# Audio-Visual Speech Recognition using SciPy

Helge Reikeras, Ben Herbst, Johan du Preez, Herman Engelbrecht

**Abstract**—In audio-visual automatic speech recognition (AVASR) both acoustic and visual modalities of speech are used to identify what a person is saying. In this paper we propose a basic AVASR system implemented using SciPy, an open source Python library for scientific computing. AVASR research draws from the fields of signal processing, computer vision and machine learning, all of which are active fields of development in the SciPy community. As such, AVASR researchers using SciPy are able to benefit from a wide range of tools available in SciPy.

The performance of the system is tested using the Clemson University audio-visual experiments (CUAVE) database. We find that visual speech information is in itself not sufficient for automatic speech recognition. However, by integrating visual and acoustic speech information we are able to obtain better performance than what is possible with audio-only ASR.

**Index Terms**—speech recognition, machine learning, computer vision, signal processing

## Introduction

Motivated by the multi-modal manner humans perceive their environment, research in Audio-Visual Automatic Speech Recognition (AVASR) focuses on the integration of acoustic and visual speech information with the purpose of improving accuracy and robustness of speech recognition systems. AVASR is in general expected to perform better than audio-only automatic speech recognition (ASR), especially so in noisy environments as the visual channel is not affected by acoustic noise.

Functional requirements for an AVASR system include acoustic and visual feature extraction, probabilistic model learning and classification.

In this paper we propose a basic AVASR system implemented using SciPy. In the proposed system mel-frequency cepstrum coefficients (MFCCs) and active appearance model (AAM) parameters are used as acoustic and visual features, respectively. Gaussian mixture models are used to learn the distributions of the feature vectors given a particular class such as a word or a phoneme. We present two alternatives for learning the GMMs, expectation maximization (EM) and variational Bayesian (VB) inference.

The performance of the system is tested using the CUAVE database. The performance is evaluated by calculating the misclassification rate of the system on a separate test data data. We find that visual speech information is in itself not sufficient for automatic speech recognition. However, by integrating visual and acoustic speech we are able to obtain better performance than what is possible with audio-only ASR.

*Helge Reikeras, Ben Herbst, and Johan du Preez are with the Stellenbosch University. E-mail: helge@ml.sun.ac.za.*

## Feature extraction

### Acoustic speech

MFCCs are the standard acoustic features used in most modern speech recognition systems. In [Dav80] MFCCs are shown experimentally to give better recognition accuracy than alternative parametric representations.

MFCCs are calculated as the cosine transform of the logarithm of the short-term energy spectrum of the signal expressed on the mel-frequency scale. The result is a set of coefficients that approximates the way the human auditory system perceives sound.

MFCCs may be used directly as acoustic features in an AVASR system. In this case the dimensionality of the feature vectors equals the number of MFCCs computed. Alternatively, velocity and acceleration information may be included by appending first and second order temporal differences to the feature vectors.

The total number of feature vectors obtained from an audio sample depends on the duration and sample rate of the original sample and the size of the window that is used in calculating the cepstrum (a windowed Fourier transform).

MFCCs are available in `scikits.talkbox.features.mfcc`. The default number of MFCCs computed is thirteen.

Example usage:

```
from scikits.audiolab import wavread
from scikits.talkbox.features import mfcc

# data: raw audio data
# fs: sample rate
data, fs = wavread('sample.wav')[:2]

# ceps: cepstral cofficients
ceps = mfcc(input, fs=fs)[0]
```

Figure 1 shows the original audio sample and mel-frequency cepstrum for the word “zero”.

### Visual speech

While acoustic speech features can be extracted through a sequence of transformations applied to the input audio signal, extracting visual speech features is in general more complicated. The visual information relevant to speech is mostly contained in the motion of visible articulators such as lips, tongue and jaw. In order to extract this information from

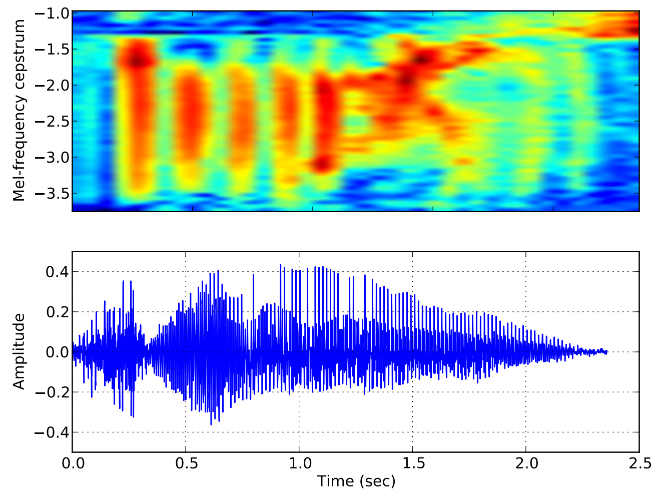

***Fig. 1:*** *Acoustic feature extraction from an audio sample of the word "zero". Mel-frequency cepstrum (top) and original audio sample (bottom).*

