Exercises for
Applied Predictive Modeling
Chapter 4 — Over–Fitting and Model Tuning

Max Kuhn, Kjell Johnson

Version 1
January 8, 2015

The solutions in this file uses several R packages not used in the text. To install all of the packages needed for this document, use:

```
> install.packages(c("AppliedPredictiveModeling", "car", "caret", "corrplot", "ggplot2",
+                    "e1071", "Hmisc", "mlbench", "reshape2", "subselect", "vcd"))
```

Exercise 1

Consider the Music Genre data set described in Sect. 1.4. The objective for these data is to use the predictors to classify music samples into the appropriate music genre.

(a) What data splitting method(s) would you use for these data? Explain.

(b) Using tools described in this chapter, provide code for implementing your approach(es).

Solutions

The frequency distribution of music genres is presented in Figure 1. When determining the data splitting method, we should focus on two primary characteristics:

- the number of samples relative to the number of predictors in the data, and
- the distribution of samples across classes.

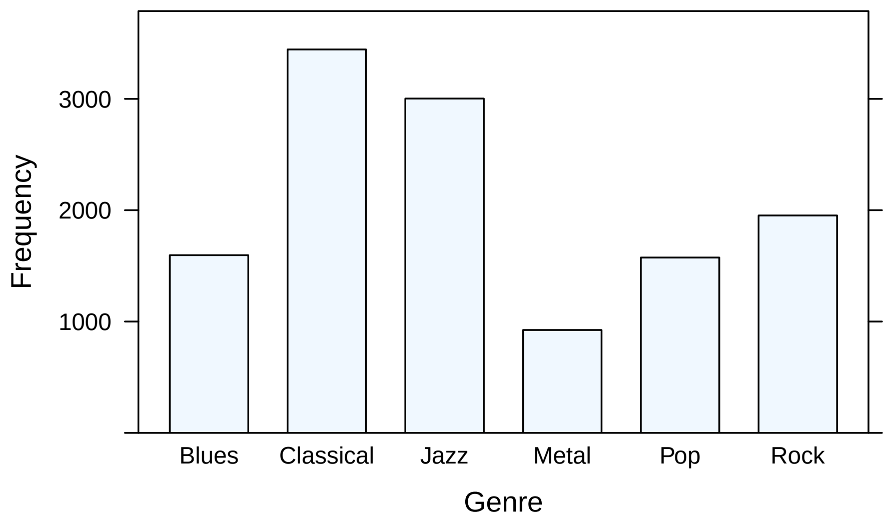

Figure 1: The frequency distribution of genres in the music data.

For these data, the number of samples is 12495 and the number of predictors is 191. Because the number of samples is more than 65 fold greater than the number of predictors, we likely can split the data into a training set and test set. This split would enable us to independently verify model performance results without impacting the estimation of the optimal tuning parameter(s) selection. However, prior to making this choice, we must examine the class distribution of the response.

Figure 1 clearly displays the imbalance across the classes of music genre with Metal having the lowest percentage of samples (7%) and Classical having the highest percentage of samples (28%). While there is a distinct imbalance, the number of samples in the overall data set is large enough such that resampling or cross-validation techniques have a good chance at randomly selecting samples across all classes in similar proportion to the entire data set.

When selecting a resampling technique on a large data set, one must consider the computational requirements of each technique and model. k-fold cross-validation, with small k (5-10) will be less computationally taxing in this scenario than repeated training/test splits or bootstrapping. However, repeated training/test splits or bootstrapping will likely provide more accurate estimates of tuning parameters and model performance.

The `createDataPartition` function in the caret package can be used to partition the data into k-folds such that each fold approximately preserves the class distribution from the entire data set. The following code could be used to create 10 folds:

```
> set.seed(31)
> tenFoldCV <- createDataPartition(trainClasses, k = 10, returnTrain = TRUE)
```

Exercise 2

Consider the permeability data set described in Section Sect. 1.4. The objective for this data is to use the predictors to model compounds' permeability.

(a) What data splitting method(s) would you use for this data? Explain.

(b) Using tools described in this chapter, provide code for implementing your approach(es).

