10 ,   
 keep\_cross\_validation\_predictions = True)

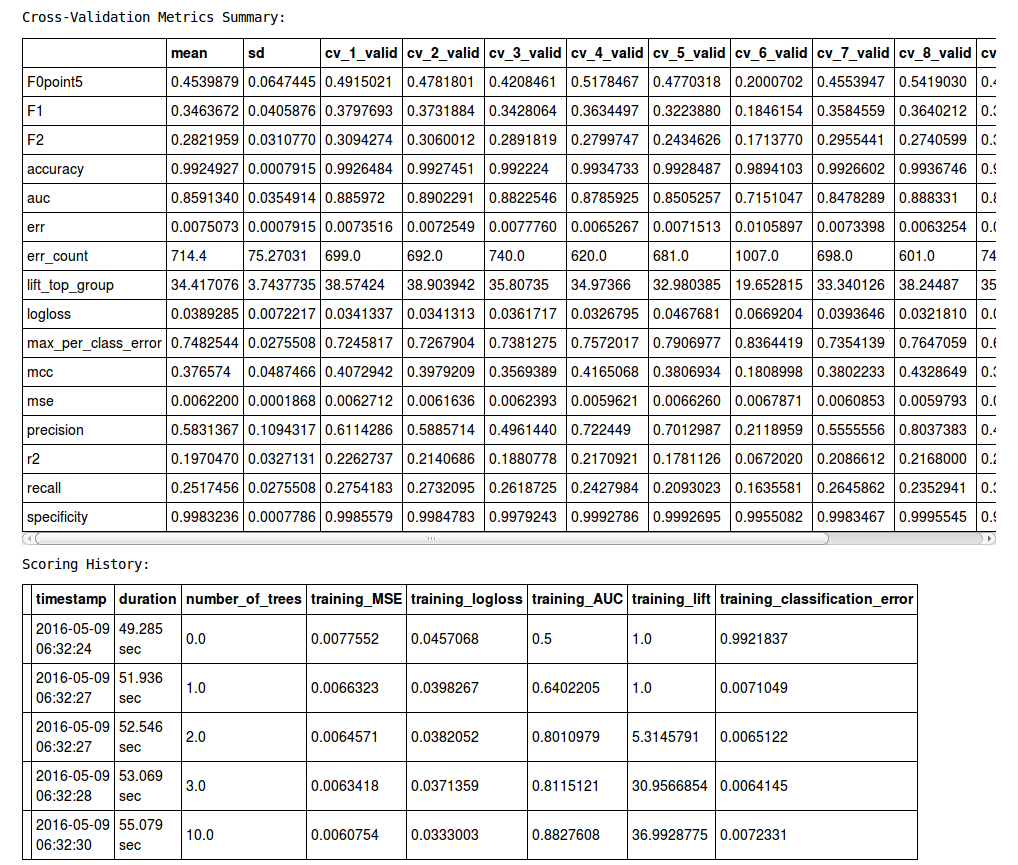
gbm\_classifier.train(x=range(1,foldpro\_df.ncol),   
 y=0,   
 training\_frame= foldpro\_df,   
 seed = 1234  
 )

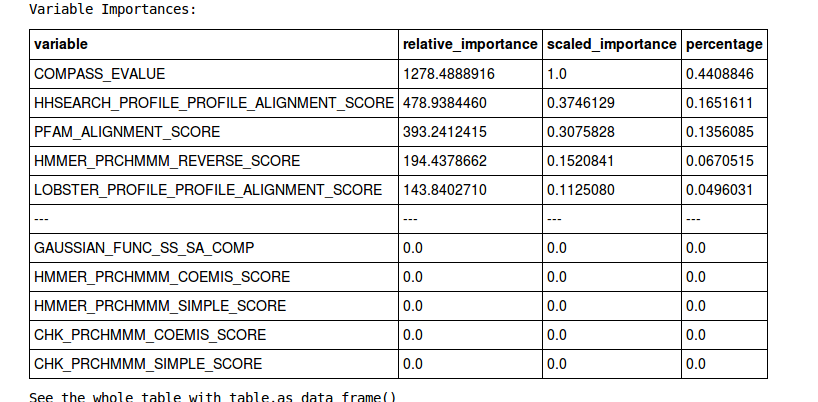
Printing model build report for the GBM classifier  
gbm\_classifier







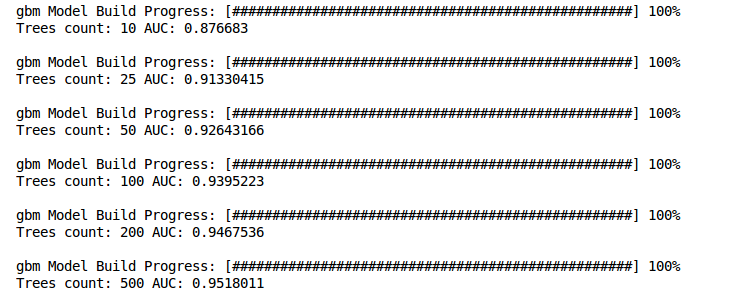




The chosen True north metric in this case Mean CV AUC  
To avoid clutter, looking at only Mean CV AUC going forward  
Mean CV AUC can be extracted from the report as follows  
  
print "Mean CV AUC:" , gbm\_classifier.cross\_validation\_metrics\_summary().as\_data\_frame().iloc[4]["mean"]

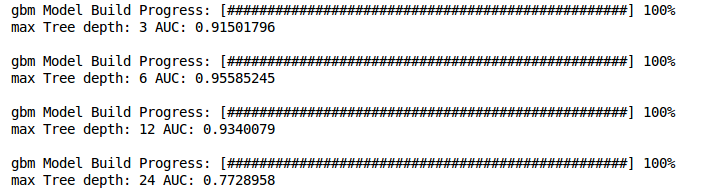
Mean CV AUC: 0.85913396

Since it was found that GridSearch cannot be done with model while using nFold cross-validation using H2O, chose to use Line search to tune one parameter at a time instead.  
  
for trees\_count in [10, 25, 50, 100, 200 , 500] :  
 gbm\_classifier2 = H2OGradientBoostingEstimator(  
 distribution="multinomial",  
 ntrees=trees\_count,   
 max\_depth=3,   
 min\_rows=2,   
 learn\_rate="0.2" ,  
 nfolds = 10 ,   
 keep\_cross\_validation\_predictions = True )  
  
 gbm\_classifier2.train(x=range(1,foldpro\_df.ncol),   
 y=0,   
 training\_frame= foldpro\_df ,  
 seed = 1234  
 )  
  
 mean\_cv\_auc = gbm\_classifier2.cross\_validation\_metrics\_summary().as\_data\_frame().iloc[4]["mean"]  
 print "Trees count:" , trees\_count , "AUC:" , mean\_cv\_auc



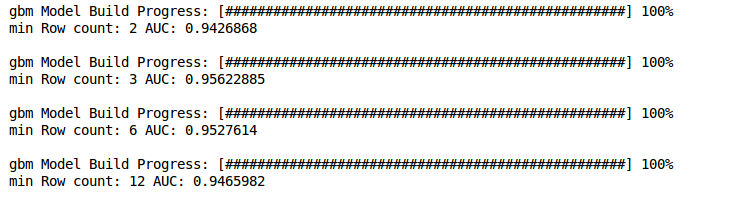
trees\_count\_opt = 200

# Chose 200 instead of 500 to reduce model build time overall , also the gain seems marginal



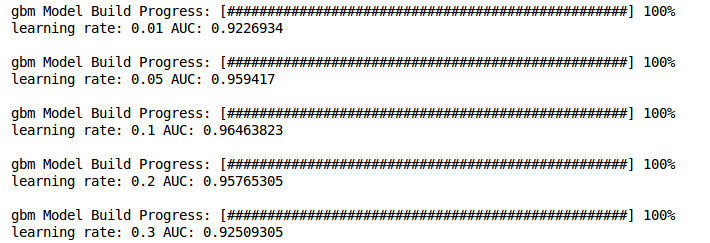
md\_opt = 6

Determining optimal min Rows using Line search  
for mr in [2, 3, 6, 12] :  
 gbm\_classifier2 = H2OGradientBoostingEstimator(  
 distribution="multinomial",  
 ntrees=trees\_count\_opt,   
 max\_depth=md\_opt,   
 min\_rows=mr,   
 learn\_rate="0.2" ,  
 nfolds = 10 ,   
 keep\_cross\_validation\_predictions = True)  
  
 gbm\_classifier2.train(x=range(1,foldpro\_df.ncol),   
 y=0,   
 training\_frame= foldpro\_df ,  
 seed = 1234  
 )  
  
 mean\_cv\_auc = gbm\_classifier2.cross\_validation\_metrics\_summary().as\_data\_frame().iloc[4]["mean"]  
 print "min Row count:" , mr , "AUC:" , mean\_cv\_auc



mr\_opt = 2

Determining optimal learn rate using Line search  
for lr in [.01, .05, 0.1, 0.2, 0.3] :  
 gbm\_classifier2 = H2OGradientBoostingEstimator(  
 distribution="multinomial",  
 ntrees=trees\_count\_opt,   
 max\_depth=md\_opt,   
 min\_rows=mr\_opt,   
 learn\_rate=lr ,  
 nfolds = 10 ,   
 keep\_cross\_validation\_predictions = True)  
  
 gbm\_classifier2.train(x=range(1,foldpro\_df.ncol),   
 y=0,   
 training\_frame= foldpro\_df ,  
 seed = 1234  
 )  
  
 mean\_cv\_auc = gbm\_classifier2.cross\_validation\_metrics\_summary().as\_data\_frame().iloc[4]["mean"]  
 print "learning rate:" , lr , "AUC:" , mean\_cv\_auc



