The Employee Attrition Dilemma:
h2o, lime, and Machine Learning

Michael Vatt

24 Jun 20

Contents

Overview

This white paper was motivated by the fact that employee turnover (attrition) is a major cost to an organization. Predicting turnover is paramount to any Human Resources division and OHR is no different.

Historically, logistic regression or survival curves were mainstream to model employee attrition. Advancements in Machine Learning (ML) have enabled augmented predictive performance and improved explanatory analysis of the critical features linked to employee attrition. ML has technically been around since the beginning of the modern ideal of a computer with Alan Turing but the statistical methods giving birth to ML have existed, adapted, and improved for centuries.

This study on employee attrition will use two automated ML algorithm packages that are free to use to develop a predictive model that is in the same ballpark as high-end commercial products in terms of accuracy. We will use *h2o's h2o.automl()* function and then the *lime's lime()* function to enable a breakdown of complex, black-box ML models into variable importance plots.

This is holistically a **fictional dataset** created by IBM Watson - *all individuals, events, identifiable information (if any), and observations are entirely fictional.*

The outline of this study is as follows: (1) Necessary packages are installed, (2) Dataset is loaded and setup configured, (3) Data Wrangling, familiarization and pre-processing of the dataset are performed, (4) Initialize and establish h2o protocols, (5) Modeling development and training, (6) Performance Analysis, (7) Run lime against the datasets, (8) Plots, and (9) Conclusions.

Performance analysis cannot be underestimated as it is critical to understand the ML algorithm and to identify whether it has concluded in informative and meaningful predictive probabilities.

- Let's Intall all Necessary Packages:

```r
# Note: This Process Could Take a Couple of Minutes for Loading Required Packages

install.packages("pacman", repos = "http://cran.us.r-project.org")
```

```
## package 'pacman' successfully unpacked and MD5 sums checked
##
## The downloaded binary packages are in
##  C:\Users\mjvat\AppData\Local\Temp\RtmpUhLZyy\downloaded_packages
```

```r
pacman::p_load("anytime", "bit", "car", "caret", "caTools", "data.table", "doParallel",
               "dplyr", "e1071", "ggplot2", "ggpubr", "glmnet", "gridExtra", "h2o", "jsonlite",
               "knitr", "lava", "lime", "lubridate", "methods", "pdp", "RColorBrewer", "RCurl",
               "readr", "readxl", "rjson", "scales", "statmod", "scales", "stats", "stringi",
               "stringr", "survival", "tibble", "tidyquant", "tidyr", "tidyverse", "timeDate",
               "tinytex", "tools", "utils", "versions", "vip")
```

Introduction

This ML algorithm is dependent upon employee attrition data from a fictional dataset created by IBM Watson. This dataset was motivated by exploratory data analysis to see how well *h2o* and *lime* would perform in ML compared to historical methods of predicting employee turnover.

Advances in ML have not only permitted for advanced predictions in employee attrition - both in methodology and accuracy - but also understanding the key variables - aka features - that influence turnover.

The *h2o* package - using the *h2oautoml()* function - uses any dataset and automatically tests a number of advanced algorithms, such as random forests, ensemble methods, deep learning, as well as traditional methods, such as logistic regression.

The *lime* package - using the *lime()* function - helps expose the inner workings of black-boxes. One of the largest complaints of ML is that users see *inputs* and *outputs* but do not see - let alone understand - how the algorithm is performing its function(s). This is largely due to their complexity and the basis for their appropriate nomenclature as *black-boxes*. Think of black holes in outerspace - we know they exist, we know certain functions and behaviors, but we can't really inspect what is going on inside to see what it is accomplishing - same sort of idea.

This dataset includes the following variables:

variables1	variables2
Age	MonthlyIncome
Attrition	MonthlyRate
BusinessTravel	NumCompaniesWorked
DailyRate	Over18
Department	OverTime
DistranceFromHome	PercentSalaryHike
Education	PerformanceRating
EducationField	RelationshipSatisfaction
EmployeeCount	StandardHours
EmployeeNumber	StockOptionLevel
EmployeeSatisfaction	TotalWorkingYears
Gender	TriningTimesLastYear
HourlyRate	WorkLifeBalance
JobInvolvement	YearsAtCompany
JobLevel	YearsInCurrentRole
JobRole	YearsSinceLastPromotion
JobSatisfaction	YearsWithCurrManager
MaritalStatus	

Objective

With the help of these tools, our objective is to uncover critical variables in the employee attrition dilemma and endeavor to discover improved predictive accuracy and intelligibility of the problem compared to traditional and burdensome methods.

This study will show how the combination of *h2o* and *lime* packages can be used with improved success in the employee attrition dilemma.

Dataset

- WARNING: Some Models in this Project Require a Lot of Computing Power. Splitting the Cores Will Aid in Parallel Processing of CPUs.
- NOTE: *h2o* uses advanced algorithms within SVM, RF, Deep Learning and others - this may take a few minutes.

```
split <- detectCores(TRUE)
split # To Make Sure it Worked
```

```
## [1] 24
```

```
cl <- makePSOCKcluster(split)
registerDoParallel(cl)
```

Methods and Analysis

Exploratory Data Analysis

Now that the data is loaded, let's gain some familiarization with the Dataset and view the first 10 rows:

```r
dim(hr_data_raw) # Dimensions of the Dataframe For Familiarity - This is Correct
```

```
## [1] 1470   35
```

```r
str(hr_data_raw) # Notice that Some Columns Are Not The Same Class
```

```
## tibble [1,470 x 35] (S3: tbl_df/tbl/data.frame)
##  $ Age                     : num [1:1470] 41 49 37 33 27 32 59 30 38 36 ...
##  $ Attrition               : chr [1:1470] "Yes" "No" "Yes" "No" ...
##  $ BusinessTravel          : chr [1:1470] "Travel_Rarely" "Travel_Frequently" "Travel_Rarely" "Travel
##  $ DailyRate               : num [1:1470] 1102 279 1373 1392 591 ...
##  $ Department              : chr [1:1470] "Sales" "Research & Development" "Research & Development"
##  $ DistanceFromHome        : num [1:1470] 1 8 2 3 2 2 3 24 23 27 ...
##  $ Education               : num [1:1470] 2 1 2 4 1 2 3 1 3 3 ...
##  $ EducationField          : chr [1:1470] "Life Sciences" "Life Sciences" "Other" "Life Sciences" ..
##  $ EmployeeCount           : num [1:1470] 1 1 1 1 1 1 1 1 1 1 ...
##  $ EmployeeNumber          : num [1:1470] 1 2 4 5 7 8 10 11 12 13 ...
##  $ EnvironmentSatisfaction : num [1:1470] 2 3 4 4 1 4 3 4 4 3 ...
##  $ Gender                  : chr [1:1470] "Female" "Male" "Male" "Female" ...
##  $ HourlyRate              : num [1:1470] 94 61 92 56 40 79 81 67 44 94 ...
##  $ JobInvolvement          : num [1:1470] 3 2 2 3 3 3 4 3 2 3 ...
##  $ JobLevel                : num [1:1470] 2 2 1 1 1 1 1 1 3 2 ...
##  $ JobRole                 : chr [1:1470] "Sales Executive" "Research Scientist" "Laboratory Technic
##  $ JobSatisfaction         : num [1:1470] 4 2 3 3 2 4 1 3 3 3 ...
##  $ MaritalStatus           : chr [1:1470] "Single" "Married" "Single" "Married" ...
##  $ MonthlyIncome           : num [1:1470] 5993 5130 2090 2909 3468 ...
##  $ MonthlyRate             : num [1:1470] 19479 24907 2396 23159 16632 ...
##  $ NumCompaniesWorked      : num [1:1470] 8 1 6 1 9 0 4 1 0 6 ...
##  $ Over18                  : chr [1:1470] "Y" "Y" "Y" "Y" ...
##  $ OverTime                : chr [1:1470] "Yes" "No" "Yes" "Yes" ...
##  $ PercentSalaryHike       : num [1:1470] 11 23 15 11 12 13 20 22 21 13 ...
##  $ PerformanceRating       : num [1:1470] 3 4 3 3 3 3 4 4 4 3 ...
##  $ RelationshipSatisfaction: num [1:1470] 1 4 2 3 4 3 1 2 2 2 ...
##  $ StandardHours           : num [1:1470] 80 80 80 80 80 80 80 80 80 80 ...
##  $ StockOptionLevel        : num [1:1470] 0 1 0 0 1 0 3 1 0 2 ...
##  $ TotalWorkingYears       : num [1:1470] 8 10 7 8 6 8 12 1 10 17 ...
##  $ TrainingTimesLastYear   : num [1:1470] 0 3 3 3 3 2 3 2 2 3 ...
##  $ WorkLifeBalance         : num [1:1470] 1 3 3 3 3 2 2 3 3 2 ...
##  $ YearsAtCompany          : num [1:1470] 6 10 0 8 2 7 1 1 9 7 ...
##  $ YearsInCurrentRole      : num [1:1470] 4 7 0 7 2 7 0 0 7 7 ...
##  $ YearsSinceLastPromotion : num [1:1470] 0 1 0 3 2 3 0 0 1 7 ...
##  $ YearsWithCurrManager    : num [1:1470] 5 7 0 0 2 6 0 0 8 7 ...
```

```r
class(hr_data_raw) # Let's Ensure it is Correct Format
```

```
## [1] "tbl_df"     "tbl"        "data.frame"
```

```r
hr_data_raw[1:10,] # Does the Data Look Right? - Yes
```

```
## # A tibble: 10 x 35
##      Age Attrition BusinessTravel DailyRate Department DistanceFromHome
##    <dbl> <chr>     <chr>              <dbl> <chr>                 <dbl>
##  1    41 Yes       Travel_Rarely       1102 Sales                     1
##  2    49 No        Travel_Freque~       279 Research ~                8
##  3    37 Yes       Travel_Rarely       1373 Research ~                2
##  4    33 No        Travel_Freque~      1392 Research ~                3
##  5    27 No        Travel_Rarely        591 Research ~                2
##  6    32 No        Travel_Freque~      1005 Research ~                2
##  7    59 No        Travel_Rarely       1324 Research ~                3
##  8    30 No        Travel_Rarely       1358 Research ~               24
##  9    38 No        Travel_Freque~       216 Research ~               23
## 10    36 No        Travel_Rarely       1299 Research ~               27
## # ... with 29 more variables: Education <dbl>, EducationField <chr>,
## #   EmployeeCount <dbl>, EmployeeNumber <dbl>, EnvironmentSatisfaction <dbl>,
## #   Gender <chr>, HourlyRate <dbl>, JobInvolvement <dbl>, JobLevel <dbl>,
## #   JobRole <chr>, JobSatisfaction <dbl>, MaritalStatus <chr>,
## #   MonthlyIncome <dbl>, MonthlyRate <dbl>, NumCompaniesWorked <dbl>,
## #   Over18 <chr>, OverTime <chr>, PercentSalaryHike <dbl>,
## #   PerformanceRating <dbl>, RelationshipSatisfaction <dbl>,
## #   StandardHours <dbl>, StockOptionLevel <dbl>, TotalWorkingYears <dbl>,
## #   TrainingTimesLastYear <dbl>, WorkLifeBalance <dbl>, YearsAtCompany <dbl>,
## #   YearsInCurrentRole <dbl>, YearsSinceLastPromotion <dbl>,
## #   YearsWithCurrManager <dbl>
```