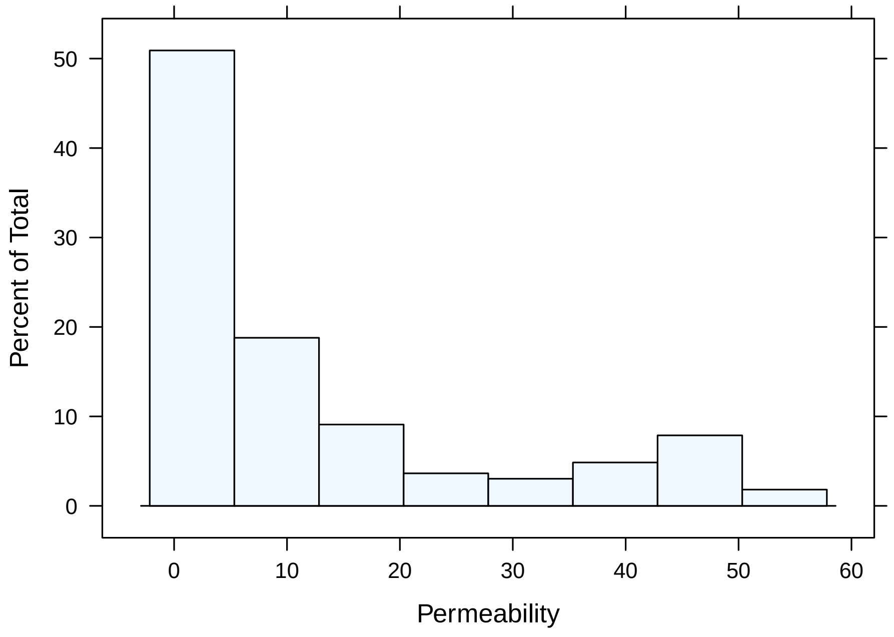

Figure 2: Distribution of permeability values.

Solutions

The frequency distribution of music genres is presented in Figure 2. In this case, the number of samples is 165 and the number of predictors is 1107. Because the number of samples is small and is much smaller than the number of predictors, splitting the data into a separate training and testing set may hinder our ability to find signal among the predictors and the response. For the permeability data, we should focus on using resampling performance measures to select optimal tuning parameters and predictive performance.

Another characteristic we should consider when determining the resampling approach is the distribution of the response. Clearly, the distribution of permeability values is skewed, with the majority