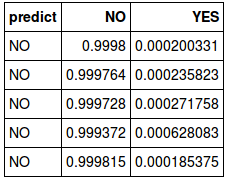
lr\_opt = 0.1

Creating 10-Fold predictions corresponding to Best GBM Model for Benchmarking Purpose

gbm\_classifier3 = H2OGradientBoostingEstimator(  
 distribution="multinomial",  
 ntrees=trees\_count\_opt,   
 max\_depth=md\_opt,   
 min\_rows=mr\_opt,   
 learn\_rate=lr\_opt ,  
 nfolds = 10 ,   
 keep\_cross\_validation\_predictions = True)  
  
gbm\_classifier3.train(x=range(1,foldpro\_df.ncol),   
 y=0,   
 training\_frame= foldpro\_df ,  
 seed = 1234  
 )

cv\_pred\_gbm = gbm\_classifier3.cross\_validation\_holdout\_predictions()

cv\_pred\_gbm.head(5)

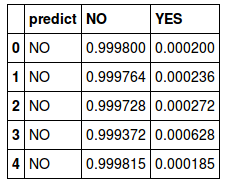


cv\_pred\_gbm.shape

(951600, 3)

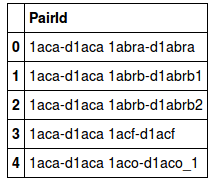
cv\_pred\_gbm\_df = cv\_pred\_gbm.as\_data\_frame(use\_pandas=True)

cv\_pred\_gbm\_df.head(5)



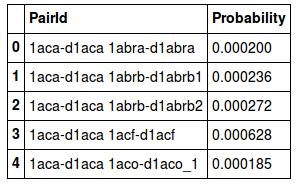
IdText = pd.read\_csv("/home/anusha/cscie-63/project/IdFile.txt" ,   
 header = None ,   
 names = ["PairId"])

IdText.head()



outDf1 = pd.DataFrame( zip (IdText.PairId , cv\_pred\_gbm\_df.YES ) , columns = ["PairId","Probability"])

outDf1.head()



import csv  
outDf1.to\_csv("gbm\_prediction.csv" ,   
 sep = " " ,   
 header = False ,   
 index = False ,   
 quoting= csv.QUOTE\_NONE ,   
 quotechar='',  
 escapechar=' ')

Completed end-to-end process for creating predictions corresponding to one h2o classifier until now. Repeating the above process for following H2O ( except Line search for optimal parameters due to lack of time)

a. H2OGeneralizedLinearEstimator  
 b .H2ORandomForestEstimator

c. H2ODeepLearningEstimator

Also choosing not to print model build summary of these classifiers since they are many pages long.  
  
glm\_classifier = H2OGeneralizedLinearEstimator(family="binomial",  
 nfolds=10,   
 alpha=0.5,  
 keep\_cross\_validation\_predictions = True)  
glm\_classifier.train(x=range(1,foldpro\_df.ncol),   
 y=0,   
 training\_frame= foldpro\_df ,  
 seed = 1234 )

#glm\_classifier

Saving cross-validation predictions to file

cv\_pred\_glm = glm\_classifier.cross\_validation\_holdout\_predictions()

cv\_pred\_glm\_df = cv\_pred\_gbm.as\_data\_frame(use\_pandas=True)  
outDf2 = pd.DataFrame( zip (IdText.PairId , cv\_pred\_glm\_df.YES ) , columns = ["PairId","Probability"])

outDf2.to\_csv("glm\_prediction.csv" ,   
 sep = " " ,   
 header = False ,   
 index = False ,   
 quoting= csv.QUOTE\_NONE ,   
 quotechar='',  
 escapechar=' ')

Training a RandomForestEstimator & Saving cross-validation predictions to file

rf\_classifier = H2ORandomForestEstimator(ntrees=10,   
 max\_depth=20,   
 balance\_classes=False,  
 nfolds = 10 ,  
 keep\_cross\_validation\_predictions = True )  
  
rf\_classifier.train(x=range(1,foldpro\_df.ncol),   
 y=0,   
 training\_frame= foldpro\_df ,  
 seed = 1234 )  
  
cv\_pred\_rf = rf\_classifier.cross\_validation\_holdout\_predictions()  
cv\_pred\_rf\_df = cv\_pred\_rf.as\_data\_frame(use\_pandas=True)  
outDf3 = pd.DataFrame( zip (IdText.PairId , cv\_pred\_rf\_df.YES ) , columns = ["PairId","Probability"])  
outDf3.to\_csv("rf\_prediction.csv" ,   
 sep = " " ,   
 header = False ,   
 index = False ,   
 quoting= csv.QUOTE\_NONE ,   
 quotechar='',  
 escapechar=' ')

#rf\_classifier

Training a DeepLearningEstimator & Saving cross-validation predictions to file

dl\_classifier = H2ODeepLearningEstimator(activation = "Tanh",   
 hidden = [150, 150],   
 epochs = 20,  
 nfolds = 10,  
 keep\_cross\_validation\_predictions = True)  
  
dl\_classifier.train(x=range(1,foldpro\_df.ncol),   
 y=0,   
 training\_frame= foldpro\_df ,  
 seed = 1234 )  
  
cv\_pred\_dl = dl\_classifier.cross\_validation\_holdout\_predictions()  
cv\_pred\_dl\_df = cv\_pred\_dl.as\_data\_frame(use\_pandas=True)  
outDf4 = pd.DataFrame( zip (IdText.PairId , cv\_pred\_dl\_df.YES ) , columns = ["PairId","Probability"])  
outDf4.to\_csv("dl\_prediction.csv" ,   
 sep = " " ,   
 header = False ,   
 index = False ,   
 quoting= csv.QUOTE\_NONE ,   
 quotechar='',  
 escapechar=' ')

#dl\_classifier

4.4 Run time comparison : PySparkling ( Local PySpark + H2O) vs H2O Python

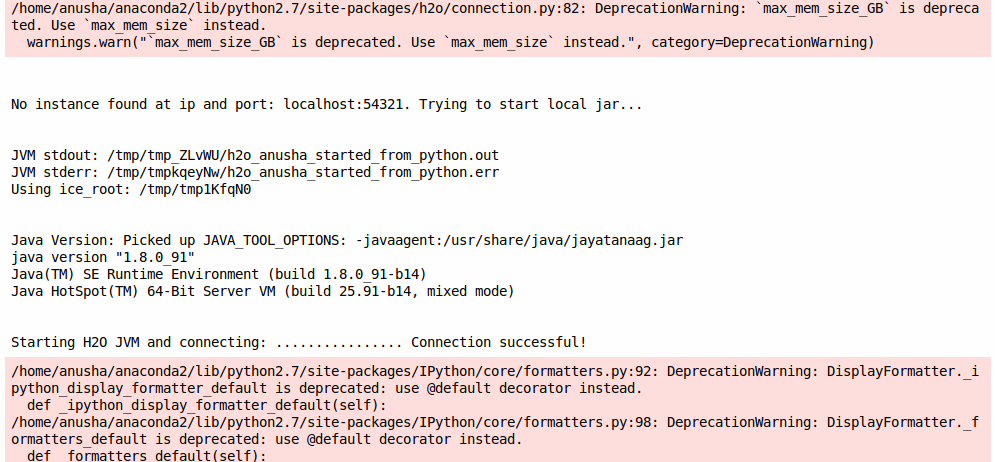
Same exact lines of Python code can be run in PySparkling mode and also run directly on H2O, it would be interesting to see if one mode implementation is vastly more efficient than other. Towards this, I run the same exact code both these modes and compare their run time.

