Stat_490_Paper

Andrew Maloney

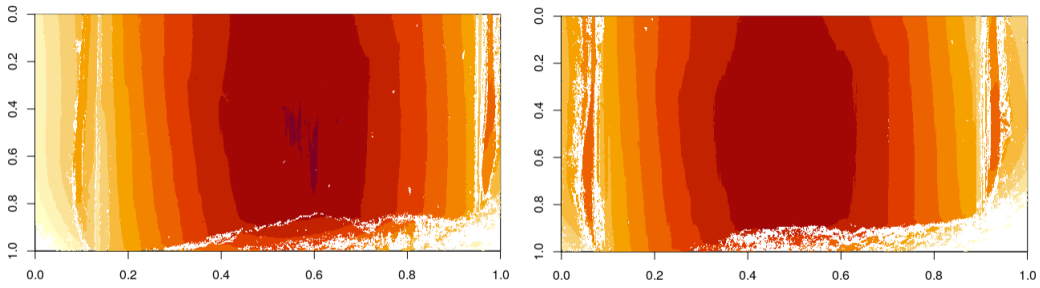
5/5/2020

Consecutive Matching Striae

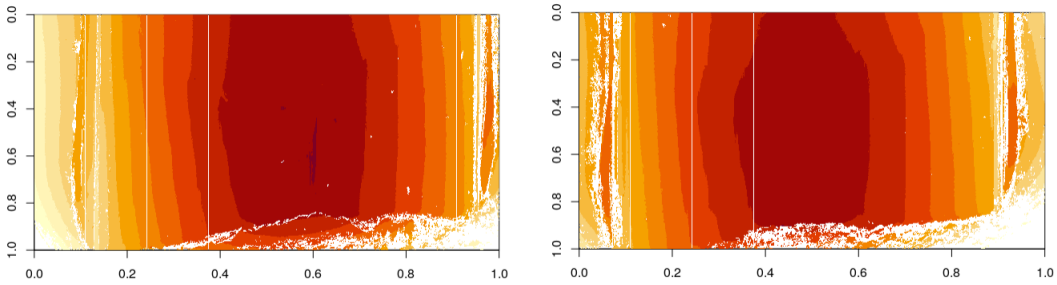
Produced in R Markdown

- When a bullet is fired from a gun, there are distinct markings left on the bullet. These markings are generally caused by the spiral engravings inside of the barrel, the igniting of the powder propellant, or foreign matter inside the barrel. These distinctive markings allow for firearm examination to be conducted. Currently, the comparison of two bullets is a subjective process involving simultaneous visual examination of two bullets by a trained practitioner. Using 3D microscopy, the bullets can be numerically characterized and the matching process can be automated. When the bullet has been numerically characterized we are left with a set of quantitative 'features', each with there own column in a data set. One of these features is known as CMS(Consecutive Matching Striae) and we will examine the underlying structure of how the CMS is calculated, how changing parameters used to calculate CMS affect the final calculation result for CMS. We expand the scope of the matching algorithm by fitting a generalized random forest model using a wide range of different CMS calculation. With this expanded model, we expect that the algorithm's results using different CMS calculations will lead to new insights on how 'important' CMS is as a feature and how different CMS impact the Accuracy of two bullets being a match or not a match. We perform all of our calculations using the R Programming Language. Before we examine the feature known as consecutive matching striae, we will need to perform a number of steps with the bullet scans.
- For this paper we use the Phoenix data set which consist of 198 scanned bullets. The scanned bullets are in a .x3p file format consisting of the surface profile between adjacent groove impressions on the bullet. These scans can be read into R using the R package x3ptools. For this study we will exam two datasets. One Phoenix dataset will have its x3p files containing the scanned bullets down sampled using the x3p_sample() function(See Figure 2). When down sampling is performed the number of pixels in each x3p file are reduced down by a half vertically and horizontally. The other Phoenix dataset will have no down sampling performed to the x3p files containg the scanned bullets(See Figure 1). We examine the scaled measurements of each bullet after being loaded into R. We use the function x3p_get_scale() from the x3ptools package to see what the scale is for each of the bullet scans in both datasets. The original measurement scale of the bullet depends on the microscope and scanner used to scan the bullet before being converted into an .x3p file format. The Phoenix dataset with no sampling has a resolution of 0.654 and the sampled Phoenix dataset has a resolution of 1.29. This makes sense because reducing the pixels by a half will result in a larger micron measurement. *Note: as a final check we make sure that the base of the bullet are at the bottom of the scan. The bottom of the bullet scan is known as the 'heel' of the bullet.*

(Figure 1): Regular image of scanned Bullet



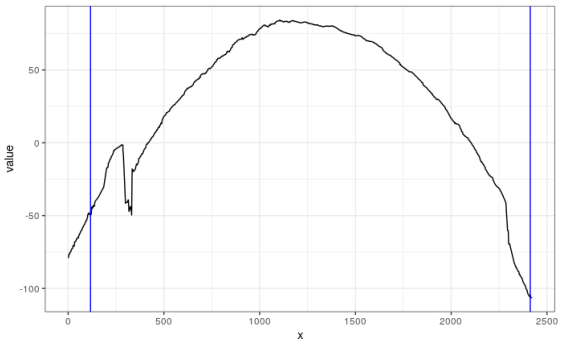
(Figure 2): Image of scanned Bullet with Reduced Pixels



