Automatic and Interpretable Machine Learning with H2O & LIME

Code Breakfast at GoDataDriven

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

Download: https://github.com/woobe/lime_water/ or bit.ly/joe_lime_water

About Joe

- My name is Jo-fai
- Majority of my British friends cannot remember Jo-fai
- Joe is the solution
- Data Scientist at H2O.ai
- For a very long time, I was the only H2O person in UK ...
- Community Manager / Sales Engineer / Photographer / SWAG Distributor

Agenda

- **Introduction**
  - Why?
  - Interpretable Machine Learning
    - LIME Framework
  - Automatic Machine Learning
    - H2O AutoML

- **Worked Examples**

  - Regression
  - Classification

- **Other Stuff + Q & A**

Acknowledgement

- **Marco Tulio Ribeiro**: Original LIME Framework and Python package
- **Thomas Lin Pedersen**: LIME R package
- **Matt Dancho**: LIME + H2O AutoML example + LIME R package improvement
- **Kasia Kulma**: LIME + H2O AutoML example
- My H2O colleagues **Erin LeDell**, **Ray Peck**, **Navdeep Gill** and many others for AutoML

Why?

Why Should I Trust Your Model?

System that performs behaviour but you don’t know how it works

Interpretable Machine Learning

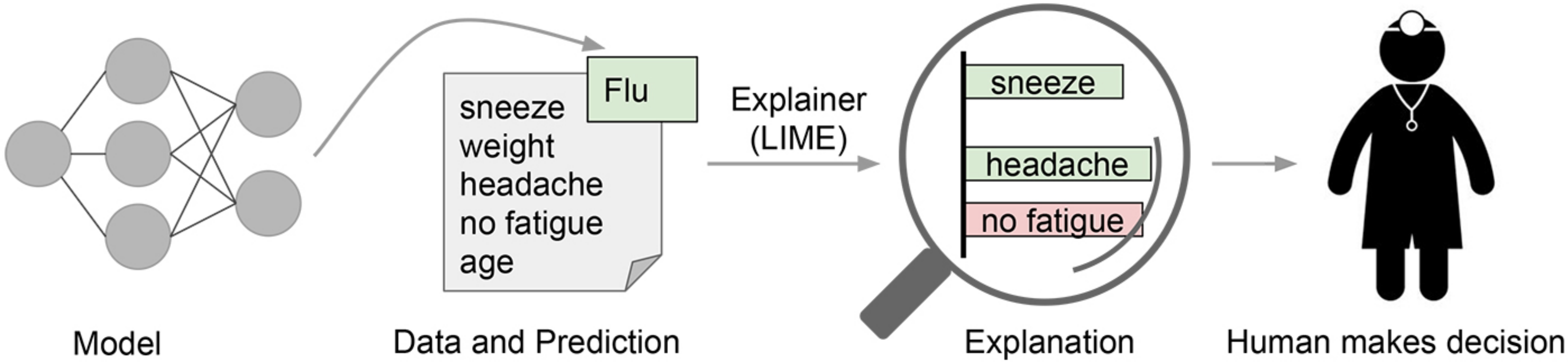


Figure 1. Explaining individual predictions to a human decision-maker. Source: Marco Tulio Ribeiro.

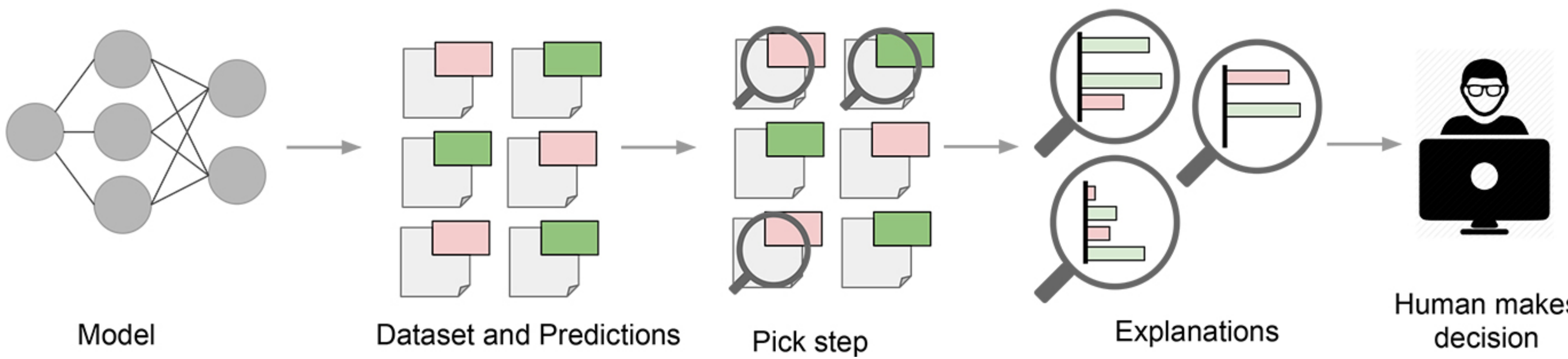


Figure 2. Explaining a model to a human decision-maker. Source: Marco Tulio Ribeiro.

The LIME Framework

Local Interpretable Model-agnostic Explanations

A framework for interpretability

Complexity of learned functions:
- Linear, monotonic
- Nonlinear, monotonic
- Nonlinear, non-monotonic

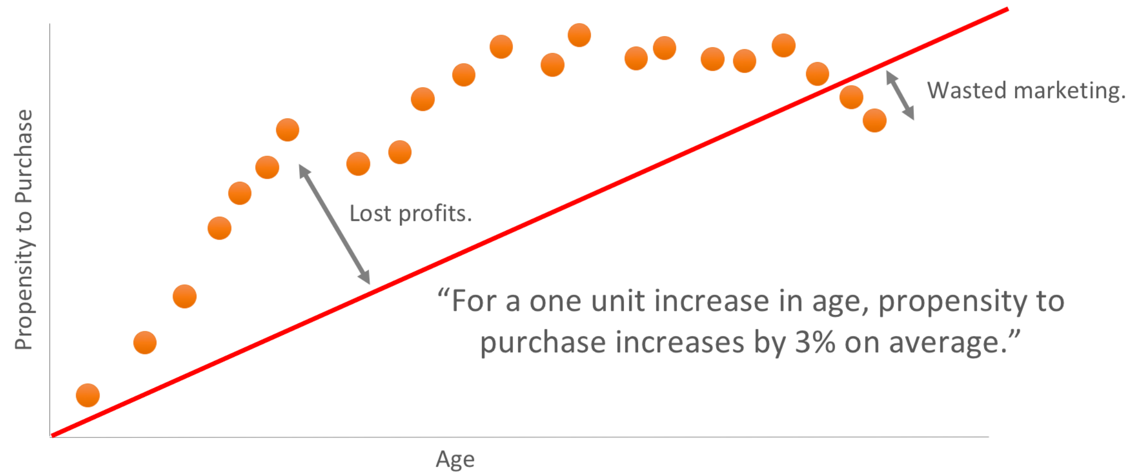
Scope of interpretability:
Global vs. local

Enhancing trust and understanding: the mechanisms and results of an interpretable model should be both transparent AND dependable.

