VIII. Appendix

A. Contents

B. Any Other Business

For more information about this research, including the original data and its licensing, please contact me directly at academia@citrusfruit.lu.

THE INTERNATIONAL PHONETIC ALPHABET (revised to 2005)

CONSONANTS (PULMONIC)

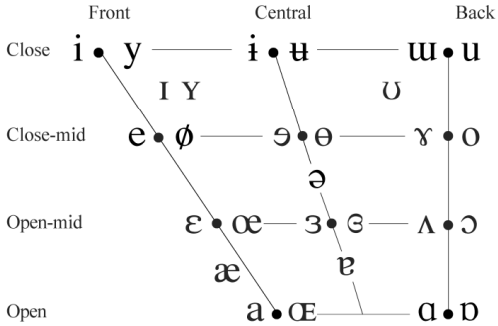
	Bilabial	Labiodental	Dental	Alveolar	Postalveolar	Retroflex	Palatal	Velar	Uvular	Pharyngeal	Glottal
Plosive	p b			t d		ʈ ɖ	c ɟ	k g	q ɢ		ʔ
Nasal	m	ɱ		n		ɳ	ɲ	ŋ	ɴ		
Trill	ʙ			r					ʀ		
Tap or Flap		ⱱ		ɾ		ɽ					
Fricative	ɸ β	f v	θ ð	s z	ʃ ʒ	ʂ ʐ	ç ʝ	x ɣ	χ ʁ	ħ ʕ	h ɦ
Lateral fricative				ɬ ɮ							
Approximant		ʋ		ɹ		ɻ	j	ɰ			
Lateral approximant				l		ɭ	ʎ	ʟ			

Where symbols appear in pairs, the one to the right represents a voiced consonant. Shaded areas denote articulations judged impossible.

CONSONANTS (NON-PULMONIC)

Clicks		Voiced implosives		Ejectives	
ʘ	Bilabial	ɓ	Bilabial	’	Examples:
|	Dental	ɗ	Dental/alveolar	pʼ	Bilabial
!	(Post)alveolar	ʄ	Palatal	tʼ	Dental/alveolar
ǂ	Palatoalveolar	ɠ	Velar	kʼ	Velar
‖	Alveolar lateral	ʛ	Uvular	sʼ	Alveolar fricative

VOWELS



Where symbols appear in pairs, the one to the right represents a rounded vowel.

OTHER SYMBOLS

ʍ	Voiceless labial-velar fricative	ɕ ʑ	Alveolo-palatal fricatives
w	Voiced labial-velar approximant	ɺ	Voiced alveolar lateral flap
ɥ	Voiced labial-palatal approximant	ʄ	Simultaneous ʃ and x
ʜ	Voiceless epiglottal fricative		
ʢ	Voiced epiglottal fricative		
ʡ	Epiglottal plosive		

Affricates and double articulations can be represented by two symbols joined by a tie bar if necessary. k͡p t͡s

