Normal vs. Abnormal Classifications of Biomechanical Features of Orthopaedic Patients

ltja

13 December, 2020

Summary

This report, a HarvardX: PH125.9x Capstone: Data Science course requirement, showcases a classification (Normal vs. Abnormal) process of three hundred and ten (310) Orthopaedic patients based on their biomechanical data. The cleaned data, sourced from kaggle, features six biomechanical attributes derived from the shape and orientation of the pelvis and lumbar spine of each patient. Some of the available data were put aside into a validation set, while the rest was further split into a test set and a training set, which were then analyzed using a number of different classification models (Linear, k-Nearest Neighbors, Naive Bayes, Decision Tree, Random Forest, and an Ensemble of all of them), resulting in varying degree of accuracy, sensitivity, and specificity. The aim of this project is two-fold: first, to pick a model that will produce the most accurate classification without compromising sensitivity; second, to review whether the "curse of dimensionality" (texbook, chapter 31[1]) plays a role in our analysis – which model will be affected, which won't, and can an accurate result still be yielded by using only one selected determining predictor? The result of this project found that Decision Tree yields the most accurate result for this dataset at over 80% accuracy and over 95% sensitivity. The analysis also found that using the one most important predictor (degree_spondylolisthesis), the model still yields a decently accurate result at over 77% and almost 81% sensitivity. To conclude, we find Decision Tree fits the dataset nicely. So much so that it would be sufficient to use one predictor, if circumstances demand. However, as outlined in more details in the report, the dataset is admittedly quite small. As such, for bigger datasets, a more robust model such as Random Forest may end up yielding stronger results.

1. INTRODUCTION

The Choose-Your-Own-Project of HarvardX: PH125.9x Capstone: Data Science allows wide latitude to students to demonstrate course understanding of machine learning. In response, I chose the "Biomechanical features of orthopedic patients" data set from kaggle because of its manageable size that is within my laptop capacity, its csv format (the original data sourced from UCI come in .dat and .arff formats), and a personal experience that involves a knee surgery. As a beginner, my goal is to present a straightforward, yet robust machine learning project that demonstrates my understanding of the course materials. Choosing a no-frills dataset helps me achieve that. Without the processing time burden imposed by big data on small capacity computer and without the need to convert the data into a more readable format (like from .arff to .csv), I was able to explore my interest in this orthopaedic dataset and focus specifically on the machine learning models. The following sections outline more details of the data and the steps involved in the analysis.

Data Description

The dataset contains biomechanical information of 310 patients. Each patient's information is laid out in a row of seven columns - six columns outlining six attributes of the patient's movement and the seventh column

[1] https://rafalab.github.io/dsbook/examples-of-algorithms.html

outlining his / her normal vs. abnormal classification.

Before we get into the attributes' details (which will be used as predictors in this machine learning exercise), it would be a good idea to first understand why these data were collected at the first place. The relevant papers[2] listed in the data source talk about "sagittal balance." In anatomy, sagittal plane divides the body into its left and right parts.[3] In Orthopaedic, this specifically refers to the alignment of the head, spine, pelvis, and legs. When things are out of alignment, the sagittal imbalance may show up in syndromes such as the hunched back syndrome, resulting in chronic fatigue and pain.[4]

For the past decades, Orthopaedists have been trying to understand the physiological relationships of the different parts of the sagittal plane. The complexity arises from the fact that "these anthropometrical parameters were very scattered because of human diversity and, therefore, it seemed difficult to define what is normal in the upright posture for a specific subject."[5] In other words, because of the diversity in human body shape, a spinal curve at a certain angle may be considered normal for me, but abnormal for you.

Orthopaedists have considered the six attributes as the determinants of whether someone's sagittal alignment is considered normal or abnormal. Although each attribute has its own normal range (e.g., pelvic incidence normal range is between 43 and 62 degrees[6]), determining whether a patient is normal or abnormal is not as simply as looking at each predictor and finding out whether each of their attributes is within the normal range. The complexity arises from the fact that one abnormal parameter may compensate for another, resulting in a somewhat balanced sagittal plane. This is where machine learning algorithms come in and take in all six predictors to determine a patient's normal vs. abnormal status. (Note that in this project, we'll also look at whether there are predictors that stand out and can be used on their own to make accurate predictions.)

A quick description of the six biomechanical attributes:

1. Pelvic incidence is defined "as the angle between the line perpendicular to the sacral plate at its midpoint, and the line connecting this point to the axis of the femoral heads"[7]
2. Pelvic tilt is "the orientation of the pelvis in respect to the thighbones and the rest of the body."[8]
3. Lumbar lordosis is defined as "abnormal inward curvature of the lumbar spine"[9]
4. Sacral slope is defined as "the angle between the horizontal and the sacral plate" [10]
5. Pelvic radius is defined as "the distance from the hip axis to the posterior-superior corner of the S1 endplate; the hip axis was located in the middle between the two femoral bead mid-points"[11]
6. Spondylolisthesis is "a spinal condition that causes lower back pain. It occurs when one of your vertebrae, the bones of your spine, slips out of place onto the vertebra below it."[12]

Project Goal

The goal of this project is two-fold: first, to pick a model that will produce the most accurate classification without compromising sensitivity; second, to review the importance of the different predictors and determine whether an accurate result can still be yielded with few selected predictors.

Steps

The following key steps were performed to explore the data, develop models, and pick the final model:

1. The dataset has been downloaded from kaggle and uploaded onto my Github. So, the first step is to read the file, and using the view function, ensure that all of the data are there.

[2]http://archive.ics.uci.edu/ml/datasets/vertebral+column
[3]https://en.wikipedia.org/wiki/Sagittal_plane
[4]https://www.columbiaspine.org/condition/sagittal-imbalance/
[5]https://www.ncbi.nlm.nih.gov/pmc/articles/PMC3489325/
[6]https://www.ncbi.nlm.nih.gov/pmc/articles/PMC2200679/
[7]https://www.ncbi.nlm.nih.gov/pmc/articles/PMC3489325/
[8]https://en.wikipedia.org/wiki/Pelvic_tilt
[9]https://en.wikipedia.org/wiki/Lordosis
[10]https://thejns.org/spine/view/journals/j-neurosurg-spine/23/6/article-p754.xml
[11]https://www.ncbi.nlm.nih.gov/pmc/articles/PMC4877554/
[12]https://my.clevelandclinic.org/health/diseases/10302-spondylolisthesis

2. Put aside 10% of the data to create a validation set. The 10%:90% split was chosen, as opposed to 50%:50% or even 20%:80% because the size of the dataset is fairly small, so we would want a significant portion of it to be available for training. (Note: the dataset will be further split into training set and test set, leaving about 80% of the available data for training. To address any possible high parameter variance due to the small dataset, I used cross validation when training the models[13])

3. Split the dataset further into train and test sets; the validation set would not be used until the very end. Then, review the new datasets (in terms of dimension, variables presented per column and row, etc.) to gain an understanding of them and ensure that the split is correct.

4. Explore the dataset by reviewing their features and impacts on the classification. Graphing techniques will also be used to communicate the findings.

- What is the ratio between abnormal and normal? Is the ratio consistent between the train and test sets? Will this be considered an imbalanced dataset that will affect each group's accuracy?
- Are there any missing data?
- What's the split between normal vs. abnormal within each predictor? Is there a predictor which determines more abnormality than others? How is that predictor (if there's one) correlates with other predictors?

5. Fit a number of different models, including an ensemble of them all, to find the most accurate one. Additionally, because we are dealing with medical data, sensitivity is also important, as we do not want to miss a positive case. The data will then be prepped by searching for any predictors that have near zero variance and since we'll be using the caret package, the train set will be subjected to trainControl cross validation method. The section below will outline why each model is chosen for this project.

6. Review and compare the results produced by the different models, then pick the strongest model for the dataset. Apply the model on the validation set.

7. Analyze the validation set result and review the model's strengths and weaknesses, as well as its validity for future use. End process.

2. ANALYSIS / METHODS

The following section outlines the step-by-step process and techniques used to explore the data, gain insights from the exploration, select models to train the dataset, and choose the final model that was to be tested against the validation set. The code that was used, the results, and any charts that help with data visualization will be presented below. (Note: A huge chunk of the code below, although self-developed, is inspired by the textbook[14]. Any other references used will be noted accordingly.)

Step 1: Read the dataset

To ensure consistency, I've downloaded the dataset from kaggle and uploaded it onto my Github page. This report will read the file from my Github page.

The code is as the following:

```r
if(!require(tidyverse)) install.packages("tidyverse", repos = "http://cran.us.r-project.org")
if(!require(caret)) install.packages("caret", repos = "http://cran.us.r-project.org")
if(!require(data.table)) install.packages("data.table", repos = "http://cran.us.r-project.org")
if(!require(gridExtra)) install.packages("gridExtra", repos = "http://cran.us.r-project.org")

library(tidyverse)
library(caret)
```

[13]https://stackoverflow.com/questions/13610074/is-there-a-rule-of-thumb-for-how-to-divide-a-dataset-into-training-and-validatio

[14]https://rafalab.github.io/dsbook/

```
library(data.table)
library(gridExtra)
```

```
url <- "http://github.com/LTjaCS50/CYO/blob/main/column_2C_weka.csv"
patients <- read_csv("https://raw.githubusercontent.com/LTjaCS50/CYO/main/column_2C_weka.csv")
patients <- as.data.frame(patients)

# Check that column names are correct
head(patients)
##   pelvic_incidence pelvic_tilt numeric lumbar_lordosis_angle sacral_slope
## 1         63.02782           22.552586              39.60912     40.47523
## 2         39.05695           10.060991              25.01538     28.99596
## 3         68.83202           22.218482              50.09219     46.61354
## 4         69.29701           24.652878              44.31124     44.64413
## 5         49.71286            9.652075              28.31741     40.06078
## 6         40.25020           13.921907              25.12495     26.32829
##   pelvic_radius degree_spondylolisthesis    class
## 1      98.67292                -0.254400 Abnormal
## 2     114.40543                 4.564259 Abnormal
## 3     105.98514                -3.530317 Abnormal
## 4     101.86850                11.211523 Abnormal
## 5     108.16872                 7.918501 Abnormal
## 6     130.32787                 2.230652 Abnormal

# Check that all of the rows of data are there
dim(patients)
## [1] 310   7
```

Step 2: Create validation set.

