KTH Computer Science
and Communication

Practical exercise in automatic speech recognition

Objective

The objective is to experiment with methods for training, recognition and evaluation for automatic speech recognition. The exercise will also give you some familiarity with the Hidden Markov Toolkit (HTK).

Task

The task of the exercise is to perform experiments with a small recognition application (sequence of digits). The exercise consists of the following sub-tasks:

- Define a grammar that describes the speech recognition task.
- Define a dictionary describing the pronunciation of each words in your grammar.
- Record training and test data.
- Train speaker dependent monophone HMM models with Gaussian emission distributions.
- Evaluate the recognition performance on the test data.
- Experiment with the online recogniser.
- Write a short report following the templates in `report/`.

The exercise makes use of the Hidden Markov Toolkit (HTK). HTK is a powerful tool for performing HMM-based speech recognition experiments. HTK is an important research component at the department for Speech Music and Hearing, and several Master thesis projects have been using it. The HTK package is available at all KTH Ubuntu stations, or can be download at `http://htk.eng.cam.ac.uk/`. Its manual, the HTK Book, is included in the lab package in the directory `doc` (see later).

Besides the dictionary design and the speech recording, a major part of the exercise consists of executing a number of scripts and HTK programs and reporting the results.

Prerequisites

- Basic familiarity with the Linux shell. You can find a good tutorial here: `http://linuxcommand.org/lc3_learning_the_shell.php`
- Reading this document before the lab.
- The Tutorial chapter in the HTK Book is recommended for those interested in further details.

Project directory

In order to run the experiment you need to download the `rec_lab_4digits.tgz` package from the course page in Canvas under `Assignments/Lab2: Automatic Speech Recognition` and unpack it.

When you unpack the lab package, the directory structure will look like the following:

```
rec_lab_4digits
     config/              <-- configuration files
     doc/                 <-- HTK documentation
     report/              <-- template files for lab report
     results/             <-- results from training and testing
     tools/               <-- shell scripts implementing training and evaluation
     Sp1/                 <-- data and models for speaker 1
          train_data/
          test_data/
          hmm0/           <-- HMM model files for different training stages
          hmm1/           <-- "
          ...
          hmm7/           <-- "
     Sp2/                 <-- data and models for speaker 2
          ...
     Sp3/                 <-- data and models for speaker 3
          ...
```

In the `report` directory you find templates for the lab report. Depending on your preference, you can use the LaTeX template `report.tex`, the Open Document version `report.odt` if you use Libre/OpenOffice or the MS Word file `report.docx`.

Note that all the files used in this exercise are in text format, in spite of the extensions, and can be viewed with the unix command `cat`, or with a text editor. In the example we will use the `gedit` editor, but feel free to use your favourite program (alternatives are e.g. `emacs` or `vi`).

When you finish your exercise and after filling one of the report templates you should create a file called `report.pdf` and submit it to Canvas. The Lab2 assignment only accepts group submissions. Only one group member should submit the report for the whole group. Don't forget to specify the names of all participants in the report as well.

1 Procedure

In these instructions, commands, printed in `Typewriter` font, can be copied and pasted into the command window by selecting them and clicking the mouse wheel (button 2) or simultaneously the left and right button. Note that the prompt character ">" should not be included.

A symbol like this on the side of the text reminds you that you are supposed to write something into the lab report at this point.

1.1 Preparatory

Login to the computer. Open a terminal in by searching for "terminal" in the Unity dash, or by pressing Ctrl+Alt+T. Change directory to the lab package directory. This command may vary