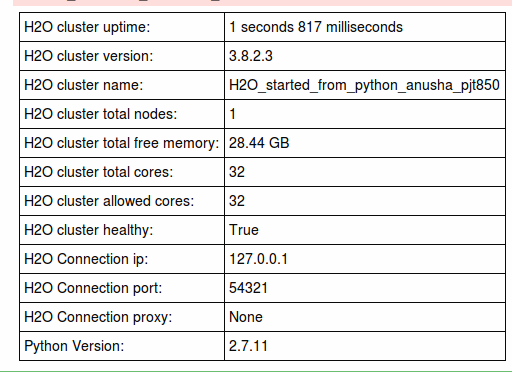
H2O Python

import h2o

from h2o.estimators import H2ORandomForestEstimator

h2o.init(nthreads = -1 , max\_mem\_size\_GB = 32)





%time foldpro\_df = h2o.import\_file("/home/anusha/cscie-63/project/foldpro\_final.csv")

rf\_classifier = H2ORandomForestEstimator(ntrees=10,

max\_depth=20,

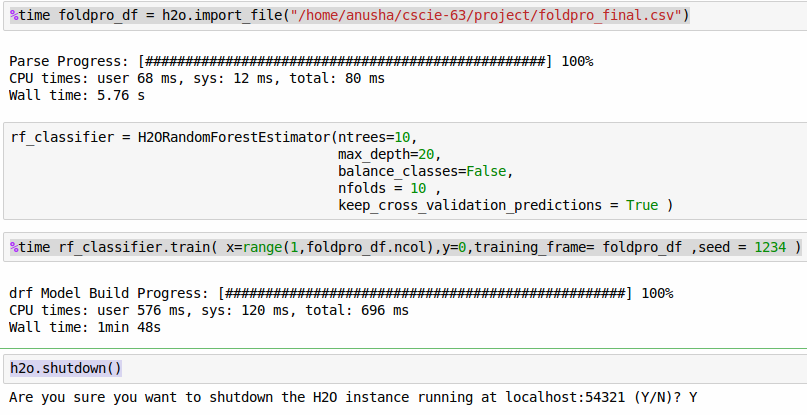
balance\_classes=False,

nfolds = 10 ,

keep\_cross\_validation\_predictions = True )

%time rf\_classifier.train( x=range(1,foldpro\_df.ncol),y=0,training\_frame= foldpro\_df ,seed = 1234 )

h2o.shutdown()



Pysparkling

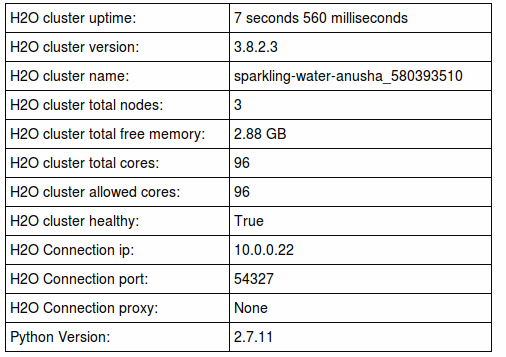
import h2o

from h2o.estimators import H2ORandomForestEstimator

from pysparkling import \*

import h2o

hc = H2OContext(sc).start()



%time foldpro\_df = h2o.import\_file("/home/anusha/cscie-63/project/foldpro\_final.csv")

rf\_classifier = H2ORandomForestEstimator(ntrees=10,

max\_depth=20,

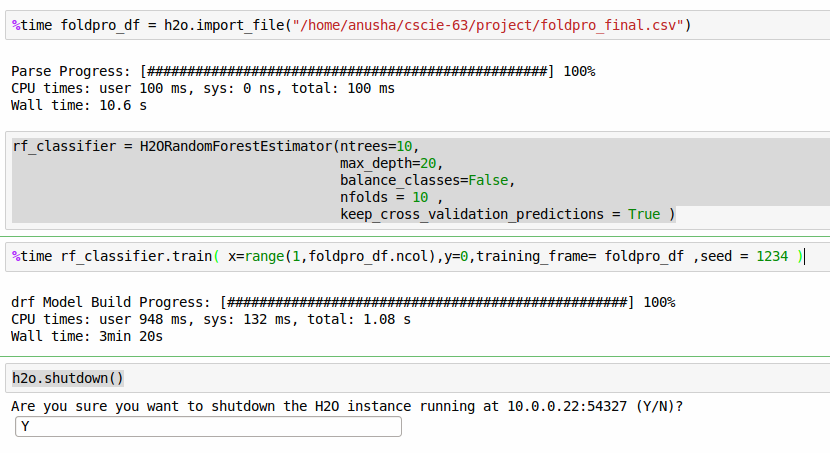
balance\_classes=False,

nfolds = 10 ,

keep\_cross\_validation\_predictions = True )

%time rf\_classifier.train( x=range(1,foldpro\_df.ncol),y=0,training\_frame= foldpro\_df ,seed = 1234 )

h2o.shutdown()



Analysis:

The results are tabulated in the table below.

|  |  |  |  |
| --- | --- | --- | --- |
|  | H2O + Python (ms) | H2O + Spark (ms) | % Increase in Time while in Spark mode |
| *Data loading* | 80 | 100 | 25.0% |
| *Random Forest Model Building* | 696 | 1008 | 44.8% |

As anticipated, Spark mode has longer execution times compared with running code directly. But, this long execution time is the trade-off for the ability to scale up the computing resources as necessary for the problem at hand.

4.5 Building ensemble model using H2O + R

h2oEnsemble is an R package which can used for building ensemble models and this package is currently available in R only.

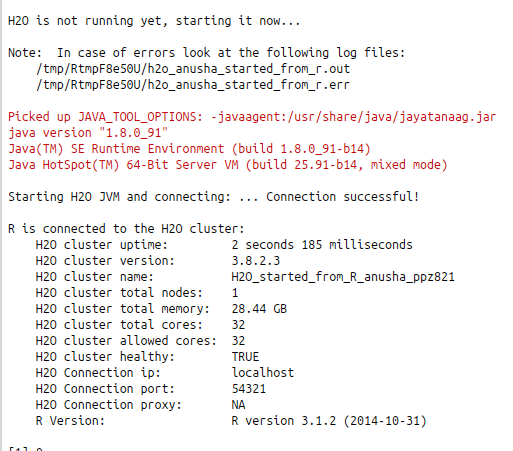
Hence the code for the following section was written in R instead of Python.

Starting H2O Cluster

library(h2oEnsemble)

h2o.init(nthreads = -1)

h2o.removeAll()



Loading Data into H2O Cluster

setwd("/home/anusha/cscie-63/project")

cv\_train <- h2o.importFile(path = normalizePath("./foldpro\_final.csv"))

y <- "LABEL"

x <- setdiff(names(cv\_train), y)

Converting response column to enum type

cv\_train[,y] <- as.factor(cv\_train[,y])

Specifying Base Learners & Metalearner

learner <- c("h2o.glm.wrapper", "h2o.randomForest.wrapper",

"h2o.gbm.wrapper", "h2o.deeplearning.wrapper")

metalearner <- "h2o.glm.wrapper"

Training + Cross-validation of Ensemble ; Folds = 10

foldCount = 10

ens <- h2o.ensemble(x = x, y = y,

training\_frame = cv\_train,

family = "binomial",

learner = learner,

metalearner = metalearner,

cvControl = list(V = foldCount))

leveloneDf = ens$levelone

folds = ens$folds

Level2 Cross validation prediction results are necessary to benchmark the h2o ensemble method against original paper. h2o Ensemble Object seems to provide no method to access internal Level2 cross validation predictions.Creating Level2 cross-validation predictions external to ensemble object for algorithm Benchmarking purposes.

Reference1 : http://mlwave.com/kaggle-ensembling-guide/

Reference2 : https://github.com/emanuele/kaggle\_pbr/blob/master/blend.py

cv\_probabilties = as.data.frame( rep( 0 , NROW(cv\_train) ) )

for (i in 1:foldCount){

print(paste("Fold Begin:", i))

train\_records = which(!( folds %in% i))

val\_records = which( folds %in% i)

glm\_cv = h2o.glm(training\_frame = leveloneDf[train\_records,] ,

#validation\_frame = leveloneDf[val\_records,],

x = learner,

y = "y",

family='binomial')

cv\_probabilties[val\_records,1] = as.data.frame( h2o.predict(object = glm\_cv, newdata = leveloneDf[val\_records,])[,3] )

print(paste("Fold Completed:", i))

}

AUC Scores : Comparison

library(cvAUC)

10 Fold - Ensemble AUC

cvAUC::AUC(predictions = cv\_probabilties, labels = as.data.frame(leveloneDf)[,"y"])

print ("10 Fold - Ensemble AUC")

print( cvAUC::AUC(predictions = cv\_probabilties, labels = as.data.frame(leveloneDf)[,"y"]) )

# 0.951009

10 Fold - Base learners AUC

L <- length(learner)

auc <- sapply(seq(L), function(l) cvAUC::AUC(predictions = as.data.frame(leveloneDf)[,l], labels = as.data.frame(leveloneDf)[,"y"] ))

print ( data.frame(learner, auc) )

# learner auc

#1 h2o.glm.wrapper 0.9372006

#2 h2o.randomForest.wrapper 0.9500034

#3 h2o.gbm.wrapper 0.9397503

#4 h2o.deeplearning.wrapper 0.9048005

Creating a predictions output file to Compute Top1/Top5 sensitivity score for Benchmarking purpose.

