1 Introduction

2 Models with Built–In Feature Selection

Many models included in `caret` have built–in feature selection, including `rpart`, `gbm`, `ada`, `glm-boost`, `gamboost`, `blackboost`, `ctree`, `sparseLDA`, `sddaLDA`, `sddaQDA` `glmnet`, `lasso`, `lars`, `spls`, `earth`, `fda`, `bagEarth`, `bagFDA` , `pam` and others. Many of the functions have an ancillary method called `predictors` that returns a vector indicating which predictors were used in the final model.

In many cases, using these models with built–in feature selection will be more efficient than algorithms where the search routine for the right predictors is "outside" the model. Built–in feature selection couples the predictor search algorithm with the parameter fitting and are usually optimize with a single objective function (e.g. error rates or likelihood).

3 Feature Selection Using Search Algorithms

3.1 Searching the Feature Space

Search routines; validation

3.2 Resampling and External Validation

3.3 Backwards Selection

The recursive feature elimination (RFE), a.k.a. backwards selection, can be used to estimate the appropriate number of prectors. First, the algorithm fits the model to all predictors. Each predictor is ranked on how important it is to teh model. Let S_i be a seqeunce of ordered numbers which are canidate values for the number of predictors to retain. At each iteration of feature selection, the S_i top raked predictors are retianed, the model is refit and performance is assessed. The value of S_i with teh best performance is chosen and the top S_i predictors are used to fit the final model. Algorithm 1 has a more complete definition.

The algorithm has an optional step where the predictor rankings are recomputed on the model on the reduced feature set. REF used RFE with ranomd forest and reported that there was XXX.

There are some cases where re–ranking the predictors could help. XXX put this below XXX.

```
Tune/train the model on the training set using all predictors
Calculate model performance
Calculate variable importance or rankings
for Each subset size $S_i$, $i = 1 \ldots S$ do
    Keep the $S_i$ most important variables
    Tune/train the model on the training set using $S_i$ predictors
    Calculate model performance
    [Optional] Recalculate the rankings for each predictor
end
Calculate the performance profile over the $S_i$
Determine the appropriate number of predictors and the final ranks of each predictor
Fit the final model based on the optimal $S_i$
```

Algorithm 1: Recursive feature elimination

As previously mentioned, computing the appropriate performance measure can be difficult. To get performace estiamtes that incorporate the variaiton due to feature selection, it is suggested that the steps in be "wrapped" inside a layer of resampling (e.g. 10–fold cross–validation). Algorithm 2 shows a version of the algorithm that uses resampling.

While this will provie better estimates of performance, it is more computationally burdensome. For users with access to machines with multiple processors, the first For loop in Algorithm 2 can be easily parallelized. Another complication to using reampling is that multiple lists of the "best" predictors are generated at each iteration. At first this may seem like a disadvantage, but it does provide a more probablistic assessment of predictor importance than a ranking based on a single, fixed data set. At the end of the algorithm, a concensus ranking can be used to determine the best predictors to retain.

```
for Each Resamping Iteration do
    Partition data into training and test/hold–back set via resampling
    Tune/train the model on the training set using all predictors
    Predict the held–back samples
    Calculate variable importance or rankings
    for Each subset size $S_i$, $i = 1 \ldots S$ do
        Keep the $S_i$ most important variables
        Tune/train the model on the training set using $S_i$ predictors
        Predict the held–back samples
        [Optional] Recalculate the rankings for each predictor
    end
end
Calculate the performance profile over the $S_i$ using the held–back samples
Determine the appropriate number of predictors and the final ranks of each predictor
Fit the final model based on the optimal $S_i$ using the original training set
```

Algorithm 2: Recursive feature elimination incorporating resampling

4 Recursive Feature Elimination via caret

4.1 An Example

To test the algorithm, the "Friedman 1" benchmark problem (Friedman, 1991) was used. there are three informative variables generated with

$$y = 10\sin(\pi x_1 x_2) + 20(x_3 - 0.5)^2 + 10x_4 + 5x_5 + N(0, \sigma^2)$$

In the simulation that follows:

```
> n <- 100
> p <- 40
> sigma <- 1
> set.seed(1)
> sim <- mlbench.friedman1(n, sd = sigma)
> x <- cbind(sim$x, matrix(rnorm(n * p), nrow = n))
> y <- sim$y
> colnames(x) <- paste("var", 1:ncol(x), sep = "")
```

Of the 50 predictors, there are 45 pure noise variables: 5 are uniform on [0, 1] and 40 are random univariate standard normals.