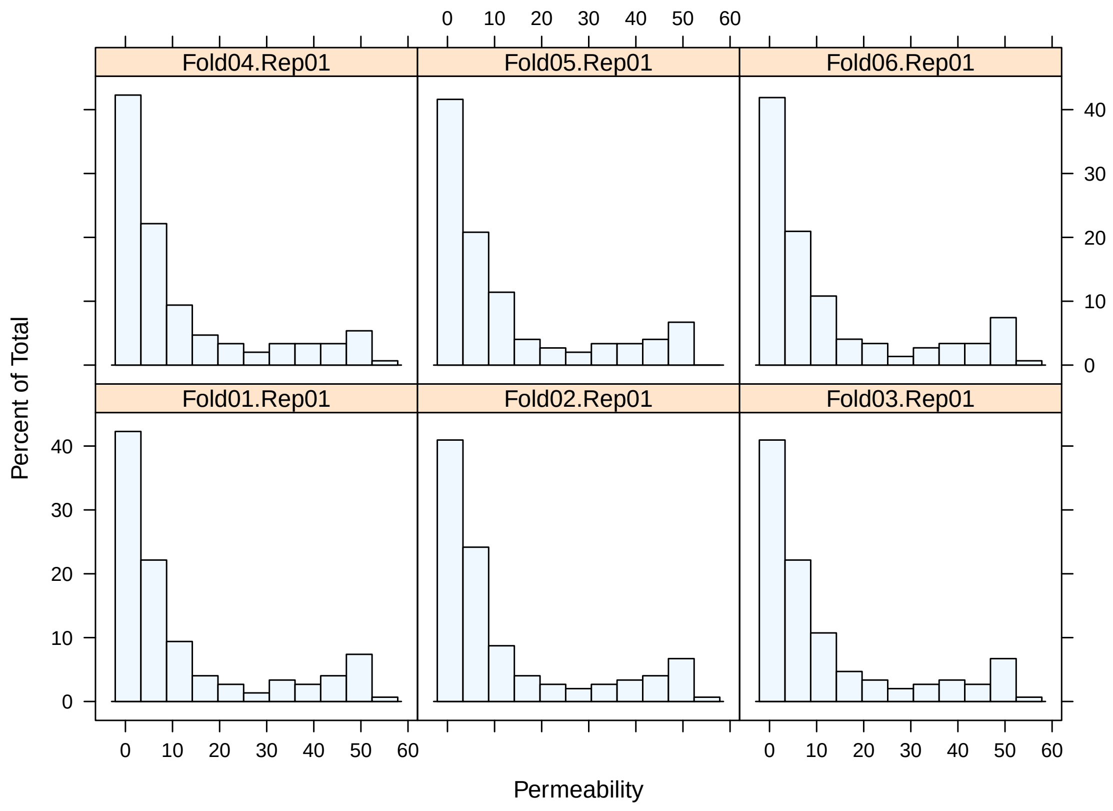

Figure 3: Distribution of permeability values within six of the folds.

of samples having low values. Because the overall number of samples is small, randomly partitioning the samples may create sets that are not representative of the overall data set. That is, we are likely to have many partitions of the data that contain relatively few of the larger permeability samples. To create more representative, but still random, selections we should use stratification.

The `createDataPartition` function in the caret package performs stratified sampling based on the quantiles of the data. Repeated cross-validation would be an appropriate resampling technique for this data. The following code could be used to create 25 iterations of 10-fold cross-validation:

```
> set.seed(72)
> repeatedCV <- createMultiFolds(permeability, k = 10, times = 25)
```

`repeatedCV` is a list that contains vectors of integers for each fold. These integers are the row numbers of the samples that should be used to model the data within the fold.

Figure 3 illustrates the distribution of response for 6 of the randomly selected partitions of the data. Notice that the distributions are representative of the overall distribution of the data in Figure 2.

Exercise 3

Partial least squares (Sect. 6.3) was used to model the yield of a chemical manufacturing process (Sect. 1.4). The data can be found in the AppliedPredictiveModeling package and can be loaded using:

```
> library(AppliedPredictiveModeling)
> data(ChemicalManufacturingProcess)
```

The objective of this analysis is to find the number of PLS components that yields the optimal R^2 value (Sect. 5.1). PLS models with 1 through 10 components were each evaluated using 5 repeats of 10-fold cross–validation and the results are presented in the following table.

(a) Using the "one–standard error" method, what number of PLS components provides the most parsimonious model?

(b) Compute the tolerance values for this example. If a 10% loss in R^2 is acceptable, then what is the optimal number of PLS components?

(c) Several other models (discussed in Part II) with varying degrees of complexity were trained and tuned and the results are presented in Figure 6. If the goal is to select the model that optimizes R^2, then which model(s) would you choose, and why?

(d) Prediction time, as well as model complexity (Sect. 4.8) are other factors to consider when selecting the optimal model(s). Given each model's prediction time, model complexity, and R^2 estimates, which model(s) would you choose, and why?

Solutions

The model was fit using:

```
> set.seed(19711230)
> plsProfileChemMod <- train(Yield ~ .,
+                            data = ChemicalManufacturingProcess,
+                            method = "pls",
+                            preProc = c("center", "scale"),
+                            tuneLength = 10,
+                            trControl = trainControl(method = "repeatedcv", repeats = 5))
```

We can get the resampling summaries in the `results` component of the object and find the number of resamples using the `index` argument of the `control` component.

Table 1: Number of components in the PLS model and their associated resampled R^2 values.