Ref: data.table http://stackoverflow.com/questions/9703068/most-efficient-way-of-exporting-large-3-9-mill-obs-data-frames-to-text-file

Id = read.csv("IdFile.txt" , header =FALSE)

Id$Pred = as.array( cv\_probabilties )

fwrite( Id , "ensemble\_prediction.csv" , sep=" ", col.names=FALSE , quote = FALSE)

#### All done, shutdown H2O

h2o.shutdown(prompt=FALSE)

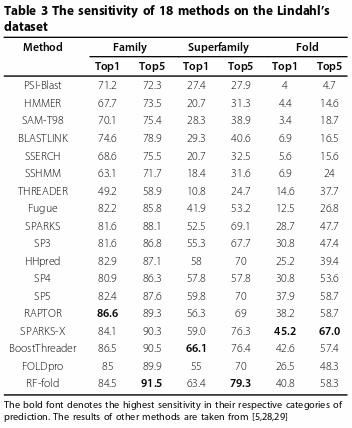
5. Results

The sensitivity for top-one ( resp. top-five ) ranked template is defined as the percentage of target proteins having at least one correct template ranked no.1 ( resp. within top-5 ) by the Prediction algorithm.

The known values of Top1/Top5 sensitivities corresponding to the existing algorithms on this dataset are as follows. I shall be benchmarking the performance of H2O algorithms against these numbers shortly.

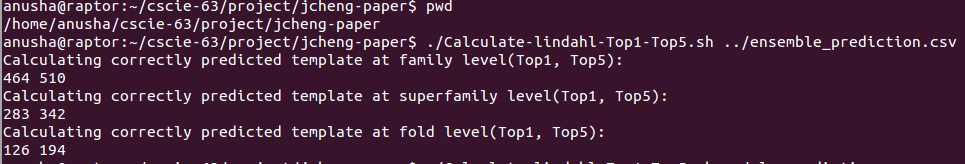
Source: Jo and Cheng *BMC Bioinformatics* 2014, 15(Suppl 11):S14

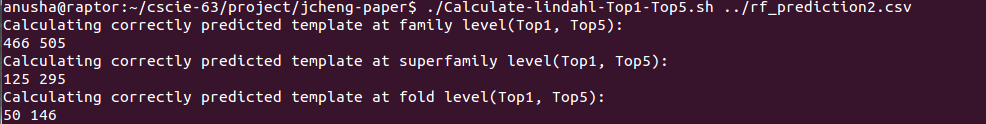
http://www.biomedcentral.com/1471-2105/15/S11/S14

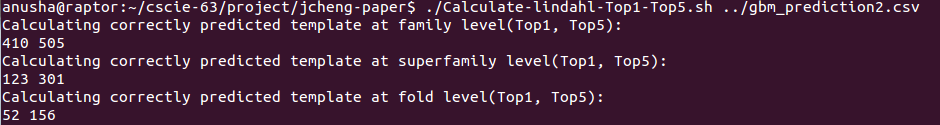


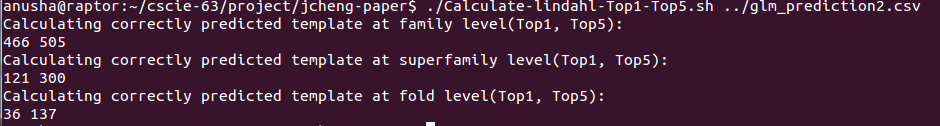
The authors have provided computational tools( <http://calla.rnet.missouri.edu/rf-fold/>) for evaluating performance on the benchmark dataset. The script ( Calculate-lindahl-Top1-Top5.sh) accepts a two-column file as input, with column1 containing a string identifier for target-template protein pair and column2 containing probability that target & template belong to the same fold. The output of the script is the value of top-one & top-five sensitivity of the protein-fold recognition.

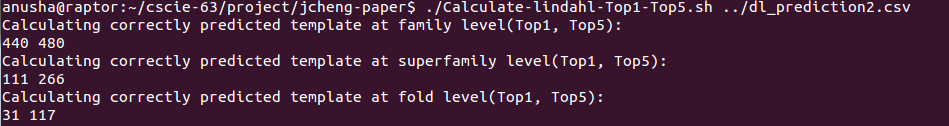
Below I proceed to compute the sensitivity metrics from the output files created by earlier during 10-fold cross validated model training. In order to use the program though, I have to first rescale the prediction values in the range [0.1,1] though. The python program I wrote for the same can be found in Appendix -2









