Chapter 9

Appendices: Model and Simulation

Structure

First, the model will be shown, then the model testing and then the statistical test.

RSA-Model

The developed RSA-Modelling Framework is implemented in the following pages.

The simple listener function determines the hypothetical listener's object choice given the objects to choose from and its preferences, determining P(obj | utt, listener's object preferences).

```
    knitr::opts_chunk$set(fig.width=4, fig.height = 3)

simpleListener <-
  function(utterance,
           mapUttToObjProbs,
           listenerObjectPreferences) {
    objPosterior <-
      mapUttToObjProbs[utterance, ] * (listenerObjectPreferences + 1e-100)
    if (sum(objPosterior) == 0) {
      return(objPosterior)
    }
    return(objPosterior / sum(objPosterior))
  }
```

The simple pragmatic speaker considers all "imaginable" (i.e. implemented) preference distributions over objects of the listener. Starting with a prior assumption over the possible listener's preferences ("preferencesPrior"). This prior has one value for each feature value possible. Only the priors for the target feature values > 0 because only they are relevant for the task. It then infers the posterior over these preferences given the listener makes a particular object choice. The priors in the beginning have a uniform distribution but with each trial the priors are the previous posterior preference presumptions. This leads to a Bayesian learning process. "utterance" is an index referring to one of the relevant utterances ("relevantUtterances) which are all present feature values in the current objects i.e. P(listener's feature value preferences | utterance, object choice by the listener, prior over preferences).

```
simplePragmaticSpeaker <-
  function(utterance,
           obj,
           preferencesPriorAll,
           relevantUtterances,
```

```
          currentObjects,
          mapUttToObjProbs,
          objectPreferenceSoftPriors) {
    preferencesPrior <- preferencesPriorAll[relevantUtterances]
    prefPost <- rep(0, length(relevantUtterances) + 1)
    for (pref in c(1:length(preferencesPrior))) {
      # prior over the preferences the speaker is interested in
      if (preferencesPrior[pref] > 0) {
        pp <-
          simpleListener(utterance,
                         mapUttToObjProbs,
                         objectPreferenceSoftPriors[[pref]])
        prefPost[pref] <- pp[obj] * preferencesPrior[pref]
      }
    }
    for (pos in c(1:length(relevantUtterances))) {
      preferencesPriorAll[relevantUtterances[pos]] <- prefPost[pos]
    }
    if (sum(preferencesPriorAll) == 0) {
      # no evidence for any preferences... -> no inference
      return(preferencesPriorAll)
    }
    return(preferencesPriorAll / sum(preferencesPriorAll))
  }
```

To know how well the model did, the evaluation number similar to the one used in the experiment is calculated. The best is 3 and the worst is 0. This number is calculated by comparing the simulated preferences the choosing listener has with the model or human predictions. To not compare absolute or relative numbers, as they might vary in quite big ranges, the hierarchy is compared. This results in having truthfully predicted which preferences the listener has (evaluation number = 3) in contrast to not having a clue about them (evaluation number = 0).

```
evaluate <-
  function(allUtterancePref,
           preferencesPrior,
           targetFeature) {
    index <- targetFeature * 3
    indices <- c(index - 2, index - 1, index)
    tarFeaPref <- allUtterancePref[indices,]
    if (length(preferencesPrior) > 3) {
      tarFeaPrefPrior <- preferencesPrior[indices]
    } else {
      tarFeaPrefPrior <- preferencesPrior
    }
    prefRank <-
      order(as.numeric(tarFeaPref[, 3]))
    prefPriorRank <-
      order(tarFeaPrefPrior)
    if (identical(prefRank, prefPriorRank)) {
      evalNum <- 3
    } else if (identical(prefPriorRank, c(prefRank[1], prefRank[3], prefRank[2])) ||
               identical(prefPriorRank, c(prefRank[2], prefRank[1], prefRank[3]))) {
      evalNum <- 2
    } else if (identical(prefPriorRank, c(prefRank[2], prefRank[3], prefRank[1])) ||
               identical(prefPriorRank, c(prefRank[3], prefRank[1], prefRank[2]))) {
      evalNum <- 1
    } else if (identical(prefPriorRank, c(prefRank[3], prefRank[2], prefRank[1]))) {
      evalNum <- 0
```

```
    }
    return(evalNum)
  }
```

Simulation

The model has some determining factors for how it performs. These are the not-obey-instant and the soft-preferences-value. The not-obey-instant is a value between 0 and 1, with 0 the listener follows always it's preferences and 1 not at all. The soft-preferences-value manipulates how strong the listener's preferences are. With the value being 0 the listener has absolute preferences and absolute non-preferences. Augmenting this value leads to a uniform-prior model.

```
notObeyInst <- 0
softPrefValue <- 1
```