SUPRASEGMENTALS

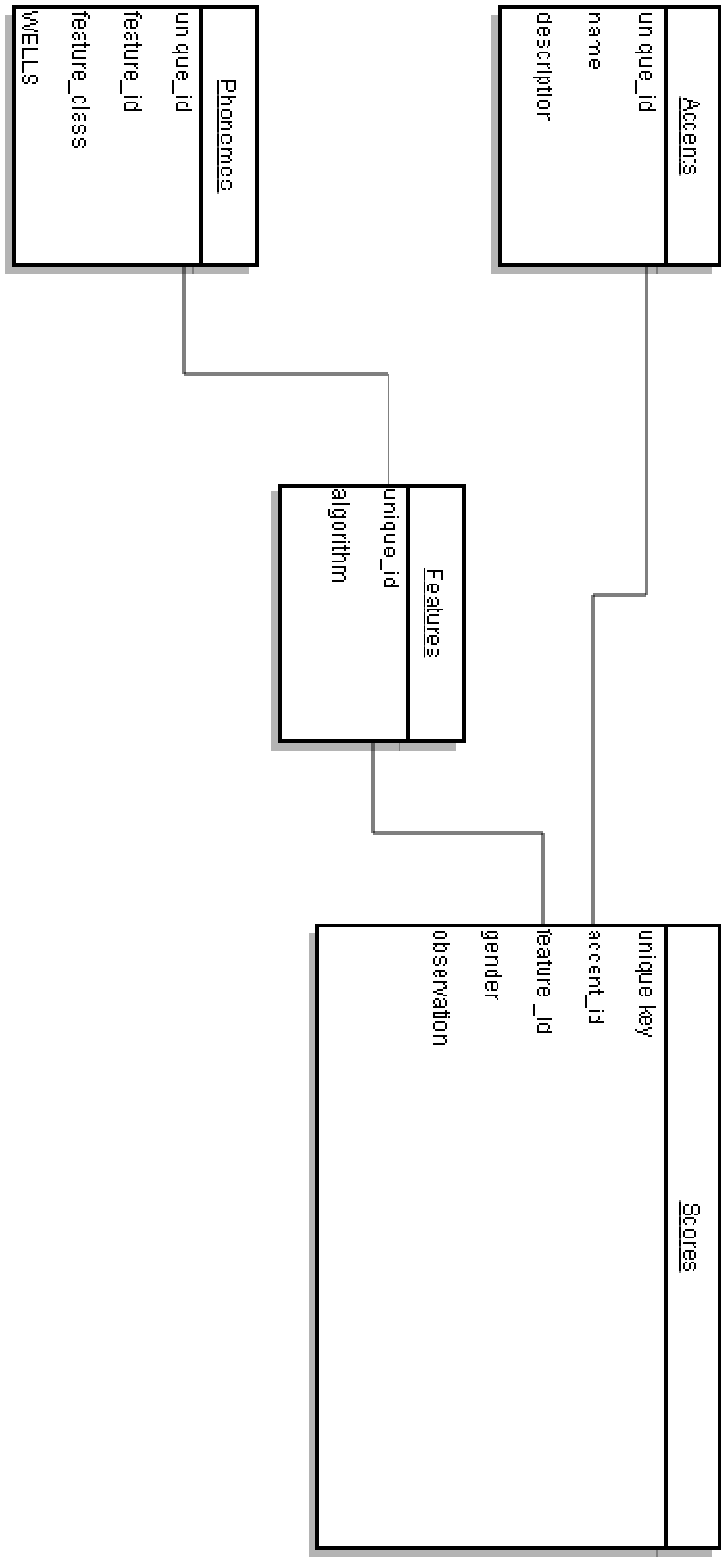
ˈ	Primary stress	ˌfoʊnəˈtɪʃən	
ˌ	Secondary stress		
ː	Long	eː	
ˑ	Half-long	eˑ	
˘	Extra-short	ĕ	
|	Minor (foot) group		
‖	Major (intonation) group		
.	Syllable break	ɹi.ækt	
‿	Linking (absence of a break)		

DIACRITICS Diacritics may be placed above a symbol with a descender, e.g. ŋ̊

◌̥	Voiceless	n̥ d̥	◌̤	Breathy voiced	b̤ a̤	◌̪	Dental	t̪ d̪
◌̬	Voiced	s̬ t̬	◌̰	Creaky voiced	b̰ a̰	◌̺	Apical	t̺ d̺
◌ʰ	Aspirated	tʰ dʰ	◌̼	Linguolabial	t̼ d̼	◌̻	Laminal	t̻ d̻
◌̹	More rounded	ɔ̹	◌ʷ	Labialized	tʷ dʷ	◌̃	Nasalized	ẽ
◌̜	Less rounded	ɔ̜	◌ʲ	Palatalized	tʲ dʲ	◌ⁿ	Nasal release	dⁿ
◌̟	Advanced	u̟	◌ˠ	Velarized	tˠ dˠ	◌ˡ	Lateral release	dˡ
◌̠	Retracted	e̠	◌ˤ	Pharyngealized	tˤ dˤ	◌̚	No audible release	d̚
◌̈	Centralized	ë	◌̴	Velarized or pharyngealized	ɫ			
◌̽	Mid-centralized	e̽	◌̝	Raised	e̝	(ɹ̝ = voiced alveolar fricative)		
◌̩	Syllabic	n̩	◌̞	Lowered	e̞	(β̞ = voiced bilabial approximant)		
◌̯	Non-syllabic	e̯	◌̘	Advanced Tongue Root	e̘			
◌˞	Rhoticity	ɚ a˞	◌̙	Retracted Tongue Root	e̙			

