Audio-Based Sentiment Analysis

Bingyue Wang, Mitchell Goff & and Justice Amoh

October 28, 2014

Introduction

In our previous writeup, we proposed to implement a sentiment analysis system that employs audio speech data eclusively, to infer the emotions of new speech data. We identified some databases we could use for training and also discussed some of the features to consider extracting for our purposes. Finally, we mentioned the algorithms we were going to implement for this learning objective. Now in this paper, we report what we have been able to implement by the milestone.

Database

We have contacted Ringeval Fabien at Université de Fribourg to retrieve the RECOLA database. Each sample audio data in the RECOLA database involved two people working together in a videoconference setting "to rank 15 items according to their significance for survival in a deserted and hostile area"[6]. There are a total of 46 French-speaking students who participated in the experiment, of which 58.7% were females. Six French-speaking annotators have annotated the video data in a time continuous fashion using the tool ANNEMO[1]. As such, an annotation is collected for every 40ms frame of the audio-video recording. We focus exclusively on the audio of the recordings only.

Data Annotation

There are two main dimensions for annotation: the social dimension and the affective dimensions. For the social dimension, the annotators graded how positive a conversation was in terms of "agreement, dominance, engagement, performance and rapport". On the other hand, the affective dimension annotation is to capture the emotions or sentiments in the conversation. Since we are interested in sentiment analysis, we solely on the affective annotations. The annotators graded conversations on a scale of -1 to $+1$ in terms of *arousal* and *valence.* The arousal scale is to represent how excited or apathetic the emotion is. The valence scale captures how positive or negative the emotion is [6].

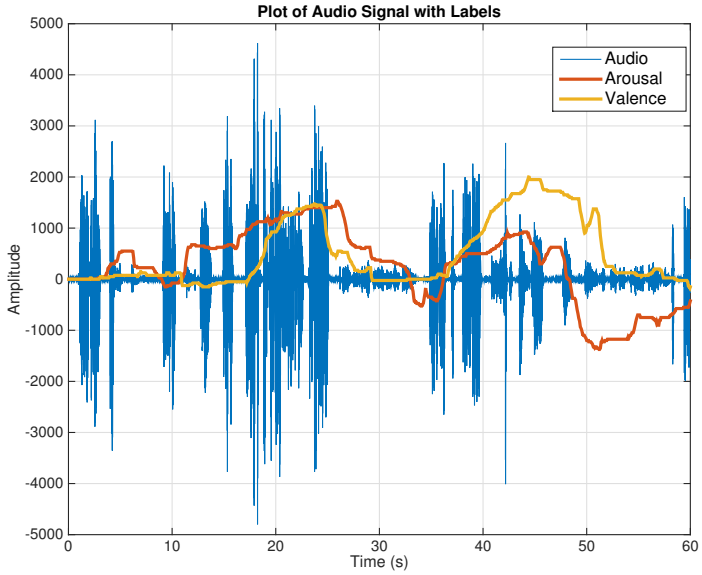


Figure 1: A plot of sample speech audio from database (blue), superimposed with arousal and valence labels.

For each audio sample, there are two csv files which contains the annotations, one for the arousal and another for the valence. All these annotations are also time-stamped. Since there are six independent annotators, each time frame has six associated annotations each for arousal and for valence. For our purposes, we use just one annotation each by finding the median of the six independent annotations. So the actual annotations we use for each time frame of the audio are the median arousal and median valence. It is worthwhile to mention that there are only annotations for the first five minutes of each audio sample, even though the sample lasts about ten minutes. And so there are far fewer annotations than audio frames. As such, we discard the audio that has no annotation. In Fig. 1, we plot the median valence and arousals superimposed on the audio.

Audio Preprocessing

The first processing we do on the audio signal from the database is to normalize by it's mean. Since experimental setup or other factors such as distance of speaker to microphone could differ across examples, this normalization ensures that all our audio samples are adjusted to be on a similar amplitude scare.

