Chapter 17
Variable/Feature Selection

As we mentioned in Chap. 16, variable selection is very important when dealing with bioinformatics, healthcare, and biomedical data, where we may have more features than observations. Variable selection, or feature selection, can help us focus only on the core important information contained in the observations, instead of every piece of information. Due to presence of intrinsic and extrinsic noise, the volume and complexity of big health data, and different methodological and technological challenges, this process of identifying the salient features may resemble finding a needle in a haystack. Here, we will illustrate alternative strategies for feature selection using filtering (e.g., correlation-based feature selection), wrapping (e.g., recursive feature elimination), and embedding (e.g., variable importance via random forest classification) techniques.

The next Chap. 18, provides the details about another powerful technique for variable-selection using *decoy features* to control the false discovery rate of choosing inconsequential features.

17.1 Feature Selection Methods

There are three major classes of variable or feature selection techniques—filtering-based, wrapper-based, and embedded methods.

17.1.1 Filtering Techniques

- *Univariate*: Univariate filtering methods focus on selecting single features with high scores based on some statistics like χ^2 or Information Gain Ratio. Each

I. D. Dinov, *Data Science and Predictive Analytics*,
https://doi.org/10.1007/978-3-319-72347-1_17

feature is viewed as independent of the others, effectively ignoring interactions between features.

  - Examples: χ^2, Euclidean distance, *i*-test, and Information gain.

- *Multivariate*: Multivariate filtering methods rely on various (multivariate) statistics to select the principal features. They typically account for between-feature interactions by using higher-order statistics like correlation. The basic idea is that we iteratively triage variables that have high correlations with other features.

  - Examples: Correlation-based feature selection, Markov blanket filter, and fast correlation-based feature selection.

17.1.2 Wrapper Methods

- *Deterministic*: Deterministic wrapper feature selection methods either start with no features (forward-selection) or with all features included in the model (backward-selection) and iteratively refine the set of chosen features according to some model quality measures. The iterative process of adding or removing features may rely on statistics like the Jaccard similarity coefficient.

  - Examples: Sequential forward selection, Recursive Feature Elimination, Plus q take-away r, and Beam search.

- *Randomized*: Stochastic wrapper feature selection procedures utilize a binary feature-indexing vector indicating whether or not each variable should be included in the list of salient features. At each iteration, we *randomly* perturb the binary indicators vector and compare the combinations of features before and after the random inclusion-exclusion indexing change. Finally, we pick the indexing vector corresponding with the optimal performance based on some metric, like acceptance probability measures. The iterative process continues until no improvement of the objective function is observed.

  - Examples: Simulated annealing, genetic algorithms, distribution- and kernel-estimation algorithms.

17.1.3 Embedded Techniques

- Embedded-feature selection techniques are based on various classifiers, predictors, or clustering procedures. For instance, we can accomplish feature selection by using decision trees where the separation of the training data relies on features associated with the highest information gain. Further tree branching, separating the data deeper, may utilize *weaker* features. This process of choosing the vital features based on their separability characteristics continues until the classifier generates group labels that are mostly homogeneous within clusters/classes and largely heterogeneous across groups, and when the information gain of further tree branching is marginal. The entire process may be iterated multiple times to select the features that appear most frequently.

- Examples: Decision trees, random forests, weighted naive Bayes, and feature selection using weighted-SVM.

The different types of feature selection methods have their own pros and cons. In this chapter, we are going to introduce the randomized wrapper method using the `Boruta` package, which utilizes the random forest classification method to output variable importance measures (VIMs). Then, we will compare its results with Recursive Feature Elimination, a classical deterministic wrapper method.

17.2 Case Study: ALS

17.2.1 Step 1: Collecting Data

First things first, let's explore the dataset we will be using. Case Study 15, Amyotrophic Lateral Sclerosis (ALS), examines the patterns, symmetries, associations and causality in a rare but devastating disease, amyotrophic lateral sclerosis (ALS), also known as *Lou Gehrig disease*. This ALS case-study reflects a large clinical trial including big, multi-source and heterogeneous datasets. It would be interesting to interrogate the data and attempt to derive potential biomarkers that can be used for detecting, prognosticating, and forecasting the progression of this neurodegenerative disorder. Overcoming many scientific, technical and infrastructure barriers is required to establish complete, efficient, and reproducible protocols for such complex data. These pipeline workflows start with ingesting the raw data, preprocessing, aggregating, harmonizing, analyzing, visualizing and interpreting the findings.

In this case-study, we use the training dataset that contains 2223 observations and 131 numeric variables. We select `ALSFRS slope` as our outcome variable, as it captures the patients' clinical decline over a year. Although we have more observations than features, this is one of the examples where multiple features are highly correlated. Therefore, we need to preprocess the variables, e.g., apply feature selection, before commencing with predictive analytics.