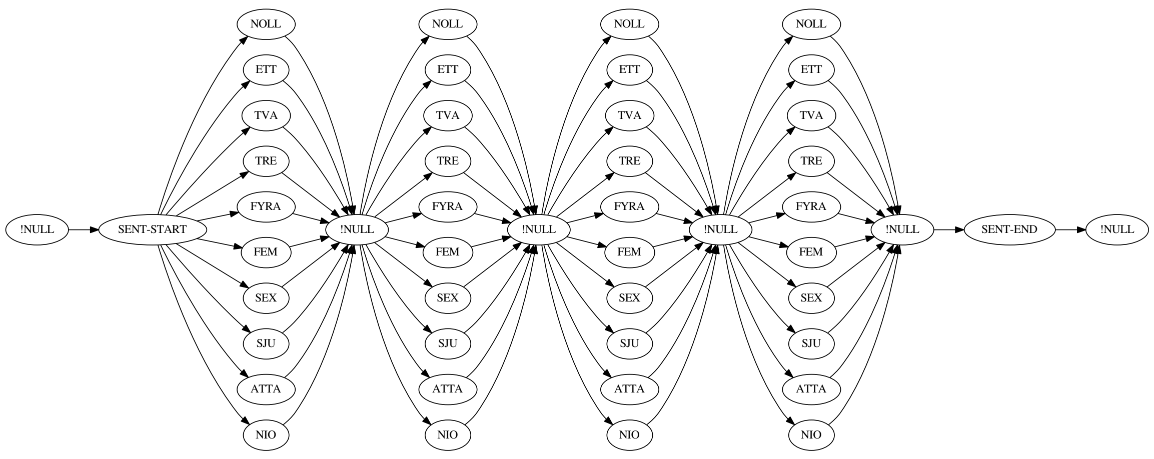


Figure 1. Four digit grammar for Swedish

depending on where you unpacked the lab package. For example, if you unpacked it in your home directory, just run:

```
> cd rec_lab_4digits
```

or if you unpacked the package in the directory `Downloads`, run:

```
> cd Downloads/rec_lab_4digits
```

On KTH Ubuntu machines run the following command to set up the environment:

```
> source config/environment
```

This will make the programs in the HTK toolkit and wavesurfer available for execution. If you interrupt the execution of the following steps and want to resume it at a later time in a new session, or if you open a new terminal, you will have to repeat the above step in order to setup the environment again.

If you are running on your own machine, go to Section A for instructions on how to install the required software.

1.2 Preparing the task grammar

Your task is to create a grammar that allows exactly four repetitions of a digit, where a digit is any of the ten digit words. A graphical representation of this grammar is given in Figure 1 for the Swedish digits. Do not worry about the NULL nodes in the graph (those are automatically added when generating the grammar).

Open the file `four_digits.grm` with

```
> gedit four_digits.grm
```

and substitute the text `fill_this` with your own definitions. Feel free to use words from your own language instead of Swedish. **Note that HTK does not handle unicode characters well. Please substitute any special characters in your words with plain ascii. For**

example write "tva" or "tvaa" instead of "två". From the HTK Book[1] (`doc/htkbook.pdf`), the grammar definition syntax is as following: A terminal symbol is simply any string not containing special characters. A string starting with `$` is a non terminal symbol. Production rules are expressed as following:

`|`	denotes alternatives
`[ ]`	encloses optional items
`{ }`	denotes zero or more repetitions
`< >`	denotes one or more repetitions
`<< >>`	denotes context-sensitive loop
`( )`	denotes beginning and end of a sentence

In order to check how your definition looks like, first generate a word graph in HTK lattice format `four_digits.lat`[2] from the grammar with the command:

```
> HParse four_digits.grm four_digits.lat
```

Then use the following commands to visualise the grammar as a graph in PDF format:

```
> tools/lat2pdf four_digits.lat
> evince four_digits.pdf
```

Check that the grammar graph you obtain is equivalent to the one in Figure 1

Explain the following things in the report:

- how did you define the rules in `four_digits.grm` in order to generate the graph defined in `four_digits.lat` and illustrated in `four_digits.pdf`?
- give examples of the word sequences that are allowed by the grammar.

Now extract a word list file `words.lst` from the grammar file with the following command:

```
> tools/gram2wlist.pl four_digits.grm > words.lst
```

Check that the file `words.lst` contains all and only the terminal symbols contained in the grammar.

1.3 Preparing the dictionary

Create a pronunciation dictionary for the words in `words.lst`: first copy the list with

```
> cat words.lst | grep -v SENT- > digits0.dic
```

Then edit file `digits0.dic` to insert the phonetic transcription of each word. Note that the words are ordered alphabetically and not by meaning, which is required by HTK. Refer to the Computer Readable Phonetic Alphabet (SAMPA)[3] for the language you have chosen to define the phonetic symbols. Note that HTK has limitations on the characters that are allowed in phonetic symbols (for example strings may not start with digits or contain special characters). If you encounter such symbols, modify them so that they are purely alphabetic. Typical examples are in the table below:

[1] You can find more information at the beginning of Section 12.3 in the HTK Book

[2] this file is also required in the following steps in the lab.