We now need to perform a little bit of pre-processing to change character data types to factors. This is needed for *h2o* to function properly.

```r
# The Attrition column is our target - we'll use all other Columns as features.

# everything() selects all variables
hr_data <- hr_data_raw %>% mutate_if(is.character, as.factor) %>%
  select(Attrition, everything())

# Let's ensure it hasn't lost data

glimpse(hr_data) # 1470 rows and 35 cols (features)
```

```
## Rows: 1,470
## Columns: 35
## $ Attrition                <fct> Yes, No, Yes, No, No, No, No, No, No, No, ...
## $ Age                      <dbl> 41, 49, 37, 33, 27, 32, 59, 30, 38, 36, 35...
## $ BusinessTravel           <fct> Travel_Rarely, Travel_Frequently, Travel_R...
## $ DailyRate                <dbl> 1102, 279, 1373, 1392, 591, 1005, 1324, 13...
```

```
## $ Department               <fct> Sales, Research & Development, Research & ...
## $ DistanceFromHome         <dbl> 1, 8, 2, 3, 2, 2, 3, 24, 23, 27, 16, 15, 2...
## $ Education                <dbl> 2, 1, 2, 4, 1, 2, 3, 1, 3, 3, 3, 2, 1, 2, ...
## $ EducationField           <fct> Life Sciences, Life Sciences, Other, Life ...
## $ EmployeeCount            <dbl> 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, ...
## $ EmployeeNumber           <dbl> 1, 2, 4, 5, 7, 8, 10, 11, 12, 13, 14, 15, ...
## $ EnvironmentSatisfaction  <dbl> 2, 3, 4, 4, 1, 4, 3, 4, 4, 3, 1, 4, 1, 2, ...
## $ Gender                   <fct> Female, Male, Male, Female, Male, Male, Fe...
## $ HourlyRate               <dbl> 94, 61, 92, 56, 40, 79, 81, 67, 44, 94, 84...
## $ JobInvolvement           <dbl> 3, 2, 2, 3, 3, 3, 4, 3, 2, 3, 4, 2, 3, 3, ...
## $ JobLevel                 <dbl> 2, 2, 1, 1, 1, 1, 1, 1, 3, 2, 1, 2, 1, 1, ...
## $ JobRole                  <fct> Sales Executive, Research Scientist, Labor...
## $ JobSatisfaction          <dbl> 4, 2, 3, 3, 2, 4, 1, 3, 3, 3, 2, 3, 3, 4, ...
## $ MaritalStatus            <fct> Single, Married, Single, Married, Married,...
## $ MonthlyIncome            <dbl> 5993, 5130, 2090, 2909, 3468, 3068, 2670, ...
## $ MonthlyRate              <dbl> 19479, 24907, 2396, 23159, 16632, 11864, 9...
## $ NumCompaniesWorked       <dbl> 8, 1, 6, 1, 9, 0, 4, 1, 0, 6, 0, 0, 1, 0, ...
## $ Over18                   <fct> Y, Y, Y, Y, Y, Y, Y, Y, Y, Y, Y, Y, Y, Y, ...
## $ OverTime                 <fct> Yes, No, Yes, Yes, No, No, Yes, No, No, No...
## $ PercentSalaryHike        <dbl> 11, 23, 15, 11, 12, 13, 20, 22, 21, 13, 13...
## $ PerformanceRating        <dbl> 3, 4, 3, 3, 3, 3, 4, 4, 4, 3, 3, 3, 3, 3, ...
## $ RelationshipSatisfaction <dbl> 1, 4, 2, 3, 4, 3, 1, 2, 2, 2, 3, 4, 4, 3, ...
## $ StandardHours            <dbl> 80, 80, 80, 80, 80, 80, 80, 80, 80, 80, 80...
## $ StockOptionLevel         <dbl> 0, 1, 0, 0, 1, 0, 3, 1, 0, 2, 1, 0, 1, 1, ...
## $ TotalWorkingYears        <dbl> 8, 10, 7, 8, 6, 8, 12, 1, 10, 17, 6, 10, 5...
## $ TrainingTimesLastYear    <dbl> 0, 3, 3, 3, 3, 2, 3, 2, 2, 3, 5, 3, 1, 2, ...
## $ WorkLifeBalance          <dbl> 1, 3, 3, 3, 3, 2, 2, 3, 3, 2, 3, 3, 2, 3, ...
## $ YearsAtCompany           <dbl> 6, 10, 0, 8, 2, 7, 1, 1, 9, 7, 5, 9, 5, 2,...
## $ YearsInCurrentRole       <dbl> 4, 7, 0, 7, 2, 7, 0, 0, 7, 7, 4, 5, 2, 2, ...
## $ YearsSinceLastPromotion  <dbl> 0, 1, 0, 3, 2, 3, 0, 0, 1, 7, 0, 0, 4, 1, ...
## $ YearsWithCurrManager     <dbl> 5, 7, 0, 0, 2, 6, 0, 0, 8, 7, 3, 8, 3, 2, ...
```

Setting up h2o

Next, we need to initialize the Java Virtual Machine (JVM) that *h2o* uses locally. Also, we will turn off output of progress bars so we aren't flooded with unnecessary detail at this time.

```
h2o.init()
```

```
##  Connection successful!
##
## R is connected to the H2O cluster:
##     H2O cluster uptime:         2 minutes 58 seconds
##     H2O cluster timezone:       America/New_York
##     H2O data parsing timezone:  UTC
##     H2O cluster version:        3.30.1.2
##     H2O cluster version age:    5 months and 4 days !!!
##     H2O cluster name:           H2O_started_from_R_mjvat_mvw943
##     H2O cluster total nodes:    1
##     H2O cluster total memory:   15.94 GB
##     H2O cluster total cores:    24
##     H2O cluster allowed cores:  24
```

```
##     H2O cluster healthy:        TRUE
##     H2O Connection ip:          localhost
##     H2O Connection port:        54321
##     H2O Connection proxy:       NA
##     H2O Internal Security:      FALSE
##     H2O API Extensions:         Amazon S3, Algos, AutoML, Core V3, TargetEncoder, Core V4
##     R Version:                  R version 3.6.3 (2020-02-29)
```

```
## Warning in h2o.clusterInfo():
## Your H2O cluster version is too old (5 months and 4 days)!
## Please download and install the latest version from http://h2o.ai/download/
```

```
h2o.no_progress()
```

Splitting the data into train, validation, and test sets is necessary in order to train the algorithm and then test how well it performs.