Application domain:
Model-agnostic vs. model-specific

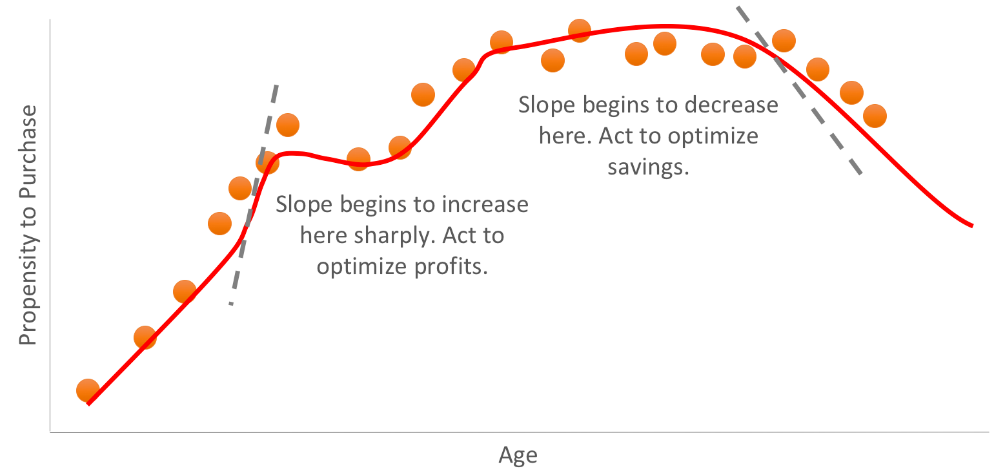
Linear Models

Exact explanations for ***approximate*** models.



Machine Learning

Approximate explanations for ***exact*** models.



How does LIME work?

Theory

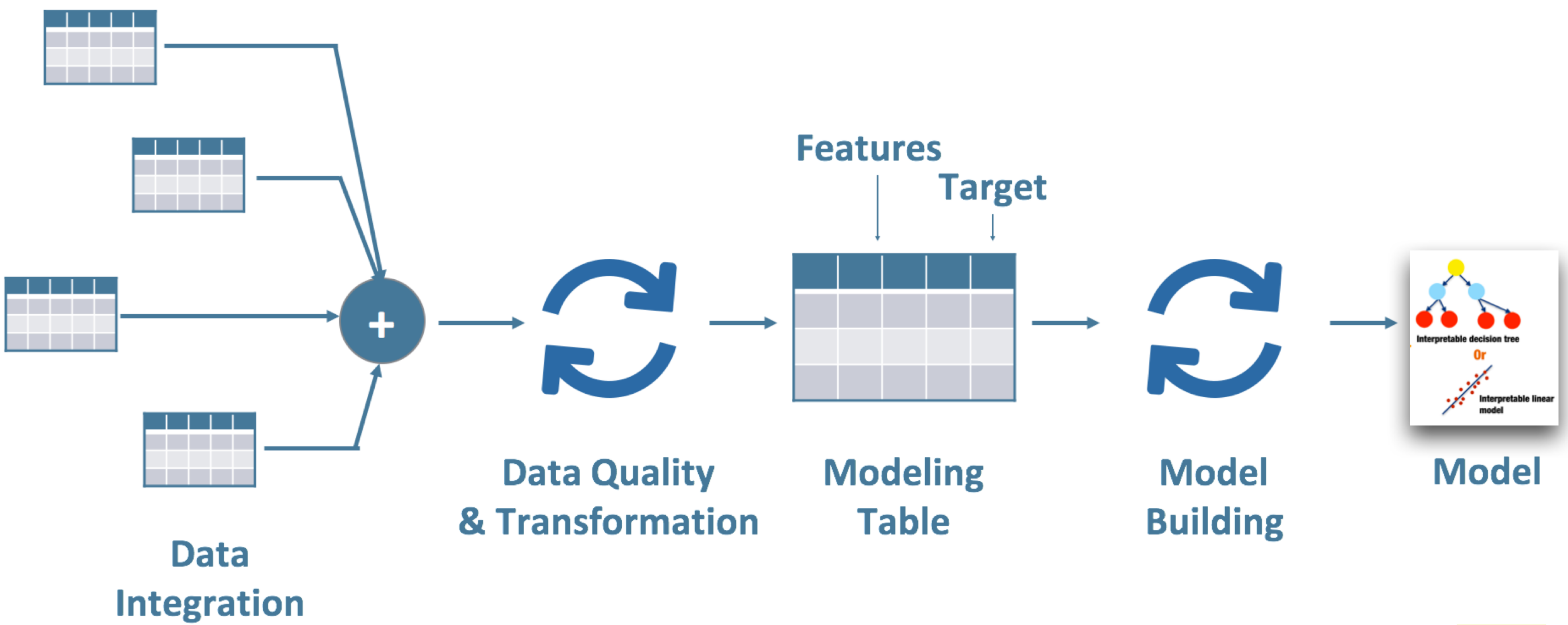
- LIME approximates model locally as logistic or linear model
- Repeats process many times
- Output features that are most important to local models

Outcome

- Approximate reasoning
- Complex models can be interpreted
    - Neural nets, Random Forest, Ensembles etc.

Automatic Machine Learning

Typical Enterprise Machine Learning Workflow



H2O AutoML

H2O's AutoML can be used for automating a large part of the machine learning workflow, which includes automatic training and tuning of many models within a user-specified time-limit. The user can also use a performance metric-based stopping criterion for the AutoML process rather than a specific time constraint. Stacked Ensembles will be automatically trained on the collection individual models to produce a highly predictive ensemble model which, in most cases, will be the top performing model in the AutoML Leaderboard.