[3] `https://www.phon.ucl.ac.uk/home/sampa/`

SAMPA	HTK Compatible SAMPA
{	ae
2	ox
3	Eh
6	ah
9	oe
&	OE
@	eh
}	uh
?	qq

The phonetic transcription of a word is written to the right of its text representation, separated by white space. The phonemes in the transcription are separated from each other by a white space. An example dictionary could look like this, taking examples from the English SAMPA page:

```
chin tS I n
furs f Eh: z
give g I v
raise r eI z
wasp w Q s p
```

Note how in the second word we substituted the SAMPA code `3:` for the vowel with the HTK compatible `Eh:` code. If in doubt, consult Section 12.7, page 171 of the HTK Book ("Constructing a Dictionary").

Run the command (based on `HDMan` from HTK):

```
> tools/refine_dict words.lst digits0.dic digits.dic monophones1.lst | grep missing
```

This inserts start and end symbols, adds optional silence at the end of each word and stores the result in the output dictionary `digits.dic`. Check the differences between `digits0.dic` and `digits.dic`. A list of phones is also created and stored in the file `monophones1.lst`. In case the command returns messages of missing words, fix the `digits0.dic` file and run the `refine_dict` command again. Report the pronunciation of each word in the protocol.

1.4 Recording the data

This section must be executed once for each group member. The group members are identified by an index: `Sp1, Sp2, Sp3`. To simplify the description we will set the index at the beginning in the variable `n` and refer to the current speaker as `Sp$n`:

```
> n=1
```

(set n = 2 , 3 in the following repetitions)

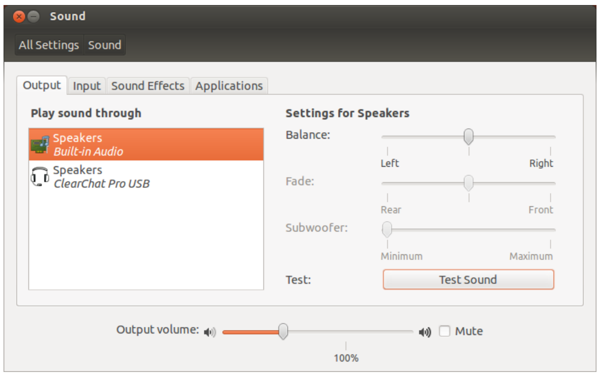
Generate 20 training and 10 test prompt sentences for recording. Each invocation produces a new random list. The command outputs a file in which each line contains a file name and a list of digits according to the grammar `four_digits.grm` (`four_digits.lat`).

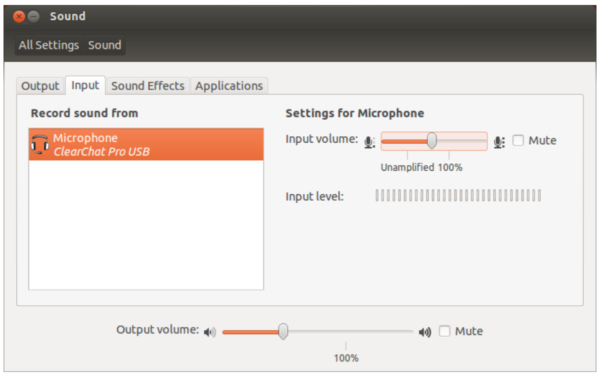
```
> tools/generate_sentences.cmd 20 > Sp$n/trainprompts
> tools/generate_sentences.cmd 10 > Sp$n/testprompts
```

Check that the prompt sentences correspond to the specification in the grammar file, that is, only contain sequences of exactly four digits.

```
> cat Sp$n/trainprompts
> cat Sp$n/testprompts
```

This is a good time to check that the headset works as it should. Your computer should have a USB headset already connected. Open the sound settings by clicking on the small loud speaker icon on the upper bar of the Unity desktop and choosing "Sound Settings…". The resulting window should look like the figures below. In the "Output" tab you can choose if you want to hear the sounds in the headset or the loudspeakers installed under the monitor. In the "Input" tab you should verify that the headset input source is selected (in case there are more alternatives)[4]. You should see the Input level meter light up when you touch the microphone. Adjust the input volume so that the level meter reaches about 3/4 of maximum when you talk close to the microphone, but make sure that the level never reaches the maximum. More adjustment can and should be done during the recordings if necessary.