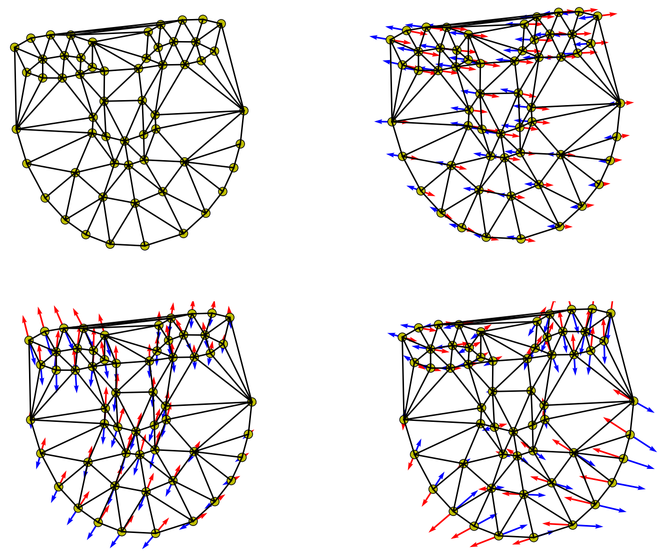

***Fig. 2:*** *Triangulated base shape* $\mathbf{s}_0$ *(top left), and first three shape vectors* $\mathbf{p}_1$ *(top right),* $\mathbf{p}_2$ *(bottom left) and* $\mathbf{p}_3$ *(bottom right) represented by arrows superimposed onto the triangulated base shape.*

a sequence of video frames it is advantageous to track the complete motion of the face and facial features.

AAM [Coo98] fitting is an efficient and robust method for tracking the motion of deformable objects in a video sequence. AAMs model variations in shape and texture of the object of interest. To build an AAM it is necessary to provide sample images with the shape of the object annotated. Hence, in contrast to MFCCs, AAMs require prior training before being used for tracking and feature extraction.

The shape of an appearance model is given by a set of $(x,y)$ coordinates represented in the form of a column vector

$$\mathbf{s} = (x_1, y_1, x_2, y_2, \ldots, x_n, y_n)^{\mathrm{T}}. \qquad (1)$$

The coordinates are relative to the coordinate frame of the image.

Shape variations are restricted to a base shape $\mathbf{s}_0$ plus a linear combination of a set of $N$ shape vectors

$$\mathbf{s} = \mathbf{s}_0 + \sum_{i=1}^{N} p_i \mathbf{s}_i \qquad (2)$$

where $p_i$ are called the shape parameters of the AAM.

The base shape and shape vectors are normally generated by applying principal component analysis (PCA) to a set of manually annotated training images. The base shape $\mathbf{s}_0$ is the mean of the object annotations in the training set, and the shape vectors are $N$ singular vectors corresponding to the $N$ largest singular values of the data matrix (constructed from the training shapes). Figure 2 shows an example of a base mesh and the first three shape vectors corresponding to the three largest singular values of the data matrix.

The appearance of an AAM is defined with respect to the base shape $\mathbf{s}_0$. As with shape, appearance variation is restricted to a base appearance plus a linear combination of $M$ appearance vectors

$$A(\mathbf{x}) = A_0 + \sum_{i=1}^{M} \lambda_i A_i(\mathbf{x}) \qquad \forall \mathbf{x} \in \mathbf{s}_0. \qquad (3)$$

To generate an appearance model, the training images are first shape-normalized by warping each image onto the base mesh using a piecewise affine transformation. Recall that two sets of three corresponding points are sufficient for determining an affine transformation. The shape mesh vertices are first triangulated. The collection of corresponding triangles in two shapes meshes then defines a piecewise affine transformation between the two shapes. The pixel values within each triangle in the training shape $\mathbf{s}$ are warped onto the corresponding triangle in the base shape $\mathbf{s}_0$ using the affine transformation defined by the two triangles.

The appearance model is generated from the shape-normalized images using PCA. Figure 3 shows the base appearance and the first three appearance images.

Tracking of an appearance in a sequence of images is performed by minimizing the difference between the base model appearance, and the input image warped onto the coordinate frame of the AAM. For a given image $I$ we minimize

$$\operatorname*{argmin}_{\boldsymbol{\lambda},\mathbf{p}} \sum_{\mathbf{x}\in\mathbf{s}_0} \left[ A_0(\mathbf{x}) + \sum_{i=1}^{M} \lambda_i A_i(\mathbf{X}) - I(\mathbf{W}(\mathbf{x};\mathbf{p})) \right]^2 \qquad (4)$$

where $\mathbf{p} = \{p_1, \ldots, p_N\}$ and $\boldsymbol{\lambda} = \{\lambda_1, \ldots, \lambda_N\}$. For the rest of the discussion of AAMs we assume that the variable $\mathbf{x}$ takes on the image coordinates contained within the base mesh $\mathbf{s}_0$ as in (4).

