Speaker Diarization using Gaussian Mixture Models

Rasul Dent

[1]University of Malta

rasul-jasir.dent.20@um.edu.mt

Abstract

Speaker diarization is the process annotating which person is speaking at a given time. Unsupervised speaker diarization can be accomplished through the implementation of Gaussian Mixture Models (GMM). By creating a large number of GMM and successively combining similar models based on Bayesian Inference Criteria (BIC)-score improvement, speech can be diarized without a GPU in a timespan similar to the duration of the file. While this method produces accurate results, optimized approaches are necessary to improve runtime.

This paper details one implementation of an algorithm that makes use of BIC-scores to merge GMMs trained on Mel-Frequency Cepstral Coefficients (MFCC).

Index Terms: speech recognition, speaker diarization, unsupervised learning

1. Introduction

Speech diarization is the task of assigning speaker labels to different periods of sound in an audio file [1]. Automatic diarization systems produce speaker labels without human guidance during runtime. These labels can be useful in their own right, as knowing the number of speakers in an audio file and the lengths of utterances can provide information about the genre of the audio as well as speaker-speaker dynamics. Moreover, accurate diarization can improve the accuracy of systems that make use of person-specific speech models for other tasks, such as speech-to-text transcription or speech-to-speech machine translation.

Speaker diarization can be approached as either a supervised or unsupervised machine-learning problem. As with other supervised learning problems, supervised speech diarization requires accurately labelled training data, which may be difficult or expensive to acquire. On the other hand, unsupervised diarization does not require training data, which can make it an appealing alternative in scenarios where a sufficient quantity of examples can not be easily obtained. BIC-based clustering of Gaussian Mixtures Models is one unsupervised approach that has been shown to have usable results [2] .

2. Algorithmic Design

2.1. Top level

This diarizer used the algorithm described in the assignment final as a general baseline, but some changes were made to improve runtime. The modified algorithm is as follows:

1. Extract MFCCs for the entire .wav file to be diarized.
2. Perform voice activity detection to obtain a list of MFCCs represent solely periods of speech.
3. Divide the list of speech MFCCs into large initial segments.
4. Create a GMM for each large segment.
5. Divide the list of speech MFCCs into smaller segments which will be used for the remainder of diarization.
6. Label each small segments with the GMM most likely to produce it.
7. Use the newly labelled small segments to train new GMMS.
8. Create a BIC improvement matrix detailing each possible merger and merge the two GMMs providing the greatest improvement in BIC, provided that improvement is positive. A permanent merger of GMMs is accomplished by relabeling the MFCCs of one merge partner to the label of the other partner and then retraining the remaining partner's GMM.
9. Delete the row and column for one of the GMMs that were merged and update only the BIC improvement scores involving the merged GMM and the other remaining GMMs.
10. Repeat steps 6-9 until there are no possible merges which will improve BIC score.

2.2. Preprocessing and feature extraction

2.2.1. Feature selection

Raw audio formats such as .wav often contain too much data for machine-learning methods to process directly. For this reason, some form of compression is necessary to generate a representation suitable for learning algorithms. In speech processing, mel-frequency cepstrum coefficients (MFCCs) are regularly taken as features for machine learning.

I decided to use 13-dimension MFCCs with a window size of 25 milliseconds and a window step of 10 milliseconds because these values are all within the standard windows for their respective parameters. Other dimensions, such as 26 and 40, were also tested, but these slowed runtime and did not appear to provide an increase in accuracy.

2.2.2. Feature extraction

Using the sample code provided by the instructor as a basis, my implementation uses scipy.io.wavfile to read the raw wav file. Then, voice activity detection is performed with inaSpeechSegmenter [3]. Python slicing is then used to join only the MFCCs corresponding to periods of voice activity together to create a new list of speech MFCCs, which is the basis for the diarization algorithm proper.

