Laboratory 6 - Extra: Statistical Methods

Classification tree and random forest

N Torelli, G Di Credico, V Gioia

21 December 2023

Contents

Example of tree-based classification

We consider an example of email spam detection. Data are available from the UCI repository of machine learning databases (http://www.ics.uci.edu/~mlearn/MLRepository.html) and they collect informations about 4601 email items, 1813 classified as spam. 57 explanatory variables describe several characteristics of the data. Following the example in the Data Analyisis and Graphics using R, we will consider only 6 of them, mostly related to the frequency of specific words and characters in the email. In detail,

- `crl.tot`: total length of words that are in capitals (column 57);
- `dollar`: the frequency of the $ symbol, as a percentage of all characters (column 53);
- `bang`: the frequency of the ! symbol, as a percentage of all characters (column 52);
- `money`: frequency of the word "money", as a percentage of all words (column 24);
- `n000`: frequency of the character string "000", as a percentage of all words (column 23);
- `make`: frequency of the word "make", as a percentage of all words (column 1).

Using these 6 explanatory variables we want to build a decision tree model which is able to classify each email correctly as spam (`y` in the binary outcome variable `yesno`, column 58) and non-spam (`n`).

```
library(reshape)
library(rpart)
library(rpart.plot)
library(rattle)
spam <-read.table("spambase.data", sep = ",")
spam <- spam[, c(58, 57, 53, 52, 24, 23, 1)]
colnames(spam) <- c("yesno", "crl.tot", "dollar", "bang", "money", "n000", "make")
spam$yesno <- factor(spam$yesno, levels = c(0, 1))
levels(spam$yesno) <- c("n", "y")
```

The model can be fitted using the function `rpart` in the library `rpart`. The option `class` in `method` is selected by default when the outcome variable is of type factor. The argument `method` allows specifying one between "anova", "poisson", "class" or "exp". In our case we must select "class" as we are in a classification problem. Instead, consider "anova" for continuous variable, while count and survival objects are dealt with "poisson" and "exp".

```
spam.rpart <- rpart(yesno ~ crl.tot + dollar + bang + money + n000 + make,
                    data = spam, method = "class")
```

By printing the fitted object on the console we can see all the information of the fitted decision tree. They differ between classification and regression, since information for each node includes

- Regression: the node number, split, size, deviance, and fitted value (* denotes terminal node);
- Classification: the node number, split, size, 0/1 losses, predicted class and predicted probabilities of the classes (* denotes terminal node);

In our example:

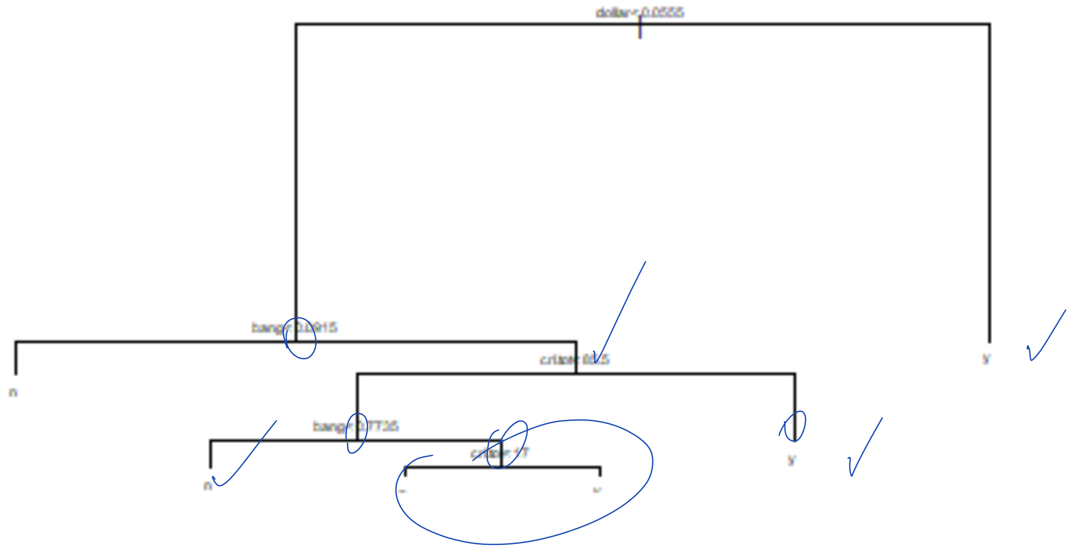
- The first row reports the number of data points (4601); the frequency of mails labeled as spam (1813); the predicted class (`y` or `n` according to the predicted probability threshold 0.5); the predicted probability of insuccess (`n`) and success (`y`)
- The other rows report similar information as the first row, except that they include the splitting rule, as:
  - dollar $<$ 0.0555 (the second row): 3471 observations having this condition; 816 of them are spam (`y`); the predicted probability is 0.235 for success (`y`) and 0.765 for insuccess (`n`), thus the predicted class is `n`;
  - dollar $>=$ 0.0555 (the last row): 1130 observations showing such condition; 133 of them are non-spam (`n`); the predicted probability is 0.118 for insuccess (`n`) and 0.882 for success (`y`), thus the predicted class is `y`; * refers to terminal node
- the following decision rule only targets the node `dollar < 0.0555` and consider the splitting according to the variable `bang` and the threshold 0.0915.
- The final decision tree is obtained by a default condition on the complexity parameter

```r
spam.rpart
```