This is one sort of method - many algorithms simply have train/test datasets - but adding a validation set enables an estimation of the model's skill while tuning the model's hyperparameters.

It is used to give an unbiased estimate of the final tuned model because the algorithm has not seen this data before; thus, it has not trained on this data. There are other methods to calculate an unbiased estimate as well (e.g. k-fold cross-validation).

```
hr_data_h2o <- as.h2o(hr_data)

split_h2o <- h2o.splitFrame(hr_data_h2o, c(0.7, 0.15), seed = 1234)

train_h2o <- h2o.assign(split_h2o[[1]], "train" ) # 70%
valid_h2o <- h2o.assign(split_h2o[[2]], "valid" ) # 15%
test_h2o  <- h2o.assign(split_h2o[[3]], "test" )  # 15%
```

We are aiming to predict employee turnover (Attrition) and the features (other columns) are used to model the prediction. Thus, we set the names for the inputs into the model.

```
y <- "Attrition"
x <- setdiff(names(train_h2o), y)
```

Data Analysis

We are now ready to run the automated ML function from the *h2o* package.

```
# x = x: names of our feature columns
# y = y: name of our target columns
# training_frame = train_h2o: training set of 70% of the data
# leaderboard_frame = valid_h2o: validation set of 15% of the data
  # this is to ensure the model does not overfit the data
# max_runtime_secs = 60: this is to speed up h2o's modeling
  # algorithm has number of large complex models - this is expedition
```

```r
  # at the expense of some accuracy

automl_models_h2o <- h2o.automl(
  x = x,
  y = y,
  training_frame    = train_h2o,
  leaderboard_frame = valid_h2o,
  max_runtime_secs  = 60)
```

```
##
## 16:12:00.868: AutoML: XGBoost is not available; skipping it.
```

Let's extract the leader model and predict on hold-out set - *test_h2o*.

All of the models are stored in the *automl_models_h2o* object. However, we really only care about the leader, which is the best model in terms of accuracy on the validation set.

```r
lb <- automl_models_h2o@leaderboard
print(lb) # only printing out top 6
```

```
##                                              model_id       auc   logloss
## 1 StackedEnsemble_BestOfFamily_AutoML_20210208_161200 0.8290201 0.3513361
## 2                          GLM_1_AutoML_20210208_161200 0.8239314 0.3564134
## 3    StackedEnsemble_AllModels_AutoML_20210208_161200 0.8216051 0.3641425
## 4                 DeepLearning_1_AutoML_20210208_161200 0.8198604 0.3620849
## 5 DeepLearning_grid__3_AutoML_20210208_161200_model_1 0.8195696 0.3591737
## 6 DeepLearning_grid__3_AutoML_20210208_161200_model_2 0.8060483 0.3663345
##       aucpr mean_per_class_error      rmse       mse
## 1 0.5924683            0.2477464 0.3291349 0.1083298
## 2 0.5157477            0.2477464 0.3300031 0.1089021
## 3 0.5747310            0.2581419 0.3360714 0.1129440
## 4 0.6034857            0.2422216 0.3221600 0.1037871
## 5 0.5629171            0.2102355 0.3294684 0.1085495
## 6 0.5355933            0.2574876 0.3334030 0.1111575
##
## [30 rows x 7 columns]
```

```r
# Extract leader model

automl_leader <- automl_models_h2o@leader
automl_leader
```

```
## Model Details:
## ==============
##
## H2OBinomialModel: stackedensemble
## Model ID:  StackedEnsemble_BestOfFamily_AutoML_20210208_161200
## Number of Base Models: 5
##
## Base Models (count by algorithm type):
##
```

```
## deeplearning          drf          gbm          glm
##            1            2            1            1
##
## Metalearner:
##
## Metalearner algorithm: glm
## Metalearner cross-validation fold assignment:
##   Fold assignment scheme: AUTO
##   Number of folds: 5
##   Fold column: NULL
## Metalearner hyperparameters:
##
##
## H2OBinomialMetrics: stackedensemble
## ** Reported on training data. **
##
## MSE:  0.03088113
## RMSE:  0.1757303
## LogLoss:  0.1268299
## Mean Per-Class Error:  0.05209601
## AUC:  0.9942664
## AUCPR:  0.9720942
## Gini:  0.9885328
##
## Confusion Matrix (vertical: actual; across: predicted) for F1-optimal threshold:
##         No Yes    Error      Rate
## No     851  19 0.021839   =19/870
## Yes     14 156 0.082353   =14/170
## Totals 865 175 0.031731  =33/1040
##
## Maximum Metrics: Maximum metrics at their respective thresholds
##                         metric threshold      value idx
## 1                       max f1  0.288494   0.904348 143
## 2                       max f2  0.189448   0.941704 176
## 3                 max f0point5  0.421826   0.929919 114
## 4                 max accuracy  0.300808   0.968269 141
## 5                max precision  0.994804   1.000000   0
## 6                   max recall  0.149533   1.000000 197
## 7              max specificity  0.994804   1.000000   0
## 8             max absolute_mcc  0.288494   0.885466 143
## 9   max min_per_class_accuracy  0.227559   0.959770 163
## 10 max mean_per_class_accuracy  0.189448   0.968830 176
## 11                     max tns  0.994804 870.000000   0
## 12                     max fns  0.994804 169.000000   0
## 13                     max fps  0.021818 870.000000 399
## 14                     max tps  0.149533 170.000000 197
## 15                     max tnr  0.994804   1.000000   0
## 16                     max fnr  0.994804   0.994118   0
## 17                     max fpr  0.021818   1.000000 399
## 18                     max tpr  0.149533   1.000000 197
##
## Gains/Lift Table: Extract with `h2o.gainsLift(<model>, <data>)` or `h2o.gainsLift(<model>, valid=<T/
##
## H2OBinomialMetrics: stackedensemble
```

```
## ** Reported on cross-validation data. **
## ** 5-fold cross-validation on training data (Metrics computed for combined holdout predictions) **
##
## MSE:  0.08709268
## RMSE:  0.2951147
## LogLoss:  0.3042822
## Mean Per-Class Error:  0.239858
## AUC:  0.8330325
## AUCPR:  0.6662774
## Gini:  0.6660649
##
## Confusion Matrix (vertical: actual; across: predicted) for F1-optimal threshold:
##         No Yes    Error       Rate
## No     816  54 0.062069    =54/870
## Yes     71  99 0.417647    =71/170
## Totals 887 153 0.120192  =125/1040
##
## Maximum Metrics: Maximum metrics at their respective thresholds
##                         metric threshold      value idx
## 1                       max f1  0.318962   0.613003 132
## 2                       max f2  0.109532   0.647003 257
## 3                 max f0point5  0.516124   0.703971  86
## 4                 max accuracy  0.516124   0.894231  86
## 5                max precision  0.991754   1.000000   0
## 6                   max recall  0.026013   1.000000 389
## 7              max specificity  0.991754   1.000000   0
## 8             max absolute_mcc  0.516124   0.559718  86
## 9   max min_per_class_accuracy  0.124328   0.752941 242
## 10 max mean_per_class_accuracy  0.222953   0.777113 165
## 11                     max tns  0.991754 870.000000   0
## 12                     max fns  0.991754 168.000000   0
## 13                     max fps  0.020073 870.000000 399
## 14                     max tps  0.026013 170.000000 389
## 15                     max tnr  0.991754   1.000000   0
## 16                     max fnr  0.991754   0.988235   0
## 17                     max fpr  0.020073   1.000000 399
## 18                     max tpr  0.026013   1.000000 389
##
## Gains/Lift Table: Extract with `h2o.gainsLift(<model>, <data>)` or `h2o.gainsLift(<model>, valid=<T/
```

Prediction, Test Performance, and Confusion Matrix

Now we can predict on our test set, which is unseen from our modeling process - it is a true test of performance.

```
# Predict on hold-out set, test_h2o

pred_h2o <- h2o.predict(object = automl_leader, newdata = test_h2o)
```

Here, we can evaluate our model - we'll reformat the test set to add the predictions column to analyze *actual* vs. *predictions* side-by-side

```r
# Prep for performance assessment

test_performance <- test_h2o %>%
  tibble::as_tibble() %>%
  select(Attrition) %>%
  add_column(Predicted = as.vector(pred_h2o$predict)) %>%
  mutate_if(is.character, as.factor)
head(test_performance, n = 10) %>% kable(align = "cc")
```

Attrition	Predicted
No	No
No	No
Yes	Yes
No	No
No	No
No	No
Yes	Yes
No	No
No	No
Yes	Yes

