Estimating speech from lip movement

Jithin D. George, Ronan Keane, Conor Zellmer

March 16, 2017

Abstract

The goal of this project is to develop a limited lip reading algorithm for a subset of the english language. We consider a scenario in which no audio information is available. We first develop a method for processing raw video and extracting the position of the lips in each frame. We then prepare the lip data for processing and classify the lips into visemes and phonemes. We then use Hidden Markov Models to predict the words the speaker is saying based on the classification. We use the GRID audiovisual sentence corpus database for our study.

1 Introduction and Overview

We consider 4 different algorithms for extracting the lip position.

- Active Contour Mask (Matlab built in)
- Dynamic Mode Decomposition
- Edge Detection (Matlab built in)
- Color Classification

We apply these techniques on a group of 1000 videos of a single speaker from the GRID audiovisual sentence corpus database. Each video is 3 second long and contains the speaker saying a number of nonsense phrases. The phrases spoken are chosen to convey a variety of sounds and therefore lip positions.

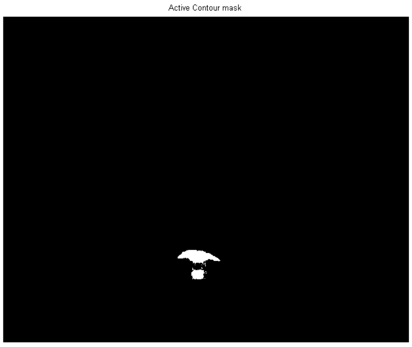

Figure 1: Active Contour

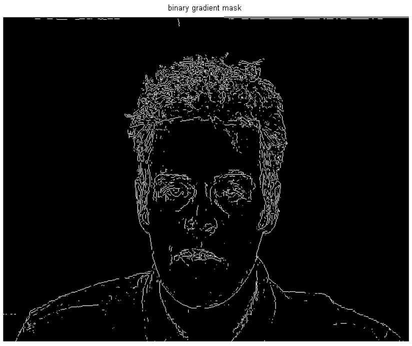

Figure 2: Binary Gradient

2 Theoretical Background

2.1 Feature Extraction

Lip reading is a complicated task and there are no "go-to" algorithms for detecting and tracking the position of an individual's lips. We can use Matlab's built in active contour and edge detection to hope to do background/foreground seperation on the video. We can also use DMD to do background/foreground seperation. The assumption is the speaker's face is stationary enough that it is possible to detect the lips as the foreground. In practice, this is not the case. We can also classify the pixel colors of the video frames and segment the lips based on the idea that the speaker's lip color is different from their skin color. For color classification, we can project the pixel color into the LAB color space to acheive better classification results. In either of these different strategies we also need to isolate the general mouth region of the speaker so that we will not also detect eyes, nose, etc.

2.2 Phonemes and Visemes

Phonemes are the smallest identifiable sound unit in the sound system of a language. [6] According to Zhong, et al, phonemes are "basic sound segments that carry linguistic distinction."[7] In theory, visemes are the analogous basic units in the visual domain. However, there is no agreement on a common definition for visemes in practice[8]. In audio speech recognition, phonemes are detected and used to reconstruct speech. In visual speech recognition, only visemes can be detected. Phonemes for the basis for a spoken language, and hence automated lipreading typically employs a mapping from phonemes to visemes. Many such mapping can be found in the literature, but all suffer from the issue of there being more phonemes than visemes, resulting in a many-to-one map.[8] For instance, this project uses 37 phonemes and 11 visemes. See Table (1) for the phoneme to viseme map used in this project.

2.3 Hidden Markov Models

A Markov model involves the transition of a particular state to other states based on transition probabilities. A future state is only depends on the current state and not the states before it. Now, consider that at every state, there would be real world observations. These observations are controlled by the emission probabilities at each state.

For example, if we were to represent the ever changing weather, the states would be sunny, rainy or snowing and the observations would be summer clothes, rain boots or snow shoes. We can see that the emission probabilities for each observation is different depending on the state. To be more clear, the emission probabilities depend on the states.

Table 1: Phoneme to Viseme Map from Lee and York, 2000, via [8].

Viseme Number	Viseme Label	Associated Phonemes
1	P	b p m
2	T	d t s z th dh
3	K	g k n l y hh
4	CH	jh ch
5	F	f v
6	W	r w
7	IY	iy ih
8	EH	eh ey ae
9	AA	aa aw ay ah
10	A0	ao oy ow
11	UH	uh uw

We look at Hidden Markov Models(HMM). We decide that this is a relevant model because the words spoken are those defined by language and thus occur in specific pattern and not randomly. For example, given the first letter of word 'k', the probability that the next letter is a vowel is much higher than it being a consonant. A machine learning algorithm without this would be as inefficient as the initial Enigma machine in the movie "The Imitation Game". HMMs are very popular in the fields of speech [2] and gesture recognition [4] [5].

