Supplementary Materials for Shaw et al. (2020)

Phonological contrast and phonetic variation: the case of velars in Iwaidja

R Markdown authors: Christopher Carignan & Jason Shaw

27/03/2020

Description

This is an R Markdown document for reproducing acoustic and articulatory metrics and figures appearing in the *Language* article entitled, “Phonological contrast and phonetic variation: the case of velars in Iwaidja” (Shaw, J., Carignan, C., Agostini, T., Mailhammer, R., Harvey, M., & Derrick, D., 2020).

This document is created to help you follow and understand the analyses appearing in the article without necessarily needing to reproduce them on your own... while also allowing you the ability to do so if you want!

Orienting yourself with the data directories

Due to the many different data types appearing in this paper, there is not really an easy way to package all of the data into a single table. Because of this, we have decided to store the data in different sub-directories. In order to recreate the analyses, you need to have each of the following data directories (obtained from *Language*) in your computer:

- **acoustics**
- **aggregate_data_files**
- **articulation**
- **variables**

Once you have those four sub-directories saved in a single main directory, set your R workspace to the path of that main directory:

```
setwd("/home/chris/Documents/research/Iwaidja/Supplementary_materials/")
```

You can list the files in the directory to ensure that you’ve set the correct path. Again, you should have directories named **acoustics**, **aggregate_data_files**, **articulation**, and **variables**, obtained from the article supplementary materials:

```
list.files(getwd())
```

```
## [1] "acoustics"          "aggregate_data_files" "articulation"
## [4] "variables"
```

Do you have those four directories and all of their respective data files? If so, you’re good to go!

Analyses

Required packages

The first thing we need to do is load the R packages that we will need for the analyses. If you don’t have these packages installed on your computer, you can do so using the `install.packages()` function, e.g., `install.packages('ggplot2')` for the *ggplot2* package. Once installed, you can load them using either the `library()` or `require()` function:

```
require(ggplot2); require(grid); require(gridExtra); require(MASS)
require(mclust); require(plyr); require(lemon)
```

We’re going to go through the figures slightly out of order from how they appear in the paper, to get the computationally intensive stuff out of the way first and to create the data metrics that we will be using for the rest of the figures.

Figure 7

Let's make a list of the speaker names and initialize some variables that we'll use in the loop that follows. This will help us iterate through some of the data objects more easily.

```
# speaker list
speakers <- c("CM150415", "IL151116", "ILM150417", "JC151112")

# initialize some variables
int.data.kh <- c(); int.data.jy <- c()
int.plots.kh <- c(); int.plots.jy <- c()
kh.iter <- 1; jy.iter <- 1; sp <- 1
```

And now we come to the main loop for Figure 7, where we load the by-speaker acoustic intensity data and create a figure for each speaker. These figures will be added to data lists for later plotting. Each of the important chunks of code has been annotated with descriptions of the code functionality.

```
for (speaker in speakers) {
  # read the acoustic intensity data (measured in Praat)
  kh <- read.csv(paste0("acoustics/", speaker, "_kh_intensity.txt"), head=T, sep=",")
  kh <- kh[kh$vowel=="a",]
  jy <- read.csv(paste0("acoustics/", speaker, "_jy_intensity.txt"), head=T, sep=",")

  # /k-h/ data
  for (token in unique(kh$token)) {
    # get the token data
    subdata <- kh[kh$token==token,]

    # time normalization
    kh$time.norm[kh$token==token] <-
      (subdata$time - min(subdata$time)) / (max(subdata$time) - min(subdata$time))

    # fit a second-degree polynomial to the intensity data
    fit <- lm(subdata$amp ~ poly(subdata$time, 2))

    # get the quadratic coefficient of the polynomial model
    int.data.kh$coeff[[kh.iter]] <- fit$coefficients[3]

    # log some variables
    int.data.kh$speaker[[kh.iter]] <- as.character(speaker)
    int.data.kh$word[[kh.iter]] <- as.character(subdata[1, "word"])
    int.data.kh$cons[[kh.iter]] <- as.character(subdata[1, "cons"])

    # standard deviation of intensity
    int.data.kh$sd[[kh.iter]] <- sd(subdata$amp)

    # iterate for next /k-h/ token
    kh.iter <- kh.iter + 1
  }
  kh$cons <- relevel(kh$cons, "k")

  # /j-y/ data
  for (token in unique(jy$token)) {
    # get the token data
    subdata <- jy[jy$token==token,]

    # time normalization
    jy$time.norm[jy$token==token] <-
      (subdata$time - min(subdata$time)) / (max(subdata$time) - min(subdata$time))

    # fit a second-degree polynomial to the intensity data
    fit <- lm(subdata$amp ~ poly(subdata$time, 2))
```

```

# get the quadratic coefficient of the polynomial model
int.data.jy$coeff[[jy.iter]] <- fit$coefficients[3]

# log some variables
int.data.jy$speaker[[jy.iter]] <- as.character(speaker)
int.data.jy$word[[jy.iter]] <- as.character(subdata[1,"word"])
int.data.jy$cons[[jy.iter]] <- as.character(subdata[1,"cons"])

# standard deviation of intensity
int.data.jy$sd[[jy.iter]] <- sd(subdata$amp)

# iterate for next /j-y/ token
jy.iter <- jy.iter + 1
}

# make the speaker plots and store them in the list
int.plots.kh[[sp]] <- ggplot(kh,aes(x=time.norm,y=amp,color=cons,linetype=cons,group=token)) +
  geom_line(alpha=0.4) + geom_smooth(aes(group=cons)) +
  theme_classic() + theme(text = element_text(size=18),
                           legend.title=element_text(size=18),
                           legend.text=element_text(size=18)) +
  ylab("SPL (dB)") + xlab("Triphone duration (normalized)") +
  ggtitle("Acoustic intensity: /aka/-/aha/")

int.plots.jy[[sp]] <- ggplot(jy,aes(x=time.norm,y=amp,color=cons,linetype=cons,group=token)) +
  geom_line(alpha=0.4) + geom_smooth(aes(group=cons)) +
  theme_classic() + theme(text = element_text(size=18),
                           legend.title=element_text(size=18),
                           legend.text=element_text(size=18)) +
  ylab("SPL (dB)") + xlab("Triphone duration (normalized)") +
  ggtitle("Acoustic intensity: /aja/-/aya/")

# iterate the plot lists for next speaker
sp <- sp + 1
}