Following normalization, we run the audio signal through a bandpass filter

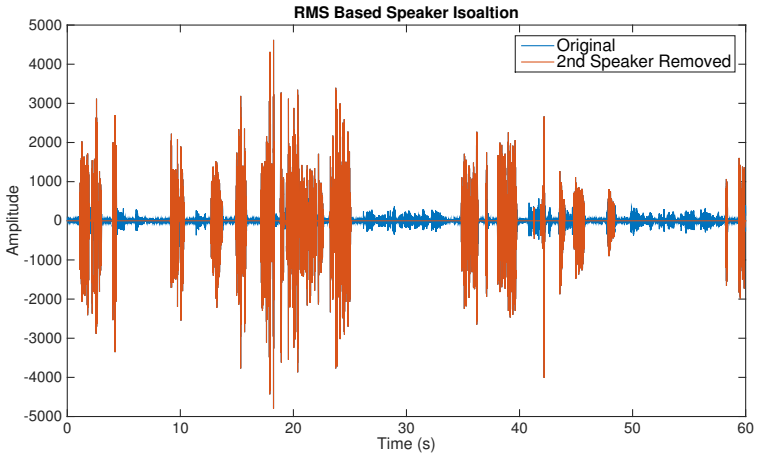


Figure 2: Voice activity detection and second speaker isolation through RMS energy thresholding.

to isolated any frequencies outside the voice frequency band. We do so to eliminate all possible high or low frequency noise signals that may occur in the signal as artifacts from the experimental or recording setup. We use a 4th order butterworth bandpass filter with cutting frequencies at 200Hz - 4KHz (voice range is popularly 300Hz-3KHz).

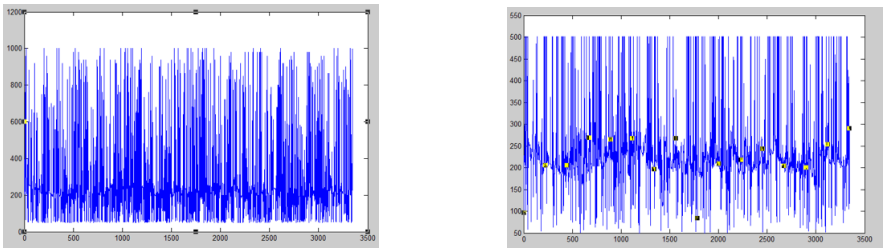
Finally, we implemented a basic voice activity detection techniques to remove the chunks of the audio that were silent. We computed the Root Mean Squared energy level (RMS) for each frame and based on a prior set RMS threshold, we admit frames that have energy levels indicating presence of speech. Also, since the audio data involves two people talking in a video conference with the recorder placed right next to only one person, one person's voice is significantly louder than the others. Naturally, since our focus is on emotion rather than social interaction, it is necessary to isolate the second speaker. We were able to do this using the same RMS energy. So our RMS threshold enables us to isolate and remove the second speaker. This is shown in a plot in Fig. 2.

Feature Extraction

In our proposal, we mentioned a number of audio features we intended to extract for training our classifier. We implement feature extraction at the frame level. So first, for each frame, we have computed features that describe the distribution of the signal amplitude within the frame. These features include the mean, standard deviation, skewness and kurtosis. We also include the root mean squared as a measure of energy, and the zero crossings rate as a measure of noise.

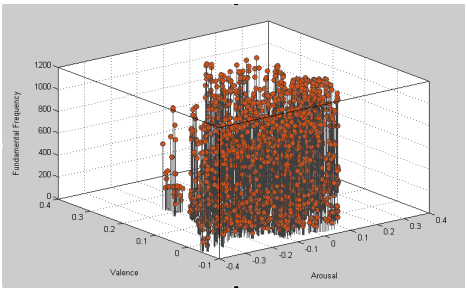
A more involving feature we have explored is the fundamental frequency. According to *Buso et al*, "The fundamental frequency of F0 contour(pitch), which is a prosodic feature, provides the tonal and rhythmic properties of the speech"[2]. Since the fundamental frequency is independent of vocal tract and lexical content, it is likely a good indicator for speaker emotion. Therefore, we have used two approaches to extract the fundamental frequency of each frame in the audio sample. The first approach uses "cepstrum, [which] is a Fourier analysis of the logarithmic amplitude spectrum of the signal." The cepstrum will look for periodicities in the signal spectrum and show peaks at quefrencies which corresponds to the frequencies of harmonics. Therefore to obtain the fundamental frequency from cepstrum, we look at the peak within quefrency region corresponding to typical speech fundamental frequencies. The fundamental frequencies for each frame of the audio sample data is shown in Fig. 3a.

