Internship Report on Predicting Listener Backchannels

Iwan de Kok

June 20, 2008

Contents

1 Introduction

In this report I will document the work I have done during my internship at Institute for Creative Technologies from 22 January to 25 April under supervision of Louis-Phillipe Morency. During this time I have done research in the field of virtual humans, more specifically in the field of predicting and producing listener backchannels. But more on that later.

I will start this report with a little background about the Institute for Creative Technologies and the project group which I was part of. After this the goal of my internship will be explained in Section 2. A general overview of our approach of achieving the goals set in Section 2 will be explained in Section 3. A more detailed description of the different steps taken will be given in Section 4. Following on that the results of the conducted research will be presented in Section 5. Finally a discussion of the work done, recommendations for improvement and future work will be given in Section 6.

1.1 Institute for Creative Technologies

In 1999 the Institute for Creative Technologies (ICT) was established as part of the University of Southern California [1]. The institute is funded by the US Army to explore the possibilities of artificial intelligence, graphics and immersion if applied to the field of learning through interactive media. This research is done in collaboration with talent from Hollywood and the game industry.

As said before their main goal is to apply interactive media to the field of learning and training experiences. This is mostly done by designing interactive environment in which the trainee can interact with a system as though it was real life. Most of the previous approaches to these kind of environments were focused on drills and mechanics. Opposed to this approach, ICT aims to enhance the human interactions and emotions of these systems. These qualities are proven to have a deep impact on the learning of critical thinking and decision-making skills.

1.2 Rapport Project

When you are designing a virtual human, not only his appearance is important, also his behavior. If it behaves unnatural people immediately recognize this and can become confused, which is not helpful when you implement virtual humans in a learning environment.

One of the behavior patterns which occurs during natural conversation is rapport. Rapport is the feeling of being on the same wavelength as the

person to whom you are talking. The conversation is going smoothly and you understand each other. The Rapport Project is trying to create this feeling between a virtual human and an actual human being. One of the main factors which contribute to this feeling is the feedback given during the conversation. This feedback can be through visual observations as gestures, eye gaze and facial expression as well as through speech. Therefor the project tries to analyze all of these observations and derive a behavioral pattern from it.

2 Goal

During autumn two French interns started working on a toolbox which is able to analyze all the observations which could have an impact on the feeling of rapport [2]. More specifically it tries to predict backchannels of a listener based on the observations from the speaker. They did this under supervision of Louis-Phillipe Morency. The toolbox analyzes data from a previously conducted user study and through machine learning it finds a model which predicts the backchannels.

My goal was to improve this toolbox in several ways. First of all there was more data collected through the same user study which needed to be prepared for use with the toolbox. Furthermore there were new analyzing tools available which provided us with new observations which may have an impact on backchannels, like eye gaze and automatic sound processing. This new information needed to be prepared for use with the toolbox as well.

Since the ultimate goal of Louis-Phillipe Morency for this part of the Rapport Project is to release the toolbox for general use in research, the documentation and the structure needed to be improved as well. The toolbox needed to be partially redesigned and several improvements in speed and efficiency are needed.

Finally the evaluation of the results of the toolbox needed to be improved. The performance measure which was implemented was not clear on the performance of this approach opposed to previous approaches to the problem.

If all these goals are achieved a publication about the toolbox would be the final goal of my internship at ICT.

3 General Overview

The main goal of this project is to learn through machine learning a model which can be used to generate listener feedback, based on observations of the speaker. In this section the way such a model can be used will be explained as well as a general description of the process needed to obtain such a model.

Just like real humans, virtual humans are provided with senses through advancements in the fields of audio and video analysis. A lot of behavior is determined by the things that happen around us and senses are the means which provide us with this information. What the exact relations are between these observations and the displayed behavior is a complex problem. Especially if subtle differences have a influence. Humans are trained through experience to pick up on those subtle signals and display the appropiate behavior automatically and without much thought. It makes you wonder whether a computer can do the same.

One way this can be achieved is by using sequential probabilistic models, like Conditional Random Fields or Hidden Markov Models, to model behavior. These kind of models take a sequence of observations as input and return, based on previously learned rules, a sequence of probabilities. In our case these probabilities will represent the probability that at that specific moment in time the listener would produce a backchannel, based on the rules it learned from previously seen real life examples and the sequence of observations it was provided.

To obtain such a model the following steps need to be taken.

- **Data Collection**: Through a user study collect real life examples of conversations. From these conversation we are going to learn the patterns which are hidden behind the behavior of the listener. This step is explained in detail in Section 4.1.

- **Feature Extraction**: Extract from the real life conversations the interesting information. We have recorded audio and video recordings from the conversations. In this step we collect all the potentially relevant information we can get from these recordings, such as which words were spoken, in which way were they spoken and where was the speaker looking while he spoke them. This step is explained in detail in Section 4.2.

- **Feature Encoding**: Represent the collected information in a suitable way. All the features we have collected from the recordings in the previous step may have a different kind of effect on listening behavior. It is hard for a sequential model to learn these different effects with a

limited data set. Therefor we represent each feature in different ways to model the different kind of effects a feature may have. This step is explained in detail in Section 4.4.1.

- **Feature Selection**: Select from all the information available the most useful. Providing a sequential model with all the information available will not produce the best results and consumes too much time. By doing a selection in advance we can eliminate less useful features such as specific words and speed up the learning process. This step is explained in detail in Section 4.6.

- **Training of the Sequential Model**: Train a sequential model with this information. With the limited number of only potentially relevant features we going to train our sequential model. This step is explained in detail in Section 4.5.

- **Performance Measure**: Evaluate the trained sequential model by measuring its performance. Finally we need to evaluate the model to see whether the results we get are good or not. The way this is done is explained in detail in Section 4.7.

4 Detailed Description

To be able to do all the steps for learning the patterns hidden behind the behavior of the listener as discussed in Section 3 we developed a MATLAB toolbox. This toolbox provides all the functionality you need to analyze real life examples of human behavior and extract through machine learning the hidden patterns behind them. In this section we will go through the steps in more detail and explain through a case study of learning listener backchannels the way the toolbox can be used and what factors need attention when attacking a similar problem.

4.1 User Study

An important factor in trying to learn human behavior from real-life examples is a well constructed user study. If your original data does not represent real life you can not expect the toolbox to learn anything which is applicable in real-life. Think carefully what you want to learn and which data you may need to do this.

Data for our case study is drawn from a study of face-to-face narrative discourse ('quasi-monologic' storytelling). 104 subjects (67 women, 37 men)

Figure 1: Setup for training and evaluation corpus. This study of face-to-face narrative discourse ('quasi-monologic' storytelling) included 104 subjects. The speaker was instructed to retell the stories portrayed in two video clips to the listener.

from the general Los Angeles area participated in this study. They were recruited using Craigslist.com and were compensated $20 for one hour of their participation. From the 52 sessions recorded, 1 was excluded from our data set because of a missing video recording and another one was missing an audio recording, making the total number of sessions used 50.