R Interface

```
aml € h2o.automl(x € x, y € y,
                 training_frame € train,
                 max_runtime_secs € 3600)
```

Python Interface

```
aml € H2OAutoML(max_runtime_secs € 3600)
aml.train(x € x, y € y,
          training_frame € train)
```

Web Interface

H2O Flow

Lime Water

Lime Water in R

LIME

```r
# Install 'lime' from CRAN
install.packages('lime')
```

or

```r
# Install development version from GitHub
devtools::install_github('thomasp85/lime')
```

H2O

```r
# Install 'h2o' from CRAN
install.packages('h2o')
```

or

```r
# Install latest stable release from H2O's
# website www.h2o.ai/download/
# Latest Version = 3.18.0.1
# (as of 19-Feb-2018)
install.packages("h2o", type="source",
repos="http://h2o-release.s3.amazonaws.com
/h2o/rel-wolpert/1/R")
```

Regression Example

Regression Example: Boston Housing

```
Data Set Characteristics:

    - Number of Instances: 506

    - Number of Attributes: 13 numeric/categorical predictive

    - Median Value (attribute 14) is the target

    - Attribute Information (in order):
        - CRIM     per capita crime rate by town
        - ZN       proportion of residential land zoned for lots over 25,000 sq.ft.
        - INDUS    proportion of non-retail business acres per town
        - CHAS     Charles River dummy variable (€ 1 if tract bounds river; 0 otherwise)
        - NOX      nitric oxides concentration (parts per 10 million)
        - RM       average number of rooms per dwelling
        - AGE      proportion of owner-occupied units built prior to 1940
        - DIS      weighted distances to five Boston employment centres
        - RAD      index of accessibility to radial highways
        - TAX      full-value property-tax rate per  10,000
        - PTRATIO  pupil-teacher ratio by town
        - B        1000(Bk - 0.63)|2 where Bk is the proportion of blacks by town
        - LSTAT    ⌡ lower status of the population
        - MEDV     Median value of owner-occupied homes in  1000's

    - Creator: Harrison, D. and Rubinfeld, D.L.
    - Source: http:⁒archive.ics.uci.edu/ml/datasets/Housing
```

Regression Example: Boston Housing

```r
library(mlbench) # for dataset
data("BostonHousing")
dim(BostonHousing)
```

```
## [1] 506  14
```

```r
# First six samples
knitr::kable(head(BostonHousing), format = "html")
```

crim	zn	indus	chas	nox	rm	age	dis	rad	tax	ptratio	b	lstat	medv
0.00632	18	2.31	0	0.538	6.575	65.2	4.0900	1	296	15.3	396.90	4.98	24.0
0.02731	0	7.07	0	0.469	6.421	78.9	4.9671	2	242	17.8	396.90	9.14	21.6
0.02729	0	7.07	0	0.469	7.185	61.1	4.9671	2	242	17.8	392.83	4.03	34.7
0.03237	0	2.18	0	0.458	6.998	45.8	6.0622	3	222	18.7	394.63	2.94	33.4
0.06905	0	2.18	0	0.458	7.147	54.2	6.0622	3	222	18.7	396.90	5.33	36.2
0.02985	0	2.18	0	0.458	6.430	58.7	6.0622	3	222	18.7	394.12	5.21	28.7

Boston Housing (Simple Split)

```r
# Define features
features € setdiff(colnames(BostonHousing), "medv")
features
```

```
##  [1] "crim"    "zn"      "indus"   "chas"    "nox"     "rm"      "age"
##  [8] "dis"     "rad"     "tax"     "ptratio" "b"       "lstat"
```

```r
# Pick four random samples for test dataset
set.seed(1234)
row_test_samp € sample(1:nrow(BostonHousing), 4)
```

```r
# Train
x_train € BostonHousing[-row_test_samp, features]
y_train € BostonHousing[-row_test_samp, "medv"]
```

```r
# Test
x_test € BostonHousing[row_test_samp, features]
y_test € BostonHousing[row_test_samp, "medv"]
```

Build a Random Forest (RF)

```r
library(caret) # ML framework
library(doParallel) # parallelisation
```

```r
# Train a Random Forest using caret
cl € makePSOCKcluster(8)
registerDoParallel(cl)
set.seed(1234)
model_rf €
  caret::train(
    x € x_train,
    y € y_train,
    method € "rf",
    tuneLength € 3,
    trControl € trainControl(method € "cv")
    )
stopCluster(cl)
```

```r
# Print model summary
model_rf
```

```
## Random Forest
##
## 502 samples
##  13 predictor
##
## No pre-processing
## Resampling: Cross-Validated (10 fold)
## Summary of sample sizes: 453, 451, 453, 451, 452, 4
## Resampling results across tuning parameters:
##
##   mtry  RMSE      Rsquared   MAE
##    2    3.532344  0.8715278  2.361838
##    7    3.204915  0.8874722  2.185825
##   13    3.256840  0.8798610  2.230762
##
## RMSE was used to select the optimal model using the
## The final value used for the model was mtry € 7.
```

RF: Making Prediction

```
# Using the Random Forest model to make predictions on test set
yhat_test € predict(model_rf, x_test)
```

```
# Create a new data frame to compare target (medv) and predictions
d_test € data.frame(x_test,
                    medv € y_test,
                    predict € yhat_test,
                    row.names € NULL)
knitr::kable(d_test, format € "html")
```

crim	zn	indus	chas	nox	rm	age	dis	rad	tax	ptratio	b	lstat	medv	predict
0.01432	100	1.32	0	0.411	6.816	40.5	8.3248	5	256	15.1	392.90	3.95	31.6	31.63876
0.36920	0	9.90	0	0.544	6.567	87.3	3.6023	4	304	18.4	395.69	9.28	23.8	24.09371
0.04932	33	2.18	0	0.472	6.849	70.3	3.1827	7	222	18.4	396.90	7.53	28.2	29.78971
0.26938	0	9.90	0	0.544	6.266	82.8	3.2628	4	304	18.4	393.39	7.90	21.6	22.69258

RF: LIME Steps 1 and 2

```
# Step 1# Create an 'explainer' object using training data and model
explainer € lime::lime(x € x_train, model € model_rf)
```

```
# Step 2# Turn 'explainer' into 'explainations' for test set
explainations € lime::explain(x € x_test,
                              explainer € explainer,
                              n_permutations € 5000,
                              feature_select € "auto",
                              n_features € 5)
```

