R Workshop

Session 5: Tree Models and their Allies

Bill Venables, CSIRO, Australia

UseR! 2012

Nashville

11 June, 2012

Contents

1 Trees and forests

- A technique that developed in machine learning and now widely used in data mining.
- The model uses *recursive partitioning* of the data and is a greedy algorithm.
- The two main types of tree models are
  - **Regression trees** — response is a continuous variable and fitting uses a least squares criterion,
  - **Classification trees** — response is a factor variable and fiting uses an entropy (multinomial likelihood) criterion.

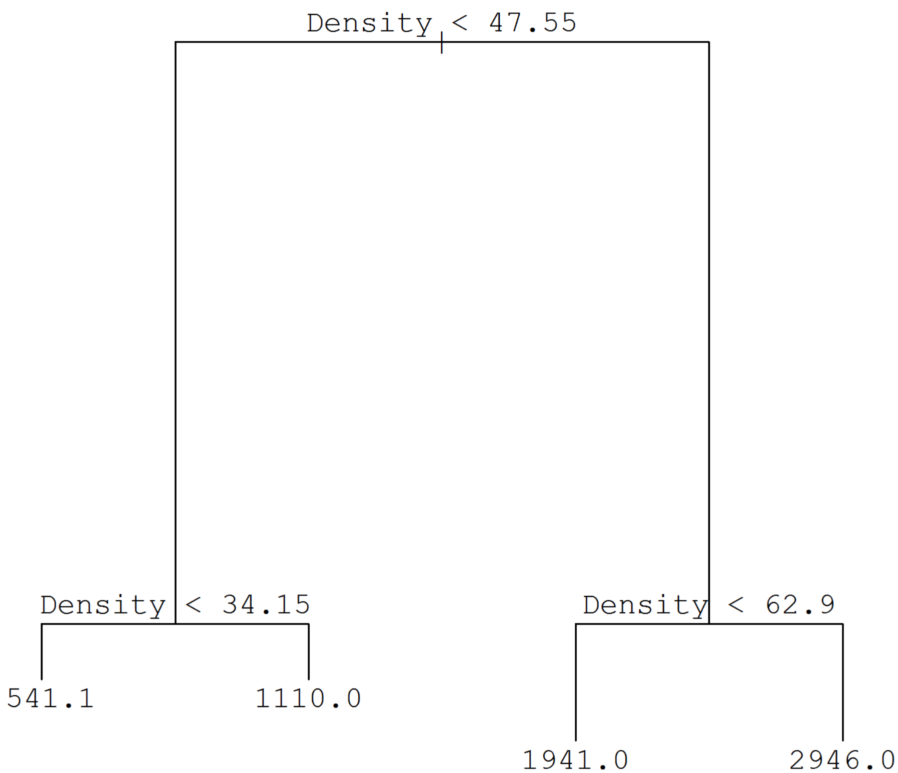
- Model fitting is easy. Inference poses more of a dilemma.
- The tree structure is very unstable. *boosting* and *bagging* (random forests) can be useful ways around this.
- Two pacakges for tree models: `rpart` (which is part of **R** itself) and the older `tree`, (Ripley., 2012), which has an **S-PLUS** flavour and a few advantages for teaching.

  Use `rpart` in practice.

