Comparison between grammar-based and n-gram language models

Aleix Sacrest Gascon
KTH Royal Institute of Technology
aleixsg@kth.se

Quim Arnau Ortega
KTH Royal Institute of Technology
quima@kth.se

Abstract

Statistical language models (SLM) are usually the choice for broad speech recognition when is necessary to deal with large vocabularies, whereas grammar-based models (GLM) are used when dealing with smaller vocabularies, often specified by hand and consisting of a few set of rules. In this paper we want to show experimentally that this is the case in the specific human-computer interaction (HCI) problem we are addressing: the recognition of a small set of terminal commands. We also check whether open-source toolkits will allow us to build such recognition system in a relatively short time to assess both models. In conclusion, our experimental work showed that GLMs clearly outperformed SLMs in this context.

1 Introduction

Over the last decade, speech technology has gained a lot more attention changing the way we interact with machines on everyday life. This is due to three main reasons: *i)* computational power keeps raising, *ii)* more data is generated and available for everyone, *iii)* smart devices are everywhere. This technology plays an important role in two applications: human-human communication (HHC) and human-computer interaction (HCI) [1, Chapter 1, Section 1.1].

However, as striking as it can seem, the field of speech recognition does not seem to be as trendy as one would think in the moment, according to web search indices from Google Trends [2]. It actually has been decreasing since 2004, but we know how important this field is becoming, we just need to be aware of the recent huge efforts made in creating voice-activated personal assistants like *Siri* (Apple), *Alexa* (Amazon) or *Google Home* (Google).

Automatic speech recognition (ASR) requires language models for two main reasons: *i)* complement acoustic models, *ii)* reduce the branching factor during recognition. In figure 1, extracted from [1, Fig. 1.3, Chapter 1, Section 1.2], we can see how these models fit into the architecture of an ASR system.

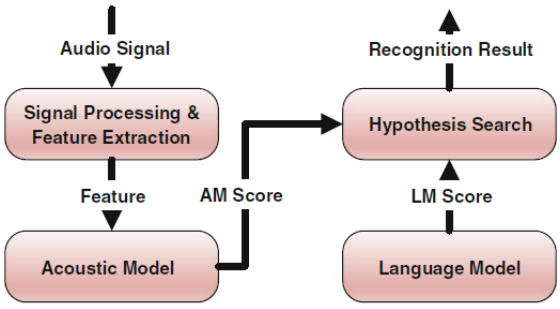

Figure 1: Usual ASR systems architecture

.

As stated in [3], "*SLMs perform extremely well when there is adequate training data available, but in practice this is not always the case. When training data is limited or, in the worst case, completely unavailable, an alternative method is to construct the language model as a hand-coded grammar.*" That is also what we expect to experimentally see in the problem we are addressing: GLMs outperforming SLMs, because the set of commands is sufficiently small to be coded in a grammar, and it would be hard to find a corpus that might properly train an SLM for this purpose. However this is our hypothesis and not any assumption made before hand.

In [4] a comparison between GLMs and SLMs was also done training the models with the same data using a different method. The language analyzed was used in the Clarissa procedure navigator, a spoken dialog system used in the International Space Station (ISS). However the problem addressed there was broader in the sense that the language assessed was way more richer than a reduced set of commands and also its evaluation was more rigorous, taking into account semantic and sentence error rates. The results obtained revealed that GLMs were a better choice than SLMs having the former lower error rates.

In the following subsections we are going to state our goals and define n-gram language models. In section 2 we discuss the method we are going to follow. The experimental data recorded will be explained in section 3. In section 4 all results will be shown and discussed in section 5, where we will also state our conclusions and possible further work to be done in the future.

1.1 Goals pursued

The main purpose of this project is to build a recognition system in real time that lets the user interact with the computer in a simple manner. Our idea resides in checking whether state-of-the-art speech technology would allow us to create such system using open-source toolkits in short time. This study serves also to: *i)* focus on the importance of language models in speech recognition and *ii)* compare grammar-based models (GLM) to statistical language models (SLM).

We are going to build statistical language models using *SRILM*, and interact with the *Julius* dialog manager with its speech-to-text features, as it was also done in [5], evaluating both SLMs and GLMs.

1.2 About n-gram language models

An **n-gram** is a sequence of n symbols, e.g., letters, phonemes, syllables, words, syntactic categories, etc. An n-gram language model is used to predict each symbol in the sequence given its *n-1* predecessors, assuming that its probability of occurrence in a test text can be estimated from the frequency of occurrence in a given training text [6, Chapter 14].