While the cepstrum looks at the logarithmic of signal spectrum for periodicities, a more direct way to extract fundamental frequency would be to look at repeated patterns in the signal spectrum itself. The sound waveform is supposed to correlate well with itself at delays of intervals of pitch period. Therefore, we can find delay times for peaks of correlation to find out the fundamental frequency. The fundamental frequency thus found is shown in the Fig. 3b. Note that the fundamental frequency values obtained from these two methods differ markedly from each other and this is an area of investigation towards completion of our project. It is worthwhile to mention that we also attempted to visualize how discriminative a feature the fundamental frequency can be by plotting a 3D scatter plot as show in Fig. 4a. However, no clear demacation could be gleaned by merely looking at the data.

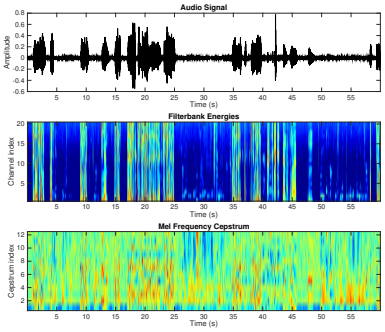


(a) (b)

Figure 3: Fundamental frequency from (a) cepstrum and from (b) AutoCorrelation.



(a)



(b)

Figure 4: (a)A 3D scatter plot showing dependence of fundamental frequency on valence and arousal. (b)Mel's Frequency Cepstral Coefficients and associated filterbank energies.

Finally, we also computed the Mel' Frequency Cepstral Coefficients using the third party toolbox called *PMTK [4].* The MFCCs capture the frequency spectrum of the audio signal. For each frame, we computed 13 cepstral coefficients. Plots of the MFCC coefficients and filterbank energies are shown in Fig. 4b. All the Matlab scripts for extracting the discussed features can be located in the Appendix.

Learning Algorithm

In our proposal, we mentioned to consider using two learning methods; Binary Emotion Detection and Hidden Markov Models. However, from further research and feedback from our course instructor, we decided to focus our attention on Hidden Markov Models solely since that was a more practical goal given the project requirements and our time constraints. So we set off to implement the Hidden Markov Model (HMM) classifier.

The HMM and it's applications was discussed in details in our proposal so we proceed to rather elaborate on our implementation of the algorithm. We solely relied on *Rabiner et al*'s famous article on how to implement hidden markov models[5]. In this paper and included source codes, we adopt the same notation used by *Rabiner*. Hence an HMM model is given as:

$\lambda = (N, M, A, B, \pi)$, where

λ - *hidden markov model*

N - *number of states of hidden markov model*

M - *number of observation symbols*

A - *state transition probabilities matrix* $N \times N$

B - *observation emission probabilities matrix* $M \times N$

The HMM tutorial segmented the HMM implementation into three problems, each with an associated algorithm:

- Evaluation problem - *Forward Algorithm*
- Recognition problem - *Viterbi Algorithm*
- Parameter estimation - *Baum-Welch Algorithm*

Currently, we have been able to implement the first two algorithms. So we discuss below those implementations and how we have tested them.