The predictors are centered and scaled:

```
> normalization <- preProcess(x)
> x <- predict(normalization, x)
> x <- as.data.frame(x)
> subsets <- c(1:5, 10, 15, 20, 25)
```

The simulation will fit models with subset sizes of 25, 20, 15, 10, 5, 4, 3, 2, 1. A linear model and random forests will be used.

```
> set.seed(10)
> ctrl <- rfeControl(functions = lmFuncs, method = "cv", verbose = FALSE,
+     returnResamp = "final")
> lmProfile <- rfe(x, y, sizes = subsets, rfeControl = ctrl)
> print(lmProfile)
```

```
Recursive feature selection
```

```
Outer resamping method was 10 iterations of cross-validation.

Resampling perfromance over subset size:

 Variables  RMSE Rsquared RMSESD RsquaredSD Selected
         1 3.473   0.5285 0.4706     0.1219
         2 3.134   0.6161 0.5937     0.1757
         3 2.954   0.6770 0.9152     0.2242        *
         4 3.055   0.6520 0.9889     0.2359
         5 3.229   0.6188 0.8714     0.1966
        10 3.493   0.5549 0.9811     0.2098
        15 3.754   0.5010 1.1806     0.2243
        20 3.893   0.4725 1.0039     0.2026
        25 4.306   0.4009 0.9284     0.1870
        50 4.306   0.4009 0.9284     0.1870

The top 3 variables (out of 3):
   var4, var5, var2
```

4.2 Helper Functions

4.2.1 the `fit` function

This function builds the model based on the current data set. The possible arguments are:

- `x`: the current training set of predictor data with the appropriate subset of variables
- `y`: the current outcome data (either a numeric or factor vector)
- `first`: a single logical values for whether the current predcitor set has all possible variables
- ...: optional arguments to pass to the fit function in the call to `rfe`

The `first` argument can be useful. For example, if a random forest model is fit, you may only want the initial model with all predictor varibles to be run wiht `importance = TRUE`.

This function should return a model fit function that can be used for prediction

```
> rfFuncs$fit
```