In (4) we are looking for the optimal alignment of the input image, warped backwards onto the frame of the base appearance $A_0(\mathbf{x})$.

For simplicity we will limit the discussion to shape variation and ignore any variation in texture. The derivation for the case including texture variation is available in [Mat03].

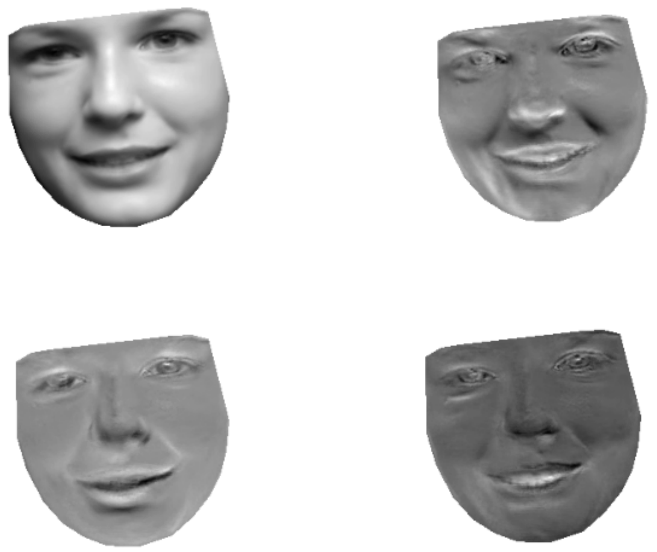

***Fig. 3:*** *Mean appearance $A_0$ (top left) and first three appearance images $A_1$ (top right), $A_2$ (bottom left) and $A_3$ (bottom right).*

Consequently (4) now reduces to

$$\underset{\mathbf{p}}{\operatorname{argmin}} \sum_{\mathbf{x}} [A_0(\mathbf{x}) - I(\mathbf{W}(\mathbf{x};\mathbf{p}))]^2. \tag{5}$$

Solving (5) for $\mathbf{p}$ is a non-linear optimization problem. This is the case even if $\mathbf{W}(\mathbf{x};\mathbf{p})$ is linear in $\mathbf{p}$ since the pixel values $I(\mathbf{x})$ are in general nonlinear in $\mathbf{x}$.

The quantity that is minimized in (5) is the same as in the classic Lucas-Kanade image alignment algorithm [Luc81]. In the Lukas-Kanade algorithm the problem is first reformulated as

$$\underset{\Delta\mathbf{p}}{\operatorname{argmin}} \sum_{\mathbf{x}} [A_0(\mathbf{X}) - I(\mathbf{W}(\mathbf{x};\mathbf{p}+\Delta\mathbf{p}))]^2. \tag{6}$$

This equation differs from (5) in that we are now optimizing with respect to $\Delta\mathbf{p}$ while assuming $\mathbf{p}$ is known. Given an initial estimate of $\mathbf{p}$ we update with the value of $\Delta\mathbf{p}$ that minimizes (6) to give

$$\mathbf{p}^{\mathrm{new}} = \mathbf{p} + \Delta\mathbf{p}.$$

This will necessarily decrease the value of (5) for the new value of $\mathbf{p}$. Replacing $\mathbf{p}$ with the updated value for $\mathbf{p}^{\mathrm{new}}$, this procedure is iterated until convergence at which point $\mathbf{p}$ yields the (local) optimal shape parameters for the input image $I$.

To solve (6) Taylor expansion is used [Bak01] which gives

$$\underset{\Delta\mathbf{p}}{\operatorname{argmin}} \sum_{\mathbf{x}} \left[A_0(\mathbf{W}(\mathbf{x};\mathbf{p})) - I(\mathbf{W}(\mathbf{x};\mathbf{p})) - \nabla I \frac{\partial \mathbf{W}}{\partial \mathbf{p}} \Delta\mathbf{p}\right]^2 \tag{7}$$

where $\nabla I$ is the gradient of the input image and $\partial\mathbf{W}/\partial\mathbf{p}$ is the Jacobian of the warp evaluated at $\mathbf{p}$.