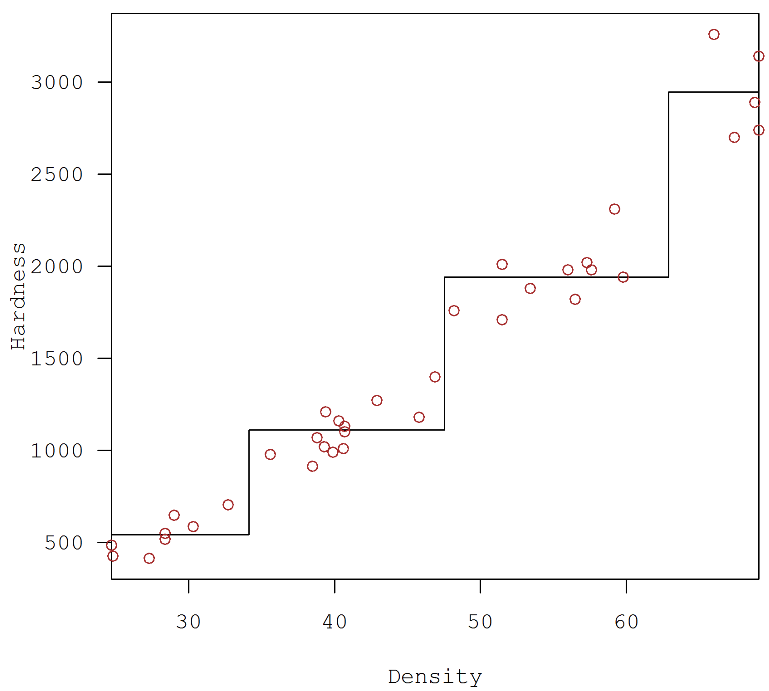
1.1 A trivial example

The `janka` data: a regression tree.

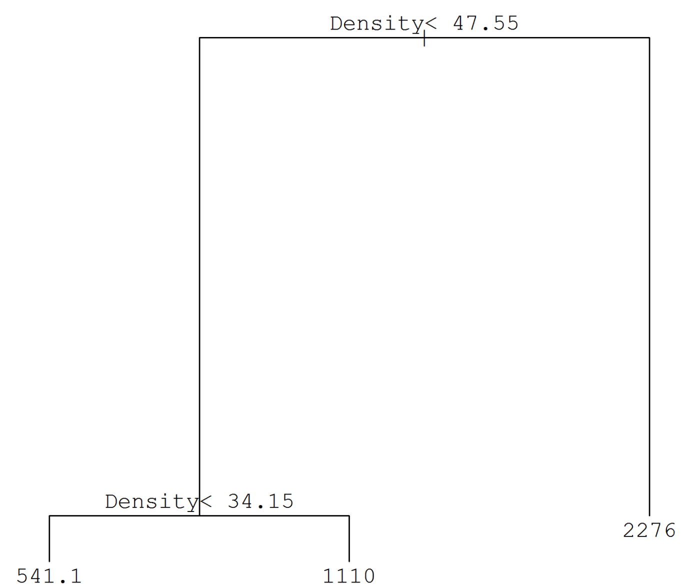
```
> if(require(tree)) {
    janka.tm <- tree(Hardness ~ Density, janka)
    plot(janka.tm); text(janka.tm)
  }
```


Density < 47.55
Density < 34.15
Density < 62.9
541.1
1110.0
1941.0
2946.0

```
> if(require(tree)) {
    partition.tree(janka.tm)
    points(Hardness ~ Density, janka, col = "brown")
  }
```

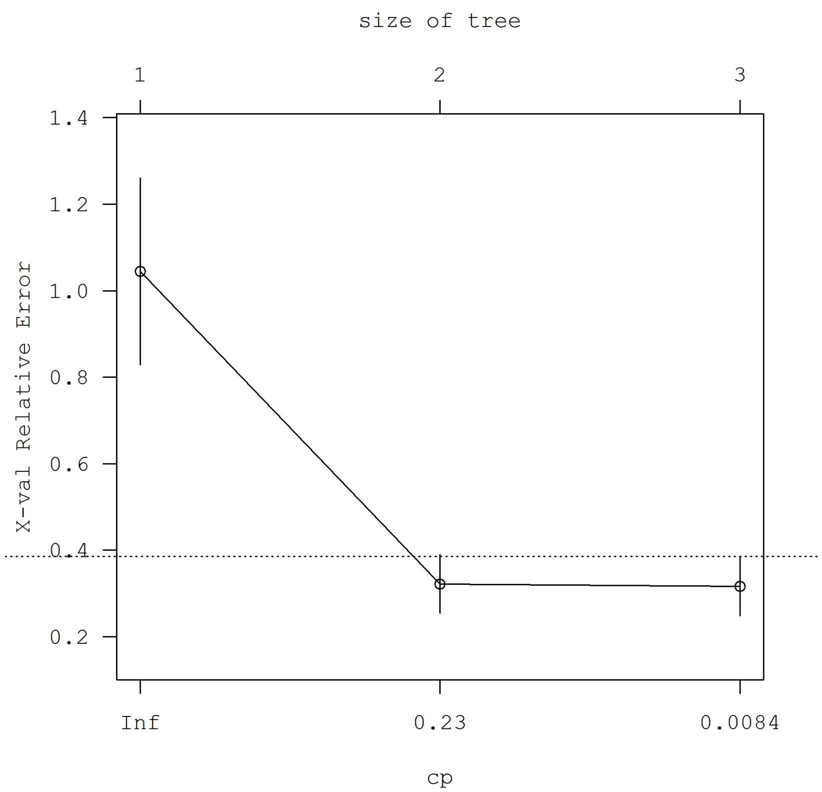


```
> require(rpart)
> janka.rm <- rpart(Hardness ~ Density, janka,
     control = rpart.control(cp = 0.001, minsize = 3))
> plot(janka.rm); text(janka.rm, xpd = NA)
```



