Practical Data Science with R - Practical 5B

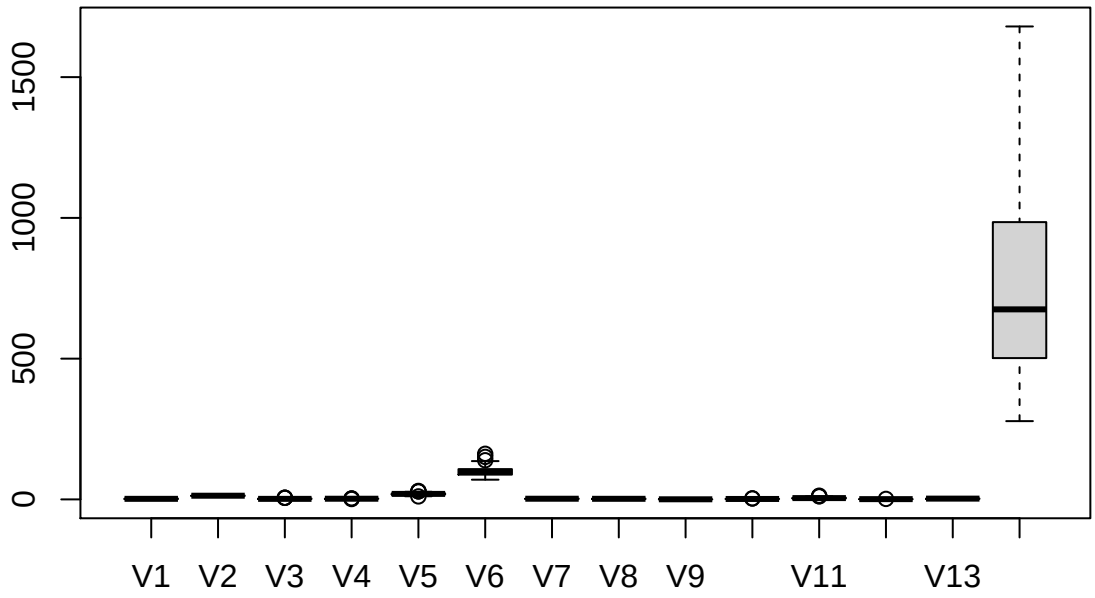
Matthew Knowles

16/02/2021

principal Component Analysis and Missing Values

First, let's read in the data, and use a boxplot to aid in the decision of whether or not we should scale the data when performing PCA.

```
wine <- read.csv("/home/matthew/Documents/University/PDS/Datasets/winedata.csv", head=F)
boxplot(wine)
```



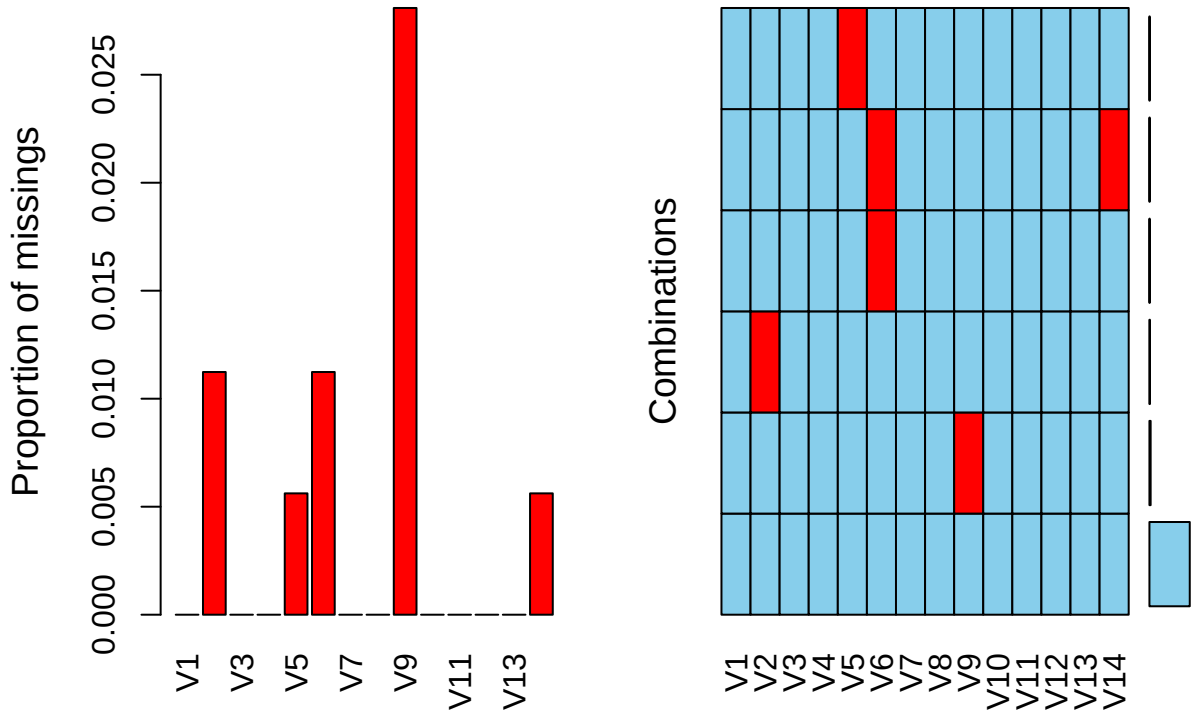
So it's very clear we do need to scale this. Moving onto the PCA:

```
winePCA <- prcomp(wine[,-1], scale = F)
```

```
## Error in svd(x, nu = 0, nv = k): infinite or missing values in 'x'
```