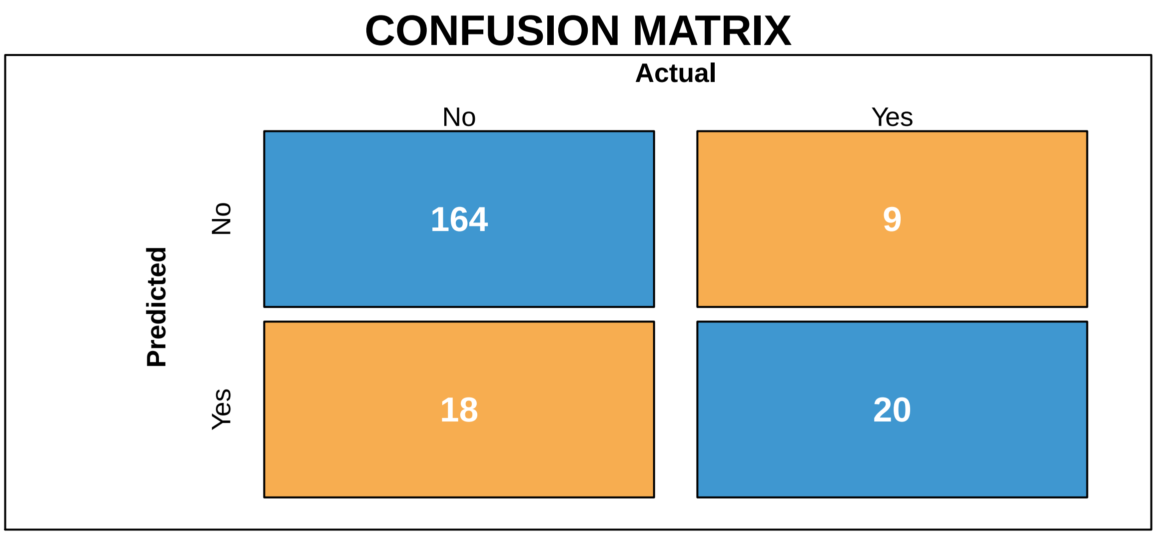
```r
# prints first 10 rows
```

We can use the *table()* function to quickly get a confusion table of the results. In the field of ML, a **confusion matrix** is a specific table layout that allows the visualization of the performance of an algorithm.

We see that the leader model wasn't perfect, but it did a decent job at identifying employees that are likely to quit. For perspective, a logistic regression would not perform nearly this well

```r
# Confusion table counts

confusion_matrix <- test_performance %>% table()
#confusion_matrix %>% kable(caption = "Predicted")
cm <- confusionMatrix(reference = test_performance$Attrition, data = test_performance$Predicted)

draw_confusion_matrix <- function(cm) {

  layout(matrix(c(1,1,2)))
  par(mar=c(2,2,2,2))
  plot(c(100, 345), c(300, 450), type = "n", xlab="", ylab="", xaxt='n', yaxt='n')
  title('CONFUSION MATRIX', cex.main=2)

  # create the matrix
  rect(150, 430, 240, 370, col='#3F97D0')
  text(195, 435, 'No', cex=1.2)
  rect(250, 430, 340, 370, col='#F7AD50')
  text(295, 435, 'Yes', cex=1.2)
  text(125, 370, 'Predicted', cex=1.3, srt=90, font=2)
  text(245, 450, 'Actual', cex=1.3, font=2)
  rect(150, 305, 240, 365, col='#F7AD50')
  rect(250, 305, 340, 365, col='#3F97D0')
```

```
  text(140, 400, 'No', cex=1.2, srt=90)
  text(140, 335, 'Yes', cex=1.2, srt=90)

  # add in the cm results
  res <- as.numeric(cm$table)
  text(195, 400, res[1], cex=1.6, font=2, col='white')
  text(195, 335, res[2], cex=1.6, font=2, col='white')
  text(295, 400, res[3], cex=1.6, font=2, col='white')
  text(295, 335, res[4], cex=1.6, font=2, col='white')

 }

draw_confusion_matrix(cm)
```



Now that we see the output from the confusion matrix, We can run through a binary classification analysis to understand the model's performance.

```
tn <- confusion_matrix[1] # true negatives
tp <- confusion_matrix[4] # true positives
fp <- confusion_matrix[3] # false positives
fn <- confusion_matrix[2] # false negatives

accuracy <- (tp + tn) / (tp + tn + fp + fn)
misclassification_rate <- 1 - accuracy
recall <- tp / (tp + fn)
```

```r
precision <- tp / (tp + fp)
null_error_rate <- tn / (tp + tn + fp + fn)

# Changed to percentages since that is usually easier to interpret

percentage <- c(percent(accuracy), percent(misclassification_rate), percent(recall), percent(precision)
row_names <- c("Accuracy", "Misclassification Rate", "Recall", "Precision", "Null Error Rate")
df <- data.frame(row_names, percentage)
colnames(df) <- c("Evaluation Method", "Percentage")
df %>% kable()
```

Evaluation Method	Percentage
Accuracy	87%
Misclassification Rate	13%
Recall	69%
Precision	53%
Null Error Rate	78%

What Does This All Mean?

Analytical Development

NOTE: It is important to understand that accuracy can be misleading. 87% accuracy sounds pretty good - especially for modeling HR data. But if we simply just picked Attrition = No on every employee, we would get an accuracy of about 78%. That doesn't sound so great now does it?

Precision is when the model predicts yes, how often it is actually yes.

Recall (aka true positive rate or specificity) is when the actual value is yes, how often the model is correct.

Most HR divisions would rather incorrectly classify folks not looking to quit as high potential than classify those likely to quit as not at risk. This means that HR will then care about *Recall*.

As stated, Recall - when the actual value is Attrition = YES, how often that model predicts YES.

Recall for our model is 69% - in an HR context, there are 69% more employees that could potentially be targeted prior to quitting. Let's say an organization loses 100 employees per year, they could possibly target 69 of them, implementing measures to retain valuable employees.

Thus far, we have a very good model that is capable of making very accurate predictions on unseen data, but what can it tell us about what causes attrition? This is where we can use *lime*.

Setting Up lime

The *lime* package implements *lime()* in R.

NOTE: *lime* is not setup out-of-the-box to work with *h2o* but two custom functions will enable everything to work smoothly:

- *model_type*: Tells *lime* what type of model we are dealing with.
- *predict_model*: Allows *lime* to perform predictions that its algorithm can interpret.

```r
# Build Model Type function for congruence with h2o

model_type.H2OBinomialModel <- function(x, ...) {

  # Function tells lime() what model type we are dealing with
  # 'classification', 'regression', 'survival', 'clustering', 'multilabel', etc
  #
  # x is our h2o model

  return("classification")
}
```

The trick here is to realize that its inputs **must** be: 'x' (a model), 'newdata' (a dataframe object - this is essential), and 'type' (not used, but can be used to switch the output type).

Output is also tricky because it **must** be in the format of probabilities by classification.

```r
# Build predict_model function

predict_model.H2OBinomialModel <- function(x, newdata, type, ...) {

  # Function performs prediction and returns dataframe with Response
  #
  # x is h2o model
  # newdata is data frame
  # type is only setup for data frame

  pred <- h2o.predict(x, as.h2o(newdata))

  # return probs
  return(as.data.frame(pred[,-1]))

}
```

We can run the next script to show what the output looks like and test our predict_model function.

```r
# Test our predict_model() function

pm <- predict_model(x = automl_leader, newdata = as.data.frame(test_h2o[,-1]), type = 'raw')
pm$EmpID <- seq.int(nrow(pm))
pm <- pm[c(3,1,2)]
kable(pm[1:10,], align = "ccc")
```

EmpID	No	Yes
1	0.8223769	0.1776231
2	0.9589187	0.0410813
3	0.0353387	0.9646613
4	0.9679815	0.0320185
5	0.9064246	0.0935754
6	0.9645216	0.0354784

EmpID	No	Yes
7	0.1530032	0.8469968
8	0.9342847	0.0657153
9	0.8362481	0.1637519
10	0.5186332	0.4813668

Now, for the fun part, we create an explainer using the *lime()* function by passing the training dataset without the *Attribution* column. It must be a data frame - this is important. Our *predict_model()* function will transform it into an *h2o* object.

We will set **model = automl_leader** and **bin_continuous = FALSE**. We could do bin_continuous variables, but this may not make sense for categorical numeric data that we didn't coerce into factors.

```
# Run lime() on training set

explainer <- lime::lime(as.data.frame(train_h2o[,-1]), model = automl_leader,
                        bin_continuous = FALSE)
```

Next, we will run *explain()*, which returns our explanation. This can take a few minutes to run, so we will limit it to the first 10 rows of the test dataset.

We will set **n_labels = 1** because we care about explaining a single class. Also, setting **n_features = 5** will return the top five fetaures that are critical to each case. Lastly, setting *kernel_width = 0.5** allows us to increase the *model_r2* value by shrinking the localized evaluation.