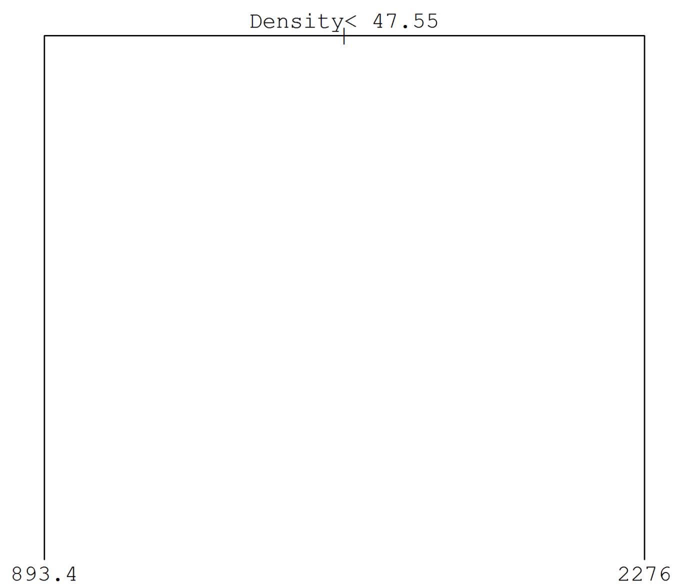
Trees need to be pruned for signal/noise improvement.

```
> plotcp(janka.rm)
```



The function(s) *oneSERule* are ours (see later).

```
> janka.rmp <- prune(janka.rm, cp = oneSERule(janka.rm))
> plot(janka.rmp); text(janka.rmp)
```



2 Do you want a credit card?

Our main example comes from a credit card marketing project in Zurich. (i.e. the dark side).

- Response: binary variable *credit.card.owner*
- Candidate predictors: banking behaviour and personal variables made on banking customers.
- Problem: build a predictive model for credit card ownership.
- Strategies: Trees, bagged trees, random forests, glms.

The data set is `creditCards`.

```
> data(creditCards)
> dim(creditCards)

[1] 2085   65

> Store(creditCards)
```

