IN4012TU "Real-time AI and Automated Speech Recognition" – Assignment I

Rogier van Dalen Pascal Wiggers

7th September 2009

In this assignment you will produce a simple speech recogniser. Not all the details you need are in the textbook we use, Jurafsky and Martin (2009). This document aims to fill in those gaps. It also gives information specific to this practical, for example about the input files.

1 The lexicon

The *lexicon* provides a mapping from words' orthography (i.e. spelling) to sequences of phonemes. This lexicon contains Dutch words with *phonemic transcriptions*. Each line contains a word followed by a sequence of phonemes that indicate how the word is pronounced. The lexicon uses the CGN notation, which in turn is based on the SAMPA notation. All your speech recogniser has to do is find out which of the phoneme sequences contained in the lexicon is most likely to have produced the acoustics. To deal with any silence before and after the word, a *sil* phoneme should be introduced before and after all phoneme sequences. For example, if the phoneme sequence with the highest probability is *sil b schwa w aa r schwa sil*, then the word that was pronounced must have been *bewaren* /bəwaːrə/.

2 Signal processing

Before an audio file (for example, a WAV file) can be used in a speech recogniser, acoustic *features* have to be extracted from every *time slice*. We will use a pre-made tool to compute MFCC features for every 10 ms time slice.

The Hidden Markov Toolkit (HTK) is free, written in C, and distributed by the University of Cambridge. Though the tools inside it were at some point owned by Microsoft Research, they are Unix-style command line tools. They are used by many companies and research institutes to train their speech recognisers and try things out. Our speech recogniser has already been trained; section 3.2 on page 4 explains the content of `hmms.mmf` in detail. From the Hidden Markov Toolkit we will only use two tools that extract feature information from audio files. `HCopy.exe` and `HList.exe` are in the `hkt_tools_windows.zip` file that should come with this manual. If you like to, you can instantly register at no cost with the Hidden Markov Toolkit website at `http://htk.eng.cam.ac.uk/` to browse or download the documentation or the tool source code. Users of

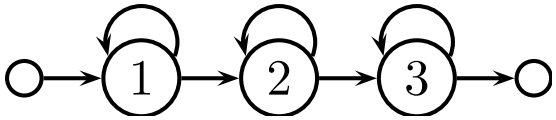

Figure 1: A standard three-state HMM, with non-emitting states 1 and 5, and emitting states 2, 3, and 4. Emitting states are associated with an output distribution b_i.

other operating systems than Windows should download the source code and compile it on their systems. This should not be much of a hassle.

The first tool you will use is HCopy, so named because it copies information from audio files into HTK format files with MFCCs. HCopy can be called thus:

```
HCopy -C hcopy_mfcc.cfg tf000116.wav tf000116.mfc
```

to convert `tf000116.wav` to `tf000116.mfc`. `hcopy_mfcc.cfg` is a configuration file that is also available from Blackboard. It instructs HCopy to produce 39 MFCC features per 10 ms time slice. These features are saved in a binary file format that is described in the HTK manual (Young *et al.*, 2005, 5.10.1). HList converts the HTK binary files to plain text. You can call it thus:

```
HList -r tf000116.mfc
```

to output the feature vectors to the standard output (i.e. the terminal). The output will look something like this:

```
1.047198e+000 2.854144e+000 5.039756e+000 2.466356e+000 3.066413e+000 -3.922853e+000
-3.282447e+000 -1.032732e+000 3.812215e+000 2.047029e+000 7.291181e+000 2.135985e+000
```

but run on for several pages per second of audio file. There are 39 floating point numbers per line: each line contains one feature vector. This list does not directly add much to our understanding of the audio file. We will need to compare this data to a profile of a phoneme.

These tools are provided as a convenience. You do not have to use HList: you can read the binary file format HCopy produces directly. You do not have to use HCopy if you want to call functions from the HTK library directly from your program. Though it will make your speech recogniser more self-sufficient, and therefore much cooler, it will not make your life easier.

