Project work
DT2118
Speech and speaker recognition

Clémence Bisot   David Jangdal   Taylor Mordan

May 23, 2014

Abstract

Thanks to advances in speech recognition technology, integrating speech recognition supports into a lot of our everyday life devices has been made possible. One of the numerous possible applications is to use speech engines inside games. Making speech part of the game is a promising way of involving the player into a game.

The goal in this project was to build a speech recognizer to play the game "Say the Color not the Word" [1]. The purpose of this game is to speak out the color in which a word is written. To confuse the player, the written word is in fact an other color.

Furthermore, an analysis of the performances of the Sphinx speech recognizer was carried out in a couple of experiments for various setups.

The Speech engine

Which Speech Engine did we use?

There exist a lot of speech recognition engines available on the market [2]. Among all the possible ones, we especially studied the qualities and defaults of three different speech engines to choose the one to use on our application.

First we looked at ATK the real-time API for HTK [3]. This API was build to facilitate building experimental application for HTK. It consists of a C++ layer lying on the top of the standard HTK library. ATK was, like HTK, developed at the departement of engineering in the university of Cambridge. A good point of using ATK is that it enables us to easily build and train a recognizer from scratch. However, ATK had some defaults which prevented us to use HTK as a speech engine for the game. First of all the last update of ATK dates back to 2007 and there is not much documentation available on the web which let us fear some troubles. Moreover, ATK does not support Mac.

It was also conceivable to use Google's Web Speech API [4]. One of the main strengths of Google's Web Speech API is that it is very accurate and quick. However, Google Web Speech API is not open-source and only access to a demonstration version of it is possible for free. Moreover, using the Web Speech API would have forced the user to have access to internet while playing the game which is not very practical.

The solution we finally chose was to use CMU Sphinx. Sphinx is a group of open-source speech recognition systems developed at Carnegie Mellon University [5]. In particular, among Sphinx package, we used Sphinx4. Sphinx4 is an adjustable, modifiable recognizer written in Java. Since Sphinx4 is entirely developed in Java it makes it easy to link the recognizer to the rest of the game. The main quality of Sphinx4 is that it is is quite easy to use, with enough available tutorials and explanations and is very flexible. Sphinx4 enables us to define our own dictionary and our own grammar for example.

How does Sphinx4 work?

Sphinx speech recognition system has three elements: the Front End, the Knowledge base and the decoder. Front End receives and processes speech signals. Knowledge base provides data for decoder and the decoder performs the recognition itself. Figure 1 presents in a visual way the structure of Sphinx recognizer.

In the following, we now look more closely at each component of Sphinx.

The Front End: Spectral analysis and feature extraction

The Front End is responsible for processing the input signal so as to give the decoder understandable information. The waveform of the signal is split into utterances. Utterances are separated by silences. Then, sphinx uses basically frame-base processing, i.e. one utterance is regularly divided into frames of length 10ms before features are extracted from each frame. The feature vector is typically of length 39. The vector features used in Sphinx are extracted from spectral analysis of the frames. There are: the Mel Frequency Cepstrum Coefficients, their first and second derivatives with respect to time (Delta and Delta-Delta coefficients) and the power coefficient (or energy) with its derivatives. Figure 2 resumes this processing.

The knowledge base: Different models

The knowledge base contains three different models which will then be used by the decoder to match the given list of feature vectors to the most probable sentence. Different models are used depending

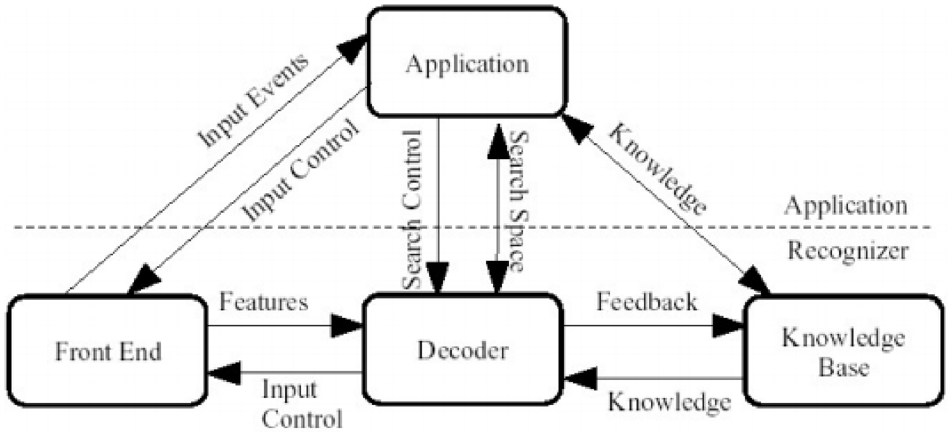

Figure 1: Sphinx Structure [6].