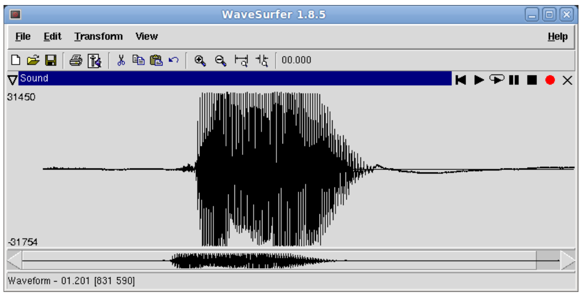


Then start the recording of the training utterances with the command:

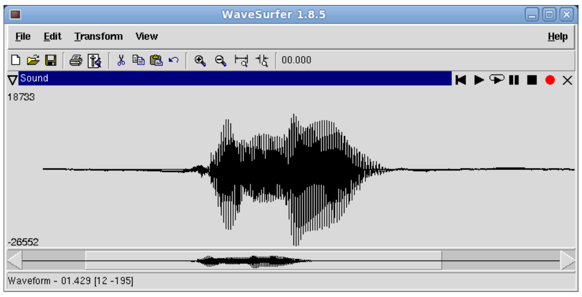
```
> tools/record_list Sp$n/train_data < Sp$n/trainprompts
```

The program `wavesurfer` will appear in a new window. It is used to record each prompted utterance. The prompt text to be spoken is displayed in the command window (which will possibly have to be moved not to be hidden by the wavesurfer window). An utterance recording is initiated by clicking the red record button and stopped by clicking the stop button. Note that you should leave a short silence interval before and after the utterance, but there is no need for pausing between the words in the phrase. **Microphone position: at the corner of the mouth. Distance: around 2 cm.** Check the amplification level on the first utterance. The waveform amplitude should be well above the background level, but the maximum amplitudes should not be clipped. If it is, adjust the input level, as described above. Repeat the recording until it is good. An example of bad and good recordings is shown in the figures below:

[4]If both the lists of Output and Input devices are empty, remove old configuration files with `rm -rf .pulse .config/pulse/` and restart PulseAudio with `pulseaudio --kill`



Bad recording (clipping)



Good recording

When you judge the recording to be good, save it by clicking on File/Save or the save button. When the file is saved, the next prompt is displayed in the command window. Repeat the recording of this and the following utterances in the same way until the end of the prompt list is reached.

Then do the same procedure for the test utterances

```
> tools/record_list Sp$n/test_data < Sp$n/testprompts
```

Remember to repeat the steps in this section for all the group members by setting the variable `n` to the corresponding index.

1.5 Training the models

In this section you will run a command that will train monophone HMM models for each speaker. The procedure is based on some configuration files that determine the kind of speech feature extraction and the topology of the models to train. Before you start, analyse the following configuration files and answer the corresponding questions questions in the report. Display the feature extraction configuration file and enter the requested parameter values answering the questions in the report template.

```
> cat config/features.cfg
```

You can check the kind of features that are implicitly computed by HTK in the training steps by running the command:

```
> tools/show_features.py config/features.cfg Sp1/train_data/TR1.wav
```

You can apply the command to any wave file you have recorded.

Display the model prototype and enter the requested parameter values in the report

```
> cat config/proto.mmf
```

Now run the training for each group member (`set n=`1, 2, 3), with the following command:

```
> tools/train.cmd Sp$n
```

This command will run a number of steps and display the current processing. In Table 1 is a list of steps that are performed. You are welcome to check the intermediate results and to consult the HTK Book for further details on the tools, although this is not required to complete the exercise.

#	description	tool	results
1	convert the prompts into ortographic transcriptions of the recordings	`tools/prompts2mlf.pl`	`Sp$n/trainprompts.mlf` `Sp$n/testprompts.mlf`
2	convert word level (ortographic) transcriptions into phone level transcriptions	`HLEd`	`Sp$n/phones0.mpl`
3	initialise the model parameters to global mean and variance of the data	`HCompV`	`Sp$n/hmm0/`
4	re-estimate the model parameters with three iterations of the Baum-Welch algorithm	`HERest`	`Sp$n/hmm1/` `Sp$n/hmm2/` `Sp$n/hmm3/`
5	adding short (one state) model for short pauses	`HHEd`, `HLEd`	`Sp$n/hmm4/` `Sp$n/hmm5/`
6	re-estimate the model parameters with two iterations of the Baum-Welch algorithm	`HERest`	`Sp$n/hmm6/` `Sp$n/hmm7/`

Table 1. Details on the training steps performed by `tools/train.cmd`

1.6 Evaluation

Evaluate the models on the test material from the same speaker (*speaker dependent*) by running:

```
> tools/recognize.cmd four_digits.lat Sp$n Sp$n
```

where the first argument specifies the grammar model (four consecutive digits in this case), the second argument the acoustic models to use and the third argument the test material, in this case they are both from the same speaker.