17.2.2 Step 2: Exploring and Preparing the Data

The dataset is located in our case-studies archive. We can use `read.csv()` to directly import the CSV dataset into R using the URL reference.

```
ALS.train<-read.csv("https://umich.instructure.com/files/1789624/download?do
wnload_frd=1")
summary(ALS.train)
```

```
##        ID             Age_mean      Albumin_max    Albumin_median
##  Min.   :   1.0   Min.   :18.00   Min.   :37.00   Min.   :34.50
##  1st Qu.: 614.5   1st Qu.:47.00   1st Qu.:45.00   1st Qu.:42.00
##  Median :1213.0   Median :55.00   Median :47.00   Median :44.00
##  Mean   :1214.9   Mean   :54.55   Mean   :47.01   Mean   :43.95
##  3rd Qu.:1815.5   3rd Qu.:63.00   3rd Qu.:49.00   3rd Qu.:46.00
##  Max.   :2424.0   Max.   :81.00   Max.   :70.30   Max.   :51.10
…
##  Urine.Ph_median  Urine.Ph_min
##  Min.   :5.000   Min.   :5.000
##  1st Qu.:5.000   1st Qu.:5.000
##  Median :6.000   Median :5.000
##  Mean   :5.711   Mean   :5.183
##  3rd Qu.:6.000   3rd Qu.:5.000
##  Max.   :9.000   Max.   :8.000
```

There are 131 features and some of variables represent statistics like *max*, *min* and *median* values of the same clinical measurements.

17.2.3 Step 3: Training a Model on the Data

Now let's explore the `Boruta()` function in the `Boruta` package to perform variables selection, based on random forest classification. `Boruta()` includes the following components:

```
vs<-Boruta(class~features, data=Mydata, pValue = 0.01, mcAdj =
TRUE, maxRuns = 100, doTrace=0, getImp = getImpRfZ, ...)
```

- `class`: variable for class labels.
- `features`: potential features to select from.
- `data`: dataset containing classes and features.
- `pValue`: confidence level. Default value is 0.01 (Notice we are applying multiple variable selection.
- `mcAdj`: Default TRUE to apply a multiple comparisons adjustment using the Bonferroni method.
- `maxRuns`: maximal number of importance source runs. You may increase it to resolve attributes left Tentative.
- `doTrace`: verbosity level. Default 0 means no tracing, 1 means reporting decision about each attribute as soon as it is justified, 2 means same as 1, plus at each importance source run reporting the number of attributes. The default is 0 where we don't do the reporting.

- `getImp`: function used to obtain attribute importance. The default is *getImpRfZ*, which runs random forest, from the ranger package, and gathers *Z*-scores of the mean decreased accuracy measure.

The resulting `vs` object is of class `Boruta` and contains two important components:

- `finalDecision`: a factor of three values: `Confirmed`, `Rejected` or `Tentative`, containing the final results of the feature selection process.
- `ImpHistory`: a data frame of importance of attributes gathered in each importance source run. Besides the predictors' importance, it contains maximal, mean and minimal importance of shadow attributes for each run. Rejected attributes get `-Inf` importance. This output is set to NULL if we specify `holdHistory=FALSE` in the Boruta call.

Note: Running the code below will take several minutes.

```
# install.packages("Boruta")
library(Boruta)
set.seed(123)
als<-Boruta(ALSFRS_slope~.-ID, data=ALS.train, doTrace=0)
print(als)

## Boruta performed 99 iterations in 4.683657 mins.
##  28 attributes confirmed important: ALSFRS_Total_max,
## ALSFRS_Total_median, ALSFRS_Total_min, ALSFRS_Total_range,
## Creatinine_median and 23 more;
##  59 attributes confirmed unimportant: Albumin_max, Albumin_median,
## Albumin_min, ALT.SGPT._max, ALT.SGPT._median and 54 more;
##  12 tentative attributes left: Age_mean, Albumin_range,
## Creatinine_max, Hematocrit_median, Hematocrit_range and 7 more;

als$ImpHistory[1:6, 1:10]

##         Age_mean Albumin_max Albumin_median Albumin_min Albumin_range
## [1,]  1.2031427   1.4969268      0.6976378   0.9385041      1.979510
## [2,] -0.1998469   0.7204092     -1.5626360   0.5777092      2.573882
## [3,]  1.9272058  -1.0274668      0.2216170  -1.2234402      1.843967
## [4,]  0.5763244   0.9097371      0.2960979   0.6137624      2.184383
## [5,]  3.3655147   1.9412326      0.3849548   1.7309793      1.134676
## [6,]  0.2603118  -0.0287943      1.4164860   2.3251879      2.259974
##      ALSFRS_Total_max ALSFRS_Total_median ALSFRS_Total_min
## [1,]         6.925233            9.551064         15.92924
## [2,]         8.124101            7.867399         14.94650
## [3,]         7.443326            8.735702         17.26469
## [4,]         7.578267            7.868885         16.95563
## [5,]         7.554582            7.248834         15.42697
## [6,]         7.516362            7.145460         14.94824
##      ALSFRS_Total_range ALT.SGPT._max
## [1,]           25.78135     4.1516252
## [2,]           26.11722     1.2187027
## [3,]           25.61523     2.1618804
## [4,]           28.19229     0.4305607
## [5,]           24.90620     1.2043325
## [6,]           26.57093     0.8463782
```