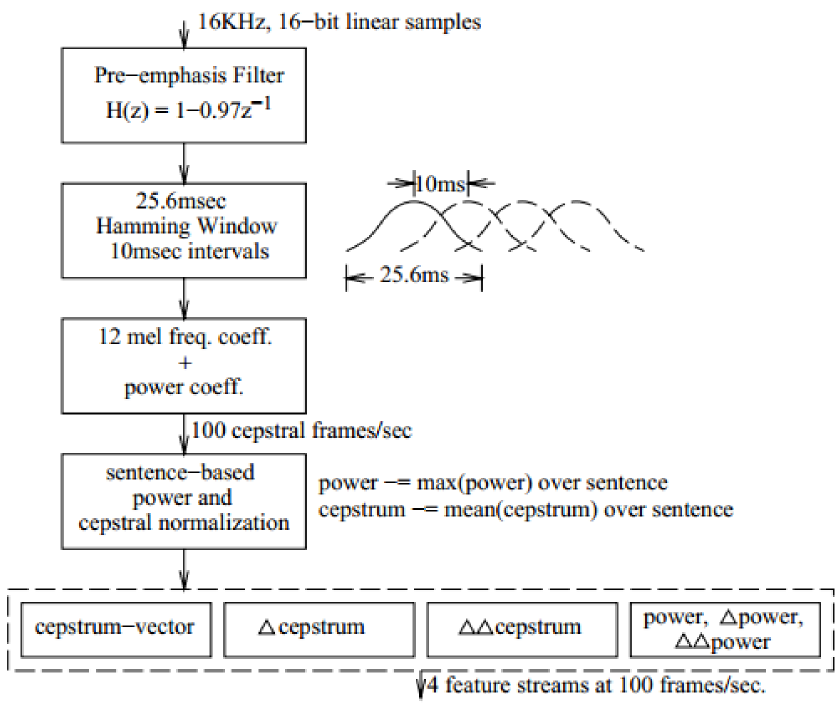

Figure 2: Signal processing in the Front End of Sphinx [7].

on the application (language, possible vocabulary...).

The acoustic model

The goal of the acoustic model is to represent the probability of a sound given a segment. In Sphinx, each phoneme is formed with five segments or states. Depending on the wideness of the vocabulary, the memory and the accuracy requirement in our application we may use either context-independent or context-dependent models. Context-dependent models aim to represent co-articulation by duplicating each phoneme model depending on its left and right contexts: triphone. Acoustic models are heavily dependent on the language and on the the way of recording. With Sphinx full acoustic models have already been trained in several languages and different recording context. It is however possible to train its own acoustic model using SphinxTrain.

Context-independent phones and triphones in the acoustic model are represented via continuous density hidden Markov models (Figure 3) where the transition probabilities in the model are approximated by Gaussian mixtures. In fact, Sphinx uses semi-continuous modeling with clustering. For fully continuous HMMs, each state in the HMM (i.e. each phone or triphone) needs its own separate weighted Gaussian mixture which is computationally and memory expensive. In Sphinx, the states are grouped into clusters called senones. Each senone is represented by a single Gaussian mixture codebook but inside the senone each state is represented by its own mixture weights.

All the parameters of the HMMs are evaluated through a modified version of the Baum-Welch algorithm.

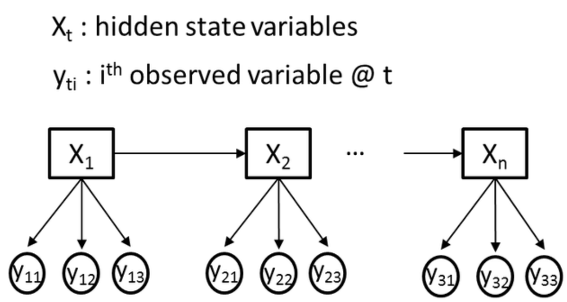

Figure 3: Hidden Markov Model.

The lexical model

The lexical model in sphinx consists in a phonetic dictionary. It contains a mapping from words to phones. This mapping is not very efficient because only two to three pronunciation variants are noted in it, but it is practical enough most of the time. The dictionary is not the only variant of mapper from words to phones. It could be done with some complex functions learned with a machine learning algorithm.