```

Now that all of the by-speaker ggplot objects have been created and stored in their respective lists, we can combine them into single figures.

Let's first make the figure for the /k-h/ data:

```

lemon::grid_arrange_shared_legend(int.plots.kh[[1]], int.plots.kh[[2]],
                                  int.plots.kh[[3]], int.plots.kh[[4]],
                                  nrow=1, ncol=4, position="right")

```

And now the one for the /j-y/ data:

```

lemon::grid_arrange_shared_legend(int.plots.jy[[1]], int.plots.jy[[2]],
                                  int.plots.jy[[3]], int.plots.jy[[4]],
                                  nrow=1, ncol=4, position="right")

```

Figure 11

This figure involves a method by which we use a linear discriminant analysis (LDA) model, trained on principal components analysis (PCA) scores from the ultrasound images, to predict “vowel dissimilarity” scores. This is probably the most quantitatively involved and complex part of the analyses, and the code is therefore pretty long. However, the relevant bits of code are annotated with descriptions of the code functionality.

```
# initialize some variables
kh <- c(); jy <- c()
art.data.kh <- c(); art.data.jy <- c()
art.plots.kh <- c(); art.plots.jy <- c()
kh.iter <- 1; jy.iter <- 1; sp <- 1

## Main loop:
# do the analyses by speaker
for (speaker in speakers) {
  # one of the speakers had a different ultrasound sampling rate than the others
  if (speaker == "ILM150417") {
    sr <- 22
  } else {
    sr <- 21
  }

  # get the PC scores for the speaker
  kh$scores <- read.csv(paste0("articulation/", speaker, "_kh_pc_scores.txt"), head=F, sep="\t")
  jy$scores <- read.csv(paste0("articulation/", speaker, "_jy_pc_scores.txt"), head=F, sep="\t")

  # number of PCs retained
  kh$PCs <- ncol(kh$scores)
  jy$PCs <- ncol(jy$scores)

  # get the PC coefficients/loadings for the speaker
  kh$coeffs <- read.csv(paste0("articulation/", speaker, "_kh_pc_heatmaps.txt"), head=F, sep="\t")
  jy$coeffs <- read.csv(paste0("articulation/", speaker, "_jy_pc_heatmaps.txt"), head=F, sep="\t")

  # get the ultrasound frames used in the PCA model
  kh$filenums <- read.csv(paste0("variables/", speaker, "_kh_frames.txt"), head=F, sep="\t")
  jy$filenums <- read.csv(paste0("variables/", speaker, "_jy_frames.txt"), head=F, sep="\t")

  # get some variables used for the analysis
  kh$vars <- read.csv(paste0("variables/", speaker, "_kh_vars.txt"), head=T, sep=",")
  jy$vars <- read.csv(paste0("variables/", speaker, "_jy_vars.txt"), head=T, sep=",")

  kh$vars$word <- as.character(kh$vars$word)
  jy$vars$word <- as.character(jy$vars$word)

  # get the /k-h/ words
  kh$words <- c()
  for (obs in 1:nrow(kh$vars)) {
```

```

# remove word from repetition number
string <- strsplit(kh$vars$word[obs], "_")
# log the word
kh$words[obs] <- unlist(string)[1]
}
kh$vars$word <- as.factor(kh$words)

# get the /j-y/ words
jy$words <- c()
for (obs in 1:nrow(jy$vars)) {
  # remove word from repetition number
  string <- strsplit(jy$vars$word[obs], "_")
  # log the word
  jy$words[obs] <- unlist(string)[1]
}
jy$vars$word <- as.factor(jy$words)

# preparation of data for LDA training
kh$scores$include <- 0; jy$scores$include <- 0

# get the PC scores for the /k-h/ data
for (obs in 1:nrow(kh$vars)) {
  # get the ultrasound frame numbers corresponding to the start and end of the word
  start <- kh$vars$Wstart.fr[obs]
  end <- kh$vars$Wend.fr[obs]

  # get the numbers of the ultrasound frames included in the PCA model
  kh$vars$PCstart[obs] <- sum(kh$filenums[1:start,1])
  kh$vars$PCend[obs] <- sum(kh$filenums[1:end,1])

  kh$vars$v1PC[obs] <- sum(kh$filenums[1:kh$vars$v1[obs],1])
  kh$vars$v2PC[obs] <- sum(kh$filenums[1:kh$vars$v2[obs],1])

  # only include data for /a/
  if (kh$vars$vowel[obs]=="a") {
    kh$scores$include[kh$vars$v1PC[obs]] <- 1
    kh$scores$include[kh$vars$v2PC[obs]] <- 1
  }
}

# get the PC scores for the /j-y/ data
for (obs in 1:nrow(jy$vars)) {
  # get the ultrasound frame numbers corresponding to the start and end of the word
  start <- jy$vars$Wstart.fr[obs]
  end <- jy$vars$Wend.fr[obs]

  # get the numbers of the ultrasound frames included in the PCA model
  jy$vars$PCstart[obs] <- sum(jy$filenums[1:start,1])
  jy$vars$PCend[obs] <- sum(jy$filenums[1:end,1])

  jy$vars$v1PC[obs] <- sum(jy$filenums[1:jy$vars$v1[obs],1])
  jy$vars$v2PC[obs] <- sum(jy$filenums[1:jy$vars$v2[obs],1])

  # only include data for /a/
  if (jy$vars$vowel[obs]=="a") {
    jy$scores$include[jy$vars$v1PC[obs]] <- 1
    jy$scores$include[jy$vars$v2PC[obs]] <- 1
  }
}