This is a fairly time-consuming computation. Boruta determines the *important* attributes from *unimportant* and *tentative* features. Here the importance is measured by the out-of-bag (OOB) error. The OOB estimates the prediction error of machine-learning methods (e.g., random forests and boosted decision trees) that utilize bootstrap aggregation to sub-sample training data. **OOB** represents the mean prediction error on each training sample x_i, using only the trees that did not include x_i in their bootstrap samples. Out-of-bag estimates provide *internal* assessment of the learning accuracy and avoid the need for an independent *external* validation dataset.

The importance scores for all features at every iteration are stored in the data frame `als$ImpHistory`. Let's plot a graph depicting the essential features.

Note: Again, running this code will take several minutes to complete (Fig. 17.1).

```
plot(als, xlab="", xaxt="n")
lz<-lapply(1:ncol(als$ImpHistory), function(i)
als$ImpHistory[is.finite(als$ImpHistory[, i]), i])
names(lz)<-colnames(als$ImpHistory)
lb<-sort(sapply(lz, median))
axis(side=1, las=2, labels=names(lb), at=1:ncol(als$ImpHistory),
cex.axis=0.5, font =4)
```

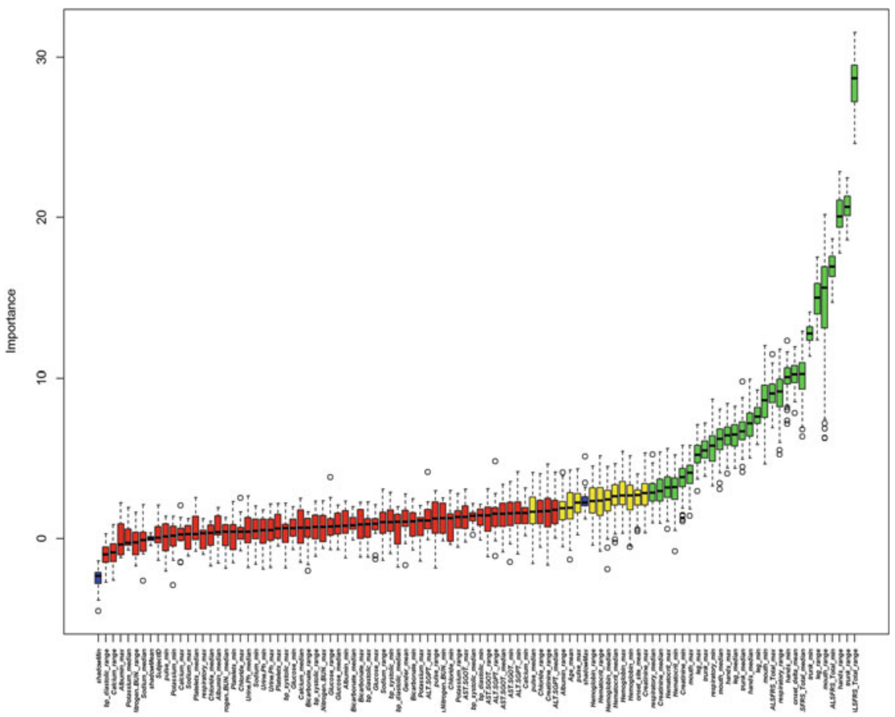

Fig. 17.1 Ranked variables importance using box and whisker plots for each feature

We can see that plotting the graph is easy but extracting matched feature names may require more work. The basic plot is done by this call `plot(als, xlab="", xaxt="n")`, where `xaxt="n"` means we suppress plotting of *x*-axis. The following lines in the script reconstruct the *x*-axis plot. `lz` is a list created by the `lapply()` function. Each element in `lz` contains all the important scores for a single feature in the original dataset. Also, we excluded all rejected features with infinite importance. Then, we sorted these non-rejected features according to their median importance and printed them on the *x*-axis by using `axis()`.

We have already seen similar groups of boxplots back in Chaps. 3 and 4. In this graph, variables with *green* boxes are more important than the ones represented with *red* boxes, and we can see the range of importance scores within a single variable in the graph.