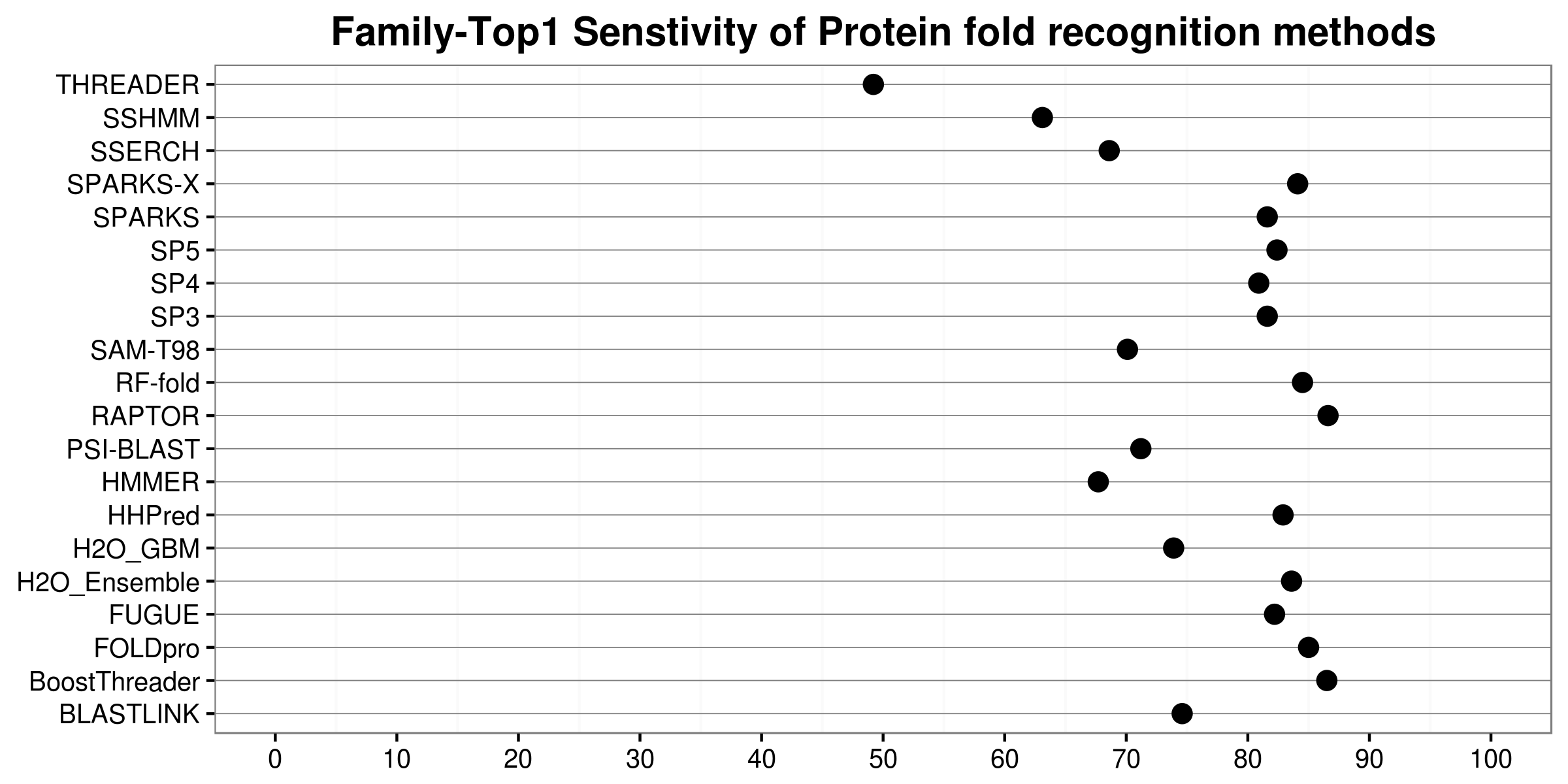


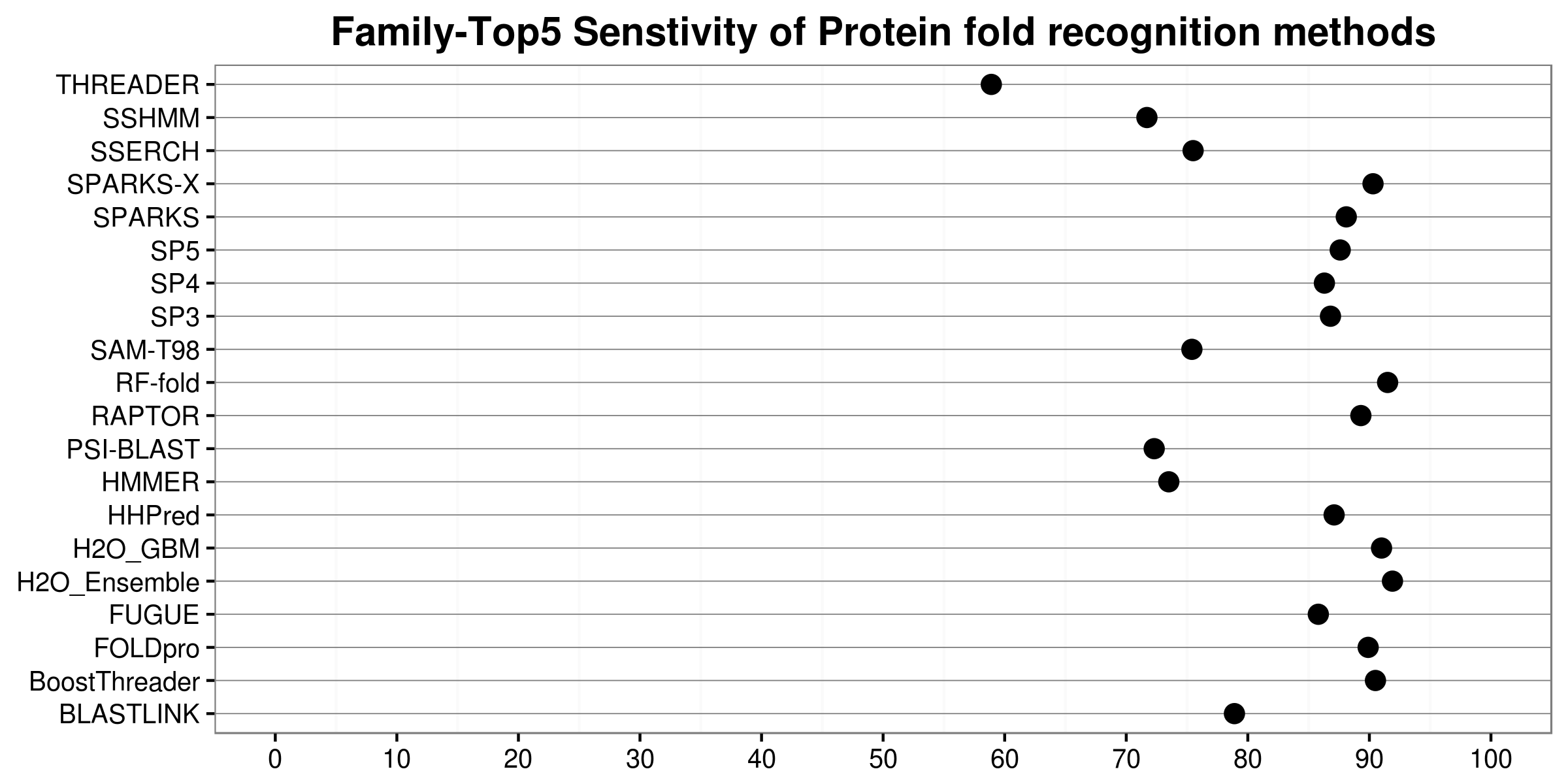
The outputs are summarized in the table below.

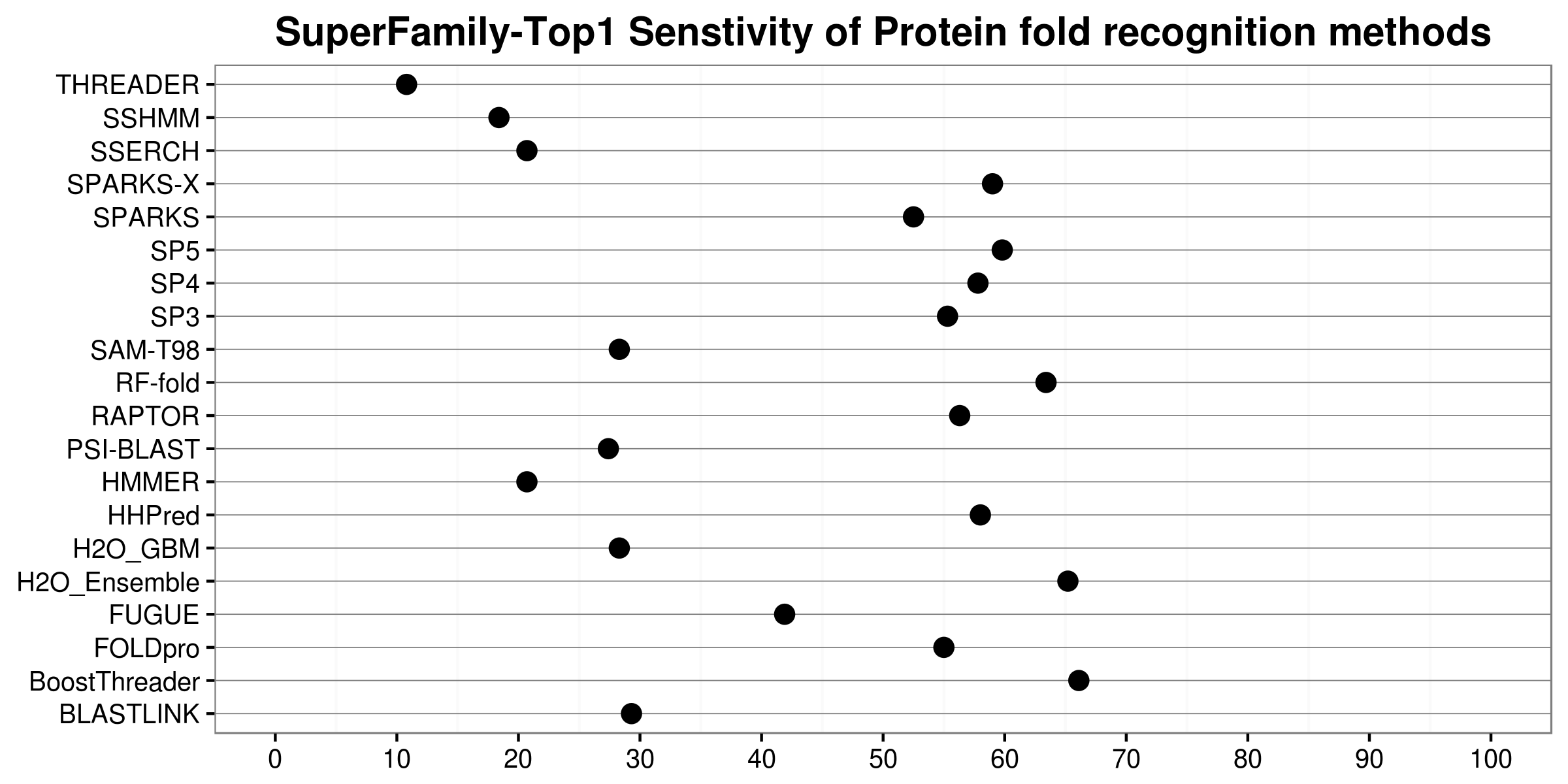
|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| # Correct Predictions | **Family** | | **Superfamily** | | **Fold** | |
| **Top1** | **Top5** | **Top1** | **Top5** | **Top1** | **Top5** |
| Ensemble of random choice of Base learners in R | 464 | 510 | 283 | 342 | 126 | 194 |
| GBM (Parameter Optimized by line search) | 410 | 505 | 123 | 301 | 52 | 156 |
| DRF | 466 | 505 | 125 | 295 | 50 | 146 |
| GLM | 466 | 505 | 121 | 300 | 36 | 137 |
| DL | 440 | 480 | 111 | 266 | 31 | 117 |

