EDX-Course: Data Science - Own Project: Guitar Chords

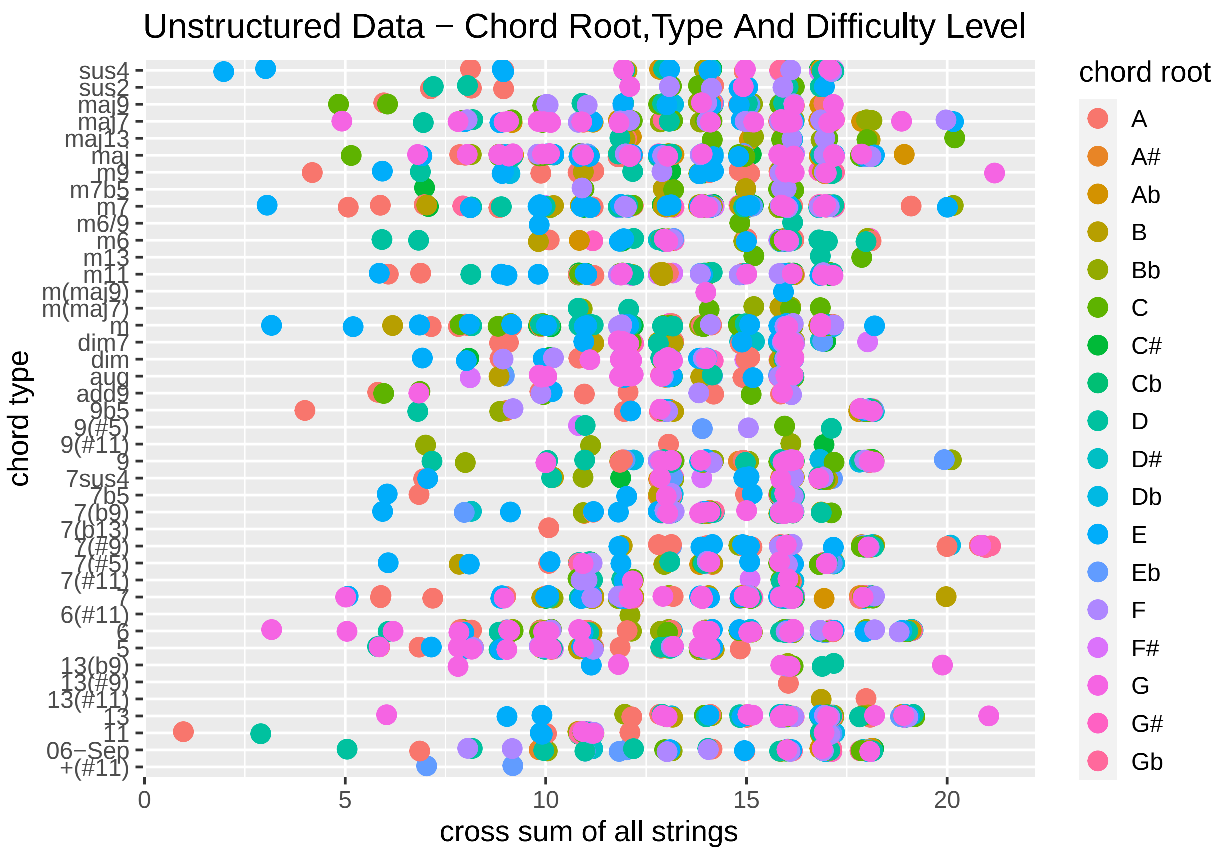
Haas Nicolas

07 12 2020

Introduction

Learn how to play guitar is for every beginner a challenge. Especially, the way how you can learn it can be different. For example you can play one "chord" (a set of notes played on a guitar) in different ways so that you need to know which finger position is most suitable to you when you start your passion. Nevertheless, I will not go too deep into musical understanding. This piece of work is more focused on the idea whether the finger position of a chord can help to predict "the root" of a chord. The root is one of the main labels of a chord such as "A", "B", etc. The finger position explains how the 6 strings are played. For example an open string ("0"), a mutated string ("x"), the first finger is on a string ("1"), the second finger is on a string ("2"), etc. Probably the chord type ("moll", "dur","sus", etc.) can also help to improve predictions. Though, the challenge is far from being easy. A first figure given by "Unstructured data - Chord root,type And difficulty level" already shows that the same kind of chord can be played by many finger positions. Hence, getting reasonable predictions for a large data set is only be possible in using a data split that divides beginner chords from more advanced ones.

```r
chord_fingers_new %>% ggplot(aes(cross, CHORD_TYPE,col=CHORD_ROOT)) +
  geom_point(size=3, position=position_jitter(h=0.1,w=0.2)) +