It may be desirable to get rid of tentative features. Notice that this function should be used only when strict decision is highly desired, because this test is much weaker than Boruta and can lower the confidence of the final result.

```
final.als<-TentativeRoughFix(als)
print(final.als)

## Boruta performed 99 iterations in 4.683657 mins.
## Tentatives roughfixed over the last 99 iterations.
##  32 attributes confirmed important: ALSFRS_Total_max,
## ALSFRS_Total_median, ALSFRS_Total_min, ALSFRS_Total_range,
## Creatinine_median and 27 more;
##  67 attributes confirmed unimportant: Age_mean, Albumin_max,
## Albumin_median, Albumin_min, Albumin_range and 62 more;

final.als$finalDecision

##                        Age_mean                    Albumin_max
##                        Rejected                       Rejected
##                  Albumin_median                    Albumin_min
##                        Rejected                       Rejected
##                   Albumin_range               ALSFRS_Total_max
##                        Rejected                      Confirmed
##             ALSFRS_Total_median               ALSFRS_Total_min
##                       Confirmed                      Confirmed
…
##                    Urine.Ph_max                Urine.Ph_median
##                        Rejected                       Rejected
##                    Urine.Ph_min
##                        Rejected
## Levels: Tentative Confirmed Rejected

getConfirmedFormula(final.als)

## ALSFRS_slope ~ ALSFRS_Total_max+ALSFRS_Total_median + ALSFRS_Total_min +
##     ALSFRS_Total_range + Creatinine_median + Creatinine_min +
##     hands_max + hands_median + hands_min + hands_range+Hematocrit_max+
##     Hematocrit_min+Hematocrit_range+Hemoglobin_median+Hemoglobin_range +
##     leg_max + leg_median + leg_min + leg_range + mouth_max +
##     mouth_median + mouth_min + mouth_range + onset_delta_mean +
##     pulse_max+respiratory_median + respiratory_min + respiratory_range+
##     trunk_max + trunk_median + trunk_min + trunk_range
## <environment: 0x000000000989d6f8>
```

```
# report the Boruta "Confirmed" & "Tentative" features, removing the
"Rejected" ones
print(final.als$finalDecision[final.als$finalDecision %in% c("Confirmed",
"Tentative")])

##    ALSFRS_Total_max ALSFRS_Total_median    ALSFRS_Total_min
##           Confirmed           Confirmed           Confirmed
##  ALSFRS_Total_range   Creatinine_median      Creatinine_min
##           Confirmed           Confirmed           Confirmed
##           hands_max        hands_median           hands_min
##           Confirmed           Confirmed           Confirmed
##         hands_range      Hematocrit_max      Hematocrit_min
##           Confirmed           Confirmed           Confirmed
##    Hematocrit_range   Hemoglobin_median    Hemoglobin_range
##           Confirmed           Confirmed           Confirmed
##             leg_max          leg_median             leg_min
##           Confirmed           Confirmed           Confirmed
##           leg_range           mouth_max        mouth_median
##           Confirmed           Confirmed           Confirmed
##           mouth_min         mouth_range    onset_delta_mean
##           Confirmed           Confirmed           Confirmed
##           pulse_max  respiratory_median     respiratory_min
##           Confirmed           Confirmed           Confirmed
##   respiratory_range           trunk_max        trunk_median
##           Confirmed           Confirmed           Confirmed
##           trunk_min         trunk_range
##           Confirmed           Confirmed
## Levels: Tentative Confirmed Rejected

# how many are actually "confirmed" as important/salient?
impBoruta <- final.als$finalDecision[final.als$finalDecision %in%
c("Confirmed")]; length(impBoruta)

## [1] 32
```

The report above shows the final features selection including only the “confirmed” and “Tentative” features.

17.2.4 Step 4: Evaluating Model Performance

Comparing with RFE

Let’s compare the `Boruta` results against a classical variable selection method—*recursive feature elimination (RFE)*. First, we need to load two packages: `caret` and `randomForest`. Then, as we did in Chap. 15, we must specify a resampling method. Here we use *10-fold CV* to do the resampling.

```
library(caret)

library(randomForest)

set.seed(123)
control<-rfeControl(functions = rfFuncs, method = "cv", number=10)
```

Now, all preparations are complete and we are ready to do the RFE variable selection.

```
rf.train<-rfe(ALS.train[, -c(1, 7)], ALS.train[, 7], sizes=c(10,20,30,40),
rfeControl=control)
rf.train

## Recursive feature selection
## Outer resampling method: Cross-Validated (10 fold)
## Resampling performance over subset size:
##  Variables   RMSE Rsquared  RMSESD RsquaredSD Selected
##         10 0.3500   0.6837 0.03451    0.03837
##         20 0.3471   0.6894 0.03230    0.03374
##         30 0.3468   0.6900 0.03135    0.02967        *
##         40 0.3473   0.6895 0.03061    0.02887
##         99 0.3503   0.6842 0.02995    0.02868

## The top 5 variables (out of 30):
## ALSFRS_Total_range,trunk_range,hands_range,mouth_range,ALSFRS_Total_min
```

This calculation may take a long time to complete. The RFE invocation is different from `Boruta`. Here we have to specify the feature data frame and the class labels separately. Also, the `sizes=` option allows us to specify the number of features we want to include in the model. Let's try `sizes=c(10, 20, 30, 40)` to compare the model performance for alternative numbers of features.