	Resampled R^2	
Components	Mean	Std. Error
1	0.447	0.0272
2	0.506	0.0298
3	0.534	0.0296
4	0.544	0.0309
5	0.542	0.0325
6	0.539	0.0329
7	0.538	0.0336
8	0.538	0.0331
9	0.526	0.0328
10	0.511	0.0325

```
> R2values <- plsProfileChemMod$results[, c("ncomp", "Rsquared", "RsquaredSD")]
> R2values$RsquaredSEM <- R2values$RsquaredSD/sqrt(length(plsProfileChemMod$control$index))
```

Table 1 show the results but let's plot them too. The easiest way to do this is using the package ggplot2. First, we can make a plot of the R^2 values, showing the resampled estimate minus one standard error.

```
> library(ggplot2)
> oneSE <- ggplot(R2values,
+                 ## Create an aesthetic mapping that plots the
+                 ## number of PLS components versus the R^2
+                 ## values and their one SE lower bound
+                 aes(ncomp, Rsquared,
+                     ymin = Rsquared - RsquaredSEM,
+                     ## don't add anything here to get
+                     ## a one-sided interval plot
+                     ymax=Rsquared))
> ## geom_linerange shoes the interval and geom_pointrange
> ## plots the resampled estimates.
> oneSE + geom_linerange() + geom_pointrange() + theme_bw()
```

```
> bestR2 <- subset(R2values, ncomp == which.max(R2values$Rsquared))
> bestR2$lb <- bestR2$Rsquared - bestR2$RsquaredSEM
> candR2 <- subset(R2values, Rsquared >=  bestR2$lb & ncomp < bestR2$ncomp)
```

Figure 4 shows the results. The best setting uses 4 PLS components with a lower bound of 0.51. There is 1 parameter setting whose resampled R^2 estimate is greater than or equal to this bound (and are simpler): a model using 3 PLS components.

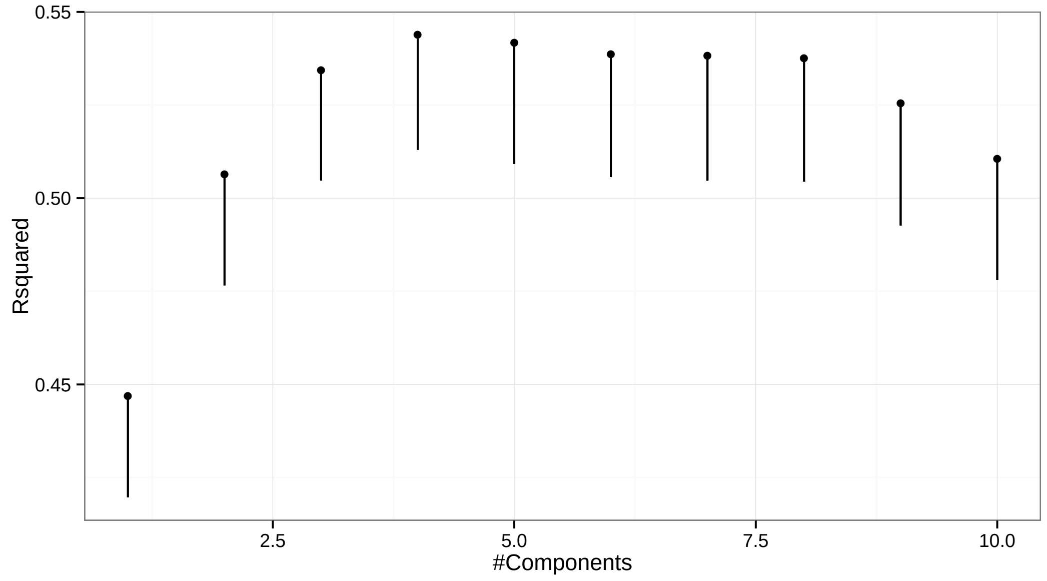

Figure 4: Resampled R^2 values with a lower bound of one standard error for the chemical manufacturing data.

The following syntax can be used to get the tolerance values:

```
> bestR2 <- subset(R2values, ncomp == which.max(R2values$Rsquared))
> R2values$tolerance <- (R2values$Rsquared - bestR2$Rsquared)/bestR2$Rsquared * 100
```

Let's stick with ggplot2 graphics and plot these in Figure 5 using `qplot(ncomp, tolerance, data = R2values)`. The lowest setting that does not exceed a 10% tolerance is a 2 component model.