Participants in groups of two entered the laboratory and were told they were participating in a study to evaluate a communicative technology. The experimenter informed participants: *"The study we are doing here today is to evaluate a communicative technology that is developed here. An example of the communicative technology is a web-camera used to chat with your friends and family."*

Participants completed a consent form and a pre-experiment questionnaire eliciting demographic and dispositional information. Subjects were randomly assigned the role of speaker and listener. The speaker remained in the computer room while the listener was led to a separate side room to wait. The speaker then viewed a short segment of a video clip taken from the Edge Training Systems, Inc. Sexual Harassment Awareness video. Two video clips were selected and were merged into one video: The first, "Cyber-Stalker," is about a woman at work who receives unwanted instant messages from a colleague at work, and the second, "That's an Order!", is about a man at work who is confronted by a female business associate, who asks him for a foot massage in return for her business.

After the speaker finished viewing the video, the listener was led back into the computer room, where the speaker was instructed to retell the stories

portrayed in the clips to the listener. Elicited stories were approximately two minutes in length on average. Speakers sat approximately 8 feet apart from the listener. Finally, the experimenter led the speaker to a separate side room. The speaker completed a post-questionnaire assessing their impressions of the interaction while the listener remained in the room and spoke to the camera what s/he had been told by the speaker. Participants were debriefed individually and dismissed.

We collected synchronized multimodal data from each participant including voice and upper-body movements. Both the speaker and listener wore a lightweight headset with microphone. Three Panasonic PV-GS180 camcorders were used to videotape the experiment: one was placed in front the speaker, one in front of the listener, and one was attached to the ceiling to record both speaker and listener.

4.2 Feature Extraction

After recording all the signals you need, it is time to extract the features from these signals you want to analyze. Providing the toolbox with just the raw signals is not likely to give you good results. Some processing of these signals is needed. This can either be done automatically using a toolbox which can extract the features you need or done by hand, having coders annotate different aspects of the signals. In our case study we used both approaches. All the features we used and how we collected them is discussed below. A feature is the binary representation of a specific event with a start and end time. Having them in an unified representation makes the addition of new features an easy step.

4.2.1 Automatic Prosodic Features

To extract the pitch and intensity from the speech signal from the speaker audio recordings we had two toolboxes available, Aizula and LAUN. Aizula is the toolbox originally designed by Ward and Tsukahara for their hand-crafted rule based approach to detecting backchannels [3]. LAUN is a reimplementation of that code developed by Lamothe and Morales [4]. Both toolboxes also provide several acoustic features derived from the raw pitch and intensity. After analysis of the output of both toolboxes we decided to use the Aizula toolbox, since this provided the most reliable results.

The features the toolbox extracted were:

- Downslopes in pitch
- Regions of low pitch
- Utterances
- Fast drop or rise in intensity of speech
- Drop or rise in intensity of speech
- Softly spoken words

Basically all of these features were thresholded versions of the raw pitch and/or energy signals. By applying threshold we get binary signals indicating whether the feature was happening at that specific time or not.

For our features the change in pitch was considered a downslope when it dropped for at least 0.015 for at least 40 milliseconds. The pitch was considered low when it was lower than the 26th percentile for at least 110 milliseconds. Utterances indicates whether someone is speaking for at least 700 milliseconds at that time. The following two features indicate a sudden drop in intensity or a more gradual drop. The function of these features is to be an automatic detection of pauses in the speech signal. Both represent the same thing, but the first one uses a more discriminative threshold than the other. The final features indicates whether a word is spoken softer than 80% of the average volume.

4.2.2 Transcriptions

Besides having the speech signal automatically analyzed, we also had human coders annotate the narratives with several relevant features. All elicited narratives were transcribed, including pauses, filled pauses (e.g. "um"), incomplete and prolonged words. These transcriptions were double-checked by a second transcriber. This provided us with the following extra lexical and prosodic features:

- All individual words (i.e., unigrams)
- Pause in speech (i.e., no speech)
- Filled pause (e.g. "um")
- Lengthening of words (e.g., "I li::ke it")

- Emphasized or slowly uttered words (e.g., "ex_a_c_tly")
- Incomplete words (e.g., "jona-")
- Words spoken with continuing intonation
- Words spoken with falling intonation (e.g., end of an utterance)
- Words spoken with rising intonation (i.e., question mark)

Note that some of them provide the same information as some of the features which we extracted with Aizula, for instance *Pause in speech* and *Fast drop or rise in intensity of speech*. It never hurts to have more, slightly different versions of the same information. The toolbox is able to select the version that works best for the specific task, in our chase the prediction of backchannels.

4.2.3 Annotations

From the speaker video the eye gaze of the speaker was annotated on whether he/she was looking at the listener. After a test on five sessions we decided not to have a second annotator go through all the sessions, since the differences in annotations were insignificant. The feature we obtained from these annotations is:

- Speaker looking at the listener

Now we have all the features we want to use for the prediction of backchannels, but the most important information is still missing, namely our ground truth on which we train and evaluate our system. In our case we want to prediction backchannels. Since the listeners in our user study were instructed not to speak, they only gave backchannels through head nods. So from the listener video recording these head nods were annotated and then double-checked by a second coder.

4.3 Data Importation

Before being able to use the features described in Section 4.2 they will have to be imported into MATLAB and formatted to the same format. The format which we use internally for all the features is displayed in Figure 2.

`Action` is a cell of matrices. For each session of the user study or each instance of data you have there is a column in `Action`. For each feature there is a row. The row corresponds to the row in `Caption` which contains