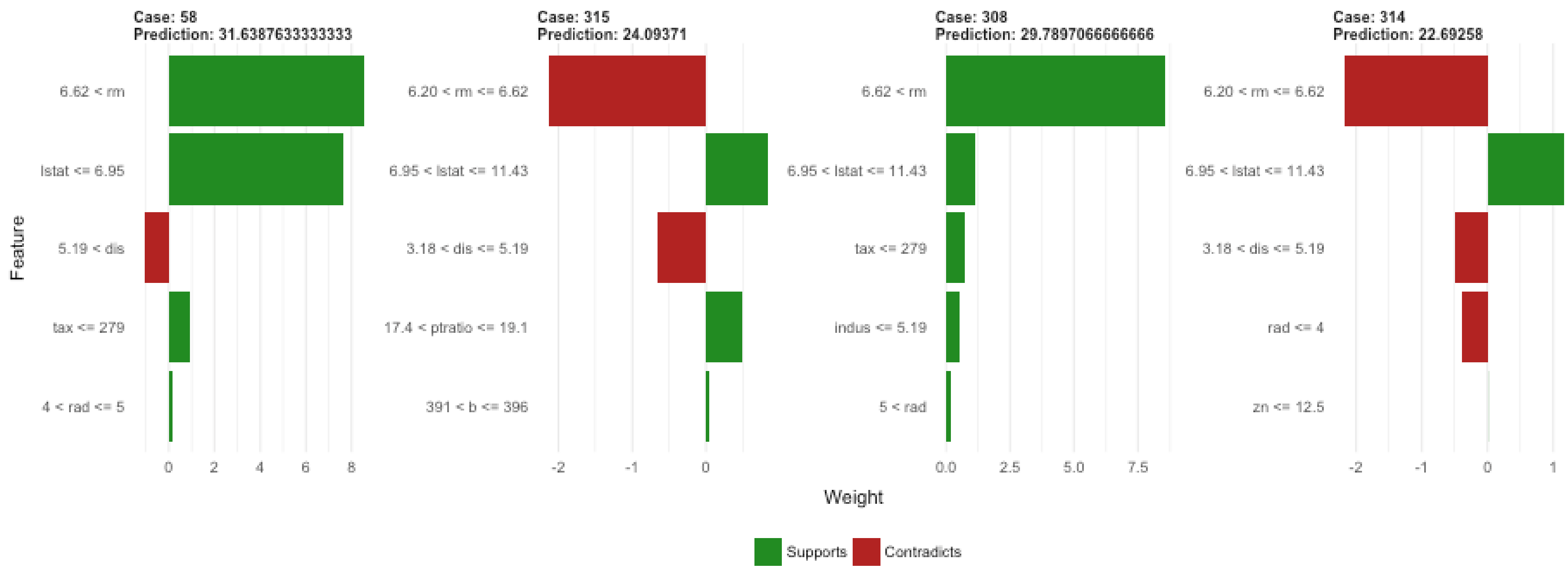
RF: LIME Explainations

```
head(explainations, 5) #LIME Pred: 36.59, Random Forest Pred: 31.64, R|2 € 0.65
```

```
—«   model_type case  model_r2 model_intercept model_prediction feature
—« 1 regression   58 0.6533429        20.43179         36.59478     rad
—« 2 regression   58 0.6533429        20.43179         36.59478      rm
—« 3 regression   58 0.6533429        20.43179         36.59478   lstat
—« 4 regression   58 0.6533429        20.43179         36.59478     dis
—« 5 regression   58 0.6533429        20.43179         36.59478     tax
—«   feature_value feature_weight  feature_desc
—« 1        5.0000      0.1918038  4 ₹ rad»— 5
—« 2        6.8160      8.5174963     6.62 ₹ rm
—« 3        3.9500      7.6363579 lstat»— 6.95
—« 4        8.3248     -1.0748818    5.19 ₹ dis
—« 5      256.0000      0.8922157    tax»— 279
—«
—« 1 0.01432, 100.00000, 1.32000, 1.00000, 0.41100, 6.81600, 40.50000, 8.32480, 5.00000, 256.00000, 15.10
—« 2 0.01432, 100.00000, 1.32000, 1.00000, 0.41100, 6.81600, 40.50000, 8.32480, 5.00000, 256.00000, 15.10
—« 3 0.01432, 100.00000, 1.32000, 1.00000, 0.41100, 6.81600, 40.50000, 8.32480, 5.00000, 256.00000, 15.10
—« 4 0.01432, 100.00000, 1.32000, 1.00000, 0.41100, 6.81600, 40.50000, 8.32480, 5.00000, 256.00000, 15.10
—« 5 0.01432, 100.00000, 1.32000, 1.00000, 0.41100, 6.81600, 40.50000, 8.32480, 5.00000, 256.00000, 15.10
—«   prediction
—« 1   31.63876
—« 2   31.63876
—« 3   31.63876
—« 4   31.63876
—« 5   31.63876
```

RF: LIME Visualisation

```
# Step 3# Visualise explainations
lime::plot_features(explainations, ncol = 4)
```



H2O AutoML

```r
# Start a local H2O cluster (JVM)
library(h2o)
h2o.init(nthreads € -1)
```

```
##  Connection successful!
## 
## R is connected to the H2O cluster: 
##     H2O cluster uptime:         2 days 19 hours 
##     H2O cluster timezone:       Europe/London 
##     H2O data parsing timezone:  UTC 
##     H2O cluster version:        3.18.0.1 
##     H2O cluster version age:    6 days  
##     H2O cluster name:           H2O_started_from_R_jofaichow_ydb410 
##     H2O cluster total nodes:    1 
##     H2O cluster total memory:   3.89 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 
##     H2O Internal Security:      FALSE 
##     H2O API Extensions:         XGBoost, Algos, AutoML, Core V3, Core V4 
##     R Version:                  R version 3.4.3 (2017-11-30)
```

Prepare H2O Data Frames

```
# Prepare Data
h_train € as.h2o(BostonHousing[-row_test_samp,])
h_test € as.h2o(BostonHousing[row_test_samp,])
```

```
head(h_test)
```

```
#>       crim  zn indus chas   nox    rm  age    dis rad tax ptratio      b
#> 1 0.01432 100  1.32    0 0.411 6.816 40.5 8.3248   5 256    15.1 392.90
#> 2 0.36920   0  9.90    0 0.544 6.567 87.3 3.6023   4 304    18.4 395.69
#> 3 0.04932  33  2.18    0 0.472 6.849 70.3 3.1827   7 222    18.4 396.90
#> 4 0.26938   0  9.90    0 0.544 6.266 82.8 3.2628   4 304    18.4 393.39
#>   lstat medv
#> 1  3.95 31.6
#> 2  9.28 23.8
#> 3  7.53 28.2
#> 4  7.90 21.6
```

Train Multiple H2O Models

```
# Train multiple H2O models with a simple API
# Stacked Ensembles will be created from those H2O models
# You tell H2O 1) how much time you have and/or 2) how many models do you want
model_automl € h2o.automl(x € features,
                          y € "medv",
                          training_frame € h_train,
                          nfolds € 5,
                          max_runtime_secs € 120, # time
                          max_models € 20,        # max models
                          stopping_metric € "RMSE",
                          seed € 1234)
```

H2O: AutoML Model Leaderboard

```r
# Print out leaderboard
model_automl@leaderboard
```

```
##                                              model_id
## 1 StackedEnsemble_BestOfFamily_0_AutoML_20180219_064235
## 2             GBM_grid_0_AutoML_20180219_064235_model_0
## 3    StackedEnsemble_AllModels_0_AutoML_20180219_064235
## 4             GBM_grid_0_AutoML_20180219_064235_model_1
## 5             GBM_grid_0_AutoML_20180219_064235_model_3
## 6                          DRF_0_AutoML_20180219_064235
##   mean_residual_deviance     rmse      mae    rmsle
## 1               10.81736 3.288976 2.149675 0.140762
## 2               10.86044 3.295518 2.224282 0.145063
## 3               10.89431 3.300653 2.161742 0.140959
## 4               11.88445 3.447383 2.285338 0.145858
## 5               12.12041 3.481438 2.324986 0.148829
## 6               12.22679 3.496683 2.339066 0.148301
##
## [22 rows x 5 columns]
```