The optimal solution to (7) is found by setting the partial derivative with respect to $\Delta\mathbf{p}$ equal to zero which gives

$$2\sum_{\mathbf{x}} \left[\nabla \mathbf{I} \frac{\partial \mathbf{W}}{\partial \mathbf{p}}\right]^{\mathrm{T}} \left[A_0(\mathbf{x}) - I(\mathbf{W}(\mathbf{x})) - \nabla I \frac{\partial \mathbf{W}}{\partial \mathbf{p}} \Delta\mathbf{p}\right] = 0. \tag{8}$$

Solving for $\Delta\mathbf{p}$ we get

$$\Delta\mathbf{p} = \mathbf{H}^{-1} \sum_{\mathbf{x}} \left[\nabla I \frac{\partial \mathbf{W}}{\partial \mathbf{p}}\right]^{\mathrm{T}} [A_0(\mathbf{x}) - I(\mathbf{W}(\mathbf{x};\mathbf{p}))] \tag{9}$$

where $\mathbf{H}$ is the Gauss-Newton approximation to the Hessian matrix given by

$$\mathbf{H} = \sum_{\mathbf{x}} \left[\nabla I \frac{\partial \mathbf{W}}{\partial \mathbf{p}}\right]^{\mathrm{T}} \left[\nabla I \frac{\partial \mathbf{W}}{\partial \mathbf{p}}\right]. \tag{10}$$

For a motivation for the backwards warp and further details on how to compute the piecewise linear affine warp and the Jacobian see [Mat03].

A proper initialization of the shape parameters $\mathbf{p}$ is essential for the first frame. For subsequent frames $\mathbf{p}$ may be initialized as the optimal parameters from the previous frame.

The Lucas-Kanade algorithm is a Gauss-Newton gradient descent algorithm. Gauss-Newton gradient descent is available in `scipy.optimize.fmin_ncg`.

Example usage:

```python
from scipy import empty
from scipy.optimize import fmin_ncg
from scikits.image.io import imread

# NOTE: The AAM module is currently under development
import aam

# Initialize AAM from visual speech training data
vs_aam = aam.AAM('./training_data/')

I = imread('face.jpg')

def error_image(p):
    """ Compute error image given p """

    # Piecewise linear warp the image onto
    # the base AAM mesh
    IW = vs_aam.pw_affine(I,p)

    # Return error image
    return aam.A0-IW

def gradient_descent_images(p):
    """ Compute gradient descent images given p """
    ...
    return gradIW_dWdP

def hessian(p):
    """ Compute hessian matrix """"
    ...
    return H

# Update p
p = fmin_ncg(f=error_image,
             x0=p0,
             fprime=gradient_descent_images,
             fhess=hessian)
```

Figure 4 shows an AAM fitted to an input image. When tracking motion in a video sequence an AAM is fitted to each frame using the previous optimal fit as a starting point.

In [Bak01] the AAM fitting method described above is referred to as “forwards-additive”.

As can be seen in Figure 2 the first two shape vectors mainly correspond to the movement in the up-down and left-right directions, respectively. As these components do not contain any speech related information we can ignore the corresponding shape parameters $p_1$ and $p_2$ when extracting visual speech features. The remaining shape parameters, $p_3, \ldots, p_N$, are used as visual features in the AVASR system.

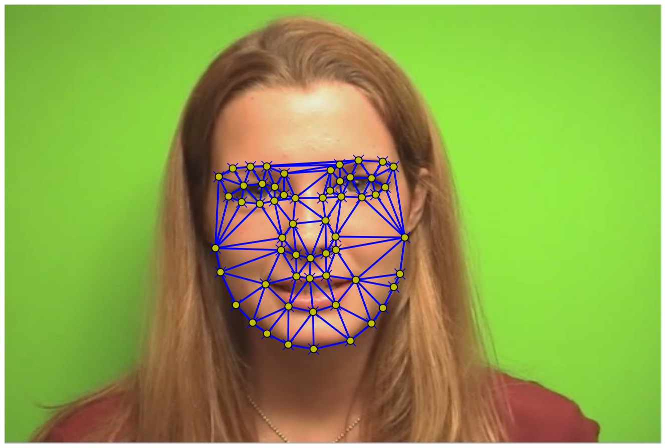

*Fig. 4: AAM fitted to an image.*

## Models for audio-visual speech recognition

Once acoustic and visual speech features have been extracted from respective modalities, we learn probabilistic models for each of the classes we need to discriminate between (e.g. words or phonemes). The models are learned from manually labeled training data. We require these models to *generalize* well; i.e. the models must be able to correctly classify novel samples that was not present in the training data.

### Gaussian Mixture Models

Gaussian Mixture Models (GMMs) provide a powerful method for modeling data distributions under the assumption that the data is independent and identically distributed (i.i.d.). GMMs are defined as a weighted sum of Gaussian probability distributions

$$p(\mathbf{x}) = \sum_{k=1}^{K} \pi_k \mathcal{N}(\mathbf{x}|\boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k) \tag{11}$$

where $\pi_k$ is the weight, $\boldsymbol{\mu}_k$ the mean, and $\boldsymbol{\Sigma}_k$ the covariance matrix of the $k$th mixture component.