2.3. Clustering process

2.3.1. Gaussian Initialization

Speaker diarization has been described as a problem of both segmentation and clusterization [4]. Segmentation is relatively simple in this algorithm. External voice activity detection first

separates speech from non-speech, and establishes the length of each speech segment is determined solely by the length of the file. Initially, I followed the instructions strictly and used exactly 10 seconds windows for the initial segmentation and 2 second windows for the refined segmentations. However, because we eventually create a two-dimensional merger matrix, this method had a negative impact on runtime for larger files. I made a slight change to the algorithm by allowing the segment sizes to scale linearly with the file length for files lasting longer than 5 minutes. I then made the size of the small segments scale at a slightly slower rate based on the assumption that speakers were not likely to change as quickly in longer audios. Because the training and test data consisted of television news reports, this seemed reasonable, but it might be worth reconsidering in other contexts.

From here on, the focus of discussion will be the process of forming and merging clusters. Strictly speaker, Gaussian Mixture Models themselves are not an optimal solution to clustering problems where the shape of the clusters is not known in advance. Rather than directly separate data points in a multi-dimensional space, GMMs model the regions of density within clusters and can predict how likely a given data point is to come from a cluster. In order to use GMMs in the context of diarization, it is therefore necessary to first determine how one will create the clusters that the GMMs are used to model.

I have chosen to use k-means clustering as the basis for my GMMs. K-means clustering is one of the most common approaches to clustering and has been used in speech diarization before with good results [5]. Furthermore, it is readily available as an initialization method in the scikit-learn module.However,because K-means clustering assumes that data in each cluster varies from the centroid equally along all dimensions, it is also not a sufficient clustering method for this problem on its own. In this case, k is chosen to be the number of components for the mixture model.

2.3.2. Clustering Criteria

After the initial segmentation, each GMM is created with 8 Gaussian components. This number was chosen because it is high enough to cover the relatively homogenous small segments but not so large that mergers will cause the models to grow too quickly. Each initial model is adapted from a previously created Universal Background Model (UBM) to fit the MFCCs of its corresponding segmen. to training period required. Using a UBM in this situation means that each model does not have to independently relearn how to model the general characteristics of speech with 8 components.

After creating the initial models, each individual MFCC within a segment asks each model to generate a probability that the model produced the MFCC. The model with the highest probability is chosen as the label for that MFCC, and the model that receives a plurality of the votes is chosen as the overall segment label. The new labels are then used to create new models with the same number of components; this step eliminates many early models based on the assumption that the initial segmentation creates far more models than there are speakers.

2.3.3. Merging Gaussians

There are many ways to score a hypothetical merger, including Akaike Information Criteria and Bayesian Information Criteria. These two measures are quite similar, but I choose BIC score based on the assignment description and the fact that this method has been in use for over a decade [6]. For my implementation, I choose to allow the algorithm to run until there were no positive improvements in BIC to be made through the merger of any two GMM. Because the BIC-improvement of a GMM depends on the number of components that both the source models and the merge have, deciding how many components to have in each merger model was very important. Initially, I tried allowing the merger models to be the same size as the target models. This method generally did not allow the algorithm to progress through one full iteration of merging. Then, I tried having the merged model contain twice the same number of components the two source models combined. This approach produced similarly negative results. It quickly became clear that a variable number of components would be need. Moreover, this number would have to be higher than the number of components in either candidate model, but less than the sum of their components.

In the end, I settled on using 3/4 the sum as the number of components in the merger model. This simple scaling factor adds relatively few components in the early merges. Additionally, in merges between models of unequal size, it allows the large model to have more of an impact on the characteristics of the merged model. This seemed more logical than a simple average, but also did not require creating any functions that my overfit the number of components to specific genres of audio.

3. Issues Encountered

3.1. Runtime

Runtime was the main issue that I encountered in implementing this algorithm. As noted by [7], training initial GMM and evaluating potential mergers are two major bottlenecks. To deal with the first problem, I followed the advice of the professor and first created a Universal Background Model from which it would be possible to adapt new GMM from at runtime instead of having to train each model from scratch. Additionally, using the warm-start parameter meant that even in cases where direct adaptation was not suitable, such as mergers that increased the overall number of components, having a general estimate of the model parameters helped reduce training time.