Looking at Figure 6, the model with the best R^2 value is random forest. However, the support vector machine has nearly equivalent results and the confidence intervals for the R^2 values have some overlap. The next best model is boosted linear regression, although this model is probably significantly worse that the support vector machine. Based on R^2 alone, the random forest or SVM models would be best. However, when execution time is factored in, the SVM model clearly wins since it is far faster. This is, of course, subjective since it is highly dependent on the implementation. If the prediction function needed to be recoded for use, neither of these models would be preferred. In that case, the regression tree or PLS model would be better choices, albeit with a substantial drop in R^2.

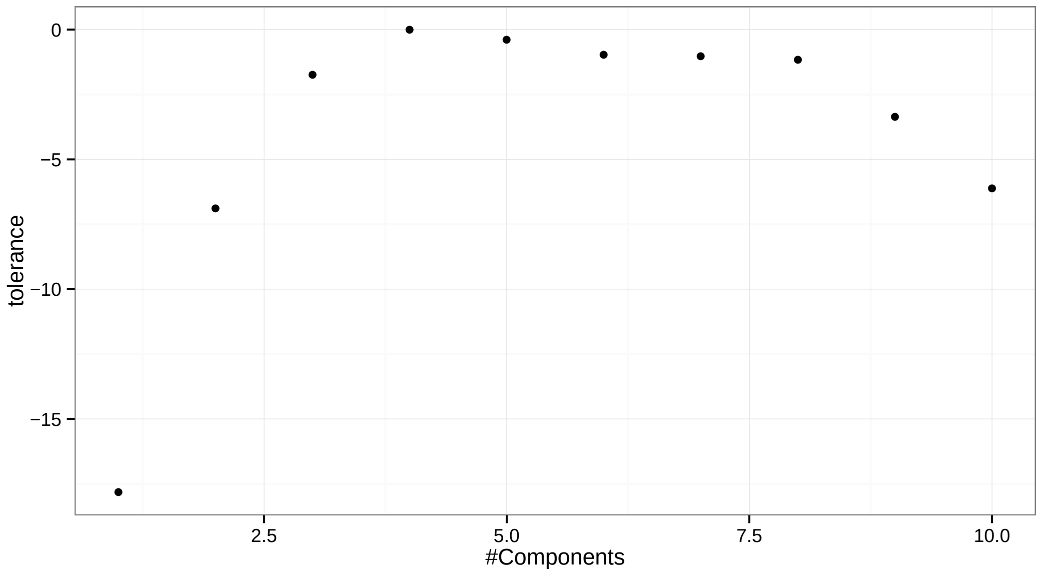

Figure 5: R^2 tolerance values for the chemical manufacturing data.

