HarvardX Capstone Project: Biomechanical Features of Orthopedic Patients

Alec Banner

Introduction

The goal of this project was to build a machine learning algorithm capable of predicting the presence of abnormalities in the spine given physical measurements of the body. The data set utilised, **Biomechanical Features of Orthopedic Patients**, was selected as it bares relevence to my area of research. It is also a small dataset which should allow me to build and test a range of machine learning algorithms, without the challenges faced so far due to working on a small laptop.

Loading the Dataset

The dataset is imported from github and then the 'readr' package used to read the data from the .csv file.

```
## # A tibble: 6 x 7
##   pelvic_incidence `pelvic_tilt nu~ lumbar_lordosis~ sacral_slope pelvic_radius
##              <dbl>            <dbl>            <dbl>        <dbl>         <dbl>
## 1             63.0             22.6             39.6         40.5          98.7
## 2             39.1             10.1             25.0         29.0         114.
## 3             68.8             22.2             50.1         46.6         106.
## 4             69.3             24.7             44.3         44.6         102.
## 5             49.7              9.65            28.3         40.1         108.
## 6             40.3             13.9             25.1         26.3         130.
## # ... with 2 more variables: degree_spondylolisthesis <dbl>, class <chr>
```

The Dataset

The name of the 'pelvic_tilt numeric' column is long and contains spaces so will be renamed to just 'pelvic_tilt'.

The dimmensions of the dataset show that it contains 310 enteries (rows) and 7 columns

```
## [1] 310   7
```

```
## tibble [310 x 7] (S3: spec_tbl_df/tbl_df/tbl/data.frame)
##  $ pelvic_incidence        : num [1:310] 63 39.1 68.8 69.3 49.7 ...
##  $ pelvic_tilt             : num [1:310] 22.55 10.06 22.22 24.65 9.65 ...
##  $ lumbar_lordosis_angle   : num [1:310] 39.6 25 50.1 44.3 28.3 ...
##  $ sacral_slope            : num [1:310] 40.5 29 46.6 44.6 40.1 ...
##  $ pelvic_radius           : num [1:310] 98.7 114.4 106 101.9 108.2 ...
##  $ degree_spondylolisthesis: num [1:310] -0.254 4.564 -3.53 11.212 7.919 ...
```

```
##  $ class                   : chr [1:310] "Abnormal" "Abnormal" "Abnormal" "Abnormal" ...
##  - attr(*, "spec")=
##   .. cols(
##   ..   pelvic_incidence = col_double(),
##   ..   `pelvic_tilt numeric` = col_double(),
##   ..   lumbar_lordosis_angle = col_double(),
##   ..   sacral_slope = col_double(),
##   ..   pelvic_radius = col_double(),
##   ..   degree_spondylolisthesis = col_double(),
##   ..   class = col_character()
##   .. )
```

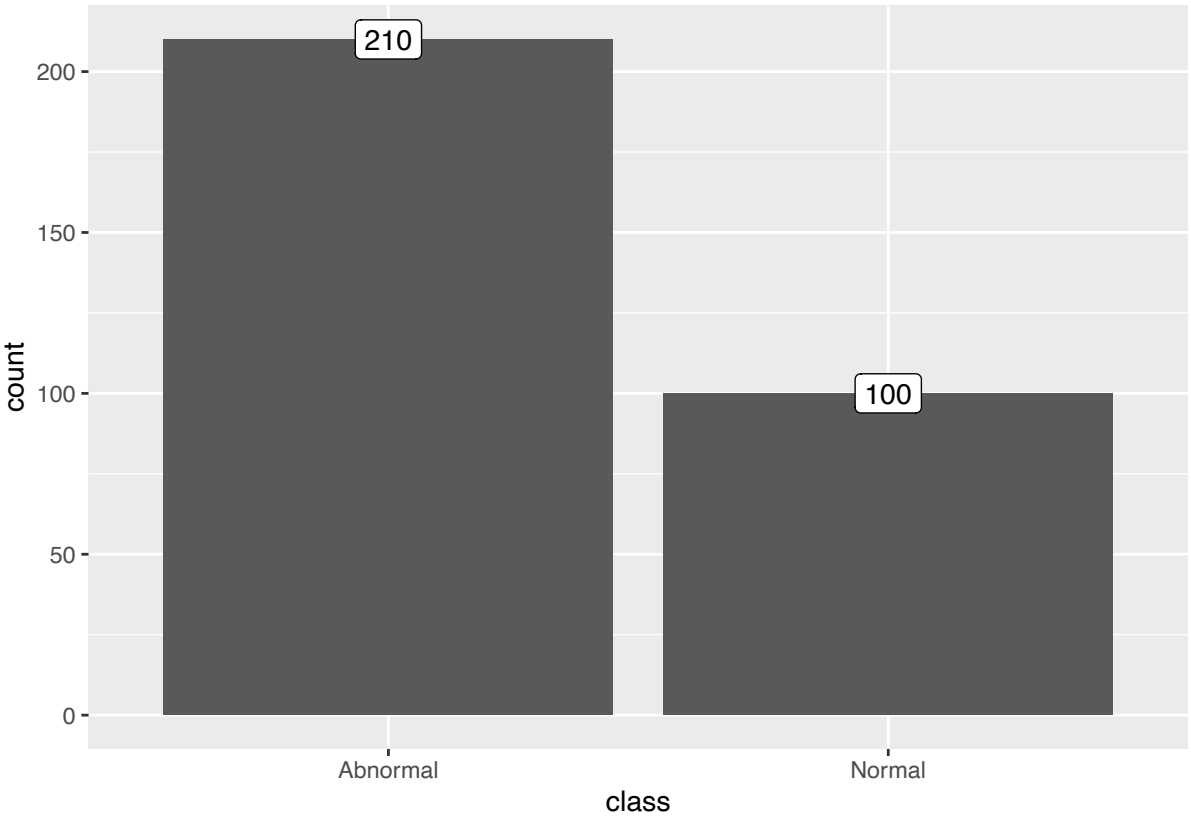
Of the 7 columns, 6 contain numeric data, these will be our factors for machine learning. The last column is a character string containing the 'class' stating whether the patient was 'Normal' or 'Abnormal' in their spine. The 6 factors are: pelvic_incidence, pelvic_tilt, lumbar_lordosis_angle, sacral_slope, pelvic_radius and degree_spondylolisthesis. These are all physical measurements of a patient which have been taken by a doctor. The 'class' was redefined as a factor with two levels, this will make predicting machine learning outcomes easier and also means that when the dataset is split into training and testing set, the 'createDataPartition' function will automatically maintain the prevelence of each factor during the split.

```
column_2C_weka$class <- as.factor(column_2C_weka$class)
```

