CommunityContribution-Fall24

AUTHOR

Marissa Inga (mi2548) and Victoria Lin (vl2534)

Machine Learning in R

We wanted to take a closer look at how Machine Learning algorithms may be leveraged in R. We found that a range of resources are available, and decided to do a deep dive into the Random Forest library using the [UCI Heart Disease Dataset](#) from Kaggle.

Initial Setup

```
library(dplyr)
```

```
Attaching package: 'dplyr'

The following objects are masked from 'package:stats':

    filter, lag

The following objects are masked from 'package:base':

    intersect, setdiff, setequal, union
```

```
library(tidyverse)
```

```
── Attaching core tidyverse packages ───────────────────── tidyverse 2.0.0 ──
✔ forcats   1.0.0     ✔ readr     2.1.5
✔ ggplot2   3.5.1     ✔ stringr   1.5.1
✔ lubridate 1.9.3     ✔ tibble    3.2.1
✔ purrr     1.0.2     ✔ tidyr     1.3.1

── Conflicts ──────────────────────────────────── tidyverse_conflicts() ──
✖ dplyr::filter() masks stats::filter()
✖ dplyr::lag()    masks stats::lag()
ℹ Use the conflicted package (<http://conflicted.r-lib.org/>) to force all conflicts to
become errors
```

```
library(ggplot2)
```

Data Preparation

Data Loading

We chose to use the [UCI Heart Disease Dataset](#) from Kaggle as our dataset for this tutorial. This dataset is multivariate and provides 14 independent variables and 1 dependent variable, which is the predicted health attribute.

```
data <- read_csv('heart_disease_uci.csv', col_names = TRUE)
```

```
Rows: 920 Columns: 16
── Column specification ─────────────────────────────────────────────
Delimiter: ","
chr (6): sex, dataset, cp, restecg, slope, thal
dbl (8): id, age, trestbps, chol, thalch, oldpeak, ca, num
lgl (2): fbs, exang

ℹ Use `spec()` to retrieve the full column specification for this data.
ℹ Specify the column types or set `show_col_types = FALSE` to quiet this message.
```

```
head(data)
```

```
# A tibble: 6 × 16
     id   age sex    dataset   cp      trestbps  chol fbs   restecg thalch exang
  <dbl> <dbl> <chr>  <chr>     <chr>      <dbl> <dbl> <lgl> <chr>    <dbl> <lgl>
1     1    63 Male   Cleveland typica…      145   233 TRUE  lv hyp…    150 FALSE
2     2    67 Male   Cleveland asympt…      160   286 FALSE lv hyp…    108 TRUE
3     3    67 Male   Cleveland asympt…      120   229 FALSE lv hyp…    129 TRUE
4     4    37 Male   Cleveland non-an…      130   250 FALSE normal     187 FALSE
5     5    41 Female Cleveland atypic…      130   204 FALSE lv hyp…    172 FALSE
6     6    56 Male   Cleveland atypic…      120   236 FALSE normal     178 FALSE
# ℹ 5 more variables: oldpeak <dbl>, slope <chr>, ca <dbl>, thal <chr>,
#   num <dbl>
```

The columns are described below:

1. `id` (Unique id for each patient)

2. `age` (Age of the patient in years)

3. `origin` (place of study)

4. `sex` (Male/Female)

5. `cp` chest pain type ([typical angina, atypical angina, non-anginal, asymptomatic])

6. `trestbps` resting blood pressure (resting blood pressure (in mm Hg on admission to the hospital))

7. `chol` (serum cholesterol in mg/dl)

8. `fbs` (if fasting blood sugar > 120 mg/dl)

9. `restecg` (resting electrocardiographic results)
   – Values: [normal, stt abnormality, lv hypertrophy]

10. `thalach` : maximum heart rate achieved

11. `exang` : exercise-induced angina (True/ False)

12. `oldpeak` : ST depression induced by exercise relative to rest

13. `slope` : the slope of the peak exercise ST segment

14. `ca` : number of major vessels (0-3) colored by fluoroscopy

15. `thal` : [normal; fixed defect; reversible defect]