After confirming that the file is read correctly, set aside 10% of the dataset to be used as the validation set. The validation set will not be touched again until the very end.

```
# Validation set will be 10% of patients data
set.seed(1, sample.kind="Rounding") # if using R 3.5 or earlier, use `set.seed(1)`
test_index <- createDataPartition(y = patients$class, times = 1, p = 0.1, list = FALSE)
cyo <- patients[-test_index,]
temp <- patients[test_index,]
validation_set <- temp

# Make sure the split is correct
dim(cyo)
## [1] 279   7
dim(validation_set)
## [1] 31  7
```

Step 3: Create training set and test set

After confirming that the validation set (31 rows) is 10% of the total set (310 rows), further split the remaining data to create the training set and the test set. We will keep the 10%:90% split ratio here, with 10% being the test set and 90% the training set.

```
# Test will be 10% of cyo data
set.seed(1, sample.kind="Rounding") # if using R 3.5 or earlier, use `set.seed(1)`
test_index <- createDataPartition(y = cyo$class, times = 1, p = 0.1, list = FALSE)
```

```r
train_set <- cyo[-test_index,]
test_set <- cyo[test_index,]

# Make sure the split is correct
dim(train_set)
## [1] 251   7
dim(test_set)
## [1] 28  7
```

Step 4: Explore the dataset

After checking from the composition that the training / test datasets were split correctly, we can begin exploring the dataset.

First, because this is a classification task, I would like to see the normal vs. abnormal ratio and figure out whether one class proportionally overwhelms the other. If so, do we have an imbalanced dataset (as discussed in the textbook[15]) that cause our result to be highly accurate at the detriment of one class. For example, the textbook talks about a data split between male / female that is overwhelmingly male. As a result, when tasked to predict the gender, the machine would be more inclined to pick male, which gives an overall high accuracy. But, when one looks deeper into the prediction accuracy within each class, one would see that accuracy within the female class was very low. With this in mind, we'll ask the same question of our dataset – is it plagued with data imbalance – and come up with the answer later on.

Let's look at the ratio of normal vs. abnormal patients within the training dataset.

```r
# Ratio of normal vs. abnormal in train_set
normal_train <- nrow((train_set %>% filter(class=="Normal")))
abnormal_train <- nrow((train_set %>% filter(class=="Abnormal")))
normal_train / abnormal_train
## [1] 0.4764706
normal_train / (normal_train + abnormal_train)
## [1] 0.3227092
```

From the above, we see that the normal vs. abnormal ratio in the training set is about 47%. Is this consistent with the ratio in the test set? Is this data imbalance?

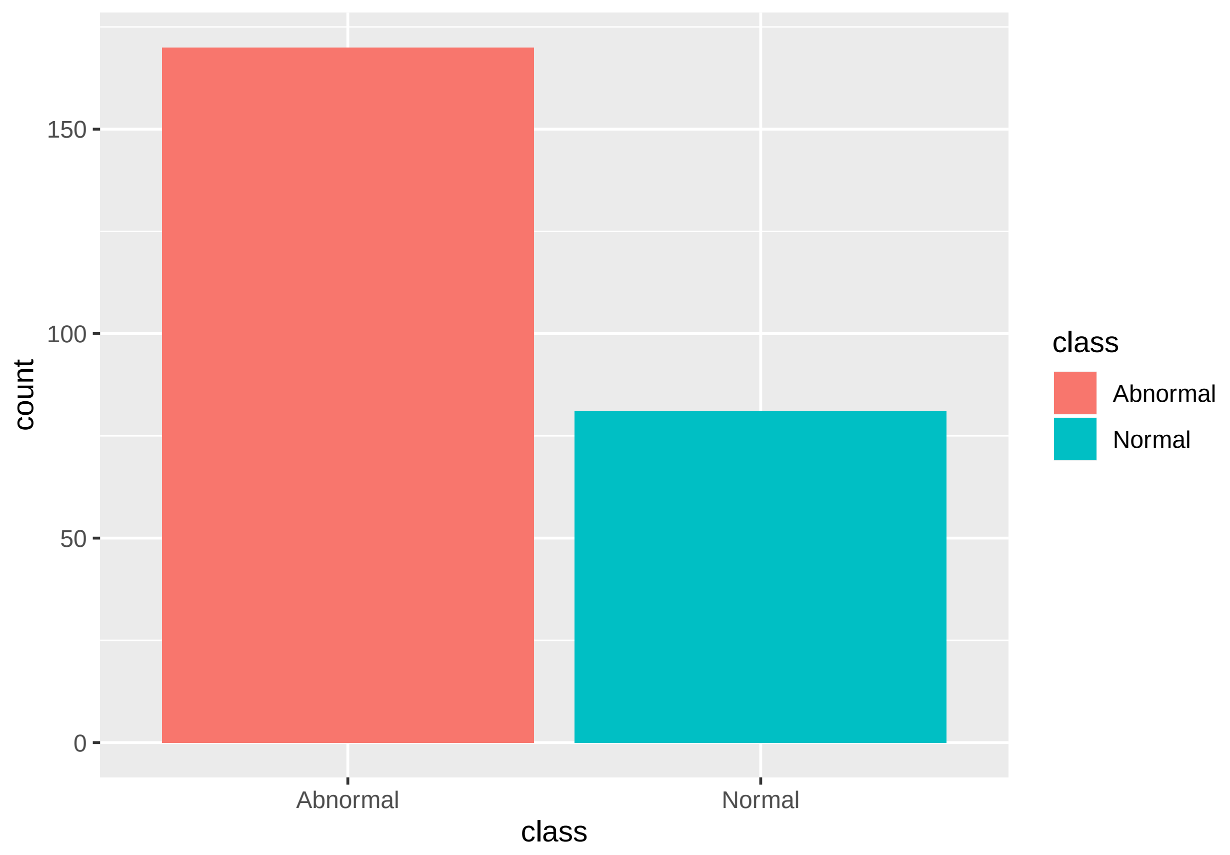
```r
normal_test <- nrow((test_set %>% filter(class=="Normal")))
abnormal_test <- nrow((test_set %>% filter(class=="Abnormal")))
normal_test / abnormal_test
## [1] 0.4736842
normal_test / (normal_test + abnormal_test)
## [1] 0.3214286
```

From the above, we see that yes, the normal vs. abnormal ratio stays consistent in the test set. We'll answer the data imbalance question later after we pick a model.

The class distribution as a graph:

```r
# Graph the distribution
ggplot(train_set, aes(class)) + geom_bar(aes(fill = class))
```

[15] https://rafalab.github.io/dsbook/introduction-to-machine-learning.html



In addition to the class distribution, there are a number of other aspects of the data that we should know. Such as:

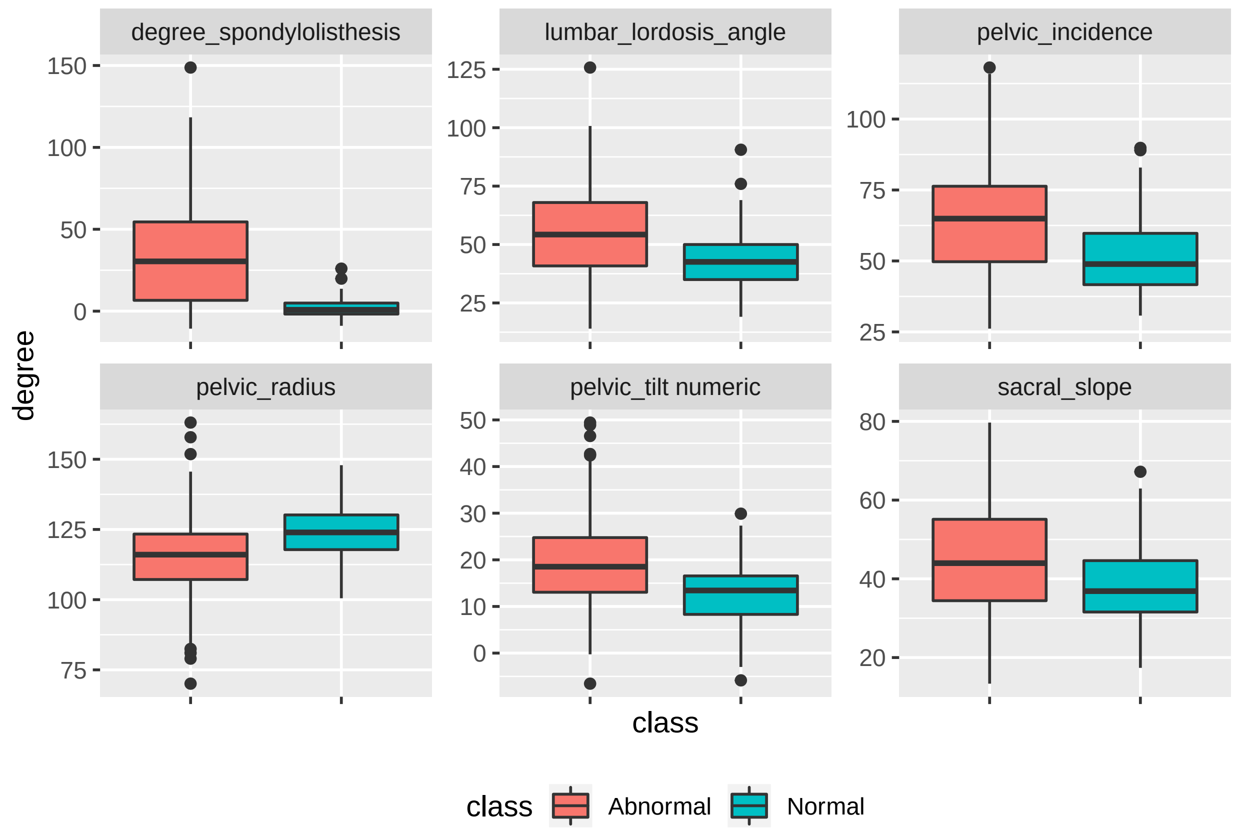
- Is there any missing information in the dataset? Answer: There is no missing information in the data set (zero NAs).

```
# Are there any NAs?
sum(is.na(patients))
## [1] 0
```