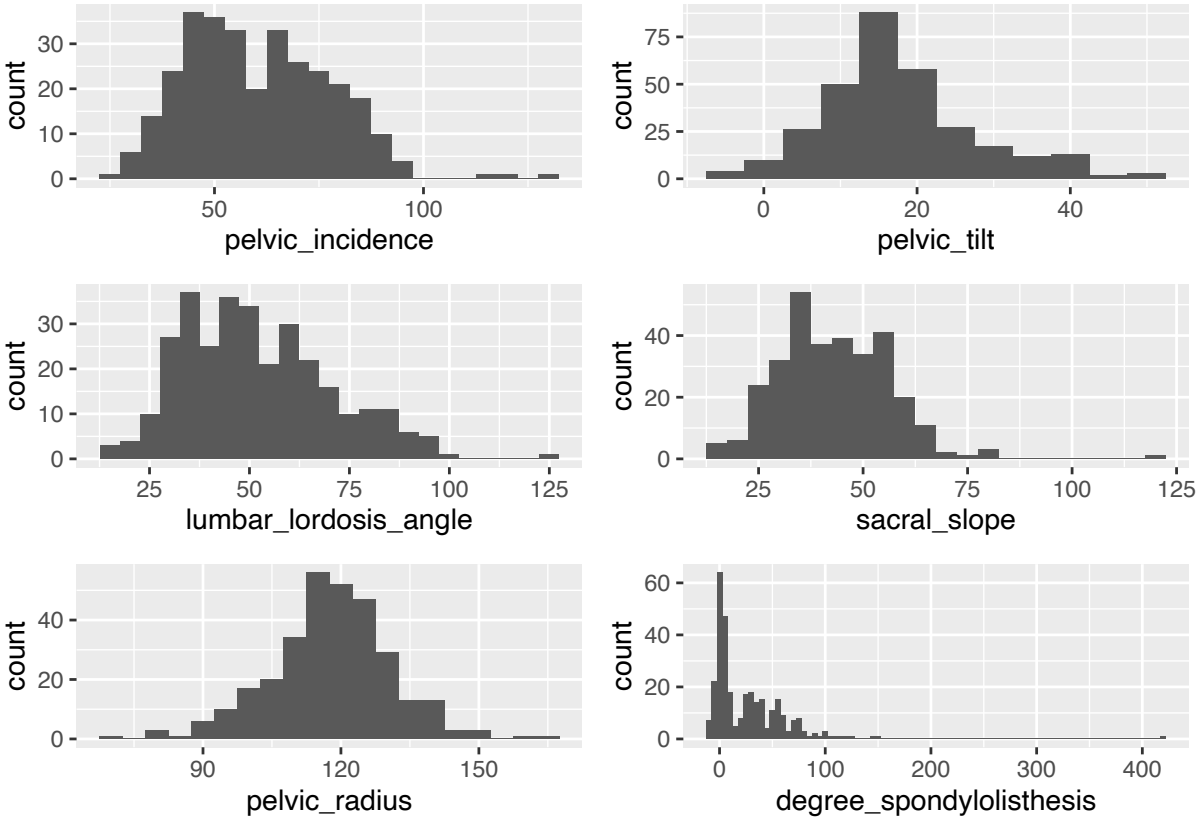
Finally, this is a dense dataframe and does not contain any NAs

```
##         pelvic_incidence               pelvic_tilt    lumbar_lordosis_angle
##                    FALSE                     FALSE                    FALSE
##             sacral_slope            pelvic_radius degree_spondylolisthesis
##                    FALSE                     FALSE                    FALSE
##                    class
##                    FALSE
```

