Language Models for WaveSurfer ASR Plugin

Michael Meyer
meye@kth.se

Daniel Carballal
carballal@kth.se

June 4, 2016

Abstract

We investigated and analyzed the use of different n-gram language models and heuristics to improve the WaveSurfer ASR plugin, which uses the Julius ASR software as a backend. We compared the performance of the different language models on the TIMIT corpus transcriptions, using the statistical modeling tools provided by the CMU SLM Toolkit.

1 Introduction

WaveSurfer is a program used for sound manipulation and visualisation. It includes tools to generate spectrograms, waveforms, display aligned phoneme and word transcriptions, and more. Since speech corpora are expensive to purchase and create, it is of interest to create a tool that performs automatic speech transcription and alignment, allowing for linguists and speech researchers to have access to resources otherwise not available. It is also of interest to create programs that are easy for non-technical users to learn.

We have based our work off the WaveSurfer ASR plugin Salvi et al (2014), which uses the open-source Julius ASR program. This langauge independent program requires an acoustic model, language model, and dictionary to work correctly. These requirements are described in detail below.

The acoustic model judges the probability the waveform being a certain phoneme. It uses hidden markov models with gaussian emissions. This part of the model was already provided by the WaveSurfer ASR plugin, and had been trained on oral readings from the Washington Post.

The dictionary is the transcription of words into phonemes. It is a simple map between a word and a list of phonemes. It is needed to bring the acoustic and language models together. This was also already provided by the WaveSurfer ASR plugin.

The language model determines the probability of a word sequence. Julius has two different flavors of language models: grammars and statistical language models. Grammars are context-free grammars that describe the set of sentences the recognizer should accept. This approach outputs a binary signal describing if a sentence was accepted or not. It has no way of comparing two valid sentences which puts pressure on the acoustic model to differentiate between utterances.

Statistical language models capture a richer representation of the sentences by creating a probablistic representation of utterances using n-grams. N-grams return the probability of a word following N-1 previous words. They are trained on text corpora, and are particularly susceptible to errors due to a lack of data. If a word does not appear in the corpus, the n-gram will report there is no chance of the word existing, which is plain wrong. Therefore, different heurstics have been created to mitigate this problem, as we will see in this paper. The Julius ASR program accepts 2-grams and 3-grams as input, so we chose to create 3-gram models.

The WaveSurfer ASR plugin came with a basic grammar that validated any sequence of words that was present in the dictionary. Our research involved changing the grammar to a statistical language model and then comparing the performance between the grammar and a group of different heuristics.

2 Method

Statistical language models are generated from text corpora. Once provided a text corpus, there are many tools available to compute various features of the data. We chose to use the CMU Statistical Langue Modeling Toolkit Clarkson et al (1997). This toolkit has the ability to generate language models with different discounting heuristics.

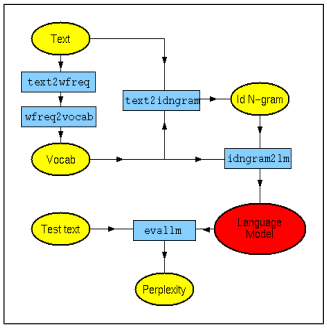

Figure 1: Typical SLM UsageClarkson et al (1997) (Taken from tookit documentation)

The steps to generate a language model are described below:

The user first generates word unigram counts using text2wfreq.

After, the user generates the vocabulary file, which is the listing of all words line by line, using wfreq2vocab.

Then, the user creates an idngram file with the command text2idngram.

This output is used in conjunction with idngram2lm, which takes in the discounting strategy and outputs a valid arpa file, which can then be used in the Julius ASR program.

We ran this process for all of the discounting strategies: Good Turing, Witten Bell, Absolute, and Linear.

- Good Turing: This method, devoloped by Alan Turing, defines the discounting factors as:

  $$d(r) = (r+1)S(c+1)/rS(c)$$

  for all words with less than K occurences (a value we tried manipulating). S is the smoothing pattern and c is the count. In other iterations, S may be a log function, but we used a linear function of words seen instead.
- Witten Bell: The probability curve for Witten Bell is defined as:

  $$P(w) = c/(n+t) \text{for seen words}$$

  $$P(w) = t/(n+t) \text{for unseen ones}$$

  and t is the number of types which followed a particular context.
- Absolute: Absolute discounting is defined as:

  $$d(r) = (r-b)/r$$

  with

  $$b = n(1)/(n(1)+2n(2))$$

  Note that this is subtracting a constant from all values.
- Linear: Linear discounting defines $d(r) = 1 - (n(1)/C)$, where C is the total number of events.