- When comparing bullet scans that were down sampled versus bullet scans that were not down sampled, we see that down sampling removes some tank rash and other markings from the bullet's surface. Thus, resulting in smoother images of the scanned bullets.
- We are now able to analyze the surface scan, by taking a cross section of the surface measurements. This can be performed using the x3p_crosscut_optimize() function and then next the x3p_crosscut() function. Both of which are in the bulletxtrctr R package. *Note: We will refer to the surface scan as a land impression from now on.* When plotting the crosscuts we are left with a side profile of the

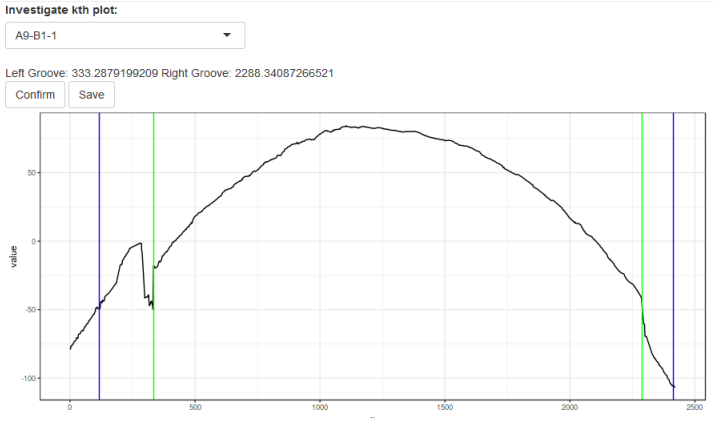
land impressions. Notice the shape of the side profile, it resembles a perfect semi circle. If we think about the shape of a bullet and look at surface of the bullet outside its casing, we can take a pencil and sketch a 180 degree line along the bullet. Which will give us the same semi circular shape. Also notice the two hills on the left and right side of the cross sections. These hills are known as shoulders and at the valleys of these shoulders is where the land impression starts and ends. The shoulders contain no useful information to date and are removed(See Figure 3).

(Figure 3): Side Profile of land impression before correctly identifying groove cutoffs



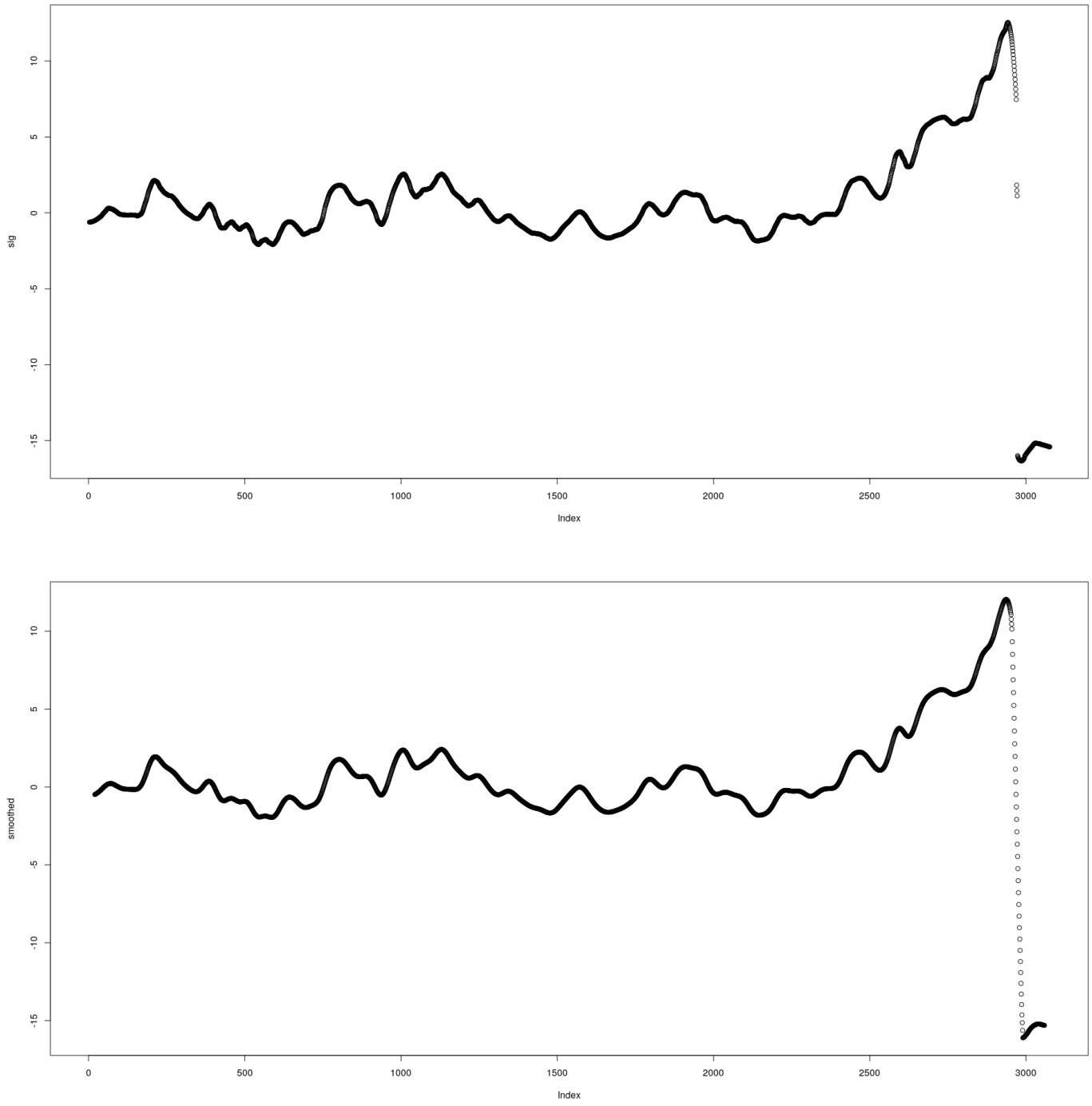
- We begin the process of removing the shoulders. First we identify the shoulder locations of each land impression. Second we smooth the land impression. We then remove the shoulders of the land impression by specifying the left and right shoulder cutoffs. Currently this process is trying to be automated using the cc_locate_grooves() function in the bulletxtrctr package but for this study we manually removed the shoulders of the smoothed land impression using a shiny app(See Figure 4).