The language model

The language model or grammar enables the recognizer to choose the most likely word sequence given the sounds and the previously recognized words. The language model is key in the recognition process because it makes it possible to significantly restrict the search space. By default, Sphinx uses trigrams, i.e. one computes the probability of one word occurrence given the two previous ones and forgets about earlier words.

The decoder: Recognition itself

The decoder performs the main part of the speech recognition. It reads features from the Front End, couples this with data from the knowledge base, and performs a search to determine the most likely sequences of words that could be represented by the series of features output by the Front End.

Sphinx recognition system has a three-pass decoder structure. The first pass consist in a forward Viterbi beam search performed on the full vocabulary. The result of this search is a single recognition hypothesis and word lattice that contains all the words recognized during the search. The second pass is a time synchronous Viterbi beam search in the backward direction. This search is restricted to the words identified in the first pass and is thus very fast. The last pass is an A* or stack search using the word segmentations and scores produced by the forward and backward Viterbi passes above. This pass outputs a list of the most likely hypothesis.

Game implementation

We decided to play the game with seven different colors and to display the words one by one. The recognizer to be build is then quite easy. It has a short dictionary of only seven words and a grammar of only one word per sentence. Moreover, the probability to appear for each word is the same. Thus, there is no need of building any language model in our case.

Performance analysis

Theoretical Background

When it comes to accuracy analysis of a recognizer two classical characteristics are used: the Word Error Rate, WER, and the Accuracy, Acc. If we call N the number of words in a sentence, D the number of deletions, I the number of insertions and S the number of substitutions, those two characteristics are computed as follow:

$$WER = \frac{(I + D + S)}{N},$$
$$Acc = \frac{(N - D - S)}{N}.$$

Accuracy does not take into account the number of insertions. Therefore, it is a worse measure than the WER for most tasks, since insertions are also often important in final results. However, for some tasks, accuracy is a reasonable measure of the decoder performance.

To compute I, D and S one may use dynamic programming. The dynamic time warping algorithm that we can use to compute the performance of a recognizer goes as follows:

```
Data: Two sentences: the true one (size N) and the recognized one (size N')
Result: Distance between the sentences
for i=1 to N' do
    for j=1 to N do
        AccD[i,j] = LocD[i,j] + min(AccD[i-1,j], AccD[i-1,j-1], AccD[i,j-1])
    end
end
```

Algorithm 1: Dynamic Time Warping Algorithm.

In the algorithm, $LocD[i, j]$ represents the distance between word number i in the first sentence and word number j in the second one. Here, we take $LocD[i, j] = 0$ if the two words are the same and $LocD[i, j] = 1$ if they are different. $AccD[i, j]$ represents the shortest possible accumulated distance between the first sentence up to the i^{th} word and the second sentence up the j^{th} word.

To access the path so as to compute I, D and S one uses backtracking, i.e. one remembers the paths followed to get the minimum. Figure 4 illustrates how the overall algorithm works for comparing two words (and not two sentences). The matrix in the background is the matrix $AccD$.

One-word grammar

In the game, our grammar is very basic. It consists of sentences of only one word. The possible words are the eight colors: Black, Blue, Pink, Green, White, Orange, Yellow and Red. Our first task was to analyze the accuracy of the model for the framework of our game.

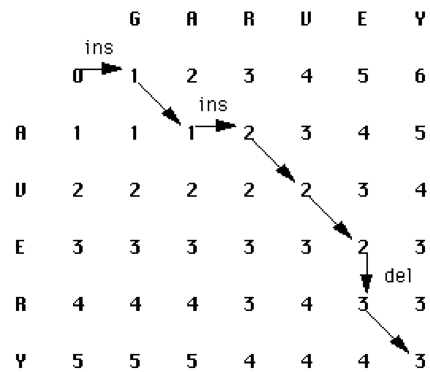

Figure 4: Dynamic Programming [8].

The results of our tests were very clear. In a quiet environment, we reached 100% of accuracy. Even the words Black and Blue which have two common phonemes were not confused. Therefore, we decided to do a more advanced performance analysis.

Loop grammar

A second accuracy analysis was performed using loop grammar. That is a less restricted grammar where a sentence can contain any number of colors. However, the dictionary was not changed. It was still composed of the eight colors: Black, Blue, Pink, Green, White, Orange, Yellow and Red. The analysis was done with 15 sentences of random lengths between three and six. The test was done on two different speakers. S_1 was a male speaker with native language Swedish and S_2 a female speaker whose native language was French.