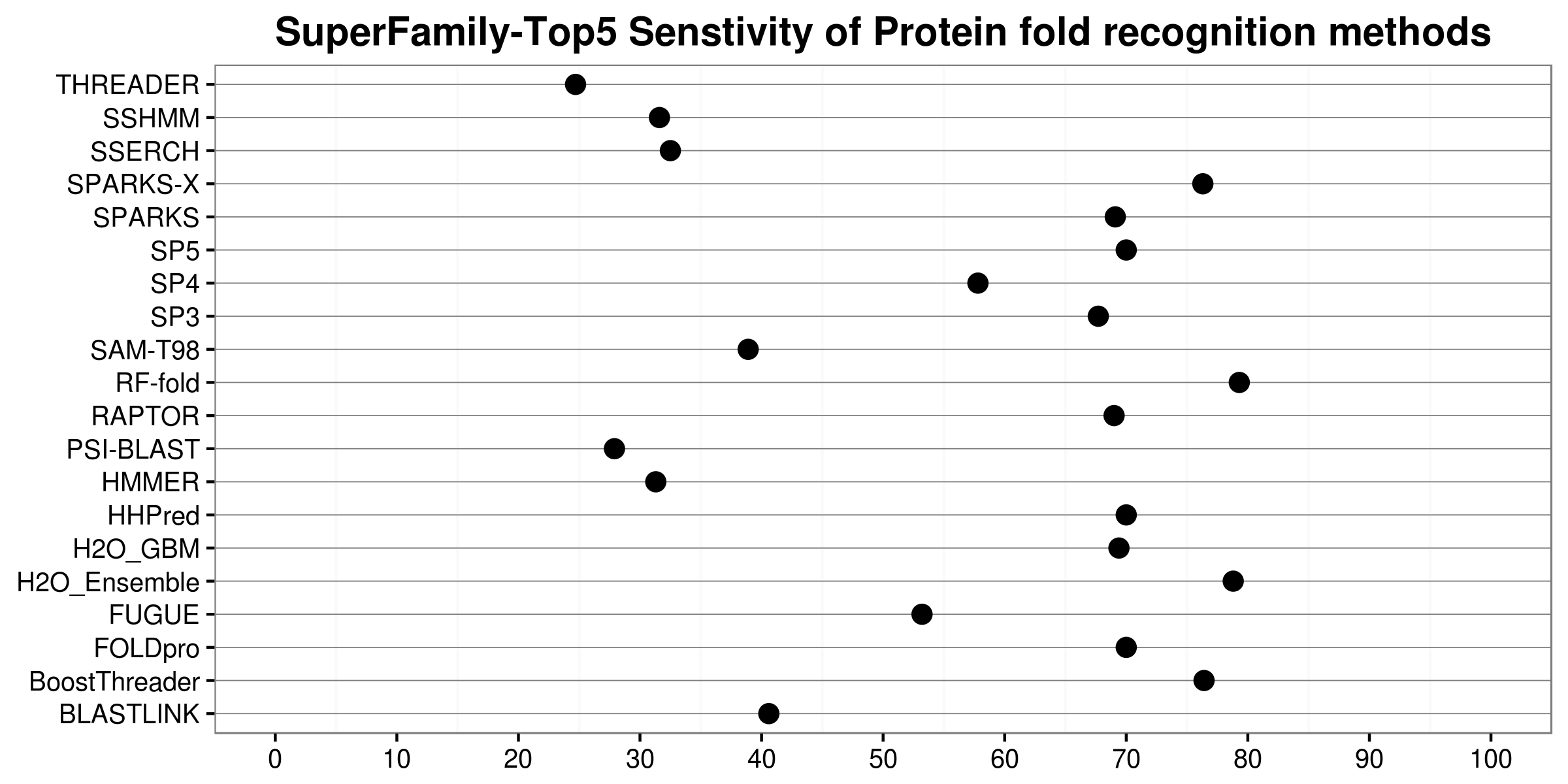
|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Sensitivity** | **Family** | | **Superfamily** | | **Fold** | |
| **Top1** | **Top5** | **Top1** | **Top5** | **Top1** | **Top5** |
| Ensemble of random choice of Base learners in R | 83.6 | 91.9 | 65.2 | 78.8 | 39.3 | 60.4 |
| GBM (Parameter Optimized by line search) | 73.9 | 91.0 | 28.3 | 69.4 | 16.2 | 48.6 |
| DRF | 84.0 | 91.0 | 28.8 | 68.0 | 15.6 | 45.5 |
| GLM | 84.0 | 91.0 | 27.9 | 69.1 | 11.2 | 42.7 |
| DL | 79.3 | 86.5 | 25.6 | 61.3 | 9.7 | 36.4 |

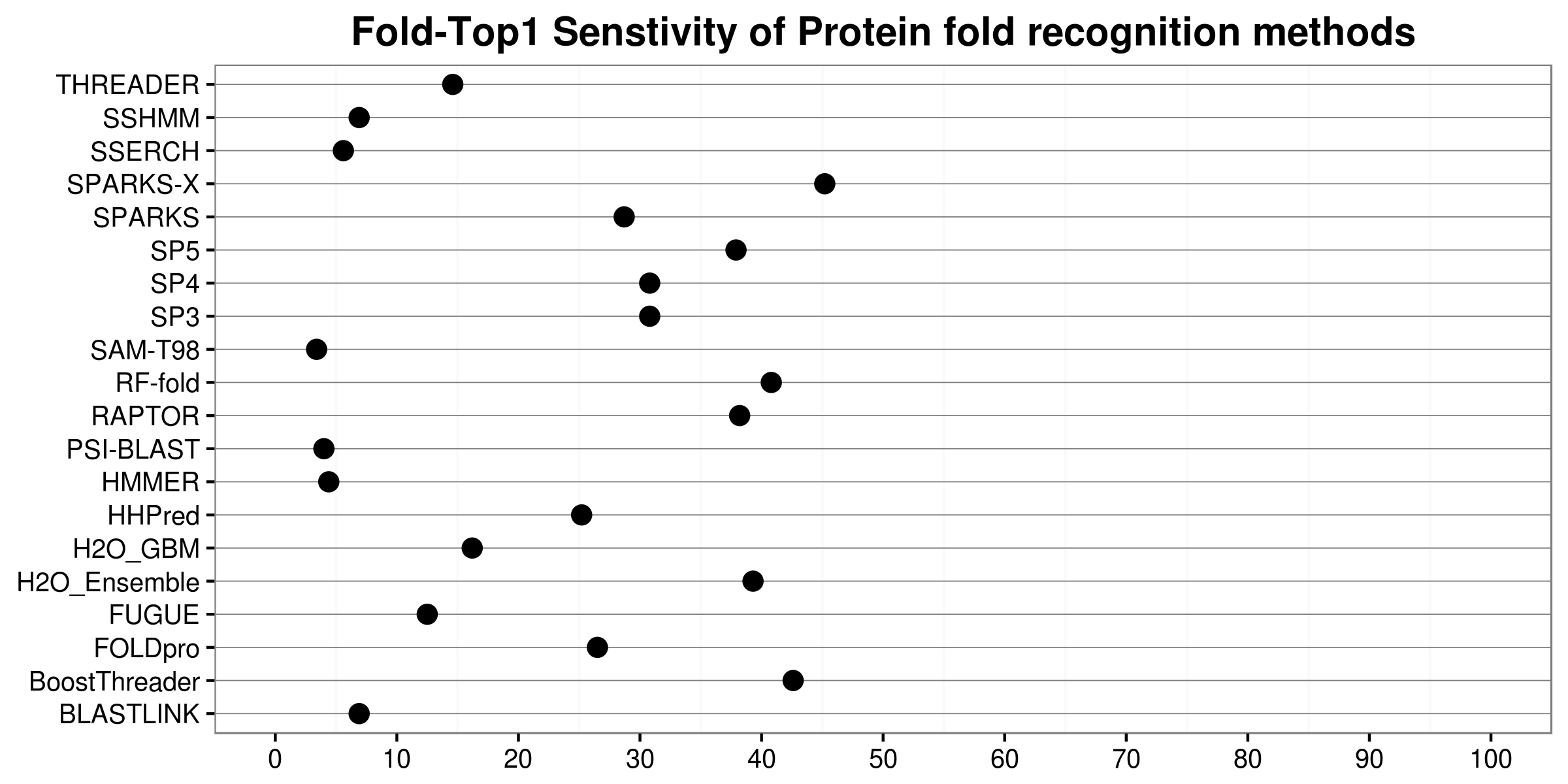
From the examination of the results, it can be seen the H2O Ensemble model is performing at least as good the paper author’s random forest model ( RF - Fold in the chart below )