TONES AND WORD ACCENTS

LEVEL			CONTOUR	
e̋ or ˥	Extra high		ě or ˧˥	Rising
é ˦	High		ê ˥˦	Falling
ē ˧	Mid		e᷄ ˦˥	High rising
è ˨	Low		e᷅ ˨˧	Low rising
ȅ ˩	Extra low		e᷈ ˩˦	Rising-falling
↓	Downstep		↗	Global rise
↑	Upstep		↘	Global fall



Note: This design only works with feature which draw on WELLS vowels, and whose algorithms are pre-defined. This highlights the need for a more elaborate back-end.

```
vowelDist = function(a, b){
      r = 0;
      for(i in 10:35) {
            p = a[i]-b[i];
            r = r + p*p;
      }
      r = sqrt(r);
      return(r[1,1]);
}

diphthongDist = function(a) {
  r = 0;
  for(i in 10:22) {
    p = a[i]-a[i+13];
    r = r + p*p;
  }
  r = sqrt(r);
  return(r[1,1]);
}

createDist = function(v1, v2) {
      v1l = length(v1[,1])
      v2l = length(v2[,1])
      if(v1l==0 || v2l==0)
            return(c(NaN))
      r = c();
      for(i1 in 1:v1l) { for(i2 in 1:v2l) {
            r = c(r, vowelDist(v1[i1,], v2[i2,]))
      } }
      return(r)
}

createDiphthongDist = function(v1) {
  v1l = length(v1[,1])
  if(v1l==0)
    return(c(NaN))
  r = c();
  for(i1 in 1:v1l) {
    r = c(r, diphthongDist(v1[i1,]))
  }
  return(r)
}

averageMFCCs = function(l) {
      res = l[1,] #template only, MFCCs will be replaced
      #fields 1-4, 6-7, 9 should be the same throughout
      res[5] = "[aggregate]" #many contexts mushed into one
aggregate value
      res[8] = "[aggregate]" #idem
      for(i in 10:35) {
            res[i] = mean(l[,i])
      }
      return(res)
}

#averageOut = function(all) {
#     res = data.frame()
#     for(sp in unique(all[, 3])) {
#           for(wls in unique(all[, 9])) {
```

```
#                 tmp = averageMFCCs(all[which(all$SPEAKER==sp &
all$WELLS==wls),])
#                 res = rbind(res, tmp)
#           }
#     }
#     return(res)
#}

loadData = function() {
      if(!exists("data.all")) {
            print("Loading data from file...")
            print("(Please select file 'anal2mfcc.csv'.)")
            flush.console()
            data.all <<- read.csv(file=file.choose(), sep=",",
header=T)
            print("done.")
            flush.console()
      }
      #if(!exists("data.avg")) {
      #     print("Averaging tokens for each speaker...")
      #     flush.console()
      #     data.avg <<- averageOut(data.all)
      #     print("done.")
      #     flush.console()
      #}
}
```

```
STRUT_FOOT__DIST = function(l) {
      l.STRUT = l[which(l$WELLS=="STRUT"),]
      l.FOOT  = l[which(l$WELLS=="FOOT"),]
      res = createDist(l.STRUT, l.FOOT)
      return(res)
}

BATH_TRAP_PALM__RATIO = function(l) {
      l.BATH <<- l[which(l$WELLS=="BATH"),]
      l.TRAP <<- l[which(l$WELLS=="TRAP"),]
      l.PALM <<- l[which(l$WELLS=="PALM"),]
      d1 = createDist(l.BATH, l.PALM)
      d2 = createDist(l.BATH, l.TRAP)
      res = c()
      for(i in 1:length(d1)) { for(j in 1:length(d2)) {
            res = c(res, d1[i]/d2[j])
      }}
      return(res)
}

PALM_TRAP__DIST = function(l) {
  l.PALM = l[which(l$WELLS=="PALM"),]
  l.TRAP= l[which(l$WELLS=="TRAP"),]
  res = createDist(l.PALM, l.TRAP)