2.1 Training and test groups

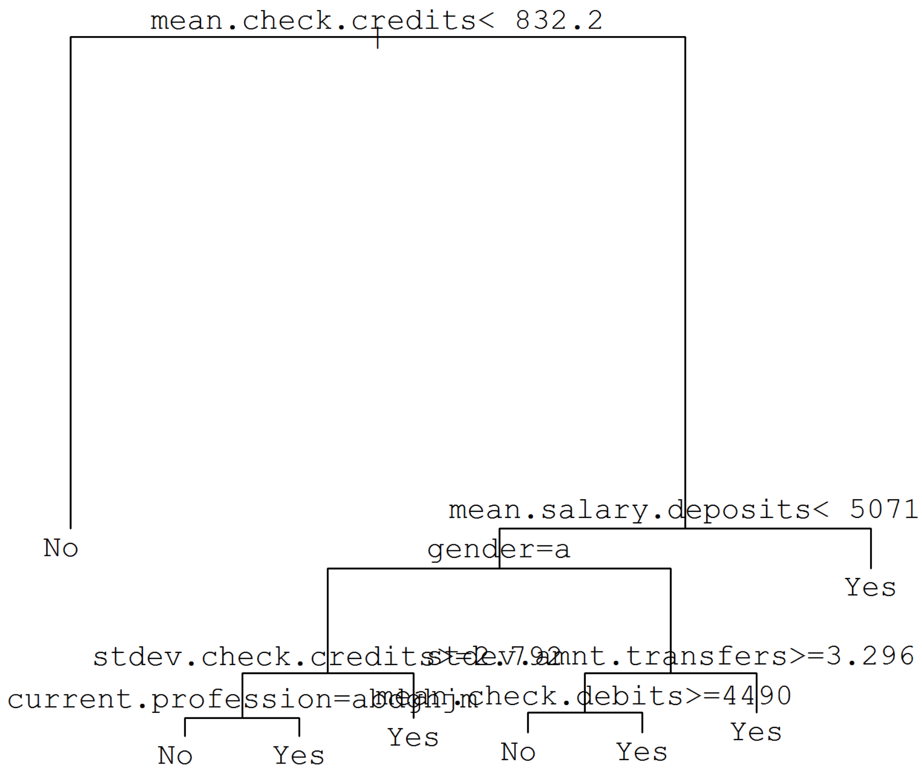
As an illustrative devide, we split the data into a *training* and a *test* group.

```
> set.seed(1234)
> nCC <- nrow(creditCards)
> train <- sample(nCC, 1000)
> CCTrain <- creditCards[train, ]
> CCTest <- creditCards[-train, ]
> Store(CCTrain, CCTest)  ## for safe keeping
```

2.2 An initial tree model

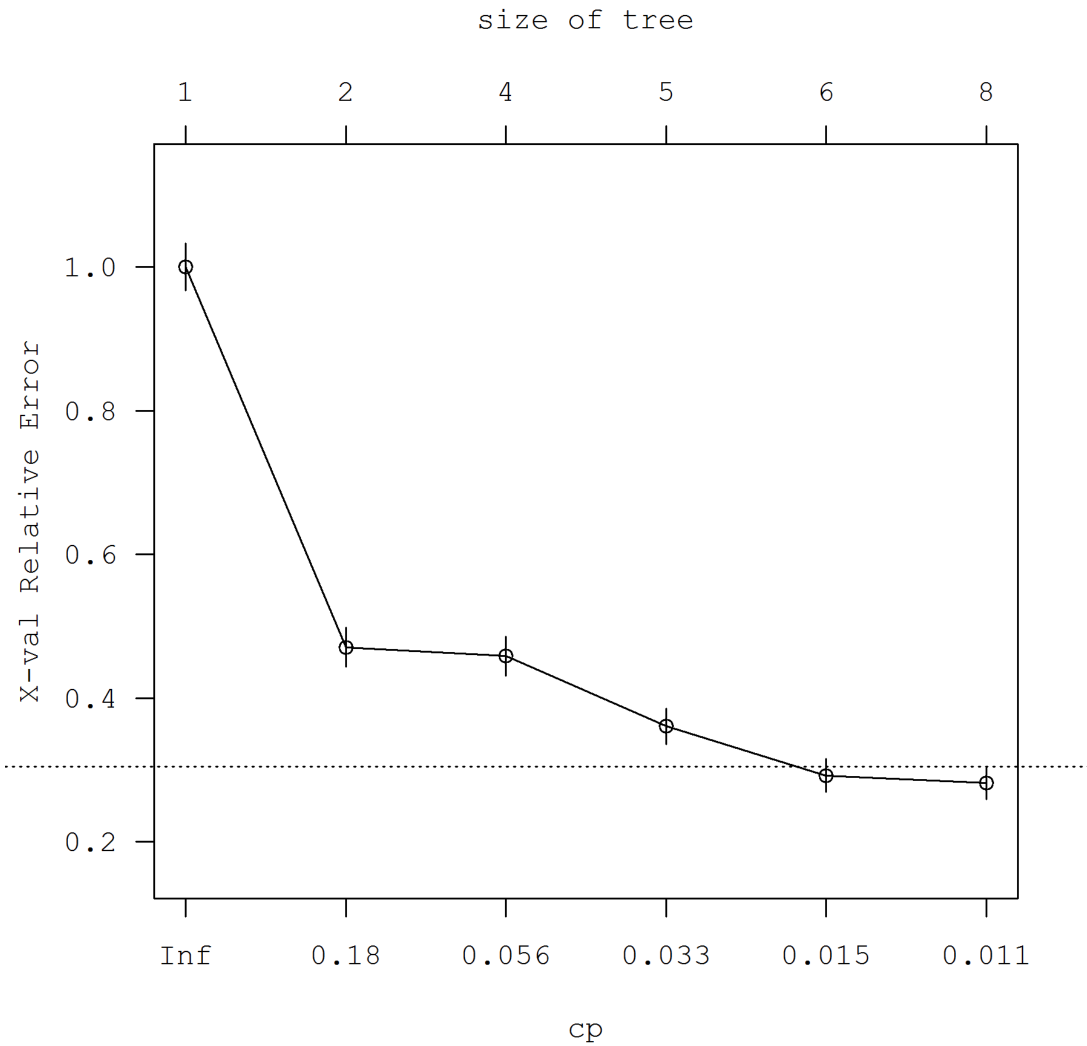
```
> library(rpart)
> CCTree <- rpart(credit.card.owner ~ ., CCTrain)
> plot(CCTree)
> text(CCTree)
```

```
> Store(CCTree)
```