To visualize the results, we can plot the RMSE error for the five different feature size combinations listed in the summary. The one with 30 features has the lowest RMSE value. This result is similar to the `Boruta` output, which selected around 30 features (Fig. 17.2).

```
plot(rf.train, type=c("g", "o"), cex=1, col=1:5)
```

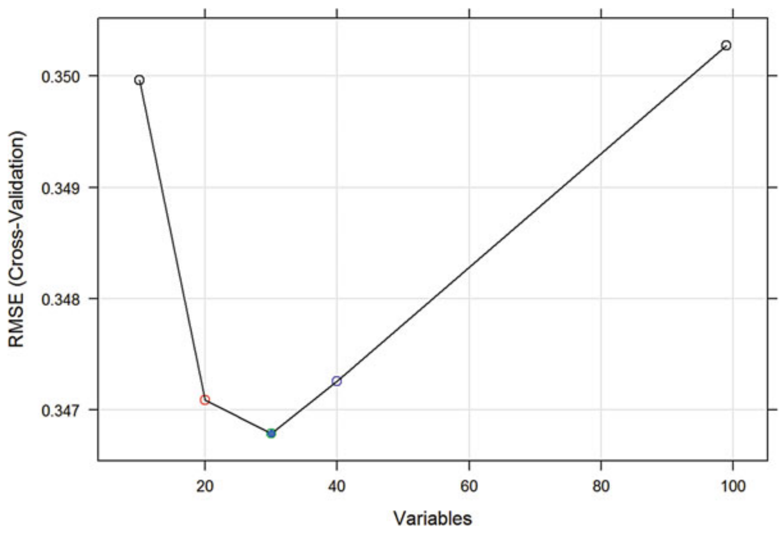

Fig. 17.2 Root-mean square cross-validation error rate for random forest classification of the ALS study against the number of features

Using the functions predictors() and getSelectedAttributes(), we can compare the final results of the two alternative feature selection methods.

```
predRFE <- predictors(rf.train)
predBoruta <- getSelectedAttributes(final.als, withTentative = F)
```

The Boruta and RFE feature-selction results are almost identical.

```
intersect(predBoruta, predRFE)
##  [1] "ALSFRS_Total_max"    "ALSFRS_Total_median" "ALSFRS_Total_min"
##  [4] "ALSFRS_Total_range"  "Creatinine_min"      "hands_max"
##  [7] "hands_median"        "hands_min"           "hands_range"
## [10] "Hematocrit_max"      "Hemoglobin_median"   "leg_max"
## [13] "leg_median"          "leg_min"             "leg_range"
## [16] "mouth_median"        "mouth_min"           "mouth_range"
## [19] "onset_delta_mean"    "respiratory_median"  "respiratory_min"
## [22] "respiratory_range"   "trunk_max"           "trunk_median"
## [25] "trunk_min"           "trunk_range"
```

There are 26 common variables chosen by the two techniques, which suggests that both the Boruta and RFE methods are robust. Also, notice that the Boruta method can give similar results without utilizing the *size* option. If we want to consider ten or more different sizes, the procedure will be quite time consuming. Thus, the Boruta method is effective when dealing with complex real world problems.