(Figure 4): Shiny displaying the new groove cutoffs.

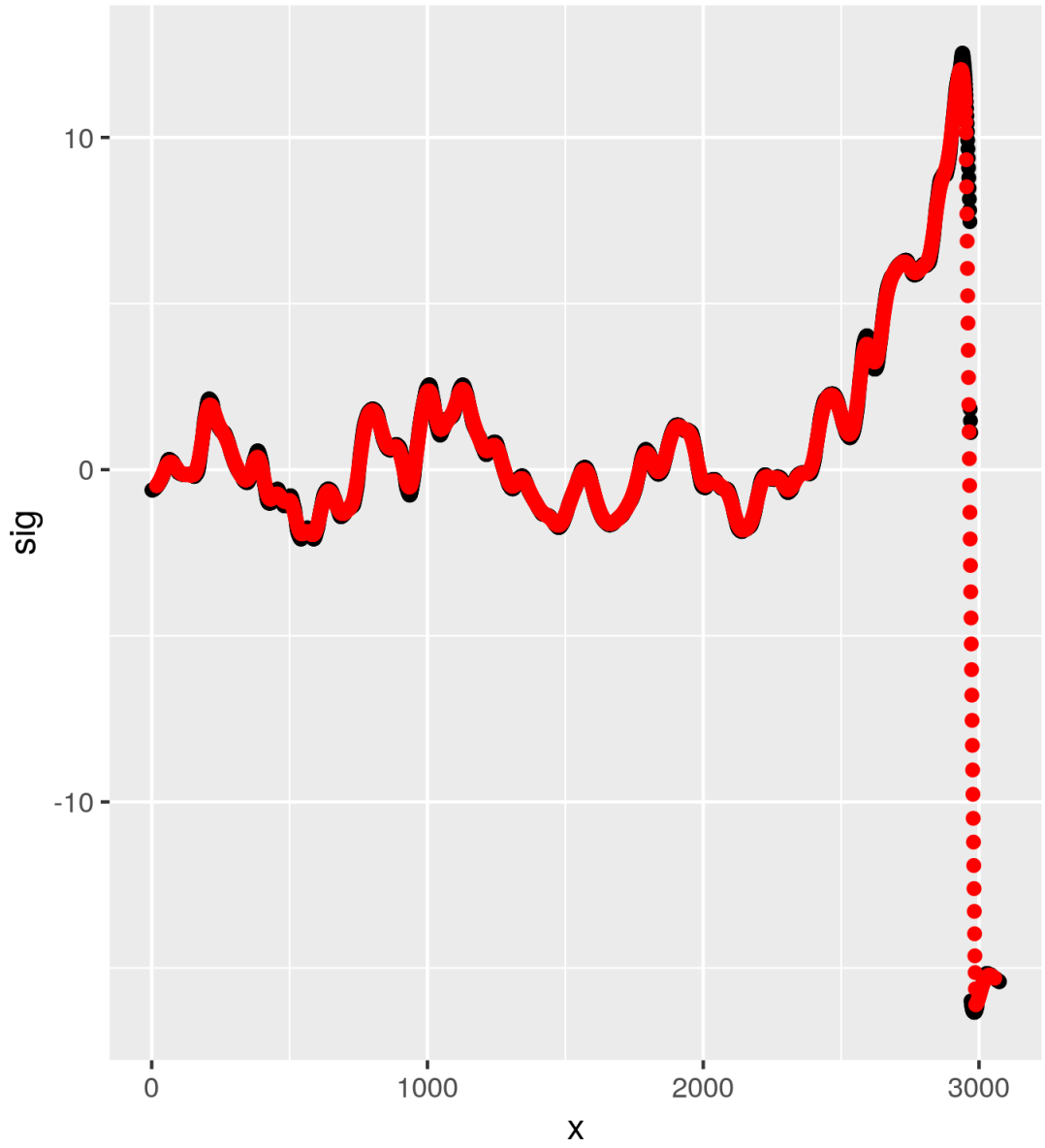


- After we have identified the groove cutoffs we look for the lowest stable region for each land impression and extract the signatures. The signatures can be obtained using the cc_get_signature() function in the bulletxtrctr package.
- In the beginning of this paper we talked about bullets being numerically characterized, which we said were features. One of these features is CMS, also known as consecutive matching striae. We are now able to extract the CMS and all the other features associated with a bullets land impression. We first smooth the signatures(See Figure 8 & 9), identifiy