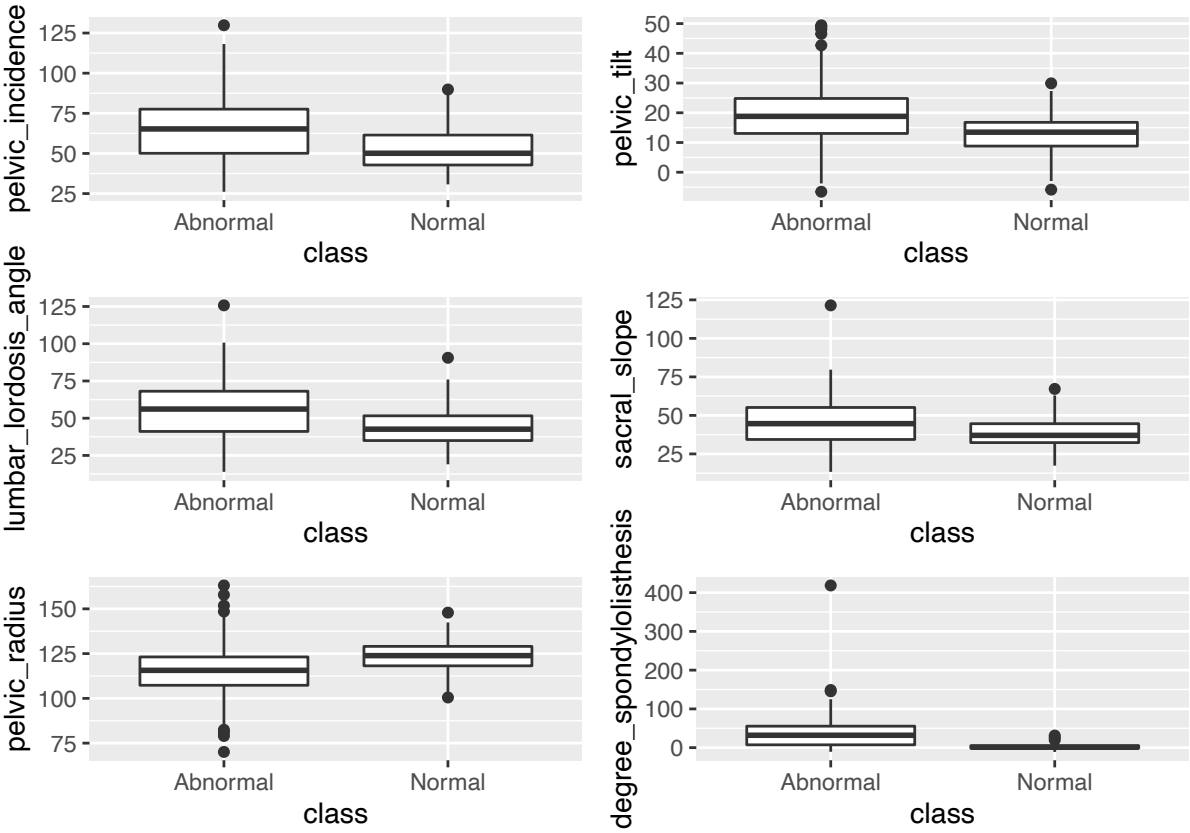
Exploring the Data



This plot shows that there are 210 'Abnormal' enteries and 100 'Normal' enteries in the dataset. Therefore, it is important to maintain this uneven prevelence when the data is split which will be done using the 'class' factors with the createDataPartition' function.



Histograms of each of the factors. These don't appear to be normally distributed which suggests that they are factors affected by 'class' as under normal circumstances they are measurements which we would predict would be normally distributed. Two of the factors, pelvic_tilt and pelvic_radius are however only slightly scewed from normal distribution so these may not be as stronger predictors and pelvic_incidence and degree_spondylolisthesis which are nearly bivarient normal, suggesting two normal distributionss, one for each factor.



Boxplots of each factor by class, show that for each factor the average of the two classes is different. This would suggest that each will have predictive power. For all factors other than degree-spondylolisthesis however, the interquartile ranges overlap, so separating the 2 factors completely could be hard.

Creating Training and Testing sets

With machine learning it is importand to separate the training and test data to prevent overtraining. The trianing set will be used for building and optimising the machine learning algorithms and the testing set only used to assess the final model. This dataset will be split 80/20 with 80% of the data going into the training set. This ratio was selected as the dataset is relatively small. There are no set rules for how to split data, but in a very large dataset a 90/10 split, the 10% training data is still likely to be representative of the overal dataset. Here the dataset is relatively small so 80/20 will hopefully allow the test_set to be representative of the overal data whilst maintaining enough data in the trainging set for the algorithms to learn from. the data partition is carried out on 'class' meaning that prevelance of each factor will be maintained during the split.

```
#createdatapartitionbased on class(factor) to maintain ratios
set.seed(1, sample.kind = "Rounding")
test_index <- createDataPartition(y = column_2C_weka$class, times = 1, p =  0.2, list = FALSE)
test_set <- column_2C_weka[test_index,]
data <- column_2C_weka[-test_index,]
#check the realative ratios are maintained
mean(test_set$class == "Normal")
```

```
## [1] 0.3225806
```

```r
mean(data$class == "Normal")
```

```
## [1] 0.3225806
```

```r
#save the partitioned data
save(test_set, file = "test_set.rdata")
save(data, file = "data.rdata")
```

Methods

Setting the seed

Whenever random number generation is used, seed will be set to 1. This is so the results are reproducible.

