Biomechanical features of orthopedic patients

Data Science: Capstone - HardvardX

Oliver, Joan Jaume

May 2019

Contents

1 Introduction

This project is the last part of the `HardvardX Data Science Capstone` and it's a your own choice type project. Here we will have to face a real Machine Learning problem of our own choice without any type of help but ourselves. With the data selection, we are provided two different interesting web pages where we can find data to test our knowledge, in this case the one from the University of California Irvine was the chosen one.

The first main choice was related to which type of outputs we want to work with. In Machine Learning there's two main types, Continuous or Regression and Categorical or Classification. Since the previous project was a Recommendation System and the outputs were continuous this project has been developed based on classification systems.

1.1 About me

Just to introduce a bit of myself, my name is Joan Jaume Oliver and I'm a last year engineering student at the Universitat Politècnica de Catalunya (Spain). I'm passionate about anything related to Machine Learning and Artificial Intelligence, main reason I have been doing this course on my free time for the last three months.

I would like to mention that English is my third language, so you would have to apologize me if something written down doesn't feel right. Sorry in advance!

Do not hesitate to contact me if you have any questions.

You can contact me at: joanjaumeoliver@gmail.com

1.2 About the data set

About all the possible data sets to choose from, i just don't know why, but this one took my attention. I don't know if it was because it's based on a field so different from what I currently work with or just because of the scope. Creating a model that could help doctors with their decisions would be fantastic.

This data set it's based on the vertebral column and each patient is represented in the data set by six biomechanical attributes derived from the shape and orientation of the pelvis and lumbar spine. In total, `310` patients were tested. (The data set can be found here.)

```
#(Downloading data set from my Github project repository).
dl <- tempfile()
download.file(paste("https://raw.githubusercontent.com/joanjaumeoliver/",
                "Vertebral-Column/master/Vertebral-Column%20Data.csv",sep=""), dl)
data <- read_csv(dl)
rm(dl)

#(Renaming the data set columns just to fit the A4 pdf file).
names(data) <- c("var1","var2","var3","var4",
                 "var5","var6","Y")
head(data) %>% knitr::kable()
```

var1	var2	var3	var4	var5	var6	Y
63.02782	22.552586	39.60912	40.47523	98.67292	-0.254400	Hernia
39.05695	10.060992	25.01538	28.99596	114.40543	4.564259	Hernia
68.83202	22.218482	50.09219	46.61354	105.98514	-3.530317	Hernia
69.29701	24.652878	44.31124	44.64413	101.86850	11.211523	Hernia
49.71286	9.652075	28.31741	40.06078	108.16872	7.918501	Hernia
40.25020	13.921907	25.12495	26.32829	130.32787	2.230652	Hernia

2 Analysis

Before start trying different algorithms, we have to check out the data set and see if there's anything that needs to be done.

2.1 Modifying the data

Let's start by looking through our data.

```
#(Check for NA values).
if(any(is.na(data))){print("There's NA values")}

#(Let's see the summary of our data)
#(Divide the data just to fit the A4 .pdf)
summary(data)[,1:3] %>% knitr::kable()
```

var1	var2	var3
Min. : 26.15	Min. :-6.555	Min. : 14.00
1st Qu.: 46.43	1st Qu.:10.667	1st Qu.: 37.00
Median : 58.69	Median :16.358	Median : 49.56
Mean : 60.50	Mean :17.543	Mean : 51.93
3rd Qu.: 72.88	3rd Qu.:22.120	3rd Qu.: 63.00
Max. :129.83	Max. :49.432	Max. :125.74

```
summary(data)[,4:7] %>% knitr::kable()
```

var4	var5	var6	Y
Min. : 13.37	Min. : 70.08	Min. :-11.058	Length:310
1st Qu.: 33.35	1st Qu.:110.71	1st Qu.: 1.604	Class :character
Median : 42.40	Median :118.27	Median : 11.768	Mode :character
Mean : 42.95	Mean :117.92	Mean : 26.297	NA
3rd Qu.: 52.70	3rd Qu.:125.47	3rd Qu.: 41.287	NA
Max. :121.43	Max. :163.07	Max. :418.543	NA

As seen in the data summary, there seems to be an outlier in the `var6` since the difference between the median and the mean is not small. Although it might be correct that the median and mean has such a deviation because of an uncommon data distribution, seems more that there's a register that contains a typo error.

```
data %>% arrange(var6) %>% select(var6) %>% tail() %>% knitr::kable()
```

var6
117.3147
118.3534
124.9844
145.3781
148.7537
418.5431

```
#(Which register is the outlier?)
tibble(Row=which.max(data$var6),Value=max(data$var6)) %>% knitr::kable()
```

Row	Value
116	418.5431

As we see there's a `418` value that is following a `148`, this seems, at least to me, that the user that introduced all these values swapped the position of the first two characters. This is what we can call as an outlier and in this situation an action is required, this value could affect our Machine Learning models.

This is a data set that's related to medicine, so if this were another situation as, for example, when a hospital has asked you help to see if there's a possibility to try applying machine learning techniques here, I would stop the project and ask if this is a typo or not.

If this was a typo, the best solution would be to update the data set and continue with our modeling, but if not, this would be that the data set we are given isn't well-done and is inconsistent. The existence of an outlier would mean that we have not taken into account all `abnormal` conditions and possibilities.

But, as this is a capstone project this register could be easily ignored or modified since both of these solutions wouldn't affect our results too much.