```

```

# create articulatory signal using LDA (/k-h/)
kh$llda      <- MASS::lda(kh$scores[,1:kh$PCs], kh$scores$include==1) # train on flanking vowels
kh$llda.pred <- predict(kh$llda, kh$scores[,1:kh$PCs]) # predict scores for ALL frames
kh$art.sig   <- kh$llda.pred$x # extract the scores

# create articulatory signal using LDA (/j-y/)
jy$llda      <- MASS::lda(jy$scores[,1:jy$PCs], jy$scores$include==1) # train on flanking vowels
jy$llda.pred <- predict(jy$llda, jy$scores[,1:jy$PCs]) # predict scores for ALL frames
jy$art.sig   <- jy$llda.pred$x # extract the scores

# data smoothing, /k-h/
kh$smooths <- c()
obs <- 1
for (i in 1:nrow(kh$vars)) {
  # get some variables to use
  start <- kh$vars$v1PC[i]
  end   <- kh$vars$v2PC[i]
  v <- kh$vars$vowel[i]
  c <- kh$vars$cons[i]
  w <- kh$vars$word[i]

  if (v == "a") {
    # get the chunk of the LDA signal corresponding to the triphone sequence
    chunk <- kh$art.sig[start:end]

    # smooth the signal chunk, if necessary
    if (length(chunk)>3) {
      smooth <- smooth.spline(chunk, spar=0.35)
    } else {
      smooth <- as.data.frame(chunk)
    }

    # data normalization
    for (j in 1:length(chunk)) {
      kh$smooths$X[obs] <- j
      kh$smooths$Xnorm[obs] <- (j - 1)/(length(chunk)-1)
      kh$smooths$Y[obs] <- chunk[j]
      if (length(chunk)>3) {
        kh$smooths$Ysmooth[obs] <- smooth$y[j]
      } else {
        kh$smooths$Ysmooth[obs] <- smooth[j,]
      }
      kh$smooths$token[obs] <- i
      kh$smooths$cons[obs] <- as.character(c)
      kh$smooths$word[obs] <- as.character(w)

      obs <- obs + 1
    }
  }
}
kh$smooths <- as.data.frame(kh$smooths)
kh$smooths$cons <- relevel(kh$smooths$cons, "k")

# make the speaker plot and store it in the list
art.plots.kh[[sp]] <- ggplot(kh$smooths, aes(x=Xnorm, y=Ysmooth, color=cons, group=token)) +
  geom_line(alpha=0.4) + geom_smooth(aes(group=cons)) +
  theme_classic() + theme(text=element_text(size=18),
                           legend.title=element_text(size=18),
                           legend.text=element_text(size=18)) +