Creating Training and Validation sets

The Training data will undergo a second split in order to have a 'training' set and a 'validation' set. the 'training' set will be used to build and optimise the models and the 'validation' set used to assess how these preform and compare them. Again an 80/20 split will be used.

```r
#split data into training and validation sets
set.seed(1, sample.kind = "Rounding")
test_index <- createDataPartition(y = data$class, times = 1, p = 0.2, list = FALSE)
validation <- data[test_index,]
training <- data[-test_index,]
#check the ratios are maintained
mean(validation$class == "Normal")
```

```
## [1] 0.32
```

```r
mean(training$class == "Normal")
```

```
## [1] 0.3232323
```

```r
#save the trainging and validation datasets
save(validation, file = "validation.rdata")
save(training, file = "training.rdata")
```

The Caret package and TrainingControl

The 'caret' package will be used to create the machine learning algorithms. The caret package is good as it takes a large number of different algorithms and runs them using the same syntax. Caret packages 'train' function automatically carries out cross validation on the dataset to estimate the accuracy. We will change cross validation method from the default to use 50 rounds of bootstrapping using 50% of the data. More rounds and larger percentage of the data should give more accurate estimates of accuracy as the dataset is small. As the dataset is small the processing time should still be acceptable.

```r
#define cross validation method
control <- trainControl(method = "boot", number = 50, p = 0.5)
```

Logistic Regression

The first model uses a generalise linear model (glm). This model applies regression to catagorical data as if the data were continuous. This is applied here as class is a factor with 2 levels and therefore the outcomes can be thought of as catagorical with outcomes of either 'Normal' or 'Abnormal'

K Nearest Nieghbours

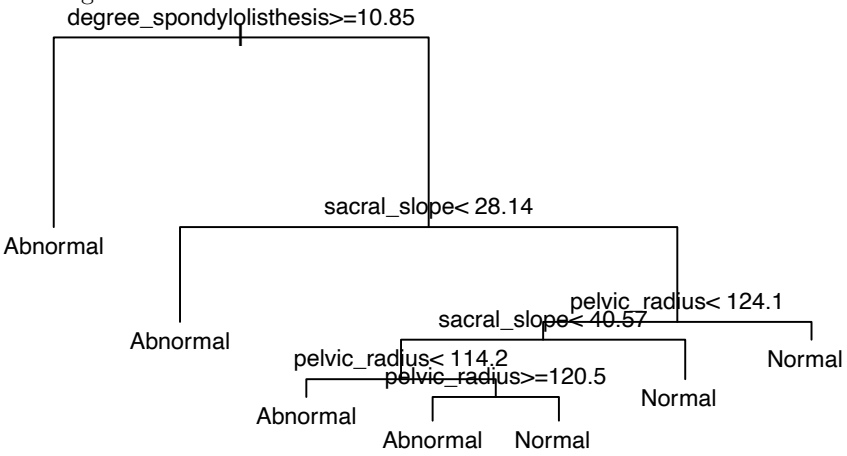
The second machine learning technique to be applied is K nearest neighbours (knn). This is a cross validation method which takes a point for which we want to predict the outcome, looks at the 'k' number of nearest points and takes the average of the outcomes, 'Normal' or 'Abnormal'. For this method the number of nearest neighbours to take into account for each prediction will be optimised by trying 'k's between 3 and 51. The optimum k will be used to test the final model.

Loess Function

Local weighted regression (Loess) is similar to knn however rather than taking the average of the 'k' nearest points, an estimate of the slope of those points is used. This can help give a smoother approximation than by knn. For this, the 'gamLoess' method will be used. Both degree and span can be optimised however degree will be fixxed as 1 (linear), and span optimised between 10% and 90% of the data.

Decision Tree

Decision trees are powerful and popular machine learning alorithms. They use the data to define rules, if this then that, else this. The data is plit multiple times until the rules pool the remaining data in nodes with the same outcomes. This is easier to understand with an example:



This is a single decision tree built using the 'rpart' package for the 'training' data. The first split looks at degree_spondylolisthesis, id the values is above 10.85 then it is defined as 'Abnormal' else the next condition, sacral_slope, is assessed. In this example 6 splits are made until all the data is defined as either 'Normal' or 'Abnormal'. The problem with decision trees is that the binary nature of the splits makes rough boundries which poorly represent the data.

Random Forests

To overcome this algorithms will be built using the 'randomForest' function. This builds a large number of decision trees and averages them into the final model making it much smoother and more flexible. The method used here will be optimised for 'mtry' which is the number of variables sampled as candidates foreach split, with integers between 1 and 10. As a large number of trees are averaged, it is not possible to visualise trees in the same way as a decision tree, the imporance of each factor can be assessed however. This looks at how often an individual factor is used in trees. A factor used more often is more important than one used less often.

Gradient Boosted Model

The final model is based on gradient boosting. Gradient boosting is similar to random forests in that they rely on the building of many decision trees, however in gradient boosting each tree is assessed. The next tree is built to try and improve on the previous tree by classifying observations which were not well predicted by the previous tree. The final model is a wieghted ensemble of all the previous trees. For this model the 'gbm' method will be used and optimised for: the number of trees built in 100s between 100 and 500; the interaction depth (number of splits) in each tree between 4 and 7. 6 is the default value and thought to generally perform well; shrinkage (learning rate) will be set to 0.01 as lower numbers can take a long time to process; and the minimum nodesize assessed for 3, 5 and 10, this is the minimum number of observations at each node to allow a split, 10 is typically used but small datasets may work better with smaller minnodes.

Ensemble Model

Finally and ensemble will be built by vote. Ensemble models aim to gain a better prediction by taking into account lots of models to predict the final outcome. In this case, for each observation the prediction of each algorithm is taken into account. The outcome with the most 'votes' will be the final outcome, either 'Normal' or 'Abnormal'.

Final model

The final model will be selected as the best performing of the model described. It will be trained on the entire training set (data) using the parameters optimised on the training and validation sets. It will then be used to make predictions on the test_set and the accuracy determined.