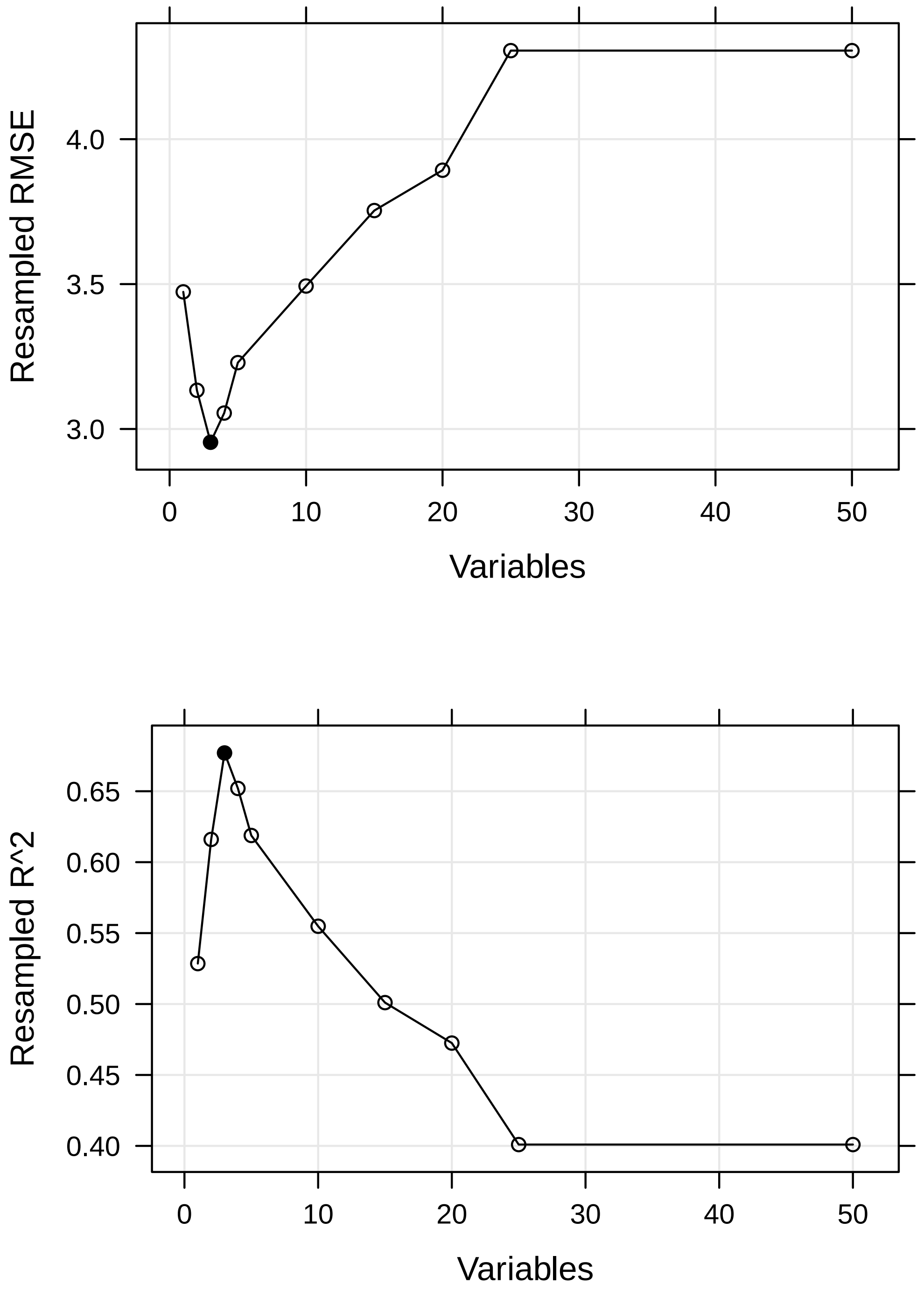

Figure 1: Hold–out performance distributions for four models with built–in feature selection.

```
function (x, y, first, last, ...)
{
    library(randomForest)
    randomForest(x, y, importance = first, ...)
}
<environment: namespace:caret>
```

4.2.2 the `pred` function

This function returns a vector of predictions (numeric or factors) from the model. The input arguments must be

- `object`: the model generated by the `fit` function
- `x`: the current set of predictor set for the held–back samples

```
> rfFuncs$pred

function (object, x)
{
    predict(object, x)
}
<environment: namespace:caret>
```

4.2.3 the `rank` function

This function should return XXXX.

Inputs are:

- `object`: the model generated by the `fit` function
- `x`: the current set of predictor set for the training samples
- `y`: the current training outcomes

```
> rfFuncs$rank

function (object, x, y)
{
    vimp <- varImp(object)
    if (is.factor(y)) {
        if (all(levels(y) %in% colnames(vimp))) {
            avImp <- apply(vimp[, levels(y), drop = TRUE], 1,
                mean)
            vimp$Overall <- avImp
        }
    }
    vimp <- vimp[order(vimp$Overall, decreasing = TRUE), , drop = FALSE]
    vimp$var <- rownames(vimp)
    vimp
}
<environment: namespace:caret>
```

4.2.4 the selectVar function

Inputs for the function are:

- y: a list of varibles importance for each resampling iteration and each subset size (generated by the user–defined rank function)
- size: the subset sized passed into the call to rfe

This function should return character string of predictor names (of length size) in the order of most important to least important

```
> rfFuncs$selectVar
```

```
function (y, size)
{
    sizes <- unlist(lapply(y[[1]], nrow))
    sizeIndex <- which(size == sizes)
    allImp <- do.call("rbind", lapply(y, function(u, pos) u[[pos]],
        pos = sizeIndex))
    meanImp <- aggregate(allImp[, grep("Overall$", names(allImp))[1]],
        list(var = allImp$var), mean)
    meanImp$imp <- meanImp$x
    counts <- aggregate(allImp[, grep("Overall$", names(allImp))[1]],
        list(var = allImp$var), length)
    counts$pct <- counts$x/length(y)
    counts$x <- NULL
    varInfo <- merge(counts, meanImp)
    varInfo <- varInfo[order(varInfo$pct, varInfo$imp, decreasing = TRUE),
        ]
    as.character(varInfo$var[1:size])
}
<environment: namespace:caret>
```

4.2.5 the `selectSize` function

Inputs for the function are:

- `x`: a matrix with columns for the performance metrics and the number of variables, called `Variables`
- `metric`: a character string of the performance measure to optimize (e.g. RMSE, Accuracy)
- `maximize`: a single logical for whether the metric should be maximized

This function should return an integer that indicates the row of `x` that is optimal

`caret` comes with two examples runtions for this purpose: `selectBest`

For example, suppose we have computed the RMSE over a series of variables sizes:

```
> set.seed(10)
> ctrl$functions <- rfFuncs
> rfProfile <- rfe(x, y, sizes = subsets, rfeControl = ctrl)
> print(rfProfile)
```

```
Recursive feature selection

Outer resamping method was 10 iterations of cross-validation.

Resampling perfromance over subset size:

 Variables  RMSE Rsquared RMSESD RsquaredSD Selected
         1 3.607   0.4670 0.2765    0.16005
         2 3.186   0.6079 0.5151    0.14583
         3 2.779   0.7409 0.3943    0.06699        *
         4 2.885   0.7356 0.2721    0.10742
         5 3.177   0.6806 0.4035    0.10557
        10 3.234   0.6726 0.3771    0.11912
        15 3.350   0.6648 0.3780    0.12272
```