In this exact case, the register will be removed.

```
#(Let's remove this register from our data)
data <- data[-which.max(data$var6),]
#(As we see, the outlier has been correctly removed)
summary(data)[,4:7] %>% knitr::kable()
```

var4	var5	var6	Y
Min. :13.37	Min. : 70.08	Min. :-11.058	Length:309
1st Qu.:33.34	1st Qu.:110.71	1st Qu.: 1.595	Class :character
Median :42.37	Median :118.34	Median : 11.463	Mode :character
Mean :42.70	Mean :117.95	Mean : 25.027	NA
3rd Qu.:52.55	3rd Qu.:125.48	3rd Qu.: 40.881	NA
Max. :79.70	Max. :163.07	Max. :148.754	NA

This data set output has been renamed with `Y`, let's see how it's distributed.

```
#(How many different classes are we working with?)
tibble(Y=data$Y %>% unique()) %>% knitr::kable()
```

Y
Hernia
Spondylolisthesis
Normal

As we see, we have two classes that correspond to an `abnormal` column spine, so as with some classification training two and just two classes are needed, let's create another data set with just two classes instead of three.

```
#(Named data2 because of the two classes).
data2 <- data %>% mutate(Y=ifelse(Y!="Normal","Abnormal","Normal"))
```

```
#(Renaming data to data3).
data3 <- data

tibble(Y=data2$Y %>% unique()) %>% knitr::kable()
```

Y
Abnormal
Normal

Now, all our data has been prepared for machine learning, let see if it works!

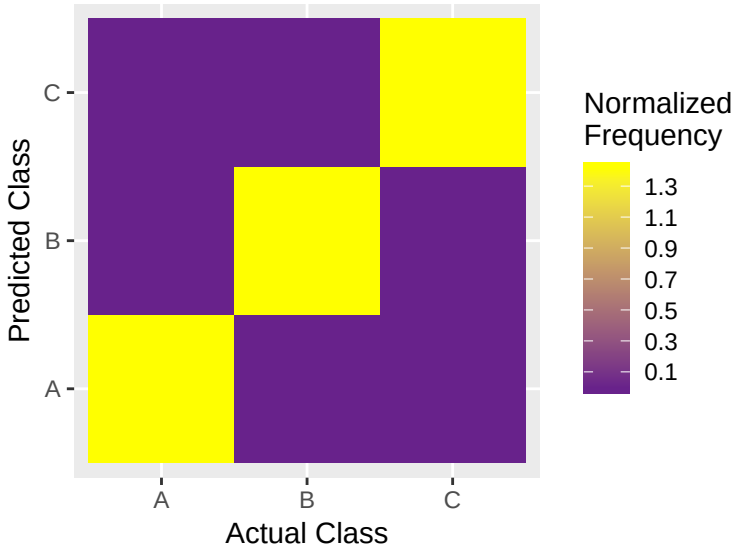
2.2 Tools for analysing the data

With this type of Machine Learning algorithms, is quite interesting plotting its confusion matrix in order to see how well they did it.

Let's create a function to do so:

```
ConfMatrixPlot <- function(Matrix){
  confusion<-as.data.frame(as.table(scale(as.matrix(Matrix),center=FALSE)))
  colnames(confusion)<-c("Reference","Prediction","Freq")
  ggplot(confusion) + geom_tile(aes(x=Reference, y=Prediction, fill=Freq)) +
    scale_x_discrete(name="Actual Class") +
    scale_y_discrete(name="Predicted Class") +
    scale_fill_gradient(breaks=seq(from=-.5, to=4, by=.2),low="darkorchid4",
                        high="yellow") +
    labs(fill="Normalized\nFrequency")
}

#(Confusion Matrix of diagonal matrix)
ConfMatrixPlot(diag(3))
```



As we have seen above, a 3x3 diagonal matrix has been plotted. This would be equivalent to an accuracy of 100%.

2.3 Train and Validation data sets

Since now, the sets had only been modified and prepared, but if we want to apply machine learning to them, it's quite essential have a train and validation set.

In this case, since we don't have an excessive amount of data, a 10% will be randomly selected as the validation set.

```
#(Using seed since probability is going to be used).
set.seed(1)
main_index <- createDataPartition(y = data$Y, times = 1, p = 0.1, list = FALSE)
main_set3 <- data3[-main_index,]
validation_set3 <- data3[main_index,]

main_set2 <- data2[-main_index,]
validation_set2 <- data2[main_index,]

rm(main_index)
```

Now that we already have the validation set, we have to create other train and test sets from the main ones in order to train the algorithm and optimize model parameters.

