The interactive origin of iconiciy: Estimating spikiness ratings

Load libraries

```
library(ngram)
library(gplots)
library(lme4)
library(party)
```

Helper functions

Estimate the spikiness ratngs of words from a finite sample of participant judgements, controlling for a random effect for each participant.

```
predictSpikinessWithLMER = function(ratings.words){
  m.words = lmer(RatingSpikiness~ Item + (1|Part), data=ratings.words)
  #plot(ratings.words$RatingSpikiness,resid(m.words))
  words = sort(unique(ratings.words$Item))
  words.predictions = predict(m.words,newdata=data.frame(Item=words, Part=1), re.form=NULL)
  names(words.predictions) = words
  #cor(words.predictions, tapply(ratings.words$RatingSpikiness, ratings.words$Item, mean))
  return(words.predictions)
}
```

Take a set of words and generate a feature matrix of ngrams.

```
makeFeatureFrame = function(dx,ngrams){
  r = matrix(nrow=nrow(dx), ncol=2+length(ngrams))
  r[,1] = dx$Item
  r[,2] = dx$RatingSpikiness
  colnames(r) = c("Item","RatingSpikiness",ngrams)
  for(i in 3:ncol(r)){
    r[,i] = grepl(colnames(r)[i],r[,1])
  }
  r = as.data.frame(r)
  for(i in 3:ncol(r)){
    r[,i] = as.logical(r[,i])
  }
  return(r)
}
```

Load data

The data includes spikiness ratings for words and individual letters from several participants.

```
ratings = read.delim("../data/ratings/SpikinessRatings", sep='\t', stringsAsFactors = F)
```

Check whether there are effects by participant sex, age or the direction that the Likert scale was presented.

```
m0 = lmer(RatingSpikiness ~ Sex + Age + Likert + (1|Item)  + (1|Part), data=ratings)
summary(m0)
```

```
## Linear mixed model fit by REML ['lmerMod']
## Formula: RatingSpikiness ~ Sex + Age + Likert + (1 | Item) + (1 | Part)
##    Data: ratings
##
## REML criterion at convergence: 9124.3
##
## Scaled residuals:
##     Min      1Q  Median      3Q     Max
## -3.3831 -0.6667  0.0133  0.6978  3.0814
##
## Random effects:
##  Groups   Name        Variance Std.Dev.
##  Item     (Intercept) 1.4523   1.2051
##  Part     (Intercept) 0.0917   0.3028
##  Residual             2.1603   1.4698
## Number of obs: 2411, groups:  Item, 163; Part, 16
##
## Fixed effects:
##              Estimate Std. Error t value
## (Intercept)  3.942780   0.448760   8.786
## Sexmale      0.091512   0.192050   0.477
## Age          0.002476   0.018451   0.134
## Likertspiky -0.040992   0.173811  -0.236
##
## Correlation of Fixed Effects:
##             (Intr) Sexmal Age
## Sexmale      0.025
## Age         -0.929 -0.215
## Likertspiky -0.288  0.283  0.084
```

There are no significant effects.

Split the data into ratings for letters and ratings for whole words. Then get an esimation of the mean rating for each item.

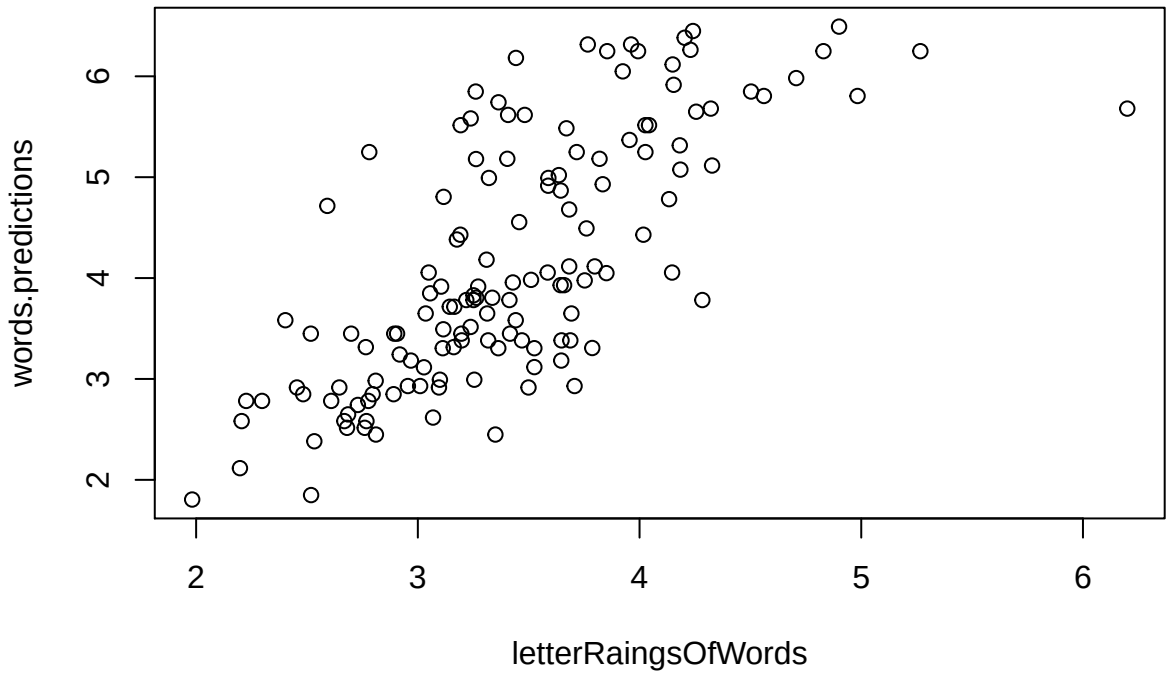
```
ratings.letters = ratings[nchar(ratings$Item)==1,]
ratings.words = ratings[nchar(ratings$Item)>1,]


letter.predictions = predictSpikinessWithLMER(ratings.letters)
words.predictions = predictSpikinessWithLMER(ratings.words)
```