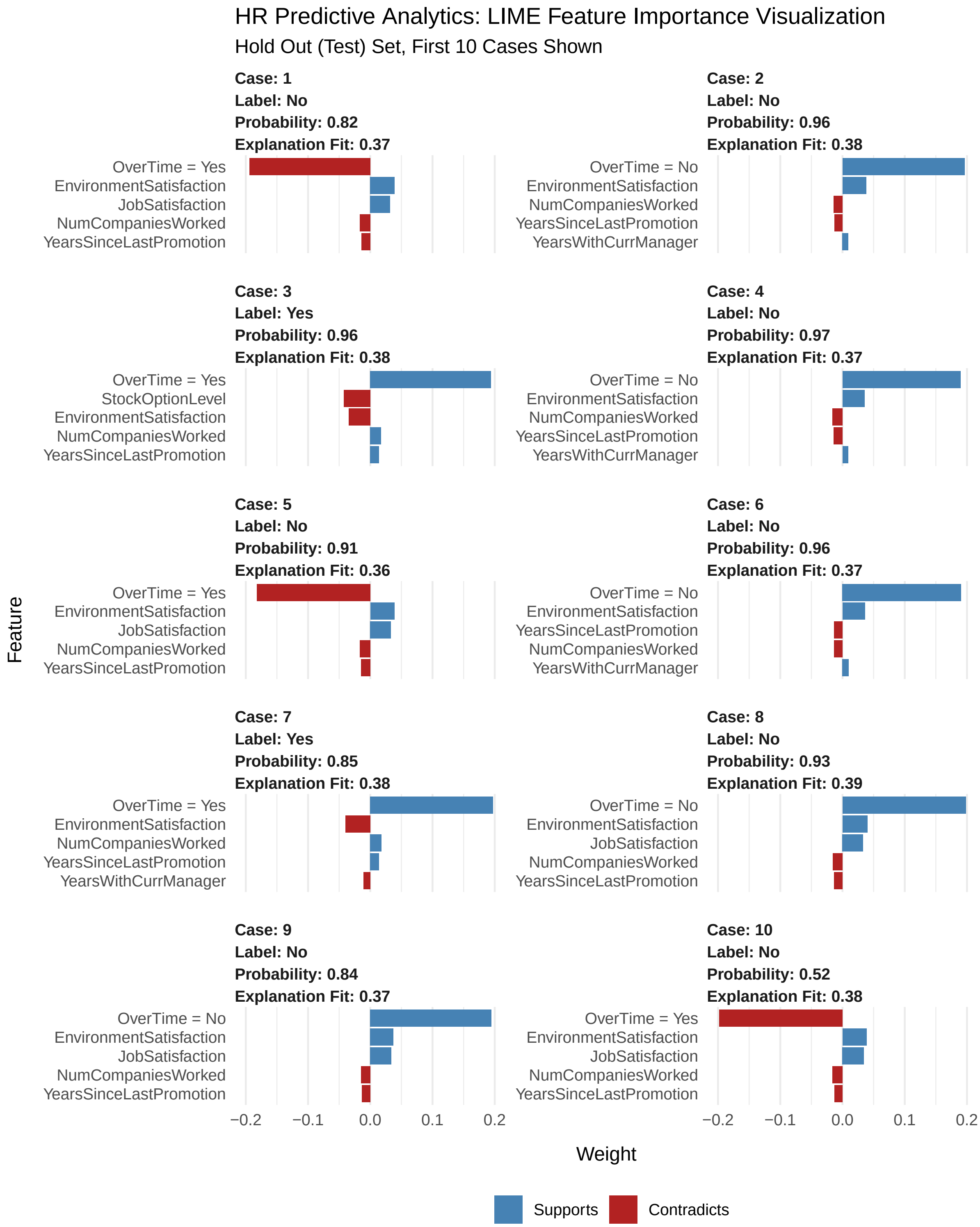
```
# Run explain() on explainer

explanation <- lime::explain(
  as.data.frame(test_h2o[1:10,-1]),
  explainer    = explainer,
  n_labels     = 1,
  n_features   = 5,
  kernel_width = 0.5)
```

Feature Importance Visualization

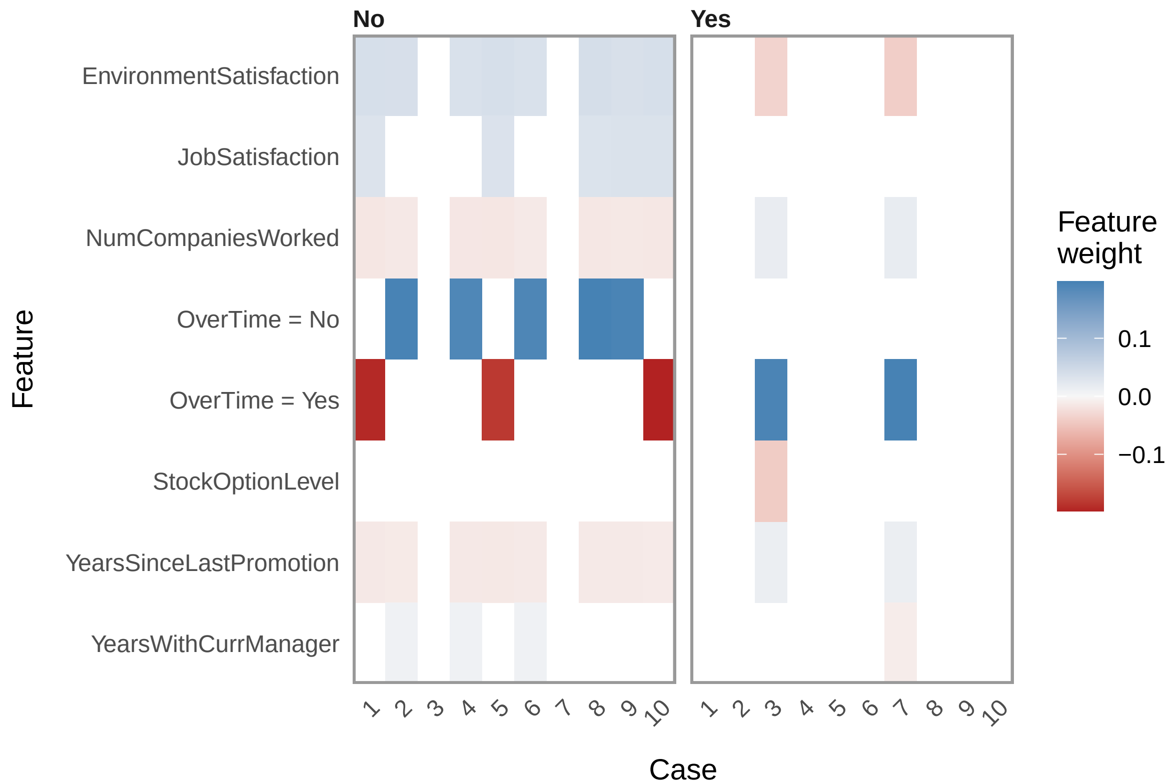
The payoff for all this work is the *Feature Importance Plot*. We can visualize each of the ten cases - *observations* - from the test dataset. The top four features for each case are shown. **NOTE:** they are not the same for each case. *Blue* bars mean that the feature **supports** the model conclusion, *Red* bars **contradict**.

Focus on the cases with **Label = YES** - which are predicted to have attrition. Are there commonalities? Common themes may only exist in a couple cases, but they can be used to potentially generalize to the larger population.

```
plot_features(explanation) +
  labs(title = "HR Predictive Analytics: LIME Feature Importance Visualization",
       subtitle = "Hold Out (Test) Set, First 10 Cases Shown")
```


HR Predictive Analytics: LIME Feature Importance Visualization
Hold Out (Test) Set, First 10 Cases Shown
Case: 1
Label: No
Probability: 0.82
Explanation Fit: 0.37
OverTime = Yes
EnvironmentSatisfaction
JobSatisfaction
NumCompaniesWorked
YearsSinceLastPromotion
Case: 2
Label: No
Probability: 0.96
Explanation Fit: 0.38
OverTime = No
EnvironmentSatisfaction
NumCompaniesWorked
YearsSinceLastPromotion
YearsWithCurrManager
Case: 3
Label: Yes
Probability: 0.96
Explanation Fit: 0.38
OverTime = Yes
StockOptionLevel
EnvironmentSatisfaction
NumCompaniesWorked
YearsSinceLastPromotion
Case: 4
Label: No
Probability: 0.97
Explanation Fit: 0.37
OverTime = No
EnvironmentSatisfaction
NumCompaniesWorked
YearsSinceLastPromotion
YearsWithCurrManager
Case: 5
Label: No
Probability: 0.91
Explanation Fit: 0.36
OverTime = Yes
EnvironmentSatisfaction
JobSatisfaction
NumCompaniesWorked
YearsSinceLastPromotion
Case: 6
Label: No
Probability: 0.96
Explanation Fit: 0.37
OverTime = No
EnvironmentSatisfaction
YearsSinceLastPromotion
NumCompaniesWorked
YearsWithCurrManager
Case: 7
Label: Yes
Probability: 0.85
Explanation Fit: 0.38
OverTime = Yes
EnvironmentSatisfaction
NumCompaniesWorked
YearsSinceLastPromotion
YearsWithCurrManager
Case: 8
Label: No
Probability: 0.93
Explanation Fit: 0.39
OverTime = No
EnvironmentSatisfaction
JobSatisfaction
NumCompaniesWorked
YearsSinceLastPromotion
Case: 9
Label: No
Probability: 0.84
Explanation Fit: 0.37
OverTime = No
EnvironmentSatisfaction
JobSatisfaction
NumCompaniesWorked
YearsSinceLastPromotion
Case: 10
Label: No
Probability: 0.52
Explanation Fit: 0.38
OverTime = Yes
EnvironmentSatisfaction
JobSatisfaction
NumCompaniesWorked
YearsSinceLastPromotion
Feature
−0.2 −0.1 0.0 0.1 0.2
−0.2 −0.1 0.0 0.1 0.2
Weight
Supports
Contradicts

```r
plot_explanations(explanation)
```