Although HMMs have fascinating problems related to evaluation and learning, our interests are in decoding. We have a sequence of observations and our aim is to estimate the states that created that. The Viterbi algorithm [3] gives us the states that maximize the occurrence of the observations.

So, given the features from the videos, we find the states. The states are the units of words, here chosen to be phonemes.

3 Implementation and Development

3.1 Feature extraction

The first step is to determine the general location of the mouth in each frame. Matlab has a built in facial recognition package "Cascade Vision Detector" which uses the Viola-Jones algorithm to detect the facial features of people. We use the Cascade Mouth detector to determine the location of the face in each frame of each movie. This algorithm is not perfect however and we often detect the eyes or chin of the subject in question. We filter out these false detections by looking at the position of the mouth. The speaker is in generally the same location in each movie so we know if a detected mouth position is too low or too high we can throw that region out. We refer to this detected mouth region as the *initial mask*. We need a good initial mask in order to properly filter our results. Without knowing the general mouth region our algorithms will act on the whole face, but we consider only the lips for our project.Therefore, really our objective is to seperate the lips from the mouth region.

Next we convert the image to grayscale. On the grayscale image we can apply Matlab's built in active contour and edge detection, as well as the DMD algorithm we develop in homework 4. We can experiment with the different segmentation types for Matlab's active contour and edge detection. For active contour, Chan-Vese gives poor results with irregular edges. Edge method for active contour gives smoother regions but irregular shapes. Active contour takes many iterations to properly converge and gives poor results. From the figures of active contour, we see the inconsistent lip region detected. Because we are trying to determine the changes in lip shape over time, active contour is not suitable because the results it produces vary too much. Additionally, because it is extremely slow, active contour is not ideal for our purposes. We need to process a large number of videos so a fast algorithm is preferable. If active contour was more computationally efficient it might be possible to use a large number of iterations for each frame in order to obtain a consistent result.

Matlab's built in edge detection has many different methods which produce different results. We consider using Canny, Prewitt and Sobel edge detection.Canny typically finds the "strong" edges whereas Sobel and Prewitt tend to pick out more detail in the image. We are really only interested in the lips, and ideally want only 1 contour, so we decide to use Canny edge detection. We want to detect the outside and inside edge of the lips, but all methods typically detect the inside edge of the lip. When using Matlab's "edge", it is important to consider the threshold. Too low a threshold value will produce other features of the face and possibly noise. Too high a threshold and we will not pick out the entire lip. Therefore we decide to start at a high threshold value, and if we do not find a large enough edge, we lower the threshold and do the detection again. Typically we only need the lower the threshold once or twice. We set a minimum number of detected pixels in the edge to decide whether or not to automatically lower the threshold. Although this may seem expensive, in practice we only need to lower the threshold for certain frames, and so most of the time we only need to do the algorithm once. Additionally, edge is much faster than active contour, so this method is still much faster than active contour, and gives more consistent results in well. Overall the Canny edge detection finds a good edge for the lips but it oftentimes only picks out part of the mouth or sometimes picks out other features around the mouth as well, like the chin or space between the nose and mouth.

Next we use the DMD algorithm we developed in homework 4 to try foreground/background seperation. We find that the face moves too much, and there is too much variation in the speaker's skin around the mouth to properly use DMD. We detect many points that are not on the lip. While DMD is good at seperating the man's face from the background, it cannot accurately seperate the lips from mouth region. It might be possible to alter the brightness and contrast of the seperated image using DMD to obtain a better result, but it would still be too inconsistent to obtain good results for the rest of the project.

Thus far, all of the methods discussed have acted on grayscale frames of the color movie. We now consider a technique for acting on the color frames. If we assume that the lips of the speaker have a distinct color from the rest of the speaker's mouth region (i.e. the skin, teeth, inside of the mouth) then we can determine the lip region by classifying the colors of the mouth region. It is possible to do this using the RGB values of each pixel. We instead represent each pixel with it's LAB color space values. The point of LAB colors is that they match how human vision perceives colors. The L stands for lighting and AB are values corresponding to color. By classifying each pixel into one of up to 4 clusters based on it's AB values with a k-means algorithm, we can seperate the lips from the rest of the mouth region. The number of clusters created varies per frame because in some frames, the teeth and inner mouth can be grouped in their own clusters. Other frames, the speaker has their mouth closed so only 2 clusters are needed. We can differentiate between the clusters by looking at the calculated centroid positions of each cluster. It is determined experimentally that the lip region has the highest A color value, and it usually has the lowest B color value as well. Color classification is fairly fast and accurate. It gives the thickness as well as the shape of the lips. It's only downside is that it often classifies the left and right sides of the lips into a different cluster. Color classification typically picks out an ellipse in the middle of the lips as opposed to the whole lip region. Overall, color classification is determined to give the most accurate results, and edge detection is a close second. In hindsight, it would have made much more sense to use a classification algorithm, like knn-search, instead of k-means. Although we choose the wrong algorithm to use our results are still acceptable.