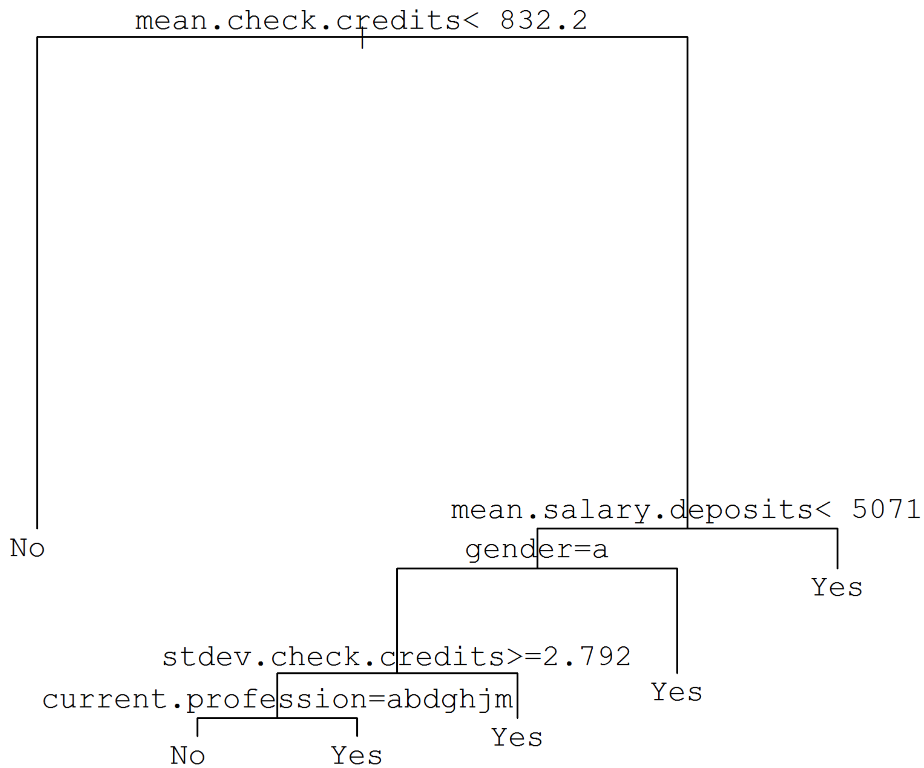
Now check for the need to prune:

```
> plotcp(CCTree)
```



Pruning is suggested by the "one standard error" rule. Get the pruned tree:

```
> CCPTree <- prune(CCTree, cp = oneSERule(CCTree))
> plot(CCPTree)
> text(CCPTree)
> Store(CCPTree)
```


mean.check.credits< 832.2
No
mean.salary.deposits< 5071
gender=a
Yes
stdev.check.credits>=2.792
current.profession=abdghjm
Yes
No
Yes
Yes

The ”one standard error rule” function(s) are listed here for completeness. The coding details are not of importantce.

```
> oneSERule <- function (tree, f, ...)
  UseMethod("oneSERule")
> oneSERule.rpart <- function (tree, f = 1, ...) {
      cp <- data.frame(tree$cptable)    #$
      imin <- with(cp, which(xerror == min(xerror))[1])
      with(cp, CP[which(xerror <= xerror[imin] + f * xstd[imin])[1]])
  }
> Store(oneSERule, oneSERule.rpart)  ## to make available later
```

2.3 Simple bagging

"Bootstrap aggregation" — invented by Leo Breimann as a device to stabilise tree methods and improve their predictive capacity. Very much a "black box" technique.