  xlab("cross sum of all strings") + ylab("chord type") + scale_color_discrete(name = "chord root") +
theme(legend.key.size = unit(0.5, "cm")) +
ggtitle("Unstructured Data - Chord Root,Type And Difficulty Level")
```



Therefore, this analysis looks more into beginner chords (chords that contain easier finger positions) and uses more general strategies (logistic regression/k nearest neighbors and decision trees) for predictions. At the end, the results will show that the first and second string can predict a certain root type with an accuracy of about 79 %.

Data set

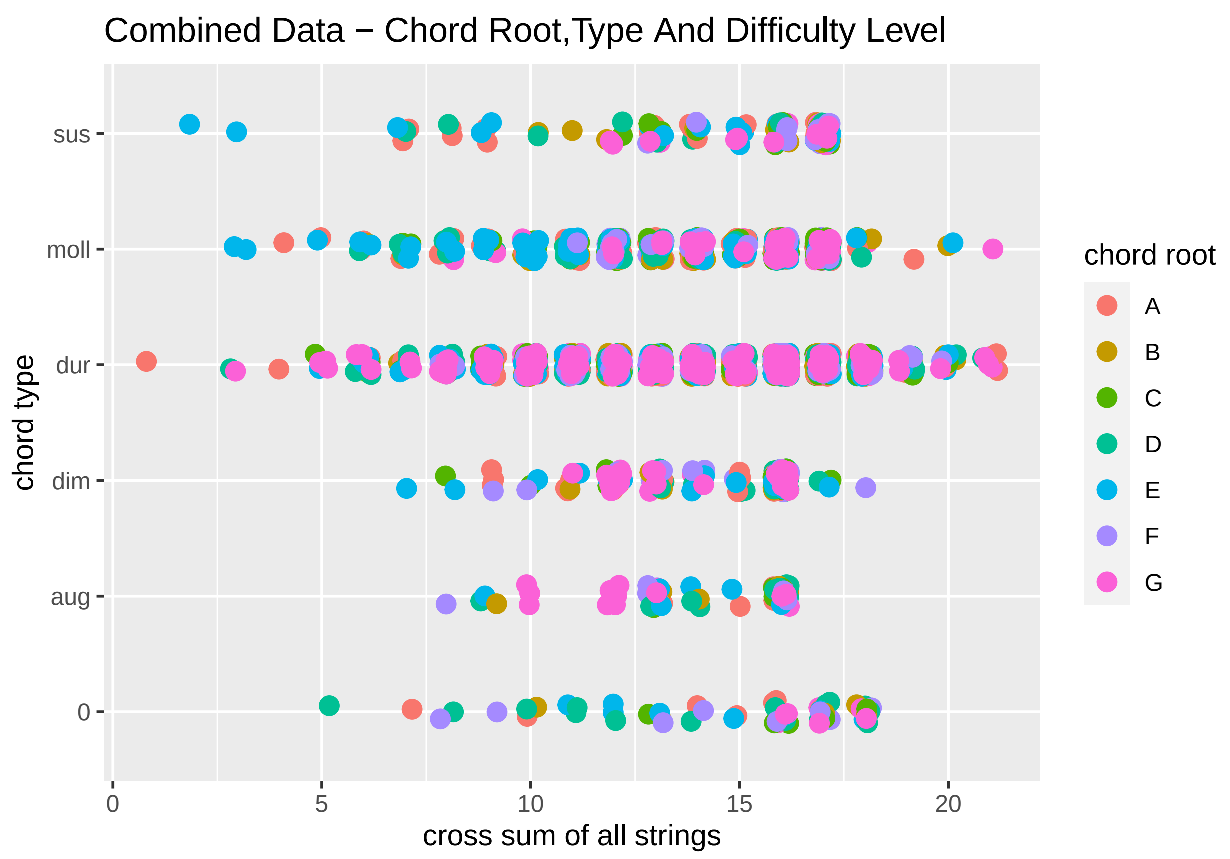
The analysis uses a data set that was published by the UCI ("Machine Learning Repository") and the original data contains five variables - root,type,finger position,structure and note name.You can find the first five observations of the original data below.

```
##   CHORD_ROOT CHORD_TYPE  CHORD_STRUCTURE FINGER_POSITIONS          NOTE_NAMES
## 1         A#         13 1;3;5;b7;9;11;13      x,1,0,2,3,4    A#,C##,G#,B#,F##
## 2         A#         13 1;3;5;b7;9;11;13      4,x,3,2,1,1    A#,G#,B#,C##,F##
## 3         A#         13 1;3;5;b7;9;11;13      1,x,1,2,3,4    A#,G#,C##,F##,B#
## 4         A#      7(#9)      1;3;5;b7;#9      x,1,0,2,4,3    A#,C##,G#,B##,E#
## 5         A#      7(#9)      1;3;5;b7;#9      2,1,3,3,3,x    A#,C##,G#,B##,E#
## 6         A#      7(#9)      1;3;5;b7;#9      1,3,1,2,1,4 A#,E#,G#,C##,E#,B##
```

