Brief machine learning intro with R and H2O

Maximilian Stroh

Data User Group - 05. Sept. 2019

Outlook

- Load and inspect dataset on fuel efficiency of cars
- Use machine learning to predict fuel efficiency
- Understand a fitted machine learning model

Fuel efficiency data

- Use dataset included in the package ‘ggplot2’ called **mpg**
- Includes information about different cars and their fuel efficiency measured by miles per gallon

Column	**Explanation**
manufacturer	
model	
displ	engine displacement, in litres
year	year of manufacture
cyl	number of cylinders
trans	type of transmission
drv	f = front-wheel drive, r = rear wheel drive, 4 = 4wd
cty	city miles per gallon
hwy	highway miles per gallon
fl	fuel type
class	type of car

Quick look into the dataset

```r
mpg = ggplot2::mpg
```

```r
mpg[c(1,51,101,151,201),]
```

manufacturer	model	displ	year	cyl	trans	drv	cty	hwy	fl	class
audi	a4	1.8	1999	4	auto(l5)	f	18	29	p	compact
dodge	dakota pickup 4wd	3.9	1999	6	auto(l4)	4	13	17	r	pickup
honda	civic	1.6	1999	4	auto(l4)	f	24	32	r	subcompact
nissan	pathfinder 4wd	3.3	1999	6	auto(l4)	4	14	17	r	suv
toyota	toyota tacoma 4wd	2.7	1999	4	manual(m5)	4	15	20	r	pickup

Basic statistics about the dataset

- Numeric columns look well behaved, no outlier treatment or robust method required

```r
summary(mpg)
```

```
##  manufacturer          model               displ            year     
##  Length:234         Length:234         Min.   :1.600   Min.   :1999  
##  Class :character   Class :character   1st Qu.:2.400   1st Qu.:1999  
##  Mode  :character   Mode  :character   Median :3.300   Median :2004  
##                                        Mean   :3.472   Mean   :2004  
##                                        3rd Qu.:4.600   3rd Qu.:2008  
##                                        Max.   :7.000   Max.   :2008  
##       cyl           trans               drv                 cty       
##  Min.   :4.000   Length:234         Length:234         Min.   : 9.00  
##  1st Qu.:4.000   Class :character   Class :character   1st Qu.:14.00  
##  Median :6.000   Mode  :character   Mode  :character   Median :17.00  
##  Mean   :5.889                                         Mean   :16.86  
##  3rd Qu.:8.000                                         3rd Qu.:19.00  
##  Max.   :8.000                                         Max.   :35.00  
##       hwy             fl               class          
##  Min.   :12.00   Length:234         Length:234        
##  1st Qu.:18.00   Class :character   Class :character  
##  Median :24.00   Mode  :character   Mode  :character  
##  Mean   :23.44                                        
##  3rd Qu.:27.00                                        
##  Max.   :44.00                                        
```

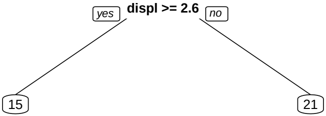
Put away some test data

```
## 75% of the sample size
smpSize <- floor(0.75 * nrow(mpg))

## set the seed to make your partition reproducible
set.seed(123)
trainInd <- sample(seq_len(nrow(mpg)), size = smpSize)

trainData <- mpg[trainInd, ]
testData <- mpg[-trainInd, ]
```

- Financial data often has a time dimension. Don't use simple random sampling to generate a test set in this case!

