Automatic phoneme recognition of continuous speech in MATLAB

A course project in DT2118, Speech and Speaker Recognition

Niklas Vanhainen
niklasva@kth.se

Urban Thunberg
ubb@kth.se

3rd June 2011

Abstract

In this project we have implemented an automatic phoneme recognition system for continuous speech in MATLAB. The system is based on Hidden Markov Models trained on individual phonemes using labeled data. The trained phoneme models are then concatenated into a larger model, which in turn is trained using unlabeled speech data. The TIMIT corpus provided us with the labeled and unlabeled data used when training and testing the models. Features used for modeling the acoustics consist of the standard Mel-Frequency Cepstral Coefficients. We make use of Gaussian Mixture Models to represent the trained speech data. The concatenated model is evaluated using the Viterbi algorithm to provide us with the most likely sequence of phonemes. An experiment with optimizing phoneme models using a genetic algorithm is performed. In the evaluation phase we will look at how well each phoneme model performs alone, as well as how well the concatenated model can represent the actual sequence of phonemes within a spoken sentence. The results are satisfactory. Recognition rate of individual phonemes is quite good at 61%, and the recognition of continuous speech is at least decent in most cases.

1 Introduction

Using our voice is a primary method for communication between humans, and to apply the same on a human-machine communication framework, a robust speech recognition system is crucial. Speech recognition and spoken language understanding can both be regarded as pattern recognition problems[1], and the sequential nature of speech makes Hidden Markov Models a particularly good choice of method to use.

In this course project, we set out to create an automatic speech recognition system (ASR). Numerous types of ASRs have been created through the years using various methods. To get a deeper understanding of the basics of a system based on the popular method of Hidden Markov Models (HMM), we have chosen to implement our own HMM-based ASR system using MATLAB. The HMM is a statistical model based on a hidden Markov chain, where each hidden sate of the chain correspond to an output distribution. In our case, each state-output distribution is a Gaussian Mixture Model (GMM). A GMM is a weighted sum of multiple Gaussian distributions which given enough mixture components can model virtually any distribution possible.
The main goal for the system is to classify a sequence of phonemes uttered in a prerecorded continuous speech sentence. The acoustics of a phoneme, which is the smallest segment within an utterance, are well described by its uncorrelated Mel-Frequency Cepstral Coefficients (MFCC), combined with its first and second derivatives as well as the total energy present within an utterance. The Mel-frequency cepstrum is a transformation of the cepstrum which is intended to model the way humans perceive sound, with a higher sensitivity to differences in lower frequencies than higher frequencies.[1] By using the MFCCs together with a HMM that makes use of Gaussian distributions to represent the state outputs, we have a good basis for our ASR.
The phoneme models are trained using the Baum-Welch algorithm which re-estimates the HMM parameters to fit the phoneme acoustics in our training data. For our ASR, three states should be sufficient to model an utterance of a phoneme. One state for the beginning, one for the middle and one for the end of the phoneme. We may further investigate the use of a variable number of states depending on the phoneme to better suit certain phonemes that tend to be very short and have limited variation of the acoustics.

Many improvement steps have been proposed in the preprocessing stage, such as Cepstral Mean Normalization (CMN) and Vocal Tract Length Normalization (VTLN). CMN is used to handle convolutional distortions[1]and VTLN to reduce the degradation in speech recognition performance caused by variation in vocal tract shape among speakers[2], so that for instance a child's voice can be evaluated against a model trained with adult voices. By evaluating the system with different parameters and creating a confusion matrix for the phonemes we can get a measure of how accurate the phoneme models are.

Once phoneme models have been trained, we concatenate these models into a larger model and once again use the Baum-Welch algorithm to re-estimate the parameters of this new, larger, HMM to take into account for instance the

transition between phonemes. Once this model has been trained, we use the Viterbi algorithm to calculate the most likely sequence of states given speech data of an entire sentence, and thereby also the most likely sequence of phonemes.

2 Method

2.1 Data

To get the features needed for training our models, we extract the MFCCs from the TIMIT corpus raw speech wav-files. We make use of the labels included in each sentence to pick out and collect phoneme data sets from the training sentences. Before training, the HMM is initialized as a finite state Left-Right HMM with equal transition probabilities, as the state propagation for a phoneme utterance always moves in one direction. The initial means of the state output distributions are estimated using the k-means algorithm, and using these centroids, initial covariances are estimated to provide good initial estimates for expectation maximization. We initiate the covariance matrix as a diagonal covariance matrix, because other types of covariance matrices requires many more parameters to be trained. This can be done with the underlying assumption that elements of the feature vector are independent[3].