```

```

ylab("LDA score") + xlab("Triphone duration (normalized)") +
ggtitle("Ultrasound LDA: /aka/-/aha/")

# data smoothing, /j-y/
jy$smooths <- c()
obs <- 1
for (i in 1:nrow(jy$vars)) {
  # get some variables to use
  start <- jy$vars$v1PC[i]
  end <- jy$vars$v2PC[i]
  v <- jy$vars$vowel[i]
  c <- jy$vars$cons[i]
  w <- jy$vars$word[i]

  if (v == "a"){
    # get the chunk of the LDA signal corresponding to the triphone sequence
    chunk <- jy$art.sig[start:end]

    # smooth the signal chunk, if necessary
    if (length(chunk)>3) {
      smooth <- smooth.spline(chunk, spar=0.35)
    } else {
      smooth <- as.data.frame(chunk)
    }

    # data normalization
    for (j in 1:length(chunk)) {
      jy$smooths$X[obs] <- j
      jy$smooths$Xnorm[obs] <- (j - 1)/(length(chunk)-1)
      jy$smooths$Y[obs] <- chunk[j]
      if (length(chunk)>3) {
        jy$smooths$Ysmooth[obs] <- smooth$y[j]
      } else {
        jy$smooths$Ysmooth[obs] <- smooth[j,]
      }
      jy$smooths$token[obs] <- i
      jy$smooths$cons[obs] <- as.character(c)
      jy$smooths$word[obs] <- as.character(w)

      obs <- obs + 1
    }
  }
}
jy$smooths <- as.data.frame(jy$smooths)

# make the speaker plot and store it in the list
art.plots.jy[[sp]] <- ggplot(jy$smooths, aes(x=Xnorm,y=Ysmooth,color=cons,group=token)) +
  geom_line(alpha=0.4) + geom_smooth(aes(group=cons)) +
  theme_classic() + theme(text=element_text(size=18),
                           legend.title=element_text(size=18),
                           legend.text=element_text(size=18)) +
  ylab("LDA score") + xlab("Triphone duration (normalized)") +
  ggtitle("Ultrasound LDA: /aja/-/aya/")

# iterate the plot lists for the next speaker
sp <- sp + 1

```



```

## Make data tables with measurements
# /k-h/ data table
for (token in unique(kh$smooths$token)) {
  subdata <- kh$smooths[kh$smooths$token==token,]

  if (nrow(subdata)>2) {
    # second degree polynomial of smoothed triphone LDA chunk
    fit <- lm(subdata$Ysmooth ~ poly(subdata$Xnorm,2))
  }

  # add some variables
  art.data.kh$speaker[[kh.iter]] <- as.character(speaker)
  art.data.kh$word[[kh.iter]] <- as.character(subdata[1,"word"])
  art.data.kh$cons[[kh.iter]] <- as.character(subdata[1,"cons"])

  # get the quadratic coefficient
  if (nrow(subdata)>2) {
    art.data.kh$coeff[[kh.iter]] <- fit$coefficients[3]
  } else {
    art.data.kh$coeff[[kh.iter]] <- 0
  }

  # standard deviation of smoothed triphone LDA chunk
  art.data.kh$sd[[kh.iter]] <- sd(subdata$Ysmooth)

  # iterate for next /k-h/ token
  kh.iter <- kh.iter + 1
}

# /j-y/ data table
for (token in unique(jy$smooths$token)) {
  subdata <- jy$smooths[jy$smooths$token==token,]

  if (nrow(subdata)>2) {
    # second degree polynomial of smoothed triphone LDA chunk
    fit <- lm(subdata$Ysmooth ~ poly(subdata$Xnorm,2))
  }

  # add some variables
  art.data.jy$speaker[[jy.iter]] <- as.character(speaker)
  art.data.jy$word[[jy.iter]] <- as.character(subdata[1,"word"])
  art.data.jy$cons[[jy.iter]] <- as.character(subdata[1,"cons"])

  # get the quadratic coefficient
  if (nrow(subdata)>2) {
    art.data.jy$coeff[[jy.iter]] <- fit$coefficients[3]
  } else {
    art.data.jy$coeff[[jy.iter]] <- 0
  }

  # standard deviation of smoothed triphone LDA chunk
  art.data.jy$sd[[jy.iter]] <- sd(subdata$Ysmooth)

  # iterate for next /j-y/ token
  jy.iter <- jy.iter + 1
}
}

```

Now that all of the by-speaker ggplot objects have been created and stored in their respective lists, we can combine them into single figures. Let's first make the figure for the /k-h/ data:


```
lemon::grid_arrange_shared_legend(art.plots.kh[[1]], art.plots.kh[[2]],
                                art.plots.kh[[3]], art.plots.kh[[4]],
                                nrow=1, ncol=4, position="right")
```

And now the one for the /j-y/ data:

```
lemon::grid_arrange_shared_legend(art.plots.jy[[1]], art.plots.jy[[2]],
                                art.plots.jy[[3]], art.plots.jy[[4]],
                                nrow=1, ncol=4, position="right")
```

Figure 8

For many of the rest of the figures, it's helpful to have the data we just created for Figures 7 and 11 aggregated together in a single table. We've done that separately, added some additional measurements, and cleaned up the data a bit. So let's now load that aggregated and "wrangled" data:

```
agg.data <- read.csv("aggregate_data_files/int_art_coefficients.csv", head=T, sep=",")
```