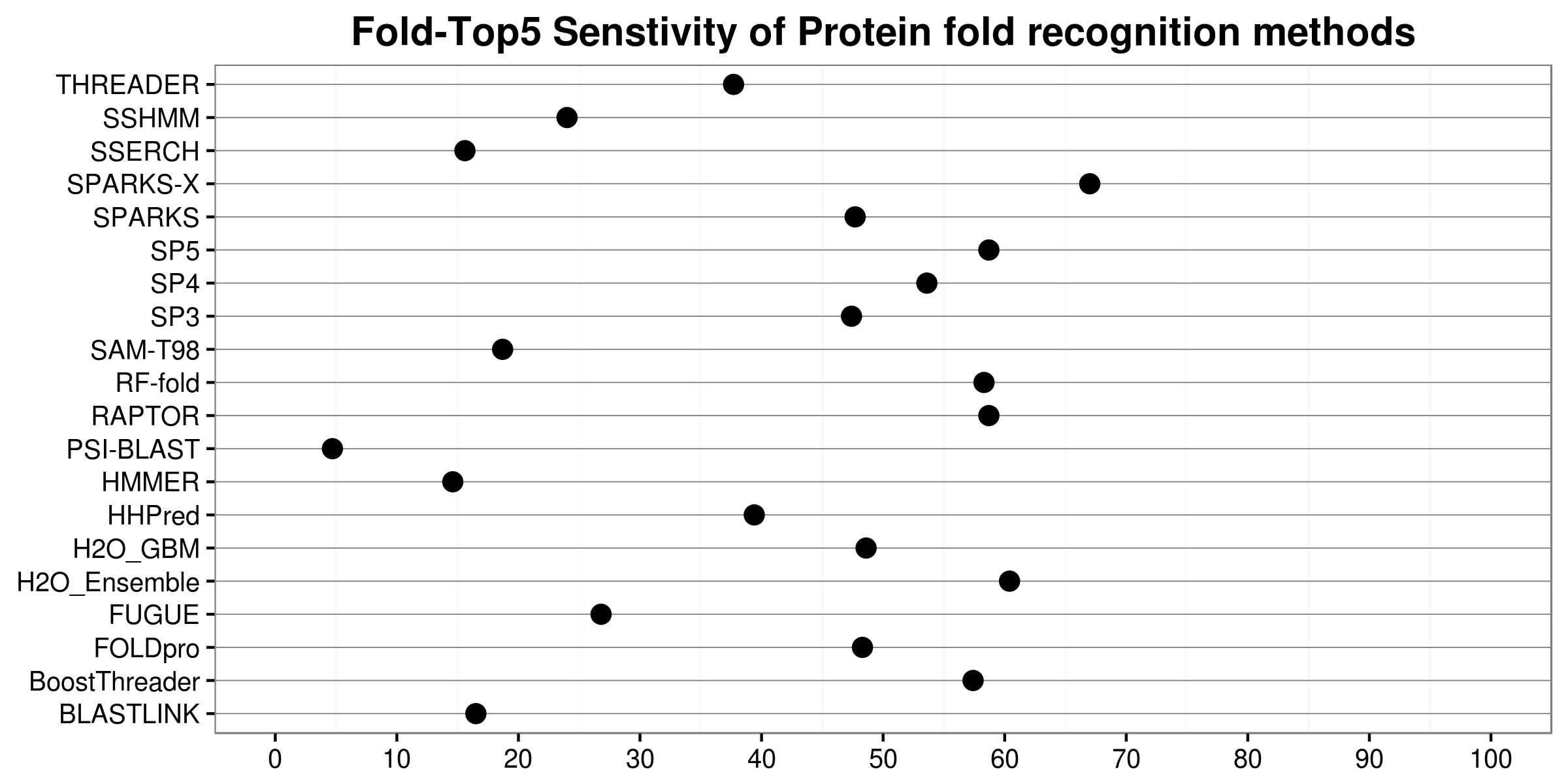












6. Pros/Cons of H2O & Learnings from Project

Pro:

1. H2O Flow is a very efficient tool and can allow for fast iteration cycle.
2. The software did not crash for the most part except for errors from the input side. The software seems to reliable & mature
3. With so many different APIs it is very easy to integrate H2O with things I already know

Cons:

1. H2O has too much documentation available on the web and it seems some of them is based on old versions of H2O and it can get too confusing.
2. H2O did could use more functions which can help building cross-validation of ensemble modeling easy
3. The SVM light parser which came with software was not able to parse the file correctly. So, there is definitely room for improvement in the supported features list.

8.1 Appendix - I

Following StackOverflow solution was used to split the odd & even lines into separate files:

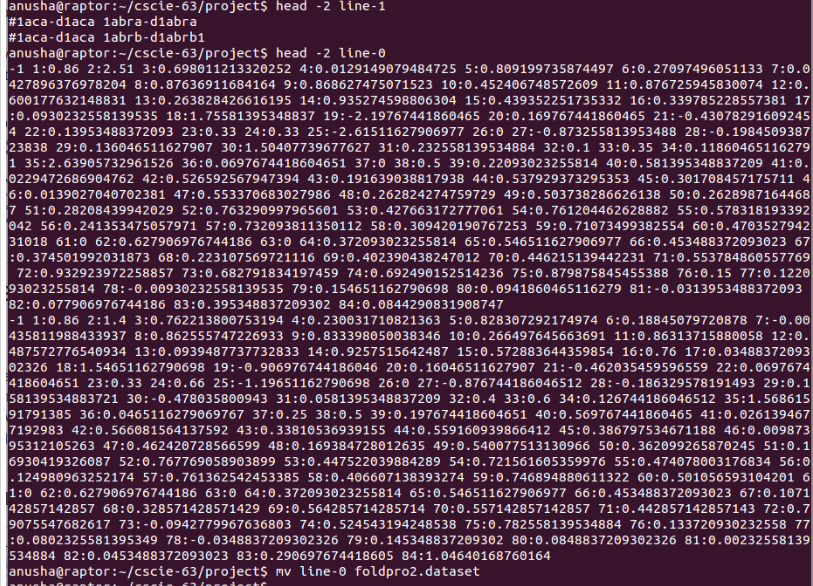
<http://unix.stackexchange.com/questions/26723/print-odd-numbered-lines-print-even-numbered-lines>

awk '{print>"line-"NR%2}' foldpro.dataset



head -2 line-1

head -2 line-0



mv line-0 foldpro2.dataset

*Creating an IdFile after removing # character*

cut -c 2- line-1 > IdFile.txt

Following open source software was used to convert cleaned LIBSVM format to CSV format

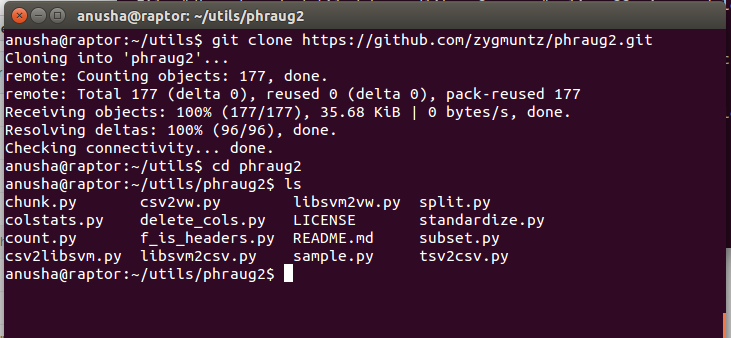
<https://github.com/zygmuntz/phraug2>

Usage notes from the website :

Convert LIBSVM to CSV. You need to specify dimensionality, that is a number of columns (not counting a label).

libsvm2vw.py <input file> <output file>

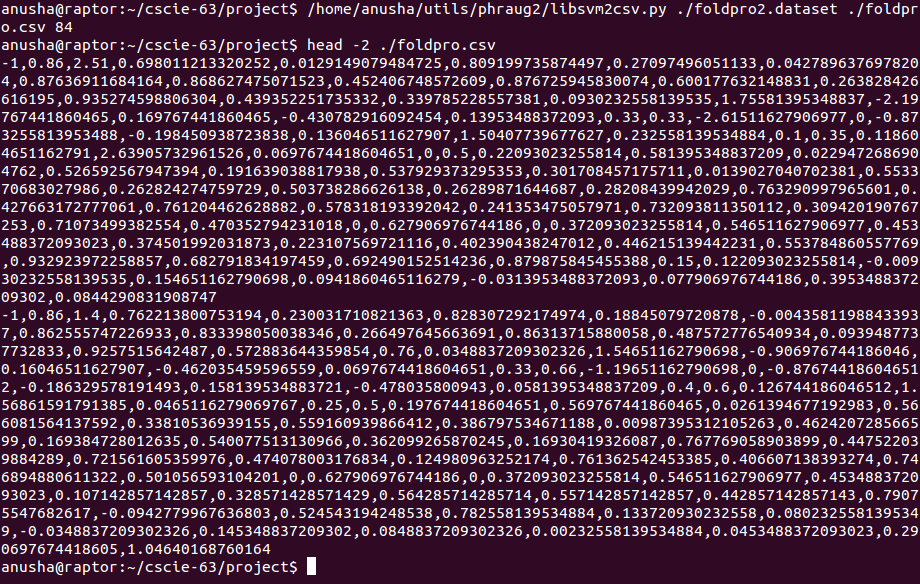
Git clone for software installation



Running phraug2 & examining the created CSV file

/home/anusha/utils/phraug2/libsvm2csv.py ./foldpro.dataset ./foldpro.csv 84

head -2 ./foldpro.csv



Since the plan is to run classification algorithms on this dataset might as well change the label from [-1,1] -> [NO, YES] so that a conversion is not required at a later stage. Following python code was used for the same.