Did we discover what features are linked to Employee Attrition? Overtime seems to be the largest motivator for attrition.

```r
# Focus on critical features of attrition

attrition_critical_features <- hr_data %>%
  select(Attrition, TrainingTimesLastYear, JobRole, OverTime, NumCompaniesWorked,
         Age, YearsSinceLastPromotion) %>% rowid_to_column(var = "Case")
kable(attrition_critical_features[1:10,2:7], align = "ccccccc")
```

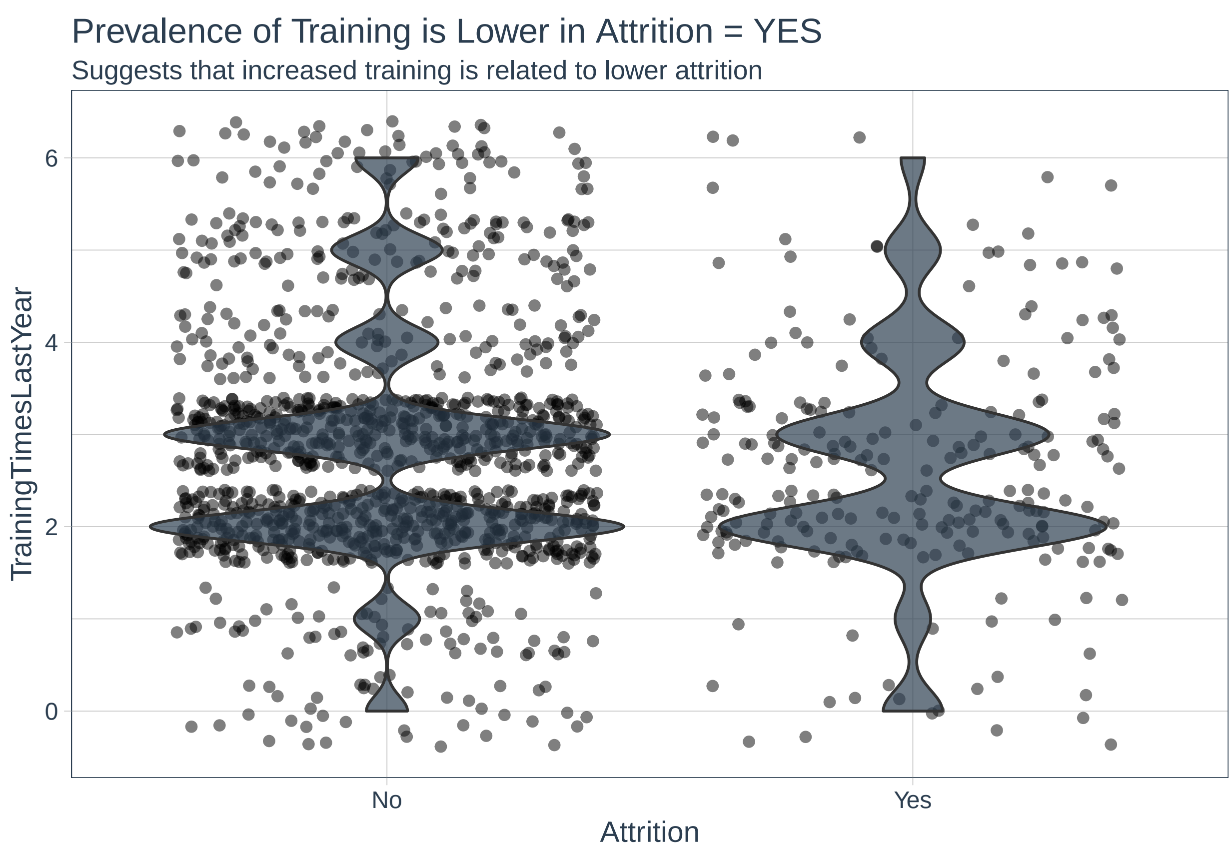
Attrition	TrainingTimesLastYear	JobRole	OverTime	NumCompaniesWorked	Age
Yes	0	Sales Executive	Yes	8	41
No	3	Research Scientist	No	1	49
Yes	3	Laboratory Technician	Yes	6	37
No	3	Research Scientist	Yes	1	33
No	3	Laboratory Technician	No	9	27
No	2	Laboratory Technician	No	0	32
No	3	Laboratory Technician	Yes	4	59
No	2	Laboratory Technician	No	1	30
No	2	Manufacturing Director	No	0	38
No	3	Healthcare Representative	No	6	36

What are the Indicators?

Charts

Training

```
attrition_critical_features %>% ggplot(aes(Attrition, TrainingTimesLastYear)) +
  geom_jitter(alpha = 0.5, fill = palette_light()[[1]]) +
  geom_violin(alpha = 0.7, fill = palette_light()[[1]]) +
  theme_tq() +
  labs(title = "Prevalence of Training is Lower in Attrition = YES",
    subtitle = "Suggests that increased training is related to lower attrition")
```

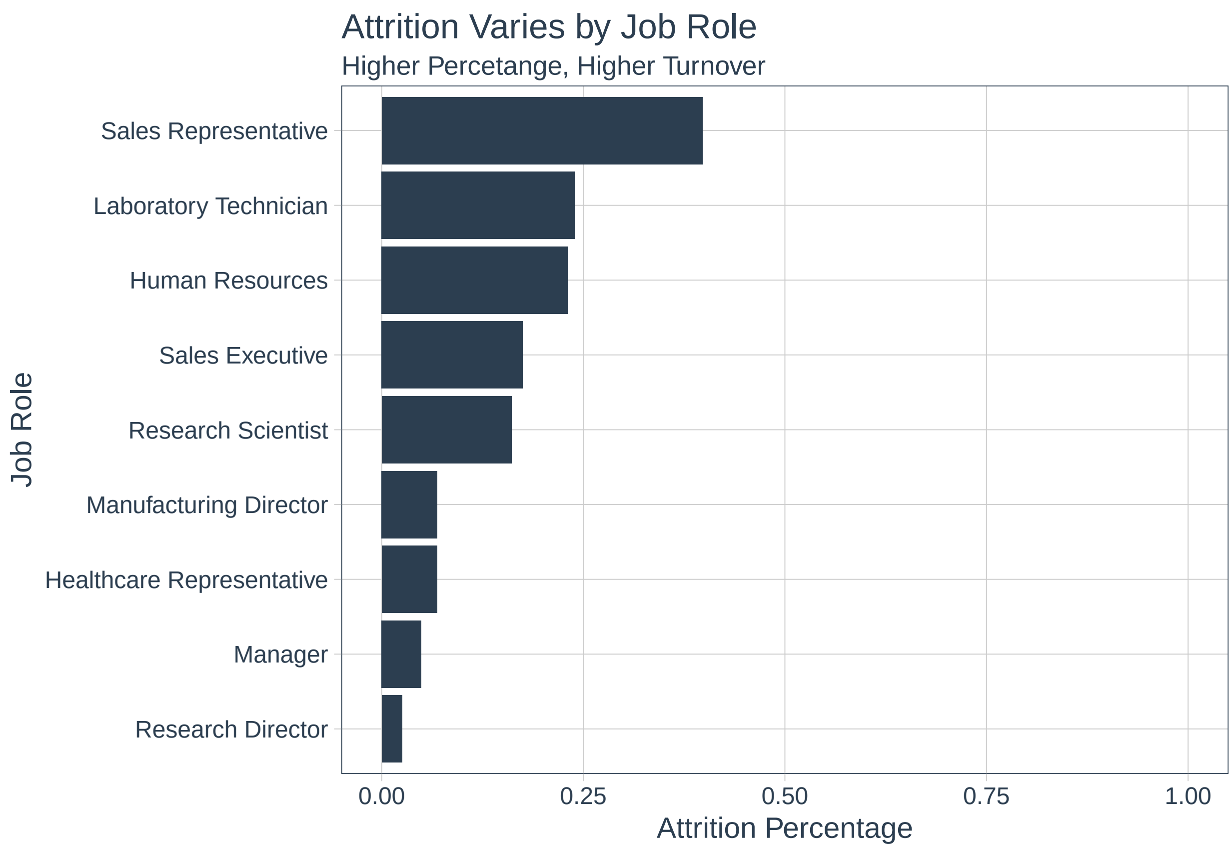


Job Role

```
attrition_critical_features %>% group_by(JobRole, Attrition) %>%
  summarize(total = n()) %>% spread(key = Attrition, value = total) %>%
  mutate(pct_attrition = Yes / (Yes + No)) %>%
  ggplot(aes(x = forcats::fct_reorder(JobRole, pct_attrition), y = pct_attrition)) +
  geom_bar(stat = "identity", alpha = 1, fill = palette_light()[[1]]) +
  expand_limits(y = c(0,1)) + coord_flip() + theme_tq() +
  labs(title = "Attrition Varies by Job Role",
```

```
       subtitle = "Higher Percetange, Higher Turnover",
       y = "Attrition Percentage",
       x = "Job Role")
```

```
## `summarise()` regrouping output by 'JobRole' (override with `.groups` argument)
```