- What does the training set look like, value rangewise? This can be answered by looking at the data's summary.

```
# What's the summary for the training set?
summary(train_set)
##  pelvic_incidence pelvic_tilt numeric lumbar_lordosis_angle  sacral_slope
##  Min.   : 26.15   Min.   :-6.555      Min.   : 14.00        Min.   :13.37
##  1st Qu.: 45.56   1st Qu.:10.600      1st Qu.: 36.84        1st Qu.:33.20
##  Median : 57.52   Median :16.061      Median : 48.00        Median :42.16
##  Mean   : 59.92   Mean   :17.328      Mean   : 51.38        Mean   :42.59
##  3rd Qu.: 72.20   3rd Qu.:22.200      3rd Qu.: 62.78        3rd Qu.:52.65
##  Max.   :118.14   Max.   :49.432      Max.   :125.74        Max.   :79.70
##  pelvic_radius    degree_spondylolisthesis    class
##  Min.   : 70.08   Min.   :-10.676          Length:251
##  1st Qu.:110.92   1st Qu.:  1.219          Class :character
##  Median :118.36   Median : 10.443          Mode  :character
##  Mean   :118.11   Mean   : 23.848
##  3rd Qu.:125.58   3rd Qu.: 39.454
##  Max.   :163.07   Max.   :148.754
```

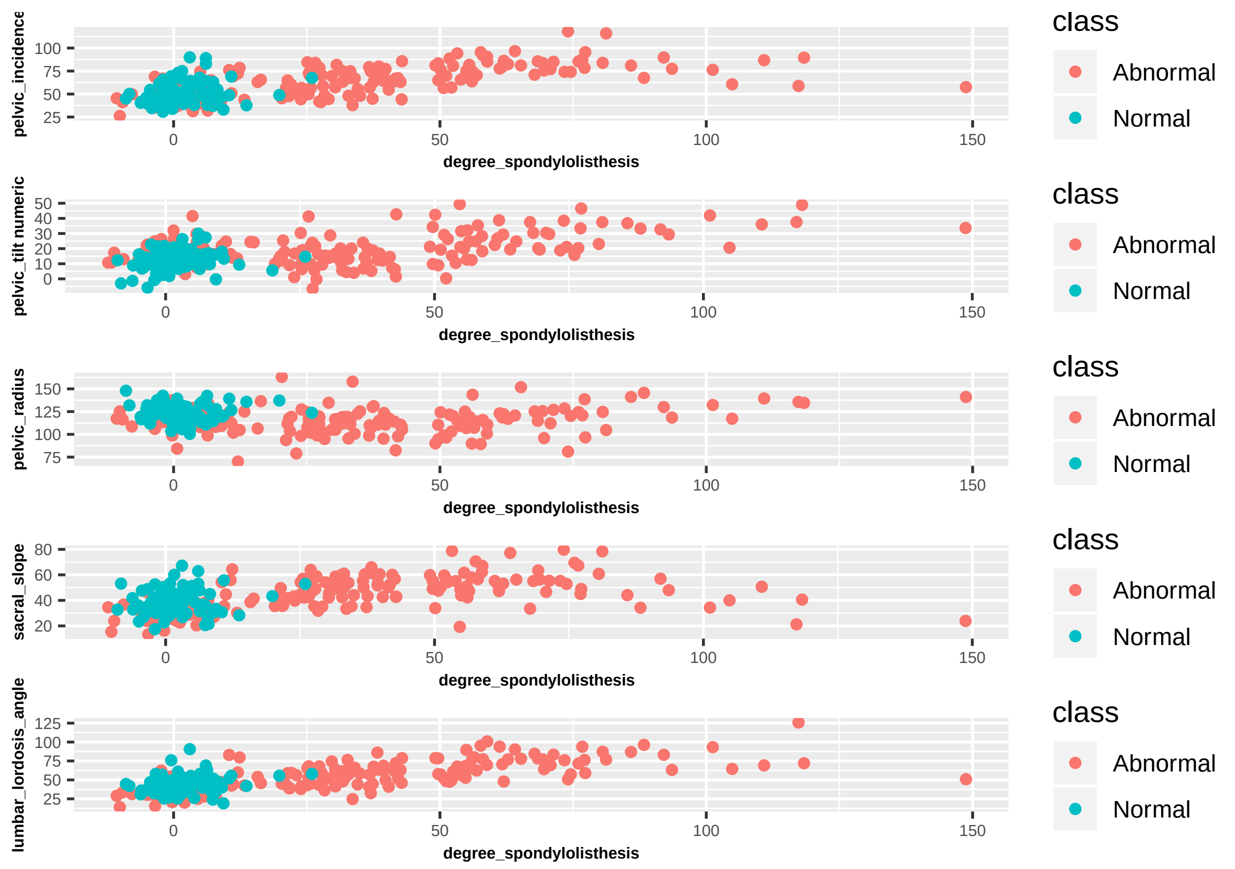
- More importantly, we should look at the normal vs. abnormal split within each predictor to determine whether any one predictor is better than the others in predicting abnormality. Granted that, as mentioned in the Introduction above, abnormality is most likely a correlation result of a number of attributes. Nevertheless, looking deeper into each predictor before the machine takes over will provide some insights, which may be useful.



From the boxplots above, there is a clear demarcation in the degree of spondylolisthesis that separates the abnormal cases from the normal cases – as in if your degree of spondylolisthesis exceeds X degrees, it's very likely that you will be considered abnormal. Such clear demarcation does not seem to exist in other predictors.

This begs the question: How accurate would our prediction be if we only have degree_spondylolisthesis as the predictor? This would be an interesting question to explore, especially in situations where data collection time is limited and only information on one predictor can be gathered. Throughout the analysis in the following section, we'll look at the accuracy of using all of the predictors and the accuracy of using just degree_spondylolisthesis (one-predictor analysis).

But, before we do that, as a part of our exploration, it is worthwhile to see the relationship between degree_spondylolisthesis and other predictors.



From the graphs above, we can again see the demarcation line put forward by degree_spondylolisthesis, which other predictors do not clearly show. For example, we can see how datapoints with degree_spondylolisthesis larger than 25 would very likely be classified as "abnormal". Meanwhile, all of the other predictors do not have the same predictive strength as normal datapoints are clustered within degree_spondylolisthesis and spread out in the other predictors.

Step 5: Train and fit different models into the training dataset

After reviewing the data and understanding that there is no missing information and that one predictor stands out, in this step, we will fit a number of different models into the training data. We will not only look into accuracy, but also sensitivity. The textbook defines sensitivity as "the ability of an algorithm to predict a positive outcome when the actual outcome is positive." This is an important feature in analyzing medical data because we would not want to miss an abnormal case. In the confusion matrix, "Abnormal" is noted as positive. Of course, we do not want to classify everything as abnormal for the sake of making sensitivity 100%, so a balance needs to be struck between ensuring accuracy while still maintaining an acceptable level of sensitivity.

The data will be prepped by searching for any predictors that have near zero variance. This is an excellent exercise for a huge dataset with many predictors, since any zero variance predictors can be removed, simplifying the analysis. However, because our dataset is fairly small already, we'll do this exercise out of curiousity.

```
nzv <- nearZeroVar(train_set, saveMetrics = TRUE)
nzv
##                          freqRatio percentUnique zeroVar   nzv
## pelvic_incidence          1.000000   100.0000000   FALSE FALSE
## pelvic_tilt.numeric       1.000000   100.0000000   FALSE FALSE
## lumbar_lordosis_angle     1.000000    92.4302789   FALSE FALSE
## sacral_slope              1.333333    92.0318725   FALSE FALSE
## pelvic_radius             1.000000   100.0000000   FALSE FALSE
## degree_spondylolisthesis  1.000000   100.0000000   FALSE FALSE
## class                     2.098765     0.7968127   FALSE FALSE
```

In addition, since we'll be using the caret package, the train set will be subjected to trainControl cross validation method. Cross validation is adopted to manage the variance caused by the small dataset.

```
# Model preparation: Cross validation
control <- trainControl(method = "cv", number = 10, p = .9)
```

The section below will outline why each model is chosen for this project and the model training process.

Model 1: Generalized Linear Model

The first model we're going to train is Generalized Linear Model (glm). The objective of a glm model is to model the expected value of a continuous variable, Y, as a linear function of the continuous predictor, X, according to this linear equation:

$$Y = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + ...$$

This model may work well for a dataset with linear correlations between its predictors and the predicted. However, in most cases, it's very unlikely that anything will be straightforward linear. Nevertheless, GLM is a good baseline model to have.

As such, we'll train a GLM model for our dataset and see how it'll perform. (Note: GLM has no tuning parameters within caret.)