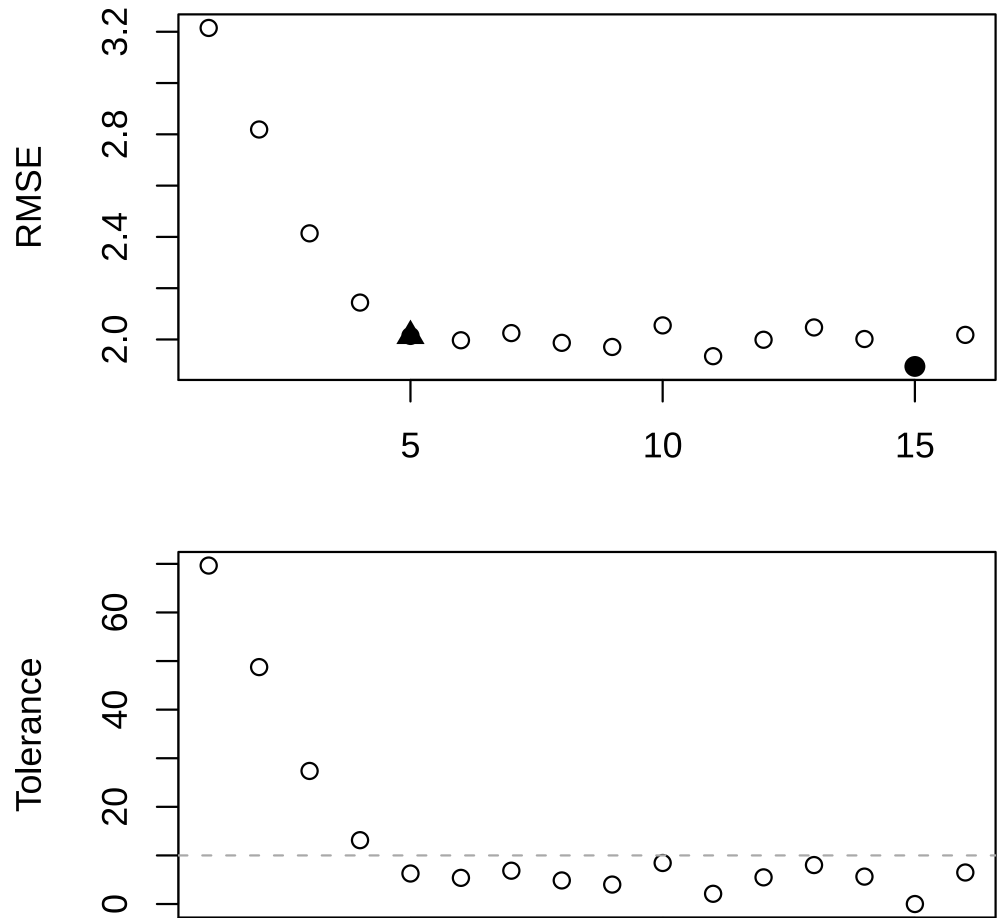

Figure 2: ads

```
    20 3.415   0.6294 0.3848    0.14043
    25 3.588   0.6166 0.3591    0.13230
    50 3.565   0.6293 0.3716    0.14347

The top 3 variables (out of 3):
   var4, var5, var2
```

5 Session Information

- R version 2.9.0 Under development (unstable) (2009-01-22 r47686), `i386-apple-darwin9.6.0`
- Locale: `en_US.UTF-8/en_US.UTF-8/C/C/en_US.UTF-8/en_US.UTF-8`
- Base packages: base, datasets, graphics, grDevices, grid, methods, splines, stats, tools, utils
- Other packages: caret 4.10, class 7.2-45, e1071 1.5-19, ellipse 0.3-5, gbm 1.6-3, Hmisc 3.5-0, ipred 0.8-6, kernlab 0.9-8, klaR 0.5-8, lattice 0.17-20, MASS 7.2-45, mlbench 1.1-5, nnet 7.2-45, pls 2.1-0, proxy 0.4-1, randomForest 4.5-28, rpart 3.1-42, survival 2.34-1
- Loaded via a namespace (and not attached): cluster 1.11.12

6 References

Chun, H. and Keles, S. (2007) "Sparse partial least squares for simultaneous dimension reduction and variable selection", `http://www.stat.wisc.edu/~keles/Papers/SPLS_Nov07.pdf`.