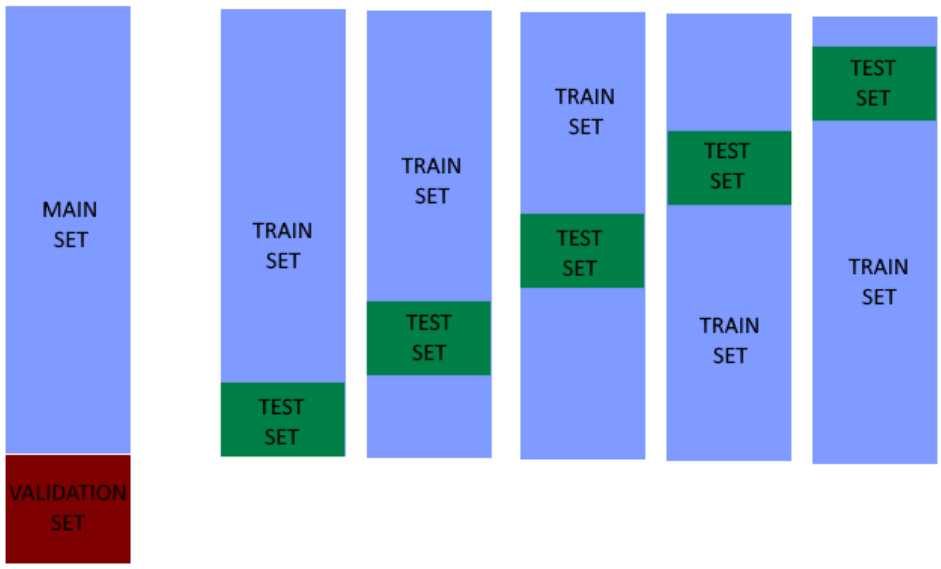


Figure 1: Cross Validation

In this case, five sets from the main one are going to be created.

```
#(Just a random seed)
set.seed(1997)

#(New index to be used)
test_index <- createDataPartition(y = main_set3$Y, times = 5, p = 0.1, list = FALSE)

#(Creating a list of train sets)
trains_set3 <- list(main_set3[-test_index[,1],],main_set3[-test_index[,2],],
                    main_set3[-test_index[,3],],main_set3[-test_index[,4],],
                    main_set3[-test_index[,5],])
```

```
#(Creating a list of test sets)
tests_set3 <- list(main_set3[test_index[,1],],main_set3[test_index[,2],],
                   main_set3[test_index[,3],],main_set3[test_index[,4],],
                   main_set3[test_index[,5],])
rm(test_index)
```

2.4 Principal Component Analysis (PCA)

We are currently working with a model for a data set that has six different variables and tree possible outputs. In this case, if we wanted to represent them, we wouldn't have enough dimensions, so if we really want to represent them in a two or three dimensional system, we have to do a process called dimension reduction.

The first principal component (PC) of a matrix X is the linear orthogonal transformation of X that maximizes its variability. The function prcomp provides us with it.

To begin with, let's see how the correlations of the matrix are.

```
x<-main_set2 %>% select(-Y) %>% as.matrix()
cor(x) %>% knitr::kable()
```

	var1	var2	var3	var4	var5	var6
var1	1.0000000	0.6607131	0.7393974	0.8035493	-0.2398504	0.6418740
var2	0.6607131	1.0000000	0.4572103	0.0841068	0.0373780	0.5604689
var3	0.7393974	0.4572103	1.0000000	0.6189777	-0.0688369	0.6669403
var4	0.8035493	0.0841068	0.6189777	1.0000000	-0.3480362	0.4076358
var5	-0.2398504	0.0373780	-0.0688369	-0.3480362	1.0000000	0.0215308
var6	0.6418740	0.5604689	0.6669403	0.4076358	0.0215308	1.0000000

We see that the logic behind it's quite difficult to understand since some variables are highly related while others are just totally different.

```
pca<-prcomp(x)
summary(pca)
```

	PC1	PC2	PC3	PC4	PC5	PC6
Standard deviation	37.0910	17.1085	11.9443	9.9579	8.7944	0.0000
Proportion of Variance	0.6922	0.1473	0.0718	0.0499	0.0389	0.0000
Cumulative Proportion	0.6922	0.8394	0.9112	0.9611	1.0000	1.0000

We see that with the just first two PC, we are already taking into account nearly 83% of the important effects of our data set.

```
data.frame(pca$x[,1:2], Class=main_set3$Y)%>%
  ggplot(aes(PC1,PC2, fill = Class))+
  geom_point(cex=3, pch=21) +
  coord_fixed(ratio = 1)
```

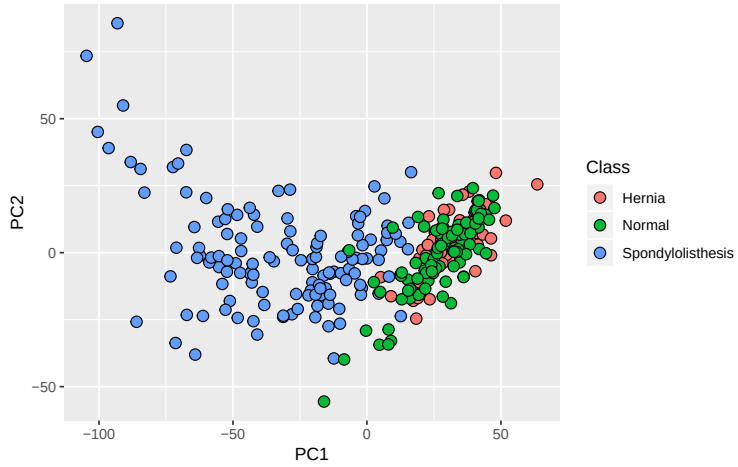


Figure 2: 2D Principal Component Analysis

Looking at the figure above, we see that the blue class, `Spondylolisthesis` would be easier to predict than the others two since they're mixed up. The third PC was near the 7% could exist a difference of Z axis between `Hernia` and `Normal` classes? Let's check it with a 3D plot.

```
temp <- data.frame(pca$x[,1:3], Class=main_set3$Y)

plot_ly(temp,x =temp$PC1, y = temp$PC2, z = temp$PC3, color = temp$Class) %>%
  add_markers() %>%
  layout(scene = list(xaxis = list(title = 'PC1'),
                      yaxis = list(title = 'PC2'),
                      zaxis = list(title = 'PC3')))
```

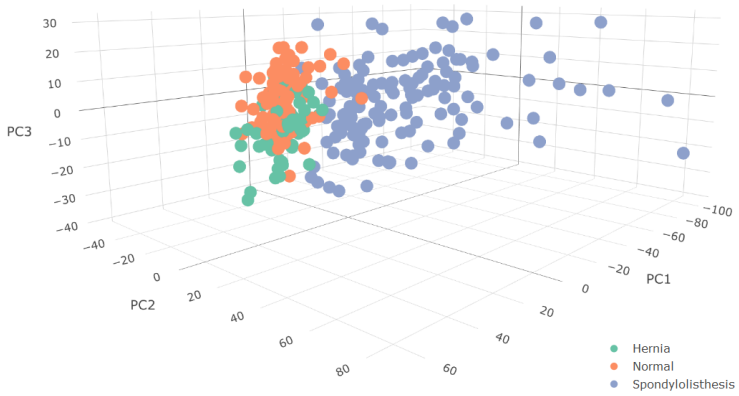


Figure 3: 3D Principal Component Analysis

If you want to interact with it, you could find it at: 3D PCA Plot (html format).