Even with this modification, due to the large number of potential merges to consider during every iteration, the algorithm as I understood it required required up to 40 minutes to diarize an 11-minute long audio sample, which I found to be unacceptably long. In the optimal case, a GPU would be used to train the potential merger models, but even then some form of optimization would be necessary to diarize in time comparable to the length of the audio. One potential solution would be to use Kullback–Leibler divergence to estimate which GMM mergers are likely to have high increases in BIC and only evaluate merge models for the top 3 candidates [7]. I did not begin with this optimization, and instead chose one that more closely followed the outline in the assignment sheet.

To explain my optimization, it is important to first consider the reason behind the large bottleneck at the time of merger comparison. Each time a merger is made, one of the GMM is deleted, and all MFCC associated with that GMM are relabelled. Because a given merge will contain obsolete information about the deleted GMM and inaccurate, a given merge matrix cannot be reused to calculate the next merge. The simplest solution would be to create a new matrix of the possible mergers between the remaining GMM at each step. This is how I initially interpreted the instruction, ”reiterate the alignment and merging”.

To make the process faster, I decided to reuse the merger

predictions for all the models that were not merged. This modification cut the reduced the time spent merging by half for shorter clips and by 3/4 for medium-length clips. Nevertheless, for longer files, such as the 24-minute long bdopb file, the program still took nearly 30 minutes to execute. This is because the initial creation of the matrix is still considering many hypothetical merges that will never be revisited because the two GMMs involved are too dissimilar. Consequently, it would still be beneficial to adopt a metric of similarity, such as Kullback–Leibler divergence [7] to prescreen which merges are worth considering at all. I have not implemented such a measure in this diarizer, but that would be doing so would be my first step in extending this project.

When implementing this change, I initially forgot to update the relabelling procedure. While this did not seem to have a negative impact on performance, it did mean that there was potential for the accuracy of the GMM to drift as non-merged MFCC could still potentially be relabelled. Fixing this mistake did not seem to change the overall accuracy of the diarizer very much on small files, which suggests that either many of the MFCC were not being relabelled after every merge, or the deviations from the recorded models were not severe enough to change the overall merger trajectory.

However, for some larger files, fixing the mistake surprisingly decreased the accuracy of diarization in some cases. I was unable to figure out why.

3.2. Other Issues

In comparison to the issue of runtime, most of the other problems that I encountered were not too difficult to resolve. During the early stages of creating the diarizer, I had trouble deciding how I would keep speech MFCC tied to their original time after eliminating the non-speech MFCC. The initial approach involved using one dictionary with speakers as keys and timed segments as values and another dictionary with time segments as keys and MFCC as values. This seemed like a good approach because the segmentation was done directly after voice activity detection, without an intervening step to gather all of the MFCC representing voice activity into one list. After our January discussion, I realized that it would be easier to combine all the MFCC into one list and use list slicing to create both the long and short segments. As a consequence, it was no longer necessary to use the double dictionary structure.

Instead, I decided to treat each segment as an autonomous unit and ignore time until the very end. Because each individual set of MFCCs is created using the same parameters, duration can easily be obtained by multiplying the number of frames by the window step. To output when a segment begins, I simply search for the segment in the list of MFCCs for the entire file. Because MFCCs contain 13 dimensions of values that are time dependent, there is a very low chance of finding identical MFCCs in two different locations within the same file. When searching for slices rather than just record, the chances are even lower. As a result, this method suffices for the purpose of this project.

4. Results

In my opinion, the final implementation of this diarization algorithm produced reasonably positive results. On shorter files, such as two-minute-long afjiv.wav file, the number of speakers and labelling of time were both consistent and quite close to the gold standard for files with more than one speaker. Furthermore, the whole algorithm only required 70 seconds to run, with half of that being for voice activity detection. For the shortest files, such as "abjxc.wav", the model did narrowed the number of speakers down two, but did not make the final merger.