H2O: Model Leader

```
# Best Model (either an individual model or a stacked ensemble)
model_automl@leader
```

```
—̃« Model Details:
—̃« €€€€€€€€€€€€€€
—̃«
—̃« H2ORegressionModel: stackedensemble
—̃« Model ID#  StackedEnsemble_BestOfFamily_0_AutoML_20180219_064235
—̃« NULL
—̃«
—̃«
—̃« H2ORegressionMetrics: stackedensemble
—̃« *̃* Reported on training data. *̃*
—̃«
—̃« MSE#  0.7958593
—̃« RMSE#  0.8921095
—̃« MAE#  0.6584744
—̃« RMSLE#  0.0446051
—̃« Mean Residual Deviance :  0.7958593
—̃«
—̃«
—̃« H2ORegressionMetrics: stackedensemble
—̃« *̃* Reported on validation data. *̃*
—̃«
—̃« MSE#  6.71778
—̃« RMSE#  2.591868
```

H2O: Making Prediction

```r
# Using the best model to make predictions on test set
yhat_test <- h2o.predict(model_automl@leader, h_test)
```

```r
# Create a new data frame to compare target (medv) and predictions
d_test <- data.frame(x_test,
                     medv <- y_test,
                     predict <- as.data.frame(yhat_test),
                     row.names <- NULL)
knitr::kable(d_test, format <- "html")
```

crim	zn	indus	chas	nox	rm	age	dis	rad	tax	ptratio	b	lstat	medv	predict
0.01432	100	1.32	0	0.411	6.816	40.5	8.3248	5	256	15.1	392.90	3.95	31.6	31.01702
0.36920	0	9.90	0	0.544	6.567	87.3	3.6023	4	304	18.4	395.69	9.28	23.8	22.86415
0.04932	33	2.18	0	0.472	6.849	70.3	3.1827	7	222	18.4	396.90	7.53	28.2	30.07423
0.26938	0	9.90	0	0.544	6.266	82.8	3.2628	4	304	18.4	393.39	7.90	21.6	21.98239

H2O: LIME Steps 1 and 2

```
# Step 1# Create an 'explainer' object using training data and model
explainer € lime::lime(x € as.data.frame(h_train[, features]),
                       model € model_automl@leader)
```

```
# Step 2# Turn 'explainer' into 'explainations' for test set
explainations € lime::explain(x € as.data.frame(h_test[, features]),
                              explainer € explainer,
                              n_permutations € 5000,
                              feature_select € "auto",
                              n_features € 5) # look at top 5 features only
```

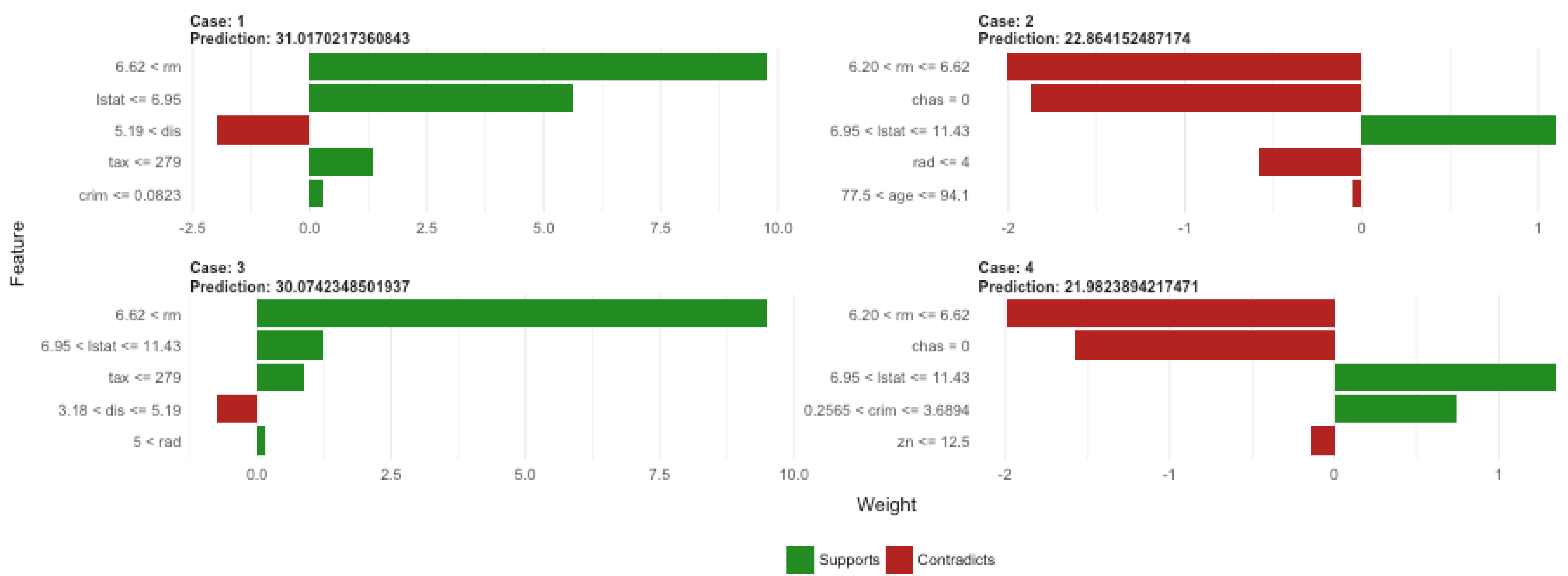
H2O: LIME Explainations

```r
head(explainations, 5)
```

```
—̃«   model_type case  model_r2 model_intercept model_prediction feature
—̃« 1 regression    1 0.6315333        20.38476         35.41604    crim
—̃« 2 regression    1 0.6315333        20.38476         35.41604      rm
—̃« 3 regression    1 0.6315333        20.38476         35.41604   lstat
—̃« 4 regression    1 0.6315333        20.38476         35.41604     dis
—̃« 5 regression    1 0.6315333        20.38476         35.41604     tax
—̃«   feature_value feature_weight   feature_desc
—̃« 1       0.01432      0.2610387 crim»̃— 0.0823
—̃« 2       6.81600      9.7661035      6.62 ₹ rm
—̃« 3       3.95000      5.6211713  lstat»̃— 6.95
—̃« 4       8.32480     -1.9773211     5.19 ₹ dis
—̃« 5     256.00000      1.3602842     tax»̃— 279
—̃«
—̃« 1 0.01432, 100.00000, 1.32000, 1.00000, 0.41100, 6.81600, 40.50000, 8.32480, 5.00000, 256.00000, 15.10
—̃« 2 0.01432, 100.00000, 1.32000, 1.00000, 0.41100, 6.81600, 40.50000, 8.32480, 5.00000, 256.00000, 15.10
—̃« 3 0.01432, 100.00000, 1.32000, 1.00000, 0.41100, 6.81600, 40.50000, 8.32480, 5.00000, 256.00000, 15.10
—̃« 4 0.01432, 100.00000, 1.32000, 1.00000, 0.41100, 6.81600, 40.50000, 8.32480, 5.00000, 256.00000, 15.10
—̃« 5 0.01432, 100.00000, 1.32000, 1.00000, 0.41100, 6.81600, 40.50000, 8.32480, 5.00000, 256.00000, 15.10
—̃«   prediction
—̃« 1   31.01702
—̃« 2   31.01702
—̃« 3   31.01702
—̃« 4   31.01702
—̃« 5   31.01702
```