1st - Evaluation Problem

The first canonical problem with HMM is how to compute the probability that an observation sequence $O = \{o_1, ..., o_T\}$ is from a given hidden markov model λ, i.e., $P(O \mid \lambda)$. To perform this evaluation, we use the Forward Algorithm. Rabiner's tutorial list the steps and expressions for the forward algorithm as follows [5]:

1. *Initialization:* $\alpha_1(i) = \pi_i \cdot B_i(o_1)$ *for* $1 \leq i \leq N$
2. *Induction:* $\alpha_{t+1}(j) = \left[\sum_{i=1}^{N} \alpha_t(i) \cdot A(i,j)\right] B_j(o_{t+1})$ *for* $1 \leq j \leq N$
3. *Terminate:* $P(O \mid \lambda) = \sum_{i}^{N} \alpha_T(i)$

where $\alpha_t(i)$ is the probability of being in state i of the markov model at time t from all possible previous transitions. For these steps, our matlab algorithm is below:

```
% Author: Justice Amoh
% Description: Implementation of Forward Algorithm
% as explained by Rabiner, 1999.
% HMM Task: Evaluation problem - Compute probability
% that observed sequence is produced by given model
% Date: 10/22/2014

function logprob = forward(model,obsv)
 T     = length(obsv);
 alpha = zeros(model.N,T);

 % initialization
 for i=1:model.N
  alpha(i,1) = model.pi(i) * model.B(i,obsv(1));
 end

 % induction
 for t=2:T
  for j=1:model.N
   for i=1:model.N
    alpha(j,t) = alpha(j,t) + alpha(i,t-1)*model.A(i,j)*\
        model.B(j,obsv(t));
```

```
    end
   end
  end
  logprob = log(sum(alpha(:,T)));
end
```

2nd - Recognition Problem

The second problem has to do with how to figure out the best sequence of states $Q = \{q_1, .., q_T\}$ that generate the observation sequence from a given hidden markov model. In other words, finding Q such that $P(Q \mid O, \lambda)$ is maximized. This is sometimes refered to as the decoding phase; where the "hidden states" behind the observations are uncovered. One algorithm used for this issue is the Viterbi Algorithm. *Rabiner*'s approach for implementing the Viterbi is as follows[5]:

1. *Initialization:*
   (a) $\delta_1(i) = \pi_i \cdot B_i(o_1)$ *for* $1 \leq i \leq N$.
   (b) $\psi_1(i) = 0$ *for* $1 \leq i \leq N$.
2. *Recursion:*
   (a) $\delta_{t+1}(j) = \max\left[\delta_t(i) \cdot A(i,j)\right] B_j(o_{t+1})$ *for* $1 \leq J \leq N$.
   (b) $\psi_{t+1}(j) = \arg\max[\delta_t(i) \cdot A(i,j)]_{1 \leq i \leq N}$ *for* $1 \leq j \leq N$.
3. *Termination:*
   (a) $P(Q \mid O, \lambda) = \max[\delta_T(i)]_{1 \leq i \leq N}$
   (b) $q_T = \arg\max[\delta_T(i)]_{1 \leq i \leq N}$
4. *Path:* $q_t = \psi_{t+1}(q_{t+1})$ *for* $t = T-1, T-2, ..1$

where $\delta_t(i)$ the most probable transition from all previous states to the current state i at time t and ψ_t is the state that yielded that max probability. For this, our Matlab algorithm is as follows:

```
% Author: Justice Amoh
% Description: Implementation of Viterbi Algorithm
% as explained by Rabiner, 1999.
% HMM Task: Recognition problem - Finding the optimal
% state sequence associated with an observation
% Date: 10/22/2014

function [prob, spath] = viterbi(model,obsv)
 T     = length(obsv);
 delta = zeros(model.N,T);
 psi   = zeros(model.N,T);
 spath = zeros(1,T);
```

```
 prob  = zeros(1,T);

 % initialization
 for i=1:model.N
  delta(i,1) = model.pi(i) * model.B(i,obsv(1));
  psi(i,1)   = 0;
 end

 % recursion
 for t=2:T
  for j=1:model.N
  temp = delta(:,t-1).*model.A(:,j).*model.B(j,obsv(t));
  [delta(j,t), psi(j,t)] = max(temp);
  end
 end

 % termination
 [prob,spath]=max(delta);
end
```