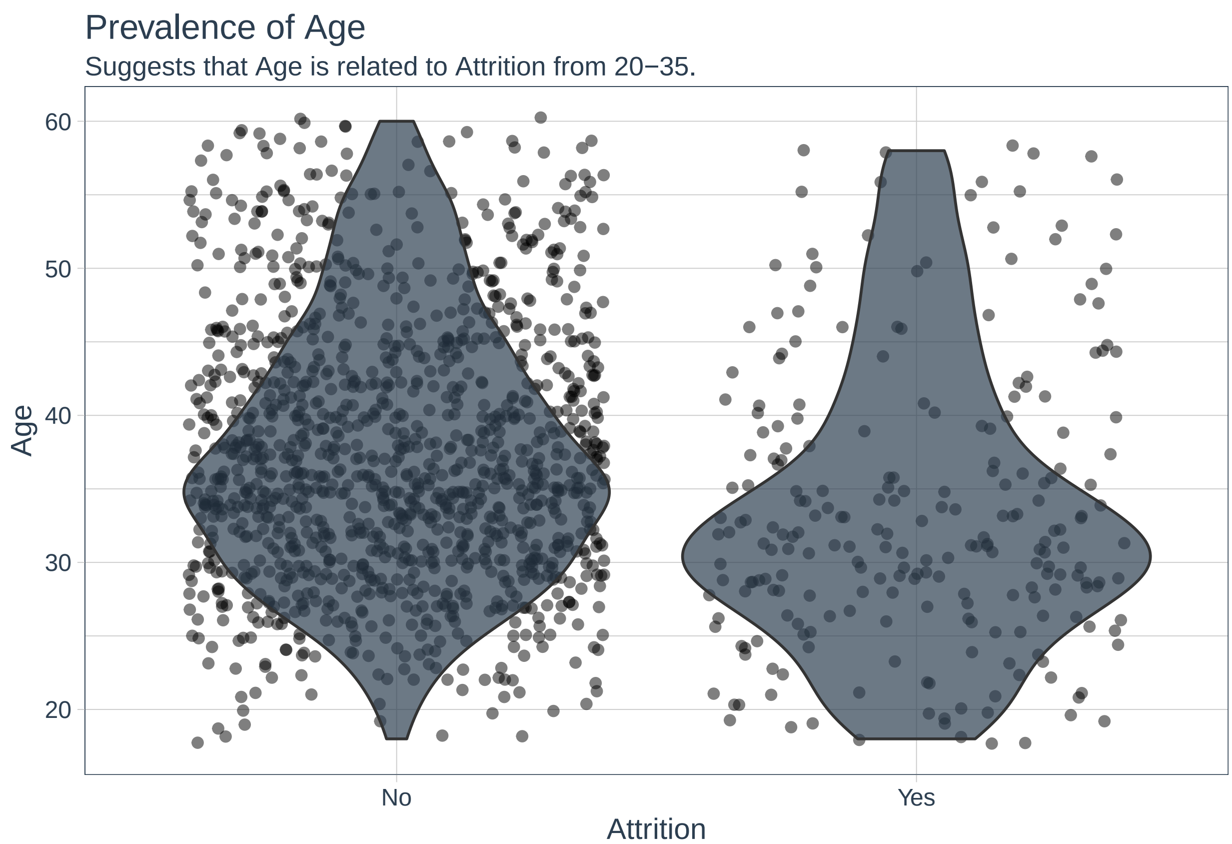


Overtime

```
attrition_critical_features %>% mutate(OverTime = case_when(
  OverTime == "Yes" ~ 1,
  OverTime == "No" ~ 0)) %>%
  ggplot(aes(Attrition, OverTime)) + geom_jitter(alpha = 0.5,
    fill = palette_light()[[1]]) + geom_violin(alpha = 0.7,
    fill = palette_light()[[1]]) + theme_tq() +
  labs(title = "Prevlance of OverTime is Higher in Attrition = YES",
       subtitle = "Suggests increased OverTime is related to higher attrition")
```

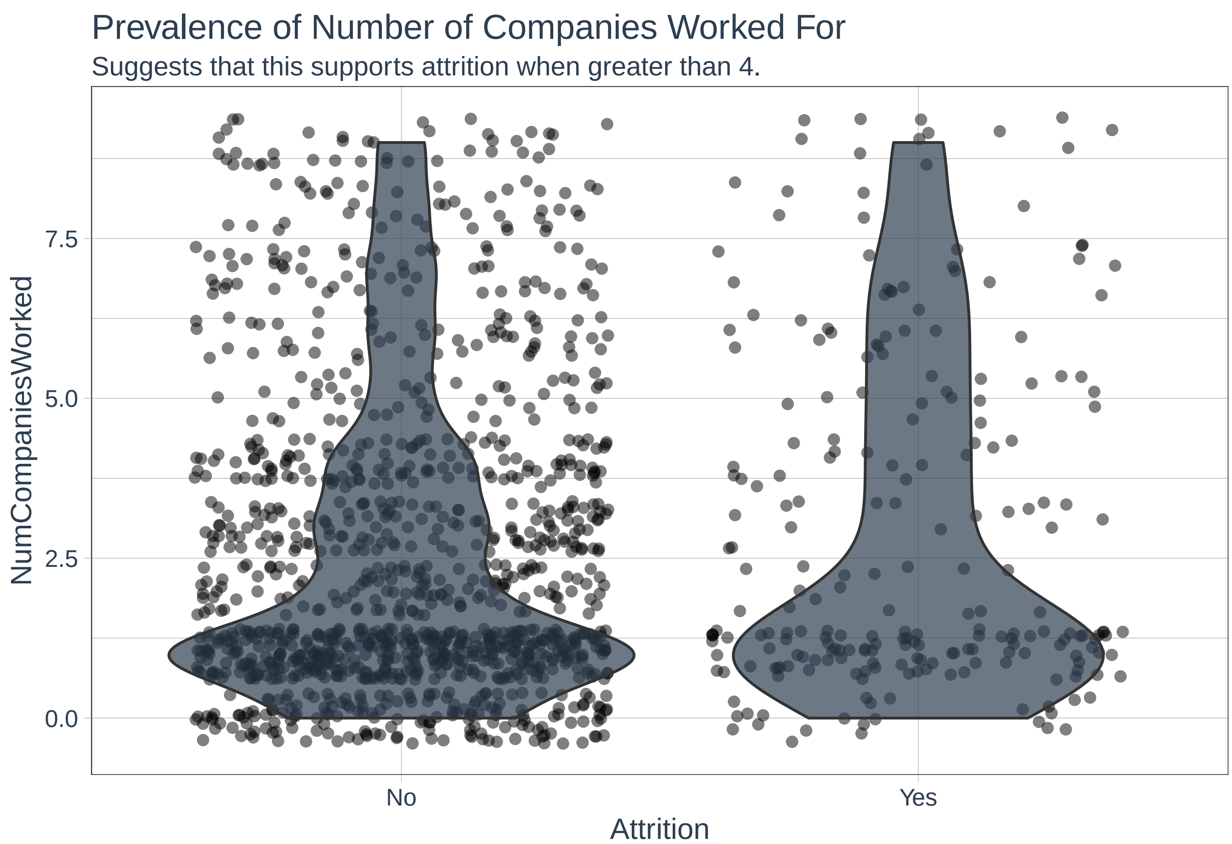
Age

```
attrition_critical_features %>% ggplot(aes(Attrition, Age)) +
  geom_jitter(alpha = 0.5, fill = palette_light()[[1]]) +
  geom_violin(alpha = 0.7, fill = palette_light()[[1]]) +
  theme_tq() +
  labs(title = "Prevalence of Age",
       subtitle = "Suggests that Age is related to Attrition from 20-35.")
```



