Binary-Feature Detection Cascades for Speech Recognition

Mark Stoehr

May 28, 2013

Contents

1 Introduction 2

2 Notation 4

3 Signal Processing 4
   3.1 Theoretical Background 4
   3.2 Spectrogram Computation 7
   3.3 Multitaper Signal Processing 8
   3.4 Visualizations 10
   3.5 Smoothed Log Spectrograms 10
   3.6 Cepstral-Smoothed Mel Spectrograms 13
   3.7 Visualization 15

4 Edge Models 17
   4.1 Edge Computation Algorithm 19
   4.2 Visualization 24

5 Statistical Models of Binary Data 35
   5.1 Bernoulli Models 35
      5.1.1 The Background Model 36
      5.1.2 Mixture Models 36
   5.2 Estimation via Maximum Likelihood 37
   5.3 Estimation via EM 38
   5.4 Part Models 41
   5.5 Part Transform 43
   5.6 Visualization of the Part Transform 44
   5.7 Sound Models 49
      5.7.1 Length Clustering in the Template Models 50

6 Detection 52
   6.1 Cluster-Based Detection 53
      6.1.1 Detection Curve Computation 55
      6.1.2 Multi-Object Configurations 57
      6.1.3 Detection Candidates Computation 58
1 Introduction

HMMs remain the dominant speech recognition technology, however there has been increasing interest in non-HMM approaches to speech recognition. HMMs perform poorly at the task of modeling the trajectory of acoustic features over a syllable [52], and, in response, a class of template-based approaches has emerged [2, 52, 10, 48, 22, 5]. Template-based speech recognition matches reference templates of phones, syllables, or whole words directly against the speech signal. A detection is declared if a template closely matches a segment of speech. Templates may take arbitrary form so they can model a wide variety of patterns in speech. Some of the earliest speech recognizers [41, 50] were template-based although the approach fell out of favor compared to the HMM because of the memory-requirements for storing all of the templates. With recent advances in computing hardware, they are becoming an attractive alternative to the HMM [40]. They are also of interest for their relationship to Exemplar Theory from phonetics and phonology [23, 25] which postulates that people perceive speech using short prototype speech units.

Nearly all template-matching-based systems make use of dynamic time warping in order to perform detection because the template could have a different length than the speech segment we wish to compare it with. A notable exception to this trend is [2] which uses a single template to detect acoustic units— they use binary features that are robust to local time shifts, so precise matching
is not required. Perhaps the only others systems not to use warping for template-matching are the oldest known recognizers [8] and Radio Rex [19]. Systems that do not use time warping necessarily have have limited ability to model the time variability of speech units.

This work, like [2], does not warp templates to match query speech segments, but we make use a data-driven training procedure that estimates multiple templates that can have different lengths. Thus, we explicitly account for speaking-rate variation without introducing explicit time-warping. We are hesitant to warp prototypes or example data because warping could potentially interfere with precise timing cues such as the voice onset time. The importance of timing cues can be seen from the fact that speech recognition can be done quite successfully by humans using primarily timing cues with fine spectral detail removed [44]. We hope this work can shed some light on what is gained or lost through warping.

Another line of work we build upon uses features computed from localized filters over the time-frequency plane. These features have been used in [2, 21, 37, 26, 29, 39, 12], and they draw their motivation, in part, from observations about the human auditory system [14, 16], and the observation that many hypothesized cues for speech recognition (e.g., formants and onsets) are localized in the time-frequency plane [27]. [2, 39, 12] use binary difference filters and [2, 59] construct binary outputs from those filters. The filters used in [2] are fixed edge-filters, whereas [39] uses boosted features in the style of [51]. The basic features employed in this paper are based on those in [2], although we also explore the use of learned features introduced in [6], where they are used in the context of computer vision. One advantage in using localized binary features is that we can potentially test a variety of different signal processing approaches and auditory models, as all such models support localized feature detectors whereas HMM-based approaches rely on the signal processing model conforming to specific statistical assumptions [19]. Although we only explore a small number of time-frequency representations in this work the methods presented could be applied to most auditory models.

The third component of this work is cascaded detection. Although cascades are commonly used in computer vision [1, 15, 51, 17] (and the references contained therein), with the exception of [54], there have been few attempts at applying cascades to speech problems. In this system we work locally on the signal and perform detection by a three-step cascade: a likelihood-ratio test that identifies a set of potential matches, a clustering algorithm to reduce this set of matches, then an SVM classifier that discards low-quality matches. Training the likelihood ratio detector involves estimating a mixture model over segmented examples of the target sequence, while training the SVM detector requires speech that is known to not contain the target sequence but is otherwise unlabeled. We find that cascades help detection performance considerably, which differs from the rationale for cascades in computer vision, where they assist with efficiency.