This analysis decided to work only with the variables CHORD_ROOT, CHORD_TYPE and FINGER_POSITION whereas we will see later that the type of a root will also not be used for predictions (Was not a good predictor). Finally, the dependent variable is the root of a chord and the independent variables are the string variables which represent each one a string, categorized/factored by the finger that is played (open string, mutated string,first finger,etc.). For reasons of handiness, the analysis divided the data in easy and difficult finger positions and only continued with beginner positions (determined by the sum of the factor level of all strings). To understand this step much better you can see in figure "Combined Data - Chord Root, Type And Difficulty Level" that after combining the root and the type of a chord in different groups a first classification is visible. The left side of the figure shows less variation in colors than the right side which includes almost all colors. Because this piece of work is interest in a machine learning mechanism for beginner, finger positions which had a difficulty level of 11 or more have been removed before starting the model predictions.

A last explanation according to the difficulty level. The difficulty level has been built on the idea it is much more difficult for a beginner to play a string with the pinkie than with the ring finger. Therefore the level has the order: 0=open string,1=mutated string,2=First finger on a string,3=middle finger on a string,4=ring finger,5=pinkie. The cross variable is then simply the sum of the factor level of all strings so that a lower cross sum stands for an easier string than a higher value.

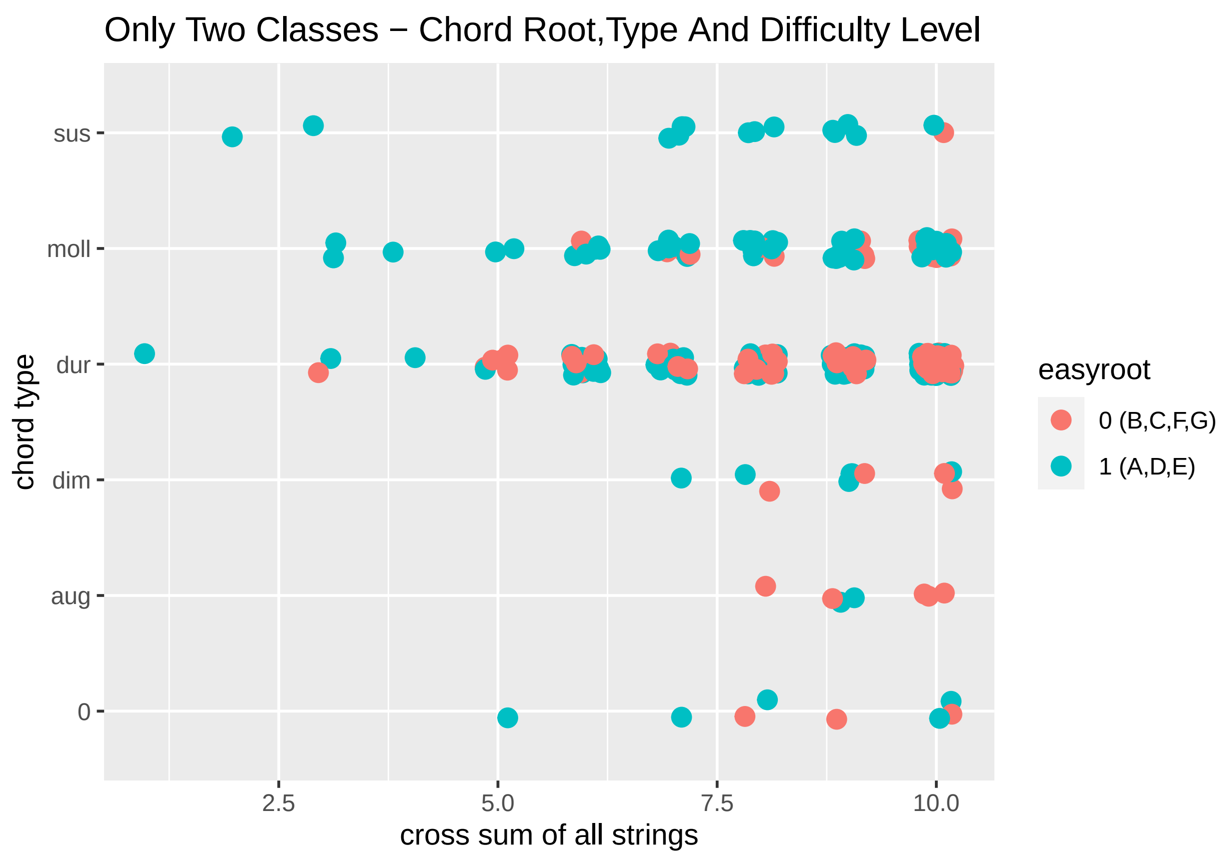
```r
rm("c")
chord_fingers_new %>% ggplot(aes(cross, type,col=root)) +
  geom_point(size=3, position=position_jitter(h=0.1,w=0.2)) +
  xlab("cross sum of all strings") + ylab("chord type") +
scale_color_discrete(name = "chord root",labels = c("A", "B", "C", "D", "E", "F","G")) +
ggtitle("Combined Data - Chord Root,Type And Difficulty Level")
```



At the end, the data set contains of 317 observations but includes all main types (dur, sus, moll, aug, dim) and two main groups of roots (First group: A,D,E and second group: B,C,F,G). The reason why this analysis used two main groups of roots can be explained by a lack of power if it would have looked only to two roots (For example A and B). The two main groups are represented by the "easyroot" variable which is equal to 1 if the root is A,D or E and 0 otherwise. Hence, the last figure "Only Two Classes - Chord Root,Type and Difficulty Level" represents the main data set of this analysis.

```r
chord_fingers_new<-chord_fingers_new%>%mutate(easyroot=ifelse(root==1 | root==4 | root==5,1,0))
chord_fingers_new$easyroot<-as.factor(chord_fingers_new$easyroot)
```

```r
chord_fingers_new %>% filter(cross<11) %>% ggplot(aes(cross, type,col=easyroot))+
geom_point(size=3, position=position_jitter(h=0.1,w=0.2)) +
xlab("cross sum of all strings") + ylab("chord type") +
  scale_color_discrete(name = "easyroot",labels = c("0 (B,C,F,G)", "1 (A,D,E)")) +
  ggtitle("Only Two Classes - Chord Root,Type And Difficulty Level")
```