The hamming window used in the feature extraction process is 10ms with an overlap of 5ms. Many use 20ms windows, but we decided to use 10ms windows in order to make sure that data from short phoneme examples can also be included in the training. As the TIMIT corpus has a very detailed segmentation of phonemes, we need to introduce some simplification. This is mostly performed in the same way as in Lee and Hon's early work using the TIMIT database[4], with the difference that we concatenate consonant closures with their bursts, so for instance a "dcl" followed by a "d", will be considered as a unit "d". This is because these phonemes are often very short, too short to fill a 10ms hamming window.

2.2 Model Training

Once all initialization is done, the Baum-Welch algorithm will train the HMM parameters to fit the training data. After a model is trained, we have two checks that may trigger a retraining if they fail. The first is to check if the log likelihood of the model is not unreasonably low, in which case the model has converged to a local maximum which we determine to be too low considering the results we have seen from training, this check is triggered if the log-likelihood of the model is negative. The second is to check that it is possible to somehow reach the final state from each of the other two states within a phoneme model. This is needed because the final state of each phoneme is the state which will later have transition probabilities to other phonemes when the models are concatenated, and if there is no way to reach the final state, there is also no way to reach the next phoneme.

As we do not know before hand which phonemes to classify in a sentence, we need to create a large HMM incorporating all possible phoneme models and transitions between them. The phoneme models are concatenated creating a large transition matrix of 144 x 144 in size (3 states for each 48 phoneme labels used). To make transitions possible between phoneme models we initialize an equal transition probability from the final state in each phoneme model to the 1st state in all other models. The parameters of this large model are re-estimated using the Baum-Welch algorithm on unlabeled data, and transition probabilities will adjust to make sure that likely phoneme sequences are more probable to be recognized by the model than unlikely ones. In the pursuit of increasing the system's classification accuracy, we can make use of CMN by subtracting the long term means from each cepstral coefficient.

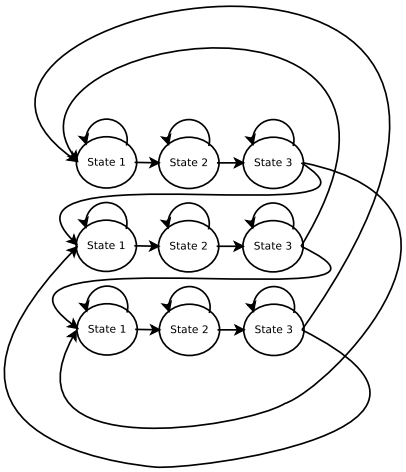

Figure 1: Concatenated HMM of three finite state phoneme HMMs.

2.3 Genetic algorithm optimization

While the Baum-Welch algorithm performs very well, it has a tendency to converge to local maxima. In an attempt to improve the performance of our speech recognizer, we implemented a genetic algorithm combined with the Baum-Welch algorithm in order to optimize the Hidden Markov models, as Kwon et. al[5] has done for isolated word Hidden Markov models. The steps involved in a genetic algorithm are intuitively quite simple to understand. It involves creating new models from existing models by combining and randomly mutating the existing models, and then choosing the best models to carry on and create even more models in the next generation. We used an initial population of 30 models trained with a single iteration of Baum-Welch, and in each new generation, four new models are created. We used 30 generations, and in order to speed up convergence, we perform a Baum-Welch retraining every 10 generations. In order to keep the computational complexity to a manageable level, after each Baum-Welch retraining, the 40 worst performing models are discarded, leaving the 30 fittest models.

We limited our use of the genetic algorithm to the initial phoneme models. Using it for the concatenated sentence model was deemed too time-consuming at this time, although applying this method to the concatenated model would be an interesting experiment to make. This would however require a more efficient implementation of the Baum-Welch and forward algorithm than what is possible in MATLAB.

