This tutorial describes version 5prealpha. It will not work in older version. Please update.

Adapting the default acoustic model

This page describes how to do some simple acoustic model adaptation to improve speech recognition in your configuration. Please note that the adaptation doesn't necessary adapt for a particular speaker. It just improves the fit between the adapatation data and the model. For example you can adapt to your own voice to make dictation good, but you also can adapt to your particular recording environment, your audio transmission channel, your accent or accent of your users. You can use model trained with clean broadcast data and telephone data to produce telephone acoustic model by doing adaptation. Cross-language adaptation also make sense, for example you can adapt English model to sounds of other language by creating a phoneset map and creating other language dictionary with English phoneset.

The adaptation process takes transcribed data and improves the model you already have. It's more robust than training and could lead to a good results even if your adaptation data is small. For example, it's enough to have 5 minutes of speech to significantly improve the dictation accuracy by adaptation to the particular speaker.

The methods of adaptation are a bit different between PocketSphinx and Sphinx4 due to the different types of acoustic models used. For more technical information on that see Acoustic Model Types.

Creating an adaptation corpus

The first thing you need to do is create a corpus of adaptation data. This will consist of a list of sentences, a dictionary describing the pronunciation of all the words in that list of sentences, and a recording of you speaking each of those sentences.

Required files

The actual set of sentences you use is somewhat arbitrary, but ideally it should have good coverage of the most frequently used words or phonemes in the set of sentences or the type of text you want to recognize. For example, if you want to recognize isolated commands, you need tor record them. If you want to recognize dictation, you need to record full sentences. For simple voice adaptation we have had good results

simply using sentences from the CMU ARCTIC text-to-speech databases. To that effect, here are the first 20 sentences from ARCTIC, a fileids file, and a transcription file

- arctic20.fileids
- arctic20.transcription

The sections below will refer to these files, so it would be a good idea to download them now. You should also make sure that you have downloaded and compiled sphinxbase and sphinxtrain.

Recording your adaptation data

In case you are adapting to a single speaker you can record the adaptation data yourself. This is unfortunately a bit more complicated than it ought to be. Basically, you need to record a single audio file for each sentence in the adaptation corpus, naming the files according to the names listed in `arctic20.transcription` and `arctic20.fileids`. In addition, you will **NEED TO MAKE SURE THAT YOU RECORD AT A SAMPLING RATE OF 16 KHZ (or 8 kHz if you adapt a telephone model) IN MONO WITH SINGLE CHANNEL.**

The simplest way would be to start sound recorder like Audacity or Wavesurfer and read all sentences in a big audio file. Then you can cut audio files on sentences in a text editor and make sure every sentence is saved in the corresponding file. The file structure should look like this:

```
arctic_0001.wav

arctic_0002.wav

.....

arctic_0019.wav

arctic20.fileids

arctic20.transcription
```

You should verify that these recordings sound okay. To do this, you can play them back with:

```
for i in *.wav; do play $i; done
```

If you already have recording of the speaker, you can split it on sentences and create fileids and transcription files.

If you are adapting to a channel, accent or some other generic property of the audio, then you need to collect a little bit more recordings manually. For example, in call center you can record and transcribe hundred calls and use them to improve the recognizer accuracy by means of adaptation.

Adapting the acoustic model

First we will copy the default acoustic model from PocketSphinx into the current directory in order to work on it. Assuming that you installed PocketSphinx under `/usr/local`, the acoustic model directory is `/usr/local/share/pocketsphinx/model/en-us/en-us`. Copy this directory to your working directory:

```
cp -a /usr/local/share/pocketsphinx/model/en-us/en-us .
```

Lets also copy the dictionary and the lm for the testing

```
cp -a /usr/local/share/pocketsphinx/model/en-us/cmudict-en-us.dict .

cp -a /usr/local/share/pocketsphinx/model/en-us/en-us.lm.dmp .
```

