IT3105 - AI programming

Project II - Speech Recognition

Students

André Christoffer Andersen
Paul Ingebrigt Huse

Date

2011-10-25

Introduction

In this project we implement a rudimentary speech recognition system using hidden Markov models (HMM) using mel-frequency cepstral coefficients (MFCCs), Gaussian mixture models (GMM) and expectation maximization (EM). MATLAB was chosen as the programing language. This was due to the fact that most of the challenges in this project were mathematical in nature, which makes the built in functions of MATLAB useful.

Part I - Encoding the Sound Signals

All of our file reading operations were handled by the wavread() function in Matlab. For each sound file we read in, we analyzed it, and kept the results in a matrix. Each sound file is divided into a number of semi-overlapping frames, which are then analyzed separately per sound file. Originally, we used fourier transform on the frames, then peak detection, as described in the lecture notes, but we ended up with accuracies below 40% on test data. We replaced this under-performing feature space with mel-frequency cepstral coefficients, which allowed us to improve our accuracy to beyond 50% on test data and 90% on training data.

The main reason for why we tried using mel-frequency cepstral coefficients was that they were mentioned in the lecture notes as something more advanced systems use, which piqued our interest. Reading about them in various reports, they seem to cover both pitch and formant information rather well, and we take the fact that they are used in a wide range of speech recognition software as a strong indication of their efficiency. The fact that our recognition software increased in accuracy by about 20 percentage points when we used them, instead of simple peak detection over a fourier transform, is also a strong practical indicator of their efficiency.

Part II - HMMs and overall classifier

For phase II of the project, we used some existing libraries. Our hidden Markov model is the hidden Markov model function found in the library HMMall. Our observation model is a mixture of gaussians, with the number of states in our hidden markov model as the number of gaussians. After our data is read, it is delivered directly to our mixture of gaussians model, and only the results are retained for later.

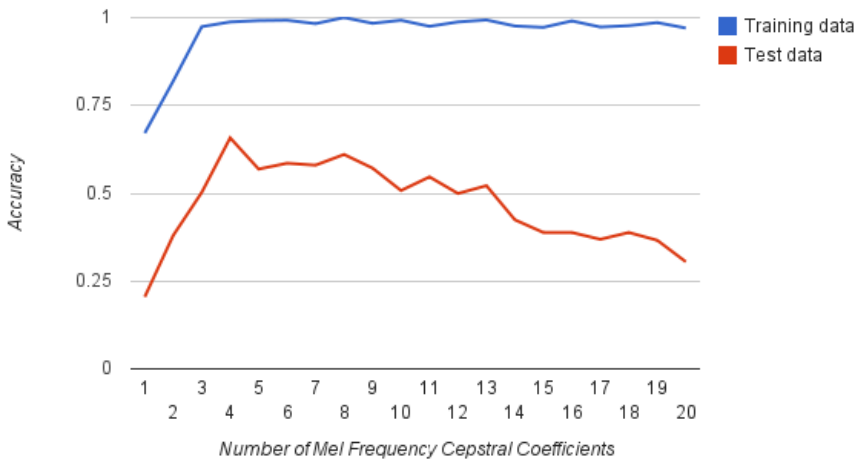
Part III - Learning from data

Our hidden Markov model is first given a GMM, then trained further using the Baum-Welch algorithm. The feature extraction of our files is described under phase 1. For our observational model, we trained it using the expectation maximization algorithm before feeding the results into the HMM.

To test files to see how well they matched any given HMM, we used the forward-backward algorithm. This allows us to find the probability of the sequence of observations that any given test sequence describes (after having been run through our feature extractor, of course).