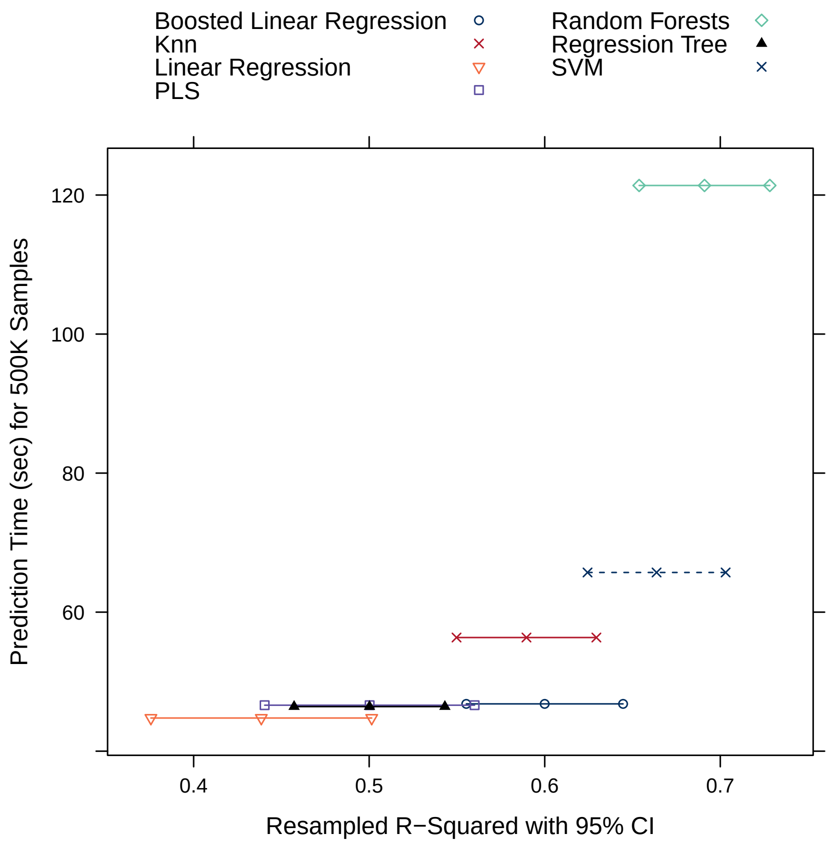

Figure 6: A plot of the estimated model performance against the time to predict 500,000 new samples using the chemical manufacturing data.

Exercise 4

? develop a methodology for food laboratories to determine the type of oil from a sample. In their procedure, they used a gas chromatograph (an instrument that separate chemicals in a sample) to measure 7 different fatty acids in an oil. These measurements would then be used to predict the type of oil in a food samples. To create their model, they used 96 samples[1] of 7 types of oils.

These data can be found in the caret package using `data(oil)`. The oil types are contained in a factor variable called `oilType`. The types are: pumpkin (coded as `A`), sunflower (`B`), peanut (`C`), olive (`D`), soybean (`E`), rapeseed (`F`) and corn (`G`). In R:

```
> library(caret)
> data(oil)
> str(oilType)
```

```
 Factor w/ 7 levels "A","B","C","D",..: 1 1 1 1 1 1 1 1 1 1 ...
```

```
> table(oilType)
```

```
oilType
 A  B  C  D  E  F  G
37 26  3  7 11 10  2
```

(a) Use the `sample` function in base R to create a completely random sample of 60 oils. How closely do the frequencies of the random sample match the original samples? Repeat this procedure several times of understand the variation in the sampling process.

(b) Use the caret package function `createDataPartition` to create a stratified random sample. How does this compare to the completely random samples.

(c) With such a small samples size, what are the options for determining performance of the model? Should a test set be used?

(d) One method for understanding the uncertainty of a test set is to use a confidence interval. To obtain a confidence interval for the overall accuracy, the based R function `binom.test` can be used. It requires the user to input the number of samples and the number correctly classified to calculate the interval. For example, suppose a test set sample of 20 oil samples was set aside and 76 were used for model training. For this test set size and a model that is about 80% accurate (16 out of 20 correct), the confidence interval would be computed using

[1]The authors state that there are 95 samples of known oils. However, we count 96 in their Table 1 (pgs. 33-35 of the article)

```
> binom.test(16, 20)


Exact binomial test

data:  16 and 20
number of successes = 16, number of trials = 20, p-value = 0.01182
alternative hypothesis: true probability of success is not equal to 0.5
95 percent confidence interval:
 0.5634 0.9427
sample estimates:
probability of success
                   0.8
```

In this case, the width of the 95% confidence interval is 37.9%. Try different samples sizes and accuracy rates to understand the trade–off between the uncertainty in the results, the model performance and the test set size.