3 hmms

A *hidden Markov model* (long for "HMM") that outputs a phoneme is depicted in figure 1. This one has five states. Two are *non-emitting*, that is, they are merely endpoints for concatenation. Three actually produce sound. A transition from state i to state j happens with probability a_{ij}. Every box in figure 1 stands for a function b_i that defines probabilistically, in terms of the acoustic features, what sounds are associated with the state. The reason there are three emitting states for one phoneme rather than one is that the first bit of a phoneme is influenced by the previous phoneme, and the last bit by the next one.

When actually recognising speech, the output we expect is compared to the observation. The *Viterbi algorithm* finds a path through the states that best explains the audio signal. See Jurafsky and Martin (2009) for an explanation

Table 1: Feature vector from an audio file.

$$\text{æ} = \begin{bmatrix} 1.713696\text{e-}001 \\ 3.440685\text{e-}001 \\ -8.538713\text{e-}001 \\ 3.463801\text{e+}000 \\ 1.651544\text{e+}000 \\ -5.355577\text{e+}000 \\ 4.956263\text{e+}000 \\ 2.541015\text{e+}000 \\ 4.452126\text{e-}001 \\ -3.111563\text{e+}000 \\ -1.992391\text{e+}000 \\ -5.376299\text{e+}000 \\ 1.777077\text{e-}001 \\ 3.770196\text{e-}001 \\ 6.087872\text{e-}001 \\ 1.686706\text{e+}000 \\ 8.062722\text{e-}001 \\ 5.015219\text{e-}001 \\ 3.260719\text{e-}001 \\ -1.908163\text{e+}000 \\ -1.791249\text{e+}000 \\ 1.938380\text{e+}000 \\ 2.100054\text{e+}000 \\ -6.479878\text{e-}001 \\ -3.242626\text{e-}002 \\ -1.169251\text{e-}003 \\ -1.236923\text{e-}001 \\ -8.561335\text{e-}002 \\ 3.323697\text{e-}001 \\ -5.539750\text{e-}001 \\ -2.726229\text{e-}001 \\ 5.833805\text{e-}001 \\ -3.306281\text{e-}001 \\ -4.484353\text{e-}001 \\ -1.029309\text{e+}000 \\ -3.137693\text{e-}001 \\ -9.858458\text{e-}002 \\ 4.486263\text{e-}001 \\ 4.803071\text{e-}003 \end{bmatrix}$$

(The Viterbi algorithm is also explained in more general terms in Russell and Norvig (2003, 15.2).) To recognise a sequence of phonemes we have to concatenate the HMMs. This is where the non-emitting states come in handy. Every HMM has a start state and an end state. We can use the non-emitting states to connect copies of various HMMs, thereby forming a larger HMM. An HMM is generated for every word in the lexicon. For example, an HMM for recognising the word *cheque* /ʃɛk/ will be a concatenation of the phonemes *sil*, *sh*, *e*, *k*, and *sil*, forming a string of 15 states altogether.[1]

3.1 Acoustic modelling

A feature vector is extracted from the raw audio for every 10 ms time slice. Table 1 shows such a feature vector, consisting of 39 MFCC features (actually, one energy feature and 12 MFCC features, and their derivatives, and *their* derivatives). To decide whether the acoustics of these 10 ms could be the result of the first part of an /aː/, we compare the feature vector to the sound of the /aː/s we trained the recogniser on. The properties of those /aː/s are captured in *Gaussians*. A Gaussian is the probability distribution that you know from statistics courses as the normal distribution. As you remember, the normal distribution has two parameters: the mean μ and the variance σ. Since one feature vector contains 39 values, the output of an HMM state is defined by 39 μs and 39 σ^2s. In the HMM definition file these are contained in two vectors $\boldsymbol{\mu}$ and $\boldsymbol{\sigma^2}$, exemplified in table 2 on the following page.

[1] All HMMs we will use have three emitting states. This is not necessarily true in general, but the canonical HMM has only three emitting states.