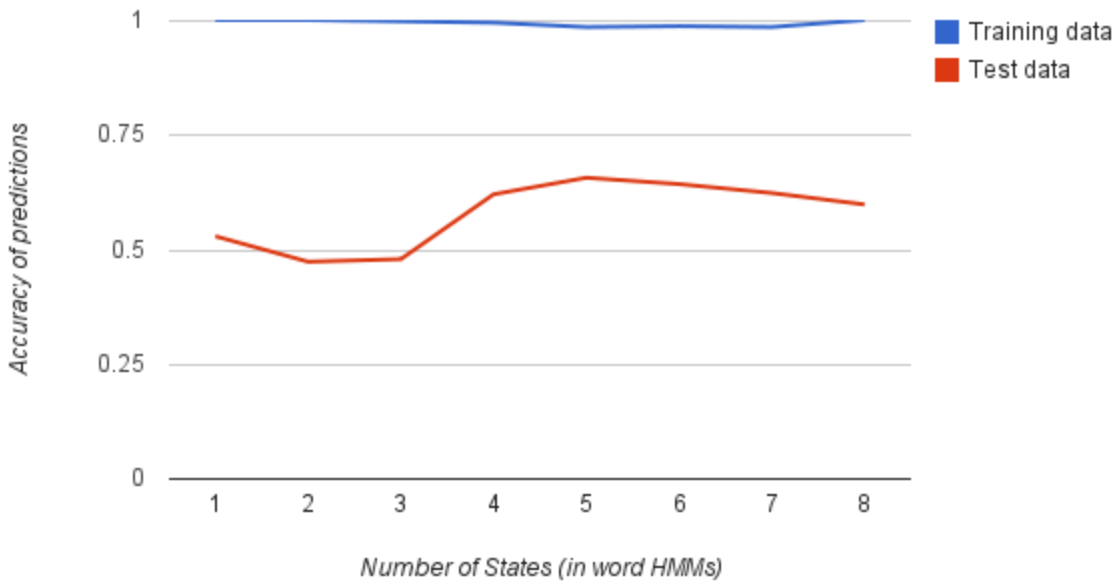
Part IV - Wrapping it all up

To optimize our model, we tested out various possible values for our parameters, using a grid search method. Not exactly efficient, and not a completely rigorous test, as that would take too long, but we made an attempt to find optimal values for (for example) cepstral coefficient numbers by running a test 10 times with each value, and considering the average for each number. Since our system uses (amongst other things) randomly created starting matrices for our HMM, the results did vary a fair bit, but did result in improving our measurement results. This gave us the (utterly non-shocking) result that 2 cepstral coefficients give better results than 1, amongst other things. Interestingly, it indicated that 4 was the optimum number of coefficients for our setup (most systems use a higher number of coefficients), which is why we have used that nfor later tests. For this grid search, we used 5 as the number of hidden states.



The result of our grid search over number of coefficients. Each prediction accuracy is the average of ten test runs.

Afterwards, we did a similar grid search over the number of hidden states. We could have combined these two (and also other parameters, as mentioned under “Possible future optimization”, below), but consider the deadline and when our system was ready for final testing, this was not possible in the remaining time.



The systems accuracy was slightly higher with 5 states (0.6556) than with 6 states (0.6417), so our system currently uses 5 states in the HMM for each word.

Testing

To test our models, we ran a series of tests. We tested our classifier on both the training data and the test data. Our goal was to classify the test data as well as possibly, but we also checked the classification of our training data as a secondary test of our classifier. A series of 10 tests follows:

Test run:	Test data prediction accuracy	Training data prediction accuracy
1	0.6111	1
2	0.6667	0.7885
3	0.5833	0.9904
4	0.6111	0.9904
5	0.6944	1
6	0.5000	0.9712

7	0.6667	1
8	0.5556	1
9	0.5833	0.9904
10	0.6944	1
Average:	0.61666	0.97309

We also ran several test runs, averaging the results. This gave us a net accuracy of 0.9907 over our training data and 0.6454 over our test data (on average over 30 runs). (To compare, we ran a identical set of runs with 13 cepstral coefficients, which gave us an accuracy of 0.9865 for our training data and 0.5111 for our test data).

Possible future optimization

It is quite probable that adding further measurements to our model may improve it's efficiency, as it will give it access to further data when training the HMM. In addition, to find optimal parameters for everything from frequency range considered to cepstrals, a more thorough search would help (many of our current parameters are best guesses, or based on some quick testing). For example, running a very large grid search might lead us to greatly improved accuracy. It might also be helpful to use a more complex HMM, for example a two layered one, or some other representation of our data. Another factor to consider is that increasing the difference between the HMMs for the different words might increase our accuracy. Currently, we use the same number of states for all HMMs in our system. However, the different words have differing number of phonemes, and adjusting the number of states to fit each word better might also increase the accuracy of our system.

Conclusion

Our speech recognition system, though not exactly the most accurate recognizer ever, did manage a reasonable degree of accuracy. It probably will not hold up if the vocabulary was extended, but for such a small selection of words, it is workable.

However, it should be mentioned that we do suspect that if we spent the same amount of time training a small group of monkeys to do this same task, we'd get similar degrees of accuracy. That approach does have the advantage of including monkeys, which is generally always a plus, but considering the amount of animal manure involved, on balance, we prefer this method.