### Maximum likelihood

The log likelihood function of the GMM parameters $\boldsymbol{\pi}$, $\boldsymbol{\mu}$ and $\boldsymbol{\Sigma}$ given a set of D-dimensional observations $\mathbf{X} = \{\mathbf{x}_1, \ldots, \mathbf{x}_N\}$ is given by

$$\ln p(\mathbf{X}|\boldsymbol{\pi}, \boldsymbol{\mu}, \boldsymbol{\Sigma}) = \sum_{n=1}^{N} \ln \left\{ \sum_{k=1}^{K} \pi_k \mathcal{N}(\boldsymbol{x}_n|\boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k) \right\}. \tag{12}$$

Note that the log likelihood is a function of the GMM parameters $\boldsymbol{\pi}, \boldsymbol{\mu}$ and $\boldsymbol{\Sigma}$. In order to fit a GMM to the observed data we maximize this likelihood with respect to the model parameters.

### Expectation maximization

The Expectation Maximization (EM) algorithm [Bis07] is an efficient iterative technique for optimizing the log likelihood function. As its name suggests, EM is a two stage algorithm. The first (*E* or *expectation*) step calculates the expectations for each data point to belong to each of the mixture components. It is also often expressed as the *responsibility* that the $k$th mixture component takes for "explaining" the $n$th data point, and is given by

$$r_{nk} = \frac{\pi_k \mathcal{N}(\mathbf{x}_n|\boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k)}{\sum_{k=1}^{K} \pi_k \mathcal{N}(\mathbf{x}_n|\boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k)}. \tag{13}$$

Note that this is a "soft" assignment where each data point is assigned to a given mixture component with a certain probability. Once the responsibilities are available the model parameters are updated ("M" or "maximization'" step). The quantities

$$N_k = \sum_{n=1}^{N} r_{nk} \tag{14}$$

$$\bar{\mathbf{x}}_k = \sum_{n=1}^{N} r_{nk} \mathbf{x}_n \tag{15}$$

$$\mathrm{S}_k = \sum_{n=1}^{N} r_{nk} (\mathbf{x}_n - \bar{\mathbf{x}}_k)(\mathbf{x}_n - \bar{\mathbf{x}}_k)^{\mathrm{T}} \tag{16}$$

are first calculated. Finally the model parameters are updated as

$$\pi_k^{\text{new}} = \frac{N_k}{N} \tag{17}$$

$$\boldsymbol{\mu}_k^{\text{new}} = \frac{\bar{\mathbf{x}}_k}{N_k} \tag{18}$$

$$\boldsymbol{\Sigma}_k^{\text{new}} = \frac{S_k}{N_k}. \tag{19}$$

The EM algorithm in general only converges to a local optimum of the log likelihood function. Thus, the choice of initial parameters is crucial. See [Bis07] for the derivation of the equations.

GMM-EM is available in `scikits.learn.em`.

Example usage:

```
from numpy import loadtxt
from scikits.learn.em import GM, GMM, EM

# Data dimensionality
D = 8

# Number of Gaussian Mixture Components
K = 16

# Initialize Gaussian Mixture Model
gmm = GMM(GM(D,K))

# X is the feature data matrix

# Learn GMM
EM().train(X,gmm)
```

Figure 5 shows a visual speech GMM learned using EM. For illustrative purposes only the first two speech-related shape parameters $p_3$ and $p_4$ are used. The shape parameters are obtained by fitting an AAM to each frame of a video of a speaker saying the word "zero". The crosses represent the training data, the circles are the means of the Gaussians and the ellipses are the standard deviation contours (scaled by the inverse of the weight of the corresponding mixture component for visualization purposes). The video frame rate is 30 frames per second (FPS) and the number of mixture components used is 16.

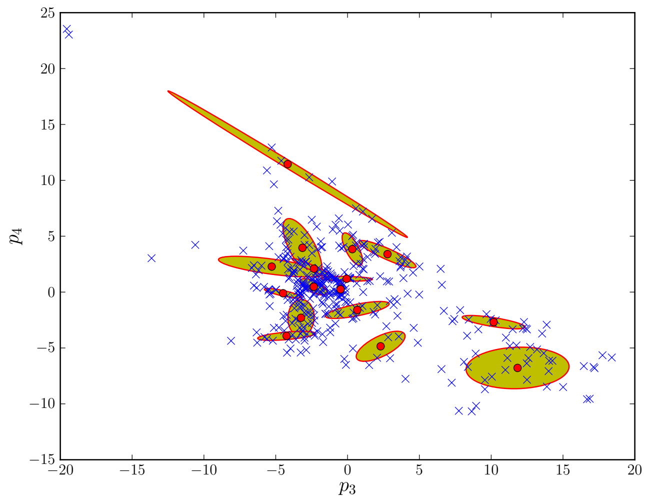

*Fig. 5: Visual speech GMM of the word "zero" learned using EM algorithm on two-dimensional feature vectors.*

Note that in practice more than two shape parameters are used, which usually also requires an increase in the number of mixture components necessary to sufficiently capture the distribution of the data.