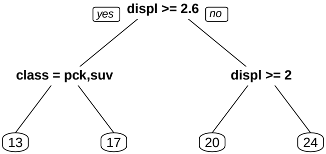
Regression trees - Simplest possible tree

```
tree1 = rpart(cty ~ displ + year + cyl + trans
                     + drv + fl + class, trainData,
                     model = T,
                     control = rpart.control(maxdepth = 1))

prp(tree1)
```



Tree of depths 2

```
tree2 = rpart(cty ~ displ + year + cyl + trans + drv
              + fl + class, trainData, model = T,
              control = rpart.control(maxdepth = 2))
```



Evaluating prediction performance on test set

```
x = c("displ","year","cyl","trans","drv","fl","class")

# use depth 1 regression tree to predict cyl on test set
testPrediction = predict(tree1, testData[,x])

# calc squared errors
predEvaluation = as.data.frame(cbind(testData$displ,testData$cty,testPrediction,
                                      (testData$cty - testPrediction)**2))
colnames(predEvaluation) = c("displ","realized","predicted","squared_error")
```

displ	realized	predicted	squared_error
1.8	21	21.07143	0.005102
2.0	20	21.07143	1.147959
2.0	21	21.07143	0.005102
2.8	18	14.52941	12.044983
3.1	17	14.52941	6.103806
5.7	16	14.52941	2.162630

```
# calc root mean squared error
sqrt(mean(predEvaluation$squared_error))
```

```
## [1] 2.549424
```

From tree to forest

- Simple regression trees as shown above are the basis for better performing methods
- One example is the **Random Forest** algorithm
- It combines forecasts from many trees
- Trees are grown *independently*
  - Draws different bootstrap samples of data, fits regression tree to each sample, then averages forecasts (*bagging*)
  - At each split, only random sample of features is used as split candidates, thus further decorrelates trees
- Uses larger trees and averages over them to reduce variance

R package "h2o"

- Comes with a couple of popular machine learning algorithms
  - Lasso, Ridge Regression, Random Forest, Gradient Boosting, Neural Nets...
- Estimates models on all cores of your machine
- Easy to set up on multiple machines to estimate a distributed model (for big data)
- Includes methods to understand a fitted model
- Limited capabilities in NN/Deep Learning

Replicating a single tree with the Random Forest algorithm

```r
# start h2o instance on this computer (requires Java)
h2o.init()
# upload data to h2o instance
h2o.train = as.h2o(trainData)
h2o.test  = as.h2o(testData)


# replication of simple regression tree with depth = 1
h2o.RF1 = h2o.randomForest(x,"cty",h2o.train,
                           ntrees = 1,max_depth = 1,
                           mtries=7,sample_rate =1,
                           col_sample_rate_per_tree=1,
                           nbins=175,
                           build_tree_one_node = T)
```

Use fitted Random Forest model to predict on test set

```r
# predict miles per galon for city usage on test set
h2o.RF1.pred = as.data.frame(h2o.predict(h2o.RF1,h2o.test))

# compare predictions to previous model
predEvaluation = cbind(predEvaluation,h2o.RF1.pred)

# name added column
colnames(predEvaluation)[5] = "predicted_h2o"
```

Successful replication of previous 1-split toy example

displ	realized	predicted	squared_error	predicted_h2o
1.8	21	21.07143	0.005102	21.07143
2.0	20	21.07143	1.147959	21.07143
2.0	21	21.07143	0.005102	21.07143
2.8	18	14.52941	12.044983	14.52941
3.1	17	14.52941	6.103806	14.52941
5.7	16	14.52941	2.162630	14.52941

```r
# calculate prediction performance stats on test set
h2o.RF1.perf = h2o.performance(h2o.RF1,h2o.test)
# display root mean squared error
h2o.RF1.perf@metrics$RMSE
```

```
## [1] 2.549424
```

Fit real Random Forest

- Random Forest algorithm has many “hyperparameters”"
- Two important ones are
  - **`ntrees`** Number of trees to fit to average over
  - **`max_depth`** Size of each tree

```
# use 50 trees, with depth up to 20 (h2o defaults)
h2o.RF = h2o.randomForest(x,"cty",h2o.train,
                          ntrees = 50,
                          max_depth = 10,
                          seed = 123)
```

Prediction error is halfed compared to toy example

```r
# calculate prediction performance stats on test set
h2o.RF.perf = h2o.performance(h2o.RF,h2o.test)
# display root mean squared error
h2o.RF.perf@metrics$RMSE
```

```
## [1] 1.112829
```

Tuning: How to chose the right hyperparameters?