H2O: LIME Visualisation

```r
# Step 3# Visualise explainations
lime::plot_features(explainations, ncol = 2)
```



Classification Example

Classification Example: Glass

```r
library(mlbench) # for dataset
data("Glass")
```

```r
# Rename columns
colnames(Glass) <- c("Refractive_Index", "Sodium", "Magnesium", "Aluminium",
                     "Silicon", "Potassium", "Calcium", "Barium", "Iron", "Type")
dim(Glass)
```

```
## [1] 214  10
```

```r
str(Glass)
```

```
## 'data.frame':	214 obs. of  10 variables:
##  $ Refractive_Index: num  1.52 1.52 1.52 1.52 1.52 ...
##  $ Sodium          : num  13.6 13.9 13.5 13.2 13.3 ...
##  $ Magnesium       : num  4.49 3.6 3.55 3.69 3.62 3.61 3.6 3.61 3.58 3.6 ...
##  $ Aluminium       : num  1.1 1.36 1.54 1.29 1.24 1.62 1.14 1.05 1.37 1.36 ...
##  $ Silicon         : num  71.8 72.7 73 72.6 73.1 ...
##  $ Potassium       : num  0.06 0.48 0.39 0.57 0.55 0.64 0.58 0.57 0.56 0.57 ...
##  $ Calcium         : num  8.75 7.83 7.78 8.22 8.07 8.07 8.17 8.24 8.3 8.4 ...
##  $ Barium          : num  0 0 0 0 0 0 0 0 0 0 ...
##  $ Iron            : num  0 0 0 0 0 0.26 0 0 0 0.11 ...
##  $ Type            : Factor w/ 6 levels "1","2","3","5",..: 1 1 1 1 1 1 1 1 1 1 ...
```

Glass (Simple Split)

```r
# Define Features
features € setdiff(colnames(Glass), "Type")
features
```

```
#> [1] "Refractive_Index" "Sodium"           "Magnesium"
#> [4] "Aluminium"        "Silicon"          "Potassium"
#> [7] "Calcium"          "Barium"           "Iron"
```

```r
# Pick four random samples for test dataset
set.seed(1234)
row_test_samp € sample(1:nrow(Glass), 4)
```

H2O AutoML

```r
# Start a local H2O cluster (JVM)
library(h2o)
h2o.init(nthreads € -1)
```

```
##  Connection successful!
## 
## R is connected to the H2O cluster: 
##     H2O cluster uptime:         2 days 19 hours 
##     H2O cluster timezone:       Europe/London 
##     H2O data parsing timezone:  UTC 
##     H2O cluster version:        3.18.0.1 
##     H2O cluster version age:    6 days  
##     H2O cluster name:           H2O_started_from_R_jofaichow_ydb410 
##     H2O cluster total nodes:    1 
##     H2O cluster total memory:   3.83 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 
##     H2O Internal Security:      FALSE 
##     H2O API Extensions:         XGBoost, Algos, AutoML, Core V3, Core V4 
##     R Version:                  R version 3.4.3 (2017-11-30)
```

Prepare H2O Data Frames

```
# Prepare Data
h_train € as.h2o(Glass[-row_test_samp,])
h_test € as.h2o(Glass[row_test_samp,])
```

```
head(h_test)
```

```
##   Refractive_Index Sodium Magnesium Aluminium Silicon Potassium Calcium
## 1          1.51720  13.38      3.50      1.15   72.85      0.50    8.43
## 2          1.51813  13.43      3.98      1.18   72.49      0.58    8.15
## 3          1.52020  13.98      1.35      1.63   71.76      0.39   10.56
## 4          1.52614  13.70      0.00      1.36   71.24      0.19   13.44
##   Barium Iron Type
## 1      0 0.00    1
## 2      0 0.00    2
## 3      0 0.18    2
## 4      0 0.10    2
```

Train Multiple H2O Models

```
# Train multiple H2O models with a simple API
# Stacked Ensembles will be created from those H2O models
# You tell H2O 1) how much time you have and/or 2) how many models do you want
model_automl € h2o.automl(x € features,
                          y € "Type",
                          training_frame € h_train,
                          nfolds € 5,
                          max_runtime_secs € 120, # time
                          max_models € 20,        # max models
                          stopping_metric € "mean_per_class_error",
                          seed € 1234)
```

H2O: AutoML Model Leaderboard

```r
# Print out leaderboard
model_automl@leaderboard
```

```
#>                                          model_id mean_per_class_error
#> 1  GBM_grid_0_AutoML_20180219_064309_model_3             0.304868
#> 2  GBM_grid_0_AutoML_20180219_064309_model_2             0.304868
#> 3  GBM_grid_0_AutoML_20180219_064309_model_1             0.304868
#> 4  GBM_grid_0_AutoML_20180219_064309_model_0             0.343727
#> 5 GBM_grid_0_AutoML_20180219_064309_model_12             0.347430
#> 6              XRT_0_AutoML_20180219_064309             0.351009
#> 
#> [22 rows x 2 columns]
```

H2O: Model Leader

```
# Best Model (either an individual model or a stacked ensemble)
model_automl@leader
```

```
—« Model Details:
—« €€€€€€€€€€€€€€
—«
—« H2OMultinomialModel: gbm
—« Model ID#  GBM_grid_0_AutoML_20180219_064309_model_3
—« Model Summary:
—«   number_of_trees number_of_internal_trees model_size_in_bytes min_depth
—« 1              44                      264               50484         1
—«   max_depth mean_depth min_leaves max_leaves mean_leaves
—« 1        10    6.29545          2         13    10.26515
—«
—«
—« H2OMultinomialMetrics: gbm
—« ** Reported on training data. **
—«
—« Training Set Metrics:
—« €€€€€€€€€€€€€€€€€€€€€
—«
—« Extract training frame with ″h2o.getFrame("automl_training_file136b76df013a3_sid_9a73_12")″
—« MSE# (Extract with ″h2o.mse″) 0.01203013
—« RMSE# (Extract with ″h2o.rmse″) 0.1096819
—« Logloss: (Extract with ″h2o.logloss″) 0.08297804
—« Mean Per-Class Error: 0
```