In general the best results come from either more computationally expensive or mathematically sophisticated techniques. For example, training a neural net to classify the lip region of each frame would probably be the most effective technique overall, but would be very computationally costly and require manually determining the lip region for training. Many papers in the literature consider fitting a vector to fit the lip region and then track the lips by moving points of the vector and measuring the change of shape. They then can compute the most likely shape by comparing it to known positions of the lip. This method relies more heavily on analysis but also requires manual training as well. In general, lip tracking is a classification problem where the goal is to classify the pixels that belong to the lip of the speaker.

3.2 Extracting Phonemes

Using the nltk library in Python, we convert every word to its constitutive arpabet phonetics. It gives the following output for the words - f', 'see', 'sea', 'compute', 'comput', 'cat'. Only 'comput' fails because it isn't a real word

```
['EH1', 'F']
['S', 'IY1']
['S', 'IY1']
['K', 'AH0', 'M', 'P', 'Y', 'UW1', 'T']
'comput'
['K', 'AE1', 'T']
```

From the words spoken in our videos, we get a set of 36 unique phonemes. The code for this is shown in using B.1 The 37 unique phonemes are

```
{'R', 'F', 'IH0', 'N', 'AA1', 'IY1', 'OW0', 'EH1', 'G', 'HH', 'DH', 'M', 'OW1', 'V', 'JH', 'IH1', 'A
```

3.3 Extracting Transcripts

From the transcripts, we extract all the data into a csv file using B.2

3.4 Extracting Subtitles and Assigning Phonemes

The Corpus Grid II database contains a *.txt* file for each video containing the words spoken and corresponding frames for each word. These were downloaded and extracted into Matlab using the *textscan* function in Matlab. The words were deconstructed into phonemes, and phonemes were assigned to each frame. The assignment of a phoneme label to each frame was done by assigning each phoneme from a word to an equal number of the video frames corresponding to each word.

3.5 Classification

The data matrix was created by reshaping each frame in each video in to a single column. When each column was reshaped, its saved phoneme was checked, and the corresponding viseme index was saved to create the labels for a classification algorithm. The columns for each video frame were then concatenated in order to form the data matrix. The singular value decomposition was computed of the matrix of video frames. Classification was performed on first 30 columns of the V matrix using both a Naive Bayes and a k-nearest neighbors algorithm. Classification was done with a random 75% of the data used for training and the remaining 25% of the data for cross validation.

4 Computational Results

4.1 Classification

For phoneme identification, the classification using a k-nearest neighbors algorithm only 11.6103% accuracy on average on the cross validation over 176 trials. For viseme identification, the k-nearest neighbors method had 19.7355% accuracy over 30 trials. More trials were not performed due to the Matlab *knnsearch* function being computationally intensive.

The naive bayes classification algorithm applied to identificaiton of phonemes had an average accuracy of 3.49% over 1000 trials. For viseme classification, the naive bayes algorithm had an average accuracy of 9.1357% over 1000 trials.

4.2 HMM

Word Classification using 37 phonemes		
Word	Set	Accuracy
bin	bin , blue	87.5 %
blue	bin, blue	36 %
blue	red, blue	76 %
four	four, white	60 %
bin	bin , white	62.5 %
five	blue , five	60 %
red	red , eight	72 %
bin	bin , blue, green	75 %
green	green, white, five	44 %
five	five, blue, four, white	50 %
green	green, white, five, red	28 %
bin	bin , blue, green , red	75 %
bin	bin , blue, red, white	56.2 %
blue	bin , blue, red, white	28 %
five	four , five, red, white	45 %
bin	bin , blue, green , red, eight	75 %
bin	bin , blue, green , red, white	50 %
four	four , five, green , red, white	30 %
five	four , five, green , red, white	40 %