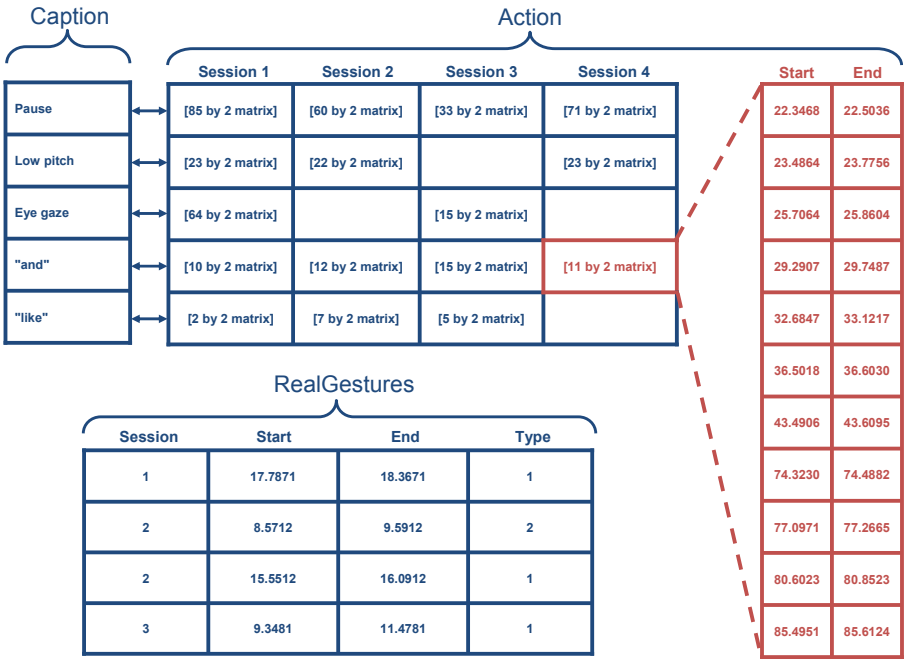

Figure 2: `Action` is a cell of matrices. For each session of the user study or each instance of data you have there is a column in `Action`. For each feature there is a row. The row of corresponds to the row in 'Caption' which contains the name of this feature. In each cell in `Action` there is a matrix containing the start and end times in seconds for each instance of the feature that occurred during the session.

the name of this feature. In each cell in `Action` there is a matrix containing the start and end times in seconds for each instance of the feature that occurred during the session. So for instance during session 4 the word *"and"* was spoken for the first time from 22.3468 to 22.5036 seconds and for the second time from 23.4864 to 23.7756 seconds. If a certain feature does not occur during a session the cell is empty.

In our case the annotations, collected in ELAN, can be imported with the function `readELAN`, while the transcriptions, who were collected in Transcriber, can be imported with the function `readTrans`. When you have new features in another format, all you need to do is write a function which imports your data into MATLAB in the format depicted in Figure 2.

Also depicted in Figure 2 is the format in which the gestures are stored you are trying to predict, namely in the table `RealGestures`. This cell contains for each instance the start and end time of the gesture, along with

the session in which it occurred. Finally the definition of its type is stored. For instance between 8.5712 and 9.5912 seconds in session 2 a gesture of type 2 (let's say a head shake) has occurred. A little while later between 15.5512 and 16.0912 seconds in that same session a gesture of type 1 (let's say a head nod) occurred. This way all your different gestures can be stored in one cell and will later be used as the labels you are trying to learn.

4.3.1 Timestamp Alignment

Because you are using different recordings it is important that you align them to the same timeline. Not every recording started at exactly the same time. You need to define a time 0.0 to which you align all your features to. In our case we had a loud beep before the actual conversation started. We used this beep as our time 0.0. To find the offset of each recording we looked at the the time this beep occurred. All these offsets were used in our import function to correct the times so that they are aligned to the same time line. A more detailed description of the way we aligned our data using the functions provided by the toolbox can be found in the Appendix, Section 7.1.

4.3.2 Normalizing Labels

As mentioned earlier maybe the most important thing are your ground truth labels. If they do not represent what you want to learn, the toolbox will not learn it. In our case we want to learn when to generate a backchannel. In order to do that you want to know the most likely times at which point you should start a backchannel. What this backchannel actually looks or sounds like is not within the scope of our research. Our labels should represent this.

From the annotations there is a lot of variation in the backchannel signals. Some people give short determined backchannels, while others give long extended backchannels. In our data the length of the backchannels varied from 0.16 to 7.73 seconds. We do not want this variation in our labels since it may influence the performance of our models, while most of these differences are caused by way that particular person generally produces backchannel and not by the features we use in training. Therefore we changed every label to a fixed length of 1 second, starting at the originally annotated start time.

4.4 Data Set Preparation

At this point we have imported all the features into the `Action` data structure and aligned them to the same timeline. We are ready to use them. But how? What do we want to learn from them? How could they effect our labels,

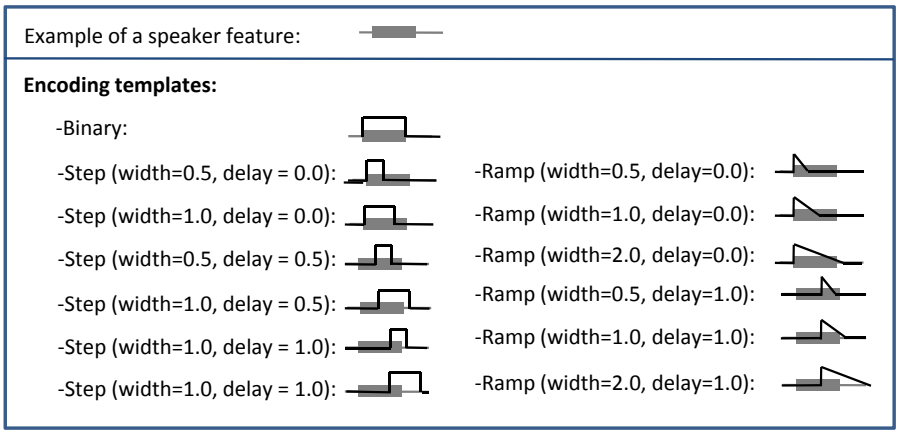

Figure 3: **Encoding dictionary.** This figure shows the different encoding templates used by our prediction model. Each encoding templates were selected to model different relationships between speaker features (e.g., a pause or an intonation change) and listener backchannels. This encoding dictionary gives a more powerful set of input features to the sequential probabilistic model used by our prediction model.

in our case backchannels. Each feature may have a different way to affect backchannels and we should somehow try to capture that. That is why we use an encoding dictionary. This encoding dictionary is explained in the following section.

4.4.1 Feature Encoding

The goal of the encoding dictionary is to propose a series of encoding templates that potentially capture the relationship between speaker features and listener backchannel. The Figure 3 shows the 13 encoding templates used in our experiments. These encoding templates were selected to represent a wide range of ways that a speaker feature can influence the listener backchannel. These encoding templates were also selected because they can easily be implemented in real-time since the only needed information is the start time of the speaker feature. Only the binary features also uses the end time. In every case, no knowledge of the future is needed.

The three main types of encoding templates we used are:

- **Binary encoding** This encoding is designed for speaker features which influence on listener backchannel is constraint to the duration of the speaker feature.
- **Step function** This encoding is a generalization of binary encoding by adding two parameters: width of the encoded feature and delay between the start of the feature and its encoded version. This encoding is useful if the feature influence on backchannel is constant but with a certain delay and duration.
- **Ramp function** This encoding linearly decreases for a set period of time (i.e., width parameter). This encoding is useful if the feature influence on backchannel is changing over time.