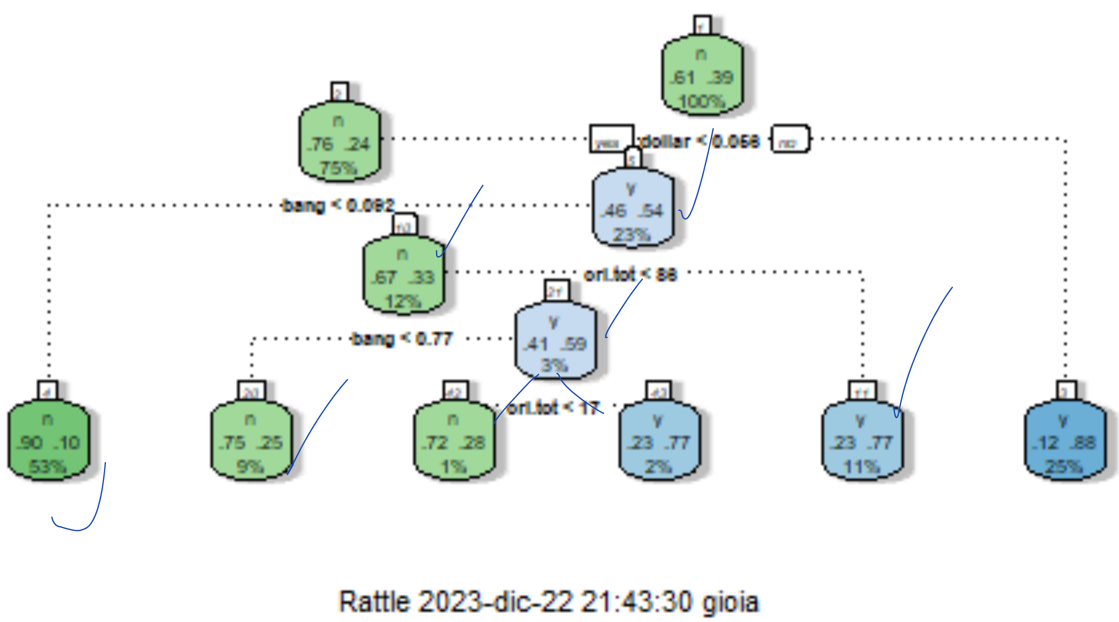
```
## n= 4601
##
## node), split, n, loss, yval, (yprob)
##       * denotes terminal node
##
##  1) root 4601 1813 n (0.6059552 0.3940448)
##    2) dollar< 0.0555 3471  816 n (0.7649092 0.2350908)
##      4) bang< 0.0915 2420  246 n (0.8983471 0.1016529) *
##      5) bang>=0.0915 1051  481 y (0.4576594 0.5423406)
##       10) crl.tot< 85.5 535  175 n (0.6728972 0.3271028)
##         20) bang< 0.7735 418  106 n (0.7464115 0.2535885) *
##         21) bang>=0.7735 117   48 y (0.4102564 0.5897436)
##           42) crl.tot< 17 43   12 n (0.7209302 0.2790698) *
##           43) crl.tot>=17 74   17 y (0.2297297 0.7702703) *
##       11) crl.tot>=85.5 516  121 y (0.2344961 0.7655039) *
##    3) dollar>=0.0555 1130  133 y (0.1176991 0.8823009) *
```