Model and Method

The main goal of this analysis is to find a classification model that can help to predict a root of a chord when only one or two strings are played by a beginner. Summary statistics show that specifically the first and the second string seem to be important predictors.

```r
chord_fingers_new %>% filter(cross<11) %>%
group_by(easyroot,string_1) %>% summarize(n=n()) %>% as.data.frame()
```

```
## `summarise()` regrouping output by 'easyroot' (override with `.groups` argument)

##    easyroot string_1   n
## 1         0        0   4
## 2         0        1  85
## 3         0        2  21
## 4         0        3  14
## 5         0        4  11
## 6         1        0  47
## 7         1        1 124
## 8         1        2   6
## 9         1        3   4
## 10        1        4   1
```

```r
chord_fingers_new %>% filter(cross<11) %>%
group_by(easyroot,string_2) %>% summarize(n=n()) %>% as.data.frame()
```

```
## `summarise()` regrouping output by 'easyroot' (override with `.groups` argument)

##    easyroot string_2  n
## 1         0        0  5
## 2         0        1 54
## 3         0        2 36
## 4         0        3 19
## 5         0        4 21
## 6         1        0 63
## 7         1        1 55
## 8         1        2 28
## 9         1        3 20
## 10        1        4 15
## 11        1        5  1
```

We can see that class 1 has more observations that uses an open string (0) or a mutated string (1) compared to the class 0. With this idea in mind, the model uses these two predictors, string_1 and string_2, which stand for the finger position of the first and second string and tries to predict the "easyroot" class. The easyroot level has only two classes as already described (1: A,D,E and 0:B,C,F,G) and therefore allows for predictions regarding a class of roots.

The main model has the following shape:

g{Pr(Easyroot=1|string_1=string_1,string_2=string_2)} = b0 + b1 * string_1 + b2 * string_2

where easyroot := {0,1} with 0: B,C,F,G and 1:A.D,E string_1: level of the first string (level: a factor between {0,1,2,3,4,5}) string_2: level of the second string (level: a factor between {0,1,2,3,4,5}) b0: An intercept which does not depend on the level of string_1 and string_2 b1: Coefficient for string_1 b2: Coefficient for string_2

This analysis is therefore interested in estimating the conditional probability of a certain root type which depends in this case on the two predictors, string_1 and string_2.

The first algorithm, logistic regression, makes use of the logistic transformation to put each data point in one of the two classes (easyroot is equal to 0 or 1). Because this analysis assumes that a certain cut-off can divide every data point in one of these two classes, the logistic transformation can be used to talk about a probability how much more likely it is to see one class compared to the other. Therefore the measure of accuracy in this case is simply defined by the proportion of correct predictions over total predictions.

The second-algorithm, decision tree or partition tree, also uses the same outcome variable (easyroot) and the same predictors but it is a bit different compared to logistic regressions. Instead of using one straight line to divide all data points, it works with several cut-offs/lines. It therefore works more in terms of a tree to divide efficiently all observations in different branches. It divides all observations in different groups/branches in simply using for each branch a different cut-off and then asks for every observation within a group/branch whether the value is below or above the cut-off. The high value of interpretability will help that we can much better understand how the classification is done at the end.

Results

As mentioned in the last chapter logistic is reasonable when you are working with categorical data and you want to use two predictors. Because the data set contains 317 observations it is not too small but also not very large so that I used a general split for a data partition (80 % for training and 20 % for test). The results of logistic regression are already quite good (Overall Accuracy:0.79 and Balanced Accuracy:0.78 so that specificity and Sensitivity are also balanced).

```
chord_fingers_new<-chord_fingers_new %>% filter(cross<11)
library(caret)
library(rpart)
chord_fingers_new$string_1<-as.numeric(chord_fingers_new$string_1)
chord_fingers_new$string_2<-as.numeric(chord_fingers_new$string_2)
chord_fingers_new$easyroot<-as.factor(chord_fingers_new$easyroot)
set.seed(2, sample.kind = "Rounding")
```

```
## Warning in set.seed(2, sample.kind = "Rounding"): non-uniform 'Rounding' sampler
## used
```

```
test_index <- createDataPartition(chord_fingers_new$easyroot, times = 1, p = 0.8, list = FALSE)
test_set <- chord_fingers_new %>% slice(-test_index)
train_set <- chord_fingers_new %>% slice(test_index)
glm_fit <- train_set %>%
  mutate(y = as.numeric(easyroot==1)) %>%
  glm(y ~ string_1+string_2, data=., family = "binomial")
p_hat_logit <- predict(glm_fit, newdata = test_set, type = "response")
y_hat_logit <- ifelse(p_hat_logit > 0.56, 1, 0) %>% factor
confusionMatrix(y_hat_logit, as.factor(test_set$easyroot))$overall[["Accuracy"]]
```

```
## [1] 0.7936508
```

```
confusionMatrix(y_hat_logit, as.factor(test_set$easyroot))
```

