Building Language Model

There are several types of models that describe language to recognize - keyword lists, grammars and statistical language models, phonetic statistical language models. You can chose any decoding mode according to your needs and you can even switch between modes in runtime.

Keyword lists

Pocketsphinx supports keyword spotting mode where you can specify the keyword list to look for. The advantage of this mode is that you can specify a threshold for each keyword so that keyword can be detected in continuous speech. All other modes will try to detect the words from grammar even if you used words which are not in grammar. The keyword list looks like this:

```
oh mighty computer /1e-40/
hello world /1e-30/
other phrase /1e-20/
```

Threshold must be specified for every keyphrase. For shorter keyphrase you can use smaller thresholds like 1e-1, for longer threshold must be bigger, up to 1e-50. If your keyphrase is very long, larger than 10 syllables, it is recommended to split it and spot for parts separately. Threshold must be tuned to balance between false alarms and missed detections, the best way to tune threshold is to use a prerecorded audio file. Tuning process is the following:

1. Take a long recording with few occurrences of your keywords and some other sounds. You can take a movie sound or something else. The length of the audio should be approximately 1 hour
2. Run keyword spotting on that file with different thresholds for every keyword, use the following command:
3.
   ```
         pocketsphinx_continuous -infile <your_file.wav> -keyphrase <"your keyphrase"> -kws_threshold \
               <your_threshold> -time yes
   ```

   It will print many lines, some of them are keywords with detection times and confidences. You can also disable extra logs with `-logfn your_file.log` option to avoid clutter.

4. From keyword spotting results count how many false alarms and missed detections you've encountered
5. Select the threshold with smallest amount of false alarms and missed detections

For the best accuracy it is better to have keyphrase with 3-4 syllables. Too short phrases are easily confused.

Keyword lists are supported by pocketsphinx only, not by sphinx4.

Grammars

Grammars describe very simple type of the language for command and control, and they are usually written by hand or generated automatically within the code. Grammars usually do not have probabilities for word sequences, but some elements might be weighed. Grammars could be created with JSGF format and usually have extension like .gram or .jsgf.

Grammars allow to specify possible inputs very precisely, for example, that certain word might be repeated only two or three times. However, this strictness might be harmful if your user accidentally skips the words which grammar requires. In that case whole recognition will fail. For that reason it is better to make grammars more relaxed, instead of phrases list just the bag of words allowing arbitrary order. Avoid very complex grammars with many rules and cases, it just slows the recognizer, you can use simple rules instead. In the past grammars required a lot of effort to tune them, to assign variants properly and so on. The big VXML consulting industry was about that.

Language models

Statistical language models describe more complex language. They contain probabilities of the words and word combinations. Those probabilities are estimated from a sample data and automatically have some flexibility. For example, every combination from the vocabulary is possible, though probability of such combination might vary. For example if you create statistical language model from a list of words it will still allow to decode word combinations though it might not be your intent. Overall, statistical language models are recommended for free-form input where user could say anything in a natural language and they require way less engineering effort than grammars, you just list the possible sentences. For example, you might list numbers like “twenty one” and “thirty three” and statistical language model will allow “thirty one” with certain probability as well.

Overall, modern speech recognition interfaces tend to be more natural and avoid command-and-control style of previous generation. For that reason most interface designers prefer natural language recognition with statistical language model than old-fashioned VXML grammars.

On the topic of design of the VUI interfaces you might be interested in the following book: It's Better to Be a Good Machine Than a Bad Person: Speech Recognition and Other Exotic User Interfaces at the Twilight of the Jetsonian Age by Bruce Balentine

There are many ways to build the statistical language models. When your data set is large, there is sense to use CMU language modeling toolkit. When a model is small, you can use an online quick web service. When you need specific options or you just want to use your favorite toolkit which builds ARPA models, you can use it.

Language model can be stored and loaded in three different format - text ARPA format, binary format BIN and binary DMP format. ARPA format takes more space but it is possible to edit it. ARPA files have `.lm` extension. Binary format takes significantly less space and faster to load.

Binary files have `.lm.bin` extension. It is also possible to convert between formats. DMP format is obsolete and not recommended.

Building a grammar

Grammars are usually written manually in JSGF format:

```
#JSGF V1.0;
/**
 * JSGF Grammar for Hello World example
 */
grammar hello;
public <greet> = (good morning | hello) ( bhiksha | evandro | paul | philip |
rita | will );
```

For more information on JSGF format see the full documentation on W3C

http://www.w3.org/TR/jsgf/

Building a Statistical Language Model

Text preparation

First of all you need to prepare a large collection of clean texts. Expand abbreviations, convert numbers to words, clean non-word items. For example to clean Wikipedia XML dump you can use special python scripts like https://github.com/attardi/wikiextractor. To clean HTML pages you can try http://code.google.com/p/boilerpipe a nice package specifically created to extract text from HTML

For example on how to create language model from Wikipedia texts please see

http://trulymadlywordly.blogspot.ru/2011/03/creating-text-corpus-from-wikipedia.html

Once you went through the language model process, please submit your langauge model to CMUSphinx project, we'd be glad to share it!

Movie subtitles are good source for spoken language.

Language modeling for many languages like Mandarin is largely the same as in English, with one addditional consideration, which is that the input text must be word segmented. A segmentation tool and associated word list is provided to accomplish this.

Using other Language Model Toolkits

There are many toolkits that create ARPA n-gram language model from text files.

Some toolkits you can try:

- IRSLM
- MITLM
- SRILM

If you are training large vocabulary speech recognition system, the language model training is outlined in a separate page Building a large scale language model for domain-specific transcription.

Once you created ARPA file you can convert the model to binary format if needed.