16. `num` : the predicted attribute

Data Cleaning

Now that we have loaded our data, we can take a look at the contents and do some cleaning. First we drop the id column:

```
data_cleaned <- data
data_cleaned <- select(data_cleaned, names(data_cleaned)[2:16])
```

Then we checked for NAs, and found a few in there, so we filled them with the median of the non-null numeric values, and for character columns used the mode.

```
colSums(is.na(data_cleaned))
```

age	sex	dataset	cp	trestbps	chol	fbs	restecg
0	0	0	0	59	30	90	2
thalch	exang	oldpeak	slope	ca	thal	num	
55	55	62	309	611	486	0	

```
data_cleaned$trestbps[is.na(data_cleaned$trestbps)] <- median(data_cleaned$trestb
data_cleaned$chol[is.na(data_cleaned$chol)] <- median(data_cleaned$chol, na.rm =
data_cleaned$fbs[is.na(data_cleaned$fbs)] <- median(data_cleaned$fbs, na.rm = TRU
data_cleaned$restecg[is.na(data_cleaned$restecg)] <- mode(data_cleaned$restecg)
data_cleaned$thalch[is.na(data_cleaned$thalch)] <- median(data_cleaned$thalch, na
data_cleaned$exang[is.na(data_cleaned$exang)] <- median(data_cleaned$exang, na.rm
data_cleaned$oldpeak[is.na(data_cleaned$oldpeak)] <- median(data_cleaned$oldpeak,
data_cleaned$slope[is.na(data_cleaned$slope)] <- mode(data_cleaned$slope)
data_cleaned$ca[is.na(data_cleaned$ca)] <- median(data_cleaned$ca, na.rm = TRUE)
data_cleaned$thal[is.na(data_cleaned$thal)] <- mode(data_cleaned$thal)

colSums(is.na(data_cleaned))
```

age	sex	dataset	cp	trestbps	chol	fbs	restecg
0	0	0	0	0	0	0	0
thalch	exang	oldpeak	slope	ca	thal	num	
0	0	0	0	0	0	0	

Now we take a look at the `character` type columns, and convert to factor:

```
data_cleaned$sex <- as.factor(data_cleaned$sex)
data_cleaned$dataset <- as.factor(data_cleaned$dataset)
data_cleaned$cp <- as.factor(data_cleaned$cp)
data_cleaned$restecg <- as.factor(data_cleaned$restecg)
data_cleaned$slope <- as.factor(data_cleaned$slope)
data_cleaned$thal <- as.factor(data_cleaned$thal)
data_cleaned$num <- as.factor(data_cleaned$num)


summary(Filter(is.factor, data_cleaned))
```

```
      sex                dataset                cp                    restecg
 Female:194    Cleveland    :304    asymptomatic  :496    character      :  2
 Male  :726    Hungary      :293    atypical angina:174   lv hypertrophy :188
               Switzerland  :123    non-anginal   :204    normal         :551
               VA Long Beach:200    typical angina : 46   st-t abnormality:179

         slope                  thal        num
 character  :309    character        :486    0:411
 downsloping: 63    fixed defect     : 46    1:265
 flat       :345    normal           :196    2:109
 upsloping  :203    reversable defect:192    3:107
                                             4: 28
```

```
summary(Filter(is.numeric, data_cleaned))
```

```
      age            trestbps         chol            fbs             thalch
 Min.   :28.00    Min.   :  0    Min.   :  0.0    Min.   :0.00    Min.   : 60.0
 1st Qu.:47.00    1st Qu.:120    1st Qu.:177.8    1st Qu.:0.00    1st Qu.:120.0
 Median :54.00    Median :130    Median :223.0    Median :0.00    Median :140.0
 Mean   :53.51    Mean   :132    Mean   :199.9    Mean   :0.15    Mean   :137.7
 3rd Qu.:60.00    3rd Qu.:140    3rd Qu.:267.0    3rd Qu.:0.00    3rd Qu.:156.0
 Max.   :77.00    Max.   :200    Max.   :603.0    Max.   :1.00    Max.   :202.0
    oldpeak             ca
 Min.   :-2.6000    Min.   :0.0000
 1st Qu.: 0.0000    1st Qu.:0.0000
 Median : 0.5000    Median :0.0000
 Mean   : 0.8533    Mean   :0.2272
 3rd Qu.: 1.5000    3rd Qu.:0.0000
 Max.   : 6.2000    Max.   :3.0000
```