Here the model gets tested with the data from the experiment. It allows direct comparison of the performance of the model with the human behavior. The data to feed the model is the data which was also fed to the participants in the experiment. For each combination of objects, utterance and previous trials the model adjusts its posterior presumptions which then become the next prior presumptions.

```
for (worker in c(0:totalWorker)) {
  for (block in c(1:totalBlock)) {
    blockdata <-
      subset(inputData,
             blockNr == block - 1 &
               workerid == unique(inputData$workerid)[worker + 1])
    targetFeatureNum <- blockdata$targetFeatureNum[1]
    preferencesPrior <- getPreferencesPrior(targetFeatureNum)
    preferencesPriorIndices <- which(preferencesPrior != 0)
    allUtterancePref <-
      getAllUtterancePref(
        c(
          blockdata$simPreference0[1],
          blockdata$simPreference1[1],
          blockdata$simPreference2[1]
        )
      )
    ambiguousUtteranceCount <- 0
    for (trial in c(1:maxTrialNum)) {
      row <- row + 1
      currentObjects <-
        c(blockdata$orderObjNum1[trial],
          blockdata$orderObjNum2[trial],
          blockdata$orderObjNum3[trial])
      allPresentFeaValues <- determineAllFeaValues(currentObjects)
      inputData$allPresentFeaValues[row] <-
        toString(allPresentFeaValues)
      relevantUtterances <- determineValidUtterances(currentObjects)
      utteranceGeneral <- as.integer(blockdata$utteranceNum[trial])
      utterance <- which(relevantUtterances == utteranceGeneral)
      ambiguous <- isAmbiguous(allPresentFeaValues,
                               utteranceGeneral,
                               currentObjects,
                               targetFeatureNum)
      inputData$ambiguous[row] <-
        ambiguous
```

```
      ambigRatio <- countAmbigUttRatio(allPresentFeaValues,
                                       currentObjects,
                                       targetFeatureNum)
      inputData$ambigRatio[row] <- ambigRatio
      if (ambiguous) {
        ambiguousUtteranceCount <- ambiguousUtteranceCount + 1
      }
      inputData$ambiguousUtteranceCount[row] <-
        ambiguousUtteranceCount
      mapObjToUtt <-
        determineObjectToUtterancesMapping(currentObjects)
      mapUttToObjProbs <-
        determineUtteranceToObjectProbabilities(relevantUtterances,
                                                currentObjects,
                                                mapObjToUtt,
                                                notObeyInst)
      mapUttToPref <-
        getMapUttToPref(relevantUtterances, allObjects, allUtterancePref)
      objectPreferenceSoftPriors <-
        getObjectPreferencePriors(
          relevantUtterances,
          currentObjects,
          softPrefValue,
          mapUttToObjProbs,
          mapUttToPref
        )
      mapUttToObjToPref <-
        getMapUttToObjToPref(
          currentObjects,
          targetFeatureNum,
          relevantUtterances,
          allUtterancePref,
          allObjects,
          mapUttToPref
        )
      obj <-
        which(currentObjects == blockdata$simulatedAnswerObjNum[trial])
      preferencesPrior <-
        simplePragmaticSpeaker(
          utterance,
          obj,
          preferencesPrior,
          relevantUtterances,
          currentObjects,
          mapUttToObjProbs,
          objectPreferenceSoftPriors
        )
      inputData$preferencesPrior1[row] <-
        preferencesPrior[preferencesPriorIndices[1]]
      inputData$preferencesPrior2[row] <-
        preferencesPrior[preferencesPriorIndices[2]]
      inputData$preferencesPrior3[row] <-
        preferencesPrior[preferencesPriorIndices[3]]
      evalNumModel <-
        evaluate(allUtterancePref, preferencesPrior, targetFeatureNum)
      inputData$evalNumModel[row] <- evalNumModel
      humanResponse <-
```

```
        c(
          blockdata$normResponse0[trial],
          blockdata$normResponse1[trial],
          blockdata$normResponse2[trial]
        )
      evalNum <-
        evaluate(allUtterancePref, humanResponse, targetFeatureNum)
      inputData$evalNum[row] <- evalNum
    }
  }
}
```

For each trial the utterance chosen by the participant is evaluated if it is ambiguous for the target feature or not. This shows which participants use ambiguity to detect information.

```
inputData$ambiguousUtteranceCount <- as.factor(inputData$ambiguousUtteranceCount)
ambiguityUsed <- matrix(nrow = totalWorker + 1, ncol = 3)
for (worker in c(0:totalWorker)) {
  ambiguityUsed[worker + 1, 1] <-

unique(inputData$workerid)[worker + 1]
  ambiguityUsed[worker + 1, 2] <-
    round(sum(inputData$ambiguous[which(inputData$workerid ==

unique(inputData$workerid)[worker + 1])]) / 16 * 100, digits = 1)
  ambiguityUsed[worker + 1, 3] <-
    sum(inputData$ambiguous[which(inputData$workerid == unique(inputData$workerid)[worker + 1])])
}
ambiguousWorker <-
  subset(ambiguityUsed, ambiguityUsed[, 2] > quantile(ambiguityUsed[, 2], 0.75))[, 1]
inputDataAmbiguous <-
  subset(inputData, workerid %in% ambiguousWorker)
nonAmbiguousWorker <-
  subset(ambiguityUsed, ambiguityUsed[, 2] < quantile(ambiguityUsed[, 2], 0.25))[, 1]
inputDataNonAmbiguous <-
  subset(inputData, workerid %in% nonAmbiguousWorker)
```

The experimental human data was cleaned for language and assurance that the task was done accurately. Only participants which specified that their native language was English were kept. Two participants with different native languages (Italian and Urdu) were excluded. This happened to avoid problems and hassles related to language barriers. Also, as the study scrutinized a psycho-linguistic phenomenon, possible interference with other native languages as English were tried to minimize. Three participants who answered to the question “Did you read the instructions and do you think you did the HIT correctly? - Yes, No or Confused” with “No” or “Confused” were excluded too. In these cases the data cannot be considered in the evaluation. (HIT is an abbreviation for Human Intelligence Task and is to be used here synonymously with “experiment”.)

Thanks for reading until here. Have a good day!