Comparing with Stepwise Feature Selection

Next, we can contrast the Boruta feature selection results against another classical variable selection method – *stepwise model selection*. Let's start with fitting a bidirectional stepwise linear model-based feature selection.

```
data2 <- ALS.train[, -1]
# Define a base model - intercept only
base.mod <- lm(ALSFRS_slope ~ 1 , data= data2)
# Define the full model - including all predictors
all.mod <- lm(ALSFRS_slope ~ . , data= data2)
# ols_step <- lm(ALSFRS_slope ~ ., data=data2)
ols_step <- step(base.mod, scope = list(lower=base.mod, upper = all.mod),
direction = 'both', k=2, trace = F)
summary(ols_step); ols_step

## Call:
## lm(formula = ALSFRS_slope ~ ALSFRS_Total_range + ALSFRS_Total_median +
##     ALSFRS_Total_min + Calcium_range + Calcium_max + bp_diastolic_min +
##     onset_delta_mean + Calcium_min + Albumin_range + Glucose_range +
##     ALT.SGPT._median + AST.SGOT._median + Glucose_max + Glucose_min +
##     Creatinine_range + Potassium_range + Chloride_range + Chloride_min+
##     Sodium_median + respiratory_min +respiratory_range+respiratory_max+
##     trunk_range + pulse_range + Bicarbonate_max + Bicarbonate_range +
##     Chloride_max + onset_site_mean + trunk_max + Gender_mean +
##     Creatinine_min, data = data2)
```

```
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -2.22558 -0.17875 -0.02024  0.17098  1.95100
##
## Coefficients:
##                       Estimate Std. Error t value Pr(>|t|)
## (Intercept)          4.176e-01  6.064e-01   0.689 0.491091
## ALSFRS_Total_range  -2.260e+01  1.359e+00 -16.631  < 2e-16 ***
## ALSFRS_Total_median -3.388e-02  2.868e-03 -11.812  < 2e-16 ***
## ALSFRS_Total_min     2.821e-02  3.310e-03   8.524  < 2e-16 ***
…

## trunk_max            2.288e-02  8.453e-03   2.706 0.006854 **
## Gender_mean         -3.360e-02  1.751e-02  -1.919 0.055066 .
## Creatinine_min       7.643e-04  4.977e-04   1.536 0.124771
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.3355 on 2191 degrees of freedom
## Multiple R-squared:  0.7135, Adjusted R-squared:  0.7094
## F-statistic:   176 on 31 and 2191 DF,  p-value: < 2.2e-16

##
## Call:
## lm(formula = ALSFRS_slope ~ ALSFRS_Total_range + ALSFRS_Total_median +
##     ALSFRS_Total_min + Calcium_range + Calcium_max + bp_diastolic_min +
##     onset_delta_mean + Calcium_min + Albumin_range + Glucose_range +
##     ALT.SGPT._median + AST.SGOT._median + Glucose_max + Glucose_min +
##     Creatinine_range + Potassium_range + Chloride_range +Chloride_min+
##     Sodium_median + respiratory_min+respiratory_range+respiratory_max+
##     trunk_range + pulse_range + Bicarbonate_max + Bicarbonate_range +
##     Chloride_max + onset_site_mean + trunk_max + Gender_mean +
##     Creatinine_min, data = data2)
##
## Coefficients:
##         (Intercept)   ALSFRS_Total_range  ALSFRS_Total_median
##           4.176e-01           -2.260e+01           -3.388e-02
##    ALSFRS_Total_min        Calcium_range          Calcium_max
##           2.821e-02            2.410e+02           -4.258e-01
##    bp_diastolic_min     onset_delta_mean          Calcium_min
##          -2.249e-03           -5.461e-05            3.579e-01
##       Albumin_range        Glucose_range     ALT.SGPT._median
##          -2.305e+00           -1.510e+01           -2.300e-03
##    AST.SGOT._median          Glucose_max          Glucose_min
##           3.369e-03            3.279e-02           -3.507e-02
##    Creatinine_range      Potassium_range       Chloride_range
##           5.076e-01           -4.535e+00            5.318e+00
##        Chloride_min        Sodium_median      respiratory_min
##           1.672e-02           -9.830e-03           -1.453e-01
##   respiratory_range      respiratory_max          trunk_range
##          -5.834e+01            1.712e-01           -8.705e+00
##         pulse_range      Bicarbonate_max    Bicarbonate_range
##          -5.117e-01            7.526e-03           -2.204e+00
##        Chloride_max      onset_site_mean            trunk_max
##          -6.918e-03            3.359e-02            2.288e-02
##         Gender_mean       Creatinine_min
##          -3.360e-02            7.643e-04
```

We can report the stepwise "Confirmed" (salient) features:

```
# get the shortlisted variable
stepwiseConfirmedVars <- names(unlist(ols_step[[1]]))
# remove the intercept
stepwiseConfirmedVars <- stepwiseConfirmedVars[!stepwiseConfirmedVars %in% "
(Intercept)"]
print(stepwiseConfirmedVars)
```

```
##  [1] "ALSFRS_Total_range"  "ALSFRS_Total_median" "ALSFRS_Total_min"
##  [4] "Calcium_range"       "Calcium_max"         "bp_diastolic_min"
##  [7] "onset_delta_mean"    "Calcium_min"         "Albumin_range"
## [10] "Glucose_range"       "ALT.SGPT._median"    "AST.SGOT._median"
## [13] "Glucose_max"         "Glucose_min"         "Creatinine_range"
## [16] "Potassium_range"     "Chloride_range"      "Chloride_min"
## [19] "Sodium_median"       "respiratory_min"     "respiratory_range"
## [22] "respiratory_max"     "trunk_range"         "pulse_range"
## [25] "Bicarbonate_max"     "Bicarbonate_range"   "Chloride_max"
## [28] "onset_site_mean"     "trunk_max"           "Gender_mean"
## [31] "Creatinine_min"
```

