Data, data documentation and analysis scripts for

A cognitively grounded measure of pronunciation distance

Martijn Wieling[1,2] & John Nerbonne[1] & Jelke Bloem[3] & Charlotte Gooskens[1] & Wilbert Heeringa[1] & R. Harald Baayen[2,4]

[1]University of Groningen, the Netherlands & [2]Eberhard Karls University, Germany & [3]University of Amsterdam, the Netherlands & [3]University of Alberta, Canada

Journal: **PLOS ONE**
Open access: `http://dx.plos.org/10.1371/journal.pone.0075734`

Abstract

In this study we develop pronunciation distances based on naive discriminative learning (NDL). Measures of pronunciation distance are used in several subfields of linguistics, including psycholinguistics, dialectology and typology. In contrast to the commonly used Levenshtein algorithm, NDL is grounded in cognitive theory of associative learning and is able to generate asymmetrical pronunciation distances. In a first study, we validated the NDL-based pronunciation distances by comparing them to a large set of native-likeness ratings given by native American English speakers when presented with accented English speech. In a second study, the NDL-based pronunciation distances were validated on the basis on perceptual dialect distances of Norwegian speakers. Results indicated that the NDL-based pronunciation distances matched perceptual distances reasonably well with correlations ranging between 0.7 and 0.8. While the correlations were comparable to those obtained using the Levenshtein distance, the NDL-based approach is more flexible as it is also able to incorporate acoustic information other than sound segments.

Keywords: naive discriminative learning, dialectometry, pronunciation distances, Levenshtein distance, Rescorla-Wagner model.

1 Packages and functions

```
library(ndl)
library(reshape)
library(ltm)
packageVersion("ndl")

## [1] '0.2.10'

packageVersion("reshape")

## [1] '0.8.4'

packageVersion("ltm")

## [1] '0.9.9'

source('functions/functions.R') # custom NDL processing functions
set.seed(111) # seed for random number generator
```

2 English accents data sets

```
# In the first load statement below, "tri" can be replaced with
# "uni", "bi", "unibi", "unitri", "bitri", or "unibitri"
# for cues consisting of unigrams, bigrams, unigrams and trigrams,
# unigrams and trigrams, bigrams and trigrams, or unigrams, bigrams
# and trigrams. In addition, "nodia" can be replaced with "dia" to
# use cues incorporating diacritics.
load("data/accents-tri-nodia.rda") # the dataframe is available as: accents
load("data/judgeAccents.rda")
load("data/accentRatings.rda")
```

Legenda accents (27100 observations of 8 variables):

1. Outcomes : the meaning outcomes (i.e. words)
2. Frequency : the corpus-based frequency of each outcome
3. Speaker : the English speaker (native or non-native)
4. Pronunciation : the pronunciation of the speaker for a given meaning in the International Phonetic Alphabet (IPA)
5. USnative : binary value indicating if the speaker is a native English speaker born in the U.S.
6. ForCorrel : binary value indicating if perceptual judgements are available for this speaker (stored per speaker in judgeAccents data frame)
7. CuesRaw : trigram cues generated from the pronunciation (# marks the word start and end) in IPA
8. Cues : as above, but using numerical coding

Legenda judgeAccents (286 observations of 3 variables):

1. Speaker : the speaker
2. Nativelikeness : the average nativelikeness rating given by native U.S. English-speaking judges
3. DistLD : average Levenshtein distance (log-transformed, using sensitive sound segment distances) with respect to the 115 U.S. English speakers in the data set

Legenda accentRatings (1143 observations of 290 variables):

1. ID : participant ID
2. Gender : the gender of the participant
3. Age : the age of the participant
4. State : the state where the participant was born
5. english23 ... korean17 : the samples for which ratings were obtained (corresponding with the Speech Accent Archive ID)