peaks and valleys of each signature(See Figure 10), and align every signature with eachother(See Figure 11). When the two signatures are aligned we are able to identify the matching peaks and valleys which coincide to the striation marks left on a bullet's surface after being fired(See figure 11). This is done by using the sig_aligned() function in the bulletxtrctr package. Notice the word "striation," that's right these are our consecutive matching striae. We can calculate the consecutive matching striae using the sig_cms_max() function in the bulletxtrctr package.

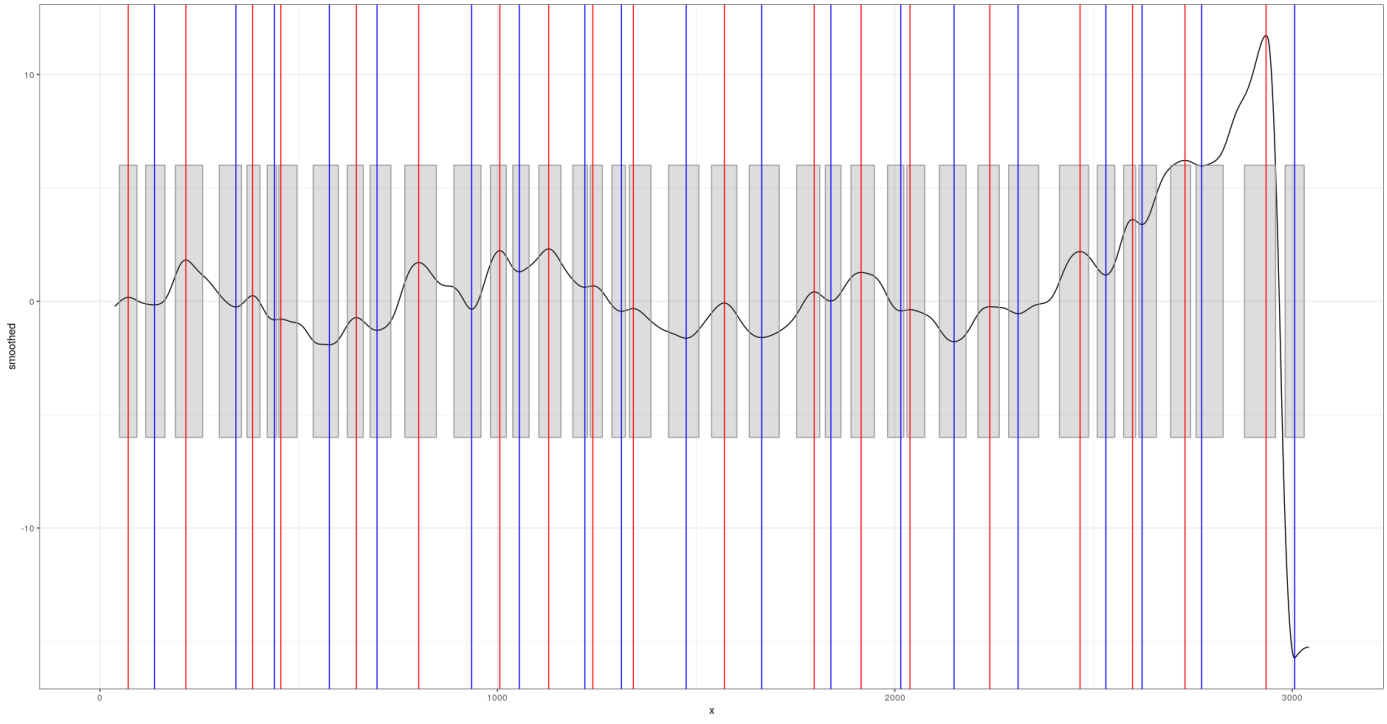
(Figure 8): Show Smoothed Vs Non Smoothed(span = default)



