David Morley

Professor Deryle Lonsdale

Linguistics 361

10 December 2019

My First German Automated Speech Recognizer

Abstract

In many domains, building an automatic speech recognizer has become a rather solved problem. The accuracy and quality that has been attained in English were both only achieved after many years of optimization and using large training sets. While the technology and some open-source data sets exist, few people have made acoustic models or speech recognizers in German. I sought to remedy this situation by the Kaldi recipes prepared by Cobalt Speech Inc. and the opensource CommonVoice German dataset.

Introduction

Large advances in Automatic Speech Recognition (ASR) have been made in the last few decades. In the English language, companies like Amazon and Google have refined their models such that they can transcribe natural speech with near-perfect accuracy. However, the quality of speech models has not reached this level of performance in other languages. An important first step in attaining an ASR with any level of quality is to build both an Acoustic Model (AM) and a Language Model (LM) in the target domain. Using easy to obtain resources I sought to build a robust AM and preliminary LM to provide preliminary models for future growth in German language ASRs. The AM was built under the direction of and for future use by Cobalt Speech Inc. (Cobalt).

Cobalt is a contracting company that specializes in automated speech solutions.[1] Customers come to them with various problems relating to speech processing and dialogue analytics. They have an international clientele pool and therefore need to have quality speech models for various languages to facilitate their work for people in different countries. After a sporadic history of correspondence with the CEO of Cobalt, Jeff Adams, I contacted him to see if there was a project that I could work with his company on. Building an AM for German was an opportunity that fulfilled both our interests.

By no means is this the first time that a German acoustic model has been built. For example, in 2008 Benjamin Milde and Arne Köhn set out to make an ASR for German using only open-source materials and tools.[2] Their dataset consisted of a total of 412 hours of speech and 510 separate speakers. They used Kaldi implementations of a Gaussian Mixture Model (GMM), Hidden Markov Models (HMM), and Time-Delayed Neural Networks (TDNNs) as their main model building tools. The best model that they produced achieved an impressive word error rate (WER) of 14.4%.

Datasets

The main data set that I worked with on this project was the opensource German CommonVoice corpus. This contained approximately 350 hours of speech with 5,007 speakers from Austria, Germany, Liechtenstein, Russia, Switzerland, United States, France, Hungary, Netherlands, Poland, Slovakia, United Kingdom, and other countries. The speech was broken up into small segments usually consisting of a 3 – 10 second phrase. We assumed that all speakers were close (within 18 inches) to the microphone in a quiet environment.

[1] Cobaltspeech.com
[2] https://arxiv.org/pdf/1807.10311.pdf

After obtaining the data, I cleaned and prepared it for use with Kaldi recipes. The data was first divided into development (dev), training (train), and testing (test) groups. The goal was to have a random sample of all the data in each group while also taking care to get a good range of nationalities, genders, and age groups represented in each set. Within each group, information files were created from the data. First utterance keys were generated for each file called the uttID. These keys contained broad information about the nationality, gender, and age group of the speaker as well as unique digits for both the speaker and the utterance (for example, austria_F_thirties_000001-0002). The main files for each group were wav.scp (maps the uttID to the file location), text (maps the uttID to the transcript of the utterance), and utt2spk (maps the uttID back to the speaker).

All of the text was unified in an uppercase format. The initial code did not have set methods for changing diacritics and other special characters to uppercase. New methods were implemented separately to ensure this. More information about this step will come later. Both the lexicon and the language model were built from the text file in the training folder. This ensured that there would be relatively few out of vocabulary (OOV) words. However, it also meant that the language model was not very robust.

Methods

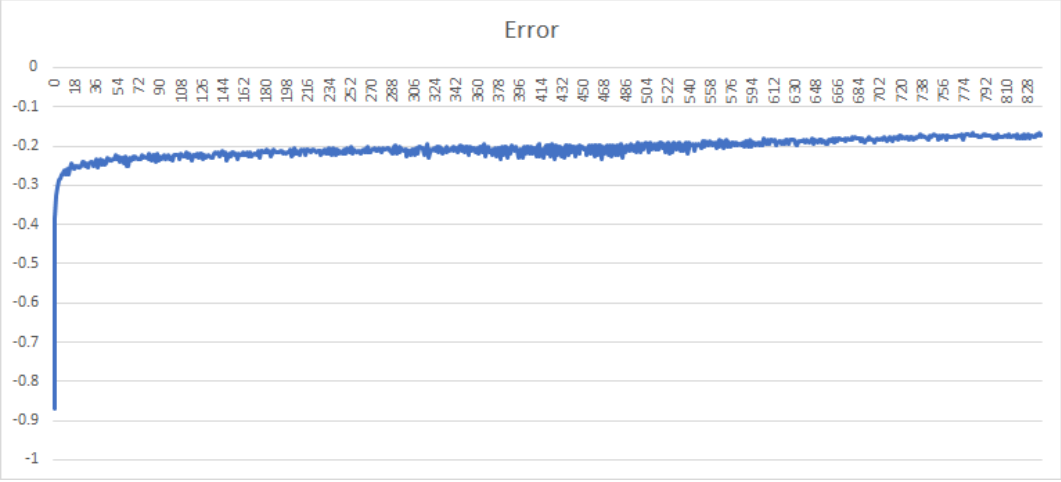
The first step in building the acoustic model was to train a Gaussian Mixture Model (GMM). This process involved a basic feature extraction using the Mel Frequency Cepstral Coefficients (MFCCs). These features were then used to iteratively train GMMs with monophones and then triphones. This data was used to better align the transcripts to the speech and to improve the probabilities of the different pronunciations in the lexicon. Using this preliminary model as a base, speaker adaptive training was performed three times using linear