- **Never** look at the test set and play around until it works
  - Now you have fitted the hyperparameters on the test set
  - No more data left to see if it really works on new data
  - Can be OK if you can easily collect new data for final test
- Need some criterion to chose hyperparameters based only on training set
- Basic idea: Split your training set again, estimate on one part, evaluate on another
- Can do this multiple times to have enough training data to estimate the model on
- Popular choice: 5-fold cross validation
  - Split trainig set into 5 parts
  - Train on 4/5 of data, evaluate on 1/5
  - Do this 5 times, average over results

5-fold cross validation in one picture

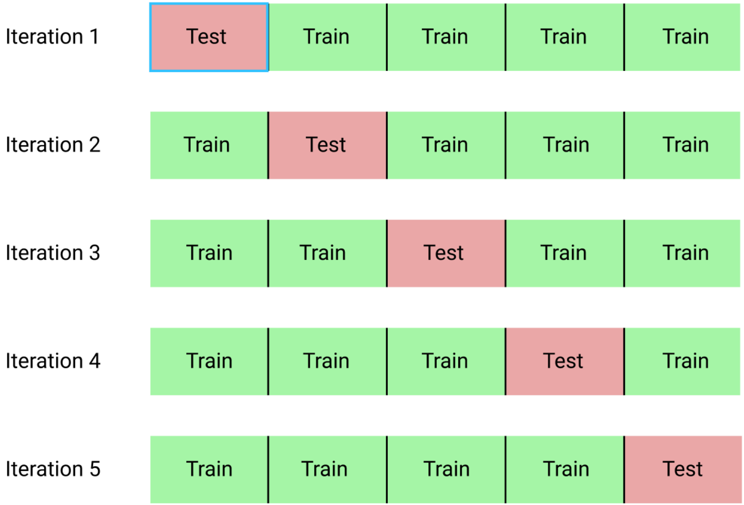


Figure 1: 5-fold cross validation

Calc CV-stats while training the model

```r
# Fit model and calc 5-fold cross validation performance
h2o.RF = h2o.randomForest(x,"cty",h2o.train,
                          ntrees = 50, max_depth = 10,
                          nfolds = 5, seed = 123)
```

```r
cvMetrics = h2o.RF@model$cross_validation_metrics_summary[6,]
```

Table 2: RMSE on cross-validation subsets

mean	sd	cv_1_valid	cv_2_valid	cv_3_valid	cv_4_valid	cv_5_valid
1.4798	0.4308	1.1273	1.1386	1.4934	2.1875	1.452

Use cross validation RMSEs to optimize hyperparameters

```r
# Define possible value of hyperparameters
set.seed(123); RF_params =
  list(ntrees = round(exp(runif(5,log(10),log(1000)))),
       max_depth = round(exp(runif(5,log(1),log(50)))))
```

```
## $ntrees
## [1]  38 377  66 583 760
##
## $max_depth
## [1]  1  8 33  9  6
```

Cross-validate all 25 Random Forest models

```r
# Train and validate a cartesian grid of GBMs
RF_grid = h2o.grid("randomForest", x = x, y = "cty",
                   grid_id = "RF_grid",
                   training_frame = h2o.train,
                   nfolds = 5,
                   seed = 123,
                   hyper_params = RF_params)
```

- H2O will train models for all possible combinations
- i.e. $(38, 1), (38, 8), \ldots, (760, 9), (760, 6)$
- This is called a 'Cartesian' grid
- It is not the most effcient tuning method in most cases
- But easy to understand and OK for searching across just 2 hyperparameters

Show cross-validation RMSE for best of the 25 models

```
RF_gridperf = h2o.getGrid(grid_id = "RF_grid",
                          sort_by = "RMSE",
                          decreasing = F)
```

```
## Hyper-Parameter Search Summary: ordered by increasing RMSE
##   max_depth ntrees         model_ids               rmse
## 1        33    760 RF_grid_model_23 1.4669141946148525
## 2         9    760 RF_grid_model_24 1.4684634805902947
## 3         8    760 RF_grid_model_22  1.472069734081821
## 4        33    583 RF_grid_model_18 1.4736238849120469
## 5         9    583 RF_grid_model_19 1.4754017707854352
## 6         8    583 RF_grid_model_17 1.4783540188668924
```

- Remember: These are not test set RMSEs
- It is not a given, that the best model in CV is also best on test set
- In particular when train and test set could not be created by random split as with financial data

How well is the best model doing on the test set?