Solutions

We can create 20 splits using the `sample` function:

```
> sampNum <- floor(length(oilType)*.6) + 1
> set.seed(629)
> oilSplits <- vector(mode = "list", length = 20)
> for(i in seq(along = oilSplits)) oilSplits[[i]] <- table(sample(oilType, size = sampNum))
> head(oilSplits, 3)

[[1]]

 A  B  C  D  E  F  G
24 15  3  4  7  4  1

[[2]]

 A  B  C  D  E  F  G
25 14  2  4  6  6  1

[[3]]

 A  B  C  D  E  F  G
24 16  2  5  5  4  2


> ## Combine the list of tables into a matrix
> oilSplits <- do.call("rbind", oilSplits)
> head(oilSplits, 3)
```

```
      A  B C D E F G
[1,] 24 15 3 4 7 4 1
[2,] 25 14 2 4 6 6 1
[3,] 24 16 2 5 5 4 2
```

```
> ## What does the distirbution of class percentages look like?
> summary(oilSplits/sampNum)
```

```
       A                B                C                 D
 Min.   :0.310   Min.   :0.224   Min.   :0.0000   Min.   :0.0345
 1st Qu.:0.345   1st Qu.:0.259   1st Qu.:0.0172   1st Qu.:0.0690
 Median :0.379   Median :0.276   Median :0.0345   Median :0.0862
 Mean   :0.381   Mean   :0.276   Mean   :0.0336   Mean   :0.0845
 3rd Qu.:0.414   3rd Qu.:0.293   3rd Qu.:0.0517   3rd Qu.:0.1035
 Max.   :0.448   Max.   :0.328   Max.   :0.0517   Max.   :0.1207
       E                 F                 G
 Min.   :0.0517   Min.   :0.0345   Min.   :0.0000
 1st Qu.:0.1035   1st Qu.:0.0862   1st Qu.:0.0172
 Median :0.1035   Median :0.1035   Median :0.0172
 Mean   :0.1069   Mean   :0.0983   Mean   :0.0198
 3rd Qu.:0.1207   3rd Qu.:0.1078   3rd Qu.:0.0215
 Max.   :0.1552   Max.   :0.1379   Max.   :0.0345
```

Using a stratified random sample using `createDataPartition`:

```
> set.seed(629)
> oilSplits2 <- createDataPartition(oilType, p = .60, times = 20)
> oilSplits2 <- lapply(oilSplits2, function(x, y) table(y[x]), y = oilType)
> head(oilSplits2, 3)
```

```
$Resample01

 A  B  C  D  E  F  G
23 16  2  5  7  6  1

$Resample02

 A  B  C  D  E  F  G
23 16  2  5  7  6  1

$Resample03

 A  B  C  D  E  F  G
23 16  2  5  7  6  1
```

```
> oilSplits2 <- do.call("rbind", oilSplits2)
> summary(oilSplits2/sampNum)
```

```
       A               B               C                D
 Min.   :0.397   Min.   :0.276   Min.   :0.0345   Min.   :0.0862
 1st Qu.:0.397   1st Qu.:0.276   1st Qu.:0.0345   1st Qu.:0.0862
 Median :0.397   Median :0.276   Median :0.0345   Median :0.0862
 Mean   :0.397   Mean   :0.276   Mean   :0.0345   Mean   :0.0862
 3rd Qu.:0.397   3rd Qu.:0.276   3rd Qu.:0.0345   3rd Qu.:0.0862
 Max.   :0.397   Max.   :0.276   Max.   :0.0345   Max.   :0.0862
       E               F               G
 Min.   :0.121   Min.   :0.103   Min.   :0.0172
 1st Qu.:0.121   1st Qu.:0.103   1st Qu.:0.0172
 Median :0.121   Median :0.103   Median :0.0172
 Mean   :0.121   Mean   :0.103   Mean   :0.0172
 3rd Qu.:0.121   3rd Qu.:0.103   3rd Qu.:0.0172
 Max.   :0.121   Max.   :0.103   Max.   :0.0172
```

The sampling done using `createDataPartition` has much less variability that using the `sample` function, and each partition has at least one sample in each class.

Choosing a data splitting strategy is difficult. One possibility would be leave–one–out cross–validation only because, with the exception of class `G`, each class will be represented in each resample. It should be noted that some classification models require at least one sample from each class, so resampling these data may place a restriction one which models can be used. As for a test set, it may be reasonable to rely on leave–one–out cross–validation as the only method for assessing performance. Alternatively, a test set could be used if it only consisted of the classes with the most samples (e.g. `A`, `B` and maybe `E` and `F`) although this would only protect against gross overfitting.