First training: With all of the predictors.

```
# Use all of the predictors with the glm model
set.seed(2020)
train_glm1 <- train(class ~ .,
                    method = "glm",
                    family="binomial",
                    trControl = control,
                    data = train_set)

y_hat_glm1 <- predict(train_glm1, test_set, type = "raw")

cm_glm1_acc <- confusionMatrix(y_hat_glm1, y_test)$overall[["Accuracy"]]

cm_glm1_sen <- confusionMatrix(y_hat_glm1, y_test)$byClass[["Sensitivity"]]

cm_glm1_spe <- confusionMatrix(y_hat_glm1, y_test)$byClass[["Specificity"]]

model_glm1 <- tibble(model = "glm - all predictors",
                     accuracy = cm_glm1_acc,
                     sensitivity = cm_glm1_sen,
                     specificity = cm_glm1_spe)

accuracy_result <- model_glm1
print.data.frame(accuracy_result)
##                    model  accuracy sensitivity specificity
## 1 glm - all predictors 0.8928571   0.8947368   0.8888889
```

Second training: With degree_spondylolisthesis as the only predictor.

```
set.seed(2020)
train_glm2 <- train(class ~ degree_spondylolisthesis,
                    method = "glm",
                    family="binomial",
                    trControl = control,
                    data = train_set)
y_hat_glm2 <- predict(train_glm2, test_set, type = "raw")

cm_glm2_acc <- confusionMatrix(y_hat_glm2, y_test)$overall[["Accuracy"]]

cm_glm2_sen <- confusionMatrix(y_hat_glm2, y_test)$byClass[["Sensitivity"]]

cm_glm2_spe <- confusionMatrix(y_hat_glm2, y_test)$byClass[["Specificity"]]

accuracy_result <- bind_rows(accuracy_result,
                          data_frame(model ="glm - one predictor",
                                     accuracy = cm_glm2_acc,
                                     sensitivity = cm_glm2_sen,
                                     specificity = cm_glm2_spe))

print.data.frame(accuracy_result)
##                    model  accuracy sensitivity specificity
## 1 glm - all predictors 0.8928571   0.8947368   0.8888889
## 2  glm - one predictor 0.7857143   0.7894737   0.7777778
```

From the above, we can see that GLM produces an accuracy and sensitivity that are almost at 90%!! Can other models beat it?

We also see that using one predictor only, GLM still produces a fairly accurate result, although 12% less accurate than when all predictors are used.

Model 2: k-Nearest Neighbors

The next algorithm model we're going to train our dataset on is k-Nearest Neighbors (kNN). With kNN, for any point (x1,x2, … x6) for which we want an estimate of p(x1,x2, … x6), we look for the k nearest points to (x1,x2, … x6) and then take an average of the all of the distance from the point to the neighbors. The probability estimate is done according to this formula[16] (noting that we have six predictors):

$$p(x_1, x_2, ...x_6) = Pr(Y = 1|X_1 = x_1, X_2 = x_2, ...X_6 = x_6)$$

The advantage of the kNN model over other models is that it's easier to adapt to multiple predictors, as kernel methods such as kNN do not have model parameters to estimate. However, the kNN model is also challenged by the curse of dimensionality, as distance between points are harder and harder to define with multiple predictors in multiple dimensions[17].

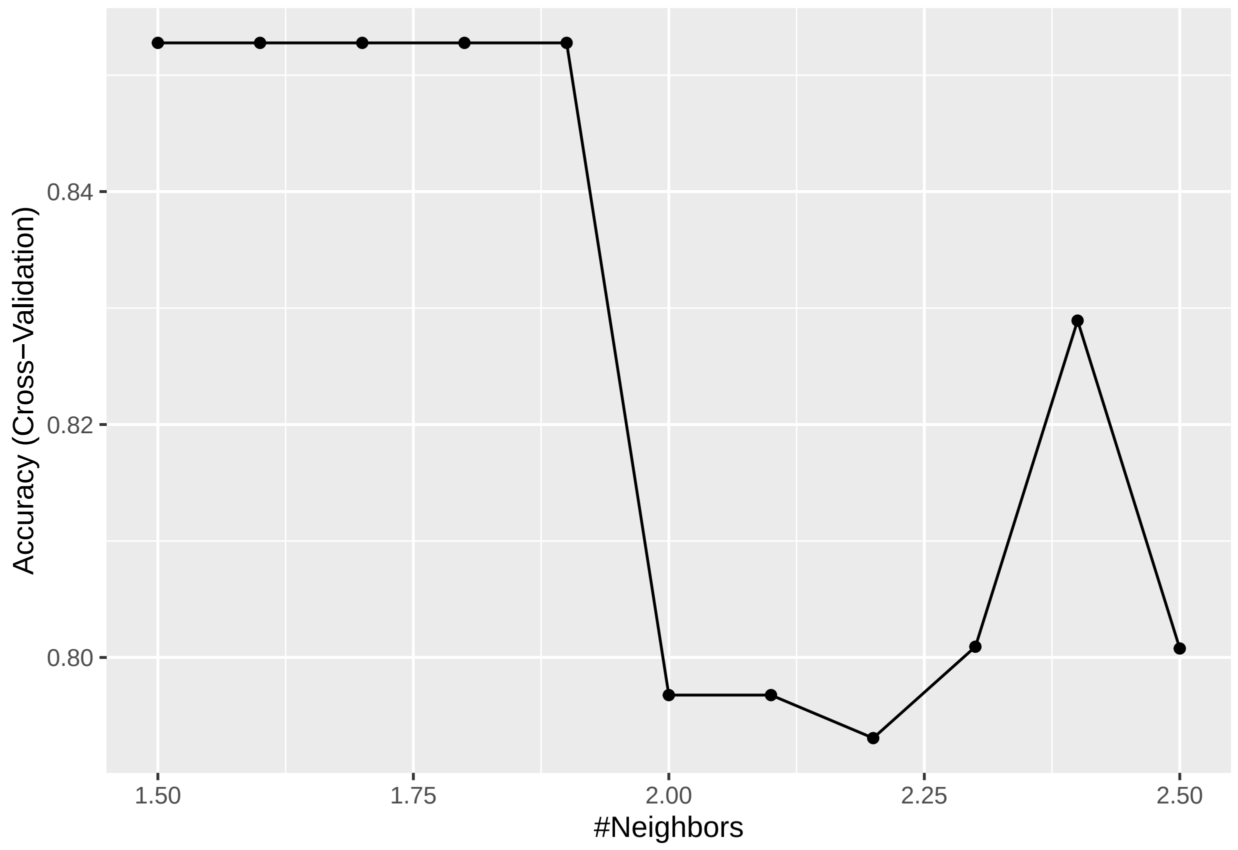
With that in mind, let's see how kNN perform on our dataset.

First training: With all of the predictors.

```
# Use all of the predictors with the knn model
set.seed(2020)

train_knn1 <- train(class ~ .,
                    method = "knn",
```

[16]https://rafalab.github.io/dsbook/cross-validation.html
[17]https://towardsdatascience.com/k-nearest-neighbors-and-the-curse-of-dimensionality-e39d10a6105d

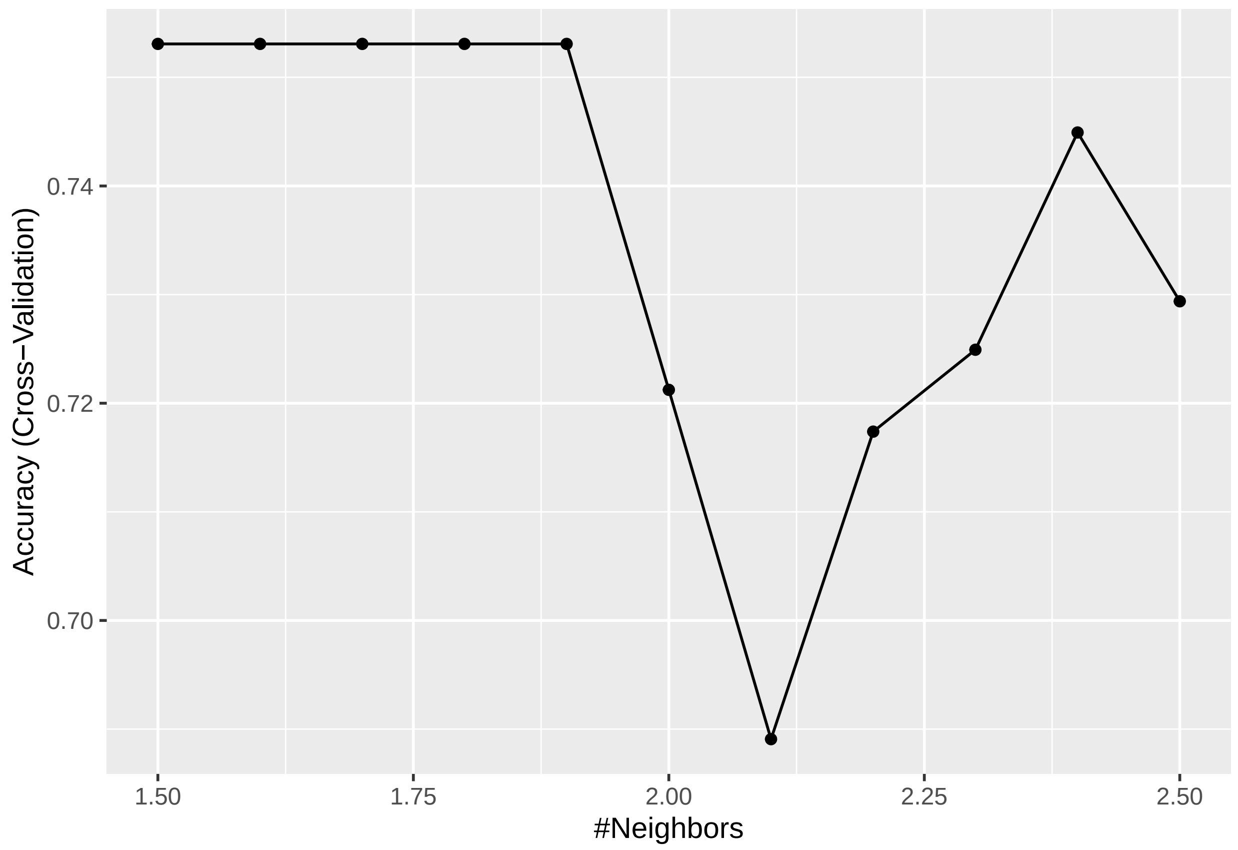
```
                     tuneGrid = data.frame(k = seq(1.5, 2.5, 0.1)),
                     trControl = control,
                     data = train_set)
ggplot(train_knn1)
```



```
train_knn1$bestTune
##     k
## 5 1.9

y_hat_knn1 <- predict(train_knn1, test_set, type = "raw")

cm_knn1_acc <- confusionMatrix(y_hat_knn1, y_test)$overall[["Accuracy"]]

cm_knn1_sen <- confusionMatrix(y_hat_knn1, y_test)$byClass[["Sensitivity"]]

cm_knn1_spe <- confusionMatrix(y_hat_knn1, y_test)$byClass[["Specificity"]]

accuracy_result <- bind_rows(accuracy_result,
                             data_frame(model ="knn - all predictors",
                                        accuracy = cm_knn1_acc,
                                        sensitivity = cm_knn1_sen,
                                        specificity = cm_knn1_spe))

print.data.frame(accuracy_result)
##                  model  accuracy sensitivity specificity
## 1 glm - all predictors 0.8928571   0.8947368   0.8888889
## 2  glm - one predictor 0.7857143   0.7894737   0.7777778
## 3 knn - all predictors 0.8214286   0.8421053   0.7777778
```