We also need to define a function that we'll use for calculating standard error for some of the plots¹:

```
summarySE <- function(data=NULL, measurevar, groupvars=NULL, na.rm=FALSE,
                      conf.interval=.95, .drop=TRUE) {
  # New version of length which can handle NA's: if na.rm==T, don't count them
  length2 <- function(x, na.rm=FALSE) {
    if (na.rm) sum(!is.na(x))
    else      length(x)
  }

  # This does the summary. For each group's data frame, return a vector with N, mean, and sd
  datac <- ddpby(data, groupvars, .drop=.drop,
                .fun = function(xx, col) {
                  c(N    = length2(xx[[col]], na.rm=na.rm),
                    mean  = mean  (xx[[col]], na.rm=na.rm),
                    sd    = sd    (xx[[col]], na.rm=na.rm)
                  )
                },
                measurevar
  )
}
```

¹This function comes from http://www.cookbook-r.com/Manipulating_data/Summarizing_data/

```

# Rename the "mean" column
datac <- rename(datac, c("mean"=measurevar))

# Calculate standard error of the mean
datac$se <- datac$sd / sqrt(datac$N)

# Calculate t-statistic for confidence interval:
# e.g., if conf.interval is .95, use .975 (above/below), and use df=N-1
ciMult <- qt(conf.interval/2 + .5, datac$N-1)
datac$ci <- datac$se * ciMult

return(datac)
}

```

First we'll plot the palatals (Figure 8, left):

```

# get the palatal data
palatals <- subset(agg.data, place=="palatal")

# use summarySE function to make error bars
palatal_stats_int <- summarySE(palatals, measurevar="int_coeff", groupvars=c("speaker", "cons"),
                               na.rm=TRUE, conf.interval=.95, .drop=TRUE)

# make the plot
ggplot(palatal_stats_int, aes(x=speaker, y=int_coeff, fill=cons)) +
  geom_bar(position="dodge", stat="identity") +
  scale_fill_manual("cons", values=c("j"="red", "y"="blue")) +
  theme(text=element_text(size=20)) +
  geom_errorbar(aes(ymin=int_coeff-ci, ymax=int_coeff+ci), size=.3, width=.2,
                position=position_dodge(.9)) +
  coord_cartesian(ylim=c(-5,33)) + labs(y="Intensity coefficient") + ggtitle("Palatals")

```

And now we'll plot the velars (Figure 8, right)

```

# get the velar data
velars <- subset(agg.data, place=="velar")

# use summarySE function to make error bars
velar_stats_int <- summarySE(velars, measurevar="int_coeff", groupvars=c("speaker", "cons"),
                              na.rm=TRUE, conf.interval=.95, .drop=TRUE)

# reorder the levels to have the same colors for the velar plot as for the palatal plot
velar_stats_int$cons = factor(velar_stats_int$cons, levels(velar_stats_int$cons)[c(3,1,2,4)])

```

```
# make the plot
ggplot(velar_stats_int, aes(x=speaker, y=int_coeff, fill=cons)) +
  geom_bar(position="dodge", stat="identity") +
  scale_fill_manual("cons", values=c("k"="red", "h"="blue")) +
  theme(text = element_text(size=20)) +
  geom_errorbar(aes(ymin=int_coeff-ci, ymax=int_coeff+ci), size=.3, width=.2,
    position=position_dodge(.9)) +
  coord_cartesian(ylim=c(-5,33)) + labs(y="Intensity coefficient") + ggtitle("Velars")
```

Figure 9

Okay, now we're going to start doing some unsupervised, bottom-up clustering of the data, starting with only the quadratic coefficient of the acoustic intensity polynomial fits:

```
# pick out the 5th columns of data, the intensity coefficient
int_coeff_p <- palatals[,5]
int_coeff_v <- velars[,5]

# cluster the data using the Bayesian information criterion (BIC)
BICp <- mclust::mclustBIC(int_coeff_p)
BICv <- mclust::mclustBIC(int_coeff_v)

# create the cluster objects
mod_p <- mclust::Mclust(int_coeff_p, x=BICp)
mod_v <- mclust::Mclust(int_coeff_v, x=BICv)
```

Now that we have done the unsupervised clustering, we can make the plots for Figure 9. First, the left plot (palatals):

```
plot(mod_p, what="classification", xlab="Intensity dip")
title(main="Palatals")
```

As indicated by the figure, the data are best fit by two categories. We can also summarize the model to verify:

```
summary(mod_p, parameters=F)
```

```
## -----
## Gaussian finite mixture model fitted by EM algorithm
## -----
##
## Mclust V (univariate, unequal variance) model with 2 components:
##
## log-likelihood  n df      BIC      ICL
##      -822.3303 217  5 -1671.56 -1730.603
##
## Clustering table:
##   1  2
## 105 112
```

Now the right plot (velars):

```
plot(mod_v, what="classification", xlab="Intensity dip")
title(main="Velars")
```

Unlike for the palatals, the velar data is best fit by three categories, which we can also see in the model summary:

```
summary(mod_v, parameters=F)
```

```
## -----
## Gaussian finite mixture model fitted by EM algorithm
## -----
```

```
##
## Mclust V (univariate, unequal variance) model with 3 components:
##
## log-likelihood    n df          BIC          ICL
##      -564.5925 170   8 -1170.271 -1252.202
##
## Clustering table:
##    1    2    3
##   37 100   33
```