The fourth component of this work is a comparison of signal processing features. We investigate the multitaper spectrogram and the effects of different smoothing algorithms. Multitaper spectral processing is common in the geophysical literature but is not often used in speech recognition [43] and [35], and we obtained poorer results with a standard Hamming-taper spectrogram and mel-spectral features [44] compared with those achieved using a multitaper spectrogram [49, 31]. We also find that smoothing the spectrogram in the frequency domain as in [34, 53, 24] hurts performance.

In summary, the main contributions of this work are investigating these four topics:

1. Learning and matching templates without warping
2. Learning binary features
3. Detecting with cascades
4. Multitapering the spectral estimate

2 Notation

We will be working with one-dimensional, two-dimensional, and three-dimensional arrays. Indexing works as follows: $x(t)$, for instance, will refer to the $t$th entry of the vector $x$, it could also mean $x$ viewed as a function being evaluated at time $t$. In the case of a two-dimensional function $S(t,f)$, since we will usually consider only discrete inputs, we can also view the function as a matrix, where the rows correspond to the discrete set of potential values for $t$ and the columns correspond to the discrete set of potential values for $f$.

Sometimes we will need to refer to a column or row vector in such a matrix, for which we would write $S(:,f)$ or $S(t,:)$, respectively. In these two cases we have used : to denote all entries along a particular axis of the matrix. In order to refer to all columns $k$ of $S$ such that $i \leq k < j$ we would write $S(:,i:j)$ noting that we include column $i$ and but not column $j$. This asymmetrical formulation is convenient because half-intervals easily form partitions.

$[n]$ denotes the set $\{0, \ldots, n-1\}$. $\mathbf{1}$ is a vector of all ones.

3 Signal Processing

In this paper we consider four different signal-processing algorithms and assess their impact on the detection of syllables in speech. Each of these four algorithms consists of two stages:

1. Estimate the time-varying power-spectral density of the input speech signal
2. Convert the power-spectral density estimates into decibels (log-domain) and smooth

For the power spectral density problem our algorithms are:

1. Spectrogram
2. MultitaperSpectrogram

The algorithms we use to solve the second problem, computing the smoothed log domain (possibly frequency-warped) spectral density, are:

1. GaussianSmoothing
2. MelCepstralSmoothing.

Each signal processing front-end considered in this paper is built from these four algorithms. We present some theoretical ideas for analyzing their behavior, but we ultimately evaluate them by the performance on recognition tasks. We will find that some of the theoretical results will give insight into the performance of these algorithms on speech-recognition tasks. In particular, our recognition algorithms perform much better with multitaper spectrograms than with Hamming spectrograms perhaps, in part, because of the lower variance of the spectral estimates. The proof of these properties requires some theoretical machinery.

3.1 Theoretical Background

Let $x(t)$ be a random, real-valued signal defined for $t = 0, \ldots, T_x - 1$. A common model for speech signals is that $x(t)$ is formed from digital samples of a stationary process $W$ convolved with a time-varying linear filter $y_t(u)$.
\[ x(t) = \int_{-\infty}^{\infty} W(t-u)y(t) \, du. \]  

(1)

Equation (1) is commonly called the *source-filter model* [13]. Priestley, in [33], shows the source-filter model is very general and can describe any stochastic process (not necessarily stationary) with a finite covariance function that admits an eigendecomposition. Additionally, a random digital signal \( x(t) \) satisfying Equation (1) may be expressed as

\[ x(t) = \int_{-\pi}^{\pi} e^{it\omega} A_t(\omega) \, dZ(\omega) \]  

(2)

where \( Z(\omega) \) is an orthogonal process on \((-\pi, \pi)\) and for each frequency \( \omega \) the sequence \( \{A_t(\omega)\}_t \) has a Fourier transform (holding \( \omega \) fixed and taking the transform with respect to \( t \)) whose modulus has an absolute maximum at zero so that \( A_t(\omega) \) is smooth and acts as the envelope of a carrier signal \( e^{it\omega} dZ(\omega) \). In this formulation \( A_t(u) \) is the Fourier transform of \( y(t) \) where \( t \) is held fixed.