- Grow a forrest of trees using bootstrap samples of the training data.
- For predictions average over the forrest:
  - For classification trees, take a majority vote,
  - For regression trees, take an average.

'Random forests', (Liaw and Wiener, 2002), is an of bagging with extra protocols imposed.

Consider bagging "by hand".

```
> bagRpart <- local({
    bsample <- function(dataFrame) # bootstrap sampling
      dataFrame[sample(nrow(dataFrame), rep = TRUE),  ]
    function(object, data = eval.parent(object$call$data),
             nBags=200, type = c("standard", "bayesian"), ...) {
      type <- match.arg(type)
      bagsFull <- vector("list", nBags)
      if(type == "standard") {
        for(j in 1:nBags)
            bagsFull[[j]] <- update(object, data = bsample(data))
        } else {
          nCases <- nrow(data)
          for(j in 1:nBags)
              bagsFull[[j]] <- update(object, weights = rexp(nCases))
          }
      class(bagsFull) <- "bagRpart"
      bagsFull
    }
  })
```

```r
> ## a prediction method for the objects (somewhat tricky!)
> predict.bagRpart <- function(object, newdata, ...) {
    X <- sapply(object, predict, newdata = newdata, type = "class")
    candidates <- levels(predict(object[[1]], type = "class"))
    X <- t(apply(X, 1, function(r) table(factor(r, levels = candidates))))
    factor(candidates[max.col(X)], levels = candidates)
  }
> Store(bagRpart, predict.bagRpart)
```

Now for an object or two:

```
> if(!exists("CCSBag")) {
    set.seed(4321)
    Obj <- update(CCTree, cp = 0.005, minsplit = 9)  ## expand the tree
    CCSBag <- bagRpart(Obj, nBags = 100)
    CCBBag <- bagRpart(Obj, nBags = 100, type = "bayes")
    rm(Obj)
    Store(CCSBag, CCBBag)
  }
```

2.4 The actual random forest

The random forest package, (Liaw and Wiener, 2002), implements this technology, and more, automatically. The number of trees is set to 500 by default. How many times does each observation get sampled if we restrict it to 100 trees?

```
> n <- nrow(CCTest)
> X <- replicate(100,
         table(factor(sample(n, rep=TRUE), levels = 1:n)))
> (lims <- range(rowSums(X > 0)))

[1] 50 77

> rm(n, X)
```

So in this simulation the cases were sampled between 50 and 77 times. This seems about enough.

We now fit the random forest.