With the two last plots we see that with the neighbor algorithms we will be able to predict `Spondylolisthesis` but not distinguish easily between `Hernia` and `Normal` spine columns.

2.5 Algorithms

The data used in this project is quite unique and specific, reason we couldn't possibly find another similar data set that would have been of any help. The normal column spine of a person isn't easily explained with numerical values.

So, as we really don't know which type of algorithms will better work on this data set, it's quite interesting using some of each type. In this case, the data is going to be tested with: - Distance based methods. (kNN and Centroids) - Probability based methods. (Naïve Bayes) - Rules based methods. (Decision Trees, Forests and AdaBoost) - SVM (Lineal, Radial and Polygonal)

2.5.1 kNN

In Machine Learning there's huge amounts of different algorithms that can be used, but commonly in distance based methods, the two more used are `kNN` and `Centroids`. Let's see if it works as we saw with the PCA.

```
#(Sequence of all sets).
sets<-seq(1:5)

#(Creating a function to aply kNN to each created set)
list<-sapply(sets, function(sets){
  train_temp <- trains_set3[[sets]]
  test_temp <- tests_set3[[sets]]
  #(Number of k that are going to be tested)
  k <- seq(1:50)
  accuracy<-sapply(k, function(k){
    knn_fit <- knn3(as.factor(Y) ~ ., data = train_temp, k = k)
    y_hat_knn <- predict(knn_fit, test_temp, type = "class")
    confusionMatrix(data = y_hat_knn,
                    reference = as.factor(test_temp$Y))$overall["Accuracy"]
  })
  list(tibble(k,accuracy))
})
#(Plotting the results)
ggplot()+
  #(Creating a line for each group of sets).
  geom_line(data=list[[1]],aes(list[[1]]$k,
                               list[[1]]$accuracy,colour="Data set 1"),size=0.8)+
  geom_line(data=list[[2]],aes(list[[2]]$k,
                               list[[2]]$accuracy,colour="Data set 2"),size=0.8)+
  geom_line(data=list[[3]],aes(list[[3]]$k,
                               list[[3]]$accuracy,colour="Data set 3"),size=0.8)+
  geom_line(data=list[[4]],aes(list[[4]]$k,
                               list[[4]]$accuracy,colour="Data set 4"),size=0.8)+
  geom_line(data=list[[5]],aes(list[[5]]$k,
                               list[[5]]$accuracy,colour="Data set 5"),size=0.8)+
  #(Adding colours)
  scale_colour_manual("",values=c("#E69F00","#56B4E9","#009E73","#F0E442","#0072B2"))+
  #(Changing theme)
  theme_light()+  labs(x = "K",y="Accuracy",title="kNN3")+
  #(Adding title and subtitle)
  theme(plot.title = element_text(hjust = 0.5),
        plot.subtitle = element_text(hjust = 0.5,face="italic"))
```

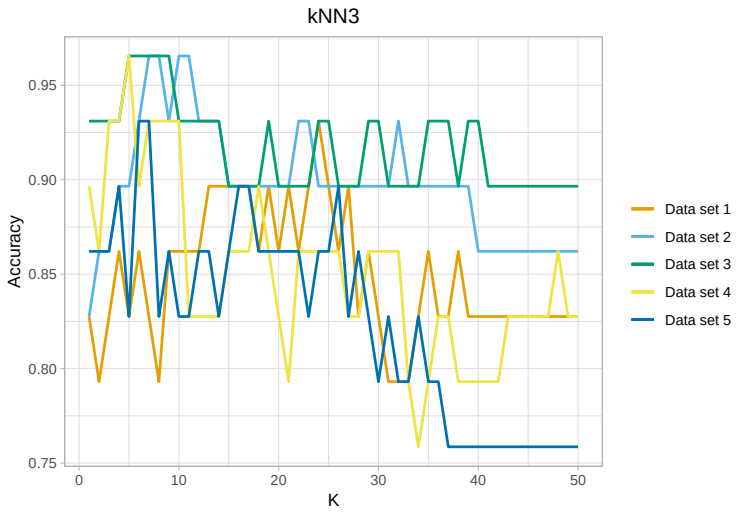


Figure 4: Multiple kNN3 for multiple data sets

As we see, depending on the data set we got more or less accuracy. See for example the comparison between `Data set 5` and `Data set 1`. This is caused because of the neighbors influence between the `Hernia` and `Normal` classes.

Let's use Caret with just one of our test and train sets. Remember, as Model `knn3` is not in the caret's built-in library, we can use `kNN` in its place.

```
#(Set to always get the same results).
set.seed(1997)
#(Training).
train_knn <- train(as.factor(Y) ~ ., method = "knn", data = trains_set3[[1]])
#(Aplying the training to our test set).
y_hat_knn <- predict(train_knn, tests_set3[[1]], type = "raw")
#(Extracting the confusion matrix).
confusion_matrix<-confusionMatrix(y_hat_knn, as.factor(tests_set3[[1]]$Y))

#(Printing the results).
tibble(Accuracy=confusion_matrix$overall[["Accuracy"]]) %>% knitr::kable()
```

Accuracy
0.862069

Plotting its confusion matrix in order to see how well it did.

```
#(Printing the confusion matrix).
ConfMatrixPlot(confusion_matrix$table)
```