### Variational Bayes

An important question that we have not yet answered is how to choose the number of mixture components. Too many components lead to redundancy in the number of computations, while too few may not be sufficient to represent the structure of the data. Additionally, too many components easily lead to overfitting. Overfitting occurs when the complexity of the model is not in proportion to the amount of available training data. In this case the data is not sufficient for accurately estimating the GMM parameters.

The maximum likelihood criteria is unsuitable to estimate the number of mixture components since it increases monotonically with the number of mixture components. Variational Bayesian (VB) inference is an alternative learning method that is less sensitive than ML-EM to over-fitting and singular solutions while at the same time leads to automatic model complexity selection [Bis07].

As it simplifies calculation we work with the precision matrix $\mathbf{\Lambda} = \mathbf{\Sigma}^{-1}$ instead of the covariance matrix.

VB differs from EM in that the parameters are modeled as random variables. Suitable conjugate distributions are the Dirichlet distribution

$$p(\boldsymbol{\pi}) = C(\boldsymbol{\alpha}_0) \prod_{k=1}^{K} \pi_k^{\alpha_0 - 1} \tag{20}$$

for the mixture component weights, and the Gaussian-Wishart distribution

$$p(\boldsymbol{\mu}, \mathbf{\Lambda}) = \prod_{k=1}^{K} \mathcal{N}(\boldsymbol{\mu}_k | \boldsymbol{m}_0, \beta_0 \Lambda_k) \mathscr{W}(\Lambda_k | \mathbf{W}_0, \boldsymbol{\nu}_0) \tag{21}$$

for the means and precisions of the mixture components.

In the VB framework, learning the GMM is performed by finding the posterior distribution over the model parameters given the observed data. This posterior distribution can be found using VB inference as described in [Bis07].

VB is an iterative algorithm with steps analogous to the EM algorithm. Responsibilities are calculated as

$$r_{nk} = \frac{\rho_{nk}}{\sum_{j=1}^{K} \rho_{nj}}. \tag{22}$$

The quantities $\rho_{nk}$ are given in the log domain by

$$\begin{aligned} \ln \rho_{nk} &= \mathbb{E}[\ln \pi_k] + \frac{1}{2}\mathbb{E}[\ln |\mathbf{\Lambda}|] - \frac{D}{2}\ln 2\pi \\ &\quad - \frac{1}{2}\mathbb{E}_{\boldsymbol{\mu}_k, \mathbf{\Lambda}_k}[(\mathbf{x}_n - \boldsymbol{\mu}_k)^{\mathrm{T}} \mathbf{\Lambda}_k (\mathbf{x}_n - \boldsymbol{\mu}_k)] \end{aligned} \tag{23}$$

where

$$\begin{aligned} \mathbb{E}_{\boldsymbol{\mu}, \mathbf{\Lambda}}[(\mathbf{x}_n - \boldsymbol{\mu}_k)^{\mathrm{T}} \mathbf{\Lambda}_k (\mathbf{x}_n - \boldsymbol{\mu}_k)] &= D\beta_k^{-1} \\ + \nu_k (\mathbf{x}_n - \mathbf{m}_k)^{\mathrm{T}} \mathbf{W}_k (\mathbf{x}_n - \mathbf{m}_k) & \end{aligned} \tag{24}$$

and

$$\ln \widetilde{\pi}_k = \mathbb{E}[\ln \pi_k] = \psi(\alpha_k) - \psi(\widehat{\alpha}_k) \tag{25}$$

$$\begin{aligned} \ln \widetilde{\Lambda}_k &= \mathbb{E}[\ln |\mathbf{\Lambda}_k|] = \sum_{i=1}^{D} \psi\left(\frac{\nu_k + 1 - i}{2}\right) \\ &\quad + D \ln 2 + \ln |\mathbf{W}_k|. \end{aligned} \tag{26}$$

Here $\widehat{\alpha} = \sum_k \alpha_k$ and $\psi$ is the derivative of the logarithm of the gamma function, also called the digamma function. The digamma function is available in SciPy as `scipy.special.psi`.

The analogous M-step is performed using a set of equations similar to those found in EM. First the quantities

$$N_k = \sum_n r_{nk} \tag{27}$$

$$\bar{\mathbf{x}}_k = \frac{1}{N_k} \sum_n r_{nk} \mathbf{x}_n \tag{28}$$

$$\mathbf{S}_k = \frac{1}{N_k} \sum_n r_{nk} (\mathbf{x}_n - \bar{\mathbf{x}}_k)(\mathbf{x}_n - \bar{\mathbf{x}}_k)^{\mathrm{T}} \tag{29}$$