Generating acoustic feature files

In order to run the adaptation tools, you must generate a set of acoustic model feature files from these WAV audio recordings. This can be done with the `sphinx_fe` tool from SphinxBase. It is imperative that you make sure you are using the same acoustic parameters to extract these features as were used to train the standard acoustic model. Since PocketSphinx 0.4, these are stored in a file called `feat.params` in the acoustic model directory. You can simply add it to the command line for `sphinx_fe`, like this:

```
sphinx_fe -argfile en-us/feat.params \

        -samprate 16000 -c arctic20.fileids \

       -di . -do . -ei wav -eo mfc -mswav yes
```

You should now have the following files in your working directory:

```
en-us

arctic_0001.mfc

arctic_0001.wav

arctic_0002.mfc

arctic_0002.wav

arctic_0003.mfc

arctic_0003.wav

.....

arctic_0020.wav

arctic20.fileids

arctic20.transcription

cmudict-en-us.dict

en-us.lm.dmp
```

Converting the sendump and mdef files

Some models like en-us are distributed in compressed version. Extra files required for adaptation are excluded to save space. For en-us model from pocketsphinx you can download the full version suitable for adaptation from the downloads:

cmusphinx-en-us-ptm-5.2.tar.gz

Make sure you are using the full model with the mixture_weights file present.

If mdef file inside the model is converted to binary, you will also need to convert the `mdef` file from the acoustic model to the plain text format used by the SphinxTrain tools. To do this, use the `pocketsphinx_mdef_convert` program:

```
pocketsphinx_mdef_convert -text en-us/mdef en-us/mdef.txt
```

In downloads the mdef is already in the text form.

Accumulating observation counts

The next step in adaptation is to collect statistics from the adaptation data. This is done using the `bw` program from SphinxTrain. You should be able to find `bw` tool in a sphinxtrain installation in a folder `/usr/local/libexec/sphinxtrain` (or under other prefix on Linux) or in `bin\Release` (in sphinxtrain directory on Windows). Copy it to the working directory along with the `map_adapt` and `mk_s2sendump` programs.

Now, to collect statistics, run:

```
./bw \

 -hmmdir en-us \

 -moddeffn en-us/mdef.txt \

 -ts2cbfn .ptm. \

 -feat 1s_c_d_dd \

 -svspec 0-12/13-25/26-38 \
```

```
-cmn current \

-agc none \

-dictfn cmudict-en-us.dict \

-ctlfn arctic20.fileids \

-lsnfn arctic20.transcription \

-accumdir .
```

Make sure the arguments in `bw` command should match the parameters in `feat.params` file inside the acoustic model folder. Please note that not all the parameters from `feat.param` are supported by `bw`, only a few of them. `bw` for example doesn't suppport `upperf` or other feature extraction params. You only need to use parameters which are accepted, other parameters from `feat.params` should be skipped.

For example, for continuous model you don't need to include the `svspec` option. Instead, you need to use just `-ts2cbfn .cont.` For semi-continuous models use `-ts2cbfn .semi.` If model has `feature_transform` file like en-us continuous model, you need to add `-lda feature_transform` argument to bw, otherwise it will not work properly.

Sometimes if you miss the file `noisedict` you also need an extra step, copy the `fillerdict` file into the directory that you choose in the `hmmdir`parameter, renaming it to `noisedict`.

Creating transformation with MLLR

MLLR transforms are supported by pocketsphinx and sphinx4. MLLR is a cheap adaptation method that is suitable when amount of data is limited. It's a good idea to use MLLR for online adaptation. MLLR works best for continuous model. It's effect for semi-continuous models is very limited since semi-continuous models mostly relies on mixture weights. If you want best accuracy you can combine MLLR adaptation with MAP adaptation below. On the other hand MAP requires a lot of adaptation data, it is not really practical to use it for continuous models. For continuous models MLLR is more reasonable.