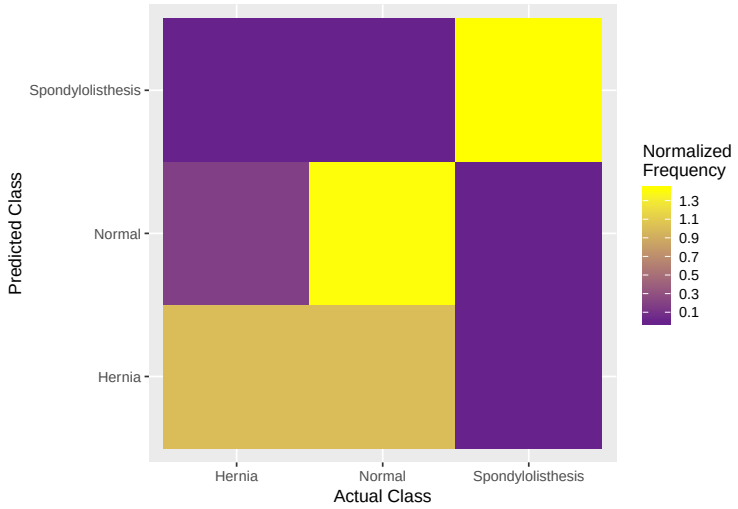


Figure 5: kNN Confusion Matrix

As we see, we got no problem at all with the `Spondylolisthesis` class. This was easily observed with the PCA plots above where we find that this class was far away from the other two that were mixed among them.

If we have to understand our results, we see that the real problem was that we predicted a few users of a hernia, while they had normal column spines.

```
#(Plotting kNN with default k parameter evolution).
ggplot(train_knn, highlight = TRUE)
```

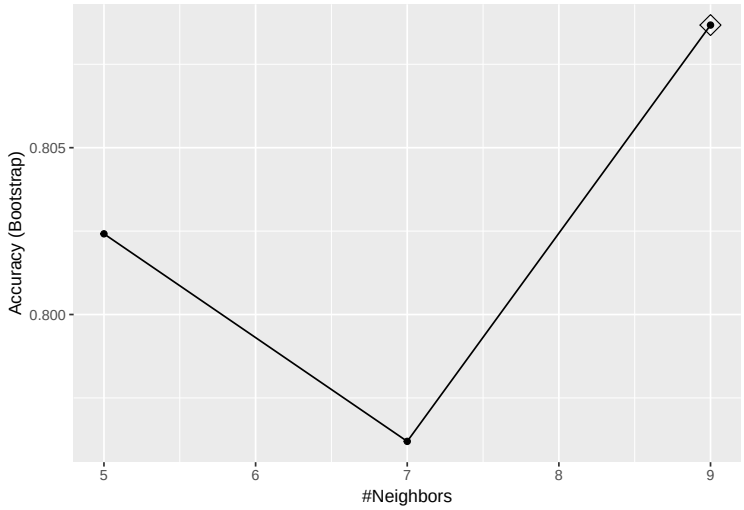


Figure 6: kNN default parameter evolution

Let's now use `Caret Package` automatic cross validation and tunning parameters system.

```
set.seed(1997)
train_knn <- train(as.factor(Y)~ ., method = "knn",
                   data = main_set3,
```

```
                    #(Just a sequence to test different parameters).
                    tuneGrid = data.frame(k = seq(9, 71, 2)))

#(Plotting it's evolution)
ggplot(train_knn, highlight = TRUE)
```

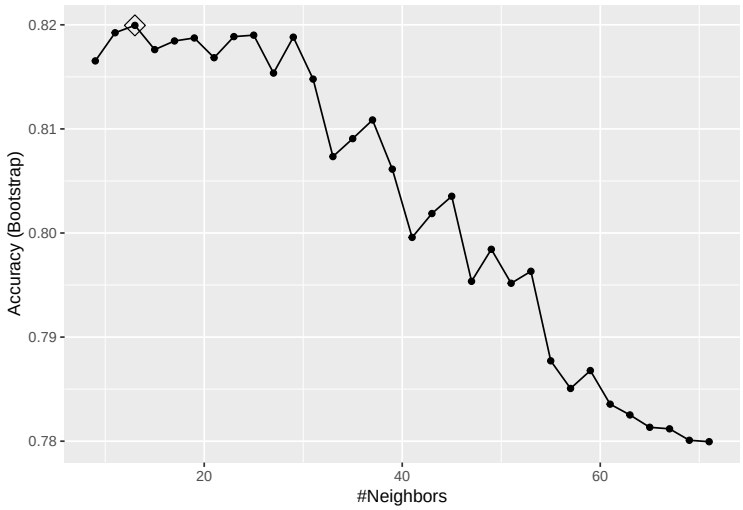


Figure 7: kNN parameter evolution

```
#(BestTune results)
tibble(`Best k`=train_knn$bestTune$k) %>% knitr::kable()
```

Best k
13

```
#(Accuracy results)
tibble(Accuracy=max(train_knn$results$Accuracy),
       SD=train_knn$results$AccuracySD[which.max
                  (train_knn$results$Accuracy)],
       bestTune=paste("k = ",train_knn$bestTune$k,sep=""))%>% knitr::kable()
```

Accuracy	SD	bestTune
0.8199456	0.0456655	k = 13

If we look closely at the k parameter evolution, we see that as the higher the number of neighbors used is, the more mixed the result gets and worse the accuracy is. If we use a group of neighbors quite big, a lot of data from `Hernia` and `Normal` column spine get mixed.

We obtained an accuracy of about `0.8`that's not bad at all, but isn't good enough. It could be a good approximation, but it wouldn't be useful with the definitive model.