are calculated. The posterior model parameters are then updated as

$$\alpha_k^{\text{new}} = \alpha_0 + N_k \tag{30}$$

$$\beta_k^{\text{new}} = \beta_0 + N_k \tag{31}$$

$$\mathbf{m}_k^{\text{new}} = \frac{1}{\beta_k}(\beta_0 \mathbf{m}_0 + N_k \bar{\mathbf{x}}_k) \tag{32}$$

$$\begin{aligned} \mathbf{W}_k^{\text{new}} &= \mathbf{W}_0 + N_k \mathbf{S}_k + \\ &\quad \frac{\beta_0 N_k}{\beta_0 + N_k} (\bar{\mathbf{x}} - \mathbf{m}_0)(\bar{\mathbf{x}} - \mathbf{m}_0)^{\mathrm{T}} \end{aligned} \tag{33}$$

$$\nu_k^{\text{new}} = \nu_0 + N_k. \tag{34}$$

Figure 6 shows a GMM learned using VB on the same data as in Figure 5. The initial number of components is again 16. Compared to Figure 5 we observe that VB results in a much sparser model while still capturing the structure of the data. In fact, the redundant components have all converged to their prior distributions and have been assigned the weight of 0 indicating that these components do not contribute towards "explaining" the data and can be pruned from the model. We

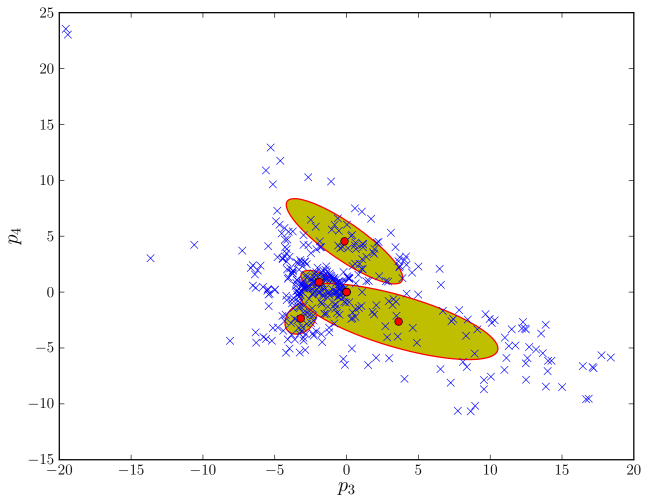

***Fig. 6:*** *Visual speech GMM of the word "zero" learned using the VB algorithm on two-dimensional feature vectors.*

also observe that outliers in the data (which is likely to be noise) is to a large extent ignored.

We have recently developed a Python VB class for `scikits.learn`. The class conforms to a similar interface as the EM class and will soon be available in the development version of `scikits.learn`.

## Experimental results

A basic AVASR system was implemented using SciPy as outlined in the previous sections.

In order to test the system we use the CUAVE database [Pat02]. The CUAVE database consists of 36 speakers, 19 male and 17 female, uttering isolated and continuous digits. Video of the speakers is recorded in frontal, profile and while moving. We only use the portion of the database where the speakers are stationary and facing the camera while uttering isolated digits. We use data from 24 speakers for training and the remaining 12 for testing. Hence, data from the speakers in the test data are not used for training. This allows us to evaluate how well the models generalize to speakers other than than those used for training. A sample frame from each speaker in the dataset is shown in Figure 7.

In the experiment we build an individual AAM for each speaker by manually annotating every 50th frame. The visual features are then extracted by fitting the AAM to each frame in the video of the speaker.

Training the speech recognition system consists of learning acoustic and visual GMMs for each digit using samples from the training data. Learning is performed using VB inference. Testing is performed by classifying the test data. To evaluate the performance of the system we use the misclassification rate, i.e. the number of wrongly classified samples divided by the total number of samples.

We train acoustic and visual GMMs separately for each digit. The probability distributions (see (11)) are denoted by $p(\mathbf{x}_A)$ and $p(\mathbf{x}_V)$ for the acoustic and visual components,

***Fig. 7:*** *Frames from the CUAVE audio-visual data corpus.*

respectively. The probability of a sample $(\mathbf{x}_A, \mathbf{x}_V)$ belonging to digit class $c$ is then given by $p_A(\mathbf{x}_A|c)$ and $p_V(\mathbf{x}_V|c)$.

As we wish to test the effect of noise in the audio channel, acoustic noise ranging from -5dB to 25dB signal-to-noise ratio (SNR) in steps of 5 dB is added to the test data. We use additive white Gaussian noise with zero mean and variance

$$\sigma_\eta^2 = 10^{\frac{-\mathrm{SNR}}{10}}. \tag{35}$$

The acoustic and visual GMMs are combined into a single classifier by exponentially weighting each GMM in proportion to an estimate of the information content in each stream. As the result no longer represent probabilities we use the term *score*. For a given digit we get the combined audio-visual model

$$\mathrm{Score}(\mathbf{x}_{AV}|c) = p(\mathbf{x}_A|c)^{\lambda_A} p(\mathbf{x}_V|c)^{\lambda_V} \tag{36}$$

where

$$0 \le \lambda_A \le 1 \tag{37}$$
$$0 \le \lambda_V \le 1 \tag{38}$$

and

$$\lambda_A + \lambda_V = 1. \tag{39}$$

Note that (36) is equivalent to a linear combination of log likelihoods.