Second training: With degree_spondylolisthesis as the only predictor.

```
#  Use one predictor with the knn model
set.seed(2020)

train_knn2 <- train(class ~ degree_spondylolisthesis,
                    method = "knn",
                    tuneGrid = data.frame(k = seq(1.5, 2.5, 0.1)),
                    trControl = control,
                    data = train_set)

ggplot(train_knn2)
```



```
train_knn2$bestTune
##     k
## 5 1.9

y_hat_knn2 <- predict(train_knn2, test_set, type = "raw")

cm_knn2_acc <- confusionMatrix(y_hat_knn2, y_test)$overall[["Accuracy"]]

cm_knn2_sen <- confusionMatrix(y_hat_knn2, y_test)$byClass[["Sensitivity"]]
```

```r
cm_knn2_spe <- confusionMatrix(y_hat_knn2, y_test)$byClass[["Specificity"]]

accuracy_result <- bind_rows(accuracy_result,
                             data_frame(model ="knn - one predictor",
                                        accuracy = cm_knn2_acc,
                                        sensitivity = cm_knn2_sen,
                                        specificity = cm_knn2_spe))

print.data.frame(accuracy_result)
##                  model  accuracy sensitivity specificity
## 1 glm - all predictors 0.8928571   0.8947368   0.8888889
## 2  glm - one predictor 0.7857143   0.7894737   0.7777778
## 3 knn - all predictors 0.8214286   0.8421053   0.7777778
## 4  knn - one predictor 0.7142857   0.8421053   0.4444444
```

From the above, we see that kNN is not more accurate nor it is more sensitive compared to the glm model. However, were we to use only the kNN model, its all-predictors model is 9.5% more accurate than when only one predictor is used. Compare this with the glm model where the all-predictors model is 13.6% more accurate than when only one predictor is use. This verifies the kNN model's challenge with the curse of dimensionality.

Model 3: Naive Bayes

In Naive Bayes algorithm, predictors are treated as if they are independent of each other. As a result, Naive Bayes is not affected by the curse of dimensionality as much as other models.[18] However, in reality, it is almost rare that the predictors are independent of each other.[19] So, how will Naive Bayes perform on our dataset?

First training: With all of the predictors.

```r
# Use all of the predictors with the Naive Bayes model
set.seed(2020)

train_naivebayes1 <- train(class ~ .,
                           method = "naive_bayes",
                           trControl = control,
                           data = train_set)

y_hat_naivebayes1 <- predict(train_naivebayes1, test_set, type = "raw")

cm_naivebayes1_acc <- confusionMatrix(y_hat_naivebayes1, y_test)$overall[["Accuracy"]]

cm_naivebayes1_sen <- confusionMatrix(y_hat_naivebayes1, y_test)$byClass[["Sensitivity"]]

cm_naivebayes1_spe <- confusionMatrix(y_hat_naivebayes1, y_test)$byClass[["Specificity"]]

accuracy_result <- bind_rows(accuracy_result,
                             data_frame(model ="naive_bayes - all predictors",
                                        accuracy = cm_naivebayes1_acc,
                                        sensitivity = cm_naivebayes1_sen,
                                        specificity = cm_naivebayes1_spe))
```

[18]https://shuzhanfan.github.io/2018/06/understanding-mathematics-behind-naive-bayes/
[19]https://www.datacamp.com/community/tutorials/naive-bayes-scikit-learn

```
print.data.frame(accuracy_result)
##                           model  accuracy sensitivity specificity
## 1         glm - all predictors 0.8928571   0.8947368   0.8888889
## 2          glm - one predictor 0.7857143   0.7894737   0.7777778
## 3         knn - all predictors 0.8214286   0.8421053   0.7777778
## 4          knn - one predictor 0.7142857   0.8421053   0.4444444
## 5 naive_bayes - all predictors 0.7142857   0.6315789   0.8888889
```

Second training: With degree_spondylolisthesis as the only predictor.

```
# Use one predictor with the Naive Bayes model
set.seed(2020)

train_naivebayes2 <- train(class ~ degree_spondylolisthesis,
                           method = "naive_bayes",
                           trControl = trainControl(method = "cv", number = 10),
                           data = train_set)

y_hat_naivebayes2 <- predict(train_naivebayes2, test_set, type = "raw")

cm_naivebayes2_acc <- confusionMatrix(y_hat_naivebayes2, y_test)$overall[["Accuracy"]]

cm_naivebayes2_sen <- confusionMatrix(y_hat_naivebayes2, y_test)$byClass[["Sensitivity"]]

cm_naivebayes2_spe <- confusionMatrix(y_hat_naivebayes2, y_test)$byClass[["Specificity"]]

accuracy_result <- bind_rows(accuracy_result,
                             data_frame(model ="naive_bayes - one predictor",
                                        accuracy = cm_naivebayes2_acc,
                                        sensitivity = cm_naivebayes2_sen,
                                        specificity = cm_naivebayes2_spe))

print.data.frame(accuracy_result)
##                           model  accuracy sensitivity specificity
## 1         glm - all predictors 0.8928571   0.8947368   0.8888889
## 2          glm - one predictor 0.7857143   0.7894737   0.7777778
## 3         knn - all predictors 0.8214286   0.8421053   0.7777778
## 4          knn - one predictor 0.7142857   0.8421053   0.4444444
## 5 naive_bayes - all predictors 0.7142857   0.6315789   0.8888889
## 6  naive_bayes - one predictor 0.7857143   0.6842105   1.0000000
```

The Naive Bayes algorithm performed the worst compared to the previous two, which may imply that the algorithm's assumption on the independence of the predictors of each other may not be completely valid. Interestingly, using only one predictor, the degree_spondylolisthesis, yields a slightly more accurate result. This could very well be caused by the small dataset, yielding variance on the analysis.

Model 4: Decision Tree

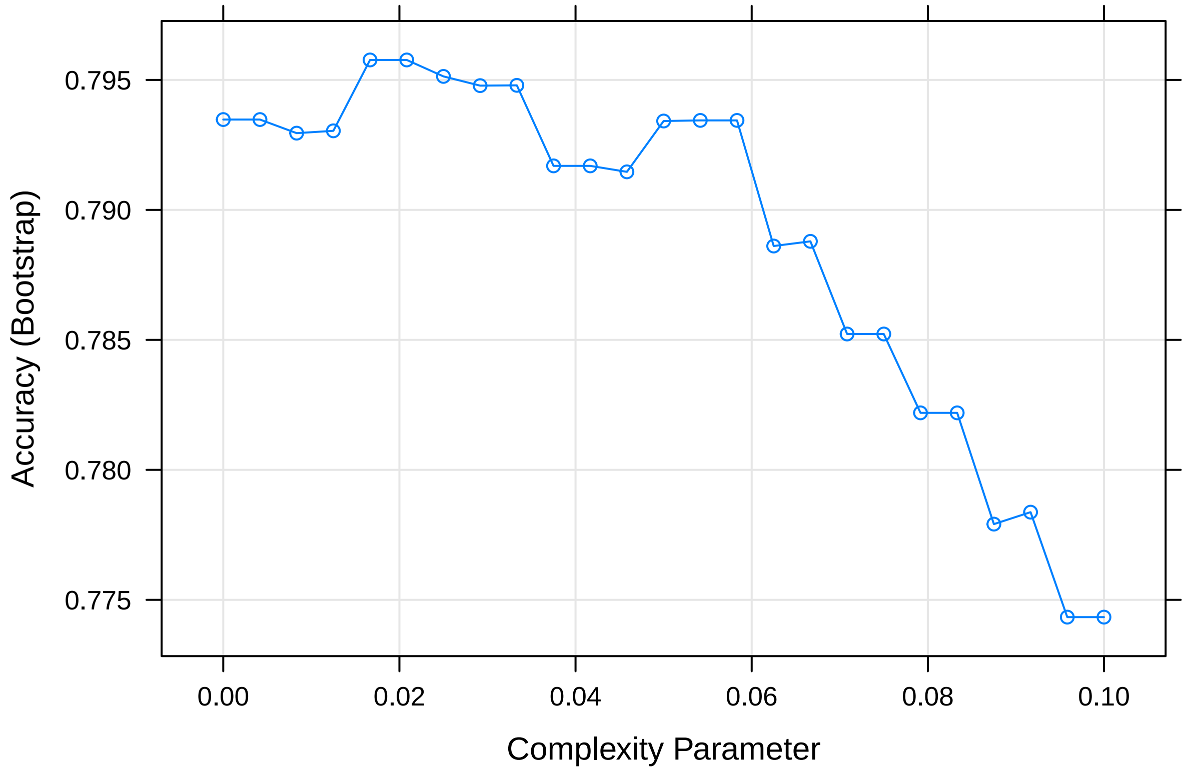
"A tree is basically a flow chart of yes or no questions."[20] The general idea of Decision Tree is that it partitions the predictors and split the data that way. Specifically in our case, knowing that degree_spondylolisthesis is the one predictor that stands out, it would make sense for our tree to have degree_spondylolisthesis as the first node. In other words, the first stop would be the algorithm seeing whether a particular patient's spondylolisthesis exceeds a certain degree, and classify the patient as normal or abnormal. If the patient is