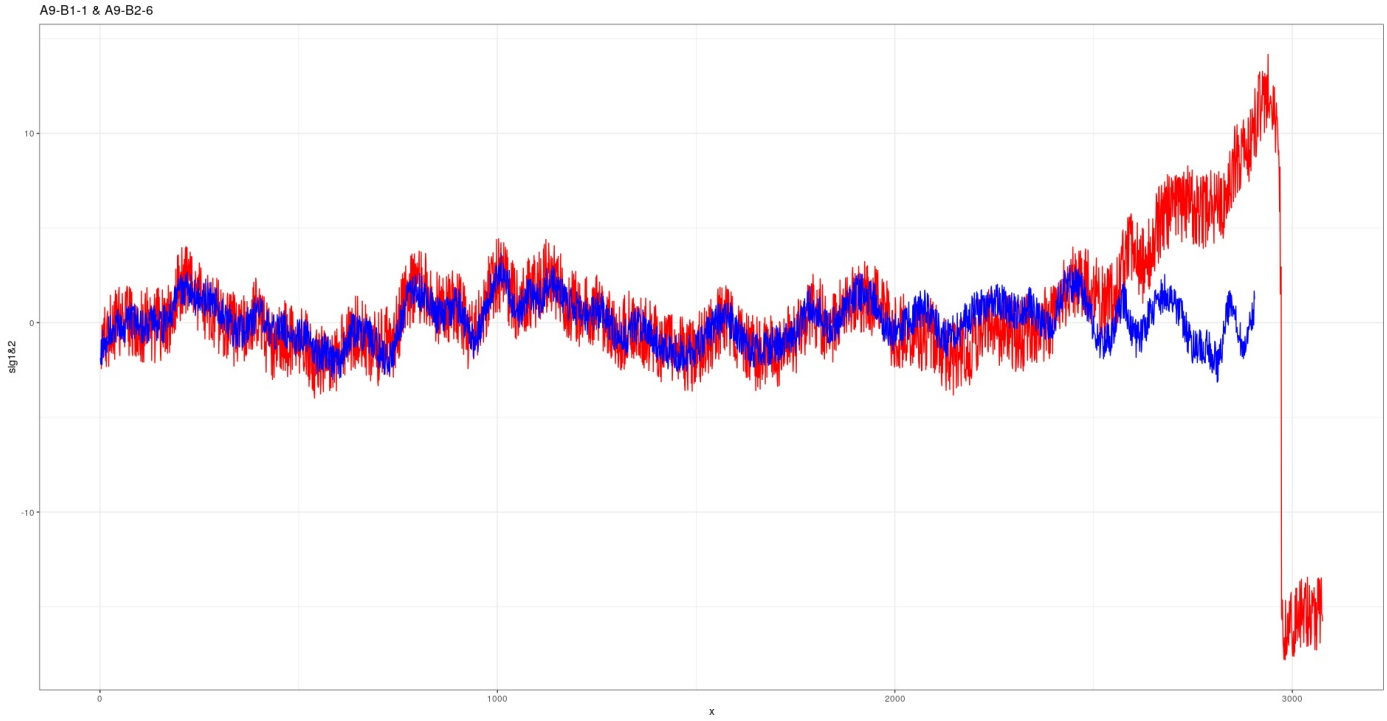
(Figure 9): Smoothing Overlayed



(Figure 10): Show identified peaks and valleys



(Figure 11): Show Aligned Signatures



(Figure 11): cite and show actually markings associated with cms)

- We explore the maximum number of CMS using different values for the span parameter in sig_cms_max(). We run 8 different spans and compare the Maximum number of CMS

Summary Statistics

Same_Comparisons_Removed using dataframe where no sampling was performed

```
#Same_Comparisons_Removed <- Same_Comparisons_Removed %>% select(-X)

head(Same_Comparisons_Removed, 8)
```

```
##   X Span Min X1st.Qu. Median  Mean X3rd.Qu. Max.
## 1 1   15   0        2      3 2.872        4   34
## 2 2   25   0        2      2 2.827        3   37
## 3 3   35   0        2      2 2.776        3   32
## 4 4   45   0        2      2 2.745        3   37
## 5 5   55   0        1      2 2.685        3   34
## 6 6   65   0        1      2 2.569        3   34
## 7 7   75   0        1      2 2.442        3   25
## 8 8   85   0        1      2 2.349        3   23
```

Same_Comparisons Removed using dataframe where sampling was performed

```
head(Same_Comparisons_Removed_Samp, 8)
```