Tables 1 and 2 gather our results. On the tables, lines represent the words really spoken (TR) and columns the recognized words (RW).

One can now compute the WER and Accuracy of the results on our experiment for each speaker and then for both together.

$$WER(S_1) = \frac{2+11+16}{67} = 0.433, \quad Acc(S_1) = \frac{67-11-16}{67} = 0.597$$

$$WER(S_2) = \frac{4+0+12}{64} = 0.250, \quad Acc(S_2) = \frac{64-0-12}{64} = 0.813$$

$$WER(tot) = \frac{2+11+16+4+0+12}{67+64} = 0.344, \quad Acc(tot) = \frac{67+64-11-16-0-12}{67+64} = 0.702$$

On the confusion matrices, it is interesting to notice that the word "Blue" is often recognized as "Pink" for both speakers. For both speakers, it is even more often confused than well recognized. Moreover, the confusion appears only in one direction: from Blue to Pink and never from Pink to Blue. This is quite surprising since those two words have no common phoneme but this error seems consistent. However, since this confusion did not appear while using a one-word grammar one may think that some co-articulation phenomenons appear here making those two words easy to confuse while spoken between two other words.

TW/RW	Black	Blue	Pink	Green	White	Orange	Yellow	Red
Black	5				2			
Blue		4	5					
Pink			11					
Green				5				
White					6			
Orange						1	4	
Yellow		2	1				4	1
Red			1					1

Table 1: Confusion matrix for S_1, $I = 2$, $D = 11$, $S = 16$.

TW/RW	Black	Blue	Pink	Green	White	Orange	Yellow	Red
Black	11				1			
Blue		3	4	1				
Pink			8					
Green				5				
White					8			
Orange						11		
Yellow							5	
Red			3			3		1

Table 2: Confusion matrix for S_2, $I = 4$, $D = 0$, $S = 12$.

Adapting the model

We are currently using the default acoustic model of Sphinx4 and it gives perfect results when the experimental conditions are optimal (silent environment, one-word grammar, close microphone...). However, when some of those requirements are not fulfilled anymore, performances can dramatically decrease, as observed above. A solution to this issue is to adapt the model in order to improve speech recognition in the considered configuration. Adaptation can be done with respect to recording environment to yield robustness to noise, to players' accents or to types of microphones for instance.

Adapting the acoustic model only requires a small amount of transcribed data: a list of sentences along with a dictionary describing the pronunciations of the words, and a record of these sentences in the targeted situations. Once the data are gathered, one can use SphinxTrain to adapt the existing model. Two methods are available for the updating step: Maximum *A Posteriori* (MAP) and Maximum Likelihood Linear Regression (MLLR).

This further step can greatly enhance the performances of the speech recognizer, especially when a loop grammar is used or when the game is played in a noisy environment. Moreover, its cost is rather low compared to the one of training a complete model, and it does not require lots of speech data. It is therefore quite easy to perform with the tools given in Sphinx.

Conclusion

In this report the use of a speech engine and its integration within a created game have been described. The choice for the recognizer has been made in favor of Sphinx for various implementation reasons and ease of use. After a thorough study of the structure of Sphinx4, the Say the Color not the Word game has been implemented in Java, making use of the considered speech recognizer. An analysis of the recognition performances of Sphinx4 has been completed with a few different grammars (one-word and loop grammars). The simplest case yields perfect results when some environmental conditions are met. On the other hand, when some of those requirements are not fulfilled or when the loop grammar is used, the recognizer sees a significant decrease in its performances. A way to overcome this issue by adapting the acoustic model has finally been discussed in the latter part of the report.

Bibliography

[1] Game "Say the Color not the Word".
http://www.brainbashers.com/colour.asp

[2] List of Speech recognition software.
http://en.wikipedia.org/wiki/List_of_speech_recognition_software

[3] ATK.
http://htk.eng.cam.ac.uk/develop/atk.shtml

[4] Google Web Speech API demonstration
https://www.google.com/intl/en/chrome/demos/speech.html

[5] CMU Sphinx.
http://cmusphinx.sourceforge.net/

[6] Understanding the CMU sphinx Speech Recognition System (Chung Feng Liao)
http://soa.csie.org/static-resources/homework/pr/pr_final_report.pdf

[7] Efficient algorithm for speech recognition (Mosur K. Ravishankar)
http://www.cs.cmu.edu/ rkm/th/th.pdf

[8] Dynamic Programming
http://berghel.net/publications/asm/asm.php