Since \( y(t) \) varies (slowly) with \( t \), Priestly defines the *evolutionary spectrum* of \( x(t) \) as

\[ dR_t(\omega) = |A_t(\omega)|^2 E(dZ(\omega))^2 \]  

(3)

and we note that since \( A_t(\omega) \) has a Fourier transform with an absolute maximum at zero the spectrum is generally slowly-varying in time. Assuming that \( E[|dZ(\omega)|^2] \) is absolutely continuous with respect to Lebesgue measure there is an evolutionary spectral density function

\[ r_t(\omega)d\omega = dR_t(\omega) \]

which is related to \( A_t(\omega) \) by Equation (3).

We may estimate the evolutionary spectrum with a spectrogram. Let

\[ U(t,f) = \sum_{u \in \mathcal{N}(t)} h(u) x(t-u) e^{-if(t-u)} \]  

(4)

where \( \mathcal{N}(t) \) is an interval over \( t \) so that \( x(t-u) \) is defined, \( h(u) \) is called the data taper, and the square magnitude of the left-hand side \( |U(t,f)|^2 \), which is denoted \( S(t,f) \), is the spectrogram. Priestley shows that the spectrogram is an inconsistent estimator of the evolutionary spectral density function because it has high variance. Let \( G(\omega,\theta) \) be the Fourier transform of the sequence \( \{|A_t(\omega)|^2\}_{t=0}^{T_x} \) so that \( \omega \) is held fixed. The stationary width \( B_x \) of the signal \( x(t) \) is defined as

\[ B_x = \left[ \sup_{\omega} \int_{-\infty}^{\infty} |\theta||dG(\omega,\theta)| \right]^{-1}, \]  

(5)

which, informally, corresponds to the longest window over which \( x(t) \) is approximately stationary (note that in a stationary process \( A_t(\omega) \) is constant so the stationary width is infinite). Define the bandwidth of the data taper \( B_h \) as

\[ B_h = \sum_{t=-\infty}^{\infty} |t||h(t)| \, dt \]  

(6)
where the sum is actually taken over the support of \( h(t) \). Priestley shows that

\[
\mathbb{E}S(t, f) = \int_{-\infty}^{\infty} |H(\omega)|^2 r_t(\omega + f) \, d\omega + O(B_h/B_x)
\]

where \( H(\omega) \) is the Fourier transform of \( h(t) \). If \( x(t) \) is Gaussian then Priestley shows

\[
\text{Var}[S(t, f)] \approx \left\{ \int_{-\infty}^{\infty} |H(\omega)|^4 \, d\omega \right\} \left\{ \int_{-\infty}^{\infty} \mathbf{1}_{H(\omega) \neq 0} |r_t(\omega + f)|^2 \, d\omega \right\}.
\]

then by the Cauchy-Schwarz inequality we have:

\[
\text{Var}[S(t, f)] \leq \left\{ \int_{-\infty}^{\infty} |H(\omega)|^4 \, d\omega \right\} \left\{ \int_{-\infty}^{\infty} |r_t(\omega + f)|^2 \, d\omega \right\}.
\]

So, the variance of \( S(t, f) \) is approximately proportional to a local of average of the square of its expectation \( \int_{-\infty}^{\infty} \mathbf{1}_{H(\omega) \neq 0} |r_t(\omega + f)|^2 \, d\omega \), which means that \( S(t, f) \) will not be a very useful estimator in practice, which we find to be the case in our experiments. The upper bound on the variance of \( S(t, f) \) is also proportional to \( \|H\|^4 \) and we will use this fact to explain some of our experimental results.

Priestley suggests an estimator with better variance properties:

\[
S_{\text{smoothed}}(t, f) = \sum_{u=\infty}^{\infty} w(u) S(t - u, f)
\]

where the limits of the sum are actually taken over the range of values where \( w(u)S(t - u, f) \) is well-defined. The estimator in Equation 10 is consistent if \( w \) has bandwidth \( B_w \) (Equation 6) much larger than \( B_h \), the bandwidth of \( h \). A large-bandwidth smoother means we will lose information about rapid spectral changes, which makes the estimator defined by Equation 10 inappropriate for speech since rapid spectral change carries important phonetic information [46]. Thus, in analogy with the stationary spectrum estimation problem, we consider two other approaches. One method is to smooth over frequency bands instead of time:

\[
S_{\text{smoothed}}(t, f) = \sum_{\omega} w_f(\omega) S(t, f - \omega),
\]

which is related to the mel spectral estimators commonly used in speech recognition [34] and to spectral density estimation techniques for stationary processes [7]. Due to the orthogonality of the Fourier basis, \( (S(t, f_1), S(t, f_2), \ldots, S(t, f)) \) are approximately uncorrelated if \( x(t) \) is a Gaussian process, so by taking an average we can achieve a lower variance estimator. The other method we consider is the multitaper spectral estimator \( S_{\text{multitaper}} \), which averages over several different spectrograms \( \{S_k\}_{k=0}^{K-1} \) each one computed with a different data taper \( \{h_k\}_{k=0}^{K-1} \).