[20]https://rafalab.github.io/dsbook/examples-of-algorithms.html

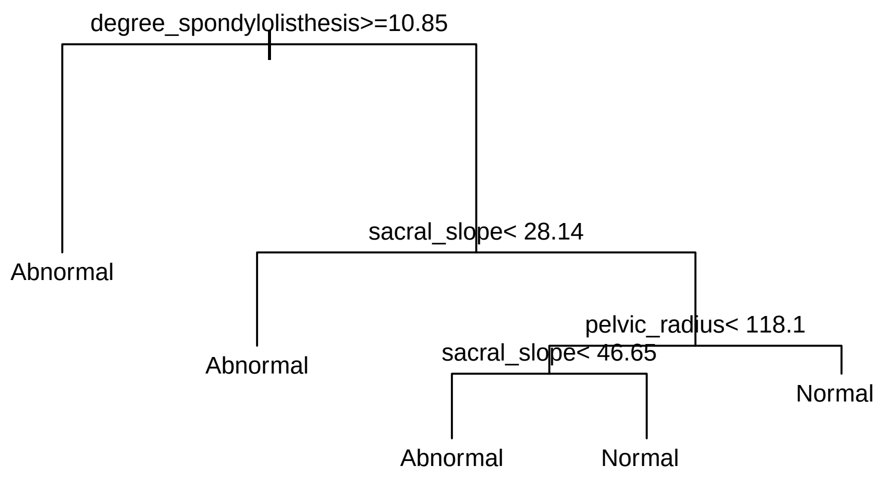
classified as normal, he / she will be subjected to other predictors' cutoff. The plot of the tree below outlines the process.

First training: With all of the predictors.

```r
# Use all of the predictors with the Decision Tree model
set.seed(2020)

train_rpart1 <- train(class ~ .,
                      method = "rpart",
                      tuneGrid = data.frame(cp = seq(0.0, 0.1, len = 25)),
                      data = train_set)

plot(train_rpart1)
```



```r
y_hat_rpart1 <- predict(train_rpart1, test_set, type = "raw")

plot(train_rpart1$finalModel, margin = 0.1)
text(train_rpart1$finalModel,  cex = 0.75)
```



```
cm_rpart1_acc <- confusionMatrix(y_hat_rpart1, y_test)$overall[["Accuracy"]]

cm_rpart1_sen <- confusionMatrix(y_hat_rpart1, y_test)$byClass[["Sensitivity"]]

cm_rpart1_spe <- confusionMatrix(y_hat_rpart1, y_test)$byClass[["Specificity"]]

accuracy_result <- bind_rows(accuracy_result,
                             data_frame(model ="decision_tree - all predictors",
                                        accuracy = cm_rpart1_acc,
                                        sensitivity = cm_rpart1_sen,
                                        specificity = cm_rpart1_spe))

print.data.frame(accuracy_result)
##                            model  accuracy sensitivity specificity
## 1          glm - all predictors 0.8928571   0.8947368   0.8888889
## 2           glm - one predictor 0.7857143   0.7894737   0.7777778
## 3          knn - all predictors 0.8214286   0.8421053   0.7777778
## 4           knn - one predictor 0.7142857   0.8421053   0.4444444
## 5  naive_bayes - all predictors 0.7142857   0.6315789   0.8888889
## 6   naive_bayes - one predictor 0.7857143   0.6842105   1.0000000
## 7 decision_tree - all predictors 0.9642857   0.9473684   1.0000000
```

Second training: With degree_spondylolisthesis as the only predictor.

```
# Use one predictor with the Decision Tree model
set.seed(2020)
```

```
train_rpart2 <- train(class ~ degree_spondylolisthesis,
                      method = "rpart",
                      tuneGrid = data.frame(cp = seq(0.0, 0.1, len = 25)),
                      data = train_set)

y_hat_rpart2 <- predict(train_rpart2, test_set, type = "raw")

cm_rpart2_acc <- confusionMatrix(y_hat_rpart2, y_test)$overall[["Accuracy"]]

cm_rpart2_sen <- confusionMatrix(y_hat_rpart2, y_test)$byClass[["Sensitivity"]]

cm_rpart2_spe <- confusionMatrix(y_hat_rpart2, y_test)$byClass[["Specificity"]]

accuracy_result <- bind_rows(accuracy_result,
                             data_frame(model ="decision_tree - one predictor",
                                        accuracy = cm_rpart2_acc,
                                        sensitivity = cm_rpart2_sen,
                                        specificity = cm_rpart2_spe))

print.data.frame(accuracy_result)
##                            model  accuracy sensitivity specificity
## 1           glm - all predictors 0.8928571   0.8947368   0.8888889
## 2            glm - one predictor 0.7857143   0.7894737   0.7777778
## 3           knn - all predictors 0.8214286   0.8421053   0.7777778
## 4            knn - one predictor 0.7142857   0.8421053   0.4444444
## 5   naive_bayes - all predictors 0.7142857   0.6315789   0.8888889
## 6    naive_bayes - one predictor 0.7857143   0.6842105   1.0000000
## 7 decision_tree - all predictors 0.9642857   0.9473684   1.0000000
## 8  decision_tree - one predictor 0.8214286   0.7368421   1.0000000
```

Compared to other algorithms, the Decision Tree exhibits highest level of accuracy when both all predictors and one predictor used. This indicates that splitting data at predictors' cutoff points seems to work best so far.

Although, it is also worth noting that using only one predictor here results in 15% less accuracy than when all predictors used. In other words, the best possible outcome so far is derived by using all predictors on Decision Tree. Additionally, if only one predictor can be used, Decision Tree is still the best model to achieve that.

As a quick intermezzo, one may have noticed that the accuracy of the decision tree with one predictor is identical as the kNN accuracy with all predictors. To see what's going on, we are going to look at the predictions produced by each method.

```
# Intermezzo: addressing identical accuracy between knn - all predictors and decision tree - one predic

cbind.data.frame(y_hat_knn1, y_hat_rpart2)
##    y_hat_knn1 y_hat_rpart2
## 1    Abnormal       Normal
## 2      Normal       Normal
## 3    Abnormal       Normal
## 4    Abnormal       Normal
## 5    Abnormal     Abnormal
## 6      Normal       Normal
## 7    Abnormal     Abnormal
## 8    Abnormal     Abnormal
## 9    Abnormal     Abnormal
## 10   Abnormal      Abnormal
## 11   Abnormal      Abnormal
## 12   Abnormal      Abnormal
## 13     Normal      Abnormal
## 14   Abnormal      Abnormal
## 15   Abnormal      Abnormal
## 16   Abnormal      Abnormal
## 17   Abnormal      Abnormal
## 18   Abnormal      Abnormal
## 19   Abnormal      Abnormal
## 20     Normal        Normal
## 21     Normal        Normal
## 22   Abnormal        Normal
## 23     Normal        Normal
## 24     Normal        Normal
## 25   Abnormal        Normal
## 26     Normal        Normal
## 27     Normal        Normal
## 28     Normal        Normal
```

From the above, one can see that both algorithms yielded different predictions. However, due to the small dataset size, both algorithms yielded the same number of Abnormal predictions, resulting in identical accuracy level. This is confirming that there is noting out of the ordinary.

Model 5: Random Forest

While Decision Tree, with an accuracy rate of 96%, already works very well for this data set, Random forests tries to improve it even more by creating random multiple trees and averaging them out.[21]

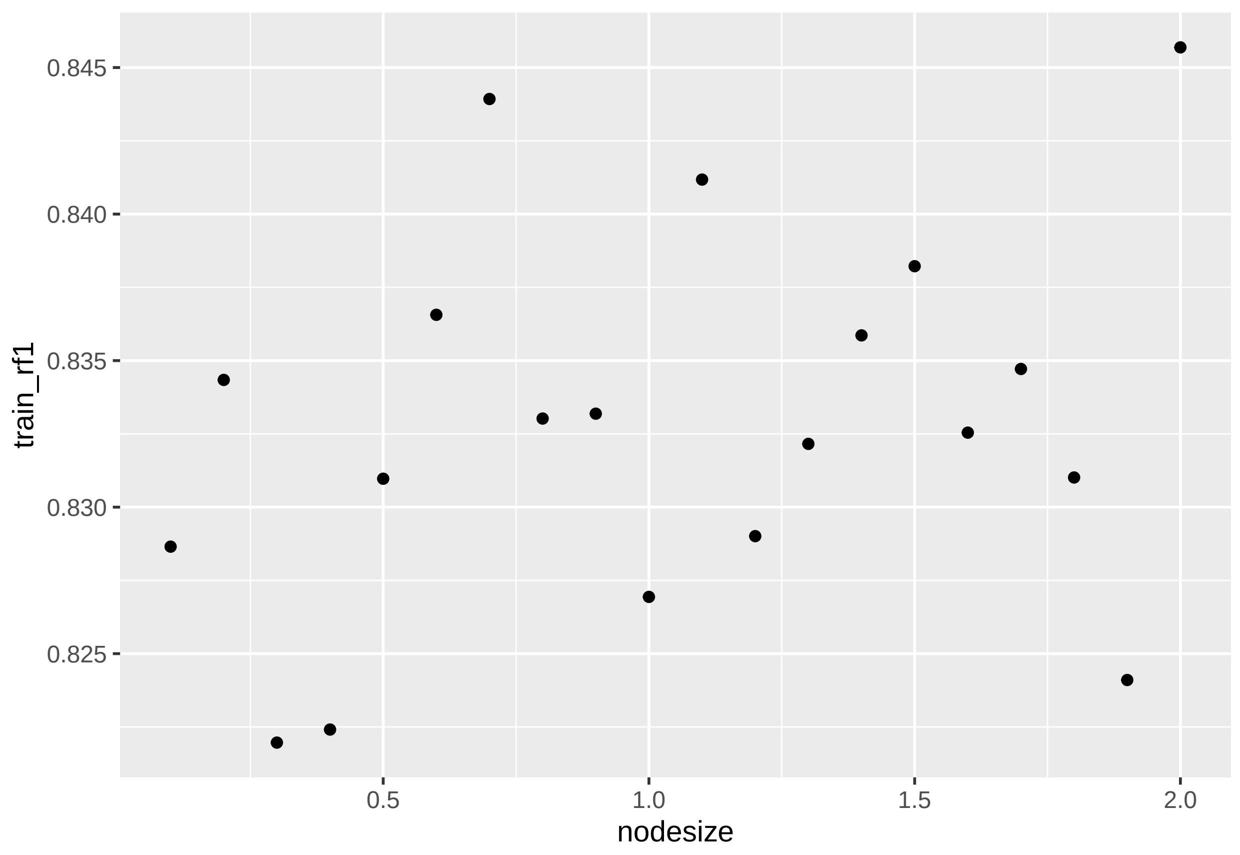
Will this work for this particular dataset, or will introducing randomness in creating the trees actually result in lower performance?