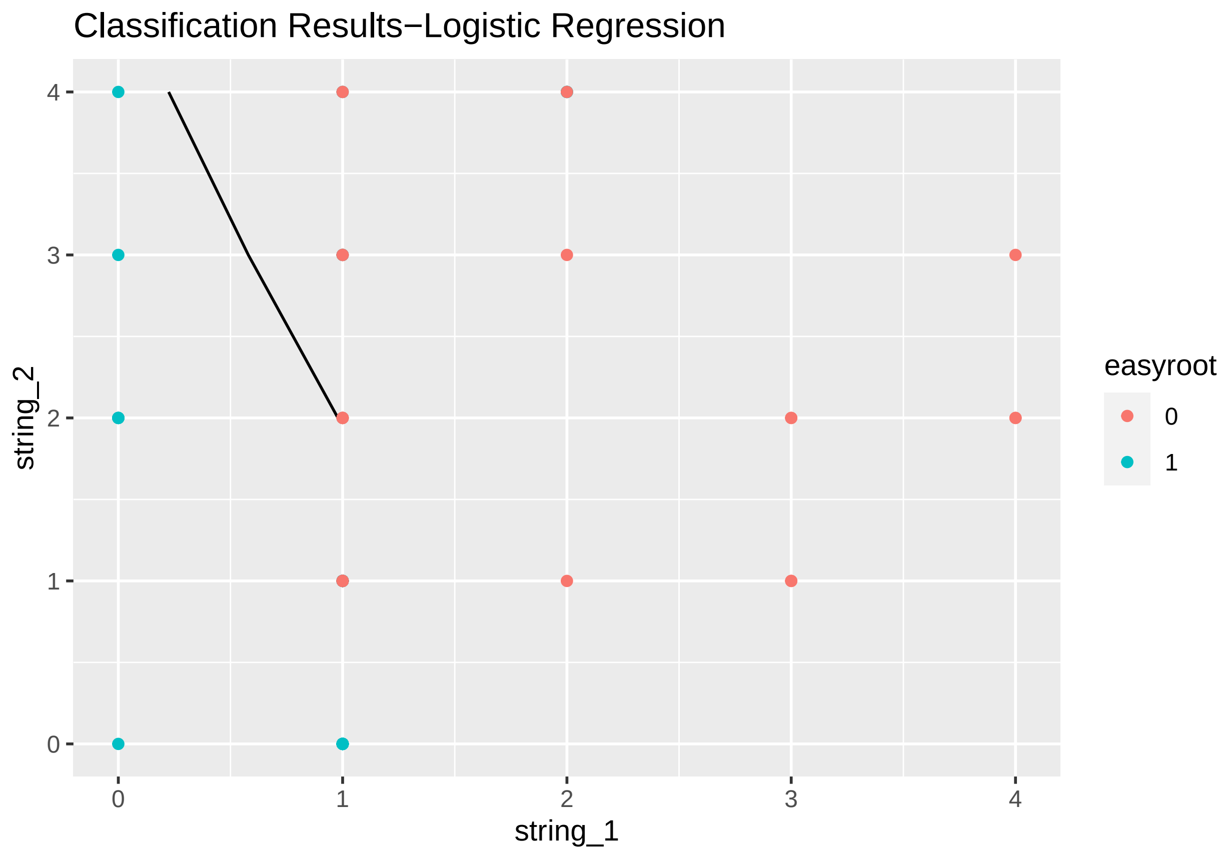
```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction  0  1
##          0 20  6
##          1  7 30
##
##                Accuracy : 0.7937
##                  95% CI : (0.673, 0.8853)
##     No Information Rate : 0.5714
##     P-Value [Acc > NIR] : 0.00018
##
##                   Kappa : 0.5767
##
##  Mcnemar's Test P-Value : 1.00000
##
##             Sensitivity : 0.7407
##             Specificity : 0.8333
##          Pos Pred Value : 0.7692
##          Neg Pred Value : 0.8108
##              Prevalence : 0.4286
##          Detection Rate : 0.3175
##    Detection Prevalence : 0.4127
##       Balanced Accuracy : 0.7870
##
##        'Positive' Class : 0
##
```