H2O: Making Prediction

```r
# Using the best model to make predictions on test set
yhat_test <- h2o.predict(model_automl@leader, h_test)
head(yhat_test)
```

```
#>   predict         p1        p2          p3          p5          p6
#> 1       1 0.48905008 0.1666208 0.324915505 0.005839232 0.005865232
#> 2       2 0.04100466 0.9440974 0.008742662 0.001362321 0.001367279
#> 3       5 0.02860819 0.2055182 0.014776797 0.705273483 0.036432625
#> 4       2 0.02329044 0.8573997 0.011031539 0.083977413 0.003666561
#>            p7
#> 1 0.007709192
#> 2 0.003425725
#> 3 0.009390700
#> 4 0.020634392
```

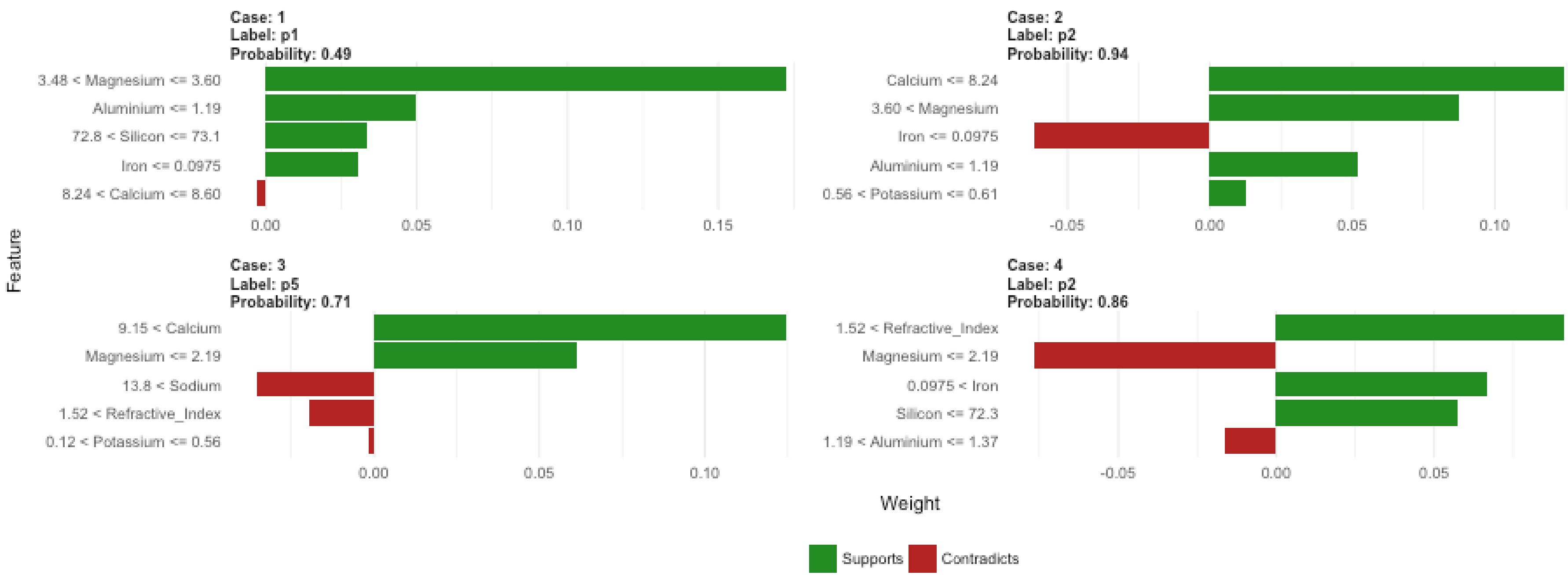
H2O: LIME Steps 1 and 2

```
# Step 1# Create an 'explainer' object using training data and model
explainer € lime::lime(x € as.data.frame(h_train[, features]),
                       model € model_automl@leader)
```

```
# Step 2# Turn 'explainer' into 'explainations' for test set
explainations € lime::explain(x € as.data.frame(h_test[, features]),
                              explainer € explainer,
                              n_permutations € 5000,
                              feature_select € "auto",
                              n_labels € 1, # Explain top prediction only
                              n_features € 5)
```

H2O: LIME Visualisation

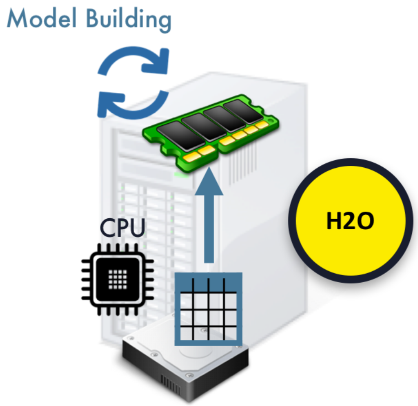
```
# Step 3# Visualise explainations
lime::plot_features(explainations, ncol = 2)
```