We can see here that there are some variables that have no values for. On inspection of the full dataset, we can see that variable 8 has a missing value for V5. There are almost certainly more, but this it the first example of what caused the above error. Luckily, we can take a deeper look at the missing values. For this, we can use *aggr*:

```
aggData <- aggr(wine, plot = T)
```




```
aggData
```

```
##
##  Missings in variables:
##  Variable Count
##        V2     2
##        V5     1
##        V6     2
##        V9     5
##       V14     1
```

Prior to this, we had only identified the 8th variable of V5 as missing, whereas now we can see which variables have a higher proportion of missing values. Further, by looking at the object that *aggr* created, we can see that V9 has the most variables missing, coming in at 5, and our favourite missing value in X5 was the only missing value for that variable- excellent.

Given that we have 14 variables in our dataset, havin 5 variables with at least 1 missing value isn't actually too worrying- and we can deal with this accordingly.

```
  wineData <- na.omit(wine)
  dim(wine)
```

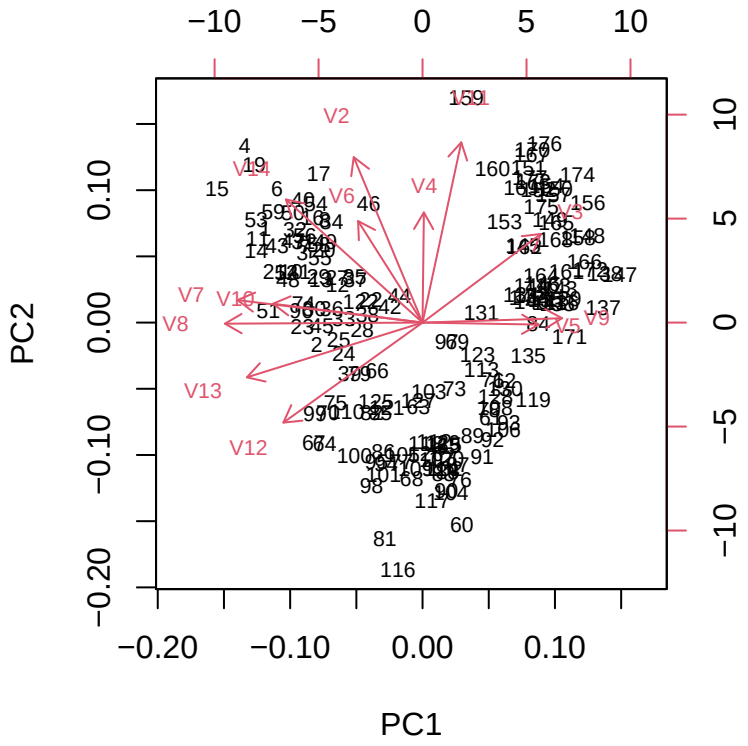
```
## [1] 178  14
```

```
  dim(wineData)
```

```
## [1] 168  14
```

The above two dimensions show us that by using *na.omit*, we have lost 10 observations, but that isn't a problem, there's still loads of data we can work with. Speaking of, let's perform that PCA we tried to use earlier.

```
  winePCA <- prcomp(wineData[,-1], scale=T)
  biplot(winePCA, cex=0.7)
```



We already knew to scale the dat, and I'm happy with the result. One could argue observation 115 is an outlier, but I', not going to remove it as it is fine in PC1 and not extreme in PC2. There is a decent spread in PC2, and we can see that no variables appear to be dominating the plot, which is to be expected after scaling.

