19 Feature Selection using Univariate Filters

Contents

19.1 Univariate Filters

Another approach to feature selection is to pre-screen the predictors using simple univariate statistical methods then only use those that pass some criterion in the subsequent model steps. Similar to recursive selection, cross-validation of the subsequent models will be biased as the remaining predictors have already been evaluate on the data set. Proper performance estimates via resampling should include the feature selection step.

As an example, it has been suggested for classification models, that predictors can be filtered by conducting some sort of *k*-sample test (where *k* is the number of classes) to see if the mean of the predictor is different between the classes. Wilcoxon tests, *t*-tests and ANOVA

models are sometimes used. Predictors that have statistically significant differences between the classes are then used for modeling.

The caret function `sbf` (for selection by filter) can be used to cross-validate such feature selection schemes. Similar to `rfe`, functions can be passed into `sbf` for the computational components: univariate filtering, model fitting, prediction and performance summaries (details are given below).

The function is applied to the entire training set and also to different resampled versions of the data set. From this, generalizable estimates of performance can be computed that properly take into account the feature selection step. Also, the results of the predictor filters can be tracked over resamples to understand the uncertainty in the filtering.

19.2 Basic Syntax

Similar to the `rfe` function, the syntax for `sbf` is:

```
sbf(predictors, outcome, sbfControl = sbfControl(), ...)
## or
sbf(formula, data, sbfControl = sbfControl(), ...)
```

In this case, the details are specificed using the `sbfControl` function. Here, the argument `functions` dictates what the different components should do. This argument should have elements called `filter`, `fit`, `pred` and `summary`.

19.2.1 The `score` Function

This function takes as inputs the predictors and the outcome in objects called `x` and `y`, respectively. By default, each predictor in `x` is passed to the `score` function individually. In this case, the function should return a single score. Alternatively, all the predictors can be exposed to the function using the `multivariate` argument to `sbfControl`. In this case, the output should be a named vector of scores where the names correspond to the column names of `x`.

There are two built-in functions called `anovaScores` and `gamScores`. `anovaScores` treats the outcome as the independent variable and the predictor as the outcome. In this way, the null hypothesis is that the mean predictor values are equal across the different classes. For regression, `gamScores` fits a smoothing spline in the predictor to the outcome using a generalized additive model and tests to see if there is any functional relationship between the two. In each function the p-value is used as the score.

19.2.2 The `filter` Function

This function takes as inputs the scores coming out of the `score` function (in an argument called `score`). The function also has the training set data as inputs (arguments are called `x` and `y`). The output should be a named logical vector where the names correspond to the column names of `x`. Columns with values of `TRUE` will be used in the subsequent model.

19.2.3 The `fit` Function

The component is very similar to the `rfe` -specific function described above. For `sbf` , there are no `first` or `last` arguments. The function should have arguments `x` , `y` and `...` . The data within `x` have been filtered using the `filter` function described above. The output of the `fit` function should be a fitted model.

With some data sets, no predictors will survive the filter. In these cases, a model with predictors cannot be computed, but the lack of viable predictors should not be ignored in the final results. To account for this issue, `caret` contains a model function called `nullModel` that fits a simple model that is independent of any of the predictors. For problems where the outcome is numeric, the function predicts every sample using the simple mean of the training set outcomes. For classification, the model predicts all samples using the most prevalent class in the training data.

This function can be used in the `fit` component function to “error-trap” cases where no predictors are selected. For example, there are several built-in functions for some models. The object `rfSBF` is a set of functions that may be useful for fitting random forest models with filtering. The `fit` function here uses `nullModel` to check for cases with no predictors:

```
rfSBF$fit
```

```
## function (x, y, ...)
## {
##     if (ncol(x) > 0) {
##         loadNamespace("randomForest")
##         randomForest::randomForest(x, y, ...)
##     }
##     else nullModel(y = y)
## }
## <bytecode: 0x7f99bbccc9c8>
## <environment: namespace:caret>
```

19.2.4 The `summary` and `pred` Functions

The `summary` function is used to calculate model performance on held-out samples. The `pred` function is used to predict new samples using the current predictor set. The arguments and outputs for these two functions are identical to the previously discussed `summary` and `pred` functions in previously described sections.

19.3 The Example

Returning to the example from (Friedman, 1991), we can fit another random forest model with the predictors pre-filtered using the generalized additive model approach described previously.

```
filterCtrl <- sbfControl(functions = rfSBF, method = "repeatedcv"
set.seed(10)
rfWithFilter <- sbf(x, y, sbfControl = filterCtrl)
rfWithFilter
```

```
##
## Selection By Filter
##
## Outer resampling method: Cross-Validated (10 fold, repeated 5
##
## Resampling performance:
##
##  RMSE Rsquared   MAE RMSESD RsquaredSD  MAESD
##  3.45   0.5509 2.926  0.592     0.2132 0.5867
##
## Using the training set, 6 variables were selected:
##    real2, real4, real5, bogus2, bogus17...
##
## During resampling, the top 5 selected variables (out of a poss
##    real2 (100%), real4 (100%), real5 (100%), bogus44 (72%), bo
##
## On average, 5.6 variables were selected (min = 3, max = 8)
```

In this case, the training set indicated that 6 should be used in the random forest model, but the resampling results indicate that there is some variation in this number. Some of the informative predictors are used, but a few others are erroneous retained.

Similar to `rfe`, there are methods for `predictors`, `densityplot`, `histogram` and `varImp`.