The probability of a word sequence $w_1, ..., w_m$ is modeled as:

$$\hat{P}(w_1, ..., w_m) = \prod_{i=1}^{m} \hat{P}(w_i \,|\, w_1, \dots, w_{i-1}) \;\approx\; \prod_{i=1}^{m} \hat{P}(w_i \,|\, w_{i-n+1}, \dots, w_{i-1}) \tag{1}$$

Where the approximation comes from limiting the context for some $n \geq 1$. As mentioned before, probability estimates in these models are based on maximum likelihood estimates, i.e., based in counting occurrences in a training text as follows, where *C* stands for the count function:

$$\hat{P}(w_i \,|\, w_{i-n+1}, \dots, w_{i-1}) \;=\; \frac{C(w_{i-n+1}, \dots, w_i)}{C(\, w_{i-n+1}, \dots, w_{i-1})} \tag{2}$$

However, a problem arises when a sequence of words does not appear in the training corpora. Then its occurrence probability is zero, which leads the recognizer to disregard it, no matter what the acoustic model said. That is when **smoothing** comes in, trying to build a more robust model that produce less extreme probabilities for unseen data. One of the simplest methods is the additive smoothing technique, which assumes that every possible word sequence is seen at least a δ number of times in the training corpus [7]. Using this simple method we do not have zero probabilities anymore. But more complex methods can be used such as interpolation or back-off models, Katz, Knesser-Ney or Good-Turing smoothing techniques, among others. An introduction to these methods can be found in [6, Chapter 14, Section 14.3] and [7, 8].

2 Method

In order to build statistical language models we are using *The SRI Language Modeling* (SRILM) a toolkit primarily used for speech recognition, statistical tagging and segmentation, as well as machine translation (sources available in [9]). We are specifically using the **ngram-count** command which generates and manipulates n-gram counts, and estimates n-gram language models from them (see manual page for further details in [10]).

We are also using *Julius*, an open-source large vocabulary continuous speech recognition engine based on GMM-HMMs. Because we are not focusing on acoustic models, we are using an already trained model from HTK that uses *MFCC_O_D* acoustic features, available from *VoxForge* in [11]. This toolkit will allow us to perform speech recognition both with statistical language models and with grammar-based models. We will also be able to do the recognition in real time using its speech-to-text features using a microphone or set the input to be a recorded utterance [12, Chapter 3, Sections 3.1-3.3].

2.1 Language we want to recognize

We have created a language for an easy interaction with the computer which allows to perform simple actions such as setting volume options, launching applications or creating directories. The language is shown in table 1 using regular expressions. Six different types of commands are recognized:

- Open the text editor emacs, a new tab in google chrome browser with a newspaper webpage (`www.ara.cat`), a new terminal window or the file system.
- Make or remove a directory in the home directory with the name *newdir*.
- Launching the command *cmatrix* in a new terminal.
- Showing statistics of the system with the command *top*.
- Rising and lowering the volume.
- Dictation, which allows the user to dictate numbers from one to ten. This command has to start with the word *dictate* followed by any sequence of numbers and the word *final* when the dictation is finished. The sequence of numbers will be written in the file *dictate.txt*.

Table 1: Commands to recognize

1	open (emacs | newspaper | terminal | file system)
2	(make | remove) directory
3	matrix
4	top
5	(raise | lower) volume
6	dictate $(\text{digit})^+$ final

2.2 Building statistical language models

We have trained several language models with different corpora. Our idea is to extract Wikipedia text entries related to HCI, terminal commands, and specifically to the ones we wanted to recognize, i.e., entries like *File system*, *Human computer interaction*, *Emacs*, among others. After that we preprocessed the texts in a simple manner: keeping only characters from the alphabet, not numbers, and converting them to lowercase. Finally it was necessary to perform the intersection between the words in the text and a dictionary of English words with its phonetic transcription, because *Julius* will need this dictionary to do the recognition. For this purpose, we used the dictionary provided with the acoustic model we are using (from VoxForge) and we added some specific words that we needed, such as *emacs*. And the texts are ready to be used to build the statistical language model.

Once we train a language model using SRILM, we need to convert the resulting file from ARPA standard format to a compact Julius binary format using **mkbingram**, a command from Julius.

2.3 Creating a grammar-based model

Grammar-based models specify a closed language using a set of rules defined over terminal and nonterminal symbols. Terminal symbols are those which can be found in the language, whereas nonterminals are used to define the rules. According to the syntax used, the grammar allows up to context-free level, but since *Julius* uses a DFA as a parser only regular expression grammar will be accepted during compilation. This implies that recursions can be expressed but only on the left hand side.