This command will run the HTK tool `HVite` that implements a Viterbi decoder and the tool `HResults` that realigns the recognition results with the reference and computes the recognition accuracy. The recognition results are presented file-by-file, as global measures over the test corpus and as a confusion matrix between individual words. In the matrix, the word identities in the first column and first row represent the correct and recognized identity, respectively. The results are also saved in two files (`recognized.mlf` and `evaluation.txt`) in the `results/` directory. **In this simple task, it is not impossible to achieve 100% accuracy. The results can vary depending on many factors. You should get suspiciuos if you get speaker dependent results much below 70-80%.** Report the accuracy, number of insertions and deletions and interesting details in the confusion matrix. At a later time, you can always run the recogniser again with the command above, if you want to inspect the results in more details.

Repeat the above commands for all speakers (n = 2, 3, etc.) in the group.

Also perform *cross-speaker* recognition by using the models trained on one speaker (e.g. Sp1) and the test data from another (e.g. Sp2) and vice versa. Discuss the difference in accuracy to that of speaker-dependent recognition and the likely cause of this. What could explain any difference in accuracy when swapping training and test speakers?

```
> tools/recognize.cmd four_digits.lat Sp1 Sp2
> tools/recognize.cmd four_digits.lat Sp2 Sp1
```

1.7 Beyond four digits

Create a grammar that can recognise a sequence of digits of any length (as opposed to the fixed length of four used above). To do this copy the file `four_digits.grm` into a new file `digit_loop.grm` and modify it's content. If you need help, consult the HTK Book, section 12.3. Expand the grammar into lattice format with the command:

```
> HParse digit_loop.grm digit_loop.lat
```

And visualise the corresponding graph with:

```
> tools/lat2pdf digit_loop.lat
> evince digit_loop.pdf
```

As before, report the grammar definition and explain how the the graph of the new grammar is obtained by the rules you defined.

Perform speaker dependent and cross-speaker recognition with the new grammar:

```
> tools/recognize.cmd digit_loop.lat Sp1 Sp1
> tools/recognize.cmd digit_loop.lat Sp1 Sp2
...
```

(or any other speaker pairs)

Compare the recognition results with the 4-digit grammar. What types of errors are special for the digit loop grammar?

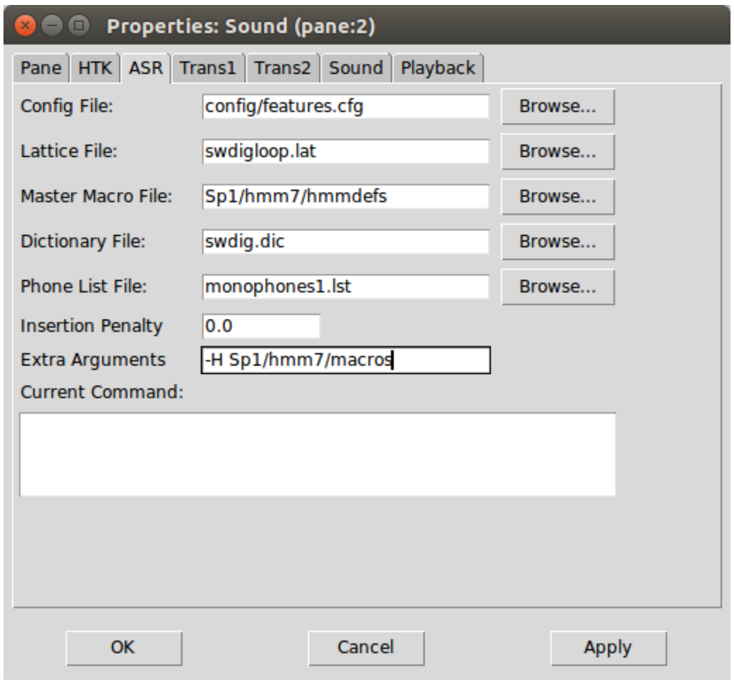
1.8 Running live

Run the recogniser live with the models trained on speaker Sp1, with microphone input using the command[5]:

```
 > wavesurfer -config config/wavesurferLiveRecognition.conf
```

Every time you record a new sound, the recognition results will be displayed in the transcription pane in wavesurfer. If you want to select new recognition models, right click on the transcription pane and click on "Properties...". Then select the HVite tab (see figure). You will need to change the string "Sp1" to "Sp2" or "Sp3" in the "Master Macro File" and "Extra Arguments" fields.