Despite it could be useful to understand the previous model output, the structure of the decision tree can be easily summarised in a plot, which retains all the information reported above. Below you can find two ways to produce a decision tree (the first one consider the `plot.rpart` of the `rpart` package; the second one leverages the `fancyRpartPlot` function of the `rattle` package).

```r
plot(spam.rpart)
text(spam.rpart, cex=.5)
```



```r
fancyRpartPlot(spam.rpart)
```



The summary of the fitted decision tree is visualized using the function `printcp`.

```r
printcp(spam.rpart)
```

```
##
## Classification tree:
## rpart(formula = yesno ~ crl.tot + dollar + bang + money + n000 +
##     make, data = spam, method = "class")
##
## Variables actually used in tree construction:
## [1] bang    crl.tot dollar
##
## Root node error: 1813/4601 = 0.39404
##
## n= 4601
##
##         CP nsplit rel error  xerror     xstd
## 1 0.476558      0   1.00000 1.00000 0.018282
## 2 0.075565      1   0.52344 0.56095 0.015525
## 3 0.011583      3   0.37231 0.39217 0.013523
## 4 0.010480      4   0.36073 0.38555 0.013429
## 5 0.010000      5   0.35025 0.38279 0.013390
```

The complexity parameter `CP` is a proxy for the number of splits. In order to avoid too complex trees, the reduction of lack-of-fit for each additional split is evaluated with an increasing cost. When the cost outweights the reduction, the algorithm stops. Since the fit via rpart consider the default parameter `cp = 0.1` the algorithm stops when such value is encountered. Small complexity parameter `CP` leads to large tree and large `CP` leads to small tree.

Let's try to decrease the default `CP` value for our model, by setting it to zero and thus obtaining a larger tree.

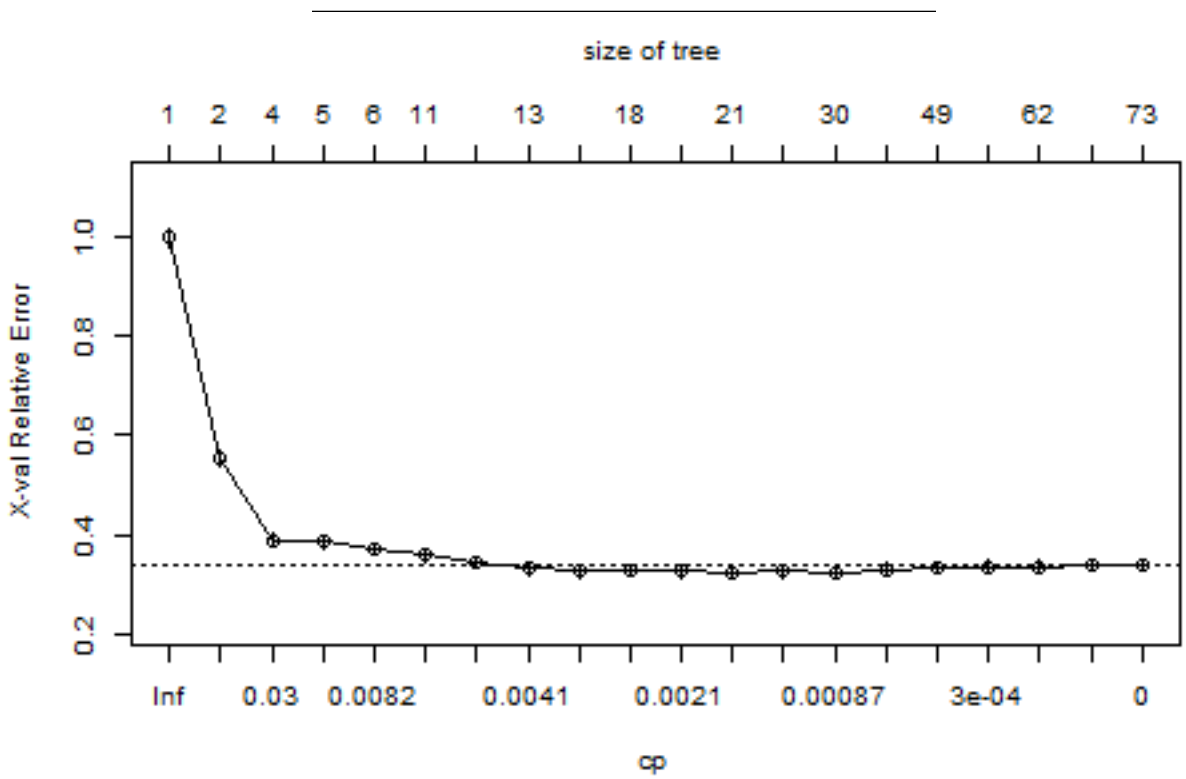
```r
spam.rpart.cp0 <- rpart(yesno ~ crl.tot + dollar + bang + money + n000 + make ,
                        data = spam, method = "class", cp=0)
printcp(spam.rpart.cp0)
```

```
##
## Classification tree:
## rpart(formula = yesno ~ crl.tot + dollar + bang + money + n000 +
##     make, data = spam, method = "class", cp = 0)
##
## Variables actually used in tree construction:
## [1] bang    crl.tot dollar  make    money   n000
##
## Root node error: 1813/4601 = 0.39404
##
## n= 4601
##
##             CP nsplit rel error  xerror     xstd
## 1  4.7656e-01      0   1.00000 1.00000 0.018282
## 2  7.5565e-02      1   0.52344 0.55323 0.015447
## 3  1.1583e-02      3   0.37231 0.38665 0.013445
## 4  1.0480e-02      4   0.36073 0.38555 0.013429
## 5  6.3431e-03      5   0.35025 0.37010 0.013205
## 6  5.5157e-03     10   0.31660 0.35852 0.013031
## 7  4.4126e-03     11   0.31109 0.34363 0.012801
## 8  3.8610e-03     12   0.30667 0.33149 0.012608
```

```
## 9  2.7579e-03     16   0.29123 0.32543 0.012509
## 10 2.2063e-03     17   0.28847 0.32653 0.012527
## 11 1.9305e-03     18   0.28627 0.32598 0.012518
## 12 1.6547e-03     20   0.28240 0.32377 0.012482
## 13 9.1929e-04     25   0.27413 0.32488 0.012500
## 14 8.2736e-04     29   0.26917 0.32322 0.012473
## 15 5.5157e-04     46   0.25427 0.32929 0.012572
## 16 3.3094e-04     48   0.25317 0.33315 0.012635
## 17 2.7579e-04     53   0.25152 0.33370 0.012643
## 18 1.8386e-04     61   0.24931 0.33480 0.012661
## 19 6.8946e-05     64   0.24876 0.33867 0.012723
## 20 0.0000e+00     72   0.24821 0.33867 0.012723
```