3 Analysis and results: English accents data

```
# Note that some of the values below are slightly different from the paper,
# due to small errors in the original dataset. The values below are based
# on the corrected dataset.
table(accentRatings$Gender)

##
##   F   M
## 485 658

table(accentRatings$State)

##
##        Alabama         Alaska        Arizona       Arkansas     California
##             12              4              7              7            151
##       Colorado    Connecticut       Delaware        Florida        Georgia
##             21             16              3             22             20
##         Hawaii          Idaho       Illinois        Indiana           Iowa
##              6              1             64             12             14
##         Kansas       Kentucky      Louisiana          Maine       Maryland
##             10              3             15              7             28
##  Massachusetts       Michigan      Minnesota    Mississippi       Missouri
##             68             38             23              3             17
##        Montana       Nebraska  New Hampshire     New Jersey     New Mexico
##              4              4             11             42              7
##       New York North Carolina   North Dakota           Ohio       Oklahoma
##            115             19              1             66             12
##         Oregon   Pennsylvania   Rhode Island South Carolina   South Dakota
##             20             54              5             10              5
##      Tennessee          Texas           Utah        Vermont       Virginia
##              9             55             14              7             36
##     Washington  Washington DC  West Virginia      Wisconsin        Wyoming
##             42             13              2             15              2

mean(accentRatings$Age)

## [1] 36.21

sd(accentRatings$Age)

## [1] 13.87

# number of rated samples per participant
mean(rowSums(!is.na(accentRatings[,5:ncol(accentRatings)])))

## [1] 41

sd(rowSums(!is.na(accentRatings[,5:ncol(accentRatings)])))

## [1] 14.02
```

```r
# cronbach's alpha of the ratings
cronbach.alpha(accentRatings[,c(5:ncol(accentRatings))],na.rm=T)

##
## Cronbach's alpha for the 'accentRatings[, c(5:ncol(accentRatings))]' data-set
##
## Items: 286
## Sample units: 1143
## alpha: 0.853
```

```r
corAccentsNDL <- function(accents,judgeAccents,forPlot=F) {
    nativeAElstnr = NA
    # select random sample of 58 native American English speakers (NAE) and train
    # the native American English listener model (associations between cues and
    # outcomes) on the basis of the pronunciations of these speakers. If the
    # training did not converge (nativeAElstnr will be NA), the process is repeated
    # with another random sampling
    while (all(is.na(nativeAElstnr))) { # training might not converge
        lstnrNAEspkrs = sample(unique(accents[accents$USnative==1, ]$Speaker), 58)
        nativeAElstnr = estimateWeightsNDL(accents, lstnrNAEspkrs)
    }
    # use the pronunciation of the remaining (57) native American English speakers
    # to model an average native American English (NAE) speaker
    speakerNAEspkrs = unique(accents[!accents$Speaker %in% lstnrNAEspkrs &
                                     accents$USnative==1, ]$Speaker)
    # obtain the average activations per meaning for the NAE listener when
    # exposed to the speech of the average NAE speaker
    avgSpkNAEact = estimateAvgActivationsNDL(accents, speakerNAEspkrs,
                                             nativeAElstnr)
    # select all individual speakers
    indivSpkrs = as.character(unique(accents[accents$ForCorrel==1, ]$Speaker))
    # store NDL-based pron. dist. in data frame with nativelikeness ratings
    judgeAccents$distNDL = NA
    # compare the activations for every individual speaker (averaged over all
    # meanings) to the average NAE speaker => NDL-based pronunciation differences
    for (spk in indivSpkrs) {
        judgeAccents[spk, ]$distNDL <-
            activationDifference(accents, spk, nativeAElstnr, avgSpkNAEact)
    }
    # correlation between NDL-based pronunciation distances and nativelikeness ratings
    correl = cor(judgeAccents$distNDL, judgeAccents$Nativelikeness)
    logcorrel = cor(log(judgeAccents$distNDL), judgeAccents$Nativelikeness)
    # correlation between log-transformed NDL and Levenshtein distance
    logcorrelLeven = cor(log(judgeAccents$distNDL), judgeAccents$DistLD)
    if (forPlot) {
        return(judgeAccents)
    } else {
        return(c(correl, logcorrel, logcorrelLeven))
    }
}
```

```
# obtain average result across 100 iterations
nrep = 100
correls = rep(NA,nrep)
logcorrels = rep(NA,nrep)
logcorrelsLeven = rep(NA,nrep)
for (i in 1:nrep) {
    result = corAccentsNDL(accents,judgeAccents) # duration: 50 seconds
    correls[i] = result[1]
    logcorrels[i] = result[2]
    logcorrelsLeven[i] = result[3]
}
# saved as this takes about 90 minutes
save(correls,logcorrels,logcorrelsLeven,file='results/corAccents.rda')
```