To define a grammar-based language model in *Julius* two files are needed, with extensions *.grammar* and *.voca*. In the first file the regular grammar is defined and in the second one there is a list of all terminal symbols taking part in the grammar.

Once the grammar is defined with the two files specified above it needs to be complied. The compilation takes place with the *perl* program *mkdfa.pl* which is included when the *Julius* binaries are downloaded. This process outputs the files that will be used for *Julius* during its execution, these are the *.dfa* and *.dict* files, which need to be specified in the *Julius'* configuration file. The grammar is defined in the *.grammar* as shown in table 2.

Table 2: Grammar definition

1	S : NS_B INS NS_E
2	INS: OPEN OPARAM
3	INS: OPEN FILE SYSTEM
4	INS: DIRMANAGE DIR
5	INS: COMMAND
6	INS: DICTATE DIC
7	INS: FACTOR NUM
8	INS: VOLUMEOPTS VOLUME
9	DIC: NUMBERS FINAL
10	NUMBERS: NUMBERS NUM
11	NUMBERS: NUM

The symbols on the left side are the nonterminals, whereas the symbols which appear on the right side and not in the left are defined in the *.voca* file with terminal symbols. The symbols NS_B and NS_E are the silences of start and end of sentence respectively. All grammars defined for *Julius* have to contain the definition of a nonterminal symbol S which is where the grammar starts, in this case this symbol is defined as $start\_silence\ instruction\ end\_silence$. Where $instruciton$ represents the set of all different instructions that can be understood by the dialog manager, and in the grammar are defined by the nonterminal symbols INS. One example of this is *INS: OPEN OPARAM*, where *OPEN* can only be substituted by the terminal symbol "open" and *OPARAM* can be substituted by the terminal symbols "newspaper", "emacs" and "terminal". All this is defined in the *.voca* file.

3 Experiments

The experiments are designed taking into account the language of commands we created for the interaction with the computer, see subsection 2.1. He have selected 12 commands to experiment with different combinations of phonemes allowed by them. The commands recorded are shown in table 3.

Table 3: Commands used in the experiments

1	open emacs
2	open terminal
3	make directory
4	dictate one two three final
5	raise volume
6	top
7	open newspaper
8	open file system
9	remove directory
10	dictate five five five final
11	lower volume
12	matrix

A total of 168 utterances by 7 non-native English male and female speakers from different nationalities were recorded in a noisy environment with a humble laptop recorder[1]. The recordings were made using **arecord** in linux, specifying them to be monoaural (1 single channel used), sampling rate of 16KHz and in *wav* format. Each command is recorded twice by each speaker. Almost all utterances last 3 seconds, except for dictations, which last 5.

We evaluate the recognition of all utterances using the metric Word Error Rate (WER), disregarding other possibilities as sentence or semantic error rates because the former is easy to compute, widely used in speech technologies and machine translation systems, as well as one of the primary metrics used to estimate the performance of language models in speech recognition systems [13, 14]. Word error rates represent how different is what the speech recognizer interprets from the actual transcription of the input. This is non-trivial and is computed as follows:

$$WER = 100 \cdot \frac{S + D + I}{N}$$

Where,

- S is the number of substitutions
- D is the number of deletions
- I is the number of insertions
- N is the total number of words of the transcription

With such metric we can compute the accuracy as $A = 100 - WER$, which gives a measure of how good is the recognizer performing.

A quick test with a small set of texts revealed that the statistical language model trained was not the appropriate for such a specific task. Because our goal was to recognize a small set of commands, not probable to appear in natural English language, e.g., *make directory*, we discarded the statistical language model created from texts really soon, as the recognizer could not figure out any command unless it explicitly appeared in the texts, which most likely would not. Not even using smoothing or interpolation techniques we managed to recognize a single command.

Instead we created a list of possible commands and we used it as corpus to train an n-gram model of order 2, without smoothing as all the commands appeared in the list. With this configuration the SLM has better performance since the commands appear in the corpus. But notice that from that point on we do not have a statistical model anymore and the comparison is not strictly fair.

[1] Audio device: Advanced Micro Devices, Inc. [AMD/ATI] Kaveri HDMI/DP Audio Controller

4 Results

In this section we present the results obtained with the grammar we have created and the last version of the statistical language model, i.e., using a list of possible commands as the training data (as specified in the previous section).

We have computed the overall accuracy for each of the models and the results are shown in table 4.