Other Stuff

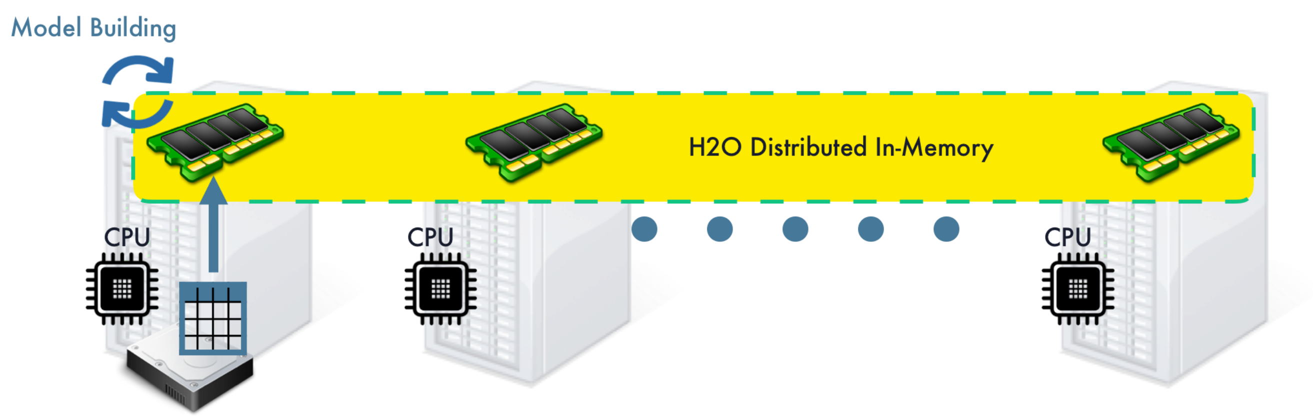
H2O in Action

H_2O Core

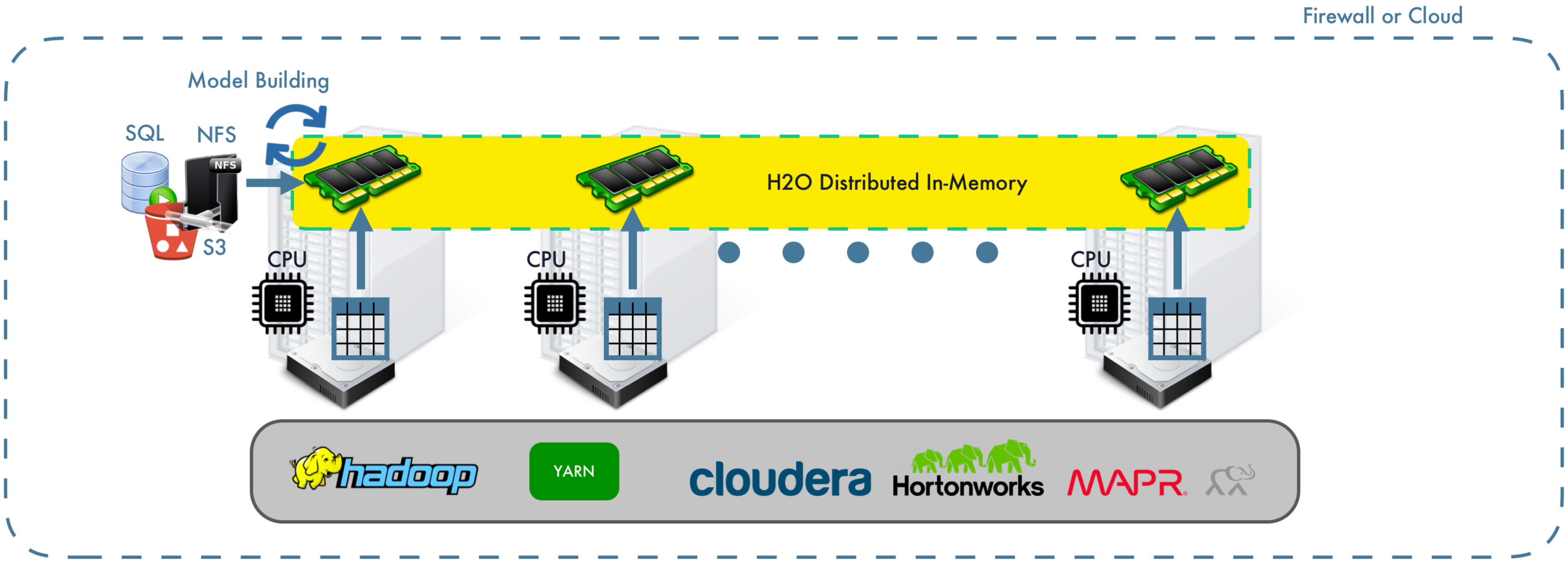


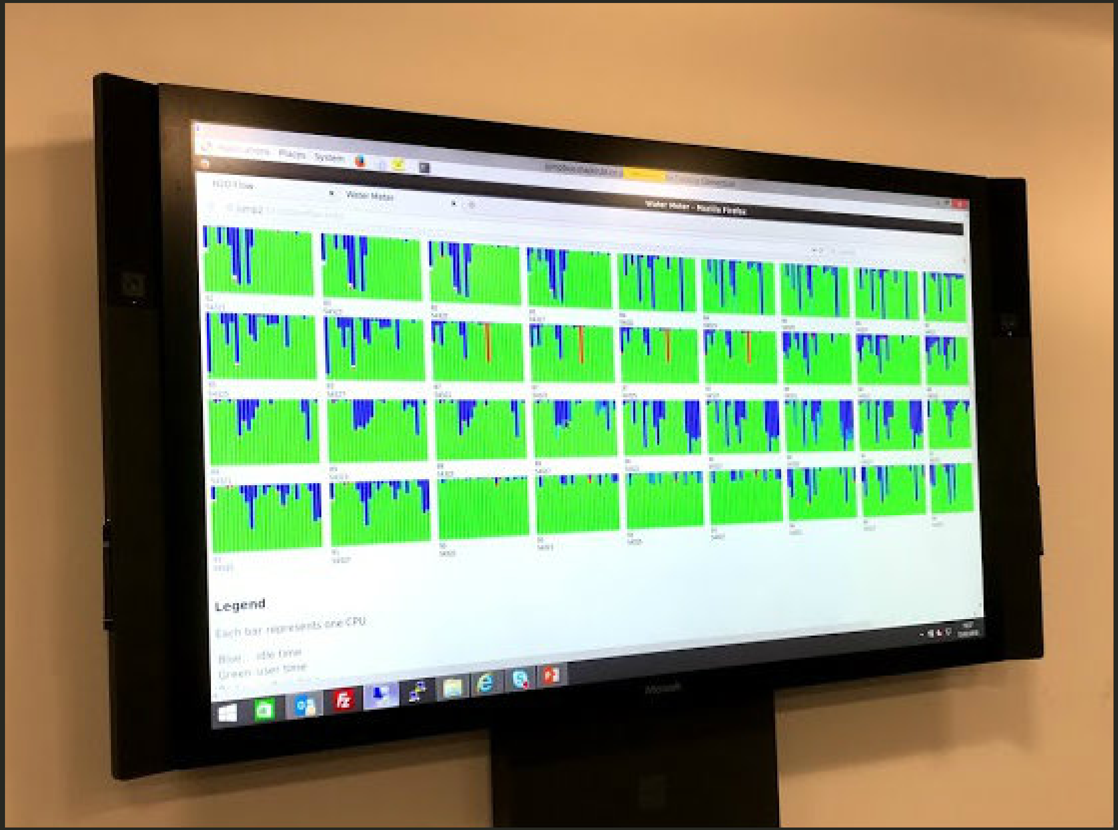
H_2O Core

H_2O Core



H_2O Core


Firewall or Cloud
Model Building
SQL
NFS
S3
H2O Distributed In-Memory
CPU
CPU
CPU
hadoop
YARN
cloudera
Hortonworks
MAPR


Legend

Tools & Examples

Python Tools

- lime (Original Python Package by Marco Ribeiro) Link

Python Examples

- Marco's Examples See GitHub README
- LIME + H2O Example Link
- LIME in Python by Erin Brown Link

R Examples

- Text Example by Thomas Link
- HR Analytics Example by Matt Link
- Cancer Example by Kasia Link

Related Topics

- SHAP (SHapley Additive exPlanations)
  - A Unified Approach to Interpreting Model Predictions
    - Paper
    - GitHub
  - http://www.f1-predictor.com/model-interpretability-with-shap/

Amsterdam Meetups

Tue 20 Feb - Sparkling Water in Production Webinar

- Link: https://www.meetup.com/Amsterdam-Artificial-Intelligence-Deep-Learning/events/247630667/

Thu 22 Feb - Meetup at ING

- Anomaly Detection in Finance using Isolation Forest by **Andreea Bejinaru**
- FoR the HoRde: WoRld of WaR-and SpaRkCRaft by **Vincent Warmerdam**
- Link: https://www.meetup.com/Amsterdam-Artificial-Intelligence-Deep-Learning/events/247356503/

Thanks!

joe@h2o.ai / @matlabulous

https://github.com/woobe/lime_water/

Slides created via the R package **xaringan**.