Figure 10

This figure shows the match between dictionary labels and the unsupervised clustering results for the acoustic data only. We first need to prepare some data for plotting:

```
# add the classifications to the data files
palatals <- cbind(mod_p$classification, palatals)
velars    <- cbind(mod_v$classification, velars)

# change column name to "intensity_prediction"
colnames(palatals)[1] <- "intensity_prediction"
colnames(velars)[1]   <- "intensity_prediction"

## for palatals
# code all 1 responses as glides "y"
palatals$intensity_prediction[palatals$intensity_prediction==1] <- "y"
# code all 2 responses as "j"
palatals$intensity_prediction[palatals$intensity_prediction==2] <- "j"

## for velars, code 1 as "a", 2 as "h", 3 as "k"
velars$intensity_prediction[velars$intensity_prediction=="1"] <- "a"
velars$intensity_prediction[velars$intensity_prediction=="2"] <- "h"
velars$intensity_prediction[velars$intensity_prediction=="3"] <- "k"

# code for dictionary match for palatals
palatals$match_int <- 1
for (x in 1:length(palatals$speaker)) {
  if (palatals$intensity_prediction[x] != palatals$cons[x]) {palatals$match_int[x] = 0}
}

# code dictionary match for velars
velars$match_int <- 1
for (x in 1:length(velars$speaker)) {
  if (velars$intensity_prediction[x] != velars$cons[x]) {velars$match_int[x] = 0}
}

# reorder the levels to have the same colors for the velar plot as for the palatal plot
velars$cons <- factor(velars$cons, levels(velars$cons)[c(3,1,2,4)])
```

Now that the plotting data has been prepared, we can go ahead and make Figure 10. First the left plot (palatals):

```
ggplot(palatals, aes(x=speaker, y=match_int, fill=cons)) +
  geom_bar(position="dodge", stat="summary", fun.y="mean") +
  scale_fill_manual("cons", values=c("j"="red", "y"="blue")) +
  theme(text=element_text(size=20)) +
  labs(y="Match to dictionary") +
  ylim(0,1) +
  ggtitle("Palatals")
```

Now the right plot (velars):

```
ggplot(velars, aes(x=speaker, y=match_int, fill=cons)) +
  geom_bar(position="dodge", stat="summary", fun.y="mean") +
  scale_fill_manual("cons", values=c("k"="red", "h"="blue")) +
  theme(text=element_text(size=20)) +
  labs(y="Match to dictionary") +
  ylim(0,1) +
  ggtitle("Velars")
```

Figure 12

This figure shows the average quadratic term of polynomials fit to the change in LDA score over time. Larger numbers indicate greater vowel dissimilarity. First, left plot (palatals):

```
palatal_stats_art <- summarySE(palatals, measurevar="art_coeff", groupvars=c("speaker","cons"),
                               na.rm=TRUE, conf.interval=.95, .drop=TRUE)

ggplot(palatal_stats_art, aes(x=speaker, y=art_coeff, fill=cons)) +
  geom_bar(position="dodge", stat="identity") + coord_cartesian(ylim=c(-.25,3)) +
  scale_fill_manual("cons", values=c("j"="red", "y"="blue")) + theme(text=element_text(size=20)) +
  geom_errorbar(aes(ymin=art_coeff-ci, ymax=art_coeff+ci), size=.3, width=.2,
                position=position_dodge(.9)) +
  labs(y="Vowel dissimilarity") + ggtitle("Palatals")
```

Now the right plot (velars):

```

velar_stats_art <- summarySE(velars, measurevar="art_coeff", groupvars=c("speaker", "cons"),
                             na.rm=TRUE, conf.interval=.95, .drop=TRUE)

ggplot(velar_stats_art, aes(x=speaker, y=art_coeff, fill=cons)) +
  geom_bar(position="dodge", stat="identity") + coord_cartesian(ylim=c(-.25,3)) +
  scale_fill_manual("cons", values=c("k"="red", "h"="blue")) + theme(text=element_text(size=20)) +
  geom_errorbar(aes(ymin=art_coeff-ci, ymax=art_coeff+ci), size=.3, width=.2,
                position=position_dodge(.9)) +
  labs(y="Vowel dissimilarity") + ggtitle("Velars")

```

Figure 13

In this figure, we've done the same unsupervised, bottom-up clustering of the data that we did with the quadratic coefficient of the acoustic intensity polynomial fits (Figure 9), but this time we're using the quadratic coefficient of the articulatory LDA score fits (i.e., the vowel dissimilarity measure).

```

## pick out the 8th columns of data, the articulatory coefficient (vowel dissimilarity measure)
art_coeff_p <- palatals[,8]
art_coeff_v <- velars[,8]

# cluster the data using the Bayesian information criterion (BIC)
BICp <- mclust::mclustBIC(art_coeff_p)
BICv <- mclust::mclustBIC(art_coeff_v)

```



```
# create cluster objects
mod_p <- mclust::Mclust(art_coeff_p, x=BICp)
mod_v <- mclust::Mclust(art_coeff_v, x=BICv)
```