It is important to note that a feature can have an *individual* influence on backchannel and/or a *joint* influence. An *individual* influence means the input feature directly influences listener backchannel. For example, a long pause can by itself trigger backchannel feedback from the listener. A *joint* influence means that more than one feature is involved in triggering the feedback. For example, saying the word "and" followed by a look back the listener can trigger listener feedback. This also means that an feature may need to be encoded more than one way since it may have a *individual* influence as well as one or more *joint* influences.

The encoding of these actions is done in the function `encodeActions`. If you want to use other encodings you can add your encoding to that function.

4.5 Model Training

Independent of which type of model you use for machine learning are the steps you need to take to do proper machine learning. You cannot learn a model with all the information you have at hand and then test it on the same information. If you do so there is no way of knowing whether your learned model is applicable on other sets or if it only fits the set you have used to train it. This phenomenon is called over-fitting. It might be too specifically trained for the data you collected.

To prevent this from happening the data is usually split into different distinct sets. The largest part of your information is usually used for training of the model. For the following steps, validation and testing, a smaller set should suffice. What is done in each step is explained below.

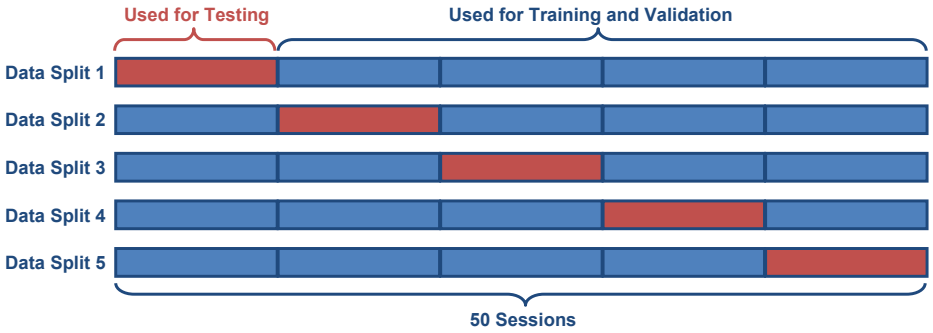

Figure 4: Our 50 sessions are split in 5 different ways. In every split 10 sessions are used for testing (red) and 40 sessions are used for training and validation (blue).

- **Training**: During training you try different settings for the unknown variable(s) in your model. For each of these settings you train a model based on the sequences in your training set.
- **Validation**: During validation you select from the different settings you have tried in the training step the one which performs best when you apply the learned model on the validation set. So during this phase you choose the settings for you model.
- **Testing**: Finally you test your model with the settings selected in the validation step on the test set to assess the performance of your learned model.

How we used these techniques in our approach is explained in the following section.

4.5.1 Data Splits

In Figure 4 the way we split our data in test data (red) and training and validation data (blue) is displayed. In every split we use 10 sessions for testing and the other 40 for training and validation. We do this 5 times so that we have test results for each of our sessions. The splits are done in such a way that in every 10 sessions of the test data the total number of backchannels in every split is about the same. This way we avoid a big difference between splits which might cause big differences in validation performance compared to test performance.

The training and validation data is also splitted into two different sets. 10 sessions are selected randomly from the set and used for validation. The

other 30 are used for training. The best way to do validation would be to use cross-validation, where you do this split 4 times so you use every sessions once for validation and the other 3 times for training. We did not do this for speed purposes, we only split it once. The toolbox does provide this functionality though. How to enable this functionality is explained in the Appendix, Section 7.2. During validation we select the best value for the unknown parameters(s) in the used model (in our case Conditional Random Fields (CRF), see Section 4.5.2). The CRF is trained with 6 different values for the regularization term and based on the validation performance the best value is selected for the model. These values were $10^k, k = -1..3$.

4.5.2 Training Models

As mentioned in the previous section we mainly used Conditional Random Fields (CRF) [5] as machine learning model. This is a discriminative probabilistic model. Being discriminative, it tries to find the best way to differentiate between cases where a backchannel is given by the listener from cases where no backchannel is given. Some of the advantages it has over other models are its speed and the applicability to our specific problem.

Besides CRF we also used Hidden Markov Models (HMM). Opposed to the discriminative strategy of CRF this model applies a generative strategy. This means that is tries the best way to generalize the moments where the listener performs a backchannel without looking at the moments where the listener does not. In Section 5 we will show that the strategy of CRF works better for our problem, which does not mean it will always be the case. For other problems the strategy of HMM may work better.

Besides these popular probabilistic models the toolbox also provides functionality for Latent-Dynamic CRF, which is a variant which tries to combine the best of both CRF and HMM.

4.5.3 Training Samples

For our training data set we do not actually use the whole sessions. The main reason for this is that the model will be biased to not give a backchannel because there are a lot more times were no backchannel is given as opposed to times were there is a backchannel. Another advantage of sampling our training data this way is that the size of our training data is reduced this way, without losing much relevant information. This speeds up the training process.

To resolve this we take samples from the original data as can be seen in Figure 5. First we select all the instances with a label (lower part of

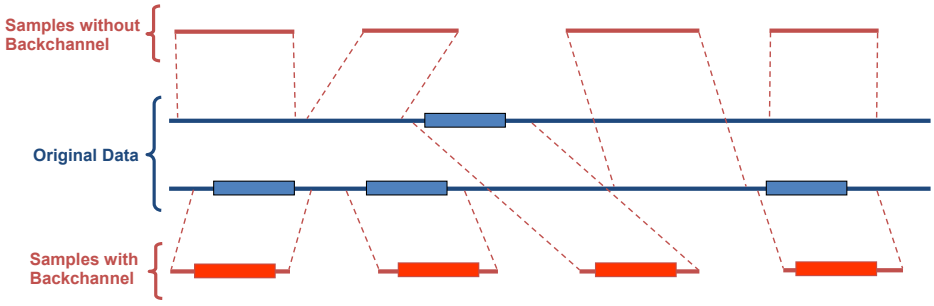

Figure 5: From the original sequences (blue) the chunks are selected which form our training data (red). We select as many training samples with a label as without. This way the model is not biased in giving a backchannel or not.

Figure 5). As you can see we do not select exactly the parts were the label is happening, but also some part around it of varying length. This way we will get the transitions, which are the most interesting parts. We vary the length because otherwise the model might learn that after a certain time it is very likely that a backchannel is happening regardless of which features there are.

Then we select just as many samples of varying length without backchannels (upper part of Figure 5). The samples are picked randomly from the original data. This way we ensure that our model is not biased in giving a backchannel or not.