2.5.2 Centroids

Centroids are the other distance based method typically used in Machine Learning, let's see how it works with our data.

```
set.seed(1997)
#(Training of Nearest Shrunken Centroids)
train_pam <- train(as.factor(Y)~ ., method = "pam",
                   data = main_set3,
                   #(Just a sequence to test different parameters).
                   tuneGrid=data.frame(threshold=seq(1,9)))

#(Plotting it's evolution)
ggplot(train_pam, highlight = TRUE)
```

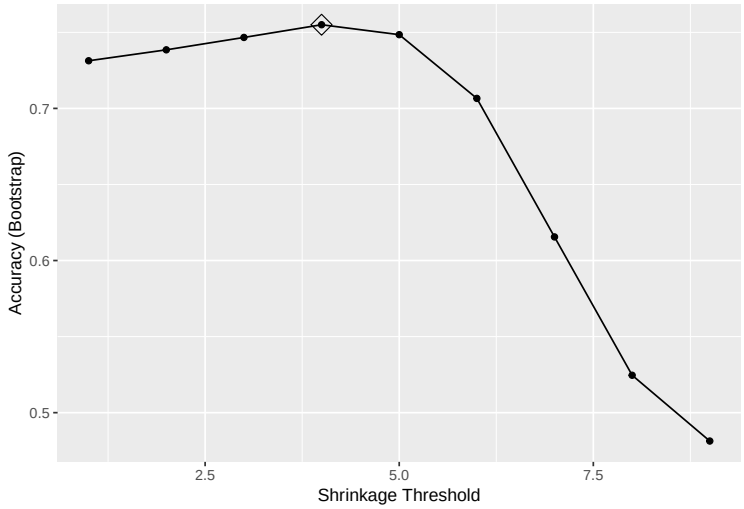


Figure 8: Centroids threshold parameter evolution

As we see and as with neighbors, the biggest is the parameter, the most points are used and the lowest the accuracy is.

```
#(BestTune results)
tibble(`Best Threshold`=train_pam$bestTune$threshold) %>% knitr::kable()
```

Best Threshold
4

```
#(Accuracy results)
tibble(Accuracy =
       max(train_pam$results$Accuracy),
       SD=train_pam$results$AccuracySD[which.max(train_pam$results$Accuracy)],
       bestTune=paste("threshold = ",train_pam$bestTune$threshold,sep=""))%>%
       knitr::kable()
```

Accuracy	SD	bestTune
0.755011	0.0442338	threshold = 4

We got an accuracy about `0.75`, that's a bit worse than `knn`.

2.5.3 Logistic Regression

Aside of distance based methods, there's also methods based on probability. Logistic Regression is an example of it. (It works with two class outputs, so since we got three, we can consider `Hernia` and `Spondylolisthesis` as `Abnormal`).

```
#(Two class outcomes)
#(Creating the formula)
fit_glm <- glm(as.factor(Y) ~ var1+var2+var3+var4+var5+var6,
               data=trains_set3[[1]], family = "binomial")
p_hat_glm <- predict(fit_glm, tests_set3[[1]])

tibble(Class=tests_set3[[1]]$Y,Probability=p_hat_glm)[1:6,] %>% knitr::kable()
```

Class	Probability
Hernia	1.6516067
Hernia	0.0314432
Hernia	-1.7703819
Hernia	-1.9624539
Hernia	0.8560492
Hernia	-1.1322535

```
tibble(Class=tests_set3[[1]]$Y,Probability=p_hat_glm)[22:29,] %>% knitr::kable()
```

Class	Probability
Normal	1.0106228
Normal	2.6156673
Normal	-0.8832481
Normal	2.7051337
Normal	5.0402122
Normal	2.7961253
Normal	1.6819624
Normal	2.9946113

Know that we got a vector of probabilities we just need to define the range for `Normal` and `Abnormal` outputs, but as we see, it doesn't matter how we define the ranges, we will always get trouble trying to distinguish between `Hernia` and `Normal`.

Let's give Caret and `glm` a try!

```
set.seed(1997)
train_glm <- train(as.factor(Y)~ ., method = "glm",
                   data = main_set2)
```

```
tibble(Accuracy=train_glm$results$Accuracy,
       AccuracySD=train_glm$results$AccuracySD) %>% knitr::kable()
```

Accuracy	AccuracySD
0.8363071	0.032368

We got an accuracy of about 0.8 as with the Neighbors model. It's not perfect, but it's a beginning.

2.5.4 Naïve Bayes

Naive Bayes is also a probabilistic method based on applying Bayes's theorem with strong independence between assumptions.

```
set.seed(1997)
train_bayes <- train(as.factor(Y)~ ., method = "naive_bayes",
                     data = main_set3)
tibble(Accuracy=train_bayes$results$Accuracy,
       AccuracySD=train_bayes$results$AccuracySD) %>% knitr::kable()
```

Accuracy	AccuracySD
0.8136826	0.0411154
0.8005406	0.0379548

As we see, it works as others probability methods as Logistic Regression.

But, if we use the Bayesian Generalized Linear Model?

```
set.seed(1997)
train_bayesglm <- train(as.factor(Y)~ ., method = "bayesglm",
                        data = main_set3)

tibble(Accuracy=train_bayesglm$results$Accuracy,
       SDAccuracy=train_bayesglm$results$AccuracySD) %>% knitr::kable()
```

Accuracy	SDAccuracy
0.4161426	0.0446854

Getting such a bad accuracy confirms our theory that our data is not linear and neither can be predicted with distance or simple probability methods.