Now that we have done the unsupervised clustering, we can make the plots for Figure 13. First, the left plot (palatals):

```
plot(mod_p, what="classification", xlab="Vowel dissimilarity")
title(main="Palatals")
```

Unlike for the acoustic intensity variable, the articulatory vowel dissimilarity metric is best fit by a single category for the palatals:

```
summary(mod_p, parameters=F)

## -----
## Gaussian finite mixture model fitted by EM algorithm
## -----
##
## Mclust X (univariate normal) model with 1 component:
##
##   log-likelihood   n df      BIC      ICL
##      -312.6237 217  2 -636.0071 -636.0071
##
## Clustering table:
##    1
## 217
```

And now the right plot (velars):

```
plot(mod_v, what="classification", xlab="Vowel dissimilarity")
title(main="Velars")
```

The velar data, on the other hand, is best fit by two categories. We can also verify this in the model summary:

```
summary(mod_v, parameters=F)
```

```
## -----
## Gaussian finite mixture model fitted by EM algorithm
## -----
##
## Mclust E (univariate, equal variance) model with 2 components:
##
## log-likelihood    n df          BIC          ICL
##      -190.7268 170  4 -401.9968 -435.6425
##
## Clustering table:
##    1    2
## 139   31
```

Figure 14

This figure shows the match between dictionary labels and the unsupervised clustering results for the articulatory data only. We first need to prepare some data for plotting:

```
# add the classifications to the velar data frame
velars <- cbind(mod_v$classification, velars)

# change column name to "articulatory_prediction"
colnames(velars)[1] <- "articulatory_prediction"

# interpret categories
## for velars, code 1 as "h", 2 as "k"
velars$articulatory_prediction[velars$articulatory_prediction=="1"] <- "h"
velars$articulatory_prediction[velars$articulatory_prediction=="2"] <- "k"

# code for dictionary match for velars
velars$match_art <- 1
for (x in 1:length(velars$speaker)) {
  if (velars$articulatory_prediction[x] != velars$cons[x]) {
    velars$match_art[x] <- 0
  }
}
```

Now that the plotting data has been prepared, we can go ahead and make Figure 14:

```
ggplot(velars, aes(x=speaker, y=match_art, fill=cons)) +
  geom_bar(position="dodge", stat="summary", fun.y="mean") +
```

```
scale_fill_manual("cons", values=c("k"="red", "h"="blue")) +
labs(y="Match to dictionary") +
ylim(0,1) +
ggtitle("Dictionary match based on classification by vowel dissimilarity")
```

Figure 15

This figure shows the relation between the two metrics that we have now created: **intensity dip** (acoustic intensity quadratic coefficient) and **vowel dissimilarity** (articulatory LDA score quadratic coefficient):

```
ggplot(subset(agg.data, place=="velar"), aes(x=int_coeff, y=art_coeff)) +
  theme(text=element_text(size=14)) +
  xlab("intensity dip (coeff)") +
  ylab("vowel dissimilarity (coeff)") +
  geom_point() +
  geom_smooth(method=lm)
```

Figure 16

Now that we've done the unsupervised clustering for both the acoustic and articulatory metrics separately, it's time to do a final clustering based on the combination of the two dimensions.

```
# combine the acoustic and articulatory metrics
v <- cbind(int_coeff_v, art_coeff_v)
```

```

# cluster the data using the Bayesian information criterion (BIC)
BICv <- mclust::mclustBIC(v)

# create the cluster object
velar_art_int <- mclust::Mclust(v, x=BICv)

```

We can see that the combined acoustic/articulatory data for the velars is best fit by two categories:

```

summary(velar_art_int, parameters=F)

## -----
## Gaussian finite mixture model fitted by EM algorithm
## -----
##
## Mclust VEV (ellipsoidal, equal shape) model with 2 components:
##
##   log-likelihood    n df         BIC          ICL
##      -731.2454 170 10 -1513.849 -1546.088
##
## Clustering table:
##   1  2
## 79 91

```

We now add the classification results to the data frame and create Figure 16:

```

# add the classifications to the velar data frame
velars <- cbind(velar_art_int$classification, velars)

# change column name to "combined_prediction"
colnames(velars)[1] <- "combined_prediction"

# interpret categories
## code 1 as "h", 2 as "k"
velars$combined_prediction[velars$combined_prediction=="1"] <- "h"
velars$combined_prediction[velars$combined_prediction=="2"] <- "k"

# code for dictionary match for velars
velars$match_combined <- 1
for (x in 1:length(velars$speaker)) {
  if (velars$combined_prediction[x] != velars$cons[x]) {
    velars$match_combined[x] <- 0
  }
}

# Figure 16
plot(velar_art_int, what="classification",
     xlab="intensity dip (coeff)", ylab="vowel dissimilarity (coeff)")
title(main="Bottom-up classification")

```

Bottom-up classification

Figures 17–20

At this point, most of the hard work is done! Figures 17–22 are mainly ways to view different aspects of the results, so we've put them all together into one section in this Markdown file.