Looking at the confidence intervals for overall accuracy, let's examine samples sizes between 10 and 30 and accuracy rates of 70% to 95%:

```
> getWidth <- function(values) {
+   binom.test(x = floor(values["size"]*values["accuracy"])+1,
+              n = values["size"])$conf.int
+ }
>
> ciInfo <- expand.grid(size = 10:30, accuracy = seq(.7, .95, by = 0.01))
> ciWidths <- t(apply(ciInfo, 1, getWidth))
> head(ciWidths)
```

```
       [,1]   [,2]
[1,] 0.4439 0.9748
[2,] 0.3903 0.9398
[3,] 0.4281 0.9451
[4,] 0.4619 0.9496
[5,] 0.4190 0.9161
[6,] 0.4490 0.9221
```

```
> ciInfo$length <- ciWidths[,2] - ciWidths[,1]
> # levelplot(length ~ size * accuracy, data = ciInfo)
```

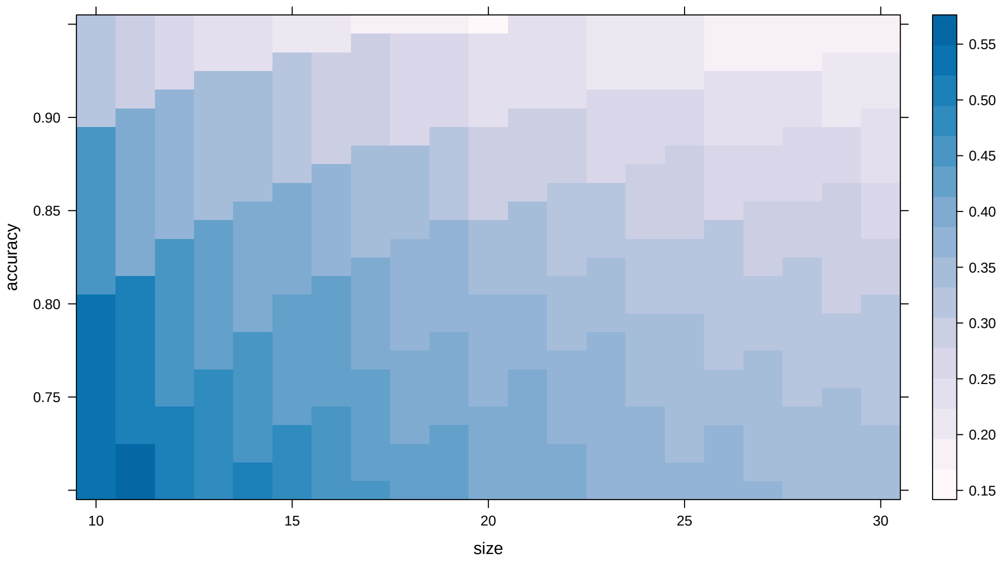

Figure 7: The width of a binomial confidence interval for overall accuracy for different sample sizes and accuracy rates.

Session Info

- R Under development (unstable) (2014-12-29 r67265), `x86_64-apple-darwin10.8.0`
- Locale: `en_US.UTF-8/en_US.UTF-8/en_US.UTF-8/C/en_US.UTF-8/en_US.UTF-8`
- Base packages: base, datasets, graphics, grDevices, grid, methods, splines, stats, utils
- Other packages: AppliedPredictiveModeling 1.1-6, caret 6.0-41, Formula 1.1-2, ggplot2 1.0.0, Hmisc 3.14-4, knitr 1.6, lattice 0.20-29, pls 2.4-3, reshape2 1.4, survival 2.37-7
- Loaded via a namespace (and not attached): BradleyTerry2 1.0-5, brglm 0.5-9, car 2.0-21, cluster 1.15.3, codetools 0.2-9, colorspace 1.2-4, compiler 3.2.0, CORElearn 0.9.43, digest 0.6.4, evaluate 0.5.5, foreach 1.4.2, formatR 0.10, gtable 0.1.2, gtools 3.4.1, highr 0.3, iterators 1.0.7, labeling 0.2, latticeExtra 0.6-26, lme4 1.1-7, MASS 7.3-35, Matrix 1.1-4, minqa 1.2.3, munsell 0.4.2, nlme 3.1-118, nloptr 1.0.4, nnet 7.3-8, plyr 1.8.1, proto 0.3-10, RColorBrewer 1.0-5, Rcpp 0.11.2, rpart 4.1-8, scales 0.2.4, stringr 0.6.2, tools 3.2.0