Now that our data is cleaned, we want to split our data into a train set and test set. R makes this easy with the sample function. We are choosing to hold out 25% of our data for testing.

```
sample <- sample.int(n = nrow(data_cleaned), size = floor(.75*nrow(data_cleaned))
train <- data_cleaned[sample, ]
test  <- data_cleaned[-sample, ]
```

Random Forest Package Exploration

Fitting the data

After loading in `randomForest`, we separate our data into X and Y vectors, then feed them to the randomForest package to get a fitted model.

```
library(randomForest)
```

```
randomForest 4.7-1.2

Type rfNews() to see new features/changes/bug fixes.


Attaching package: 'randomForest'

The following object is masked from 'package:ggplot2':

    margin

The following object is masked from 'package:dplyr':

    combine
```

```r
trainX <- train[, !(names(train) %in% c('num'))]
trainY <- train$num

testX <- test[, !(names(test) %in% c('num'))]
testY <- test$num


# num is what we are trying to predict
# 0 is no heart disease, 1-4 is severity
train <- cbind(trainX, num = trainY)
fit <- randomForest(num ~ ., data = train,
                    ntree = 100, # Number of trees
                    mtry = 10, # Number of variables sampled as candidates per tr
                    maxnodes = 7, # Max # of terminal nodes (limit tree growth)
                    nodesize = 0.01 * nrow(train)) # Stopping criterion limiting
```

Generating predictions

Generating predictions with `randomForest` is straightforward - just call `predict` on the fitted model, and feed in the X vector to predict for.

```r
pred <- predict(fit, testX)
test <- cbind(testX, pred = pred)
```

Analyzing performance

The `randomForest` package has some helpful functions to analyze performance right out of the box. The `varImpPlot` function automatically generates a visualization of variable importance by MeanDecreaseGini.

```
varImpPlot(fit)
```

