

Predictive Analytics: practical 2

Penalised regression

The diabetes data set in the `lars` package contains measurements of a number of predictors to model a response y , a measure of disease progression. There are other columns in the data set which contain interactions so we will extract just the predictors and the response. The data has already been normalized.

```
data(diabetes, package = "lars")
diabetesdata = cbind(diabetes$x, y = diabetes$y)
```

- Try fitting a lasso, ridge and elastic net model using all of the main effects, pairwise interactions and square terms from each of the predictors.¹
- Try to narrow in on the region of lowest RMSE for each model, don't forget about the `tuneGrid` argument to the `train` function. We can view what the coefficients will be by using

```
coef = predict(m.lasso$finalModel,
               mode = "fraction",
               s = 0.1, # which ever fraction was chosen as best
               type = "coefficients"
)
```

- How many features have been chosen by the lasso and enet models?
- How do these models compare to principal components and partial least squares regression?

Advanced

So far we have only used default functions and metrics to compare the performance of models, however we are not restricted to doing this. For example, training of classification models is typically more difficult when there is an imbalance in the two classes in the training set. Models trained from such data typically have high specificity but poor sensitivity or vice versa. Instead of training to maximise accuracy using data from the training set we could try to maximise according to some other criteria, namely sensitivity and specificity being as close to perfect as possible (1, 1).

To add our function we need to make sure we mirror the structure of those included in `caret` already. The following code creates a new function that could be used to summarise a model

```
fourStats = function(data, lev = NULL, model = NULL) {
  # This code will use the area under the ROC curve and the sensitivity and
  # specificity values from the built in twoClassSummary function
  out = twoClassSummary(data, lev = levels(data$obs), model = NULL)
  # The best possible model has sensitivity of 1 and specificity of 1. How far
  # are we from that value?
```

¹ Hint: see notes for shortcut on creating model formula. Also be aware that if the predictor is a factor a polynomial term doesn't make sense
`fraction = 0` is the same as the null model.

$y \sim (.)^2$ is short hand for a model that includes pairwise interactions for each predictor, so if we use this we should only need to add the square terms

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.

We can view a functions code by typing its name with no brackets.

```

coords = matrix(c(1, 1, out["Spec"], out["Sens"]), ncol = 2, byrow = TRUE)
# return the distance measure together with the output from two class
# summary
c(Dist = dist(coords)[1], out)
}

```

we could then use this in the train function

```

data(Sonar, package = "mlbench")
mod = train(Class ~ ., data = Sonar,
            method = "knn",
            # Minimize the distance to the perfect model
            metric = "Dist",
            maximize = FALSE,
            tuneLength = 20,
            trControl =
            trainControl(method = "cv", classProbs = TRUE,
                        summaryFunction = fourStats))

```

The plot function

```
plot(mod)
```

will then show the profile of the resampling estimates of our chosen statistic against the tuning parameters, see figure 1.

- Have a go at writing a function that will allow a regression model to be chosen by the absolute value of the largest residual and try using it to fit a couple of models.

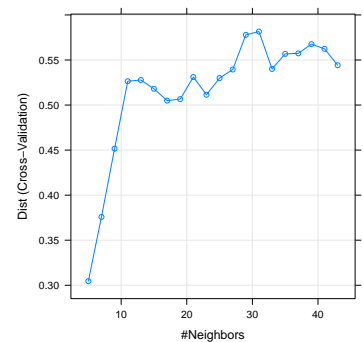


Figure 1: Plot of the distance from a perfect classifier measured by sensitivity and specificity against tuning parameter for a k nearest neighbour model.