ARPA model training with SRILM

Training with SRILM is easy, that's why we recommend it. Morever, SRILM is the most advanced toolkit up to date. To train the model you can use the following command:

```
ngram-count -kndiscount -interpolate -text train-text.txt -lm your.lm
```

You can prune the model afterwards to reduce the size of the model

```
ngram -lm your.lm -prune 1e-8 -write-lm your-pruned.lm
```

After training it is worth to test the perplexity of the model on the test data

```
ngram -lm your.lm -ppl test-text.txt
```

ARPA model training with CMUCLMTK

You need to download and install cmuclmtk. See CMU Sphinx Downloads for details.

The process for creating a language model is as follows:

1) Prepare a reference text that will be used to generate the language model. The language model toolkit expects its input to be in the form of normalized text files, with utterances delimited by `<s>` and `</s>` tags. A number of input filters are available for specific corpora such as Switchboard, ISL and NIST meetings, and HUB5 transcripts. The result should be the set of sentences that are bounded by the start and end sentence markers: <s> and </s>. Here's an example:

```
<s> generally cloudy today with scattered outbreaks of rain and drizzle
persistent and heavy at times </s>
<s> some dry intervals also with hazy sunshine especially in eastern parts in
the morning </s>
<s> highest temperatures nine to thirteen Celsius in a light or moderate
mainly east south east breeze </s>
<s> cloudy damp and misty today with spells of rain and drizzle in most
places much of this rain will be
```

```
light and patchy but heavier rain may develop in the west later </s>
```

More data will generate better language models. The `weather.txt` file from sphinx4 (used to generate the weather language model) contains nearly 100,000 sentences.

2) Generate the vocabulary file. This is a list of all the words in the file:

```
    text2wfreq < weather.txt | wfreq2vocab > weather.tmp.vocab
```

3) You may want to edit the vocabulary file to remove words (numbers, misspellings, names). If you find misspellings, it is a good idea to fix them in the input transcript.

4) If you want a closed vocabulary language model (a language model that has no provisions for unknown words), then you should remove sentences from your input transcript that contain words that are not in your vocabulary file.

5) Generate the arpa format language model with the commands:

```
% text2idngram -vocab weather.vocab -idngram weather.idngram <
weather.closed.txt
% idngram2lm -vocab_type 0 -idngram weather.idngram -vocab \
     weather.vocab -arpa weather.lm
```

6) Generate the CMU binary form (BIN)

```
sphinx_lm_convert -i weather.lm -o weather.lm.bin
```

The CMUCLTK tools and commands are documented at The CMU-Cambridge Language Modeling Toolkit page.

Building a simple language model using web service

If your language is English and text is small it's sometimes more convenient to use web service to build it. Language models built in this way are quite functional for simple command and control tasks. First of all you need to create a corpus.

The “corpus” is just a list of sentences that you will use to train the language model. As an example, we will use a hypothetical voice control task for a mobile Internet device. We'd like to tell it things like “open browser”, “new e-mail”, “forward”, “backward”, “next window”, “last window”, “open music player”, and so forth. So, we'll start by creating a file called `corpus.txt`:

```
open browser
new e-mail
forward
backward
next window
last window
open music player
```

Then go to the page http://www.speech.cs.cmu.edu/tools/lmtool-new.html. Simply click on the "Browse…" button, select the `corpus.txt` file you created, then click "COMPILE KNOWLEDGE BASE".

The legacy version is still available online also here: http://www.speech.cs.cmu.edu/tools/lmtool.html

You should see a page with some status messages, followed by a page entitled "Sphinx knowledge base". This page will contain links entitled "Dictionary" and "Language Model". Download these files and make a note of their names (they should consist of a 4-digit number followed by the extensions `.dic` and `.lm`). You can now test your newly created language model with PocketSphinx.

Converting model into binary format

To quickly load large models you probably would like to convert them to binary format that will save your decoder initialization time. That's not necessary with small models. Pocketsphinx and sphinx3 can handle both of them with `-lm` option. Sphinx4 automatically detects format by extension of the lm file.

ARPA format and BINARY format are mutually convertable. You can produce other file with `sphinx_lm_convert` command from sphinxbase:

```
sphinx_lm_convert -i model.lm -o model.lm.bin
sphinx_lm_convert -i model.lm.bin -ifmt bin -o model.lm -ofmt arpa
```

You can also convert old DMP models to bin format this way.

Using your language model

This section will show you how to use, test, and improve the language model you created.

Using your language model with PocketSphinx

If you have installed PocketSphinx, you will have a program called `pocketsphinx_continuous` which can be run from the command-line to recognize speech. Assuming it is installed under `/usr/local`, and your language model and dictionary are called `8521.dic` and `8521.lm` and placed in the current folder, try running the following command:

```
pocketsphinx_continuous -inmic yes -lm 8521.lm -dict 8521.dic
```

This will use your new language model and the dictionary and default acoustic model. On Windows you also have to specify the acoustic model folder with `-hmm` option

```
bin/Release/pocketsphinx_continuous.exe -inmic yes -lm 8521.lm -dict 8521.dic
-hmm model/en-us/en-us
```

You will see a lot of diagnostic messages, followed by a pause, then “READY…”. Now you can try speaking some of the commands. It should be able to recognize them with complete accuracy. If not, you may have problems with your microphone or sound card.

Using your language model with Sphinx4

In Sphinx4 high-level API you need to specify the location of the language model in Configuration:

```
configuration.setLanguageModelPath("file:8754.lm");
```

If the model is in resources you can reference it with resource: URL

```
configuration.setLanguageModelPath("resource:/com/example/8754.lm");
```

See Sphinx4 tutorial for details

tutoriallm.txt · Last modified: 2016/07/27 14:14 by admin

Page Tools

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