fit
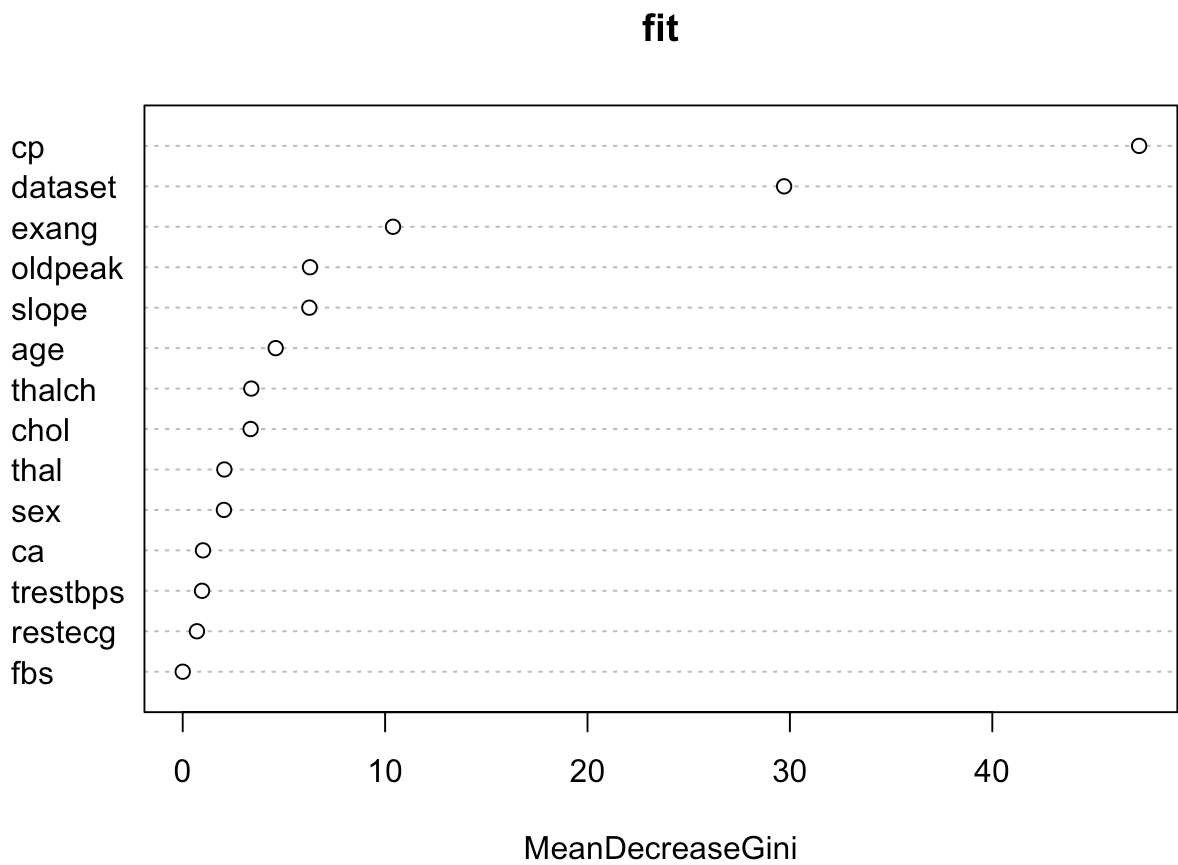
MeanDecreaseGini

The `importance` function returns a table version of variable importance by MeanDecreaseGini.

```
importance(fit)
```

```
         MeanDecreaseGini
age            4.5934941
sex            2.0348336
dataset       29.7134277
cp            47.2537144
trestbps       0.9548551
chol           3.3548672
fbs            0.0000000
restecg        0.7016639
thalch         3.3883495
exang         10.3906070
oldpeak        6.2891274
slope          6.2558574
```

```
ca              1.0011488
thal            2.0564746
```

With the help of the `caret` library, we can get additional helpful information like the confusion matrix, which breaks down our performance by class.

```
library(caret)
```

Loading required package: lattice

Attaching package: 'caret'

The following object is masked from 'package:purrr':

    lift

```
confusionMatrix(table(pred,testY))
```

Confusion Matrix and Statistics

```
    testY
pred  0  1  2  3  4
   0 91 18  3  3  0
   1 21 52 18 16  8
   2  0  0  0  0  0
   3  0  0  0  0  0
   4  0  0  0  0  0
```

Overall Statistics

```
               Accuracy : 0.6217
                 95% CI : (0.5556, 0.6847)
    No Information Rate : 0.487
    P-Value [Acc > NIR] : 2.703e-05

                  Kappa : 0.3741

 Mcnemar's Test P-Value : NA
```

Statistics by Class:

	Class: 0	Class: 1	Class: 2	Class: 3	Class: 4
Sensitivity	0.8125	0.7429	0.0000	0.00000	0.00000
Specificity	0.7966	0.6062	1.0000	1.00000	1.00000
Pos Pred Value	0.7913	0.4522	NaN	NaN	NaN
Neg Pred Value	0.8174	0.8435	0.9087	0.91739	0.96522
Prevalence	0.4870	0.3043	0.0913	0.08261	0.03478
Detection Rate	0.3957	0.2261	0.0000	0.00000	0.00000

```
Detection Prevalence    0.5000    0.5000    0.0000    0.00000    0.00000
Balanced Accuracy       0.8046    0.6746    0.5000    0.50000    0.50000
```

Summary

The `randomForest` library in R is straightforward to use, and provides a powerful out of the box solution for classification problems in R. The package allows us to adjust parameters like the number of trees, node size, max tree depth, and number of variables used in each tree. It also includes helpful functions for feature analysis and performance evaluation. Applying it to the UCI Heart Disease dataset, we were able to get a meaningful accuracy when detecting absence of heart disease (Class 0), and a relatively high accuracy overall.