Neural Networks

Before we can do anything fancy, we are going to split our data up into test and training sets. I'm going to split at a ratio of 75:25 just out of preferance.

```
index = createDataPartition(wineData$V1, p=0.75, list=F)
trainData = wineData[index,]
testData = wineData[-index,]
trainData$V1 <- as.factor(trainData$V1)
testData$V1 <- as.factor(testData$V1)
```

With the logisitics out the way, we can proceed using the *nnet* package. Further the results from this can be summarised using the *confusionMatrix()* function from the *caret* package.

```
numFolds =  trainControl(method="cv", number = 10, savePredictions = T)
grid = expand.grid(size=10, decay=0.1)
nnmodel = train(V1~., data=trainData, method="nnet", preProcess=c("center", "scale"),
                trControl=numFolds, tuneGrid=grid)
```

The output from the above code is quite long, but luckily, we can see a handy summary of it below:

```
nnmodel$results
```

```
##   size decay  Accuracy     Kappa AccuracySD    KappaSD
## 1   10   0.1 0.9525641 0.9287769 0.05466078 0.08224848
```

So we see we have a very high accuracy which is ideal. We can look at how well the

```
nnpred =  predict(nnmodel, testData[,-1])
confusionMatrix(as.factor(nnpred), testData$V1)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction  1  2  3
##          1 13  0  0
##          2  0 17  0
##          3  0  0 11
##
## Overall Statistics
##
##                Accuracy : 1
##                  95% CI : (0.914, 1)
##     No Information Rate : 0.4146
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 1
##
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: 1 Class: 2 Class: 3
## Sensitivity            1.0000   1.0000   1.0000
## Specificity            1.0000   1.0000   1.0000
## Pos Pred Value         1.0000   1.0000   1.0000
```

```
## Neg Pred Value         1.0000   1.0000   1.0000
## Prevalence             0.3171   0.4146   0.2683
## Detection Rate         0.3171   0.4146   0.2683
## Detection Prevalence   0.3171   0.4146   0.2683
## Balanced Accuracy      1.0000   1.0000   1.0000
```

It is clear from the above two results that the accuracy for training and data is superb. Note that I chose to run the *nnet* through Caret instead of on it's own, as it meant I could scale the data automatically and didn't have to mess around with things like *cbind()* and *rbind()*.

KNN

Recall we had previously omitted the values which were NA. We now use the *k-nearest neihbours approach*, by running the following:

```
wineKNN <- knnImputation(wine)
```

Note that the dimensions for this new dataset are exactly the same as that of the original. This is because the *knnimputation* function fills in each of these values by using the k-nearest-neighbours algorithm to find approximate values. We can now essentially just repeat exactly what we did earlier.

```
index = createDataPartition(wineKNN$V1, p=0.75, list=F)
trainData = wineKNN[index,]
testData = wineKNN[-index,]
trainData$V1 <- as.factor(trainData$V1)
testData$V1 <- as.factor(testData$V1)
numFolds =  trainControl(method="cv", number = 10, savePredictions = T)
grid = expand.grid(size=10, decay=0.1)
nnmodel = train(V1~., data=trainData, method="nnet", preProcess=c("center", "scale"),
                trControl=numFolds, tuneGrid=grid)
```

And finally, the confusion matrix:

```
nnpred =  predict(nnmodel, testData[,-1])
confusionMatrix(as.factor(nnpred), testData$V1)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction  1  2  3
##          1 14  0  0
##          2  0 18  0
##          3  0  0 12
##
## Overall Statistics
##
##                Accuracy : 1
##                  95% CI : (0.9196, 1)
##     No Information Rate : 0.4091
##     P-Value [Acc > NIR] : < 2.2e-16
##
```

```
##                   Kappa : 1
##
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: 1 Class: 2 Class: 3
## Sensitivity            1.0000   1.0000   1.0000
## Specificity            1.0000   1.0000   1.0000
## Pos Pred Value         1.0000   1.0000   1.0000
## Neg Pred Value         1.0000   1.0000   1.0000
## Prevalence             0.3182   0.4091   0.2727
## Detection Rate         0.3182   0.4091   0.2727
## Detection Prevalence   0.3182   0.4091   0.2727
## Balanced Accuracy      1.0000   1.0000   1.0000
```

So infact this is't all that much more accurate, but I would argue this isn't the better approach to take, using algorithms to approximate values for things isn't always going to work, so its best to just get rid of NA values and test on data you know is 100% accurate. Especially as it (certainly in this case) has only sacrificed ~1% of accuracy.