Note: per textbook, this is not one of the parameters that the caret package optimizes by default, thus we'll optimize separately in the code below.

First training: With all of the predictors.

```r
# Use all of the predictors with the Random Forest model
set.seed(2020)
nodesize <- seq(0.1, 2, 0.1)
train_rf1 <- sapply(nodesize, function(ns){
  train(class ~ ., method = "rf", data = train_set,
        tuneGrid = data.frame(mtry = 2),
        nodesize = ns)$results$Accuracy
})
qplot(nodesize, train_rf1)
```

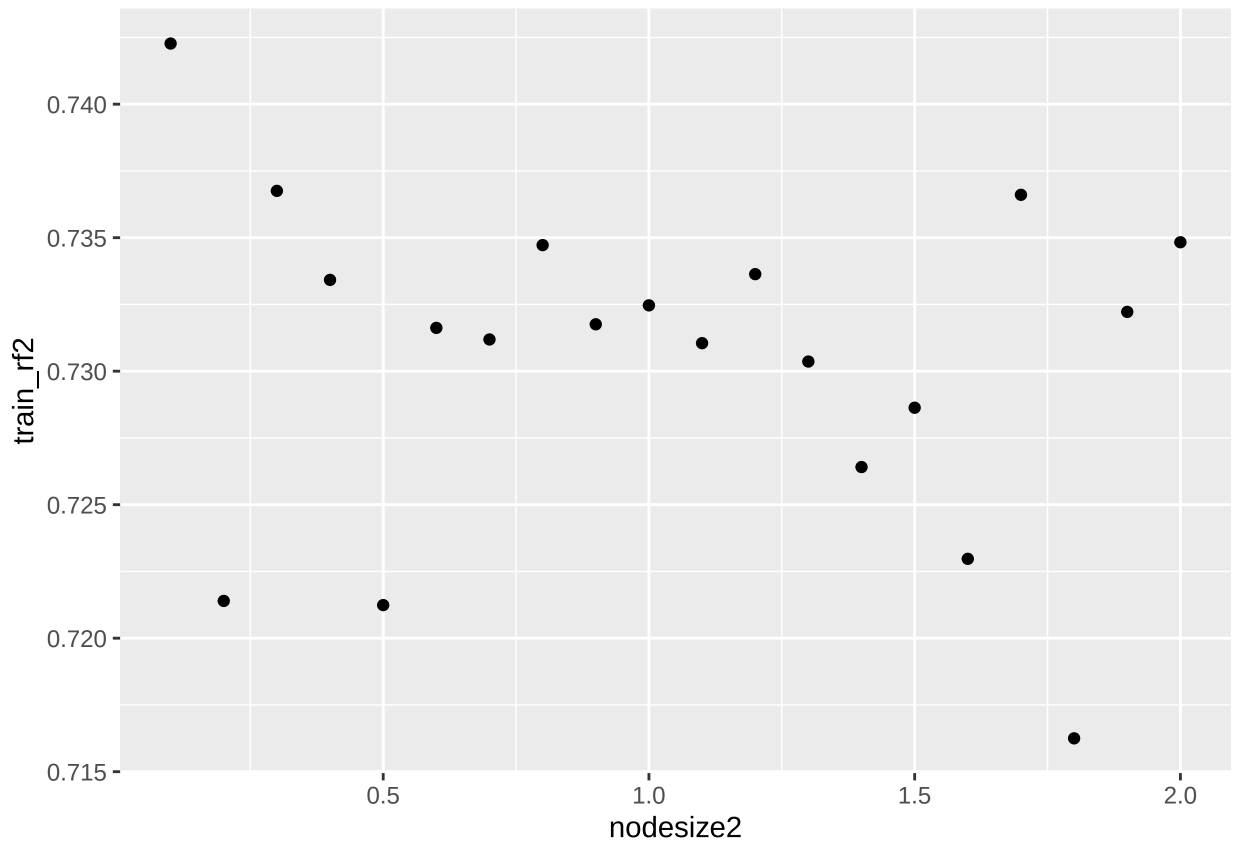
[21] https://rafalab.github.io/dsbook/examples-of-algorithms.html



```r
n <- nodesize[which.max(train_rf1)]

train_rf1 <- train(class ~ ., method = "rf", nodesize = n, data = train_set)

y_hat_rf1 <- predict(train_rf1, test_set, type = "raw")

cm_rf1_acc <- confusionMatrix(y_hat_rf1, y_test)$overall[["Accuracy"]]

cm_rf1_sen <- confusionMatrix(y_hat_rf1, y_test)$byClass[["Sensitivity"]]

cm_rf1_spe <- confusionMatrix(y_hat_rf1, y_test)$byClass[["Specificity"]]

accuracy_result <- bind_rows(accuracy_result,
                             data_frame(model ="random_forest - all predictors",
                                        accuracy = cm_rf1_acc,
                                        sensitivity = cm_rf1_sen,
                                        specificity = cm_rf1_spe))

print.data.frame(accuracy_result)
##                            model  accuracy sensitivity specificity
## 1           glm - all predictors 0.8928571   0.8947368   0.8888889
## 2            glm - one predictor 0.7857143   0.7894737   0.7777778
## 3           knn - all predictors 0.8214286   0.8421053   0.7777778
## 4            knn - one predictor 0.7142857   0.8421053   0.4444444
## 5   naive_bayes - all predictors 0.7142857   0.6315789   0.8888889
## 6    naive_bayes - one predictor 0.7857143   0.6842105   1.0000000
## 7 decision_tree - all predictors 0.9642857   0.9473684   1.0000000
## 8  decision_tree - one predictor 0.8214286   0.7368421   1.0000000
## 9 random_forest - all predictors 0.8928571   0.9473684   0.7777778
```

Second training: With degree_spondylolisthesis as the only predictor.

```r
set.seed(2020)
nodesize2 <- seq(0.1, 2, 0.1)

train_rf2 <- sapply(nodesize2, function(ns){
  train(class ~ degree_spondylolisthesis, method = "rf", data = train_set,
        tuneGrid = data.frame(mtry = 2),
        nodesize = ns)$results$Accuracy
})
qplot(nodesize2, train_rf2)
```



```r
n2 <- nodesize[which.max(train_rf2)]
n2
## [1] 0.1
train_rf2 <- train(class ~ degree_spondylolisthesis, method = "rf", nodesize = n2, data = train_set)

y_hat_rf2 <- predict(train_rf2, test_set, type = "raw")

cm_rf2_acc <- confusionMatrix(y_hat_rf2, y_test)$overall[["Accuracy"]]

cm_rf2_sen <- confusionMatrix(y_hat_rf2, y_test)$byClass[["Sensitivity"]]
```

```
cm_rf2_spe <- confusionMatrix(y_hat_rf2, y_test)$byClass[["Specificity"]]

accuracy_result <- bind_rows(accuracy_result,
                             data_frame(model ="random_forest - one predictor",
                                        accuracy = cm_rf2_acc,
                                        sensitivity = cm_rf2_sen,
                                        specificity = cm_rf2_spe))

print.data.frame(accuracy_result)
##                            model  accuracy sensitivity specificity
## 1           glm - all predictors 0.8928571   0.8947368   0.8888889
## 2            glm - one predictor 0.7857143   0.7894737   0.7777778
## 3           knn - all predictors 0.8214286   0.8421053   0.7777778
## 4            knn - one predictor 0.7142857   0.8421053   0.4444444
## 5   naive_bayes - all predictors 0.7142857   0.6315789   0.8888889
## 6    naive_bayes - one predictor 0.7857143   0.6842105   1.0000000
## 7 decision_tree - all predictors 0.9642857   0.9473684   1.0000000
## 8  decision_tree - one predictor 0.8214286   0.7368421   1.0000000
## 9 random_forest - all predictors 0.8928571   0.9473684   0.7777778
## 10 random_forest - one predictor 0.7142857   0.8421053   0.4444444
```

Surprisingly, Random Forests does not perform as well as Decision Tree for this dataset. It could be that the randomness introduced in making the trees reduces its accuracy.

Model 6: Ensemble

After reviewing a number of different models, we should look into the possibility of having an Ensemble model. In the Ensemble model, all of the models' predictions are laid out next to each other, and a majority vote is taken. For example, patient 1 may receive Abnormal Abnormal Normal Abnormal Normal from each of the five models we've looked at. In this case, since the majority of the models voted Abnormal, the Ensemble model's classification for patient 1 will be "Abnormal." The 50% cutoff for vote counting should be easy since we have an odd number of models. In the following, we'll see whether an Ensemble model would be more accurate than the rest.