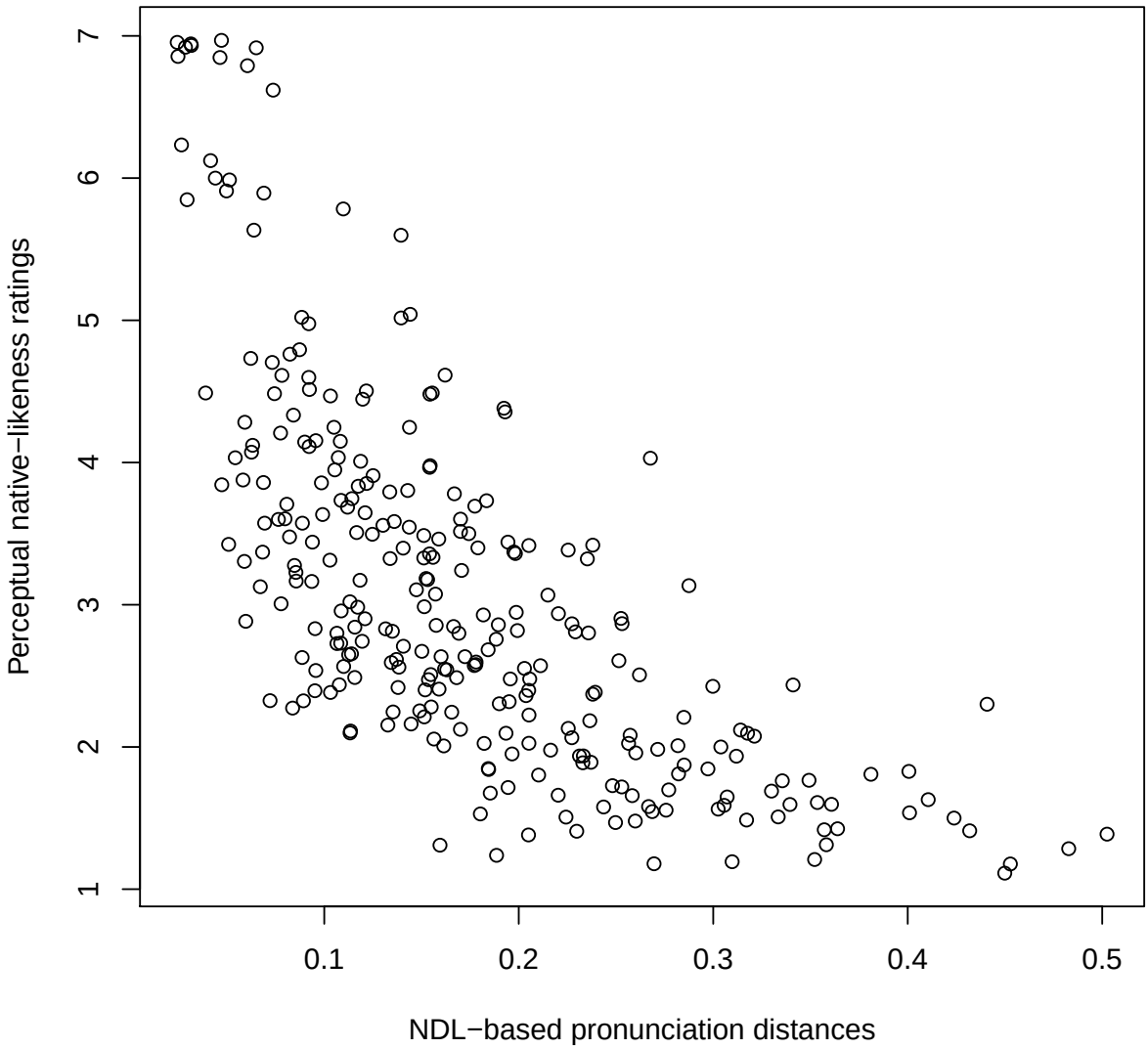
```
load('results/corAccents.rda')
# main results
mean(correls)

## [1] -0.7166

mean(logcorrels)

## [1] -0.8007

# example plot showing the logarithmic relationship
plotData = corAccentsNDL(accents,judgeAccents,forPlot=TRUE)
plot(plotData$distNDL,plotData$Nativelikeness,
     xlab='NDL-based pronunciation distances',
     ylab='Perceptual native-likeness ratings')
```




```
# correlation between Levenshtein distances and accent judgements
cor(judgeAccents$Nativelikeness,judgeAccents$DistLD)

## [1] -0.8147

# correlation between Levenshtein and NDL distances
mean(logcorrelsLeven)

## [1] 0.8896

# average agreement of the individual raters with mean ratings
```

```
scores = accentRatings[,c(5:ncol(accentRatings))] # select ratings
sums = as.data.frame(colMeans(scores,na.rm=T)) # mean rating per sample
cors = cor(sums,t(scores),use='pairwise.complete.obs')
mean(cors,na.rm=T)

## [1] 0.8414

# Distinct performance using all combinations of unigrams, bigrams
# and trigrams, with and without diacritics. For speed, only a single
# run is conducted and results may differ slightly from the paper.
load("data/accents-uni-nodia.rda")
corAccentsNDL(accents,judgeAccents)[1:2] # normal, log-transformed

## [1] -0.5650 -0.5979

load("data/accents-uni-dia.rda")
