UCD Smurfit Graduate School of Business

Data Mining

Assignment Three - Data Mining

Authors:
Conor Reid (16202630 - MSc B.A Part-time)

Lecturer:
Dr. Michael O'Neill

April 17, 2017

Classification and Clustering- Q1

Classification and clustering are two very common data exploration and interrogation techniques, but used to achieve different goals.

In general, classification involves a number of predefined classes (to which observations are assigned) and the goal is to take a new observation and predict which of these classes it belongs to. In this way classification is a type of supervised learning. A training dataset is required, consisting of training examples. Each example is a pair, made up of the input vector (a sum of each feature in the observation) and the signal (or label, the class the observation belongs to). Classification algorithms take these training examples and infer a mapping function, which is used for new and unseen examples.

On the other hand, clustering belongs to the family of unsupervised learning. Here, a mapping function is still inferred as above, however this time observations are unlabelled. We do not know which class observations belong to, or even how may classes there may be in the first place. There is thus no way to evaluate the accuracy of the results. Clustering then attempts to group observations in such a way that observations within each group are more similar to each other that those in other groups. Ideally these groups would be highly distinct. That is, spatially very separate when plotted.

R Code Analysis - Q2

The R code being analysed here is shown below. In general, this code aims to take some dataset and partition it into training and test subsets for model learning, and testing respectively.

```
> set.seed(1234)
> dataPartition <- sample(2,nrow(data),replace=TRUE,prob=c(0.7,0.3))
> trainData <- data [dataPartition ==1,]
> testData <- data [dataPartition ==2,]
```

Since there is a random element to the above procedure, a seed is set. This contains the randomness, and allows the same results to be obtained each time the code is ran. Changing the seed changes the results.

The `sample()` function in R takes a sample of a given size, with or without replacement. Here, `sample(2)` returns the numbers 1 and 2 in a random order. Passing in the data dataset, we can assign one of the two "labels" to each observation, thus designating which of the training (1) or test (2) datasets we want that observation to be in. `replace=TRUE` means that we are sampling with replacement. `sample(2, replace=TRUE)` may return "2 , 2" or "1, 2" etc.

The `prob(...)` term at the end indicates the split of the data between the two subsets. 70% goes to training, 30% to test.

The Weka implementation of C4.5 algorithm - Q3

The C4.5 algorithm is used in generating decision trees used for classification, and is an extension the earlier ID3 algorithm, of the same authorship. The "M" parameter sets the minimum number of instances per leaf. That is, it dictates the degree of splitting at each node and thus effects the resulting accuracy of the results. For this reason a number of iterations of the algorithm is often run, with differing values of M. The model that results in the best accuracy is then chosen.

The `churnTrain` Dataset - Q4, Q5

The `churnTrain` dataset was analysed as fulfilment of this question. Upon inspection, it immediately became clear that the dataset was highly unbalanced. The dependant variable, *churn*, contains roughly 85.5% *"no"* values with the remainder being *"yes"* values. It is likely then that any model that is built on this data as-is will be more able to accurately predict the *"no"* class since it has more observations to base rules on.

Ideally one would use some balancing function to resample the minority class (*"yes"*) and increase predictive power here. One such function is `SMOTE`.

Results - Non-Pruned Classification Tree
$TreeSize = 32$
$Errors = 130(3.9\%)$

Confusion Matrix

	Yes	No	Total
Yes	369	114	483
No	16	2834	2850
Total	385	2948	3333

```
treeModel = C5.0(x = churnTrain[, -20], y = churnTrain$churn,
                 control = C5.0Control(noGlobalPruning=TRUE))
treeModel
summary(treeModel)
```

The above results relate to a classification tree that was built without a pruning step. The tree size is 32, and a relatively small error rate of 3.9%. The confusion matrix confirms that 130 instances were misclassified, being identified as *"no"* cases when they are actually *"yes"* or visa versa.

```
Attribute usage:

100.00% total_day_minutes
 93.67% number_customer_service_calls
 87.73% international_plan
 20.73% total_eve_charge
 11.34% voice_mail_plan
  8.01% total_intl_calls
  6.48% total_intl_minutes
  6.33% total_night_minutes
  5.34% total_eve_minutes
  0.57% total_eve_calls
  0.51% total_night_charge
  0.18% account_length
  0.18% total_day_charge
```

The attributes of the dataset that are used in constructing the tree are listed above. *total_day_minutes* is shown to be most important in determining churn, followed by 12 more variables in decreasing order of importance. There are 20 variables in the dataset, and so since 13 are used in building the model, it is quite dependant on the data.

Results - Pruned Classification Tree
$TreeSize = 27$
$Errors = 136(4.1\%)$

Confusion Matrix

	Yes	No	Total
Yes	365	118	483
No	18	2832	2850
Total	383	2950	3333

```
treeModel = C5.0(x = churnTrain[, -20], y = churnTrain$churn,
                 control = C5.0Control(noGlobalPruning=FALSE))
treeModel
summary(treeModel)
```

The results above relate to a classification tree that was built using a final pruning step, controlled by the $C50$ R package. The goal of pruning is to reduce the size of the tree, reducing the complexity of the final classifier. This is done by removing sections of the tree that contribute little predict power. Predictive accuracy should improve by reducing overfitting.