Results

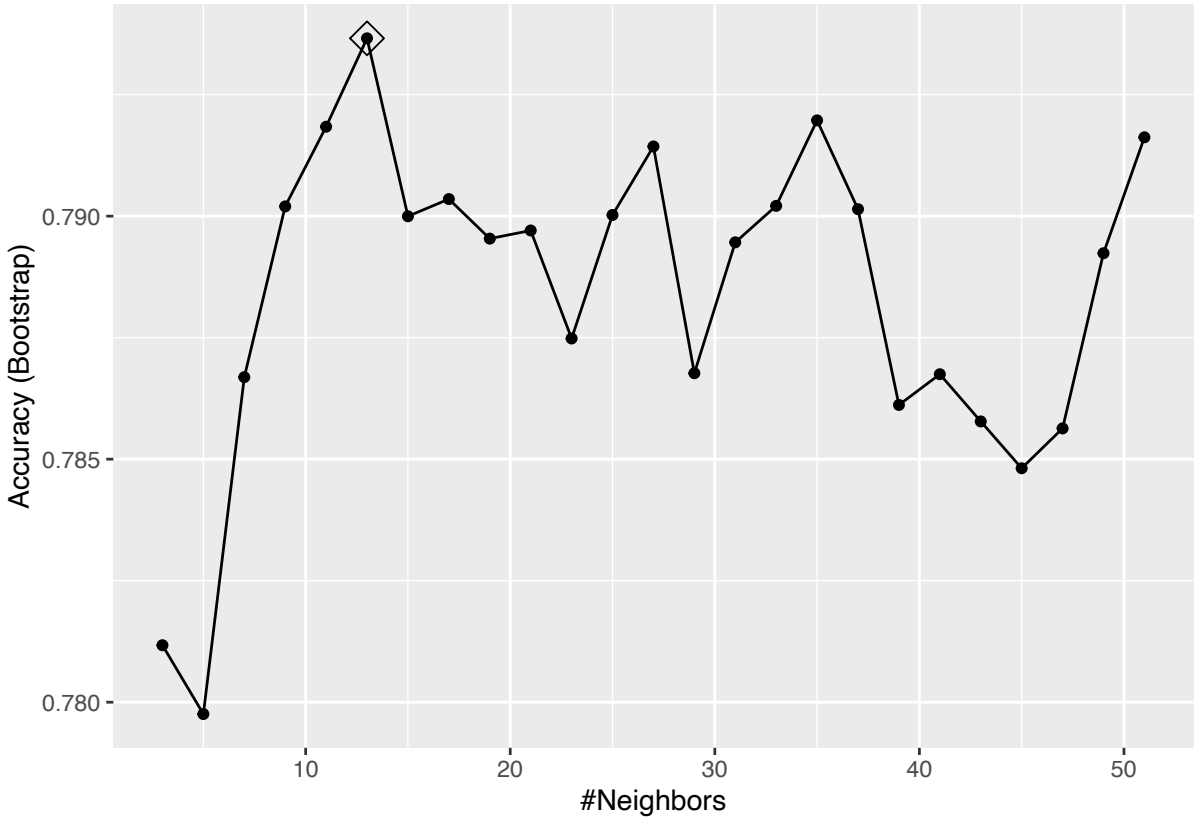
GLM

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction Abnormal Normal
##   Abnormal       27      3
##   Normal          7     13
##
##                Accuracy : 0.8
##                  95% CI : (0.6628, 0.8997)
##     No Information Rate : 0.68
##     P-Value [Acc > NIR] : 0.04366
```

```
##
##                  Kappa : 0.569
##
##  Mcnemar's Test P-Value : 0.34278
##
##             Sensitivity : 0.7941
##             Specificity : 0.8125
##          Pos Pred Value : 0.9000
##          Neg Pred Value : 0.6500
##              Prevalence : 0.6800
##          Detection Rate : 0.5400
##    Detection Prevalence : 0.6000
##       Balanced Accuracy : 0.8033
##
##        'Positive' Class : Abnormal
##
```

The glm model gives an accuracy of 0.8.

KNN



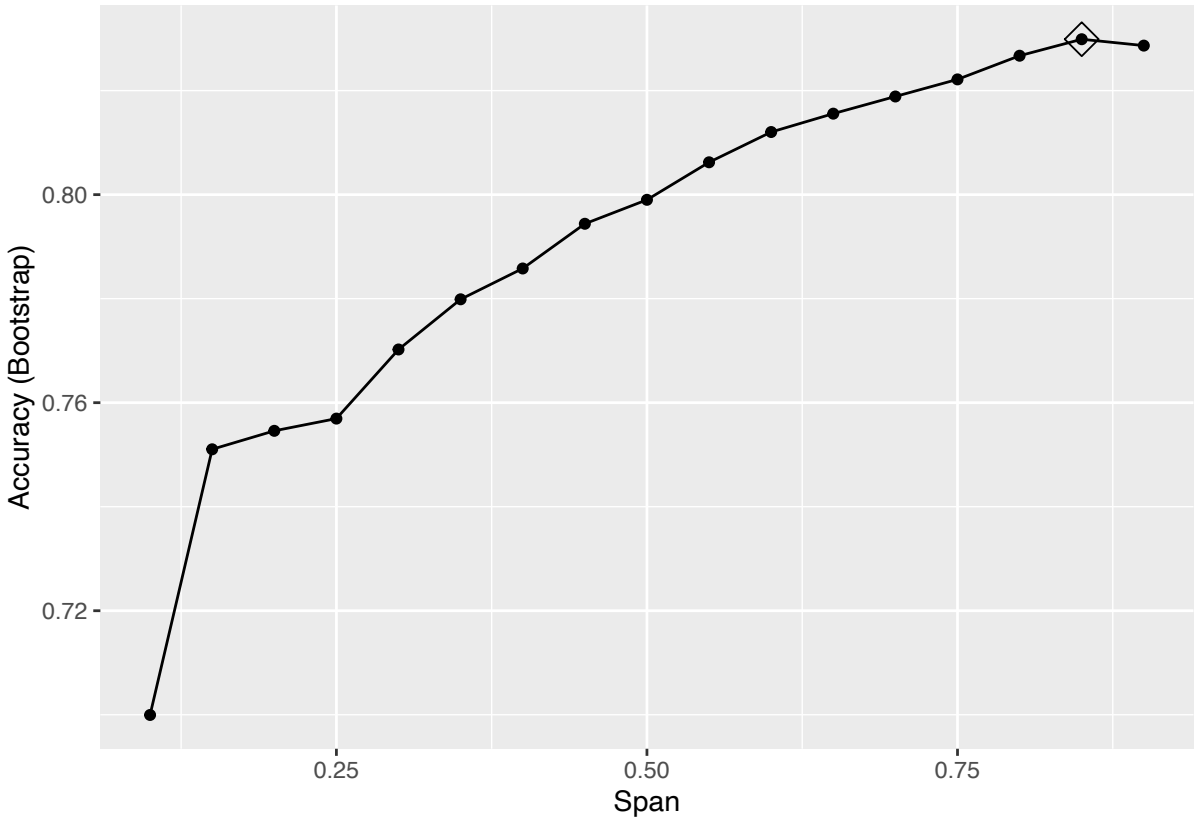
From the plot of number of neighbours against estimated accuracy, we can see that a k of 13 gives the highest accuracy, therefore this is the value which will be used in the final model.