Word Classification using 11 Visemes		
Word	Set	Accuracy
bin	bin , blue	50 %
blue	bin, blue	68 %
blue	red, blue	92 %
four	four, white	60 %
bin	bin , white	68.8 %
five	blue , five	60 %
red	red , eight	65 %
bin	bin , blue, green	25 %
red	red , white, green	35 %
green	green, white, five	44 %
five	five, blue, four, white	50 %
green	green, white, five, red	28 %
bin	bin , blue, green , red	12.5 %
bin	bin , blue, red, white	12.5%
blue	bin , blue, red, white	68 %
five	four , five, red, white	45 %
bin	bin , blue, green , red, eight	12.5 %
bin	bin , blue, green , red, white	12.5 %
four	four , five, green , red, white	30 %
five	four , five, green , red, white	40 %

5 Summary and Conclusions

This report detailed the classification of phonemes and visemes based upon visual data of a person speaking, and speech prediction based upon identified phonemes and visemes using Hidden Markov Models. This is a complex problem that is typically handled by a Long Short-Term Memory (LSTM) recurrent neural network. The results presented here seem very poor, but considering the complex nature of the problem, the results are reasonable.

The best way to improve the results in this project is to improve the lip detection. The videos used in this project were of low quality, making the accurate detection of lip contours difficult. The database used has higher quality versions of the videos, however high quality videos are ≈ 2.4 Gb each, for a total of 2.4 Terabytes of data for the whole data set. This is an unrealistic amount of data for the hardware available for this project. Another short coming of this project is the assignment of an equal number video frames to each phoneme from the given frame locations of each word. This is a poor way to determine which phoneme is being spoken in each video frame. Results could have further been improved with a more sophisticated classification algorithm, for instance a classification tree or neural network.

References

[1] J. Proctor, S. Brunton and J. N. Kutz, Dynamic mode decomposition with control, arXiv:1409.6358.

[2] Rabiner, Lawrence R. ”A tutorial on hidden Markov models and selected applications in speech recognition.” Proceedings of the IEEE 77.2 (1989): 257-286.

[3] Forney, G. David. ”The viterbi algorithm.” Proceedings of the IEEE 61.3 (1973): 268-278.

[4] Yang, Jie, and Yangsheng Xu. Hidden markov model for gesture recognition. No. CMU-RI-TR-94-10. CARNEGIE-MELLON UNIV PITTSBURGH PA ROBOTICS INST, 1994.

[5] Starner, Thad E. Visual Recognition of American Sign Language Using Hidden Markov Models. MASSACHUSETTS INST OF TECH CAMBRIDGE DEPT OF BRAIN AND COGNITIVE SCIENCES, 1995.

[6] Hassanat, Ahmad B., ‘Visual Words for Automatic Lip- Reading.’ PhD diss., University of Buckingham, 2009.

[7] J. Zhong, W. Chou, and E. Petajan, ‘Acoustic Driven Viseme Identification for Face Animation.’ Bell Laboratories. Murray Hill, NJ. IEEE 0-7803-378. Aug. 1997.

[8] L. Cappelletta and N. Harte. ‘Phoneme-to-Viseme Mapping for Visual Speech.’ Department of Electronic and Electrical Engineering, Trinity College Dublin, Ireland. May 2012.

A MATLAB Code

A.1 Contours.m

```
obj=VideoReader('vid1.mpg');
vidFrames = read(obj);
numFrames = get(obj,'numberOfFrames');
[mov]= getmovout(vidFrames,numFrames-1);
X=frame2im(mov(50));
A=rgb2gray(X);
mask = zeros(size(A));
mask(400:450,320:400) = 1;
bw = activecontour(A,mask,300);
figure, imshow(bw), title('Active Contour mask');
[~, threshold] = edge(A, 'sobel');
fudgeFactor = .5;
BWs = edge(A,'sobel', threshold * fudgeFactor);
figure, imshow(BWs), title('binary gradient mask');
```

A.2 Assign Labels

```
function [ labels ] = assignlabels2(cropvid,frameLocs,words)
%UNTITLED10 Summary of this function goes here
%   Detailed explanation goes here
%Inputs:
%cropped videos (struct)
%frameLocs
% words
%
%
% Outputs:
% labels: cell array of cells containing the phoneme at each frame
%
%Begin Function

numVids = length(cropvid);

labels = cell(numVids,1);

for k = numVids:-1:1
    v = cropvid{k};
    fL = floor(frameLocs{k}/1000);
    lab = cell(size(v,3),1);
    for j = 1:length(fL)-1
        %Break this word into phonemes
        ph = assignphoneme(words{k}(j));
        %Get number of phonemes in this word
        numPh = length(ph);
        %get frame indices for this word to be distributed accross
        x1 = fL(j);
        x2 = fL(j+1);
        %number of frames per phoneme
        nf = round((x2-x1)/numPh);
        xprev = x1;
        if xprev == 0
            xprev = 1; %make sure 0 index isnt called
        end
        for i = 1:numPh-1
            xnext = xprev + nf; %overwrite next
            for ii = xprev:xnext-1
                lab{ii} = ph{i};
            end
            xprev = xnext; %overwrite prev
        end
        for ii = xprev:size(v,3)
            lab{ii} = ph{end};
        end
    end
    labels{k} = lab; %store in output
    clear lab %clear temp variable
end
end
```

