Demonstrator – HTK Tool Kit **Speech Recognition I**

Zhengkun Mei S5075580
Veronica Valente S5268516

WORKFLOW

- HTK Installation on a linux based system: Ubuntu 20.04
notes: convert spaces into tabs in makefile, important to use sudo while running “make install”

- absolute paths lists extracted with the following python code:

```
import os

directory="/directory_Path/directory”

def absolute_file_paths(directory):
    path = os.path.abspath(directory)
    return [entry.path for entry in os.scandir(path) if entry.is_file()]

print("\n".join(absolute_file_paths(directory)))
```

- after the creation of training_wavlist.scp, the mfc paths list is created copying the wavlist in a new file training_mfclist.scp and substituting “wav” with “mfc” using command line: sed -i 's/wav/mfc/g' training_mfclist.scp . A similar procedure has been followed for digit-individual absolute paths lists (e.g. train_0_mfc.scp etc.)

- created file training.scp with the provided command line: paste -d ' ' training_wavlist.scp training_mfclist.scp > training.scp

- created folder HMM0 containing initial HMM definitions. We prepared the definitions accordingly to the model given in the tutorial, hmm definitions ZERO, ONE, TWO etc. We can notice, at the beginning of the file (the header), before the actual beginning of HMM definition, the number of feature vectors (<VecSize>26) that are needed to create the HMM here specified, this HMM will be created from 26 component feature vectors. It’s important that the parameters of every state (mean and variance) match this dimension.
The next line specifies the number of states in the HMM, here is 5, however we can notice that just 3 states are emitting and the last one is related to transitions. The first symbol in the file ~h indicates that the following string is the name of a macro of type **h** which means that it is a HMM definition (~r for example would have meant “regression class tree”).
Then we have a definition for each emitting state j, each of which has a single mean vector μ_j introduced by the keyword <Mean> and a diagonal variance vector Σ_j introduced by the keyword <Variance>. The definition ends with the transition matrix to produce transitions’ observations between states, introduced by the keyword <TransP>.

These definitions will be first used to initialize actual values for parameters of each state (HInit) for each digit and the resulting values will be used for re-estimating (HRest) such parameters to return the optimal state sequence for each digit.

(Note: on the tutorial, in the code given for initial HMM definitions, there are 27 empty values under the mean of state 3, that raised the HMError +7050 that on the htk manual corresponds to “Model file format error”, we need to look better at the error to find the problem, we also got the

error line “HMM Def Error: GetToken: Symbol expected at line 15/col 10/char 429 in HMM0/ZERO” that gives us the position of the Error in order to fix it.)
Mean initial values for each state are set to 0.0 and for Variance to 1.0 (fixed also variance state 3).

- MFCCs have been extracted using the provided command line: HCopy -A -D -T 1 -C config -S training.scp . As we can see from the script, this extraction is possible through the Htk tool Hcopy which extracts mfcc features and copy them in the “empty” mfc paths defined in training.scp. MFCCs characteristics are based on the file “config” where we can specify for example the window size and weather or not to use an hamming technique for windowing, in this case the window size is set on 25 ms and usehamming set to True. Here we can specify also the number of channels (26) that will be reflected in the number of feature vectors used by the HMM model. The first configuration parameter TARGETKIND requires a string value specifying the name of a speech parameter kind, in this case, as suggested in the tutorial, MFCC energy component and 1 st order difference coefficients. NUMCHANS requires an integer value specifying the number of filter-bank channels to use in the analysis. WINDOWSIZE requires in fact a floating-point value specifying the window size in units of 100ns. However, an integer can always be given wherever a float is required.

Image from HTK book to better understand Hcopy command. It extracts MFCCs features and copy them in the previously created path lists.