```
## Confusion Matrix and Statistics
##
```

```
##           Reference
## Prediction Abnormal Normal
##   Abnormal       26      3
##   Normal          8     13
##
##                Accuracy : 0.78
##                  95% CI : (0.6404, 0.8847)
##     No Information Rate : 0.68
##     P-Value [Acc > NIR] : 0.08318
##
##                   Kappa : 0.5331
##
##  Mcnemar's Test P-Value : 0.22780
##
##             Sensitivity : 0.7647
##             Specificity : 0.8125
##          Pos Pred Value : 0.8966
##          Neg Pred Value : 0.6190
##              Prevalence : 0.6800
##          Detection Rate : 0.5200
##    Detection Prevalence : 0.5800
##       Balanced Accuracy : 0.7886
##
##        'Positive' Class : Abnormal
##
```

Making predictions on the validation set, we estimate an accuracy of 0.78.

gamLoess



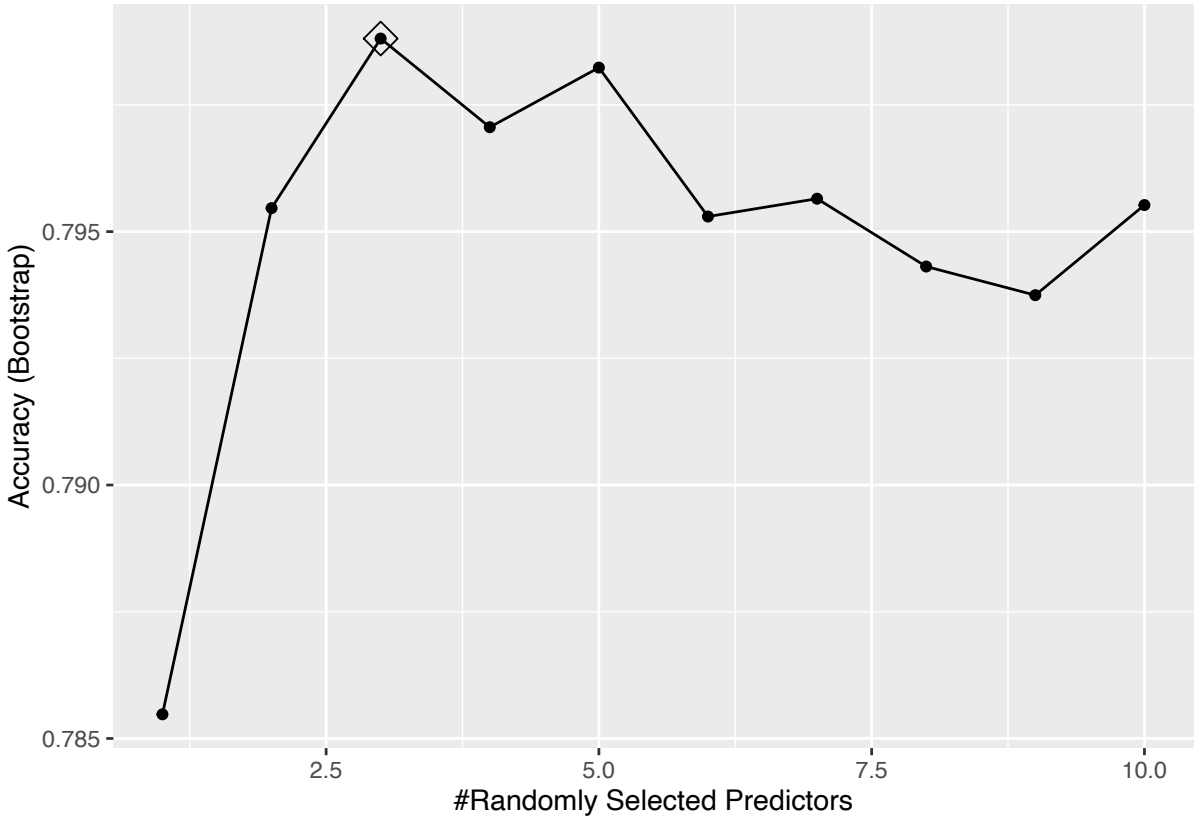
From the plot of span against estimated accuracy we can see that 85% gives the highest accuracy and will be used in the final model.

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction Abnormal Normal
##   Abnormal       28      3
##   Normal          6     13
##
##                Accuracy : 0.82
##                  95% CI : (0.6856, 0.9142)
##     No Information Rate : 0.68
##     P-Value [Acc > NIR] : 0.02057
##
##                   Kappa : 0.606
##
##  Mcnemar's Test P-Value : 0.50499
##
##             Sensitivity : 0.8235
##             Specificity : 0.8125
##          Pos Pred Value : 0.9032
##          Neg Pred Value : 0.6842
##              Prevalence : 0.6800
##          Detection Rate : 0.5600
```

