Automatic Speech Recognition (ASR) Report

Final project for EE516 PMP at UW, prepared by Paul Adams 7-Jun-2015

Implementation Notes

Definitions

1. *TIDIG(train, test)*
   a. An open-source database of training and test utterances for a speaker-independent speech recognition system. For this project, the isolated digit set was used. The database is found at https://catalog.ldc.upenn.edu/docs/LDC93S10/
2. *lexicon* → Project vocabulary set: {Zero, One, Two, Three, Four, Five, Six, Seven, Eight, Nine}
3. $\mathrm{w}^{\mathrm{i}}_{\mathrm{train,k}}$ → The k^{th} training utterance of the i^{th} word, for k in *TIDIG(train)* and for i in *lexicon*
4. N → Observation feature dimensionality
   a. For this project, Mel-Frequency Cepstral Coefficients (MFCC) were used with $N = 26$, or 13 cepstral coefficients + 13 delta coefficients
5. M^i → Number of hidden states for $\mathrm{w}^{\mathrm{i}}_{\mathrm{k}}$
6. T^i_k → Number of 25 ms frames in $\mathrm{w}^{\mathrm{i}}_{\mathrm{k}}$
7. $\mathrm{x}^{\mathrm{i}}_{\mathrm{k}}$ → MFCC featureset for $\mathrm{w}^{\mathrm{i}}_{\mathrm{train,k}}$ with dimensions *N* x $\mathrm{T}^{\mathrm{i}}_{\mathrm{k}}$
8. $\lambda^i = \pi, A_{ij}, B$ → Trained Hidden Markov Model (HMM) parameters using Expectation Maximation (EM) algorithm where, A is the state transition probability distribution, B is the observation symbol probability distribution and π is the initial state distribution, or prior.

EM HMM Implementation Pseudocode

$x^i_k = \mathrm{mfcc}(\mathrm{w}^{\mathrm{i}}_{\mathrm{train,k}})$ for each i in lexicon and for each k in training utterances
$B_0 = \mathcal{N}\left(x^i_k \middle| \mu(x^i_k), \sigma(x^i_k)\right)$
for i in lexicon:
    while not converged:
        for k in training utterances: % E Step
            load x^i_k
            $B = \mathcal{N}\left(x^i_k \middle| \mu_0, \sigma_0\right)$
            $\alpha, \beta, \gamma, \xi = forwardbackward(\pi, A_{ij}, B)$
        update μ, σ % M Step
        update A_{ij}
        update π
        converged? If not, go to next epoch, else break
save $\lambda^i = \pi, A_{ij}, B$

ASR Procedure

$\mathrm{w}_{\mathrm{test}}$ ← load or record a candidate utterance for recognition
$x = \mathrm{mfcc}(\mathrm{w}_{\mathrm{test}})$
for i in lexicon:
    load $\lambda^i_M = \pi, A_{ij}, B$
    $\alpha, \beta, \gamma, \xi = forwardbackward(\pi, A_{ij}, B)$
    $LogLikelihood(i) = \log(p(x \,|w^i, \lambda^i)$
$w^* = argmax(LogLikelihood_i)$

HMM Testing Procedure

Once a working ASR framework was established, a word error rate (WER) performance measure was achieved by utilizing utterances from *TIDIG(test)* and by iterating over each set of $\mathrm{w}^{\mathrm{i}}_{test}$ and tallying the ratio of correctly recognized utterances to the running total.

This enabled searching a small parameter space for an optimal M^i for each $\mathrm{w}^{\mathrm{i}}_{\mathrm{test,k}}$. That is, iteratively train and save λ^i_M for several values of M. Using the test utterance set, tally a WER for each of the λ^i_M. Select M^i and corresponding λ^i_M such that the WER is minimized.

ASR Software Overview

The Matlab code is organized into a main function, `myASR.m` containing the three elements of the system – Training, Testing and Interactive Recognition. By passing options to `myASR.m` the user is able switch various functionality.

The Training portion consists of implementing the EM algorithm and saving the results. The Testing portion consists of optionally retraining to new inputs and then performing recognition of a set of test utterances, while recording the WER. To reduce the amount of time needed to retrain the HMMs while developing, two constants were created to select out a subset from the TIDIG corpus. For training, this meant using `nTrainWords` per word to train the HMM. For testing, this meant randomly selecting out `nTestWords` from the available set.

Finally, the Interactive Recognition portion utilizes the trained HMMs, which have been validated by Testing, to recognize new speech utterances.

At the front-end is setup code to pre-compute all MFCC features from the training set of utterances and then compute a global mean and variance. The initial state distribution is set as a random vector of length M^i. Precomputing observations and initial Gaussian parameters reduces the time to train the HMMs.

Notes on Performance

Word Error Rate

Given enough training data, the system can perform with word error rate (WER) < 4% for most words in the lexicon. However, this is using the curated TIDIG corpus for test samples. Using the system on my own or my wife's speech, with background noise and occasionally garbled utterances, the system is less accurate, but still mostly correct. Results for the case, $M = 8$, for all words, using `nTestWords` $= 100$ and `nTrainWords` $=$ 150 are shown in Figure 1.

```
Being testing word Z...     iWord: Z, WER: 0.00 Percent
Being testing word 1...     iWord: 1, WER: 0.00 Percent
Being testing word 2...     iWord: 2, WER: 0.00 Percent
Being testing word 3...     iWord: 3, WER: 0.00 Percent
Being testing word 4...     iWord: 4, WER: 0.00 Percent
Being testing word 5...     iWord: 5, WER: 1.00 Percent
Being testing word 6...     iWord: 6, WER: 0.00 Percent
Being testing word 7...     iWord: 7, WER: 0.00 Percent
Being testing word 8...     iWord: 8, WER: 2.00 Percent
Being testing word 9...     iWord: 9, WER: 4.00 Percent
```

Figure 1 - WER Results

I found that the WER is not a strong function of M^i, assuming enough states, say 8. Similar results, are seen for each word for each number of states. Five, eight and nine had consistently higher WER. It is likely that some words are more sensitive to the number of hidden states. In most case however, with enough training data, it seems as if the exact HMM input parameters are less important.

Computational Cost

Training a new set of HMMs takes ~ 10 minutes using 150 training utterances per word, from a set of 10 words. I found that EM converged sooner with more training utterances, so that compute time actually reduced if enough training words were used. Testing, which is essentially performing the Recognition step several hundred times, takes ~ 5 minutes. Given a trained HMM the Recognition step happens in a second or two.

Next Steps

It is fascinating to see the accuracy achievable using a relatively simple system. An obvious next step would be to extend to n-grams of increasing complexity and investigate incorporating deep learning techniques. This program is made available at https://github.com/p5a0u9l/myASR.