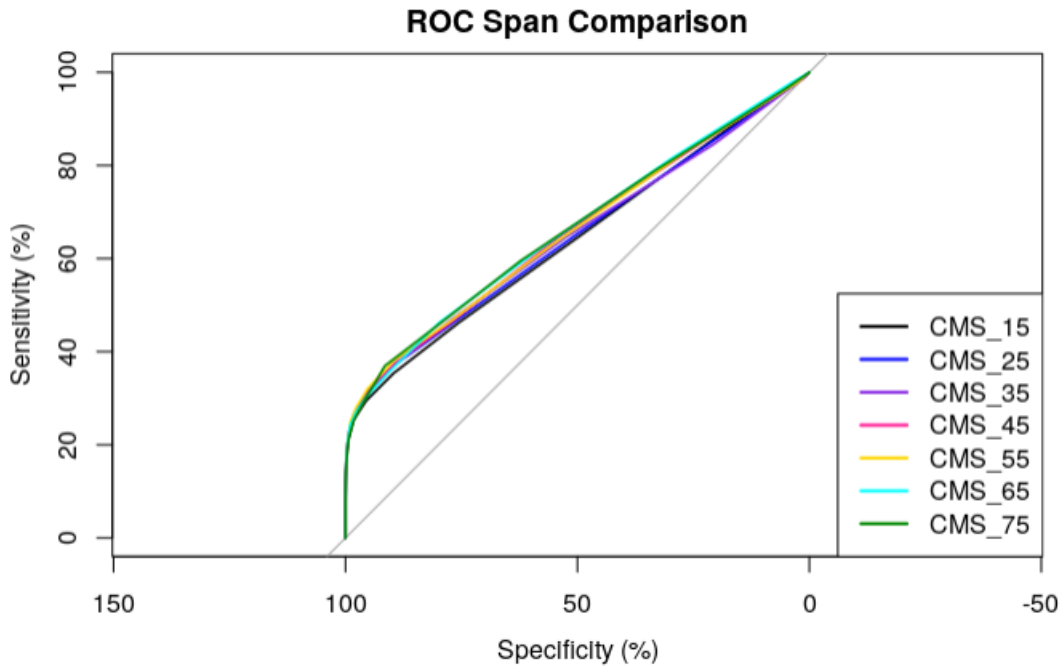
```
## # A tibble: 8 x 7
##    Span  Min. `1st Qu.` Median  Mean `3rd Qu.`  Max.
##   <dbl> <dbl>     <dbl>  <dbl> <dbl>     <dbl> <dbl>
## 1    15     0         2      2  2.77         3    38
## 2    25     0         2      2  2.70         3    37
## 3    35     0         1      2  2.51         3    38
## 4    45     0         1      2  2.32         3    22
## 5    55     0         1      2  2.22         3    21
## 6    65     0         1      2  2.12         3    20
## 7    75     0         1      2  1.98         3    16
## 8    85     0         1      2  1.87         3    15
```

- We first removed Same Land Comparisons from both the nonsampled and sampled dataframes. This is to ensure we remove bullet land comparisons we know are already matches. We can think of these as sorta outliers and they could have negative effects when it

comes to our future random forest model's performance when distinguishing the majority and minority class values.

- Looking at the CMS statistics for Same_Comparisons_Removed the lower values of span have higher mean and max statistics. Looking at the CMS statistics for Same_Comparisons_Removed_Samp the lower values of the span have higher mean and max statistics. Comparing the CMS values for bullet land scans that did not have sampling performed with bullet land scans that did have sampling performed; we see that sampling tends to lower the overall mean cms value. Looking at the Max CMS values for bullet land scans where sampling was performed we see higher CMS values at the lower values of span compared to the bullet land scans that did not have sampling performed on them. This could make our previous statement less justifiable. These results suggest we perform less smoothing or at least avoid very high span values when we smooth.

Class Seperatability based on CMS span values



Distinct

AUC Scores for dataframe where sampling was not performed

Span of 15: Area under the curve: 0.6414

Span of 25: Area under the curve: 0.6498

Span of 35: Area under the curve: 0.6487

Span of 45: Area under the curve: 0.6577

Span of 55: Area under the curve: 0.6611

Span of 65: Area under the curve: 0.6614

Span of 75: Area under the curve: 0.6606

Span of 85: Area under the curve: 0.6602

AUC Scores for dataframe where sampling was performed

Span of 15: Area under the curve: 0.6496

Span of 25: Area under the curve: 0.6623

Span of 35: Area under the curve: 0.6587

Span of 45: Area under the curve: 0.6519

Span of 55: Area under the curve: 0.6537

Span of 65: Area under the curve: 0.6487

Span of 75: Area under the curve: 0.644

Span of 85: Area under the curve: 0.6378

- Notice, for all the span values we see slight separability. The CMS feature does seem to explain some of the models ability to distinquish between Matchs and Non-matches. All AUC scores were in the mid 60's. AUC scores differ between bullet land scans

where sampling was not performed and bullet land scans where sampling was peformed. For bullets where sampling was not performed we see a span of 65 having the highest AUC value and for bullets where sampling was performed we see a span of 25 having the highest AUC value.

Random Forest Model

- Data where span = 55 was used to train and test our random forest model. We train a model using the down sampled bullet scans and we train another model using the data where the bullet scans were not sampled. We also train each using the full dataset where the same bullet comparisons are kept and where the same bullet comparisons are removed. So we have a total of 2 x 2 = 4 models. Feature selection and cross validation were used for training and testing the random forest models. This was done using a custom cross validation loop that applies feature selection to the training data within each fold. Previous studies, important variables determined by random forest, and boruta feature selection suggested using columns: ccf, rough_cor, D sd_D, matches, mismatches, cms, non_cms, and sum_peaks out of the (20~25) original columns. Looking at the results, we notice that using data where the same land comparisons were kept lowers the random forest models overall performance. We see slightly better performance when we remove the same comparisons, leaving only distinct values. The results also show that down sampled bullet scans hurt our random forest models over all performance. We only tried down sampling using x3p_sample(m = 2, mY = 2), there might be better results using different parameter values for x3p_sample()… to be continued. We show the output of the confusion matrix and calculate precision, recall, F-measure, and Class Balance Accuracy to get a better understanding of our models performance. The models ability to distinguish between matches and non-matches is quite low. This is most likely due to class imbalance from our target variable.