Table 2: Mean and variance for the first emitting state of a trained HMM for the phoneme /aː/.

$$\boldsymbol{\mu} = \begin{bmatrix} 3.494631e{+}000 \\ -1.041954e{+}001 \\ -2.617091e{+}000 \\ -1.806885e{+}000 \\ 2.389358e{-}001 \\ 1.490016e{+}000 \\ 1.718114e{+}000 \\ -4.997996e{+}000 \\ -3.919698e{+}000 \\ -1.377204e{+}000 \\ -7.303527e{-}001 \\ 6.678443e{-}002 \\ 8.345999e{-}001 \\ -1.137392e{-}001 \\ -1.621469e{+}000 \\ -8.307990e{-}002 \\ 1.002544e{+}000 \\ 7.614670e{-}001 \\ 4.673814e{-}001 \\ -1.236874e{-}001 \\ -8.338841e{-}001 \\ -4.626290e{-}001 \\ 7.504206e{-}002 \\ 3.008948e{-}001 \\ 1.628574e{-}001 \\ 3.587579e{-}002 \\ -1.958797e{-}001 \\ 4.565939e{-}001 \\ 3.003341e{-}001 \\ 1.854419e{-}001 \\ -5.208018e{-}002 \\ -2.726975e{-}001 \\ -1.469530e{-}001 \\ 2.463291e{-}001 \\ 2.193729e{-}001 \\ 9.490074e{-}002 \\ 4.042798e{-}002 \\ -3.333719e{-}002 \\ -1.544064e{-}002 \end{bmatrix}, \quad \boldsymbol{\sigma^2} = \begin{bmatrix} 1.449228e{+}001 \\ 2.109260e{+}001 \\ 2.907835e{+}001 \\ 4.842424e{+}001 \\ 3.910004e{+}001 \\ 4.323362e{+}001 \\ 5.273534e{+}001 \\ 6.665879e{+}001 \\ 5.475682e{+}001 \\ 4.622672e{+}001 \\ 4.192024e{+}001 \\ 3.376055e{+}001 \\ 1.242084e{-}002 \\ 1.612679e{+}000 \\ 2.157096e{+}000 \\ 2.408889e{+}000 \\ 3.135437e{+}000 \\ 3.136988e{+}000 \\ 3.991289e{+}000 \\ 4.134397e{+}000 \\ 4.163263e{+}000 \\ 4.508152e{+}000 \\ 3.797008e{+}000 \\ 3.507058e{+}000 \\ 2.850974e{+}000 \\ 1.700877e{-}003 \\ 3.036627e{-}001 \\ 3.269685e{-}001 \\ 3.568493e{-}001 \\ 6.031566e{-}001 \\ 5.706108e{-}001 \\ 6.718093e{-}001 \\ 7.402067e{-}001 \\ 7.851482e{-}001 \\ 7.109802e{-}001 \\ 6.530800e{-}001 \\ 6.206974e{-}001 \\ 4.968702e{-}001 \\ 2.659142e{-}004 \end{bmatrix}$$

How do we calculate whether the observation, as in table 1 on the previous page, fits the acoustics we expect for /aː/? We separately calculate the probability of every feature fitting the acoustics. This we do with the probability density function for the normal distribution, which as you remember is

$$\frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{(x-\mu)^2}{2\sigma^2}}. \tag{1}$$

We calculate this probability for all features and then multiply those numbers to get the probability of the observation being the output of one state. Note that this assumes that all the features are independent from one another, which is nonsense, but hey! it makes the calculations a lot easier.

This is what takes place in the boxes that are associated with the HMMs as in figure 1 on page 2.

3.2 hmms.mmf

`hmms.mmf` contains a description of a trained speech recogniser. It is an HTK HMM definition file, slightly edited to make it simpler to parse. Its format looks, but is not quite, like XML. Table 3 on the next page contains an excerpt from the file.

The description of the phoneme Y[2] starts with `~h "Y"`. `<NUMSTATES>` precedes the total number of states for this HMM. This is always 5. The first and last states are always non-emitting states. `<STATE>` starts the description of a

[2] /yː/ in IPA notation.