  return(res)
}

#HAPPY_KIT_FLEECE__RATIO = function(l) {
#  l.HAPPY   <<- l[which(l$WELLS=="HAPPY"),]
#  l.KIT     <<- l[which(l$WELLS=="KIT"),]
#  l.FLEECE  <<- l[which(l$WELLS=="FLEECE"),]
#  d1 = createDist(l.HAPPY, l.KIT)
#  d2 = createDist(l.HAPPY, l.FLEECE)
#  res = c()
#  for(i in 1:length(d1)) { for(j in 1:length(d2)) {
#    res = c(res, d1[i]/d2[j])
#  }}
#  return(res)
#}

HAPPY_KIT_FLEECE__RATIO = function(l) {
  l.HAPPY   <<- l[which(l$WELLS=="HAPPY"),]
  l.KIT     <<- l[which(l$WELLS=="KIT"),]
  l.FLEECE  <<- l[which(l$WELLS=="FLEECE"),]
  res = c()
  for(i in 1:length(l.HAPPY[,1])) { for(j in 1:length(l.KIT[,1])) {
for(k in 1:length(l.FLEECE[,1])) {
    tmp = createDist(l.HAPPY[i,],
l.KIT[j,])[[1]]/createDist(l.HAPPY[i,], l.FLEECE[k,])[[1]]
    res = c(res, tmp)
  }}}
  return(res)
}

FOOT_GOOSE__DIST = function(l) {
  l.FOOT = l[which(l$WELLS=="FOOT"),]
  l.GOOSE  = l[which(l$WELLS=="GOOSE"),]
  res = createDist(l.FOOT, l.GOOSE)
  return(res)
}
```

```
LOT_THOUGHT__DIST = function(l) {
  l.LOT = l[which(l$WELLS=="LOT" | l$WELLS=="CLOTH"),]
  l.THOUGHT  = l[which(l$WELLS=="THOUGHT"),]
  res = createDist(l.LOT, l.THOUGHT)
  res = mean(res)
  return(res)
}

GOAT__DIPHTHONG = function(l) {
  l.GOAT = l[which(l$WELLS=="GOAT"),]
  res = createDiphthongDist(l.GOAT)
  return(res)
}

FACE__DIPHTHONG = function(l) {
  l.FACE = l[which(l$WELLS=="FACE"),]
  res = createDiphthongDist(l.FACE)
  return(res)
}

START__RHOTIC = function(l) {
  l.START = l[which(l$WELLS=="START"),]
  res = createDiphthongDist(l.START)
  return(res)
}

FORCE.NORTH__RHOTIC = function(l) {
  l.FORCE.NORTH = l[which(l$WELLS=="FORCE" | l$WELLS=="NORTH"),]
  res = createDiphthongDist(l.FORCE.NORTH)
  return(res)
}

NURSE__RHOTIC = function(l) {
  l.NURSE = l[which(l$WELLS=="NURSE"),]
  res = createDiphthongDist(l.NURSE)
  return(res)
}

################################

featureFunctions = list(STRUT_FOOT__DIST, BATH_TRAP_PALM__RATIO,
PALM_TRAP__DIST, HAPPY_KIT_FLEECE__RATIO, START__RHOTIC,
FORCE.NORTH__RHOTIC, NURSE__RHOTIC, FOOT_GOOSE__DIST,
GOAT__DIPHTHONG, FACE__DIPHTHONG)
featureNames     = list("STRUT.FOOT",     "BATH",
"PALM.TRAP",     "HAPPY",                 "START",
"FORCE/NORTH",       "NURSE",       "FOOT.GOOSE",     "GOAT",
"FACE")
numFeatures      = length(featureFunctions)

################################

getFeatures = function(data, fFeature) {
      res = fFeature(data)
      return(res)
}

runFeature = function(featureID, sp, pop_n) {
  #featureID = n indicated the nth feature (see part between hash-
lines above)
  #sp = the speaker's name
```

```
  #pop = the population's names

  obs <<- distances[which(distances$SPEAKER==sp &
distances$FEATURE.NAME==featureNames[featureID]),"FEATURE.VALUE"]
  pop <<- distances[which(distances$SPEAKER %in% pop_n &
distances$FEATURE.NAME==featureNames[featureID]),"FEATURE.VALUE"]

  obs.avg = mean(obs, na.rm=T)

  fit = bestFit(pop)
  p_obs = getFitImage(fit, obs.avg)
  #DEPRECATED
  #fit = fitGauss(pop)
  #p_obs = dnorm(obs, mean=fit[1], sd=fit[2])
  return(p_obs)
}
```

```
fitGauss = function(d) {