Again, the feature selection results of `Boruta` and `step` are similar.

```
library(mlbench)
library(caret)

# estimate variable importance
predStepwise <- varImp(ols_step, scale=FALSE)
# summarize importance
print(predStepwise)
```

```
##                        Overall
## ALSFRS_Total_range   16.630592
## ALSFRS_Total_median 11.812263
## ALSFRS_Total_min      8.523606
## Calcium_range         5.754045
## Calcium_max           4.812942
## bp_diastolic_min      2.539766
## onset_delta_mean      2.758465
## Calcium_min           3.767450
## Albumin_range         2.812018
## Glucose_range         5.156259
## ALT.SGPT._median      2.876338
## AST.SGOT._median      2.641369
## Glucose_max           4.629759
## Glucose_min           4.022642
## Creatinine_range      2.293301
## Potassium_range       1.739268
## Chloride_range        4.474709
## Chloride_min          4.403551
## Sodium_median         2.118710
## respiratory_min       5.948488
## respiratory_range     5.756735
## respiratory_max       5.041816
## trunk_range           2.819029
## pulse_range           1.696811
## Bicarbonate_max      2.568068
## Bicarbonate_range    2.303757
## Chloride_max         1.750666
## onset_site_mean      1.663481
## trunk_max            2.706410
## Gender_mean          1.919380
## Creatinine_min       1.535642

# plot predStepwise
# plot(predStepwise)

# Boruta vs. Stepwise feataure selection
intersect(predBoruta, stepwiseConfirmedVars)

## [1] "ALSFRS_Total_median" "ALSFRS_Total_min"    "ALSFRS_Total_range"
## [4] "Creatinine_min"      "onset_delta_mean"    "respiratory_min"
## [7] "respiratory_range"   "trunk_max"           "trunk_range"
```

There are about nine common variables chosen by the Boruta and Stepwise feature selection methods.

There is another more elaborate stepwise feature selection technique that is implemented in the function `MASS::stepAIC()` that is useful for a wider range of object classes.

17.3 Practice Problem

You can practice variable selection using the SOCR_Data_AD_BiomedBigMetadata on the SOCR website. This is a smaller dataset that has 744 observations and 63 variables. Here we utilize `DXCURREN` or current diagnostics as the class variable.

Let's import the dataset first.

```
library(rvest)

wiki_url <- read_html("http://wiki.socr.umich.edu/index.php/SOCR_Data_AD_Bio
medBigMetadata")
html_nodes(wiki_url, "#content")

## {xml_nodeset (1)}
## [1] <div id="content" class="mw-body-primary" role="main">\n\t<a id="top
...

alzh <- html_table(html_nodes(wiki_url, "table")[[1]])
summary(alzh)

##       SID              MMSCORE          FAQTOTAL             GDTOTAL
##  Min.   :   2.0   Min.   :18.00   Length:744         Min.   :0.000
##  1st Qu.: 355.5   1st Qu.:25.00   Class :character   1st Qu.:0.000
##  Median : 697.5   Median :27.00   Mode  :character   Median :1.000
##  Mean   : 707.5   Mean   :26.81                      Mean   :1.367
##  3rd Qu.:1063.0   3rd Qu.:29.00                      3rd Qu.:2.000
##  Max.   :1435.0   Max.   :30.00                      Max.   :6.000
```

```
…
##      CDHOME            CDCARE           CDGLOBAL
##  Min.   :0.0000   Min.   :0.0000   Min.   :0.0000
##  1st Qu.:0.0000   1st Qu.:0.0000   1st Qu.:0.0000
##  Median :0.0000   Median :0.0000   Median :0.0000
##  Mean   :0.2513   Mean   :0.2849   Mean   :0.0672
##  3rd Qu.:0.5000   3rd Qu.:0.5000   3rd Qu.:0.0000
##  Max.   :2.0000   Max.   :2.0000   Max.   :2.0000
```

The data summary shows that we have several factor variables. After converting their type to numeric we find some missing data. We can manage this issue by selecting only the complete observation of the original dataset or by using multivariate imputation, see Chap. 3.

```
chrtofactor<-c(3, 5, 8, 10, 21:22, 51:54)
alzh[chrtofactor]<-data.frame(apply(alzh[chrtofactor], 2, as.numeric))

alzh<-alzh[complete.cases(alzh), ]
```

For simplicity, here we eliminated the missing data and are left with 408 complete observations. Now, we can apply the `Boruta` method for feature selection.

```
## Boruta performed 99 iterations in 9.413648 secs.
##  12 attributes confirmed important: adascog, BCBREATH, CDCARE,
## CDCOMMUN, CDGLOBAL and 7 more;
##  47 attributes confirmed unimportant: Age, BC.USEA, BCABDOMN,
## BCANKLE, BCCHEST and 42 more;
##  2 tentative attributes left: ApoEGeneAllele1, ApoEGeneAllele2;
```

You might get a result that is a little bit different. We can plot the variable importance graph using some previous knowledge (Fig. 17.3).