Friedman, J. H. (1991) "Multivariate Adaptive Regression Splines (with discussion)," *Annals of Statistics*, 19, 1–'141

Friedman, J. H. (2001) "Greedy Function Approximation: A Gradient Boosting Machine," *Annals of Statistics*, 29, 1189–1232

Zou, H. and Hastie, T. (2005) "Regularization and Variable Selection via the Elastic Net," *Journal of the Royal Statistical Society, Series B*, 67, 301–320.

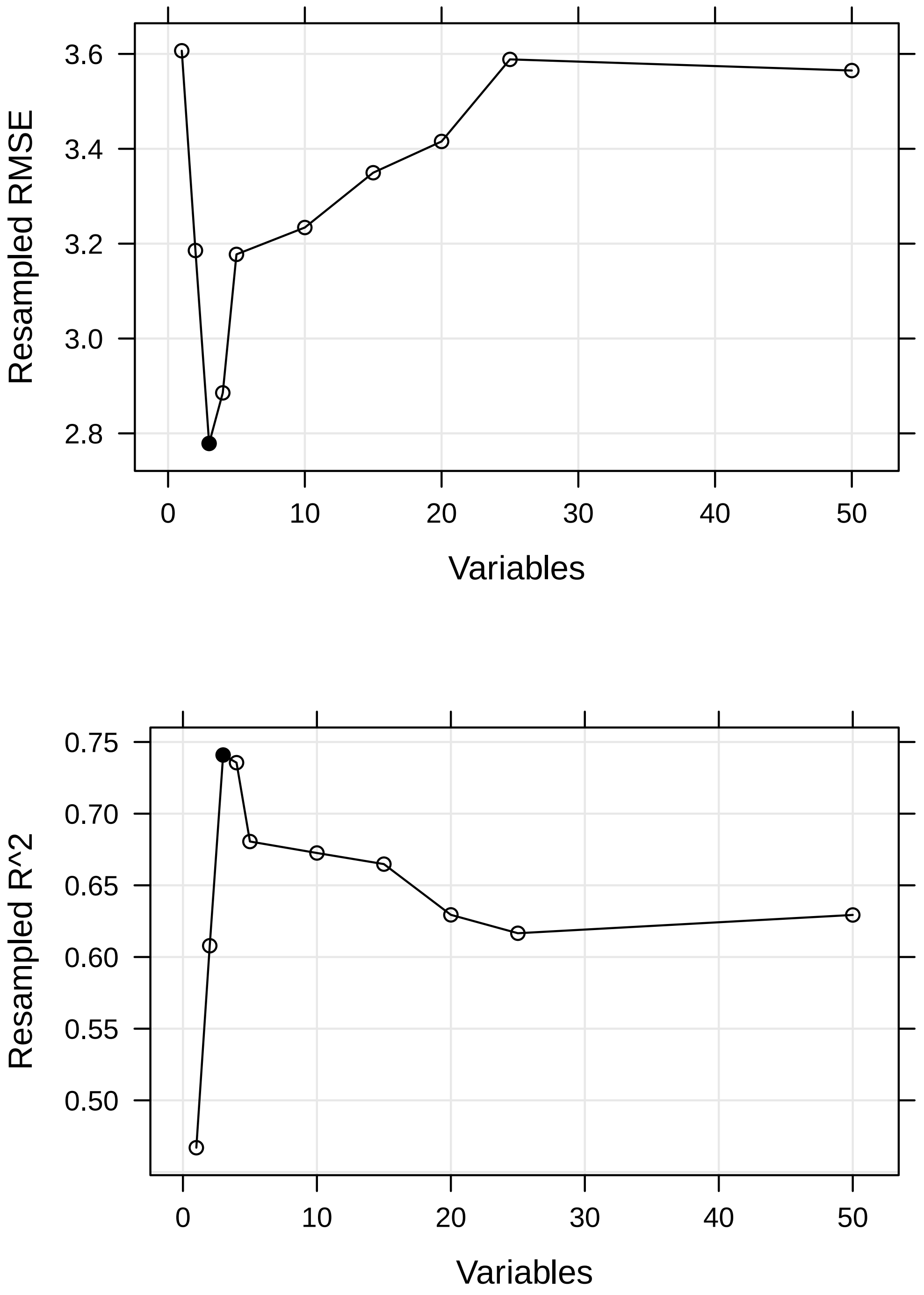

Figure 3: Hold–out performance distributions for four models with built–in feature selection.