4.6 Feature Selection

Dealing with machine learning speed is always an issue. If you want to train your model with all the features you have available, patience is a good quality to have. The long waits can be avoided by filtering your features before starting to use machine learning. Another reason to do this is the fact that we do not have enough data to let the algorithm itself find the relevant features. In the following section we will explain the steps we have taken to reduce our original number of features of over 8000 to a more manageable number.

4.6.1 Individual Selection

In our case we have more than 8000 features. Most of them are words which are most likely too specific for our data and can not be used in a general application of the learned model. The goal of the first step is to eliminate

those features. This is done by looking at the performance of a feature by itself, so called individual selection.

Individual feature selection is designed to do a pre-selection based on (1) the statistical co-occurence of speaker features and listener backchannel, and (2) the individual performance of each speaker feature when trained with any encoding template and evaluated on a validation set.

The first step of individual selection looks at statistics of co-occurence between backchannel instances and speaker features. The number of co-occurence is equal to the number of time a listener backchannel instance happened between the start time of the feature and up to 2 seconds after it. This threshold was selected after analysis of the average co-occurence histogram for all features. After this step the number of features is reduced to 50. The function `findTopFeatures` executes this step in our toolbox.

The second step is to look at the best performance an individual feature can reach when trained with any of the encoding templates in our dictionary. We train for each feature a sequential model with each encoding from our encoding dictionary. So if we have 50 features and 10 encodings we train 500 models. Because for each model only one feature with one encoding is used, the training of these models takes hardly any time. For each feature we select the encoding which performed best and after analyzing the performance of each of the individual models we select a subset of the 12 best performing features.

4.6.2 Iterative Selection

We now have a subset of the 12 best performing features. Simply training a model with all these features with different encodings will still not give us the best performance. We need to find the best combination of features and encodings within these 12. Trying every combination is too time consuming so we need to use a smarter strategy.

Figure 6 shows the first two iterations of our algorithm which aims to find which features best complement each other. The algorithm starts with the complete set of feature hypothesis (combination of a feature and an encoding) and an empty set of *best* features. At each iteration, the best feature hypothesis is selected and added to the best feature set. For each feature hypothesis, a sequential model is trained and evaluated using the feature hypothesis and all features previously selected in the best feature set. While the first iteration of this process is really similar to the individual selection, every iteration afterward will select a feature that best complement the current best features set. Note that during the selection process, the same feature can be selected more than once with different encodings, but it will only do so if the new

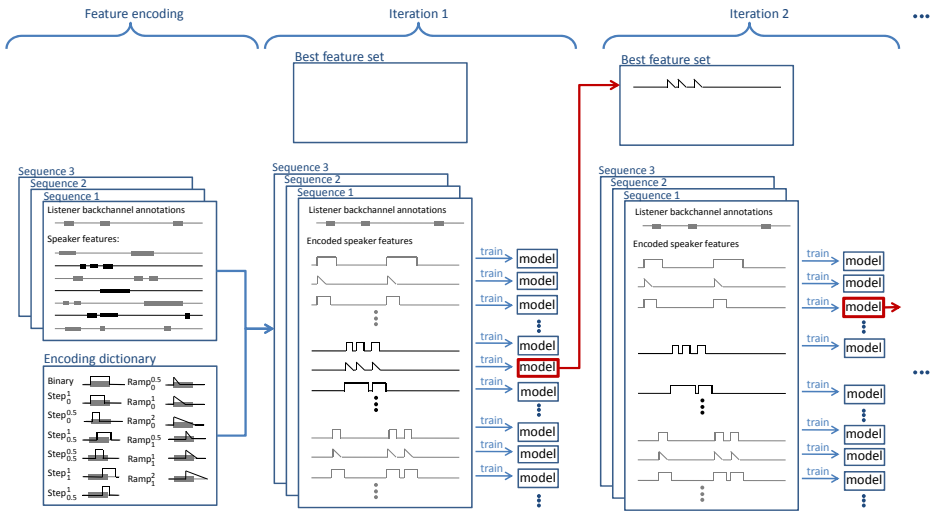

Figure 6: This figure illustrates the feature encoding process using our encoding dictionary as well as two iterations of our iterative feature selection algorithm. The goal of iterative feature selection is to find a subset of features that best complement each other for prediction of listener backchannel.

encoding actually complements the previous features. The procedure can be stopped when the validation performance starts decreasing.

4.7 Performance Measure

In the previous sections the term performance occurred several times. But how do you actually measure this? What is the best strategy or measurement for the performance of your model. In this section some of the different approaches which are implemented in the toolbox will be discussed.

The output of a probabilistic model will typically look like Figure 7. Over the course of time the model will produce a probability indicating a listener should produce a backchannel or not, based on the input features from the speaker. So how do you translate this into predictions?

4.7.1 Frame Based Predictions

One way to make the predictions is to threshold the curve and produce backchannels during the time the probability is above the threshold. This may result in unnaturally long backchannels. For instance if we would apply this technique to the probability curve of Figure 7 with the threshold set at

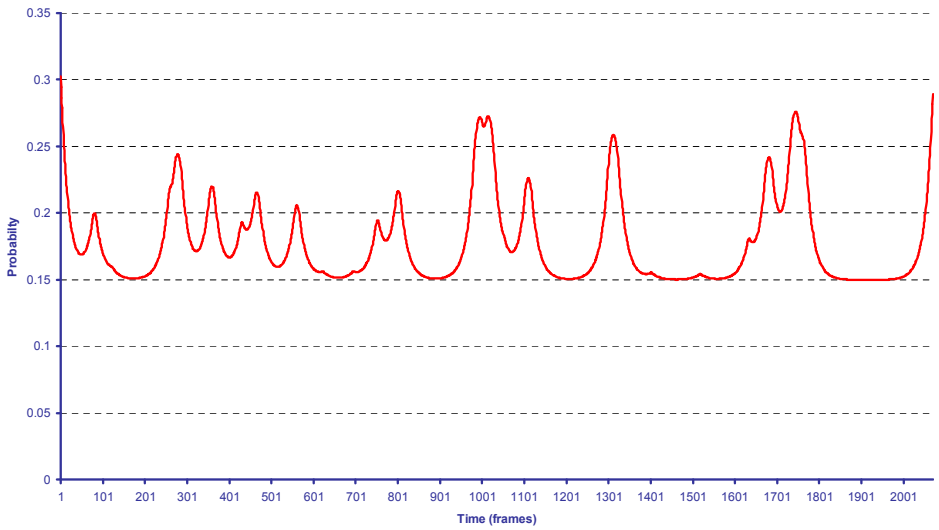

Figure 7: This figure illustrates the output of a probabilistic model. At each time frame (the sampling rate is 30 Hz) there is a probability indicating whether a listener backchannel should occur or not.