Table 4: Overall accuracy by model

Model	Accuracy (%)
GLM	95.56
SLM	77.33

The commands chosen for the experiments contain different combinations of sounds, so that some may be easier to recognize than others. Moreover, taking into account that our set of speakers is compounded by non-native English speakers, the performance of the recognition of one command to another can vary. For this reason, we have arranged the results in order to classify the accuracy of each command recorded for each of the models. With this classification we aim to observe whether both models have similar results, e.g., the easiest/hardest command to recognize is the same for both. The results can be seen in table 5.

Table 5: Accuracy by command

Command	Accuracy (%)	
	GLM	SLM
open terminal	100.0	67.86
open emacs	100.0	92.29
open newspaper	100.0	100.0
raise volume	100.0	96.43
open file system	100.0	76.19
dictate five five five final	78.57	5.71
matrix	100.0	85.71
make directory	89.29	53.57
top	100.0	14.29
dictate one two three final	100.0	77.14
lower volume	100.0	67.86
remove directory	100.0	75.0

5 Discussion and Conclusions

The results show that the grammar-based language model clearly outperforms the statistical language model. This can be seen in table 4 where the overall accuracy is shown but it is also detailed in table 5 where we can observe the accuracy in recognition for each command. This last one shows that the accuracy for the GLM is higher than the SLM for all the commands.

This observations fulfill our initial hypothesis which was that for a closed *language*, such as a set of commands for the interaction with the computer, a GLM would get better results than a SLM. However, as mentioned in section 3, the SLM trained in the end was not a statistical model as it is known.

Another aspect we wanted to determine was whether the general performance, in terms of accuracy, for each command would be similar for both language models. This question is difficult to answer because for GLM most commands get a 100% accuracy whereas for SLM there is much variability. However, the coincidence is that the command getting the least accuracy for GLM is also the one getting the least for SLM. We believe this is because it was rather a large command and it contained the same word (*five*) three consecutive times.

5.1 Future work

Regarding future work for this application, developing different acoustic and language models in our own language, Catalan or Spanish, could be an interesting idea and right now *VoxForge* is trying to persuade users to get their recordings, but a lot of data is currently needed.

References

[1] D. Yu and L. Deng, *Automatic Speech Recognition: A Deep Learning Approach*. Springer Publishing Company, Incorporated, 2014.

[2] Google, "Web searches comparison over time between Speech recognition and Machine Learning on Google Trends." [Online]. Available: https://goo.gl/OHiIz1

[3] B. A. Hockey, M. Rayner, and G. Christian, *Training Statistical Language Models from Grammar-Generated Data: A Comparative Case-Study*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2008, pp. 193–204. [Online]. Available: http://dx.doi.org/10.1007/978-3-540-85287-2_19

[4] B. A. Hockey and M. Rayner, "Comparison of Grammar-Based and Statistical Language Models Trained on the Same Data," no. Nuance, 2005.

[5] A. Azzarone, "English N-Gram Language Models for Robotic Spatial Commands," Kungliga Tekniska Högskolan, Tech. Rep., 2016.

[6] S. Young, G. Evermann, D. Kershaw, G. Moore, J. Odell, D. Ollason, V. Valtchev, and P. Woodland, *The HTK book*, 2002, vol. 3, no. July 2000.

[7] S. F. Chen and J. T. Goodman, "An Empirical Study of Smoothing Techniques for Language Modeling," pp. 310–318, 1996. [Online]. Available: http://arxiv.org/abs/cmp-lg/9606011

[8] R. Levy, "Working with n-grams in SRILM," pp. 1–6, 2015.

[9] SRI International, "STAR Laboratory: SRI Language Modeling Toolkit." [Online]. Available: http://www.speech.sri.com/projects/srilm/

[10] ——, "Manual page for the ngram-count command," 2013. [Online]. Available: http://www.speech.sri.com/projects/srilm/manpages/ngram-count.1.html

[11] VoxForge, "Acoustic models for English," 2017. [Online]. Available: http://www.repository.voxforge1.org/downloads/Nightly%5C_Builds/AcousticModel-2017-05-03/

[12] A. Lee, *The Julius book*, 2010. [Online]. Available: http://julius.sourceforge.jp

[13] S. F. Chen, D. Beeferman, and R. Rosenfeld, "Evaluation metrics for language models." [Online]. Available: http://www.itl.nist.gov/iad/mig/publications/proceedings/darpa98/html/lm30/lm30.htm

[14] S. Seljan and I. Dunđer, "Automatic word-level evaluation and error analysis of formant speech synthesis for Croatian," pp. 172–178.