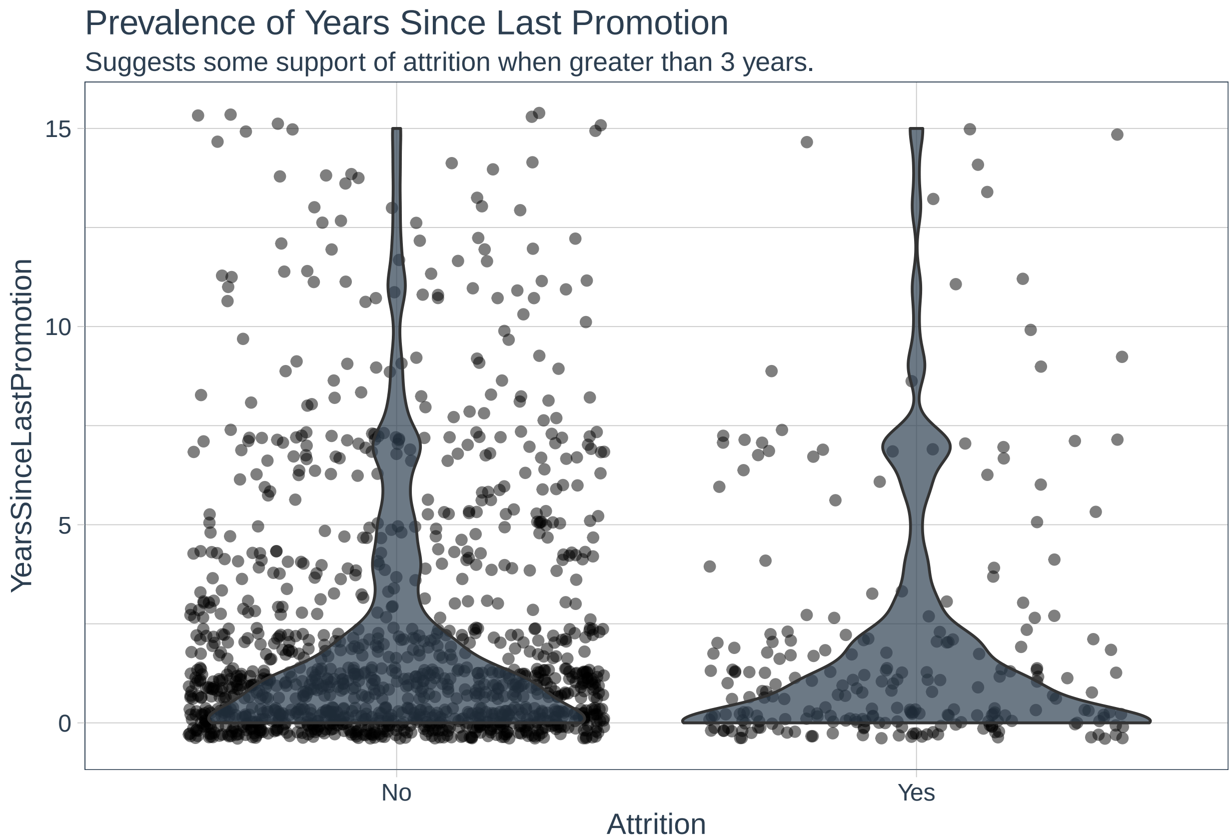
Number of Companies Employee has Worked For

```
attrition_critical_features %>% ggplot(aes(Attrition, NumCompaniesWorked)) +
  geom_jitter(alpha = 0.5, fill = palette_light()[[1]]) +
  geom_violin(alpha = 0.7, fill = palette_light()[[1]]) +
  theme_tq() +
  labs(title = "Prevalence of Number of Companies Worked For",
       subtitle = "Suggests that this supports attrition when greater than 4.")
```



Years Since Last Promotion

```
attrition_critical_features %>% ggplot(aes(Attrition, YearsSinceLastPromotion)) +
  geom_jitter(alpha = 0.5, fill = palette_light()[[1]]) +
  geom_violin(alpha = 0.7, fill = palette_light()[[1]]) +
  theme_tq() +
  labs(title = "Prevalence of Years Since Last Promotion",
       subtitle = "Suggests some support of attrition when greater than 3 years.")
```



Conclusions

The *autoML* algorithm from *H2O.ai* worked well for classifying attrition with an accuracy around 87% on unseen / unmodeled data.

We then used *lime* to breakdown the complex ensemble model returned from *h2o* into critical features that are related to attrition.

Overall, this is a really useful example of where we can see how much ML and DS can be used in HR applications. These packages may be able to support OHR and augment the current attrition models.

```
sessionInfo()
```

```
## R version 3.6.3 (2020-02-29)
## Platform: x86_64-w64-mingw32/x64 (64-bit)
## Running under: Windows 10 x64 (build 19042)
##
## Matrix products: default
##
## locale:
## [1] LC_COLLATE=English_United States.1252
## [2] LC_CTYPE=English_United States.1252
## [3] LC_MONETARY=English_United States.1252
## [4] LC_NUMERIC=C
## [5] LC_TIME=English_United States.1252
```

```
##
## attached base packages:
## [1] tools     parallel  stats     graphics  grDevices utils     datasets
## [8] methods   base
##
## other attached packages:
##  [1] vip_0.2.2                  versions_0.3
##  [3] tinytex_0.25               timeDate_3043.102
##  [5] forcats_0.5.0              purrr_0.3.4
##  [7] tidyverse_1.3.0            tidyr_1.1.2
##  [9] tidyquant_1.0.0            quantmod_0.4.17
## [11] TTR_0.23-6                 PerformanceAnalytics_2.0.4
## [13] xts_0.12-0                 zoo_1.8-8
## [15] tibble_3.0.1               survival_3.2-3
## [17] stringr_1.4.0              stringi_1.4.6
## [19] statmod_1.4.34             scales_1.1.1
## [21] rjson_0.2.20               readxl_1.3.1
## [23] readr_1.3.1                RCurl_1.98-1.2
## [25] RColorBrewer_1.1-2         pdp_0.7.0
## [27] lubridate_1.7.9            lime_0.5.1
## [29] lava_1.6.7                 knitr_1.30
## [31] jsonlite_1.7.1             h2o_3.30.1.2
## [33] gridExtra_2.3              glmnet_4.0-2
## [35] Matrix_1.2-18              ggpubr_0.4.0
## [37] e1071_1.7-3                dplyr_1.0.0
## [39] doParallel_1.0.15          iterators_1.0.12
## [41] foreach_1.5.0              data.table_1.12.8
## [43] caTools_1.18.0             caret_6.0-86
## [45] ggplot2_3.3.2              lattice_0.20-41
## [47] car_3.0-10                 carData_3.0-4
## [49] bit_4.0.4                  anytime_0.3.7
##
## loaded via a namespace (and not attached):
##  [1] backports_1.1.10     plyr_1.8.6           splines_3.6.3
##  [4] digest_0.6.25        htmltools_0.5.0      fansi_0.4.1
##  [7] magrittr_1.5         openxlsx_4.1.5       recipes_0.1.12
## [10] modelr_0.1.8         gower_0.2.1          colorspace_1.4-1
## [13] blob_1.2.1           rvest_0.3.6          haven_2.3.1
## [16] xfun_0.17            crayon_1.3.4         glue_1.4.1
## [19] gtable_0.3.0         ipred_0.9-9          Quandl_2.10.0
## [22] shape_1.4.4          abind_1.4-5          DBI_1.1.0
## [25] rstatix_0.6.0        Rcpp_1.0.5           xtable_1.8-4
## [28] foreign_0.8-75       stats4_3.6.3         prodlim_2019.11.13
## [31] htmlwidgets_1.5.1    httr_1.4.2           ellipsis_0.3.1
## [34] farver_2.0.3         pkgconfig_2.0.3      nnet_7.3-14
## [37] dbplyr_1.4.4         utf8_1.1.4           tidyselect_1.1.0
## [40] labeling_0.3         rlang_0.4.7          reshape2_1.4.4
## [43] later_1.1.0.1        munsell_0.5.0        cellranger_1.1.0
## [46] cli_2.0.2            generics_0.0.2       pacman_0.5.1
## [49] broom_0.7.0          evaluate_0.14        fastmap_1.0.1
## [52] yaml_2.2.1           ModelMetrics_1.2.2.2 bit64_4.0.5
## [55] fs_1.5.0             zip_2.1.1            nlme_3.1-148
## [58] mime_0.9             xml2_1.3.2           compiler_3.6.3
## [61] shinythemes_1.1.2    rstudioapi_0.11      curl_4.3
```

```
## [64] ggsignif_0.6.0       reprex_0.3.0        highr_0.8
## [67] vctrs_0.3.2          pillar_1.4.4        lifecycle_0.2.0
## [70] bitops_1.0-6         httpuv_1.5.4        R6_2.4.1
## [73] promises_1.1.1       rio_0.5.16          codetools_0.2-16
## [76] MASS_7.3-51.6        assertthat_0.2.1    withr_2.2.0
## [79] hms_0.5.3            quadprog_1.5-8      grid_3.6.3
## [82] rpart_4.1-15         class_7.3-17        rmarkdown_2.3
## [85] pROC_1.16.2          shiny_1.4.0.2
```