Model Summary RF

Random Forest

Span 55 where FALSE is the positive class value

Same Comparisons kept

```
Precision = 7357/(7357+187) = .9752

Recall = 7357/(7357+133) = 0.9822

F-measure(Harmonic Mean) = 2(7357)/(2(7357)+187+133) = 0.9787

CBA_p = 7357/max{7357+187, 7357+133} = .9752

CBA_n = 163/max{163+187, 163+133} = 0.4657

CBA = (CBA_p + CBA_n)/2 = 0.7204
```

————Reference————

Prediction FALSE TRUE

```
FALSE  7357  187

TRUE    133  163
```

Span 55 where FALSE is the positive class value

Same comparisons removed

```
Precision = 3747/(3747+104) = 0.9729

Recall = 3747/(3747+1) = 0.9997

F-measure(Harmonic Mean) = 2(3747)/(2(3747)+104+1) = 0.9861

CBA_p = 3747/max{3747+104, 3747+1} = 0.9729

CBA_n = 48/max{48+104, 48+1} = 0.3157

CBA = (CBA_p + CBA_n)/2 = 0.6443
```

————Reference————

Prediction FALSE TRUE

```
FALSE  3747  104

TRUE      1   48
```

Accuracy : 0.9731

Span 55 sample run where FALSE is the positive class value

Same comparisons kept

```
Precision = 7338/(7338+205) = 0.9728

Recall = 7338/(7338+145) = 0.9806

F-measure(Harmonic Mean) = 2(7338)/(2(7338)+205+145) = 0.9767

CBA_p = 7338/max{7338+205, 7338+145} = 0.9728

CBA_n = 152/max{152+205, 152+145} = 0.4257

CBA = (CBA_p + CBA_n)/2 = 0.6992
```

————Reference————

Prediction FALSE TRUE

```
FALSE  7338  205

TRUE    145  152
```

Span 55 sample run where FALSE is the positive class value

Same comparisons removed

```
Precision = 3738/(3738+121) = 0.9686

Recall = 3738/(3738+5) = 0.9986

F-measure(Harmonic Mean) = 2(3738)/(2(3738)+121+5) = 0.9834

*CBA_p = 3738/max{3738+121, 3738+5} = 0.9686*

*CBA_n = 36/maax{36+121, 36+5} = 0.2292*

CBA = (CBA_p + CBA_n)/2 = 0.5989
```