Testing Algorithms

As mentioned previously, we are yet to implement the third and final algorithm; the Baum-Welch. However, since the Baum-Welch is the algorithm for training and hence building an HMM model in the first place, we have not yet been able to build our own HMM on the database. And so in order to test our Forward and Viterbi algorithms without the Baum-Welch, we found two example models online [3]. With these two examples, we run our algorithms and verified that our $P(O \mid \lambda)$ and predicted state sequences matched the reported results on the website. Testing was done with observation vectors of length 10 and 100. Find below our test scripts:

```
% Author: Justice Amoh
% Date: 10/16/2014
% Description: A script to test
% hmm parameter estimation algorithms:
%=================
% Test data source: http://www.bioss.ac.uk/people
% /dirk/SuperV/AdrianoAnna/ForwardBackward/
% Dependencies:
% - hmm.m
% - forward.m
% - backward.m
% - viterbi.m

obsv  = [2 1 2 1 1 1 1 1 1 2 ];
obsv1 = [2 2 2 2 1 1 2 1 1 1 1 ...
         2 1 1 2 1 2 1 2 1 1 1 ...
         2 1 2 1 1 2 2 1 1 2 2 ...
```

```
          2 2 1 1 1 1 2 1 2 1 2 ...
          1 1 1 1 2 1 1 1 1 2 2 ...
          2 2 2 1 2 1 1 2 2 2 2 ...
          1 1 1 1 1 2 1 1 1 1 2 ...
          1 2 2 2 1 1 2 1 2 2 1 ...
          2 1 2 1 2 1 1 1 2 2 1 1];
N = 3;
M = 2;

% Model A
A = [ 0.90 0.05 0.05; 0.45 0.10 0.45; 0.45 0.45 0.10 ];
B = [ 0.50 0.50; 0.75 0.25; 0.25 0.75 ];
prior_pi = [0.333 0.333 0.333];
modelA   = hmm(N,M,A,B, prior_pi);

% Model B
A = [ 0.50 0.25 0.25; 0.20 0.40 0.40; 0.10 0.45 0.45 ];
B = [ 0.50 0.50; 0.75 0.25; 0.25 0.75 ];
prior_pi  = [0.333 0.333 0.333];
modelB = hmm(N,M,A,B, prior_pi);

% Run Forward Algorithm on Models
loglik_A = forward(modelA,obsv)
loglik_B = forward(modelB,obsv)

% Run Viterbi Algorithm
[logprob_A,spath_A] = viterbi(modelA,obsv);
[logprob_B,spath_B] = viterbi(modelB,obsv);
```

Upcoming Works

In the coming weeks, we look forward to bringing our project to completion by first completing our learning algorithm. We will implement the Baum-Welch algorithm from *Rabiner*'s tutorial just like we did for the Forward and Viterbi. This should hopefully be less daunting since the Baum-Welch actually uses the Forward Algorithm which we have already implemented. Once the Baum-Welch is completed, we will use it to train a hidden markov model on a subset of our database and run tests to verify the model's performance. Then if time permits, we will explore ways to augment the structure of our hmms to increase classification accuracy.

Another area we are hoping to explore is how to identify the most effective features. Since we are computing a large array of common audio features, we suspect there will be some redundancy in the features. Also, some features may provide little or no information for determining sentiments. And so, we will perform principal component analysis to find those features that are most useful for sentiment classification.