The relative error in the summary decreases at any additional split, so it is not useful to evaluate the predictive accuracy of the model, while the xerror (which stands for cross-validated relative error) reaches a minimum. Since the xerror is computed using 10-fold cross-validation procedure by default, the values slightly changes every time we fit a new model. The relative error remains exactly the same.

```
plotcp(spam.rpart.cp0)
```



Now that we have a very large (overfitted) tree, what is the best number of splits for pruning our tree? Changes in the xerrors are so small that running the model again would probably lead to a different choice of number of splits if it is based on selecting the tree with the absolute minimum xerror. To reduce instabilty in the choice we can use the one-standard-deviation rule. The horizontal dashed line in the plot shows the minimum xerror + standard deviation. So choosing the smallest tree whose xerror is less or equal than this value will lead us to a more conservative choice if the interest is in choosing the optimal predictive tree.

So we select the overall best split and the best split according to the standard deviation rule

```r
best.cp <- spam.rpart.cp0$cptable[which.min(spam.rpart.cp0$cptable[,"xerror"]),]
best.cp
```

```
##           CP       nsplit    rel error       xerror         xstd
##  0.000827358 29.000000000  0.269167126  0.323221180  0.012472906
```

```r
sd.rule<- best.cp["xerror"] + best.cp["xstd"]
cptable.sd.rule <- spam.rpart.cp0$cptable[spam.rpart.cp0$cptable[,"xerror"] <= sd.rule,]
best.cp.sd <- cptable.sd.rule[which.min(cptable.sd.rule[,"nsplit"]),]
best.cp.sd
```