For illustration purposes, you can find below the figure "Classification Results-Logistic Regression which shows the cut-off of 0.56 that has been chosen by the algorithm (in using the test set). It shows that easyroot for a value of 1 (roots A,D,E) is more visible in the space where the first string will be 0 (open played first string - most left of the figure). On the other hand, for a value of 0 it is more likely to be to the right part of the figure.

```
rm("c")
```

```
## Warning in rm("c"): Objekt 'c' nicht gefunden
```

```
test_set %>%
  mutate(p_hat = p_hat_logit) %>%
  ggplot() +
  stat_contour(aes(string_1, string_2, z=p_hat), breaks=c(0.56), color="black") +
  geom_point(mapping = aes(string_1, string_2, color=easyroot), data = test_set) +
  ggtitle("Classification Results-Logistic Regression")
```



One can now ask if another algorithm such as K-nearest neighbors (KNN) couldn‘t have led to more precision in simply incorporating more predictors or including more neighbors when making predictions. But the answer is no, see for example the results below when adding a third string and also using KNN with 5 neighbors (The same is true when using all strings as predictors).

```
fit <- knn3(easyroot ~ string_1+string_2+string_3,data = train_set)
y_hat_knn <- predict(fit, test_set,type="class")
confusionMatrix(y_hat_knn, as.factor(test_set$easyroot))$overall["Accuracy"]
```