————Reference————-

Prediction FALSE TRUE

```
FALSE  3738  121

TRUE      5   36
```

Accuracy : 0.9677

Conclusion

Code Appendix

```
image(Phoenix$x3p[[7]])
ggsave("No_Sampling.png")

Phoenix <- Phoenix %>% mutate(x3p = map(x3p, sample_x3p, m = 2)) # Down Sampling

image(Phoenix$x3p[[7]])
ggsave("Sampling.png")
```

```
Phoenix <- Phoenix %>%
  mutate(CrossSection = map_dbl(x3p, x3p_crosscut_optimize))
```

```
Phoenix <- Phoenix %>%
  mutate(CrossCut = map2(.x = x3p, .y = CrossSection, .f = x3p_crosscut))

crosscuts_phoenix <- select(Phoenix, -path, -x3p) %>%
      tidyr::unnest(CrossCut)
```

```
library(shiny)
if (file.exists(saved_grooves_location_Phoenix)) {
  Phoenix$Grooves <- readRDS(saved_grooves_location_Phoenix)
}
if (interactive()) { # only run when you're manually running chunks... don't run when the whole document is
compiled.
  shinyApp(
    ui = fluidPage(
      selectInput("k", "Investigate kth plot:",
                  selected = 1,
                  choices = (1:length(Phoenix$Grooves)) %>% set_names(Phoenix$id)
      ),
      textOutput("groovelocations"),
      actionButton("confirm", "Confirm"),
      actionButton("save", "Save"),
      plotOutput("groovePlot", click = "plot_click"),
      verbatimTextOutput("info")
    ),

    server = function(input, output, session) {
      output$groovePlot <- renderPlot({
        k <- as.numeric(input$k)
        p <- Phoenix$Grooves[[k]]$plot

        p
      })
      output$groovelocations <- renderText({
        paste(
          "Left Groove: ", Phoenix$Grooves[[as.numeric(input$k)]]$groove[1],
          " Right Groove: ", Phoenix$Grooves[[as.numeric(input$k)]]$groove[2]
        )
      })
      observeEvent(input$confirm, {
        cat(paste(Phoenix$id[as.numeric(input$k)], "\n"))
        updateSelectInput(session, "k", "Investigate kth plot:",
                          selected = as.numeric(input$k) + 1,
                          choices = (1:length(Phoenix$Grooves)) %>% set_names(Phoenix$id)
        )
      })
      observeEvent(input$save, {
        saveRDS(Phoenix$Grooves, file = saved_grooves_location_Phoenix)
        message("groove data saved\n")
      })

      observeEvent(input$plot_click, {
        k <- as.numeric(input$k)
        xloc <- input$plot_click$x

        gr <- Phoenix$Grooves[[k]]$groove
        if (abs(gr[1] - xloc) < abs(gr[2] - xloc)) {
          Phoenix$Grooves[[k]]$groove[1] <<- xloc
        } else {
          Phoenix$Grooves[[k]]$groove[2] <<- xloc
        }
        output$groovePlot <- renderPlot({
          k <- as.numeric(input$k)
          p <- Phoenix$Grooves[[k]]$plot +
            geom_vline(xintercept = Phoenix$Grooves[[k]]$groove[1], colour = "green") +
            geom_vline(xintercept = Phoenix$Grooves[[k]]$groove[2], colour = "green")

          p
        })
```

```
      })
      output$info <- renderText({
        paste0("x=", input$plot_click$x, "\ny=", input$plot_click$y)
      })
    },
    options = list(height = 500)
  )
  saveRDS(Phoenix$Grooves, file = saved_grooves_location_Phoenix)
} else {
  if (!file.exists(saved_grooves_location_Phoenix)) {
    message("run script in interactive mode to fix grooves")
  } else {
    Phoenix$Grooves <- readRDS(saved_grooves_location_Phoenix)
  }
}
```

```
Phoenix <- Phoenix %>%
 mutate(Signatures = future_map2(.x = CrossCut, .y = Grooves, .f = cc_get_signature, span = 0.75, span2 = .0
3))

 Signatures_Phoenix <- Phoenix %>%
  select(id, Barrel, Bullet, Land, Signatures) %>%
   tidyr::unnest()
```

```
Phoenix_Comparisons <- crossing(Bullet1 = Phoenix$id, Bullet2 = Phoenix$id) %>%
  left_join(nest(Phoenix, -id) %>% magrittr::set_names(c("Bullet1", "Bullet1_data"))) %>%
  left_join(nest(Phoenix, -id) %>% magrittr::set_names(c("Bullet2", "Bullet2_data")))
```

```
get_sig <- function(data) {
  map(data$Signatures, "sig")
}
Phoenix_Comparisons <- Phoenix_Comparisons %>%
  mutate(sig1 = map(Bullet1_data, get_sig), sig2 = map(Bullet2_data, get_sig))

Phoenix_Comparisons <- Phoenix_Comparisons %>%
  select(-Bullet1_data, -Bullet2_data)

Phoenix_Comparisons <- Phoenix_Comparisons %>%
  mutate(Aligned = map2(sig1, sig2, ~sig_align(unlist(.x), unlist(.y))))
```

```
Phoenix_Comparisons <- Phoenix_Comparisons %>%
 mutate(Striae = map(Aligned, sig_cms_max, span = 15))

Phoenix_Comparisons <- Phoenix_Comparisons %>%
  mutate(features = map2(.x = Aligned, .y = Striae, .f = extract_features_all, resolution = resolution_p))

Phoenix_Comparisons <- Phoenix_Comparisons_15 %>%
  tidyr::unnest(features)
```

Span of 15:

Min. 1st Qu. Median Mean 3rd Qu. Max.

0.000 2.000 3.000 3.141 4.000 73.000 [1*]

0.000 2.000 3.000 2.872 4.000 34.000 [2*]

Span of 25:

Min. 1st Qu. Median Mean 3rd Qu. Max.

0.000 2.000 3.000 3.062 3.000 62.000 [1]

0.000 2.000 2.000 2.827 3.000 37.000 [2]

Span of 35:

Min. 1st Qu. Median Mean 3rd Qu. Max.

0.000 2.000 2.000 2.974 3.000 56.000 [1]

0.000 2.000 2.000 2.776 3.000 32.000 [2]

Span of 45:

Min. 1st Qu. Median Mean 3rd Qu. Max.

0.000 2.000 2.000 2.915 3.000 47.000 [1]

0.000 2.000 2.000 2.745 3.000 37.000 [2]

Span of 55:

Min. 1st Qu. Median Mean 3rd Qu. Max.

0.000 1.000 2.000 2.832 3.000 41.000 [1]

0.000 1.000 2.000 2.685 3.000 34.000 [2]

Span of 65:

Min. 1st Qu. Median Mean 3rd Qu. Max.

0.000 1.000 2.000 2.698 3.000 37.000 [1]

0.000 1.000 2.000 2.569 3.000 34.000 [2]

Span of 75:

Min. 1st Qu. Median Mean 3rd Qu. Max.

0.000 1.000 2.000 2.557 3.000 34.000 [1]

0.000 1.000 2.000 2.442 3.000 25.000 [2]

Span of 85:

Min. 1st Qu. Median Mean 3rd Qu. Max.

0.000 1.000 2.000 2.451 3.000 30.000 [1]

0.000 1.000 2.000 2.349 3.000 23.000 [2]