  dd = density(d, na.rm=T)
  tab = data.frame(x=dd$x, y=dd$y)
  r = nls(y ~ dnorm(x, mean=mu, sd=sig), start=c(mu=mean(d,
na.rm=T), sig=sd(d, na.rm=T)), data=tab)
  res = list(fit="GAUSS", fitID=1, mu=coef(r)[["mu"]],
sig=coef(r)[["sig"]], residual=r$m$deviance())
  return(res)
}

fitGamma = function(d) {
  dd = density(d, na.rm=T)
  tab = data.frame(x=dd$x, y=dd$y)
  mu = mean(d, na.rm=T)
  sig = sd(d, na.rm=T)
  var = sig*sig
  r = nls(y ~ dgamma(x, shape=shp, scale=scl), start=c(scl=var/mu,
shp=mu*mu/var), data=tab)
  res = list(fit="GAMMA", fitID=2, shp=coef(r)[["shp"]],
scl=coef(r)[["scl"]], residual=r$m$deviance())
  return(res)
}

fitWeibull = function(d) {
  dd = density(d, na.rm=T)
  tab = data.frame(x=dd$x, y=dd$y)
  r = nls(y ~ dweibull(x, shape=shp, scale=scl), start=c(shp=5,
scl=mean(d, na.rm=T)), data=tab)
  res = list(fit="WEIBULL", fitID=3, shp=coef(r)[["shp"]],
scl=coef(r)[["scl"]], residual=r$m$deviance())
  return(res)
}
#####
numFits = 3
fitFunctions = list(fitGauss, fitGamma, fitWeibull)
#make sure this order matches the "fitID"s in the respective fit
functions above and in getFitImage below!
#####

bestFit = function(d) {
  fits = list()
  lowi = 1
  for(i in 1:numFits) {
    fits[[i]]=list(residual=Inf)
    tryCatch(expr={fits[[i]]=fitFunctions[[i]](d)},
warning=function(e){}, error=function(e){})
    if(fits[[i]]$residual<fits[[lowi]]$residual) {lowi = i}
  }
  if(fits[[lowi]]$residual==Inf) {
    res = list(fit="SIMPLE", fitID=0, mu=mean(d, na.rm=T), sig=sd(d,
na.rm=T), residual=NaN)
    return(res)
  }
  return(fits[[lowi]])
}

getFitImage = function(fit, obs) {
  if(fit$fitID==0 | fit$fitID==1) {
    return(dnorm(obs, mean=fit$mu, sd=fit$sig))
  }
```

```
  if(fit$fitID==2) {
    return(dgamma(obs, shape=fit$shp, scale=fit$scl))
  }
  if(fit$fitID==3) {
    return(dweibull(obs, shape=fit$shp, scale=fit$scl))
  }
}
```

```
init = function() {
  if(exists(".INIT"))
    return(T)
  print("Loading external functions...")
  source("common_functions.r")
  source("features.r")
  source("fitting.r")
  print("Loading external data...")
  loadData()
  .INIT = T
  return(T)
}


################################
################################

generateDistances = function() {
  init()
      #distances <<- data.frame()
      count = 261; maxcount = length(unique(data.all[,3]))

      for(sp in unique(data.all[,3])[261:274]) {
      #for each speaker
        tim = Sys.time()
            print(paste("[", count, "/", maxcount, "] Now
processing: ", sp))
        count = count+1
            print("    Isolating data from this speaker...")
            flush.console()
            data.sp <<- data.all[which(data.all$SPEAKER==sp),]
            print("        done.")
            flush.console()
            newLine.sp <<- data.frame(ACCENT=data.sp[1,1],
GENDER=data.sp[1,2], SPEAKER=sp, SPEAKER.NO=data.sp[1,4])
            for(i in 1:numFeatures) {
            #for each feature
                  print(paste("    Current feature: ",
featureNames[i]))
                  newField =
data.frame(FEATURE.NAME=featureNames[[i]])