Table 3: The first HMM definition from hmms.mmf.

```
~h "Y"
<BEGINHMM>
<NUMSTATES> 5
<STATE> 2
<MEAN> 39
 2.354841e+000 5.610937e+000 5.067012e+000 -1.055201e+001 -4.812210e+000 -4.014295e+000 ...
<VARIANCE> 39
 1.762409e+001 2.296807e+001 2.819459e+001 5.337686e+001 4.616655e+001 4.337509e+001 ...
<GCONST> 1.113207e+002
<STATE> 3
<MEAN> 39
 3.699092e+000 6.638847e+000 7.107974e+000 -1.402351e+001 -6.396945e+000 -3.252827e+000 ...
<VARIANCE> 39
 1.022347e+001 1.661865e+001 2.492645e+001 2.507180e+001 3.740020e+001 3.738739e+001 ...
<GCONST> 8.564902e+001
<STATE> 4
<MEAN> 39
 5.037427e+000 2.506331e+000 4.572251e+000 -9.221249e+000 -9.585092e+000 -4.441896e+000 ...
<VARIANCE> 39
 1.094445e+001 2.240830e+001 2.484098e+001 2.858117e+001 3.546447e+001 3.863903e+001 ...
<GCONST> 9.314183e+001
<TRANSP> 5
 0.000000e+000 1.000000e+000 0.000000e+000 0.000000e+000 0.000000e+000
 0.000000e+000 5.805153e-001 4.194847e-001 0.000000e+000 0.000000e+000
 0.000000e+000 0.000000e+000 7.690973e-001 2.309027e-001 0.000000e+000
 0.000000e+000 0.000000e+000 0.000000e+000 7.553328e-001 2.446672e-001
 0.000000e+000 0.000000e+000 0.000000e+000 0.000000e+000 0.000000e+000
<ENDHMM>
```

state. In this edited version of the file the distribution is always contained in the state description. The vectors $\boldsymbol{\mu}$ and $\boldsymbol{\sigma^2}$ (see table 2 on page 4) follow `<MEAN>` or `<VARIANCE>` and the vector size, 39. You can skip the value for `<GCONST>`, a precalculated value that the HTK programmers included in a fit of fear of hurting performance. The number is the logarithm of the squared product of $\sigma\sqrt{2\pi}$ from equation (1) on page 4:

$$\ln \prod_{i=1}^{N} 2\pi \boldsymbol{\sigma}_i^{\mathbf{2}} \tag{2}$$

with $\boldsymbol{\sigma}_i^{\mathbf{2}}$ the ith entry of the variance vector. You may wonder why the HTK programmers forgot to take the reciprocal of the square root (see equation (1)). We will leave the answer as an exercise, but give one hint: it is a slight optimisation possible because probabilities are represented by their logarithms (see appendix A on the facing page). we left the figure in so you may check whether your implementation calculates the number correct up to this point.

The section marked `<TRANSP>` contains the transition probabilities (see figure 1 on page 2) in matrix form:

$$\begin{bmatrix} a_{11} & a_{12} & a_{13} & a_{14} & a_{15} \\ a_{21} & a_{22} & a_{23} & a_{24} & a_{25} \\ a_{31} & a_{32} & a_{33} & a_{34} & a_{35} \\ a_{41} & a_{42} & a_{43} & a_{44} & a_{45} \\ a_{51} & a_{52} & a_{53} & a_{54} & a_{55} \end{bmatrix} \tag{3}$$

Almost all acoustic models are like the one in figure 1 on page 2. This means that almost all transition probability matrices look like

$$\begin{bmatrix} 0 & 1 & 0 & 0 & 0 \\ 0 & a_{22} & a_{23} & 0 & 0 \\ 0 & 0 & a_{33} & a_{34} & 0 \\ 0 & 0 & 0 & a_{44} & a_{45} \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix} \tag{4}$$

but not all! The *sil* phoneme is the exception. You are allowed to assume that the transition matrix for *sil* is also like in (4). If so, please say so in the documentation you submit. It will make your speech recogniser perform slightly worse, but it will significantly reduce the strain on your programming skills.

3.3 lexicon.txt

`lexicon.txt` contains all words that can be recognised with their phonemic transcription.

3.4 hcopy_mfcc.cfg

`hcopy_mfcc.cfg` tells HCopy how it should produce MFCC features (see section 2 on page 2). Your program does not have to parse this file.