2.3.1 Crossover

The crossover operation combines two models and creates a new one. Given two models, A and B, the crossover operation creates a new Hidden Markov model by combining a random state from model A, with two states from model B. This is illustrated in figure 2. We do this by simply copying the output distributions corresponding to the three states into the new model, as well as the transition probabilities and prior probabilities, the transition probabilities are re-normalized in order to make sure that all state-transition probabilities still sum to one. The hope is that combining the states of multiple models will create a model that combines the good traits of both models.

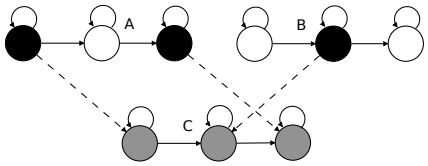

Figure 2: Crossover operation.

2.3.2 Mutation

The mutation operation randomly changes all HMM parameters. It does this by simply multiplying random parameters by small random modifiers. There is a 0.25 random chance that each parameter will be be mutated. The random modifiers for the transition matrix and the mixture Gaussian mixture matrix are distributed according to $\mathcal{N}(1, 0.01)$, and the modifiers for the mean and covariance matrix is distributed according to $\mathcal{N}(0, 0.001)$. No experimentation has been performed to find the best modifiers, simply because training using the genetic algorithm takes a very long time and time was limited.

2.3.3 Selection

In each generation, a number of models are selected as part of the "mating pool", that is selected to create new models using the mutation or crossover operations. The models are sorted by fitness, and a larger proportion of the better performing models are chosen to be part of this pool from which new models can be created. Fitness is determined by simply applying the forward algorithm to the population using the training data, where the fitness measure is the log-probability generated by the forward algorithm. If we only include the most fit models in the mating pool however, the amount of exploration performed to find new solutions will be severely limited, so we also include some mediocre as well as poorly performing models[6]. In order to avoid loosing good solutions, we use what is called "elitism"[6], that is we keep well-performing models in the population instead of simply replacing them with their offspring.

3 Evaluation and Results

By evaluating test phonemes in the TIMIT corpus against trained phoneme models, we get a measure for the how well individual phoneme models perform. This is done by calculating the log likelihood of all models against the phoneme test data using the forward algorithm, and picking the model with the highest probability as the recognized phoneme.Commonly confused phonemes when using the genetic approach are shown in table 1 on page 5. The percentage of correct classification for each phoneme model is presented in table 3 on page 6 for two system configurations. Using a genetic algorithm and cepstral mean normalization, the system was tested with phonemes included in a subset of 150 randomly chosen sentences within the TIMIT Corpus. A mean phoneme model accuracy of 61.7% was achieved. Using the same subset of 150 sentences and evaluating the system without genetic algorithm gave an accuracy of 62.6%.

Phoneme	Recognized phoneme		
	z	y	ix
z	0%	68%	13%
	ch	jh	sh
ch	0%	65%	15%
	uh	ah	ax
uh	7%	33%	13%
	dh	v	b
dh	38%	9%	8%

Table 1: Commonly confused phonemes

On the sentence level, the trained large concatenated HMM model is evaluated against a whole sentence of unlabeled data using the Viterbi algorithm to get the most likely state sequence, which is then translated to the phonemes represented by the states. The Viterbi sequence of classified phonemes of a test sentence in the TIMIT corpus gives a measure of the system's phonemes classification and their transitions within the sentence.

$$acc = 1 - \frac{L(ref_i, rec_i)}{N_{ref}} \tag{1}$$

Looking at the most likely phoneme sequence and the TIMIT corpus reference sequence of a sentence we can take the Levenshtein distance between the two and calculate the accuracy according to equation (1). The mean accuracy for a few configurations of our system is presented in table 2 on page 6.

TIMIT Corpus				
Training set	Test set	Genetic algorithm	CMN	Mean accuracy
M[1]+F[2] i DR[3]*	M Sa1[4]	On (30 gen[5] 2 sent. iter)	On	57%
M+F DR*	F+M Sa1	On (30 gen 2 sent. iter)	On	55%
M+F DR*	F+M Sa1	Off (3 sent. iter)	On	47%
M+F DR*	Sa2	On (30 gen 2 sent. iter)	On	43%
M+F DR*	Sa2	Off (3 sent. iter)	On	27%
M+F DR*	Si	On (30 gen 2 sent. iter)	On	35%
M+F DR*	Si	Off (2 sent. iter)	On	29%
M DR1-3	M Sa1	Off	On	54%
M DR1-3	M Sa2	Off	On	42%
F DR*	F Sa1	Off	On	50%
M DR1-3	F Sa1	Off	On	39%

Table 2: Phoneme transition accuracy.