0.2, we would have a backchannel from approximately frame 1675 to frame 1800. With a sampling rate of 30Hz, this means approximately 4 seconds. Furthermore the width of is not that significant for our study. We are mostly interested in the start point of the gesture, although for other applications the width may make more sense.

For this approach we calculate the error rates by comparing each frame of the ground truth to the predictions.

4.7.2 Peak Based Predictions

Another way to make the prediction is to look at the peaks in the curve. Especially the highest peaks are the most opportune moments to produce a backchannel according to our model. So we can make our predictions from the curve by finding all the peaks and than selecting only the peaks that exceed our threshold as our predictions.

Since we only have one frame for each of our predictions we calculate the error rates a little bit differently. We check whether the prediction made by our model is during an actual backchannel. Since we have normalized our gestures to one second, we use this value as the margin of error. The toolbox provides functionality to widen or to narrow this margin though.

$$F_1 = \frac{2 \cdot precision \cdot recall}{precision + recall} \tag{1}$$

$$Recall = \frac{truepositives}{truepositives + falsenegatives} \tag{2}$$

$$Precision = \frac{truepositives}{truepositives + falsepositives} \tag{3}$$

4.7.3 Expressiveness Level

In both the frame based and the peak based prediction a threshold is used. This threshold can be seen as an expressiveness level. The lower your threshold, the more backchannels will be produced and thus the more expressive your listener is. This is one of the advantages of using a probabilistic model for modeling this type of behavior as opposed to deterministic decision rule based approaches used by other researchers [3, 6].

4.7.4 F-Measure

From the error rates calculated either on frame based or peak based prediction we can compute the F-measure. As can be seen in Equation 1 this is the weighted harmonic mean of precision and recall. Precision is the probability that a backchannel produced by a listener in our test set was predicted by the model, while recall is the probability that predicted backchannels correspond to actual listener behavior. We use the same weight for both precision and recall, so called F_1.

4.7.5 Gesture Prediction Error

Besides the generally used F-Measure method to measure the performance, we also came up with our own measurement. Because there is a lot of variation in the expressiveness of the recorded listener in our data set, it may not be fair to use one threshold for the whole testing phase. A different threshold for each listener, adjusted to his/her expressiveness level may give a more accurate reflection of the performance of our model.

Since we know for each sequence the number of backchannels the listener gave, we can ask our model the same number of predictions by selecting the same number of highest peaks. So if the listener produced 4 backchannels during the sequence of Figure 7, we will select the 4 peaks which exceed the 0.25 probability, and if the listener produced 11 backchannels we can select all the peaks which exceed the 0.20 probability.

Algorithm 1 Rule Based Approach of Ward and Tsukahara [3]

Upon detection of
P1: a region of pitch less than the 26th percentile pitch level and
P2: continuing for at least 100 milliseconds
P3: coming after at least 700 milliseconds of speech,
P4: providing you have not output backchannel feedback within the preceding 800 milliseconds,
P5: after 700 milliseconds wait,
you should produce backchannel feedback.

We can compare these predictions to the ground truth labels to get the percentage we agree. We then calculate the weighted mean over the length of each sequence as our final performance.

We have not yet formally evaluated this performance measure, so we use F_1 in the following result section. There are some more performance measure that have been implemented. All these performance measures can be calculated using the function `ComputeError`.

5 Results

We compared our prediction model with the rule based approach of Ward and Tsukahara [3] since this method has been employed effectively in virtual human systems and demonstrates clear subjective and behavioral improvements for human/virtual human interaction [7]. We re-implemented their rule based approach summarized in Algorithm 1. The two main features used by this approach are *low pitch regions* and *utterances* (see Section 4.2.1). We also compared our model with a "random" backchannel generator as defined in [3]: randomly generate a backchannel cue every time conditions P3, P4 and P5 are true (see Algorithm 1). The frequency of the random predictions was set to 60% which provided the best performance for this predictor, although differences were small.

Table 1 shows a comparison of our prediction model with both approaches. As can be seen, our prediction model outperforms both random and the rule based approach of Ward and Tsukahara. It is important to remember that a backchannel is correctly predicted if a detection happens during an actual listener backchannel. Our goal being to objectively evaluate the performance of our prediction model, we did not allow for an extra delay before or after the actual listener backchannel. Our error criterion does not use any extra parameter (e.g., the time window for allowing delays before and/or after

	Results			T-Test (p-value)	
	F_1	Precision	Recall	Random	Ward
Our prediction model (with feature selection)	0.2236	0.1862	0.4106	<0.0001	0.0020
Ward's rule-based approach [12]	0.1457	0.1381	0.2195	0.0571	-
Random	0.1018	0.1042	0.1250	-	-

Table 1: Comparison of our prediction model with previously published rule-based system of Ward and Tsukahara [3]. By integrating the strengths of a machine learning approach with multimodal speaker features and automatic feature selection, our prediction model shows a statistically significant improvement over the unimodal rule-based and random approaches.

the actual backchannel). This stricter criterion can explain the lower performance of Ward and Tsukahara approach in Table 1 when compared with their published results which used a time window of 500ms [3]. We performed an one-tailed t-test comparing our prediction model to both random and Ward's approach over our 50 independent sessions. Our performance is significantly higher than both random and the hand-crafted rule based approaches with p-values comfortably below 0.01. The one-tailed t-test comparison between Ward's system and random shows that that difference is only marginally significant.

Our prediction model uses two types of feature selections: individual feature selection and iterative feature selection (see Section **??** for details). It is very interesting to look at the features and encoding selected after both processes:

- *Pause* using binary encoding
- *Speaker looking at the listener* using ramp encoding with a width of 2 seconds and a 1 second delay
- *'and'* using step encoding with a width 1 second and a delay of 0.5 seconds
- *Speaker looking at the listener* using binary encoding

The joint selection process stopped after 4 iterations, the optimal number of iterations on the validation set. Note that *Speaker looking at the listener* was selected twice with two different encodings. This reinforces the fact that having different encodings of the same feature reveals different information of a feature and is essential to getting high performance with this approach.

	Results			T-Test
	F_1	Precision	Recall	(p-value)
Joint and individual feature selections	0.2236	0.1862	0.4106	0.1312
Only individual features selection	0.1928	0.1407	0.5145	

Table 2: Compares the performance of our prediction model before and after joint feature selection(see Section 2). We can see that joint feature selection is an important part of our prediction model.

	Results			**T-Test**
	F_1	**Precision**	**Recall**	**(p-value)**
Multimodal Features	0.1928	0.1407	0.5145	0.1454
Unimodal Features	0.1664	0.1398	0.3941	

Table 3: Compares the performance of our prediction model with and without the visual speaker feature (i.e., speaker looking at the listener). We can see that the multimodal factor is an important part of our prediction model.