regression. This computationally intensive process took about a day to run. This step essentially created an ASR and validation showed an initial WER of 30.9%.

With this preliminary ASR trained, more data was needed in order to make subsequent models more robust. Instead of finding and cleaning a new dataset, I ran a data augmentation step that took the present data and created modified versions of it. On certain of the new recordings, volume was randomized, speed was varied, echo was added, or random noise was included. This step focused on currently challenging areas of speech recognition with a goal to make the final product more adept at recognizing speech under these circumstances. This new data was (automatically) processed and stored identically to the original data. New transcripts alignments and language model trees were also generated to include the new additions.

With the new dataset, everything was now prepared to run the data and models through a deep neural network (DNN) for further training. DNNs are a subset of machine learning that enables a computer to find the best way to "learn" how to perform a specific task. In this case, the learning entailed minimizing the error on between the transcribed text and the true transcript. The DNN chosen for this had 5 epochs and 840 iterations in its training period. At each step, progress was measured through computing the average log probability and per-frame accuracy on the validation set.

Figure 1.



Plot of per-frame accuracy after each iteration.

As Figure 1 shows, the error generally decreased over successive interactions of the DNN. Running the complete DNN took a little over a day to complete. The fact that the line did not plateau or spike shows that our model did not overfit to the supplied data. This is desired as it ensures that the model will be better suited to function with new "real world" inputs.

The final step of the model creation process was to decode and evaluate the model. The format that enabled the computer to process and train with the data is not best suited for consumer use. To facilitate future use, a decoding step had to happen first. I ran a Cobalt script that condensed, stored, and backed up the generated acoustic model. This step allowed the company to create a demo of the model that performs near-simultaneous German speech transcription. Also included in this script was a method to test the model against various metrics.

Figure 2.

hyp								
SPKR	# Snt	# Wrd	Corr	Sub	Del	Ins	Err	S.Err
Sum/Avg	28510	217570	70.2	14.5	15.3	3.3	33.1	68.0
Mean	65.1	496.7	30.2	46.3	23.4	13.2	82.9	87.3
S.D.	340.6	2599.8	40.8	34.2	17.1	20.8	53.2	21.8
Median	9.5	69.0	3.0	60.2	21.3	8.5	105.6	100.0

Sample output of final model test.

Discussion of the Results

As shown in Figure 2, the average WER across all of the recorded test utterances was 33.1% after running the DNN. This showed a marked decrease in accuracy after running the GMM when the WER was only 30.9%. Close analysis showed that this lackluster result was caused in the initial cleaning steps. As mentioned earlier, the code that I modified uniformly cleaned the data set by changing all transcript text to uppercase. However, the method used was not compatible with diacritics. I was not very experienced with Bash scripting in UNIX before

this project and so in my initial attempts to correct this bug, I somehow managed to wipe all the transcripts. I assumed based on the advice I received from those I was working with, that the text in the file mapping utterance to speakers was sorted the same as the specific text file. We discovered after the fact that it wasn't, so when I went in to correct my error, I actually misaligned the utterances from their speakers. Therefore, the model was trained to believe that some speech samples were saying completely inaccurate things. This mistake went unnoticed until the evaluation performed after the final training had been completed, leaving me with a model trained on faulty data and no time left to fix it.

Further, the language model used along with the acoustic model is relatively small. At minimum, a corpus with 1 billion words is recommended for an effective language model. Such corpora are easily accessible through the Linguistic Data Consortium. Thus, the steps to improving this model are clear; first, remedy the dataset and then create a new and more robust language model to pair with the improved acoustic model. These steps should b6e relatively straightforward given that I have done them before, and the code has already been tailored to this dataset.

Conclusion

While the results of my initial attempt to create a working automatic speech recognizer for German are less impressive than desired, the results are still good. Cobalt now has a working ASR that at the very least does better than randomly guessing would; at best it is a solid place to start. Further, repairing this model should now be mostly trivial. More important than that; however, is what I have learned in the process. I am now much more comfortable with many of the terms and technologies used in natural language processing in industry today. I have gained real-world experience scripting in a UNIX based environment and feel competent enough to

continue growing those skills in the future. Lastly, I now know exactly what goes into building a successful ASR and how to avoid possible pitfalls in the future.