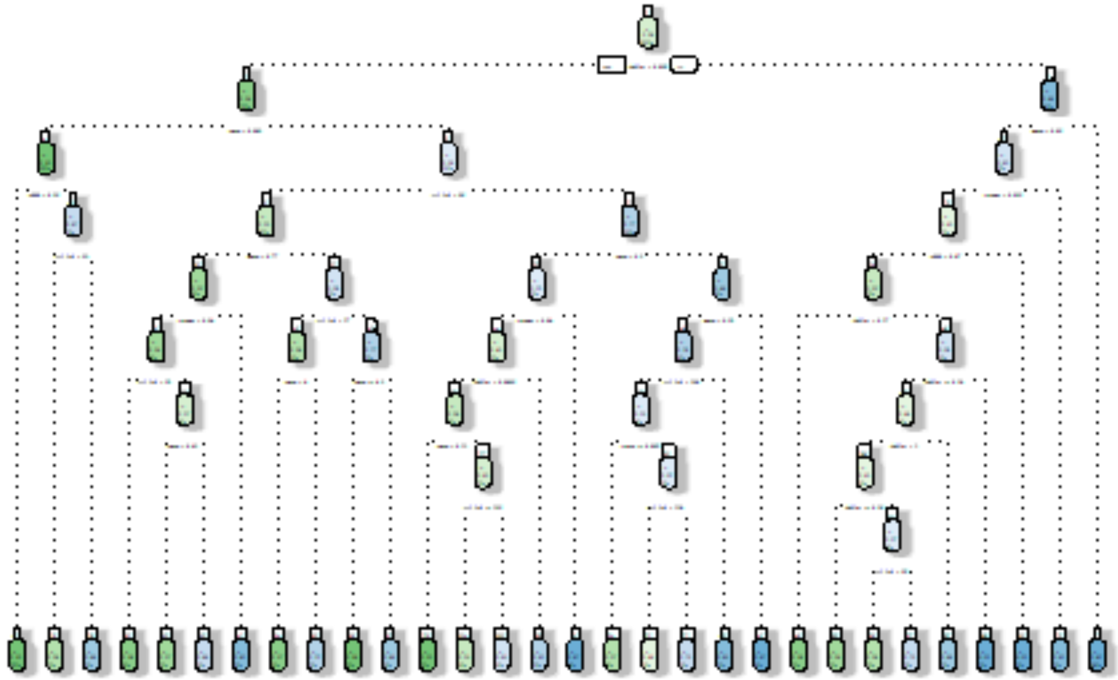
```
##           CP       nsplit    rel error       xerror         xstd
##  0.003861004 12.000000000  0.306674021  0.331494760  0.012607916
```

Then, we can prune the final tree by means of the prune function and analyse the summary of both the models, as well as the graphs reporting the decision rule.

```r
tree.pruned <-prune(spam.rpart.cp0, cp = best.cp[1])
printcp(tree.pruned)
```

```
##
## Classification tree:
## rpart(formula = yesno ~ crl.tot + dollar + bang + money + n000 +
##     make, data = spam, method = "class", cp = 0)
##
## Variables actually used in tree construction:
## [1] bang    crl.tot dollar  make    money   n000
##
## Root node error: 1813/4601 = 0.39404
##
## n= 4601
##
##            CP nsplit rel error  xerror     xstd
## 1  0.47655819      0   1.00000 1.00000 0.018282
## 2  0.07556536      1   0.52344 0.55323 0.015447
## 3  0.01158301      3   0.37231 0.38665 0.013445
## 4  0.01047987      4   0.36073 0.38555 0.013429
## 5  0.00634308      5   0.35025 0.37010 0.013205
## 6  0.00551572     10   0.31660 0.35852 0.013031
## 7  0.00441258     11   0.31109 0.34363 0.012801
## 8  0.00386100     12   0.30667 0.33149 0.012608
## 9  0.00275786     16   0.29123 0.32543 0.012509
## 10 0.00220629     17   0.28847 0.32653 0.012527
## 11 0.00193050     18   0.28627 0.32598 0.012518
## 12 0.00165472     20   0.28240 0.32377 0.012482
## 13 0.00091929     25   0.27413 0.32488 0.012500
## 14 0.00082736     29   0.26917 0.32322 0.012473
```