We also had three varients of each of the discounting strategies using three different methods for out-of-vocabulary (OOV) errors:

- Normal: Words that aren't found are mapped to the one symbol that has a given probabilty.

- Closed: Do not consider words that aren't in the training set.
- Test-Only: ignore OOV words in training, but allow in testing.

With the generated language models, we ran the Julius ASR program on 100 test utterances. We computed the Word Error Rate, otherwise known as the Levenshtein distance, by computing the following formula for each sentence:

$$E = \frac{S + D + I}{N}$$

Where

- S is the number of substitutions,
- D is the number of deletions,
- I is the number of insertions,
- N is the number of words in the reference text

Note that a sentence can easily end up with an error rate higher than 100% by outputting a string that is significantly longer than the target string.

3 Experiments

We used the TIMIT dataset transcriptions for our text corpus. This speech corpora is of American English, and has slurred and unclear speech in it. As natural American English speakers, we had trouble understanding some of the utterances the first time around. We simplified the dataset by removing all punctuation and capitalization in letters, and likewise for the test transcriptions during the testing phase.

The Julius ASR has two passthroughs. Sometimes the second passthrough fails, unable to find the best sentence. We chose to output the first pass, which is what is done in the ASR plugin. This allows us to test all outputs.

We ran Julius with the following parameters. (Where LM.arpa is the language model)

```
julius --fallback1pass -h mfcc.mmf -hlist mfcc.lst -nlr {LM.arpa}
-v vocab.dict -input rawfile -filelist flist
```

Since the baseline uses a grammar, julius is run with this commands instead.

```
julius --fallback1pass -h hmm.mff -hlist hmm.lst -gram gramfile
-input rawfile -filelist flist
```

Other options can be added to improve the model, but they have not been used to simplify the comparison.

4 Results

Table 1: Language Model Heuristics

Out-Of-Vocab Rule	Discount Strategy	Word Error Rate (%)
Grammar Reference		134.7
Basic 3-gram		87.3
Normal	Linear	87.6
	Absolute	86.2
	Good Turing	87.3
	Witten Bell	87.6
Test-Only	Linear	95.6
	Absolute	97.3
	Good Turing	87.3
	Witten Bell	98.3
Closed	Linear	—
	Absolute	—
	Good Turing	—
	Witten Bell	—

The closed language models above failed because there were test words that were not present in the training data, so we were unable to score them.

5 Discussion and Conclusions

The statistical language models performed much better on the test set when compared to the basic grammar. However, the different discounting strategies had little effect compared to the OOV rules. This is probably because of the corpus we used. TIMIT is relatively small for text, and the test data uses mostly the same words as the training data. While we were able to train on the entire TIMIT training set, since ASR is computationally expensive we were only able to test on 100 sentences given the time we had, which should be taken into account. It is worth noting that the TIMIT data set is intentionally difficult– even native speakers can have a hard time parsing what the speaker is saying. Thus, a word error rate of 80% is not totally suprising.

In addition, the acoustic models were trained off a spoken version of the Washington Post. This may affect the acoustic model's ability because the the clear and precise reading aloud of the paper is most likely opposite of the slurred speech of the TIMIT dataset. Therefore, we shouldn't expect too high of a performance.

It would be interesting to perform the same experiments on a larger corpus, such as the Gigaword corpus Vertanen (2003). It would also be interesting to compare different corpora with eachother, such as different newspaper publications. Different languages could also be studied. Different n-gram models sizes should also behave differently.

As a result of our research, any of the language models can now be incorporated into the WaveSurfer plugin to achieve better results when parsing English.

References

Clarkson, P.R. "Statistical Language Modeling Using the CMU-Cambridge Toolkit" *Proceedings ESCA Eurospeech* (1997)

Akinobu, L. "The Julius Book" (2008) `https://julius.osdn.jp/juliusbook/en/`

Salvi, G. Vanhainen, N. "The WaveSurfer Automatic Speech Recognition Plugin" *Proceedings of LREC* (2014)

Vertanen, K. "English Gigaword language model training recipe" *Retrieved from* `https://www.keithv.com` (2003)