Phoneme	aa	ae	ah	ao	aw	ax	ay	b	ch	cl
G,C,DR*	74%	80%	42%	73%	56%	41%	71%	76%	0%	77%
C,DR*	73%	77%	39%	72%	44%	39%	79%	59%	65%	80%
Phoneme	d	dh	dx	eh	el	en	epi	er	ey	f
G,C,DR*	58%	38%	58%	31%	61%	61%	71%	74%	87%	84%
C,DR*	56%	38%	58%	39%	54%	57%	73%	72%	74%	82%
Phoneme	g	hh	ih	ix	iy	jh	k	l	m	n
G,C,DR*	71%	80%	44%	43%	73%	65%	63%	58%	70%	58%
C,DR*	69%	82%	41%	45%	69%	49%	61%	56%	73%	54%
Phoneme	ng	ow	oy	p	r	s	sh	t	th	uh
G,C,DR*	39%	60%	92%	54%	62%	92%	88%	56%	56%	67%
C,DR*	57%	46%	92%	57%	61%	82%	91%	53%	44%	7%
Phoneme	uw	v	vcl	w	y	z	zh	sil		
G,C,DR*	69%	73%	73%	83%	68%	0%	67%	92%		
C,DR*	71%	63%	73%	85%	68%	63%	67%	90%		

Table 3: Phoneme model accuracy. G:Genetic, C:CMN, DR:dialects

[1]Male
[2]Female
[3]DR1-9 are the different dialects available in TIMIT
[4]Sa1, Sa2 and Si are types of sentences included in TIMIT
[5]Generations of the genetic algorithm

4 Discussion

Because the majority of speakers in the TIMIT corpus are male, our system tends to perform better with male speakers than with female. The results of optimizing the phoneme models with a genetic algorithm were generally encouraging, though further study is needed to determine whether the increase in accuracy is because of the genetic algorithm or whether simply training multiple phoneme models and choosing the ones with the highest probability generates similar results. On the phoneme level, some phonemes got very poor results using the genetic algorithm optimization compared to without, our hypothesis is that this is due to overfitting to the training data, resulting a less generalized model. Because of these very poor models, the mean accuracy on the phoneme level was actually somewhat lower than without genetic optimization, even though the accuracy was generally higher for most phonemes. What is most striking is the drastic increase in average accuracy on continuous speech using genetically optimized phoneme models as part of the concatenated sentence-model, despite the few very poor phoneme models and despite the fact that we only had time to train the sentence model with two iterations of Baum-Welch.

Using a genetic approach to sentence model training may be prohibitively expensive in terms of computational resources, though this would be an interesting exercise which would likely require implementation in a native language rather than in MATLAB's interpreted code. We have several ideas for improving our system which we may attempt to implement. By implementing duration modeling, we could probably decrease misclassification due to inserts of unreasonably short phonemes. Another improvement step would be to add phoneme transition penalty to gain accuracy in the phoneme sequence classification. We would also like to try implementing Vocal Tract Length Normalization in order to reduce the variability between male and female speakers.

References

[1] X. Huang, A. Acero, H. Hon, *et al.*, *Spoken language processing*. Prentice Hall PTR New Jersey, 2001.

[2] L. Lee and R. Rose, "Speaker normalization using efficient frequency warping procedures," in *icassp*, pp. 353–356, IEEE, 1996.

[3] M. Gales, "Semi-tied covariance matrices for hidden markov models," *IEEE Transactions on Speech and Audio Processing*, pp. 272–281, 1999.

[4] K. F. Lee and H. W. Hon, "Speaker-independent phone recognition using hidden markov models," *IEEE Transactions on Acoustics, Speech and Signal Processing*, vol. 37, no. 11, pp. 1641–1648, 1989.

[5] S. Kwong, C. W. Chau, K. F. Man, and K. S. Tang, "Optimisation of HMM topology and its model parameters by genetic algorithms," *Pattern Recognition*, vol. 34, no. 2, pp. 509–522, 2001.

[6] S. Marsland, *Machine Learning: An Algorithmic Perspective*. Chapman & Hall/CRC, 1st ed., 2009.