First ensemble: With all of the predictors.

```
# Build an ensemble model based on all predictors predictions

a <- as.character(y_hat_glm1)
b <- as.character(y_hat_knn1)
c <- as.character(y_hat_naivebayes1)
d <- as.character(y_hat_rpart1)
e <- as.character(y_hat_rf1)

ensemble_model <- cbind.data.frame(a, b, c, d, e)

ensemble_accuracy <- colMeans(ensemble_model == test_set$class)
ensemble_accuracy
##         a         b         c         d         e
## 0.8928571 0.8214286 0.7142857 0.9642857 0.8928571
mean(ensemble_accuracy)
## [1] 0.8571429

votes <- rowMeans(ensemble_model == "Normal")
```

```
# if more than half of the models predicted "Normal," then the ensemble's prediction
# is also "Normal." This should be easy to determine since we have an odd number of
# models.

y_hat_ensemble1 <- ifelse(votes > 0.5, "Normal", "Abnormal")

y_hat_ensemble1 <- as.factor(y_hat_ensemble1)
cm_ensemble1_acc <- confusionMatrix(y_hat_ensemble1, y_test)$overall[["Accuracy"]]

cm_ensemble1_sen <- confusionMatrix(y_hat_ensemble1, y_test)$byClass[["Sensitivity"]]

cm_ensemble1_spe <- confusionMatrix(y_hat_ensemble1, y_test)$byClass[["Specificity"]]

accuracy_result <- bind_rows(accuracy_result,
                             data_frame(model ="ensemble - all predictors",
                                        accuracy = cm_ensemble1_acc,
                                        sensitivity = cm_ensemble1_sen,
                                        specificity = cm_ensemble1_spe))

print.data.frame(accuracy_result)
##                             model  accuracy sensitivity specificity
## 1            glm - all predictors 0.8928571   0.8947368   0.8888889
## 2             glm - one predictor 0.7857143   0.7894737   0.7777778
## 3            knn - all predictors 0.8214286   0.8421053   0.7777778
## 4             knn - one predictor 0.7142857   0.8421053   0.4444444
## 5    naive_bayes - all predictors 0.7142857   0.6315789   0.8888889
## 6     naive_bayes - one predictor 0.7857143   0.6842105   1.0000000
## 7  decision_tree - all predictors 0.9642857   0.9473684   1.0000000
## 8   decision_tree - one predictor 0.8214286   0.7368421   1.0000000
## 9  random_forest - all predictors 0.8928571   0.9473684   0.7777778
## 10  random_forest - one predictor 0.7142857   0.8421053   0.4444444
## 11      ensemble - all predictors 0.9285714   0.9473684   0.8888889
```

Second training: With degree_spondylolisthesis as the only predictor.

```
# Build an ensemble model based on one-predictor predictions

f <- as.character(y_hat_glm2)
g <- as.character(y_hat_knn2)
h <- as.character(y_hat_naivebayes2)
i <- as.character(y_hat_rpart2)
j <- as.character(y_hat_rf2)

ensemble_model2 <- cbind.data.frame(f, g, h, i, j)

ensemble_accuracy2 <- colMeans(ensemble_model2 == test_set$class)
ensemble_accuracy2
##         f         g         h         i         j
## 0.7857143 0.7142857 0.7857143 0.8214286 0.7142857
mean(ensemble_accuracy2)
## [1] 0.7642857

votes2 <- rowMeans(ensemble_model2 == "Normal")
```

```
# if more than half of the models predicted "Normal," then the ensemble's prediction
# is also "Normal." This should be easy to determine since we have an odd number of
# models.

y_hat_ensemble2 <- ifelse(votes2 > 0.5, "Normal", "Abnormal")

y_hat_ensemble2 <- as.factor(y_hat_ensemble2)
cm_ensemble2_acc <- confusionMatrix(y_hat_ensemble2, y_test)$overall[["Accuracy"]]

cm_ensemble2_sen <- confusionMatrix(y_hat_ensemble2, y_test)$byClass[["Sensitivity"]]

cm_ensemble2_spe <- confusionMatrix(y_hat_ensemble2, y_test)$byClass[["Specificity"]]

accuracy_result <- bind_rows(accuracy_result,
                             data_frame(model ="ensemble - one predictor",
                                        accuracy = cm_ensemble2_acc,
                                        sensitivity = cm_ensemble2_sen,
                                        specificity = cm_ensemble2_spe))

print.data.frame(accuracy_result)
##                             model  accuracy sensitivity specificity
## 1            glm - all predictors 0.8928571   0.8947368   0.8888889
## 2             glm - one predictor 0.7857143   0.7894737   0.7777778
## 3            knn - all predictors 0.8214286   0.8421053   0.7777778
## 4             knn - one predictor 0.7142857   0.8421053   0.4444444
## 5    naive_bayes - all predictors 0.7142857   0.6315789   0.8888889
## 6     naive_bayes - one predictor 0.7857143   0.6842105   1.0000000
## 7  decision_tree - all predictors 0.9642857   0.9473684   1.0000000
## 8   decision_tree - one predictor 0.8214286   0.7368421   1.0000000
## 9  random_forest - all predictors 0.8928571   0.9473684   0.7777778
## 10  random_forest - one predictor 0.7142857   0.8421053   0.4444444
## 11      ensemble - all predictors 0.9285714   0.9473684   0.8888889
## 12       ensemble - one predictor 0.8214286   0.7894737   0.8888889
```

From the above, we can see that when all predictors are used, the Ensemble model is impressively quite accurate, but not as accurate as the decision tree. However, when only one predictor is used, the Ensemble model presents the most accurate model.

As such, for this dataset, we will choose the Decision Tree to test against the validation set.

However, before we move on, we should revisit our earlier question about Data Imbalance. We'll look at whether the Abnormal Normal ratio in the original data set somehow affects each class's accuracy.

```
# Check decision tree
test_set %>%
  mutate(y_hatA = y_hat_rpart1) %>%
  group_by(class) %>%
  summarize(accuracy = mean(y_hatA == class))
## # A tibble: 2 x 2
##   class    accuracy
##   <chr>       <dbl>
## 1 Abnormal    0.947
## 2 Normal      1
```

Looking at the above result, the accuracy of each class does not seem to be majorly affected by the Normal - Abnormal split. In fact, the accuracy of both classes using Decision Tree is close to 100%. In other words, our dataset is not affected by data imbalance, and we do not need to alter the dataset.[22]

Lastly, now that we have run models through our data, we should confirm our assumption that data_spondylolisthesis is the most important predictor.

```
# Var imp
imp <-varImp(train_rpart1)
imp
## rpart variable importance
##
##                           Overall
## degree_spondylolisthesis 100.000
## lumbar_lordosis_angle     38.017
## pelvic_radius             25.974
## pelvic_tilt.numeric       16.826
## sacral_slope               1.689
## pelvic_incidence           0.000
```

3. RESULTS

As shown in the previous section, Decision Tree is the most accurate and sensitive model for this data set. We'll test it against the validation set and see the level of accuracy and sensitivity we'll get.

```
# Prep
colnames(validation_set) <- make.names(colnames(validation_set))
y_validation <- factor(validation_set$class)

# use train_rpart1, the model that has been trained, against the validation set

y_hat_rpart3 <- predict(train_rpart1, validation_set, type = "raw")

cm_rpart3_acc <- confusionMatrix(y_hat_rpart3, y_validation)$overall[["Accuracy"]]

cm_rpart3_sen <- confusionMatrix(y_hat_rpart3, y_validation)$byClass[["Sensitivity"]]

cm_rpart3_spe <- confusionMatrix(y_hat_rpart3, y_validation)$byClass[["Specificity"]]

validation_result <- tibble(model = "decision_tree_validation - all predictors",
                     accuracy = cm_rpart3_acc,
                     sensitivity = cm_rpart3_sen,
                     specificity = cm_rpart3_spe)

print.data.frame(validation_result)
##                                       model  accuracy sensitivity specificity
## 1 decision_tree_validation - all predictors 0.8064516    0.952381         0.5

# use train_rpart2, the model that has been trained, against the validation set

y_hat_rpart4 <- predict(train_rpart2, validation_set, type = "raw")

cm_rpart4_acc <- confusionMatrix(y_hat_rpart4, y_validation)$overall[["Accuracy"]]

cm_rpart4_sen <- confusionMatrix(y_hat_rpart4, y_validation)$byClass[["Sensitivity"]]
```

[22] https://machinelearningmastery.com/what-is-imbalanced-classification/

```
cm_rpart4_spe <- confusionMatrix(y_hat_rpart4, y_validation)$byClass[["Specificity"]]

validation_result <- bind_rows(validation_result,
                               data_frame(model = "decision_tree_validation - one predictor",
                               accuracy = cm_rpart4_acc,
                               sensitivity = cm_rpart4_sen,
                               specificity = cm_rpart4_spe))

print.data.frame(validation_result)
##                                     model  accuracy sensitivity specificity
## 1 decision_tree_validation - all predictors 0.8064516   0.9523810         0.5
## 2  decision_tree_validation - one predictor 0.7741935   0.8095238         0.7
```

From the above, we see that Decision Tree still yields a fairly accurate result at close to 81% when all predictors are used and 77% when only one predictor is used, albeit lower than during training.

4. CONCLUSION

There are a number of things that can be summarized from the analysis of this dataset.

First, the data is not affected by Data Imbalance. As such, prediction accuracy within each class is not affected and there is no further work needed to alter the dataset and address any imbalance.

Second, one of the predictors, degree of spondylolisthesis is proven to be a very influential predictor. For future studies, if resources are limited, researches may be able to focus solely on this predictor, bearing in mind any accuracy compromise.

Third, Decision Tree as a model fits very well in this dataset in terms of accuracy and sensitivity. However, there is always that danger of overfitting. As such, were the dataset expanded exponentially, more robust model like Random Forests may yield more accurate result.

For future studies, I would recommend an expansion of dataset and refitting of different models in the new dataset in order to achieve higher confidence result.

REFERENCES

1. Irizarry, Rafael A., "Introduction to Data Science," 2020, https://rafalab.github.io/dsbook/
2. Dalpiaz, David, "R for Statistical Learning," 2020, https://daviddalpiaz.github.io/r4sl/
3. Kuhn, Max, "The Caret Package," 2019, https://topepo.github.io/caret/index.html
4. Brownlee, Jason, "A Gentle Introduction to Imbalanced Classification," 2020, https://machinelearningmastery.com/what-is-imbalanced-classification/