The stream exponents cannot be determined through a maximum likelihood estimation, as this will always result in a solution with the modality having the largest probability being assigned a weight of 1 and the other 0. Instead, we discriminatively estimate the stream exponents. As the number of classes in our experiment is relatively small we perform this optimization using a brute-force grid search, directly minimizing the misclassification rate. Due to the constraint (39) it is only necessary to vary $\lambda_A$ from 0 to 1. The corresponding $\lambda_V$ will then be given by $1-\lambda_A$. We vary $\lambda_A$ from 0 to 1 in steps of 0.1. The set of parameters $\lambda_A$ and $\lambda_V$ that results in the lowest misclassification rate are chosen as optimum parameters.

In the experiment we perform classification for each of the SNR levels using (36) and calculate the average misclassification rate. We compare audio-only, visual-only, and

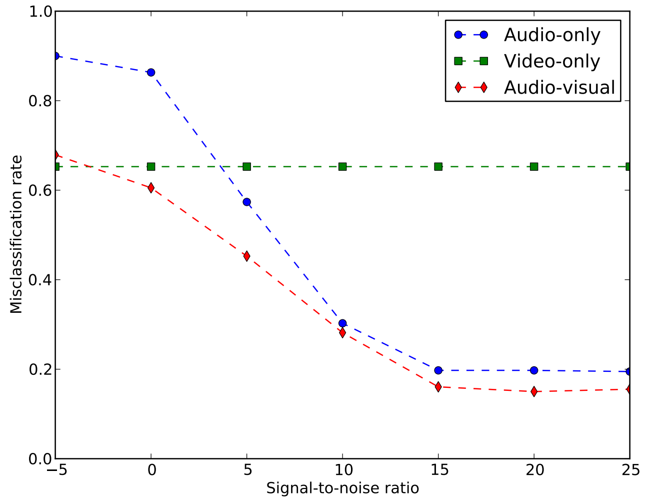

***Fig. 8:*** *Misclassification rate.*

audio-visual classifiers. For the audio-only classifier the stream weights are $\lambda_A = 1$ and $\lambda_V = 0$ and for visual-only $\lambda_A = 0$ and $\lambda_V = 1$. For the audio-visual classifier the discriminatively trained stream weights are used. Figure 8 shows average misclassification rate for the different models and noise levels.

From the results we observe that the visual channel does contain information relevant to speech, but that visual speech is not in itself sufficient for speech recognition. However, by combining acoustic and visual speech we are able to increase recognition performance above that of audio-only speech recognition, especially the presence of acoustic noise.

## Conclusion

In this paper we propose a basic AVASR system that uses MFCCs as acoustic features, AAM parameters as visual features, and GMMs for modeling the distribution of audio-visual speech feature data. We present the EM and VB algorithms as two alternatives for learning the audio-visual speech GMMs and demonstrate how VB is less affected than EM by overfitting while leading to automatic model complexity selection.

The AVASR system is implemented in Python using SciPy and tested using the CUAVE database. Based on the results we conclude that the visual channel does contain relevant speech information, but is not in itself sufficient for speech recognition. However, by combining features of visual speech with audio features, we find that AVASR gives better performance than audio-only speech recognition, especially in noisy environments.

## Acknowledgments

The authors wish to thank MIH Holdings for funding the research presented in this paper and for granting permission to contribute the research source code to SciPy.

## References

[Dav80] S. Davis, I. Matthews. *Comparison of parametric representations for monosyllabic word recognition in continuously spoken sentences*, IEEE Transactions on Acoustics, Speech, and Signal Processing, 28(8),357-366, 1980

[Luc81] B.D. Lucas, T. Kanade. *An iterative image registration technique with an application to stereo vision*, Proceedings of Imaging understanding workshop, 121-130, 1981

[Coo98] T.F. Cootes, G.J. Edwards, C. J .Taylor, *Active appearance models*, Proceedings of the European Conference on Computer Vision, 1998

[Bak01] S. Baker and I. Matthews, *Lucas Kanade 20 Years On: A Unifying Framework*, International Journal of Computer Vision, 2000

[Pat02] E.K. Patterson, S. Gurbuz, Z. Tufekci, J.N. Gowdy, *CUAVE: A new audio-visual database for multimodeal human-compuer inferface research*, 2002

[Mat03] I. Matthews, S. Baker, *Active Appearance Models Revisited*, International Journal of Computer Vision, 2003

[Bis07] C.M. Bishop. *Pattern recognition and machine learning*, Springer, 2007