# *Predictive Analytics: practical 4*

## *Advanced models*

The final practical is intended to let you have a go fitting models to any of the data sets that we have seen so far. Or indeed feel free to use your own data sets.

- One possible avenue to explore is the `cars2010` data set from the first practical.
- You could use the validate and mark functions to see how the more complex models compare with the simpler models from yesterday and to see who can make the best predictive model.
- Models to try could be the regression trees, neural networks, support vector machines, anything that we have covered in the notes.

Alternatively you could pick a data set and try to follow the worked example and find optimal sets of predictors for your chosen response.

If you have a classification response you would typically optimize for the probability of the event.

In addition the website

`http://topepo.github.io/caret/modelList.html`

contains a list of models currently supported by `caret`. If one of these is suitable for you, you could try to use a model we haven't directly covered in the course. The syntax and model fitting procedure is the same for all of them. This was one of the reasons we use `caret` in the first place.

## *Advanced*

This section is intended for users who have a more in depth background to R programming. Attendance to the Programming in R course should be adequate background.

`caret` is a very useful package, neatly boxing up model fitting and prediction, including finding tuning parameters based on resampling estimates of model statistics. In practical 3 we saw that we could specify our own statistics by which to choose models. Here we will extend that example adding a new model method that can be used with train.

The practical 3 advanced section motivates choice of model based on a custom metric, the distance from perfect classification model. Each classification in a two class model has so far been made when the probability of an event is greater than 50%. If we had a scenario where a false positive is more expensive than a false negative we might consider changing the classification threshold. We will consider this by creating a custom model fit based on a LDA.

To start we need an idea of how the model methods are looked at from the `caret train` function. When a method is called `caret` calls the `getModelInfo()` function.

```
modInfo = getModelInfo(model = "lda", regex = FALSE)[[1]]
names(modInfo)

##  [1] "label"      "library"    "loop"       "type"       "parameters"
##  [6] "grid"       "fit"        "predict"    "prob"       "predictors"
## [11] "tags"       "levels"     "sort"
```

We can use `str` to get an idea of the structure, essentially we just want to modify bits of this list in order to specify out custom model. We will create a list to match each necessary component and where no change is necessary we will just assign the components from modInfo.

```
customModel = list(label = "custom",
    library = modInfo$library,
    type = modInfo$type,
    parameters  = data.frame(
      parameter = c("threshold"),
      class = c("numeric"),
      label = c("Probability Cutoff")),
    grid = function(x, y, len = NULL) {
      data.frame(threshold = seq(.01, .99, length = len))
    },
    loop = modInfo$loop,
    fit = function(x, y, wts, param, lev, last, classProbs, ...) {
      if(length(levels(y)) != 2) # we have only added the warning here
        stop("This works only for 2-class problems")
      lda(x, y, ...)
    },
    predict = function(modelFit, newdata, submodels = NULL) {
      class1p = predict(modelFit,newdata)$posterior[,1]
      ## Raise the threshold for class #1 and a higher level of
      ## evidence is needed to call it class 1 so it should
      ## decrease sensitivity and increase specificity
      out = ifelse(class1p >= modelFit$tuneValue$threshold,
                   modelFit$obsLevels[1],
                   modelFit$obsLevels[2])
      out
    },
    prob = modInfo$prob,
    predictors = modInfo$predictors,
    tags = modInfo$tags,
    levels = modInfo$levels,
    sort = modInfo$sort)
```

This seems like quite a lot of code but we have really only changed

- parameters – add the threshold parameter to tune over;
- grid – a mechanism to set up the set of tuning values to try;
- fit – we added a warning as it makes no sense for problems with more than 2 classes;

- predict – add the mechanism by which to make prediction based on threshold rather than the default

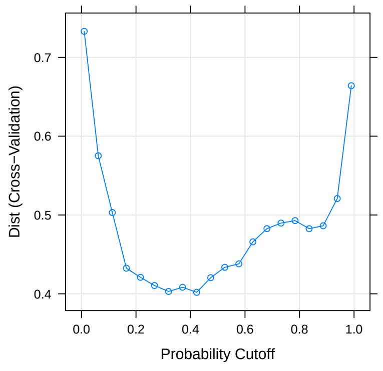
If define the fourStats function from practical 3 we can then fit the custom model

```r
fourStats = function (data, lev = NULL, model = NULL) {
  out = twoClassSummary(data, lev = levels(data$obs),
                        model = NULL)
  coords = matrix(c(1, 1, out["Spec"], out["Sens"]),
                   ncol = 2, byrow = TRUE)
  c(Dist = dist(coords)[1], out)
}
set.seed(9)
data(Sonar, package = "mlbench")
mod = train(Class ~ ., data = Sonar,
  method = customModel, metric = "Dist",
  maximize = FALSE,tuneLength = 20,
  trControl = trainControl(method = "cv",
    classProbs = TRUE, summaryFunction = fourStats))
```

If we examine the model output we can see that the best model has a prediction threshold of less than 0.5. The area under ROC is constant, but the specificity and sensitivity change. With following the structure of the caret models all of the other things still work like the plots in figure 1.

```r
plot(mod)
plot(varImp)
```

We can still use the model for predictions too, give it a try.



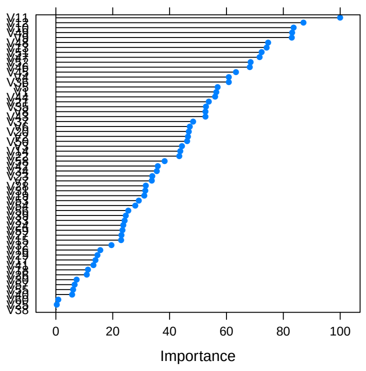


Figure 1: Using the standard caret plot functions with our custom model.