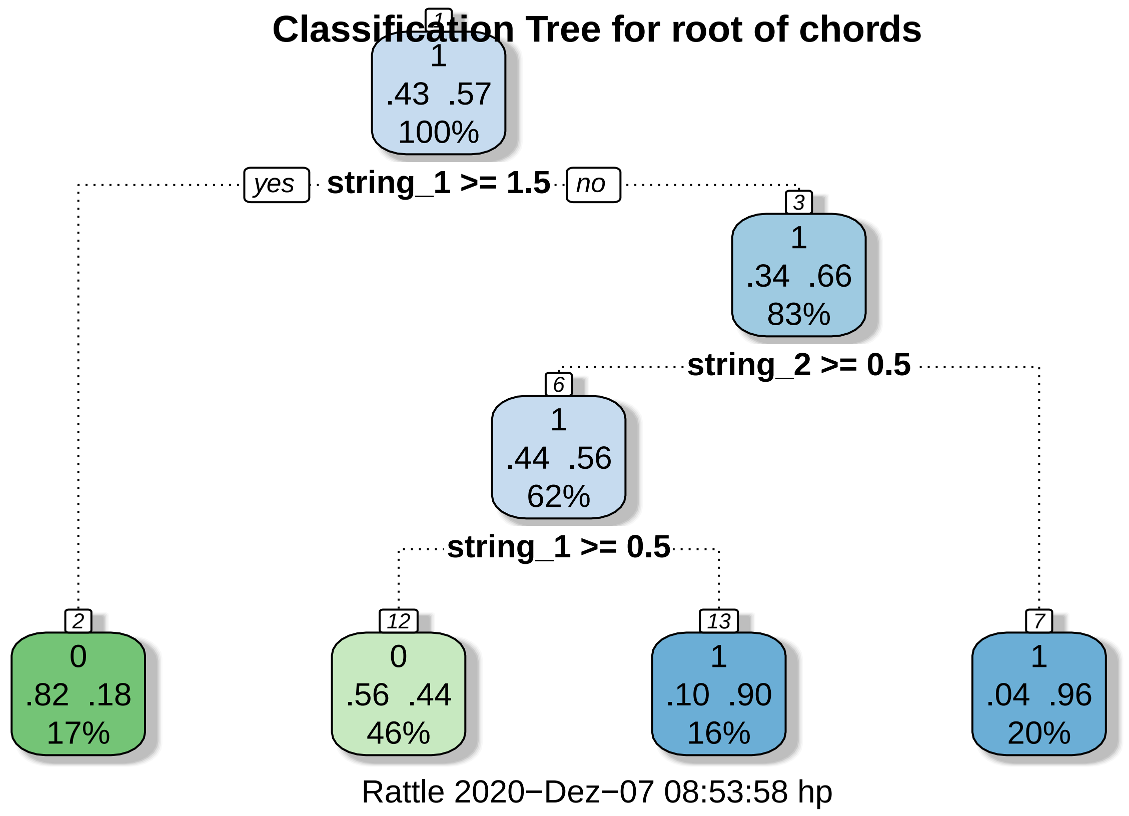
```
##  Accuracy
## 0.7301587
```

Although logistic regression already gives consistent predictions, a decision tree can eventually help to increase accuracy by some few proportions. Unfortunately, the results of the decision tree doesn‘t lead to higher accuracy. it is actually less precise, overall accuracy: 73%.

```r
fit <- rpart(easyroot ~ string_1+string_2,data = train_set,method="class")
library(bitops)
library(rattle)
library(rpart.plot)
library(RColorBrewer)
y_hat <- predict(fit, test_set,type="class") %>% factor()
confusionMatrix(y_hat, as.factor(test_set$easyroot))
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction  0  1
##          0 27 17
##          1  0 19
##
##                Accuracy : 0.7302
##                  95% CI : (0.6035, 0.8343)
##     No Information Rate : 0.5714
##     P-Value [Acc > NIR] : 0.0068123
##
##                   Kappa : 0.4893
##
##  Mcnemar's Test P-Value : 0.0001042
##
##             Sensitivity : 1.0000
##             Specificity : 0.5278
##          Pos Pred Value : 0.6136
##          Neg Pred Value : 1.0000
##              Prevalence : 0.4286
##          Detection Rate : 0.4286
##    Detection Prevalence : 0.6984
##       Balanced Accuracy : 0.7639
##
##        'Positive' Class : 0
##
```

```r
fancyRpartPlot(fit,main="Classification Tree for root of chords")
```



The decision tree algorithm mostly predicts a 0 (chords class: B,C,F,G) and only in the case that the first string is played open or mutated the algorithm predicts a 1 (in 36 of 100 cases). This is result stays in accordance to the figure given by the logistic regression approach. It is interesting result in the way that an open string or a not played string (mutated string) are easier to recognize for a machine learning mechanism because they need less human interaction. For example if a machine learning system can‘t precisely listen to the chord that has been played by a beginner but it can still hear the open/mutated first string, then he could give a small tip to a learner according to the class that the mechanism assumes.

Conclusion