For longer files, the results were not quite as accurate, but still usable. Sometimes there would be one or two additional speaker labels. Upon reviewing the BIC improvement scores, it became clearer that those labels were nearly merged into others, but the improvement scores were slightly under the threshold. This is to be expected as there are far more segments to merge, which increases the chance of segments belonging to the same speaker having different effects or nuances that make them more difficult to automatically group together.

In some cases, the model was able to ascertain the correct number of speakers, but confused specific speakers with each other. This was particularly common in clips with conversations between two women, such as "sldwj.wav". A notable aspect of this confusion is that it was inconsistent. Sometimes, the diarizer get the alternation correct.

Upon listening to the audio files, I could hear that the voices were somewhat similar, which is an indication that the errors made were overall consistent with the expectations of a diarization system. Curiously, the system seemed to either recognize most of the turns between speakers or collapse most of the talk into one continuous label, with very little middle ground for a given track. I interpret this to mean that some individuals merges have a larger impact on subsequent merges than one might initially consider.

Overall, I was positively surprised by the accuracy of this diarization algorithm. However, the speed made me question the usefulness of running the program on a CPU. For longer files, it would probably be more practical to manually diarize the files, because you can accomplish this task while listening at faster-than-real speed. I would be very interested in comparing the performance of an implementation that is able to make use of GPUs.

5. Conclusions

Speaker diarization is the process of determining who spoke when in a given audio file. This paper has explored one implementation of an unsupervised speaker-diarization algorithm that combines uses Gaussian Mixture Models and Bayesian Information Criteria to determine when different individuals are speaking The approach described has produced labels which mostly coincide with those of the reference files on short files, and have a slightly lower degree of accuracy on longer files. Without access to a GPU, this program requires time roughly comparable to the duration of speech for files with under 30 minutes of speech. Beyond 30 minutes, the algorithm runs increasingly slower than real-time.

There are multiple possible ways to refine this algorithm in addition to introducing a GPU. One such method would be using alternative measures to estimate the probability of model mergers being fruitful without directly generating a merged model for each pair. This is most important with longer files, because the size of the merger matrix is quadratic with respect to file duration.

6. References

[1] X. Anguera, S. Bozonnet, N. Evans, C. Fredouille, G. Friedland, and O. Vinyals, "Speaker diarization: A review of recent research," *IEEE Transactions on Audio, Speech, and Language Processing*, vol. 20, no. 2, pp. 356–370, 2012.

[2] S. sian Cheng, H. min Wang, and H. chia Fu, "Bic-based speaker segmentation using divide-and-conquer strategies with application to speaker diarization," *IEEE Trans. Audio Speech Lang. Process*, pp. 141–157, 2010.

[3] D. Doukhan, J. Carrive, F. Vallet, A. Larcher, and S. Meignier, "An open-source speaker gender detection framework for monitoring gender equality," in *Acoustics Speech and Signal Processing (ICASSP), 2018 IEEE International Conference on*. IEEE, 2018.

[4] T. J. Park, N. Kanda, D. Dimitriadis, K. J. Han, S. Watanabe, and S. Narayanan, "A review of speaker diarization: Recent advances with deep learning," 2021.

[5] H. Dubey, A. Sangwan, and J. H. L. Hansen, "Robust speaker clustering using mixtures of von mises-fisher distributions for naturalistic audio streams," *CoRR*, vol. abs/1808.06045, 2018. [Online]. Available: http://arxiv.org/abs/1808.06045

[6] T. Liu, X. Liu, and Y. Yan, "Speaker diarization system based on gmm and bic," 2006.

[7] E. Gonina, G. Friedland, H. Cook, and K. Keutzer, "Fast speaker diarization using a high-level scripting language," in *2011 IEEE Workshop on Automatic Speech Recognition Understanding*, 2011, pp. 553–558.