Figure 17: distribution of dictionary labels in phonetic space by speaker

```
# predicted labels by dictionary label
velars$cons <- factor(velars$cons, levels(velars$cons)[c(3,1,2,4)])

ggplot(velars,aes(x=int_coeff, y=art_coeff, colour=cons)) +
  ggtitle("dictionary labels") +
  xlab("intensity dip (coeff)") +
  ylab("vowel dissimilarity (coeff)") +
  geom_point() +
  geom_density2d() +
  facet_grid(~speaker) +
  scale_color_manual(values=c("blue","red"))
```

Figure 18: distribution of predicted labels in phonetic space by speaker

```
# predicted labels by speaker
velars$combined_prediction <- factor(velars$combined_prediction,
                                     levels=c("k","h"), labels=c("less vocalic","more vocalic"))

ggplot(velars, aes(x=int_coeff, y=art_coeff, colour=combined_prediction)) +
  ggtitle("predicted labels") +
  xlab("intensity dip (coeff)") +
  ylab("vowel dissimilarity (coeff)") +
  geom_point() +
  geom_density2d() +
  facet_grid(~speaker) +
  scale_color_manual(values=c("red","blue")) +
  theme(legend.title=element_blank())
```

Figure 19: distribution of bottom-up classification categories by word

```
# classification categories by word
# order words: <k> then <h> within each order from less vocalic to more vocalic
velars$word <- factor(velars$word, levels(velars$word)[c(3,26,10,7,15,24,23,2,19,22,21,13,1,4,
                                                         5,6,8,9,11,12,14,16,17,18,20,25,27,28)])

ggplot(velars, aes(word)) +
  scale_fill_manual("combined_prediction",
                   values=c("more vocalic"="blue", "less vocalic"="red")) +
  geom_bar(aes(fill=combined_prediction)) +
  theme(legend.title=element_blank())
```

Figure 20: distribution of bottom-up classification categories by word and by speaker

```
# by speaker plot of classification category by lexical item
ggplot(velars, aes(word)) +
  scale_fill_manual("combined_prediction",
                    values=c("more vocalic"="blue", "less vocalic"="red")) +
  geom_bar(aes(fill=combined_prediction)) +
  facet_wrap(~speaker, ncol=1) +
  theme(legend.title=element_blank())
```

Figures 21–23

These final three figures all involve duration measurements, so we have likewise bundled them together in this section.

Figure 21: trigram duration by dictionary label

```
# convert duration to ms
velars$dur <- velars$dur*1000

# duration by dictionary label
velar_stats_dur_dict <- summarySE(velars, measurevar="dur", groupvars=c("speaker", "cons"),
                                   na.rm=TRUE, conf.interval=.95, .drop=TRUE)

# reorder the levels of the consonants to match other figures
velar_stats_dur_dict$cons <- factor(velar_stats_dur_dict$cons,
                                    levels(velar_stats_dur_dict$cons)[c(3,1,2,4)])

ggplot(velar_stats_dur_dict, aes(x=speaker, y=dur, fill=cons)) +
  geom_bar(position="dodge", stat="identity") +
  scale_fill_manual("cons", values=c("k"="red", "h"="blue")) +
  geom_errorbar(aes(ymin=dur-ci, ymax=dur+ci), size=.3, width=.2,
                position=position_dodge(.9)) +
  labs(y="trigram duration (ms)") +
  ggtitle("Duration across VCV by dictionary label")
```

Figure 22: trigram duration by predicted category

```

velar_stats_dur_predicted <- summarySE(velars, measurevar="dur",
                                       groupvars=c("speaker", "combined_prediction"),
                                       na.rm=TRUE, conf.interval=.95, .drop=TRUE)

ggplot(velar_stats_dur_predicted, aes(x=speaker, y=dur, fill=combined_prediction)) +
  geom_bar(position="dodge", stat="identity") +
  scale_fill_manual("combined_prediction",
                   values=c("less vocalic"="red", "more vocalic"="blue")) +
  geom_errorbar(aes(ymin=dur-ci, ymax=dur+ci), size=.3, width=.2,
               position=position_dodge(.9)) +
  labs(y="trigram duration (ms)") +
  ggtitle("Duration across VCV by predicted label") + theme(legend.title=element_blank())

```

Figure 23: duration comparisons between realization types

The final figure of the article compares the duration of singleton stressed [a], “more vocalic” velar trigram realizations, and “less vocalic” velar trigram realizations. We can see that, while the more vocalic trigrams are indeed shorter than the less vocalic trigrams, they are nonetheless greater than twice the duration of the singleton stressed [a]. This suggests that, although the velar may be lenited to a vocalic quality, the temporal duration is nonetheless partially preserved.

The duration measurements for this figure were made separately in Praat. We therefore load that data separately before creating Figure 23.

```
# load duration measures from vowels to compare to VCV trigrams
dur_data <- read.csv("aggregate_data_files/comparison_trigram_durations.csv", head=T, sep=",")

# make the box plot figure
boxplot(dur ~ trigram, data=dur_data, main="Interval duration",
        xlab="Interval", ylab="Duration (ms)", notch=T)
```