Model based on letter scores

Estimate the spikiness rating of a word by taking the mean spikiness score for each of the letters in the word. This can be used as baseline to see if it's worth building a more complicated model.

```
letterRaingsOfWords = sapply(names(words.predictions), function(X){
  mean(letter.predictions[strsplit(X,'')[[1]]])
})
```

```r
plot(letterRaingsOfWords, words.predictions)
```




```r
# Baseline for just using letters:
cor(letterRaingsOfWords, words.predictions)
```

```
## [1] 0.733007
```

The model predictions correlate with the real values with r = 0.73 (on seen data).

Random forests model based on unigrams and bigrams

Build a model of spikiness ratings based on a training set, then predict the spikiness ratings of an unseen test set.

Set parameters:

```
proportionOfDataInTrainingSet = 0.75
numberOfFolds = 20
maxNGram = 2
```

Run the trainig and test cycles:

```
# set random seed
set.seed(2189)

# variable for storing correlation between predictions and real ratings for each run
res = c()

for(run in 1:numberOfFolds){
  # get list of items
  items = unique(ratings$Item)
  # select training items:
  #  all single characters plus a random selection of words
  trainItems = c(items[nchar(items)==1],
                 sample(items[nchar(items)>1],
                 sum(nchar(items)>1)*proportionOfDataInTrainingSet))
  # test items - unseen items
  testItems = items[!items %in% trainItems]

  # get data for training and test items
  trainSet = ratings[ratings$Item %in% trainItems,]
  testSet = ratings[ratings$Item %in% testItems,]

  # make list of ngrams in training set
  ngrams = unique(unlist(sapply(trainSet$Item, function(X){
    if(nchar(X)==1){
      return(X)
    }
    unique(ngram_asweka(X,min=1,max=maxNGram,sep=''))
  })))

  # make feature frame of ngrams
  rTrain = makeFeatureFrame(trainSet,ngrams)
  rTrain$RatingSpikiness = as.numeric(rTrain$RatingSpikiness)

  # predict mean spikiness with lmer for test set
  rTest.predictions = predictSpikinessWithLMER(testSet)
  # biuld feature frame of ngrams for test set
  rTest = makeFeatureFrame(testSet[!duplicated(testSet$Item),],ngrams)
  rTest$RatingSpikiness = rTest.predictions[rTest$Item]

  colselect = 2:ncol(rTrain)

  # Build the random forest
  cf = cforest(RatingSpikiness ~ . ,
```

```
                data= rTrain[,colselect],
                controls = cforest_control(mtry = 10))
  #importance=  varimp(cf)
  #dotplot(sort(importance))

  predictedRatings = predict(cf,newdata=rTest[,colselect])

  res = c(res,cor(predictedRatings,rTest$RatingSpikiness))
}
```

The mean correlation between predictions and real data was r = 0.886. This is an acceptable level and a marked improvement on the baseline model (also consiering the random forests predictions were on unseen data).

Informal testing found that performance did not increase significantly when including trigrams.

Make model with whole data

```
ngrams.all = unique(unlist(sapply(ratings$Item, function(X){
  if(nchar(X)==1){
    return(X)
  }
  unique(ngram_asweka(X,min=1,max=maxNGram,sep=''))
})))

rAll = makeFeatureFrame(ratings,ngrams.all)
rAll$RatingSpikiness = as.numeric(rAll$RatingSpikiness)

rAll.predictions = predictSpikinessWithLMER(ratings)
rAll = makeFeatureFrame(ratings[!duplicated(ratings$Item),],ngrams.all)
rAll$RatingSpikiness = rAll.predictions[rAll$Item]

cf.all = cforest(RatingSpikiness ~ . ,
             data= rAll[,2:ncol(rAll)],
             controls = cforest_control(mtry = 10))

tx =ctree(RatingSpikiness ~ ., data=rAll[,2:ncol(rAll)])
```

Build a function to predict iconicity results.

```
getIconicityFromRForest = function(words){

  xdat = t(sapply(
    words,
    function(word){
      sapply(
        ngrams.all,
        function(X){
          grepl(X,word)
    })}))
  xdat = as.data.frame(cbind(rep(NA,nrow(xdat)),rep(NA,nrow(xdat)),xdat))

  predictedRatings = predict(cf.all,newdata=xdat)
  return(as.vector(predictedRatings))
}
```

Save the function and variables to be used in other scripts.

```
save(getIconicityFromRForest, ngrams.all, cf.all, file='PredictSpikinessModel.RDat')
```

Here's a sample tree from the forest:

```
plot(tx, terminal_panel=node_boxplot(tx,id=F))
```