Estimating speech from lip movement

Jithin D. George, Ronan Keane, Conor Zellmer

March 16, 2017

Abstract

1 Introduction and Overview

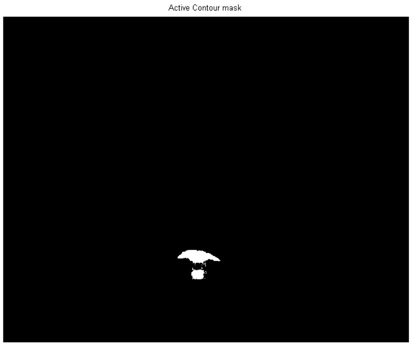

Figure 1: Active Contour

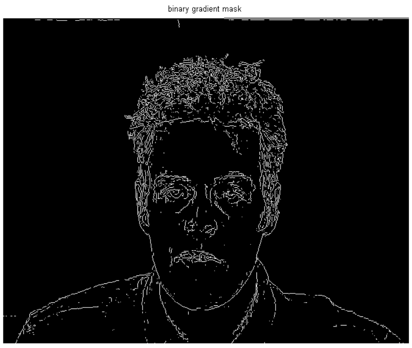

Figure 2: Binary Gradient

2 Theoretical Background

2.1 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.

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

2.2 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.

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 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
```

The list of words and the phonemes in them are given by

```
a :AH0 EY1
b :B IY1
c :S IY1
d :D IY1
e :IY1
f :EH1 F
g :JH IY1
h :EY1 CH
i :AY1
j :JH EY1
k :K EY1
l :EH1 L
m :EH1 M
n :EH1 N
o :OW1
p :P IY1
q :K Y UW1
r :AA1 R
s :EH1 S
t :T IY1
u :Y UW1
v :V IY1
w :D AH1 B AH0 L Y UW0
x :EH1 K S
y :W AY1
z :Z IY1
again :AH0 G EH1 N AH0 G EY1 N
soon :S UW1 N
now :N AW1
please :P L IY1 Z
bin :B IH1 N
lay :L EY1
place :P L EY1 S
```

```
set :S EH1 T
blue :B L UW1
green :G R IY1 N
red :R EH1 D
white :W AY1 T HH W AY1 T
at :AE1 T
by :B AY1
with :W IH1 DH W IH1 TH W IH0 TH W IH0 DH
in :IH0 N IH1 N
zero :Z IH1 R OW0 Z IY1 R OW0
one :W AH1 N HH W AH1 N
two :T UW1
three :TH R IY1
four :F AO1 R
five :F AY1 V
six :S IH1 K S
seven :S EH1 V AH0 N
eight :EY1 T
nine :N AY1 N
```

3.2 Extracting Transcripts

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

3.3 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.

3.4 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	75 %
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	75 %
bin	bin , blue, red, white	56.2 %
blue	bin , blue, red, white	68 %
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 %

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)
```