Next we will generate an MLLR transformation which we will pass to the decoder to adapt the acoustic model at run-time. This is done with the mllr_solve program:

```
./mllr_solve \

    -meanfn en-us/means \

    -varfn en-us/variances \

    -outmllrfn mllr_matrix -accumdir .
```

This command will create an adaptation data file called `mllr_matrix`. Now, if you wish to decode with the adapted model, simply add `-mllr mllr_matrix` (or whatever the path to the mllr_matrix file you created is) to your pocketsphinx command line.

Updating the acoustic model files with MAP

MAP is different adaptation method. In this case unlike for MLLR we don't create a generic transform but update each parameter in the model. We will now copy the acoustic model directory and overwrite the newly created directory with adapted model files:

```
cp -a en-us en-us-adapt
```

To do adaptation, use the `map_adapt` program:

```
./map_adapt \

    -moddeffn en-us/mdef.txt \

    -ts2cbfn .ptm. \

    -meanfn en-us/means \

    -varfn en-us/variances \
```

```
    -mixwfn en-us/mixture_weights \

    -tmatfn en-us/transition_matrices \

    -accumdir . \

    -mapmeanfn en-us-adapt/means \

    -mapvarfn en-us-adapt/variances \

    -mapmixwfn en-us-adapt/mixture_weights \

    -maptmatfn en-us-adapt/transition_matrices
```

Recreating the adapted sendump file

If you want to save space for the model you can use sendump file supported by pocketsphinx. For sphinx4 you don't need that. To recreate the `sendump` file from the updated `mixture_weights` file:

```
./mk_s2sendump \

    -pocketsphinx yes \

    -moddeffn en-us-adapt/mdef.txt \

    -mixwfn en-us-adapt/mixture_weights \

    -sendumpfn en-us-adapt/sendump
```

Congratulations! You now have an adapted acoustic model! You can delete the files `en-us-adapt/mixture_weights` and `en-us-adapt/mdef.txt` to save space if you like, because they are not used by the decoder.

Other acoustic models

For Sphinx4, the adaptation is the same as for PocketSphinx, except that sphinx4 can not read binary compressed mdef and sendump files, you need to leave mdef and mixture weights.

Testing the adaptation

After you have done the adaptation, it's critical to test the adaptation quality. To do that you need to setup the database similar to the one used for adaptation. To test the adaptation you need to configure the decoding with the required paramters, in particular, you need to have a language model `<your.lm>`. For more details see Building Language Model. The detailed process of testing the model is covered in other part of the tutorial.

You can try to run decoder on the original acoustic model and on new acoustic model to estimate the improvement.

Using the model

After adaptation, the acoustic model is located in the folder

```
en-us-adapt
```

You need only that folder. The model should have the following files:

```
mdef

feat.params

mixture_weights

means

noisedict

transition_matrices

variances
```

Depending on the type of the model you trained.

To use the model in pocketsphinx, simply put the model files to the resources of your application. Then point to it with the `-hmm` option:

```
pocketsphinx_continuous -hmm <your_new_model_folder> -lm <your_lm> -dict
<your_dict> -infile test.wav
```

Or with `-hmm` engine configuration option through `cmd_ln_init` function. Alternatively you can replace the old model files with the new ones.

To use the trained model in sphinx4, you need to update the model location in the code.

Adaptation didn't improve results. Troubleshooting

Now test your accuracy to see it's good.

I have no idea where to start looking for the problem

1. test accuracy on adaptation set if it improves
2. accuracy improves on adaptation set → check if your adaptation set match with your test set
3. accuracy didn't improve on adaptation set → you made mistake during adaptation

or how much improvement I might expect through adaptation

From few sentences you should get about 10% relative WER improvement.

I'm lost

whether it's needing more/better training data, whether I'm not doing the adaptation correctly, whether my language model is the problem here, or whether there is something intrinsically wrong with my configuration

Most likely you just ignored error messages that were printed to you. You obviosly need to provide more information and give access to your experiment files in order to get more definite advise.

What next

We hope adapted model give you acceptable results. If not, try to improve your adaptation process:

1. Add more adaptation data
2. Adapt your language model / use better language model

Page Tools

- Show pagesource
- Old revisions
- Backlinks
- Back to top