corAccentsNDL(accents,judgeAccents)[1:2]

## [1] -0.5796 -0.6053

load("data/accents-bi-nodia.rda")
corAccentsNDL(accents,judgeAccents)[1:2]

## [1] -0.6884 -0.7845

load("data/accents-bi-dia.rda")
corAccentsNDL(accents,judgeAccents)[1:2]

## [1] -0.7062 -0.7964

load("data/accents-tri-nodia.rda")
corAccentsNDL(accents,judgeAccents)[1:2]

## [1] -0.7086 -0.7931

load("data/accents-tri-dia.rda")
corAccentsNDL(accents,judgeAccents)[1:2]

## [1] -0.7504 -0.8228

load("data/accents-unibi-nodia.rda")
corAccentsNDL(accents,judgeAccents)[1:2]

## [1] -0.6706 -0.7723

load("data/accents-unibi-dia.rda")
corAccentsNDL(accents,judgeAccents)[1:2]

## [1] -0.7117 -0.8013

load("data/accents-unitri-nodia.rda")
corAccentsNDL(accents,judgeAccents)[1:2]
```

```
## [1] -0.7069 -0.7974

load("data/accents-unitri-dia.rda")
corAccentsNDL(accents,judgeAccents)[1:2]

## [1] -0.7482 -0.8218

load("data/accents-bitri-nodia.rda")
corAccentsNDL(accents,judgeAccents)[1:2]

## [1] -0.7180 -0.8091

load("data/accents-bitri-dia.rda")
corAccentsNDL(accents,judgeAccents)[1:2]

## [1] -0.7363 -0.8125

load("data/accents-unibitri-nodia.rda")
corAccentsNDL(accents,judgeAccents)[1:2]

## [1] -0.6949 -0.7784

load("data/accents-unibitri-dia.rda")
corAccentsNDL(accents,judgeAccents)[1:2]

## [1] -0.7379 -0.8155
```