file\_in = open("foldpro.csv","r")

file\_out = open("foldpro2.csv","w")

for line in file\_in:

tokens = line.split(",")

if tokens[0] == '-1' :

tokens[0] = '"NO"'

if tokens[0] == '+1' :

tokens[0] = '"YES"'

file\_out.write( ','.join(tokens) )

file\_in.close()

file\_out.close()

New header record created by hand after reading details of the dataset at - <http://contact.ics.uci.edu/foldpro_feature_list.txt>

echo LABEL,QUERY\_LENGTH,TARGET\_LENGTH,COSINE\_COMP\_MONOMER\_QUERY\_TEMPLATE\_SEQ,CORRELATION\_COMP\_MONOMER\_QUERY\_TEMPLATE\_SEQ,GAUSSIAN\_FUNCTION\_COMP\_MONOMER\_QUERY\_TEMPLATE\_SEQ,COSINE\_COMP\_DIMER\_QUERY\_TEMPLATE\_SEQ,CORRELATION\_COMP\_DIMER\_QUERY\_TEMPLATE\_SEQ,GAUSSIAN\_FUNCTION\_COMP\_DIMER\_QUERY\_TEMPLATE\_SEQ,COSINE\_COMP\_MONOMER\_QUERY\_TEMPLATE\_FAMILIES,CORRELATION\_COMP\_MONOMER\_QUERY\_TEMPLATE\_FAMILIES,GAUSSIAN\_FUNCTION\_COMP\_MONOMER\_QUERY\_TEMPLATE\_FAMILIES,COSINE\_COMP\_DIMER\_QUERY\_TEMPLATE\_FAMILIES,CORRELATION\_COMP\_DIMER\_QUERY\_TEMPLATE\_FAMILIES,GAUSSIAN\_FUNCTION\_COMP\_DIMER\_QUERY\_TEMPLATE\_FAMILIES,PALIGN\_SEQ\_ALIGNMENT\_SCORE\_1,PALIGN\_SEQ\_ALIGNMENT\_SCORE\_2,CLUSTALW\_SEQ\_ALIGNMENT\_SCORE,CLUSTALW\_PROFILE\_PROFILE\_ALIGNMENT\_SCORE,LOBSTER\_PROFILE\_PROFILE\_ALIGNMENT\_SCORE,PSIBLAST\_PROFILE\_SEQ\_ALIGNMENT\_SCORE,PSIBLAST\_ALIGNMENT\_EVALUE,PSIBLAST\_NORMALIZED\_ALIGNMENT\_LENGTH,PSIBLAST\_IDENTITY\_RATE\_ALIGNMENT,PSIBLAST\_POSITIVE\_RATE\_ALIGNMENT,PFAM\_ALIGNMENT\_SCORE,PFAM\_ALIGNMENT\_EVALUE,SEARCH\_ALIGNMENT\_SCORE,SEARCH\_ALIGNMENT\_EVALUE,IMPALA\_SEQ\_PROFILE\_ALIGNMENT\_SCORE,IMPLALA\_ALIGNMENT\_EVALUE,IMPALA\_NORMALIZED\_ALIGNMENT\_LENGTH,IMPALA\_ALIGNMENT\_IDENTITY\_RATE,IMPALA\_ALIGNMENT\_POSITIVE\_RATE,RPSBLAST\_SEQ\_PROFILE\_ALIGNMENT\_SCORE,RPSBLAST\_ALIGNMENT\_EVALUE,RPSBLAST\_ALIGNMENT\_NORMALIZED\_LENGTH,RPSBLAST\_ALIGNMENT\_IDENTITY\_RATE,RPSBLAST\_ALIGNMENT\_POSITIVE\_RATE,SECONDARY\_STRUCTURE\_MATCH\_RATIO,RELATIVE\_SOLVENT\_ACCESSIBILITY\_MATCH\_RATIO,AVG\_CONTACT\_PROBABILITY\_cm\_8Angstrom,COSINE\_RESIDUE\_CONTACT\_NUM\_cm\_8Angstrom,CORRELATION\_RESIDUE\_CONTACT\_NUM\_cm\_8Angstrom,COSINE\_RESIDUE\_CONTACT\_ORDER\_cm\_8Angstrom,CORRELATION\_RESIDUE\_CONTACT\_ORDER\_cm\_8Angstrom,AVG\_CONTACT\_PROBABILITY\_bm\_8Angstrom,COSINE\_RESIDUE\_CONTACT\_NUM\_bm\_8Angstrom,CORRELATION\_RESIDUE\_CONTACT\_NUM\_bm\_8Angstrom,COSINE\_RESIDUE\_CONTACT\_ORDER\_bm\_8Angstrom,CORRELATION\_RESIDUE\_CONTACT\_ORDER\_bm\_8Angstrom,AVG\_CONTACT\_PROBABILITY\_cm\_12Angstrom,COSINE\_RESIDUE\_CONTACT\_NUM\_cm\_12Angstrom,CORRELATION\_RESIDUE\_CONTACT\_NUM\_cm\_12Angstrom,COSINE\_RESIDUE\_CONTACT\_ORDER\_cm\_12Angstrom,CORRELATION\_RESIDUE\_CONTACT\_ORDER\_cm\_12Angstrom,AVG\_CONTACT\_PROBABILITY\_bm\_12Angstrom,COSINE\_RESIDUE\_CONTACT\_NUM\_bm\_12Angstrom,CORRELATION\_RESIDUE\_CONTACT\_NUM\_bm\_12Angstrom,COSINE\_RESIDUE\_CONTACT\_ORDER\_bm\_12Angstrom,CORRELATION\_RESIDUE\_CONTACT\_ORDER\_bm\_12Angstrom,BETA\_RESIDUE\_BETA\_SHEET\_PAIRING\_PROBABILITY,PERCENTAGE\_HELIX\_QUERY,PERCENTAGE\_BETA\_STRAND\_QUERY,PERCENTAGE\_COIL\_QUERY,PERCENTAGE\_EXPOSED\_RESIDUE\_QUERY,PERCENTAGE\_BURIED\_RESIDUE\_QUERY,PERCENTAGE\_HELIX\_TEMPLATE,PERCENTAGE\_BETA\_STRAND\_TEMPLATE,PERCENTAGE\_COIL\_TEMPLATE,PERCENTAGE\_EXPOSED\_RESIDUE\_TEMPLATE,PERCENTAGE\_BURIED\_RESIDUE\_TEMPLATE,COSINE\_SS\_SA\_COMP,CORRELATION\_SS\_SA\_COMP,GAUSSIAN\_FUNC\_SS\_SA\_COMP,DOT\_PRODUCT\_SS\_SA\_COMP,HMMER\_PRCHMMM\_COEMIS\_SCORE,HMMER\_PRCHMMM\_SIMPLE\_SCORE,HMMER\_PRCHMMM\_REVERSE\_SCORE,CHK\_PRCHMMM\_COEMIS\_SCORE,CHK\_PRCHMMM\_SIMPLE\_SCORE,CHK\_PRCHMMM\_REVERSE\_SCORE,HHSEARCH\_PROFILE\_PROFILE\_ALIGNMENT\_SCORE,COMPASS\_ALIGNMENT\_SCORE,COMPASS\_EVALUE > header.txt

Appending header by creating a new file

cat header.txt foldpro2.csv > foldpro\_final.csv

8.1 Appendix - II

Python code to rescale the probabilities in the range [0.1,1] : ReScale\_Predictions.py

Usage

./ReScale\_Predictions.py <input\_file> <output\_file>

#!/home/anusha/anaconda2/bin/python

import sys

import pandas as pd

import csv

from sklearn.preprocessing import MinMaxScaler

import warnings

warnings.filterwarnings("ignore")

r = MinMaxScaler((0.01,1))

inputFile = open( sys.argv[1] , "r" )

id\_list = []

data\_list = []

for line in inputFile :

tokens = line.split(" ")

data\_list.append ( float(tokens[-1]))

id\_list.append ( " ".join (tokens[:-1]))

c2 = r.fit\_transform ( data\_list)

outDf1 = pd.DataFrame( zip ( id\_list , c2 ) , columns = ["PairId","Probability"])

outDf1.to\_csv(sys.argv[2] ,

sep = " " ,

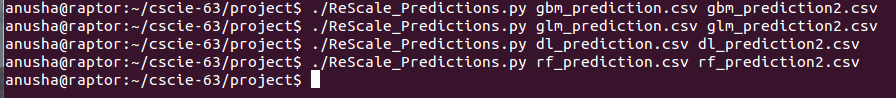
header = False ,

index = False ,

quoting= csv.QUOTE\_NONE ,

quotechar='',

escapechar=' ')



**YouTube URLs:**

2 min video: <https://www.youtube.com/watch?v=v2yGoV2HDTc>

15min video: <https://www.youtube.com/watch?v=-q4orm8zFDQ>