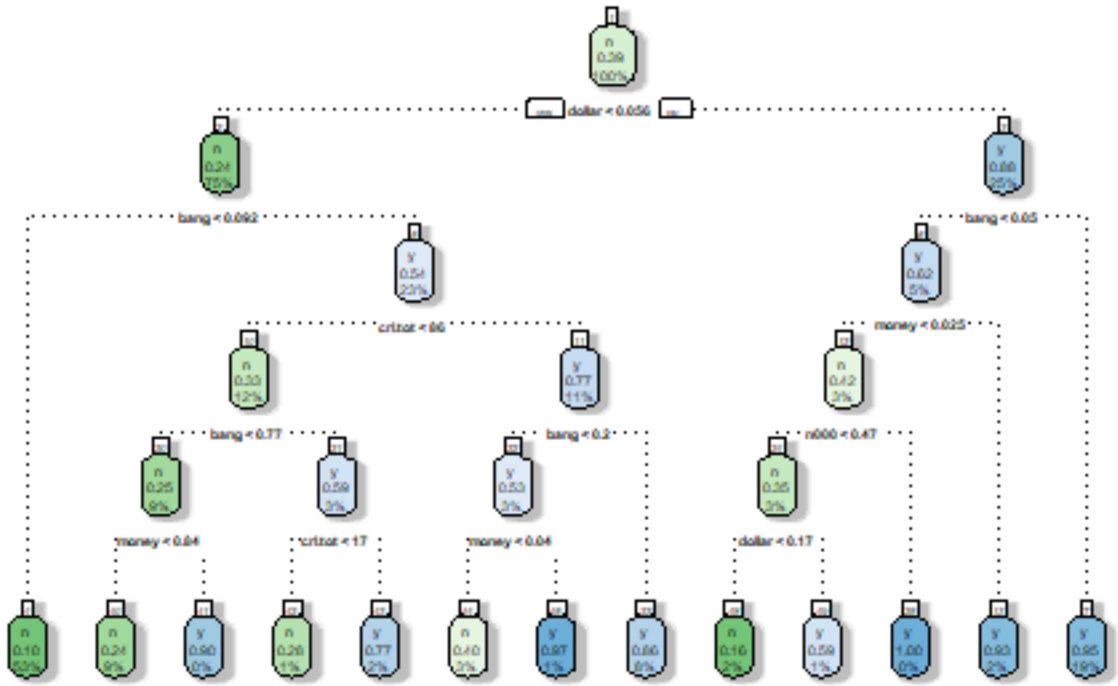
```r
rpart.plot(tree.pruned, extra = 106, box.palette = "GnBu", branch.lty = 3,
           shadow.col = "gray", nn = TRUE)
```



```r
tree.pruned.sd <-prune(spam.rpart.cp0, cp = best.cp.sd[1])
printcp(tree.pruned.sd)
```

```
##
## Classification tree:
## rpart(formula = yesno ~ crl.tot + dollar + bang + money + n000 +
##     make, data = spam, method = "class", cp = 0)
##
## Variables actually used in tree construction:
## [1] bang    crl.tot dollar  money   n000
##
## Root node error: 1813/4601 = 0.39404
##
## n= 4601
##
##          CP nsplit rel error  xerror     xstd
## 1 0.4765582      0   1.00000 1.00000 0.018282
## 2 0.0755654      1   0.52344 0.55323 0.015447
## 3 0.0115830      3   0.37231 0.38665 0.013445
## 4 0.0104799      4   0.36073 0.38555 0.013429
## 5 0.0063431      5   0.35025 0.37010 0.013205
## 6 0.0055157     10   0.31660 0.35852 0.013031
## 7 0.0044126     11   0.31109 0.34363 0.012801
## 8 0.0038610     12   0.30667 0.33149 0.012608
```

```r
rpart.plot(tree.pruned.sd, extra = 106, box.palette = "GnBu", branch.lty = 3,
           shadow.col = "gray", nn = TRUE)
```



Absolute cross validation error for this last model is: $0.33756 * 0.39404 = 0.1330121$, thus the model achieved an error rate of 13.3%. (The value of 0.33756 could be different since it depends on the CV)

Extra: RandomForest

Let us explore the most important function for considering a random forest for the spam data.

```
library(randomForest)
spam.rf <- randomForest(yesno ~ ., data = spam, importance = TRUE)
print(spam.rf)
```

```
##
## Call:
##  randomForest(formula = yesno ~ ., data = spam, importance = TRUE)
##                Type of random forest: classification
##                      Number of trees: 500
## No. of variables tried at each split: 2
##
##         OOB estimate of  error rate: 11.67%
## Confusion matrix:
##      n    y class.error
## n 2638  150  0.05380201
## y  387 1426  0.21345836
```

```
importance(spam.rf)
```

```
##                n         y MeanDecreaseAccuracy MeanDecreaseGini
## crl.tot 50.72899  56.34976             73.86364        258.34416
## dollar  58.37230  55.48474             77.17146        405.59115
## bang    92.33322 100.24580            116.28277        597.93553
## money   31.71152  50.24642             49.99576        204.55280
## n000    58.13733  16.33049             62.97053        126.77642
## make    14.37853  21.63179             26.38483         42.43996
```

```
varImpPlot(spam.rf, sort = TRUE)
```