4 Norwegian dialect data sets

```
# In the first load statement below, "tri" can be replaced with
# "uni", "bi", "unibi", "unitri", "bitri", or "unibitri"
# for cues consisting of unigrams, bigrams, unigrams and trigrams,
# unigrams and trigrams, bigrams and trigrams, or unigrams, bigrams
# and trigrams. In addition, "nodia" can be replaced with "dia" to
# use cues incorporating diacritics.
load("data/norwegian-tri-nodia.rda") # the dataframe is available as: norwegian
load("data/judgeNorwegian.rda")
load("data/ldNorwegian.rda")
```

Legenda norwegian (1253 observations of 6 variables):

1. Outcomes : the meaning outcomes (i.e. words)
2. Frequency : the corpus-based frequency of each outcome
3. Speaker : the norwegian dialect speaker
4. Pronunciation : the pronunciation of the speaker for a given meaning in the International Phonetic Alphabet (IPA)
5. CuesRaw : trigram cues generated from the pronunciation (# marks the word start and end) in IPA
6. Cues : as above, but using numerical coding

Legenda judgeNorwegian (15 observations, i.e. listeners, of 15 variables, i.e. speakers):

1. Bergen : perceptual pronunciation distance from a speaker from Bergen
2. Bjugn : perceptual pronunciation distance from a speaker from Bjugn
3. Bodø : perceptual pronunciation distance from a speaker from Bodø
4. Bø : perceptual pronunciation distance from a speaker from Bø
5. Borre : perceptual pronunciation distance from a speaker from Borre
6. Fræna : perceptual pronunciation distance from a speaker from Fræna
7. Halden : perceptual pronunciation distance from a speaker from Halden
8. Herøy : perceptual pronunciation distance from a speaker from Herøy
9. Larvik : perceptual pronunciation distance from a speaker from Larvik
10. Lesja : perceptual pronunciation distance from a speaker from Lesja
11. Lillehammer : perceptual pronunciation distance from a speaker from Lillehammer

12. Stjørdal : perceptual pronunciation distance from a speaker from Stjørdal
13. Time : perceptual pronunciation distance from a speaker from Time
14. Trondheim : perceptual pronunciation distance from a speaker from Trondheim
15. Verdal : perceptual pronunciation distance from a speaker from Verdal

Legenda ldNorwegian (15 observations of 15 variables):

1. Bergen : Levenshtein distance compared to the pronunciation of Bergen
2. Bjugn : Levenshtein distance compared to the pronunciation of Bjugn
3. Bodø : Levenshtein distance compared to the pronunciation of Bodø
4. Bø : Levenshtein distance compared to the pronunciation of Bø
5. Borre : Levenshtein distance compared to the pronunciation of Borre
6. Fræna : Levenshtein distance compared to the pronunciation of Fræna
7. Halden : Levenshtein distance compared to the pronunciation of Halden
8. Herøy : Levenshtein distance compared to the pronunciation of Herøy
9. Larvik : Levenshtein distance compared to the pronunciation of Larvik
10. Lesja : Levenshtein distance compared to the pronunciation of Lesja
11. Lillehammer : Levenshtein distance compared to the pronunciation of Lillehammer
12. Stjørdal : Levenshtein distance compared to the pronunciation of Stjørdal
13. Time : Levenshtein distance compared to the pronunciation of Time
14. Trondheim : Levenshtein distance compared to the pronunciation of Trondheim
15. Verdal : Levenshtein distance compared to the pronunciation of Verdal