2.5.5 QDA and LDA

QDA and LDA are two classic classifiers, with, as their names suggest, with a linear or a quadratic decision surface, respectively.

```
set.seed(1997)
#(Train QDA - Caret)
train_qda <- train(as.factor(Y)~ ., method = "qda",
                     data = main_set3)
```

When `QDA` runs, returns `Model fit failed, rank deficiency in group Hernia` error because of the small number of classes we've got.

```
set.seed(1997)
#(Train LDA)
train_lda <- train(as.factor(Y)~ ., method = "lda",
                     data = main_set3)

#(Print results)
tibble(Accuracy= train_lda$results$Accuracy,
```

```
       AccuracySD=train_lda$results$AccuracySD) %>% knitr::kable()
```

Accuracy	AccuracySD
0.8266248	0.0380545

Unlike `QDA`, `LDA` has run successfully reaching an accuracy of about `0.83`.

2.5.6 Recursive Partitioning and Regression Trees

Let's try a few tree's alternatives.

```
set.seed(1997)
#(Running Recursive Partitioning and Regression Trees)
train_rpart <- train(as.factor(Y) ~ .,
                     method = "rpart",
                     tuneGrid = data.frame(cp = seq(0.0, 0.1, len = 25)),
                     data = main_set3)
```

As with most of the rules based methods, we got an accuracy of about `0.80`.

```
#(Plotting Complexity parameter default values evolution)
plot(train_rpart)
```

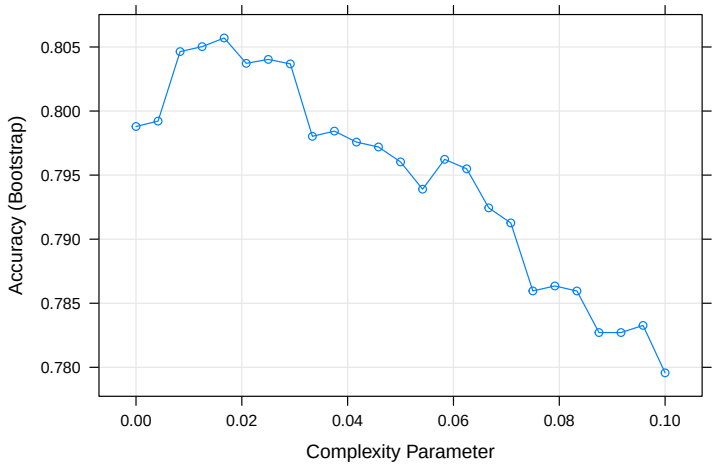


Figure 9: Rpart complexity parameter evolution

```
#(Printing results)
tibble(Accuracy=max(train_rpart$results$Accuracy),
       AccuracySD=train_rpart$results$AccuracySD
       [which.max(train_rpart$results$Accuracy)],
         BestTune=paste("cp = ",train_rpart$bestTune,sep="")) %>% knitr::kable()
```

Accuracy	AccuracySD	BestTune
0.8057019	0.0421866	cp = 0.0167

2.5.7 Rapid Decision Tree Construction and Evaluation

```
set.seed(1997)
#(Running Rapid Decision Tree Construction and Evaluation - Rborist)
train_rborist <- train(as.factor(Y) ~ .,
                       method = "Rborist",
                       #(A bit of tunning parameters)
                       tuneGrid = data.frame(predFixed = 2, minNode = c(3, 50)),
                       data = main_set3)
#(Printing results)
tibble(Accuracy=max(train_rborist$results$Accuracy),
       AccuracySD=train_rborist$results$AccuracySD
       [which.max(train_rborist$results$Accuracy)],
         BestTune=paste("predFixed = ",train_rborist$bestTune[1],
                        " minNode = ",train_rborist$bestTune[2])) %>% knitr::kable()
```

Accuracy	AccuracySD	BestTune
0.8300039	0.0361228	predFixed = 2 minNode = 50

2.5.8 RF

```
#(Applying Random Forests)
set.seed(1997)
train_rf <- train(as.factor(Y) ~ .,
                        method = "rf",
                        data = main_set3)

#(Printing results)
tibble(Accuracy=max(train_rf$results$Accuracy),
       AccuracySD=train_rf$results$AccuracySD
       [which.max(train_rf$results$Accuracy)],
         BestTune=paste("MTYR = ",train_rf$bestTune,sep="")) %>% knitr::kable()
```

Accuracy	AccuracySD	BestTune
0.8252798	0.0348128	MTYR = 2

2.5.9 Adaboost

```
set.seed(1997)
#(Running Adaboost)
train_adaboost <- train(as.factor(Y) ~ .,
                        method = "adaboost",
                        data = main_set2)

#(Printing results)
tibble(Accuracy=max(train_adaboost$results$Accuracy),
       AccuracySD=train_adaboost$results$AccuracySD
       [which.max(train_adaboost$results$Accuracy)],
       BestTune=paste("Iterations =",train_adaboost$bestTune$nIter)) %>% knitr::kable()
```

Accuracy	AccuracySD	BestTune
0.8220755	0.03428	Iterations = 50

Even Adaboost is a based rule algorithm and works providing the data different weights to easily classify, we see that in this case isn't working well.

2.5.10 Others

Just to check if the used algorithms were the adequate ones, all of the following were tested out without reaching a solid solution.

```
#(List of methods)
method<-c("svmLinear3","svmLinearWeights2","svmRadialSigma","svmRadialCost",
          "svmRadial","gbm","monmlp","mlp","avNNet","wsrf","ranger","loclda",
          "Mlda","pda","pda2","stepQDA","bagFDA","BstLm","LogitBoost","C5.0",
          "C5.0Cost","xgbLinear","lvq","svmLinear","gamboost","sda","sparseLDA",
          "dwdPoly","gamLoess","kknn")

#(Function to apply to all the methods)
Others<-sapply(method,function(x){
  set.seed(1997)
  train <- train(as.factor(Y) ~ .,
                 method=x,
                 data = main_set2)
  return(tibble(Accuracy=max(train$results$Accuracy),SD=max(train$results$AccuracySD)))
})
```

Although some of them may have achieved an accuracy of about `0.85` it was principally because of the seed and caret cross validation randomness.