A.3 SVD and Classify

```

% Required functions:

% lipcrop
% assignlabels
% assignphoneme

%Required variables to already be in workspace:
%
% frameLocs
% words
%Begin Script:

clearvars -except lipread frameLocs words
%% Crop, Assign Labels, create individual data matrices

%Crop videos based on contour mask
cropVid = lipcrop(lipread);

%Assign labels to each frame
labels = assignlabels2(cropVid,frameLocs,words);

%Sort and create data arrays
[X,tags,vinds] = hmmdata(cropVid,labels);
numPix = size(X,1);
numVids = size(X,2);

%% SVD

[U,S,V] = svd(X,'econ');

%% Create Classification Matrices

for k = 1:1000
    q = randperm(numVids);
    qind = round(numVids*0.75);
    q1 = q(1:qind);
    q2 = q(qind+1:end);

    trainData = V(q1,1:30);
    testData = V(q2,1:30);
    trainTags = tags(q1)';
    testTags = tags(q2)';

    hmmNBdata = V(:,1:30);

    nb = NaiveBayes.fit(trainData,trainTags);
    pre = nb.predict(testData);

    acc(k) = 100*sum(pre==testTags)/length(pre);
end

```

```
%compute accuracy

%disp(['Accuracy was ' num2str(acc) '%'])

knnind = knnsearch(trainData,testData);
acc2 = 100*sum(knnind==testTags)/length(knnind);
% disp(['Accuracy was ' num2str(acc2) '%'])
```

A.4 Creation of Data Matrix and Classification Labels

```
function [X,tags,vinds] = hmmdata(cropVid,labels)
%UNTITLED Summary of this function goes here
%   Detailed explanation goes here

kmax =size(cropVid{1}(:,:,:),3);
numPix = size(cropVid{1}(:,:,:),1)*size(cropVid{1}(:,:,:),2);
numVids = kmax*length(cropVid);

%Initialize Data Matrix
counter = 1;
vinds = cell(length(cropVid),1);
X = zeros(numPix,36749);
for j = length(cropVid):-1:1
    thisLab = labels{j};
    thisVid = cropVid{j}(:,:,:);
    numv = 1;
    for k = 1:size(thisVid,3)
        %Get DM index of this phoneme
        phonemeInd = checkviseme(thisLab(k));

        %If SIL, store in SIL array
        if phonemeInd <= 0
            continue
        end
        %this phoneme is a regular phoneme:

        %Increment to store in right place

        %Get Frame to store
        thisFrame = thisVid(:,:,k);

        %Reshape frame
        xframe = reshape(thisFrame,numPix,1);

        %Store frame in corresponding DM
        X(:,counter) = xframe;
        tags(counter) = phonemeInd;
        vinds{j}(numv) = counter;
        counter = counter +1;
        numv = numv + 1;
    end
end
end
```

B Python Code

B.1 Phonemes.py

```
import nltk
```

```
arpabet = nltk.corpus.cmudict.dict()
k =[j for j in ’abcdefghijklmnopqrstuvwxyz’]
t= [’again’, ’soon’, ’now’, ’please’,’bin’, ’lay’, ’place’,’set’, ’blue’, \
 ’green’,’red’,’white’ ,’at’,’by’, ’with’, ’in’, ’zero’,’one’, ’two’,\
 ’three’,’four’,’five’,’six’,’seven’,’eight’,’nine’];
g = k+t
ph =[]
for word in g:
        wl =arpabet[word]
        myString = ’ ’.join(str(r) for v in wl for r in v)
        print( word+’ :’+ myString)
        for w in wl:
                ph = ph +w
uniqueph = set(ph)
```

B.2 Transcripts.py

```
import csv
import os
os.chdir("align")
beach = os.listdir()
with open("tes.csv", "w") as f:
        for sand in beach[:-1]:
                text_file = open(sand, "r")
                lines = text_file.read().split(’,’)
                k = lines[0]
                g = k.split()
                writer = csv.writer(f)
                writer.writerow(g)
```