5 Analysis and results: Norwegian dialect data

```
corNorwegianNDL <- function(norwegian,judgeNorwegian) {
    dialect = as.character(unique(norwegian$Speaker))
    # store NDL-based pronunciation distances in matrix
    distNDL = matrix(NA,nrow=length(dialect),ncol=length(dialect),byrow=T)
    rownames(distNDL) = colnames(distNDL) = dialect
    for (i in 1:length(dialect)) {
       # train model representing listener from dialect i
       Li <- estimateWeights(norwegian[norwegian$Speaker == dialect[i],
                             c("Cues","Outcomes","Frequency")],removeDuplicates=F)
       # obtain activation per meaning when Li is exposed to pronunciations of
       # dialect i
       baselineAct = estimateAvgActivationsNDL(norwegian, dialect[i], Li)
       # calculate activation differences (across all meanings) between the
       # activation of Li when exposed to dialect i and all other dialects j
       for (j in 1:length(dialect)) {
           distNDL[i,j] = activationDifference(norwegian, dialect[j],
                                               Li, baselineAct)
       }
    }
    # correlate NDL-based pronunciation distances and perceptual dialect distances
    judgeNorwegian = as.matrix(judgeNorwegian)
    correl = cor(matrix(distNDL, ncol=1), matrix(judgeNorwegian, ncol=1),
                 use="pairwise.complete.obs") # correlation without diagonal
    logcorrel = cor(matrix(log(distNDL), ncol=1), matrix(judgeNorwegian, ncol=1),
                    use="pairwise.complete.obs") # log-transformed value
    distLeven = data.matrix(read.table('data/source/norwegianlevenshtein.txt',sep='\t',row.names=1,heade
    diag(distLeven) = NA
    levencorrel = cor(matrix(distNDL, ncol=1), matrix(distLeven, ncol=1),
                    use="pairwise.complete.obs") # correlation with Levenshtein
    return(c(correl,logcorrel,levencorrel))
}
```

```
# main results: correlation between NDL and perceptual dialect distances
results = corNorwegianNDL(norwegian,judgeNorwegian)
results[1:2] # normal, log-transformed
```

```
## [1] 0.6800 0.7156
```

```
# Correlation between Levenshtein and NDL-based pronunciation distances.
# Note that the reported result in the paper was too high (0.95), as the
# values on the diagonal were incorrectly taken into account.
results[3]
```

```
## [1] 0.8872
```

```
# Distinct performance using all combinations of unigrams, bigrams
# and trigrams, with or without diacritics.
load("data/norwegian-uni-nodia.rda")
corNorwegianNDL(norwegian,judgeNorwegian)[1:2] # normal, log-transformed
```

```
## [1] 0.1004 0.3137

load("data/norwegian-uni-dia.rda")
corNorwegianNDL(norwegian,judgeNorwegian)[1:2]

## [1] 0.3496 0.4924

load("data/norwegian-bi-nodia.rda")
corNorwegianNDL(norwegian,judgeNorwegian)[1:2]

## [1] 0.6710 0.7065

load("data/norwegian-bi-dia.rda")
corNorwegianNDL(norwegian,judgeNorwegian)[1:2]

## [1] 0.6539 0.6722

load("data/norwegian-tri-nodia.rda")
corNorwegianNDL(norwegian,judgeNorwegian)[1:2]

## [1] 0.6800 0.7156

load("data/norwegian-tri-dia.rda")
# Note that the two correlations reported in the paper are slightly different
# from those below due to a small error. The values below are correct.
corNorwegianNDL(norwegian,judgeNorwegian)[1:2]

## [1] 0.6461 0.6619

load("data/norwegian-unibi-nodia.rda")
corNorwegianNDL(norwegian,judgeNorwegian)[1:2]

## [1] 0.6668 0.7035

load("data/norwegian-unibi-dia.rda")
corNorwegianNDL(norwegian,judgeNorwegian)[1:2]

## [1] 0.6473 0.6674

load("data/norwegian-unitri-nodia.rda")
corNorwegianNDL(norwegian,judgeNorwegian)[1:2]

## [1] 0.6777 0.7139

load("data/norwegian-unitri-dia.rda")
corNorwegianNDL(norwegian,judgeNorwegian)[1:2]

## [1] 0.6475 0.6659

load("data/norwegian-bitri-nodia.rda")
corNorwegianNDL(norwegian,judgeNorwegian)[1:2]
```

```
## [1] 0.6823 0.7167

load("data/norwegian-bitri-dia.rda")
corNorwegianNDL(norwegian,judgeNorwegian)[1:2]

## [1] 0.6528 0.6699

load("data/norwegian-unibitri-nodia.rda")
corNorwegianNDL(norwegian,judgeNorwegian)[1:2]

## [1] 0.6797 0.7148

load("data/norwegian-unibitri-dia.rda")
corNorwegianNDL(norwegian,judgeNorwegian)[1:2]

## [1] 0.6493 0.6678
```

6 Appendix: custom functions

Functions for estimating weights

```
estimateWeightsNDL

## function(dat,speakersForTraining) {
## # estimates the weights using NDL
##     trainingProns = dat[dat$Speaker %in% speakersForTraining,]
##     trainingProns = droplevels(trainingProns) # drop unused levels
##     trainingProns = correctFrequencies(trainingProns) # corrects frequencies
##
##     # training might not converge, NA is returned when it did not converge
##     convergenceError <- tryCatch(
##         assign("result",
##             estimateWeights(trainingProns[,c("Cues","Outcomes","Frequency")],
##                             removeDuplicates=F)),
##         error=function(e) e
##     )
##     if(!inherits(convergenceError, "error")){
##         return(result)
##     } else {
##         return(NA)
##     }
## }

correctFrequencies

## function(dat) {
## # corrects frequencies, such that the summed frequencies match the frequency
## # extracted from the corpus
##     tmp = data.frame(table(dat$Outcomes))
##     colnames(tmp) = c("Outcomes","Count")
##     dat2 = merge(dat,tmp,by="Outcomes")
##     dat = dat2
##     dat$Frequency = round(dat$Frequency / dat$Count)
##     dat$Count = NULL
##     return(dat)
## }
```