It is also interesting to see that our prediction algorithm outperform Ward and Tsukahara without using their feature corresponding of low pitch.

In Table 2 we show that the addition joint feature selection improved performance over individual feature selection alone. In the second case the sequential model was trained with all the 12 features returned by the individual selection algorithm and every encoding templates from our dictionary. These speaker features were: pauses, energy fast edges, lowness, speaker looking at listener, "and", vowel volume, energy edge, utterances, downslope, "like", falling intonations, rising intonations.

In Table 3 the importance of multimodality is showed. Both of these models were trained with the same 12 features described earlier, except that the unimodal model did not include the *Speaker looking at the listener* feature. Even though we only added one visual feature between the two models, the performance of our prediction model increased by approximately 3%. This result shows that multimodal speaker features is an important concept.

6 Discussion and Future Work

Even though we already got good results as shown in Section 5 there is still room for improvement in the toolbox. On the technical side there are some memory issues when trying to perform large tasks at once. The latest version of Matlab may solve this.

More interestingly are the improvements which can be made to the algorithms. The automatic iterative feature selection as discussed in Section 4.6.2 works well, but the search is very linear at the moment. It is greedy in the way that it only selects the best in each iteration and only explores that path, although other features also increase the performance. One could imagine that selecting the second best will provide a better solution in the long run. A tree based search algorithm could provide this functionality.

Another point which may increase the performance is the selection of the samples as discussed in Section 4.5.3. At this point the selection of samples without backchannels is done at random over all the sequences. Since people are different in their listening behavior one can not know for sure that a sample without a backchannel is a bad time to provide a backchannel. This particular listener did not nod his head, but another person might have done it. To be more certain that your samples without backchannel are valid, selecting more of them from sequences where the listener provided a lot of backchannels would be a good strategy. The listener already provided a backchannel at almost every opportune moment. The instances where he did not, are probably a bad time to do it, so you should use those moments as negative samples.

By designing a new user study you can also solve the previously discussed problem. The problem with the current data set is the individual differences between persons. The amount of backchannels provided can be a difference in the behavior of the listener, but also the engagement of the speaker may have had an effect on the expressiveness of the listener. It is hard to generalize from such a diffuse data set. Also when measuring your performance you can not know for sure if a prediction made by the model is really a bad one. Maybe another person would have performed a backchannel at that moment.

The user study which may solve these problems may look like this. The same video screen set up is used as in the current study. Instead of having a different person as the speaker every time, a prerecorded video of a speaker is played. It is important that the listener believes it is not a video, but a direct live stream and the speaker in the video can see the listener. Several listeners, for instance ten, will interact with the same speaker using the same intonations and so on. By having more than one of those speaker videos you can create a large data set.

With this data set you can compare the different listeners which interacted with the same video. You can see at which point nobody provided a backchannel and maybe even more interesting the times at which everybody did. You really want your model to predict those instances correctly, so this information can also be used in evaluation stage.

With the gesture prediction error (GPE) as discussed in Section 4.7.5 we tried to simulate this. An official evaluation of this performance measure is not performed yet and we mostly used the F_1 measure at this point, but maybe GPE is a better way to measure the performance of your model with the data set currently available. Also we use the same weight for precision as for recall when calculating the F-measure. An evaluation should be made about whether precision or recall is the most important factor to optimize and adjust the weight of both factors accordingly.

Beside improvements to the toolbox there are still many options to explore available using the toolbox. We only applied it to the general case of backchannels. There are more than one type of backchannel. You have the continuation signal, but also a conformation signal. Each of those backchannels have a different application and presumably a different set of triggers. Learning a different model for each of those backchannel types would be a good next step in exploring backchannels with this toolbox.

Also the addition of more observations and encodings will hopefully increase the performance. One could think of gestures or facial expression of the speaker as new multimodal features. But the toolbox provides a lot more possibilities for other research as well.

Acknowledgement

Finally I would like to thank everyone at the Institute for Creative Technologies for giving me a great time overseas in which I have learned a lot. Especially I would like to thank Jonathan Gratch for giving me the opportunity to do my internship over there and Louis-Phillipe for the great supervision and the interesting work I was able to do. Furthermore I would like to thank Dirk Heylen for his supervision from the University of Twente.

References

[1] http://ict.usc.edu. USC Institute for Creative Technologies Website, june 2008.

[2] David Carre and Marco Levasseur. Multimodal toolbox: Analyzing gestures. Technical report, Institute for Creative Technologies, 2007.

[3] N. Ward and W. Tsukahara. Prosodic features which cue back-channel responses in english and japanese. *Journal of Pragmatics*, 23:1177–1207, 2000.

[4] Francois Lamothe and Mathieu Morales. Response behavior. Technical report, Institute for Creative Technologies, 2006.

[5] J. Lafferty, A. McCallum, and F. Pereira. Conditional random fields: probabilistic models for segmenting and labelling sequence data. In *ICML*, 2001.

[6] Ryota Nishimura, Norihide Kitaoka, and Seiichi Nakagawa. A spoken dialog system for chat-like conversations considering response timing. *Lecture notes in computer science*, 4629:599–606, 2007.

[7] Jonathan Gratch, Ning Wang, Jillian Gerten, and Edward Fast. Creating rapport with virtual agents. In *Proceedings of the 7th International Conference on Intelligent Virtual Agents*, 2007.

7 Appendix

7.1 Aligning the Recordings

The following section is a manual of how we used the `alignVideo` and `alignWav` functions of the toolbox to align the different recordings to the same timeline.

7.1.1 Available Recordings

From the face to face conversations recordings were made using a digital video camera. One is made of the speaker and one recording is made of the listener. Each sequence started recording and after a few seconds varying from 1 tot 22 a beep is heard which can be used to align them to the same timeline. The speaker is asked to put on a head set which records the speech. After approximately 2 seconds the beep sound which can also be heard in the video sequences is played.

7.1.2 Choosing Time 0

The first thing we need to do is decide which time we take as our time 0. We have different data sources and each of those has its own timeline. To align them to the same timeline we choose the start of the beep as time 0. This moment can be found in each of our data sources at our disposal.

7.1.3 Aligning Video Recordings with `alignVideo`

The first one, `alignVideo`, is designed for finding the beep in the video sequences. It automatically finds the first time the sound level reaches a threshold handed as a parameter to the function and returns this time in seconds. By default this parameter is set to 0.95 which is reached by most of the video sequences. You may want to lower this threshold if the function returns -1, which means the sound level never reached the threshold, or if the returned value is very high (above 25 seconds), meaning the threshold is reached late in the video. 0.60 seemed like a good value of the threshold parameter for our video recordings. It is also good to first play the video to get a rough estimate of which value the function should return.