Appendix

Feature Extraction Code

```
% COSC 174 Project
% Project: Sentiment Analysis - Team Blistering Barnacles
% Description: Function to extract audio features

function featVect = getFeatures(signal,t,frame_len)
 % Input:
 % signal    - audio signal amplitude vector
 % t         - associated time vector for signal
 % frame_len - length of a frame for processing (in seconds)

 % Cut signal into frames of length 'frame_len'
 dt = t(2)-t(1);
 fs = 1/dt;
 spf = round(frame_len/dt);       % Samples in a frame_len
 sig_frames = buffer(signal,spf);
 t_frames = buffer(t,spf);
 [~,nf] = size(sig_frames);
 Skewness    = zeros(nf,1);
 Kurtosis    = zeros(nf,1);
 SDeviation  = zeros(nf,1);
 Mean        = zeros(nf,1);
 RMS         = zeros(nf,1);       % Root Mean Squared Level
 ZCR         = zeros(nf,1);       % Zero Crossings Rate
 F0          = zeros(nf,1);       % Fundamental Frequency

 for i=1:nf
  Skewness(i)   = skewness(sig_frames(:,i));
  Kurtosis(i)   = kurtosis(sig_frames(:,i));
  SDeviation(i) = std(sig_frames(:,i));
  Mean(i)       = mean(sig_frames(:,i));
  RMS(i)        = rms(sig_frames(:,i));
  ZCR(i)        = numel(zerocross(sig_frames(:,i)))/spf;
  F0(i)         = fundamental(sig_frames(:,i),t_frames(:,i));
 end

 %% Compute MFCC
 % Define MFCC Extraction Parameters
 Tw = frame_len*1000;     % analysis frame duration (ms)
 Ts = frame_len*1000;     % analysis frame shift (ms)
 alpha = 0.97;            % preemphasis coefficient
 M  = 20;                 % number of filterbank channels
 C  = 12;                 % number of cepstral coefficients
 L  = 22;                 % cepstral sine lifter parameter
 LF = 0;                  % lower frequency limit (Hz)
 HF = 3700;               % upper frequency limit (Hz)
```

```
 % MFCC extraction (feature vectors as columns)
[MFCC,~] = mfcc( signal, fs, Tw, Ts,...
                 alpha, @hamming, [LF HF], M, C+1, L );
 MFCC = MFCC';
 featVect = table(MFCC,Skewness,Kurtosis,...
                  SDeviation,Mean,RMS,ZCR,F0);
end
```

Preprocessing Code

```
% COSC 174 Project
% Project: Sentiment Analysis - Team Blistering Barnacles
% Description: Function to preprocess audio before
% feature extraction
function[signal,t] = preprocess(filename)
 [signal, fs] = audioread(filename);
 t  = 0:1/fs:(length(signal)-1)/fs;

 % Normalize audio signal by mean
 signal = signal/mean(signal);

 % Bandpass filter to voice Freq. Range: 200Hz - 4KHz
 n  = 4;                    % 4th Order filter
 fc =[200 4000];            % BPF Cutoff Frequencies
 Wn = 2.*pi.*fc;
 [B,A] = butter(n,Wn,'bandpass','s');
 signal = lsim(tf(B,A),signal,t);
end
```

Hidden Markov Model Code

```
function model = hmm(N,M,A,B,prior_pi)
% Author: Justice Amoh
% Date: 10/16/2014
% Description: A function for building a Hidden
% Markov Model from input parameters:
%=================
% Input:
%=================
% N = number of states
% M = number of observation symbols
% A = transition matrix - N x N
% B = emission matrix - M x N
% prior = initial state distribution - N x 1
%=================
% Output:
%=================
% model = struct containing hmm model params
model.N  = N;
```

```
model.M   = M;
model.pi  = prior_pi;
model.A   = A;
model.B   = B;
end
```

References

[1] Ilya Boyandin. ANNEMO, 2012.

[2] Carlos Busso, Sungbok Lee, and Shrikanth Narayanan. Analysis of emotionally salient aspects of fundamental frequency for emotion detection. *IEEE Transactions on Audio, Speech and Language Processing*, 17:582–596, 2009.

[3] Dirk Husmeier. Test your implementation of the forward-backward algorithm.

[4] Kevin P. Murphy. PMTK, 2010.

[5] Lawrence R. Rabiner. A tutorial on Hidden Markov Models and Selected Applications in Speech Recognition. In *Proceedings of the IEEE*, pages 257–286, 1989.

[6] Fabien Ringeval, Andreas Sonderegger, Juergen Sauer, and Denis Lalanne. Introducing the RECOLA multimodal corpus of remote collaborative and affective interactions. In *Automatic Face and Gesture Recognition (FG), 2013 10th IEEE International Conference and Workshops on*, pages 1–8. IEEE, 2013.