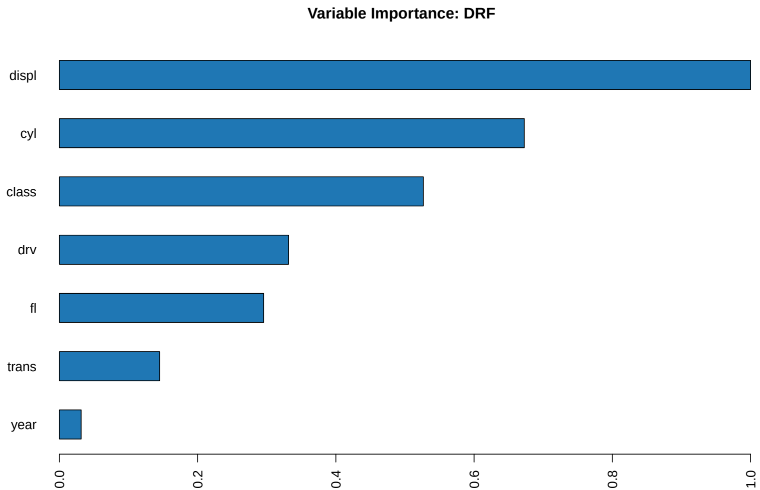
```r
# Evaluate best model from CV on test set
best_RF = h2o.getModel(RF_gridperf@summary_table$model_ids[1])
best_RF_perf = h2o.performance(model = best_RF,
                               newdata = h2o.test)
h2o.rmse(best_RF_perf)
```

```
## [1] 1.034678
```

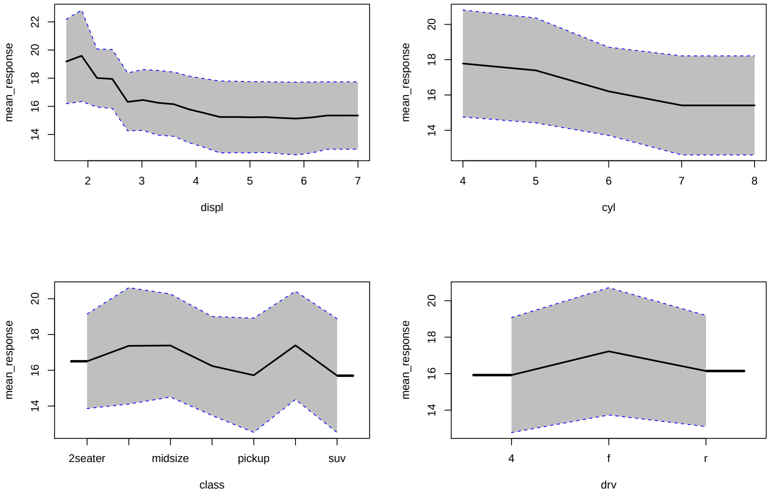
- Barely an improvement compared to RF with default parameters with RMSE of 1.09
- Depending on the data, tuning can help more or less
- Often only a second order effect compared to choice of features

Engine displacement and number of cylinders most important features

```
h2o.varimp_plot(best_RF)
```



Sensitivity analysis



Charts created with function `h2o.partialPlot`.

Further reading

- http://docs.h2o.ai/h2o/latest-stable/h2o-docs/index.html
- https://www.h2o.ai/wp-content/uploads/2018/01/RBooklet.pdf
- https://github.com/maximilianstroh/henley_dudes