                  newLine.feat <<- cbind(newLine.sp, newField)
                  featVals = getFeatures(data.sp,
featureFunctions[[i]])
                  for(f in featVals) {
                    newEntry = cbind(newLine.feat,
data.frame(FEATURE.VALUE=f))
                    distances <<- rbind(distances, newEntry)
                  }
            }
            print(cat("Delta t: ", Sys.time()-tim))
      }
      print(" D O N E .")
      flush.console()
}
```

```
init = function() {
  if(exists(".INIT"))
    return(T)
  print("Loading external functions...")
  source("common_functions.r")
  source("features.r")
  source("fitting.r")
  print("Loading external data...")
  loadData()
  .INIT <<- T

  maxSOE <<- 3000
  #any one feature can only shift the LR by a factor of maxSOE
either way

  return(T)
}

runAnalysis = function(sp, acc) {
  #dichotomise speaker's data
  sp1.all     = data.all[which(data.all$SPEAKER==sp),]
  sp1.gender  <<- sp1.all[1, 2]
  #sp1.avg     = averageOut(sp1.all)

  bg.all    <<- data.all[which(data.all$SPEAKER!=sp &
data.all$GENDER==sp1.gender),]

  #All other speakers of the same accent and same gender
  bg_acc.all <<- bg.all[which(bg.all$ACCENT==acc),]
  #bg_acc.avg = averageOut(bg_acc.all)
  bg_acc.names <<- unique(bg_acc.all$SPEAKER)

  #All speakers of all other accents, but the same gender
  bg_null.all <<- bg.all[which(bg.all$ACCENT!=acc),]
  #bg_null.avg = averageOut(bg_null.all)
  bg_null.names <<- unique(bg_null.all$SPEAKER)

  #initialise data structure for return vlaue
  res = list()
  res$speaker=sp
  res$accent=acc
  res$features = list()

  endLR = 1

  for(i in 1:numFeatures) {
    thisFeature = list()
    thisFeature$name = paste(i,featureNames[[i]])



    PFeat_bg    = runFeature(i, sp, bg_acc.names)
    PFeat_null  = runFeature(i, sp, bg_null.names)
    print(cat("PFeat_null", PFeat_null, "PFeat_bg", PFeat_bg))
    if(is.nan(PFeat_bg) || is.nan(PFeat_null) ||
!is.finite(PFeat_null) || !is.finite(PFeat_bg)) {
      featLR = NaN
    } else if(PFeat_null==0 && PFeat_bg==0) {
      featLR = NaN
    } else if(PFeat_bg==0) {
```

```
      featLR = 1/maxSOE
    } else if(PFeat_null==0) {
      featLR=maxSOE
    } else {
      featLR = PFeat_bg/PFeat_null
      if(featLR >   maxSOE) {featLR = maxSOE}
      if(featLR < 1/maxSOE) {featLR = 1/maxSOE}
    }


    thisFeature$p_acc   = PFeat_bg
    thisFeature$p_null  = PFeat_null
    thisFeature$LR      = featLR
    res$features[[i]] = thisFeature

    if(is.finite(featLR)) {
      endLR = endLR * featLR
    }
  }

  res$LR = endLR
  return(res)
}

interactive = function() {
  init()

  print("All speakers:")
  print(unique(data.all[,3]))
  sp = readline(prompt="Select one speaker: ")

  print("All accents:")
  print(unique(data.all[,1]))
  acc <<- readline(prompt="Select one accent: ")

  analyses <<- list()
  analyses[[1]] <<- runAnalysis(sp, acc)
}

complete = function() {
  init()

  numAnalyses = 0
  analyses <<- list()



  for(sp in unique(data.all$SPEAKER)) {
    for(acc in unique(data.all$ACCENT)) {
      numAnalyses = numAnalyses + 1
      analyses[[numAnalyses]] <<- runAnalysis(sp, acc)