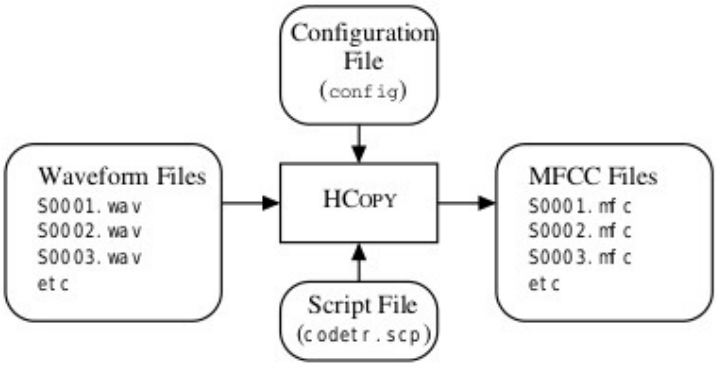

Fig. 3.6 Step 5

- In creating scripts for initializing HMMs and then re-estimating HMMs (hinit_script.sh and hrest_script.sh), it’s important that the names of .scp files match the actual .scp filenames, since these are the files from which mfcc paths will be taken to use extracted mfccs for initializing and re-estimating HMMs parameters. So if I have a file “**training_0_mfc.scp**”, this will be my command to initialize HMM for ZERO, HInit -T 2 -S **training_0_mfc.scp** -H HMM0/ZERO -M HMM1 ZERO and store it in folder HMM1 and this other will be the command for re-estimating the just stored parameters HRest -T 2 -S **training_0_mfc.scp** -H HMM1/ZERO -M HMM2 ZERO and store the resulting model into the folder HMM2. It is important, indeed, that folders HMM1 and HMM2 are created before running the scripts.

- **Training:** Initialization of parameters with Hinit tool taking mfcc paths from individual .scp files (e.g. train_0_mfc.scp etc.), and running of tool HRest for parameters’ re-estimation of HMMs in folder HMM1.

Note: in both models definitions resulting from initialization (HMM1) and re-estimation (HMM2), we can notice a new line <GCONST>, that, according to the manual, is that part of the log

probability of a Gaussian that can be precomputed. If it is omitted, then it will be computed during load-in (as in this case), including it simply saves some time. HTK tools which output HMM definitions always include this field.

- **Testing:**
1) extraction of MFCCs features from wavs chosen for testing based on the same config file used in training
2) creation of the file gram.txt to specify which are the “recognizable” words
3) creation of .slf file from gram.txt to make it understandable to Hvite tool that will be used to perform Viterbi decoding on test segments’ extracted mfcc features.
To perform this step the tool Hparse is used, its running generates word level lattice files (for use with e.g. HVite) from a text file syntax description containing a set of extended BNF format grammar rules.
4) creation of the file dict.txt to map words to HMM names (e.g. <~h "ZERO"> etc.)
5) creation of hmmlist.txt file and Viterbi decoding performing with the last command line:

HVite -H HMM2/ZERO -H HMM2/ONE -H HMM2/TWO -H HMM2/THREE -H HMM2/FOUR -H HMM2/FIVE -H HMM2/SIX -H HMM2/SEVEN -H HMM2/EIGHT -H HMM2/NINE -i reco.mlf -w net.slf dict.txt hmmlist.txt -S test_mfc.scp

TEST RESULTS AND OBSERVATIONS

Metrics for Evaluation
Accuracy: 100*(number of correct predictions / total number of predictions)
WER: 100*(number of wrong predictions / total number of predictions)

Note: here the equation to find WER (World Error Rate) is simplified considering at the numerator just the “substitutions”

First Test (with “suza” samples – more explanation later)

Train: 390 samples (without “vero” samples)
Test: 30 samples (vero)
correct predictions: 9
wrong predictions: 21
Accuracy: 30%
WER: 73.3%

Since the results were not satisfying, we decided to test the model on “mei” samples, keeping them both in TRAIN and TEST folders to check the results.

Second Test (with “suza” samples)

Train: 390 samples (with “mei” samples, without “vero” samples)
Test: 30 samples (mei)
correct predictions: 26
wrong predictions: 4
Accuracy: 86.7%
WER: 13.3%

We can see from the results the there are four errors in 30 samples considered and the accuracy is approximately 86.7%, the tool recognizes 0 as 2, 1 as 5, and 9 as 5 two times, so we get the "obvious" conclusion that we will have higher accuracy when the test data is also part of the train data. However, it also made us thinking that there was something wrong (e.g. noise) with the training samples, so we have decided to check their quality (noisiness, signal "purity", volume, intelligibility of the digit), listening to them. After listening we found some longer samples "suza" of 18s , we also realized that in "vero" samples there is background noise, and they are overall worst in quality, compared to "mei" samples. We suppose that this could be the reason why "vero" recordings are difficult to recognize.
We then decided to remove "suza" samples from TRAIN and test the model again.