7.1.4 Aligning Audio or Video Recordings with `alignWav`

The `alignVideo` function works fine for the video sequences, but didn't perform that well on the .wav files. Since the source of the sound of the beep

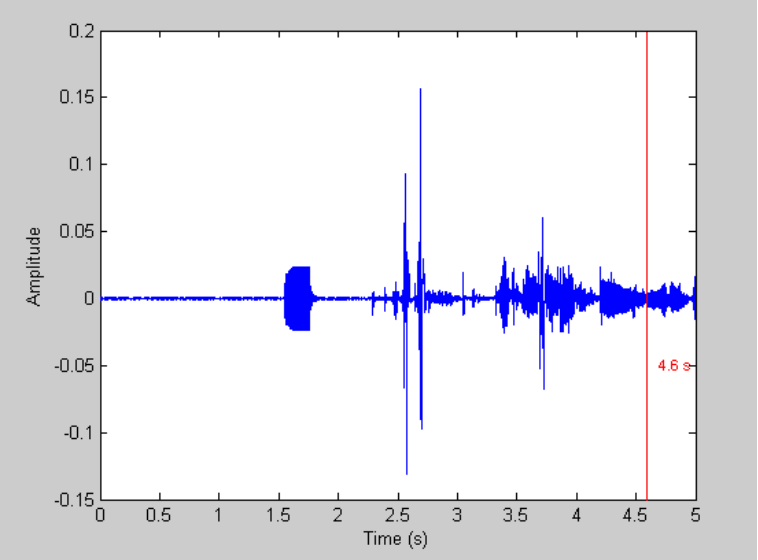

Figure 8: This is the interface of the `alignWav` function. In blue you can see the wave curve of the first 5 seconds of the audio recording. In red is the cursor which moves along the wave curve as you play the audio.

is further away from the microphone than the mouth of the speaker, the beep is usually softer than the speech. Also the sound in general is softer. You have to set the threshold to 0.01 to have a chance to detect the beep using `alignVideo`, but usually it detects some breathing happening before the beep instead. Another function was thus written to align those, called `alignWav`. This function semi-automatically detects the beep using the following approach. In blue it displays the wave curve of the first 5 seconds of the .wav file as can be seen in Figure 8. At the same time the audio is played and a cursor (in red) follows the sound across the curve. This way the user can identify the curve of the beep. The user now has three options:

- Play the same segment of the video again (hit the 'r' button on the keyboard)
- Play the next segment of the video (hit any other key on the keyboard)
- Identify the curve of the beep (by left-clicking on the figure just before the curve which represents the beep)

When the user didn't hear the exact location of the beep or has any doubts he/she can choose to replay the same segment by hitting the 'r' button on

the keyboard. This will replay the same segment along with the cursor like it did the first time.

If the user is sure the beep wasn't in the first segment it can go to the next segment to see if it is in that one. If the beep is at about the border between these segments it is advised to change the length of the window which is used for segmentation to make sure you pair the right curve to the beep. This can be done by passing this value as an input parameter.

Finally if the user has identified the curve representing the beep he/she can click just before the curve on the figure. The function will begin searching for the first time the sound level exceeds the threshold which is by default 0.01. Again this value can be tweaked by passing a different value as an input parameter. The function finally displays a green line at the time it has detected the beginning of the beep.

This function can actually be used for aligning the videos as well if the user prefers. Keep in mind though that since the audio of the videos is stereo the function runs a little slow. It can of course also be used for finding the exact timing of other sounds than the beep.

7.1.5 Storing the Offsets

The timings obtained by the two functions are collected in a matrix called `OffsetDing`. This matrix 165 rows and four columns. The 165 rows correspond to the session number of each recording session. The four columns contain the offset information needed to align the various data sources in such a way that time 0 is when the beep starts. Keep in mind that in some cases a negative number might be required if the beep isn't included in the data source, which occurs in some occasions of the headphone source. If this happens the start of speech, obtained by using `alignWav` on the headphone file and the speaker video file, can be used to calculate the offset.

In the first column the offset of the speaker video is stored. The second column contains the offset of the listener video recording and the third one has the offset of the audio recording of the speaker. The final column contains the offset for ELVIN. It is basically the type of notification is used; the ding which has no delay (earlier sessions) or the beep which has 2 seconds delay.

7.2 Variables in `paramsdata`

Within the toolbox there are a lot of variables that can be influenced. All of these variables are combined in a MATLAB struct called `paramsData`. In this section all the fields of paramsdata will be explained.

- **randSeed** Every time we use random we want to be able to repeat the process. Therefore we initialize the random generator with this value.

The following fields are settings for the creation of the data splits as discussed in Section 4.5.1 which is done in the function `createDataSplits`.

- **NFold** The number of times you want to split the data. In Figure 4 was set to 5.
- **validLabels** The labels of the ground truth feature which are valid. In our case we only had 2 labels, 0 and 1. 1 indicates a backchannel from the listener is happening, while 0 indicates no backchannel is happening. This way you can identify which labels you want to use if you have more labels, for instance a different label for the different kind of backchannels.
- **bLeaveOneOut** This number indicates the number of sequence which are used for validation as discussed in Section 4.5.1. In our case this value was set to 10.

The following fields are settings for the sampling process as discussed in Section 4.5.3 which is done in the function `chunkTrainData`.

- **rangeSizeChunks** These two values indicate the range of the length in frames the selected samples without backchannels (label 0). In our case these values were between 30 and 50 frames.
- **timeBorderGestures** These two values indicate the length in frames of the transition phase (the frames before the label becomes 1) which is added to the samples with backchannels (label 1). In our case these values were between 3 and 60 frames.
- **trainNbSeqsOnlyOneLabel** Indicates for each label how many samples are selected without a transition phase during training. In our case the values was set to [500 0], which means 500 samples with only label 0 were selected and 0 samples with only label 1 for training.

- **trainNbSeqsWithThisLabel** Indicates for each label how many samples are selected with a transition phase during training. In our case the values was set to [0 500], which means 0 samples with label 0 were selected with a transition phase and 500 samples with label 1 were selected with a transition phase for training.

- **testNbSeqsOnlyOneLabel** Indicates for each label how many samples are selected without a transition phase during testing. We did not use the sampling in the testing phase.

- **testNbSeqsWithThisLabel** Indicates for each label how many samples are selected with a transition phase during testing. We did not use the sampling in the testing phase.

- **validateNbSeqsOnlyOneLabel** Indicates for each label how many samples are selected without a transition phase during validation. We did not use the sampling in the validation phase.

- **validateNbSeqsWithThisLabel** Indicates for each label how many samples are selected with a transition phase during validation. We did not use the sampling in the validation phase.