A Representing small probabilities

Applying Viterbi on hidden Markov models yields very low probabilities.[3] Even on short audio fragments the probability of the most likely path will routinely become as low as 10^{-10000}. 64-bit floating point numbers (e.g. Java `doubles`) can represent values as low as $2.22507 \cdot 10^{-308}$; anything lower is rounded to 0. By representing a probability p as its logarithm $\ln p$ we can reach a lowest non-zero value of $e^{-1.79769 \cdot 10^{308}}$ using doubles. Operations then have to work on log-probabilities as well. For example, to add probabilities a and b we need to formulate $\ln(a+b)$ in terms of $\ln a$ and $\ln b$. Table 4 on the following page summarises how computers can use log-probabilities in calculations.

A.1 Multiplication

$$\ln(a \cdot b) = \ln a + \ln b \tag{5}$$

$$\ln\left(\frac{a}{b}\right) = \ln a - \ln b \tag{6}$$

A.2 Addition

Note that addition and subtraction are not needed for recognising with the Viterbi algorithm, i.e. you do not need to implement these. For training, however, they are necessary.

$$\ln(a+b) = \ln(e^{\ln a} + e^{\ln b}) \tag{7}$$

This simple solution underflows if $a < \epsilon$ or $b < \epsilon$, which is exactly what we aimed to cure by using log-probabilities. Let us assume $a \geq b$.

$$\begin{aligned}\ln(a+b) &= \ln\left(a + a \cdot \frac{b}{a}\right) = \ln\left(a\left(1 + \frac{b}{a}\right)\right) = \ln a + \ln\left(1 + \frac{b}{a}\right) \\ &= \ln a + \ln\left(1 + e^{\ln \frac{b}{a}}\right) = \ln a + \ln\left(1 + e^{\ln b - \ln a}\right)\end{aligned} \tag{8}$$

If $a = 0$, however (represented as $\ln a = -\infty$), then this trick is not possible. This begs to be special-cased.

Subtraction works similarly:

$$\begin{aligned}\ln(a-b) &= \ln\left(a - a \cdot \frac{b}{a}\right) = \ln\left(a\left(1 - \frac{b}{a}\right)\right) = \ln a + \ln\left(1 - \frac{b}{a}\right) \\ &= \ln a + \ln\left(1 - e^{\ln \frac{b}{a}}\right) = \ln a + \ln\left(1 - e^{\ln b - \ln a}\right)\end{aligned} \tag{9}$$

This is an invalid operation when $1 \leq e^{\ln b - \ln a} \Rightarrow \ln a \leq \ln b$ which is obvious because when $a \leq b$, $a - b \leq 0$. The logarithmic representation can only represent positive numbers.

[3] Viterbi uses the maximum rather than the sum of probabilities, and the normalising factor (the denominator in Bayes' rule) is usually left out.

Table 4: Representating small probabilities as their logarithms.

Probability	**Representation**
p	$\ln p$
0	$-\infty$
1	0
$+\infty$	$+\infty$
$a \cdot b$	$\ln a + \ln b$
$\frac{a}{b}$	$\ln a - \ln b$
$a + b$	$\ln a + \ln\left(1 + e^{\ln b - \ln a}\right)$
$a - b$	$\ln a + \ln\left(1 - e^{\ln b - \ln a}\right)$
$\sqrt{a}$	$\frac{1}{2} \ln a$

A.3 Comparison

$$a = b \Leftrightarrow \ln a = \ln b \tag{10}$$

$$a < b \Leftrightarrow \ln a < \ln b \tag{11}$$

$$a > b \Leftrightarrow \ln a > \ln b \tag{12}$$

References

D. Jurafsky and J. H. Martin (2009), *Speech and Language Processing: An Introduction to Natural Language Processing, Computational Linguistics and Speech Recognition*, Prentice Hall, Upper Saddle River.

S. Russell and P. Norvig (2003), *Artificial Intelligence: A Modern Approach*, Prentice-Hall, Eaglewood Cliffs, 2nd ed.

S. Young, G. Evermann, M. Gales, T. Hain, D. Kershaw, G. Moore, J. Odell, D. Ollason, D. Povey, V. Valtchev, and P. Woodland (2005), 'The HTK book (for HTK version 3.3)', URL `http://htk.eng.cam.ac.uk/docs/docs.shtml`.