Third Test (reduced train samples)

Train: 330 samples (without "suza" and "vero")
Test: 30 samples (vero)
correct predictions: 3
wrong predictions: 27
Accuracy: 10%
WER: 90%

As we can see, the results are at the opposite of being satisfying, the tool recognizes just samples were the digit ZERO is spoken, showing that the noisiness of "vero" samples plays a key role in making the tool recognizing the spoken digit, an interesting fact is that, compared to the very first test (with "suza" in TRAIN and "vero" in TEST), here the results are worst, showing that "suza" samples were somehow improving the performance of the models on "noisy" samples.

For comparison with previous tests, we run again the test on "mei" samples keeping the same training files, here the results

Train: 330 samples (without "suza" and "vero")
Test: 30 samples (mei)
correct predictions: 100
wrong predictions: 0
Accuracy: 100%
WER: 0%

the results are now in line with our expectations, having the same data for testing also in training, we have an accuracy of 100%.

Finally, to measure the influence of signal "noisiness" on performance, we also run a test on "vero" samples, keeping them also in TRAIN as we did with "mei" samples on the previous test, below our results:

Train: 360 samples (without "suza")
Test: 30 samples (vero)
correct predictions: 25
wrong predictions: 5
Accuracy: 83.3%
WER: 16.6%

As we can see, even keeping the same files in train and test, we still got some errors and a WER of 16.6%, the problem comes to spoken digits "six" and "four", both misrecognized for the word digit

“zero”, 3 and 2 times respectively. We suppose then, that the misrecognition occurs on states regarding the prediction of sounds “s/z” and “r”.

Last Test and Submission

We finally decided to repeat the procedure with the following data splitting:

training – 360 samples (without “suza” and “mei”)
testing – 30 samples (mei)

we then run the hvite tool, on the hmm definitions resulting after initialization and re-estimation of the parameters based on mfcc extraction from files in training, and test on mfcc features extracted from “mei” samples. The results as follows:

Train: 360 samples (without “mei” and “suza”)
Test: 30 samples (mei)
correct predictions: 28
wrong predictions: 2
Accuracy: 93.3%
WER: 6.6%

testing on the final version of our prediction model, among the 30 samples task, we only have two recognition errors during testing, they are on spoken digits “mei_3_2” and “mei_8_3”, it misrecognized the first one as word digit TWO and the second one as SIX.
Listening to the recordings of spoken digits “three” and “eight” and comparing them with the recordings of respectively “ two” and “six”, we observed that “mei” samples “3_1” and “3_3” “3_2” contain a shorter vowel [i:] , so we suppose that the vowel’s feature could be replaced partly by the consonant feature [r], and if we compare [r] in three and the [u:] in two, we can observe that both these sounds are produced by the speaker with the rounded lip, their difference is in the retroflection of the tongue. In fact, in a non-native speaker as Mei, the retroflection part could not be pronounced very clearly, and this could be one of the reasons, that leads to spoken digit “three” mis-recognization. For what that concerns “mei_8_3”, comparing it with the other two samples from “8”, we realized that the third sample has a stronger emphasis on the [i], occuring in the initial diphone [ei], in the pronunciation of “ six”, the [i] sound also represents the core part and, for this reason, may lead to this misrecognition.

Conclusions

Overall the results are satisfying. We can notice from them how the noisiness and quality of the data could negatively affect the prediction model performance and also that using more data for training (as in the first two tests with “suza” samples) could improve it, in presence of noisy samples.
We also have noticed that main errors occur in presence of sounds that can be “exchanged” as in s/z or in segments that share some peculiar sounds like [i] or [r].
Along with this report, a .zip file containing HMM2 folder and the reco.mlf file resulting from the last test are submitted.

References:

- Steve Young, et al., “The HTK Book” (for HTK Version 3.4), December 2006