Functions for estimating activations

```
activationDifference

## function(dat,speaker,listenerWeights,compareAct) {
## # calculate activation difference between a specific speaker, compared
## # to the given activations
##     act = estimateAvgActivationsNDL(dat,speaker,listenerWeights)
##
##     # subset if words are missing
##     compareAct = compareAct[rownames(compareAct) %in% rownames(act),
```

```
##                              colnames(compareAct) %in% colnames(act)]
##     act = act[rownames(act) %in% rownames(compareAct),
##               colnames(act) %in% colnames(compareAct)]
##
##     return(mean(abs(diag(compareAct) - diag(act)))) # difference at diagonal
## }

estimateAvgActivationsNDL

## function(dat,speakersSubset,listenerWeights) {
## # estimate NDL activations
##     speakers = droplevels(dat[dat$Speaker %in% speakersSubset,])
##     speakersAct = estimateActivations(speakers[,c("Cues","Outcomes")],
##                                       listenerWeights)$activationMatrix
##     return(averageActivations(speakersAct,speakers))
## }

averageActivations

## function(act, input) {
## # calulate average activations over all outcomes
##     actdf = data.frame(act)
##     actdf <- rename(actdf, c("for."="for")) # correction of wrong column name
##     actdf$Outcomes = input$Outcomes
##
##     # average activations over all outcomes
##     actdf = aggregate(actdf[ ,c(1:(ncol(actdf)-1))],
##                       by=list(Outcomes = actdf$Outcomes), FUN="mean")
##     rownames(actdf) = actdf$Outcomes
##     actdf$Outcomes = NULL
##     actdf = actdf[order(row.names(actdf)), order(colnames(actdf))]
##
##     # square matrix, each row must have its associated column
##     actdf = actdf[rownames(actdf) %in% colnames(actdf), ]
##     actdf = actdf[ ,colnames(actdf) %in% rownames(actdf)]
##
##     return(data.matrix(actdf))
## }
```

Functions for cue generation and conversion

```
generateCues

## function (strings, grams = c(3), tokenized = T, sepToken = '.') {
## # generates trigram cues (tokens in the input file are separated by '.')
##
##     ngram.fnc = function(s, n) {
##         if (n == 1) { # remove hash cues for unigrams
##             s = sub(paste('#', sepToken, sep=''), '', s)
##             s = sub(paste(sepToken, '#', sep=''), '', s)
##         }
```

```
##         tokens = unlist(strsplit(s, sepToken, fixed=T))
##         len = length(tokens)
##         ng = NULL
##         for (i in 1:(len-n+1)) {
##             ng = c(ng, paste(tokens[i:(i+n-1)], collapse=''))
##         }
##         return(paste(ng, collapse = "_"))
##     }
##
##     if (!tokenized) sepToken = ''
##     letters = sapply(strings, FUN = function(s) paste('#', sepToken, s,
##                                                       sepToken,'#', sep = ""))
##
##     for (i in 1:length(grams)) {
##         cuesi = unlist(lapply(letters, FUN = ngram.fnc, grams[i]))
##         if (exists("cues") == 0) {
##             cues = cuesi
##         } else {
##             cues = paste(cues, cuesi, sep = "_")
##         }
##     }
##     return(cues)
## }

convertCues

## function(cuelists) {
## # converts cues to numerical cues
##
##     convertCueList.fnc = function(cuelist) {
##
##         toInt.fnc = function(char) {
##             if (char != '_') {
##                 return(paste(utf8ToInt(char), "*", sep=''))
##             } else {
##                 return(char)
##             }
##         }
##
##         return( paste(unlist(lapply(unlist(strsplit(cuelist, '')),
##                                     FUN=toInt.fnc)), collapse=''))
##     }
##
##     return(unlist(lapply(cuelists, FUN=convertCueList.fnc)))
## }

convertBack

## function(codedCuelists) {
## # converts numerical cues back to character-based cues
##
```

```
##     convertCueListBack.fnc = function(codedCuelist) {
##
##         toUTF8.fnc = function(token) {
##             token = sub('_', '', token)
##
##             tok = type.convert(token, as.is=T)
##             if (is.numeric(tok)) {
##                 return(intToUtf8(tok))
##             } else {
##                 return(token)
##             }
##         }
##
##         return(paste(unlist(lapply(unlist(strsplit(codedCuelist, '*', fixed=T)),
##                                                     FUN=toUTF8.fnc)), collapse=''))
##     }
##
##     return(unlist(lapply(codedCuelists, FUN=convertCueListBack.fnc)))
## }
```