The starting point of the analysis was building an algorithm that could help to predict the root of a guitar chord in using the finger position. Nevertheless, chords can be played in different ways so that the same chord can use different finger positions. This led to the assumption that getting good predictions would only be possible in using a data split or looking for more variables (such as a band variable which indicates on which band the finger is positioned). But finding suitable variables is far from being easy. Hence, the first possibility to split the data was more suitable to this analysis and it therefore looked only to finger positions which are much easier to play ("beginner"). Though, a second issue was then to define the two classes of outcome. Comparing two roots explicitly would have led to a lack of power to train and test the data set. Therefore, the final solution was to use the beginner data set and group the root in two different groups - 0 for root A,D,E and 1 for B.C.F,G. Only in using this strategy this analysis could gain predictions which could deliver an accuracy of 79 % (No problem between specificity and sensitivity). At the end, the logistic regression was the most accurate algorithm (79 % accuracy) but not so easy to understand what was going on. The decision tree gave much more inside and specifically showed that an open or a mutated played first string play an important role in the classification of both classes. Therefore, this information gain can probably help to understand what a beginner is trying to play only by listening to the first string.

The limitation of this analysis is especially the data set which should be adapted by further information in terms of for example the band on which a finger positioned. This information could probably have led to a better classification for the whole data set. Moreover, a good classification also depends on insights given by musicians. The way to classify the level of the finger position in this analysis was one way to do it and there are certainly more possibilities to define the level of the finger positions. Although, the algorithm could only be applied to a restricted data set, the accuracy was at the end close to 80 % (logistic regression). The general message, that the first string plays an important role in the recognition of a chord can probably be a good hint for future research.