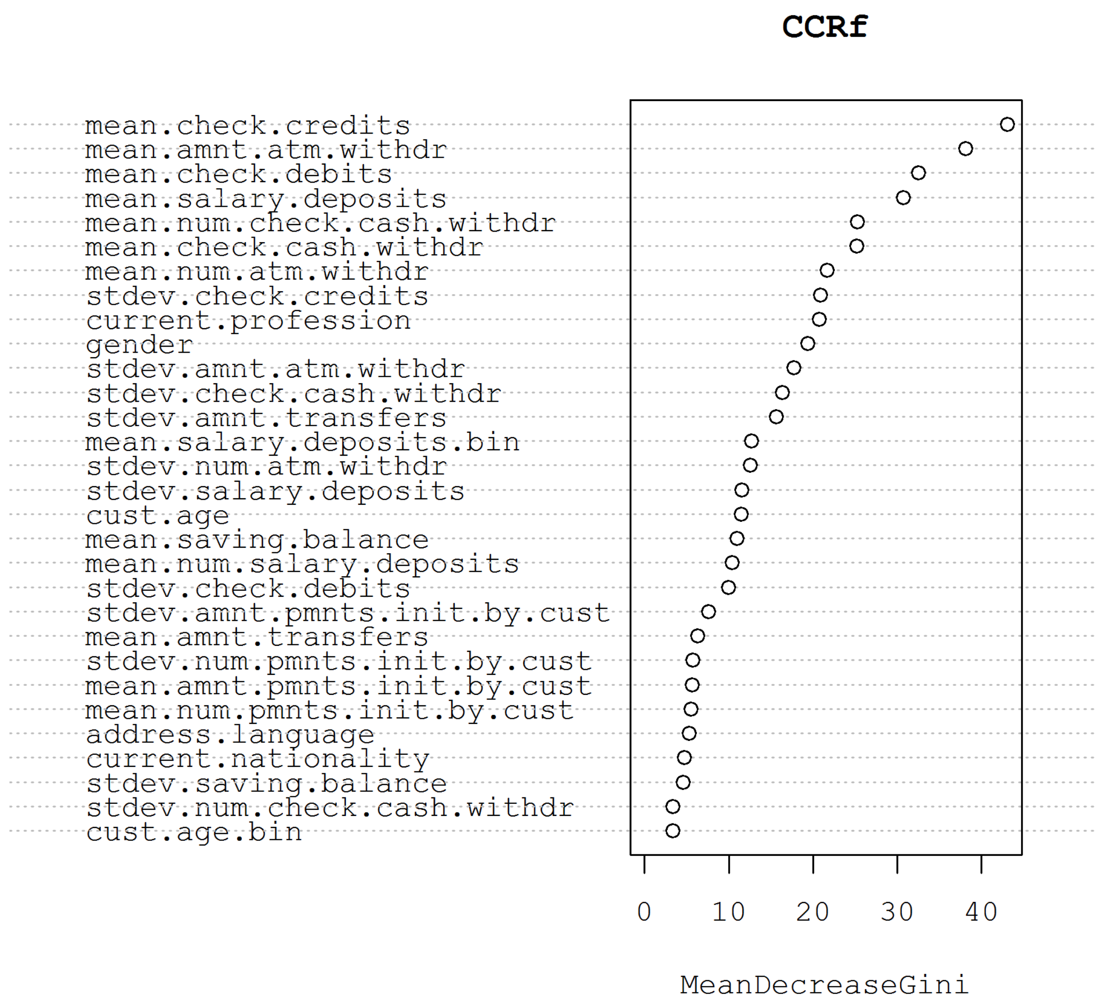
```
> suppressPackageStartupMessages(library(randomForest))
> (CCRf <- randomForest(credit.card.owner ~ ., CCTrain, ntree = 100))

Call:
 randomForest(formula = credit.card.owner ~ ., data = CCTrain,      ntree = 100)
               Type of random forest: classification
                     Number of trees: 100
No. of variables tried at each split: 8
        OOB estimate of  error rate: 10.8%
Confusion matrix:
     No Yes class.error
No  404  78  0.16182573
Yes  30 488  0.05791506

> Store(CCRf)
```

One nice by-product is variable importances.

```
> v <- varImpPlot(CCRf)  ## causes a plot
> v <- sort(drop(v), decreasing = TRUE)
> v[1:6]
        mean.check.credits        mean.amnt.atm.withdr
                  43.10637                    38.14263
         mean.check.debits        mean.salary.deposits
                  32.51816                    30.68258
mean.num.check.cash.withdr      mean.check.cash.withdr
                  25.24800                    25.19548
> bestFew <- setdiff(names(v)[1:20], "current.profession")  ## used later
```


CCRf
mean.check.credits
mean.amnt.atm.withdr
mean.check.debits
mean.salary.deposits
mean.num.check.cash.withdr
mean.check.cash.withdr
mean.num.atm.withdr
stdev.check.credits
current.profession
gender
stdev.amnt.atm.withdr
stdev.check.cash.withdr
stdev.amnt.transfers
mean.salary.deposits.bin
stdev.num.atm.withdr
stdev.salary.deposits
cust.age
mean.saving.balance
mean.num.salary.deposits
stdev.check.debits
stdev.amnt.pmnts.init.by.cust
mean.amnt.transfers
stdev.num.pmnts.init.by.cust
mean.amnt.pmnts.init.by.cust
mean.num.pmnts.init.by.cust
address.language
current.nationality
stdev.saving.balance
stdev.num.check.cash.withdr
cust.age.bin
0
10
20
30
40
MeanDecreaseGini