[5]The HVite plug-in is currently only available in the module version of wavesurfer at the CSC computers. The plug-in requires HTK, if you run the command from a new terminal, run "source config/environment" first.


Properties: Sound (pane:2)

Config File: config/features.cfg
Lattice File: swdigloop.lat
Master Macro File: Sp1/hmm7/hmmdefs
Dictionary File: swdig.dic
Phone List File: monophones1.lst
Insertion Penalty 0.0
Extra Arguments -H Sp1/hmm7/macros
Current Command:

Try to recognise sequences of digits spoken quickly without pause. Are there any missing digits in the recognition? Report your experience.

Now open the recognition properties and change the "Insertion Penalty" from 0.0 to positive and negative value. Try extreme values such as -100.0 and +100.0. Note that every time you click on "Apply" the recognition result will be updated. What do you observe? How do you explain it?

Now try to recognise sequences of digits that you have recorded making long pauses between them. Are there any insertions of digits? Can you improve the situation with the Insertion Penalty?

1.9 Adding extra silence

Modify the `digit_loop.grm` grammar and the dictionary `digits.dic` so to introduce a new terminal symbol (word) for silence segments. This symbol should be called `SIL` and should be optional after each digit in the loop. Remember to run `HParse digit_loop.grm digit_loop.lat` each time you modify `digit_loop.grm`. If you want, inspect the corresponding graph by running `tools/lat2pdf` as before. Try again to recognise sequence of digits with long pauses between them. Does your new grammar work better? Explain your modifications in the report.

1.10 Robustness to Microphone location

Using the live recogniser, try recording while holding the microphone a different distances from your mouth. Can you see any difference in the recognition accuracy?

1.11 Beyond digits

Finally add new words to your liking. Note that you will only be able to use the phonemes that are already in `monophones1.lst` to define the new word's pronunciations. Report your experience with recognising the new words.

A Own computer: Installing the required software

If you are running on you own machine instead of the CSC Ubuntu computer, you will need to run the following commands in order to install the required software (Tested on Ubuntu 14.04.3, 16.04.1 and 18.04.01):

1. Download HTK version 3.4.1 from `http://htk.eng.cam.ac.uk/` (this requires registering to the site)
2. extract the content with your favourite archive tool, for example from the command line:

   ```
   cd Downloads
   tar xvfz HTK-3.4.1.tar.gz
   cd htk
   ```

3. Install the required dependencies:

   ```
   sudo apt-get install libc6-dev-i386
   ```

4. configure and build HTK (you will get many warnings, but no error, hopefully):

   ```
   ./configure --disable-hslab --prefix=/opt/htk
   make
   ```

   If you receive the error: "`Makefile:77:  *** missing separator (did you mean TAB instead of 8 spaces?).`", edit the `HLMTools/Makefile` removing the spaces and adding a tab at the beginning of line 77. Then run make again.
5. install the software in the corresponding directory and add the path to your `PATH` variable:

   ```
   sudo mkdir /opt/htk
   sudo make install
   echo "export PATH=/opt/htk/bin:\$PATH" >> ~/.bashrc
   ```

6. test the installation: open a new terminal and run `HList`. You should get a usage description for the command
7. add the GraphViz package to be able to display the graph of the grammars:

   ```
   sudo apt install graphviz
   ```

8. install Wavesurfer: the version that is available on the Ubuntu repositories depends on TclTk 8.5 (the default version is 8.6 at the moment). If you don't mind installing the extra version of TclTk, just run:

   ```
   sudo apt-get install tk8.5 libsnack-alsa wavesurfer
   ```

   Otherwise, you can download the latest source version from `https://sourceforge.net/projects/wavesurfer/`, install snack with

   ```
   sudo apt-get install libsnack-alsa
   ```

and then run `wish .../src/app-wavesurfer/wavesurfer.tcl`

9. install the Wavesurfer plugin for live recognition: after you started wavesurfer for the first time, there will be a hidden directory called `.wavesurfer` in your home directory. Download the plugin from `https://www.speech.kth.se/asr/ws_plugin/hvite.plug` and copy the file into ~`/.wavesurfer/1.8/plugins/`. Restart wavesurfer.