The final step is to get rid of the tentative features.

```
## Boruta performed 99 iterations in 9.413648 secs.
## Tentatives roughfixed over the last 99 iterations.
##  14 attributes confirmed important: adascog, ApoEGeneAllele1,
## ApoEGeneAllele2, BCBREATH, CDCARE and 9 more;
##  47 attributes confirmed unimportant: Age, BC.USEA, BCABDOMN,
## BCANKLE, BCCHEST and 42 more;
##  [1]  "MMSCORE"         "FAQTOTAL"        "adascog"
##  [4]  "sobcdr"          "DX_Confidence"    "BCBREATH"
##  [7]  "ApoEGeneAllele1" "ApoEGeneAllele2" "CDORIENT"
##  [10] "CDJUDGE"         "CDCOMMUN"         "CDHOME"
##  [13] "CDCARE"          "CDGLOBAL"
```

Can you reproduce these results? Also try to apply some of these techniques to other data from the list of our Case-Studies.

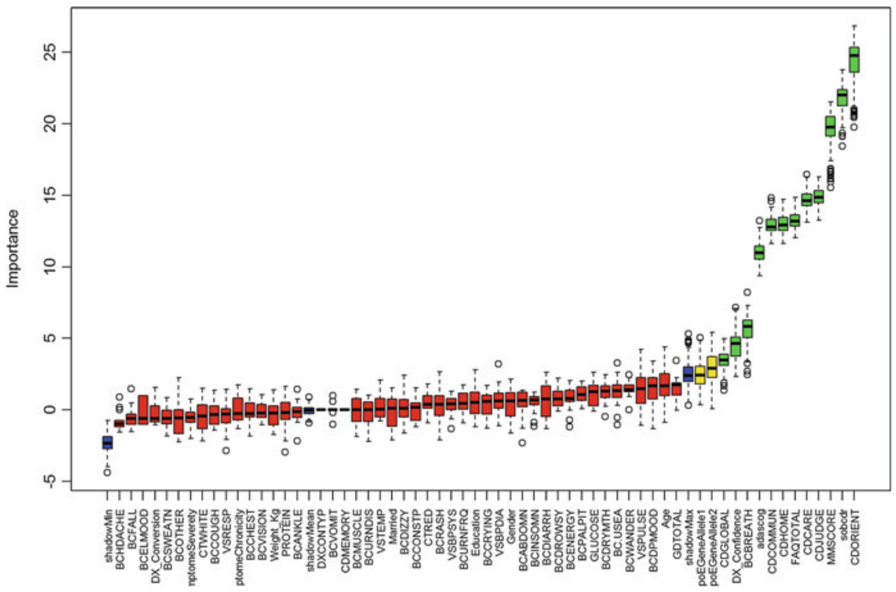

Fig. 17.3 Variable importance plot of predicting diagnosis for the Alzheimer's disease case-study

17.4 Assignment: 17. Variable/Feature Selection

17.4.1 Wrapper Feature Selection

- Explain the three major types of feature selection methods
  - Filter,
  - Wrapper, and
  - Embedded.

17.4.2 Use the PPMI Dataset

Use the 06_PPMI_ClassificationValidationData_Short dataset setting `ResearchGroup` as class variable.

- Delete irrelevant columns (e.g. `X`, `FID_IID`) and select only the *PD* and *Control* cases.
- Properly convert the variable types.
- Apply `Boruta` to train a model, try different parameters (e.g., try different `pValue`, `maxRuns`). What are the differences?
- Summarize and visualize the results.

- Apply *Random Feature Elimination* (RFE) and tune the model size.
- Evaluate the `Boruta` model performance by comparing with REF.
- Output and compare the variables selected by both methods. How much overlap is there in the selected variables?

References

Guyon, E, Gunn, S, Nikravesh, M, Zadeh, LA (eds.) (2008) *Feature Extraction: Foundations and Application*s, Springer, ISBN 3540354883, 9783540354888

Liu, H and Motoda, H (eds.) (2007) *Computational Methods of Feature Selection*, Chapman & Hall/CRC, ISBN 1584888792, 9781584888796

Pacheco, ER (2015) *Unsupervised Learning with R*, Packt Publishing, ISBN 1785885812, 9781785885815