2.5 Parametric models

Tree models and random forests are natural competitors to the standard parametric models, notably GLMs. We begin with a naive model based only on what appear good variables in the random forest, and then consider other modest versions, but automatically produced.

```
> form <- as.formula(paste("credit.card.owner~", paste(bestFew, collapse="+")))
> Call <- substitute(glm(FORM, binomial, CCTrain), list(FORM = form))
> CCGlmNaive <- eval(Call)
> Store(CCGlmNaive)
> if(!exists("CCGlmAIC")) {
    upp <- paste("~", paste(setdiff(names(CCTrain), "credit.card.owner"),
                 collapse="+"))
    upp <- as.formula(upp)
    start <- glm(credit.card.owner ~ mean.check.credits+gender,
                 binomial, CCTrain)
    CCGlmAIC <- stepAIC(start, list(upper=upp, low=~1), trace=FALSE)
    CCGlmBIC <- stepAIC(CCGlmAIC, trace = FALSE, k = log(nrow(CCTrain)))
```

```
    Store(CCGlmAIC, CCGlmBIC)
    rm(start, upp)
  }
```

2.6 The final reckoning

Now to see how things worked out this time. First a helper function

```
> Class <- function(object, newdata, ...)
      UseMethod("Class")
> Class.rpart <- function(object, newdata, ...)
      predict(object, newdata, type = "class")
> Class.bagRpart <- function(object, newdata, ...)
      predict(object, newdata)
> Class.randomForest <- predict
> Class.glm <- function(object, newdata, ...) {
    ## only applies for binomial glms and symmetric link fns
    predict(object,newdata) > 0
  }
```

The helper function *Class* streamlines things a bit:

```
> errorRate <- function(tab) 100*(1 - sum(diag(tab))/sum(tab))
> true <- CCTest$credit.card.owner  #$
> sort(sapply(list(Tree = CCTree,
                   Pruned = CCPTree,
                   Bagging = CCSBag,
                   Bayes = CCBBag,
                   RandomF = CCRf,
                   NaiveGLM = CCGlmNaive,
                   Glm_AIC = CCGlmAIC,
                   Glm_BIC = CCGlmBIC),
               function(x) errorRate(table(Class(x, CCTest),
                                            true))))
 RandomF  Bagging    Bayes     Tree   Pruned  Glm_BIC NaiveGLM  Glm_AIC
10.96774 11.98157 12.90323 13.36406 14.47005 14.47005 14.83871 15.02304
```

2.7 Some notes on the outcome

- Random forests a winner, but not by much ($\approx 1\%$) and the "hand made" versions were next in line. This is not unusual.

  Note that the random forest error rate was very close to the internally estimated "out of bag" estimate from the construction process.

- The tree models slightly out-performed the parametric models, but again, not by much.

- Pruning did not improve the tree model, but automatic construction was about as good as picking variables after some data snooping! The latter is unusual.

3 Technical highlights

- Slide ...

References

Liaw, A. and M. Wiener (2002). Classification and regression by `randomForest`. *R News 2*(3), 18–22.

Ripley., B. (2012). *tree: Classification and regression trees*. CRAN. R package version 1.0-29.

Venables, W. N. and B. D. Ripley (2002). *Modern Applied Statistics with* **S** (Fourth ed.). New York: Springer. ISBN 0-387-95457-0.

Session information

- R version 2.15.0 (2012-03-30), `i386-pc-mingw32`
- Locale: `LC_COLLATE=English_Australia.1252`, `LC_CTYPE=English_Australia.1252`, `LC_MONETARY=English_Australia.1252`, `LC_NUMERIC=C`, `LC_TIME=English_Australia.1252`
- Base packages: base, datasets, graphics, grDevices, methods, stats, utils
- Other packages: randomForest 4.6-6, rpart 3.1-53, SOAR 0.99-10, tree 1.0-29
- Loaded via a namespace (and not attached): tools 2.15.0