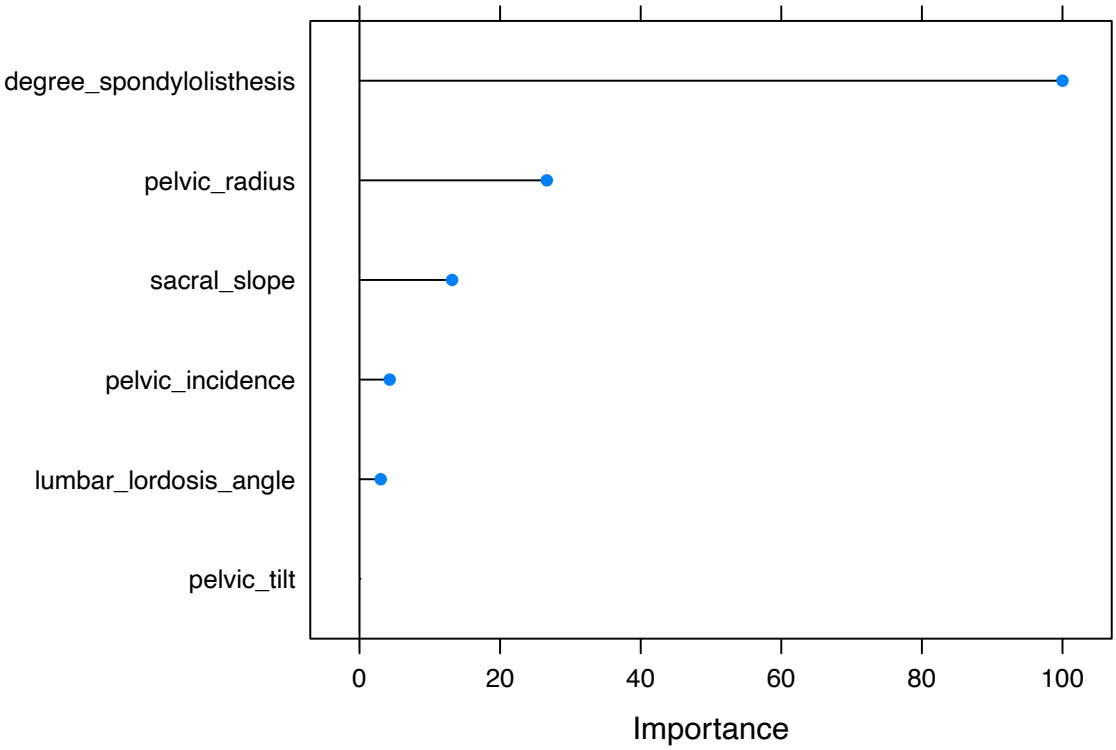
```
##    Detection Prevalence : 0.6200
##       Balanced Accuracy : 0.8180
##
##        'Positive' Class : Abnormal
##
```

The estimated accuracy of gamLoess on the validation set is 0.82.

Random Forest



From the plot of mtry against accuracy we can see that a value of 3 gives the highest estimated accuracy and will be used in the final model.