      flush.console()
    }
  }
  print(" D O N E .")
}

sampleTest=function() {
  init()
  sp = "brm_f_01"
  acc ="crn"
```

```
  analyses<<-list()
  analyses[[1]] <<- runAnalysis(sp, acc)
}

export = function() {

  exp = data.frame()
  for(i in 1:length(analyses)){
    ii=analyses[[i]]
    newLine = data.frame(SPEAKER=ii$speaker, ACCENT=ii$accent,
LR=ii$LR)
    for(f in 1:numFeatures) {
      newFeat = data.frame(ii$features[[f]]$LR)
      colnames(newFeat) = ii$features[[f]]$name
      newLine = cbind(newLine, newFeat)
    }
    exp = rbind(exp, newLine)
  }
  write.table(exp, "out.csv", sep=",")
}
```

```
Cllr = function() {
  Nso = 0;
  Ndo = 0;
  sub_so = 0;
  sub_do = 0;
  FALSE_NEGATIVES = 0;
  FALSE_POSITIVES = 0;
  for(i in 1:length(analyses.xls[,1])) {
    #cat("i=", i)
    if(substr(analyses.xls[i, "SPEAKER"], 1, 3)==analyses.xls[i,
"ACCENT"]) {
      #LR SHOULD be >1
      tmp = log2(1+(1/analyses.xls[i,"LR"]))
      sub_so = sub_so + tmp
      Nso = Nso + 1
      if(analyses.xls[i,"LR"]<1) {FALSE_NEGATIVES =
FALSE_NEGATIVES+1}
    } else {
      #LR SHOULD be <1
      tmp = log2(1+analyses.xls[i,"LR"])
      sub_do = sub_do + tmp
      Ndo = Ndo + 1
      if(analyses.xls[i,"LR"]>1) {FALSE_POSITIVES =
FALSE_POSITIVES+1}
    }
  }
  print(cat(Nso, "TRUE  comparisons"))
  print(cat(Ndo, "FALSE comparisons"))
  print(cat(FALSE_POSITIVES, "FALSE POSITIVES"))
  print(cat(FALSE_NEGATIVES, "FALSE NEGATIVES"))
  sub_so = sub_so / Nso
  sub_do = sub_do / Ndo
  return((sub_do+sub_so)/2)
}

evaluate = function() {
  TP=0;FN=0;
  evalTable <<- data.frame()
  for(a in unique(data.all$ACCENT)) {
    print("----------")
    print(cat("ACCENT: ", a))
    sp.a = unique(data.all[which(data.all$ACCENT==a), "SPEAKER"])
    for(sp in sp.a) {
      newLine = data.frame(SPEAKER=sp, ACCENT=a)
      print(cat("    ", "Speaker: ", sp))
      index_best = 0
      LR_best = 0
      for(i in 1:length(analyses.xls[,1])) {
        if(!analyses.xls[i,"SPEAKER"]==sp) {
          next()
        }
        if(analyses.xls[i,"ACCENT"]==a) {
          print(cat("        LR for ", a, ": ",
analyses.xls[i,"LR"]))
          newLine = cbind(newLine,
data.frame(TRUE.LR=analyses.xls[i,"LR"]))
          if(analyses.xls[i,"LR"] < 1) {
            FN=FN+1
            #print(cat("    FALSE NEGATIVE"));
          }
```

```
        }
        if(analyses.xls[i,"LR"] > LR_best) {
          LR_best = analyses.xls[i,"LR"]
          index_best = i
        }
      }
      print(cat("        Best fit: ",
analyses.xls[index_best,"ACCENT"], " (LR=",
analyses.xls[index_best,"LR"], ")"))
      newLine = cbind(newLine,
data.frame(BEST.ACC=analyses.xls[index_best,"ACCENT"],
BEST.LR=analyses.xls[index_best,"LR"]))
      if(analyses.xls[index_best,"ACCENT"]==a) {
        print(cat("    TRUE POSITIVE"));
        TP=TP+1
      }
      evalTable <<- rbind(evalTable, newLine)
    }
  }
  #print("")
  print(cat("TRUE POSITIVES:  ", TP))
  print(cat("FALSE NEGATIVES: ", FN))
  print(cat("Correct 2nd, 3rd, ... choices: ",
length(unique(data.all$SPEAKER))-TP-FN))
}
```