The error rate has shown to slightly increase here, by 0.2% which amounts to an extra 6 misclassified observations. This is very small, and the reduced dependancy of the model on the underlying data means that generalisability is greatly improved.

```
Attribute usage:

100.00% total_day_minutes
 93.67% number_customer_service_calls
 87.73% international_plan
 20.73% total_eve_charge
  8.97% voice_mail_plan
  8.01% total_intl_calls
  6.48% total_intl_minutes
  6.33% total_night_minutes
  4.74% total_eve_minutes
  0.57% total_eve_calls
  0.18% account_length
  0.18% total_day_charge
```

Again, the attributes most important to the pruned decision tree model are listed above in decreasing order of importance. This output is similar to that seen before for the un-pruned tree, however here we have 12 variables total which represents a reduction by 1 over the un-pruned tree. This may not seem significant but any reduction in model dependancy is certainly advantageous, especially when model performance is not greatly effected as is the case here.

The `GermanCredit` dataset - Q6

The `GermanCredit` dataset was analysed and modelled in a similar way to the previous question, involving some preliminary exploration followed by the growing of a classification decision tree. The following code was used to begin with.

```
library(caret)
data(GermanCredit)
table(GermanCredit$Class) #unbalanced; 300 - 700
```

The data is read in, and a table constructed of the explanatory variable (in this case, the *Class* variable which takes either a *Good* or a *Bad* value. It is apparent that this is quite unbalanced, with 300 *Bad* cases and 700 *Good* cases. As discussed previously, an imbalance like this in the dataset allows the majority class to be accurately modelled, but makes it difficult to pick up the rules that govern the minority class. In many cases, it is the minor class that is actually of interest, where one wants to be able to detect what is known as a *rare event*. The imbalance here probably wouldn't be too detrimental to model building, but it can be assumed that interested parties are more motivated to find instances of *Bad* credit than *Good* and act on that information.

For this reason it was decided to use the `SMOTE` function in R to balance the classes. `SMOTE` acts to artificially generate new instances of the minority class by using the nearest neighbours of these cases. In addition, majority instances are under-sampled resulting in a more balanced dataset. The code below was used to achieve this.

```
GermanCredit$Class <- as.factor(GermanCredit$Class)
GermanCredit <- SMOTE(Class ~ ., GermanCredit, perc.over = 100, perc.under=200)
GermanCredit$Class <- as.numeric(GermanCredit$Class)
table(GermanCredit$Class) #balanced; 600 - 600
dim(GermanCredit) #[1] 1200 62
```

With a balanced dataset, we can continue and build the now familiar classification decision tree. First we split the dataset into training and test subsets, that will be used to create the tree and test its performance respectively. The code below is used to do this.

```
#dataPartition <- sample(2,nrow(GermanCredit),replace=TRUE,prob=c(0.5,0.5))
dataPartition <- sample(2,nrow(GermanCredit),replace=TRUE,prob=c(0.7,0.3))
trainData <- GermanCredit[dataPartition ==1,]
testData <- GermanCredit[dataPartition ==2,]
trainData$Class <- as.factor(trainData$Class)
dim(trainData) #[1] 864 62
dim(testData) #[1] 336 62
```

A split of 70% for test and 30% is primarily used, although a 50% split each way was also tested. The former is a very common split used in literature, but the results of both will be discussed further on. Our primary split is used to create each dataset, which have dimensions as shown.

We are now ready to fit the model. The code below is used to do this.

```
treeModelPruned = C5.0(x = trainData[, -10], y = trainData$Class,
                  control = C5.0Control(noGlobalPruning=FALSE))
treeModelPruned
summary(treeModelPruned)
```

The results on the training dataset are outlined below.

$Treesize = 67$
$Errors = 44(5.1\%)$

Confusion Matrix

	Good	Bad	Total
Good	393	34	427
Bad	10	427	437
Total	403	461	864

The above results can be considered good, with a low error rate and by and large the correct allocation of *Good* and *Bad* labeled instances into their respective groups. We now look at the results when we used our test data on this model.

```
predictions <- predict(treeModelPruned, testData, type="class")
table(testData$Class, predictions)
```

The below confusion matrix is achieved.

Confusion Matrix

	Good	Bad	Total
Good	134	39	173
Bad	19	144	163
Total	153	183	336

This confusion matrix can be used to find the sensitivity (proportion of positive instances correctly identified) and the specificity (proportion of negative instances correctly identified). They are calculated as:

$$Sensitivity = \frac{TruePositives}{TruePositives + FalseNegatives} = \frac{134}{134 + 39} = 77\% \tag{1}$$

and

$$Specificity = \frac{TrueNegatives}{TrueNegatives + FalsePositives} = \frac{144}{144 + 19} = 88\% \tag{2}$$

These could be considered good results from our training set, albeit worse than that seen in the training set results. This may suggest overfitting at the training stage. To counter act this, an alternative split of 50% each way was used.The resulting using the same methodology on this split are shown below.

$$Sensitivity = \frac{236}{236 + 61} = 79\% \tag{3}$$

and

$$Specificity = \frac{246}{246 + 63} = 80\% \tag{4}$$

This split reduced the tree to a size of 47 which is significantly less than the previous result. It can be said then that the model dependancy is less, and it should generalise better. The sensitivity and specificity above are interesting results; out ability to detect *Good* instances has grown, while our ability to detect *Bad* instances has reduced. In the context of our problem, it may be that the first model is better since we can have greater confidence when we find a *Bad* instance that it is in fact just that. Further analysis of the problem may further inform which model is better.

The `college` dataset

References

[1] This is a sample reference