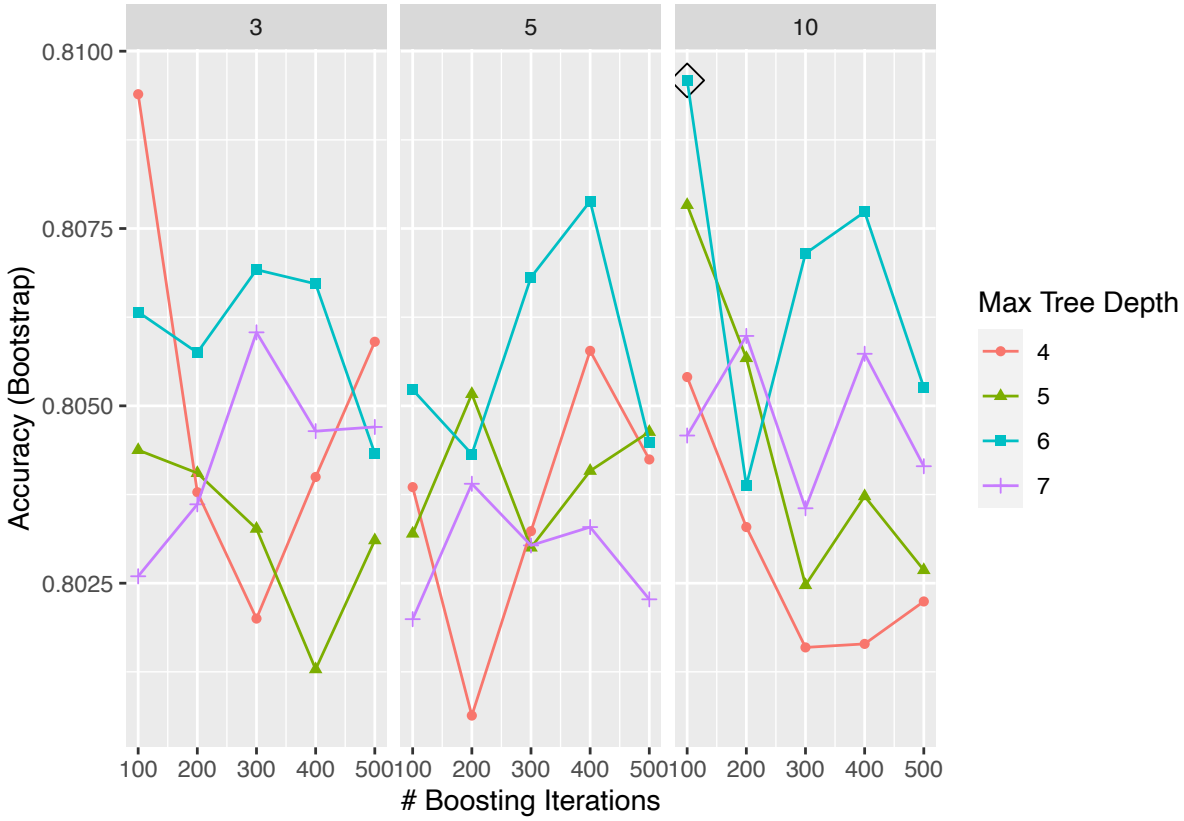
By plotting the importance of each factor in the final model we can see that degree_spondylolisthesis is by far the most important, followed by pelvic_radius and sacral_slope with pelvic_tilit and lumbar_lordosis_angle being relatively unimportant.

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction Abnormal Normal
##   Abnormal       28      3
##   Normal          6     13
##
##                Accuracy : 0.82
##                  95% CI : (0.6856, 0.9142)
##     No Information Rate : 0.68
##     P-Value [Acc > NIR] : 0.02057
##
##                   Kappa : 0.606
##
##  Mcnemar's Test P-Value : 0.50499
##
##             Sensitivity : 0.8235
##             Specificity : 0.8125
##          Pos Pred Value : 0.9032
##          Neg Pred Value : 0.6842
##              Prevalence : 0.6800
##          Detection Rate : 0.5600
##    Detection Prevalence : 0.6200
##       Balanced Accuracy : 0.8180
##
```

```
##         'Positive' Class : Abnormal
##
```

The esitmated accuracy of the final rf model is 0.82.

GBM



The plots of GBM tuning show that the optimised model used 100 trees, with and interaction depth of 6 and 10 nodes.

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction Abnormal Normal
##   Abnormal       31      6
##   Normal          3     10
##
##                Accuracy : 0.82
##                  95% CI : (0.6856, 0.9142)
##     No Information Rate : 0.68
##     P-Value [Acc > NIR] : 0.02057
##
##                   Kappa : 0.5648
##
##  Mcnemar's Test P-Value : 0.50499
##
```

```
##             Sensitivity : 0.9118
##             Specificity : 0.6250
##          Pos Pred Value : 0.8378
##          Neg Pred Value : 0.7692
##              Prevalence : 0.6800
##          Detection Rate : 0.6200
##    Detection Prevalence : 0.7400
##       Balanced Accuracy : 0.7684
##
##        'Positive' Class : Abnormal
##
```

The accuracy of the final gbm model on the validation set is 0.82.

Ensemble Model

```
#Ensemble Model
fits <- list(train_rf, train_gamloess, train_glm, train_knn, train_gbm)
pred <- sapply(fits, function(object){
  predict(object, validation)
})
colMeans(pred == validation$class)
```

```
## [1] 0.82 0.82 0.80 0.78 0.82
```

```
x <- rowMeans(pred == "Abnormal")
ensemble_vote <- ifelse(x>0.5, "Abnormal", "Normal")
ensemble_acc <- mean(ensemble_vote == validation$class)

#Ensemble model 2
fits <- list(train_gamloess, train_rf, train_gbm)
pred <- sapply(fits, function(object){
  predict(object, validation)
})
x <- rowMeans(pred == "Abnormal")
ensemble_vote <- ifelse(x>0.5, "Abnormal", "Normal")
ensemble_2_acc <- mean(ensemble_vote == validation$class)
```

The ensemble model built using the average vote across all models gave and overall accuracy of 0.8. This model performs worse than 3 of the models individually. Remaking the enemble to only include the gamLoess model, rf model and gbm model which all preformed better than the first ensemble giving an accuracy of 0.88.

The second ensemble model will therefore be used as the final model.

Final Model

The final model uses an ensemble vote model with: gamLoess with a span of 0.85, rf with a mtry of 3, and gbm with 100 trees, an interaction depth of 6 and minimum node size of 10. This gives an accuracy on the test_set, when training on data using the given parameters, of 0.9193548.

Conclusion

This project successfully evaluated the estimated accuracy of 5 machine learning algorithms for their ability to predict normailty from the **Biomechanical Features of Orthopedic Patients** dataset. A final ensemble model was built using gamLoess, randomForest and Gradient Boosted models with optimised parameters. This final model had an accuracy of 0.9193548 on the test_set. This appears to be a good model. Models which are not perfect can always be improved by further optimising the algorithms or assessing more machine learning methods for inclusion in the final ensemble model. One of the limitations of this project was the small dataset possibly gives a poor representation of the overall population. By including the measurements from more patients a more representative model could be built.