3 Results

A lot of different algorithms were tested in order to find which of them was the most accurate with this specific data set, but the best obtained results were always about `0.80`. This is because of the strange data distribution between `Hernia` and `Normal` output classes.

Remember that since now, all the algorithms were tested on the main set, so we are not sure at 100% of how will they act once tested on the validation set.

Since the obtained results weren't the ideal ones, just in order to learn a bit more about Machine Learning I decided to test a Neuronal Network, at the most basic level, towards the data set.

To do so, `Neuralnet` library and a few documentations were used.

3.1 PCA kNN

Although I decided to test the kNN with the PCA data. As we have previously seen with the first three or four principal components, we can achieve nearly 96% of the data variance. So, will this improve our kNN results?

```
#(Seed)
set.seed(1997)

#(PCA all data)
pca_trainset = main_set2 %>% select( -Y )
dat<-as.matrix(validation_set2 %>% select(-Y))

pca = prcomp( pca_trainset)
train = data.frame( Y = main_set2$Y, pca$x[,1:4] )
test = sweep(dat,2,colMeans(dat)) %*% pca$rotation
test <- test[,1:4]

#(Training data)
train_knn <- train(as.factor(Y)~ ., method = "knn",
                   data = train)
#(Results)
tibble("Caret Accuracy"=max(train_knn$results$Accuracy)) %>% knitr::kable()
```

Caret Accuracy
0.8167356

```
tibble("Real Accuracy"=confusionMatrix(predict(train_knn,test),factor(validation_set2$Y))$overall["Accu
```

Real Accuracy
0.8387097

As we see, the obtained accuracy is similar to all the others, and that's because the data distribution we have in this specific data set.

3.2 Neuronal Network

Let's try a totally different alternative.

```r
#(Seed)
set.seed(1997)

#(Binarize data)
main_set2_nn <- main_set2 %>% mutate(Y=ifelse(Y=="Normal",0,1))
validation_set2_nn <- validation_set2 %>% mutate(Y=ifelse(Y=="Normal",0,1))

#(Creating NN Formula)
n = names( main_set2_nn )
f = as.formula( paste( "Y ~", paste( n[!n %in% "Y"], collapse = "+" ) ) )

#(Neuronal Network training)
nn = neuralnet(f,main_set2_nn,hidden = 4, linear.output = FALSE, threshold = 0.1 )

#(Neuronal Network "predict")
nn.results = neuralnet::compute( nn, validation_set2_nn )
results = data.frame( actual = validation_set2_nn$Y,
                      prediction = round( nn.results$net.result ) )
t = table(results)

#(Print Results)
tibble("Accuracy"=confusionMatrix(t)$overall["Accuracy"]) %>%
        knitr::kable()
```

Accuracy
0.8387097

As opposed to the previous section, in this one the used model Neuronal Network was used against the `validation set` in order to obtain the real results. But, although the obtained value is quite bigger than normal, we see that there's a huge deviation between the lower and upper accuracy values, this is because of the section of the data that's been taken into account on the Caret cross validation. Since cross-validation is based on data partitions and randomness, we could find that there's better train and test set combinations.

Would this system improve its accuracy if we applied the Neuronal Network to a preprocess data with principal component analysis?

3.2.1 PCA Neuronal Network

```r
#(Seed)
set.seed(1997)

#(Preprocessing data)
#(Getting all the data without the output)
pca_trainset = main_set2_nn %>% select( -Y )
dat<-as.matrix(validation_set2_nn %>% select(-Y))

#(Computing the PCA)
pca = prcomp( pca_trainset)

#(Applying pca to our data)
train = data.frame( Y = main_set2_nn$Y, pca$x[,1:4] )
test = sweep(dat,2,colMeans(dat)) %*% pca$rotation
```

```r
test <- test[,1:4]

#(Creating NN Formula)
n = names( train )
f = as.formula( paste( "Y ~", paste( n[!n %in% "Y" ], collapse = "+" ) ) )

#(Neuronal Network training)
nn = neuralnet( f, train, hidden =4, linear.output = FALSE, threshold = 0.1)

#(Neuronal Network "predict")
nn.results = neuralnet::compute( nn, test )

#(Results)
results = data.frame( actual = validation_set2_nn$Y,
                      prediction = round( nn.results$net.result ) )
t = table(results)

#(Print Results)
tibble("Accuracy"=confusionMatrix(t)$overall["Accuracy"]) %>%
        knitr::kable()
```

Accuracy
0.8709677

As we see, our accuracy has suffered quite an interesting increase in its value with just a simple preprocessing method as PCA. `0.87` is possibly not enough, but at least it's a really good improvement. Our model just failed 4 out of the 31 cases from the validation set.

4 Conclusions

As we see, the final obtained results are not the desired ones, although with Neuronal Network we improved a bit our accuracy we still got lots of problems. The data from this set isn't possibly the best one, there's some `var` with such a high correlation that basically affect our data and difficult the outputs classification.

In a situation like this one is highly recommend analyzing with more detail all the variables that are being used, and try to redefine the structure of our data deleting columns, adding new ones or modifying others.

We found a critical point where we weren't able to distinguish between two of the three classes. With this in mind and the accuracy being so low, we would not be able to assure with confidence that the created model it's working well.

Even it has been quite a tricky project that hasn't reached positive results, I'm quite satisfied with all done. Has been an interesting way of introducing ourselves on this wide and complex Machine Learning world that's full of possibilities and alternatives and it has forced me into learning more about Machine Learning and revise all what I had previously studied. So, unlikely of the results, I have extended my knowledge and I will possibly continue doing some more Machine Learning algorithms on my free time.