Formally, the multitaper estimator is

\[
S_{\text{multitaper}}(t, f) = \frac{1}{K} \sum_{k=0}^{K-1} S_k(t, f).
\]

The multitaper estimator has controlled variance since the tapers \( \{h_k\}_{k=0}^{K-1} \) are chosen to be orthogonal so each of the \( K \) spectrograms, \( S_k(t, f) \), are approximately uncorrelated hence averaging
over the different estimates reduces the variance as $K$ increases. In [31], it is shown that in the stationary case $S^{\text{multitaper}}(t, f)$ approximates

$$\int_{f-W}^{f+W} |A(\omega)|^2 \, d\omega,$$

(13)

where $W$ is a bandwidth parameter and the subscript $t$ in $A_t(\omega)$ is dropped to emphasize that we are dealing with a stationary process. We note that the variance of $S(t, f)$ in all these cases is always proportional to $h_t(f)^2$ (in the stationary case the density $h_t(\omega)$ is the same for all $t$), and the expectation is always proportional to $h_t(\omega)$ so it is natural to apply the logarithmic transform since if the variance is sufficiently small we can use the Taylor expansion of the logarithm to show:

$$\text{Var}[\log S(t, f)] \approx \frac{\text{Var}[S(t, f)]}{\mathbb{E}^2[S(t, f)]}$$

which will mean that the variance of $\log S(t, f)$ is independent of the magnitude of $S(t, f)$ [33]. Wahba suggests directly smoothing the log-spectral estimate [53], and Riedel shows that smoothing the multitaper estimate in the log-domain decreases the variance further [36].

In summary, we have evidence for the following:
1. Pick data tapers $h(t)$ with smaller bandwidths to get a less-biased estimate of the envelope $|A(t)|^2$
2. Pick data tapers with Fourier transform $H(t)$ that have small

$$\|H\|_4^4 = \int_{-\infty}^{\infty} |H(f)|^4 \, df$$

3. Work with $\log S(t, f)$
4. To control the variance of the log-spectrum potentially work with averages over frequency:

$$\log S^{\text{smoothed}}(t, f) = \log \sum_\omega w_f(\omega) S(t, f - \omega)$$

5. Another way to control the variance of the log-spectrum is to work with a multitaper average

$$\log S^{\text{multitaper}}(t, f) = \log \frac{1}{K} \sum_k S_k(t, f)$$

### 3.2 Spectrogram Computation

We now consider practical algorithms to compute the quantities mentioned in the previous section. Spectrogram (Algorithm 1) computes a simple single-taper spectrogram estimate is $S \in \mathbb{R}^{T \times F}$ where $T$ is the number of time points and $F$ is the number of frequency channels for which we compute the spectrogram estimate based on Equation 4. The first step of Spectrogram is line 2 where we apply a preemphasis filter that emphasizes the higher-frequency portion of the spectrogram:

$$x^{\text{preemphasis}}(t) = x(t) - \alpha x(t - 1).$$

(14)

Standard choices for $\alpha$ are $[0.4, 1.0]$ [32] and we use $\alpha = 0.94$ in all of our experiments. Preemphasis is not essential for spectrogram computation but we find that it is essential for good results. Detection
of the spectral dynamics in the high-frequency section of the spectrogram degrades significantly without preemphasis. One reason why this might happen is that the evolutionary transfer function for \( y_i(t) \) which represents the action of the vocal tract on the glottal source \( W(t) \), which is modeled by \( e^{ihf}dZ(f) \).

We assume that the data taper \( h(t) \) used by \texttt{SPECTROGRAM} has finite support, \( T_w \), and we only compute a spectral estimate once every \( T_s \) samples of the signal \( x(t) \). The samples for which we compute the estimate are \( 0, T_s, 2T_s, \ldots, \tau T_s, \ldots \) so that

\[
S(t, f) = \sum_{u=0}^{T_w-1} h(u)x(tT_s + u)e^{-iftu}, \tag{15}
\]

which can be shown to be a special case of Equation 4. We compute each spectral estimate \( S(t, f) \) from a segment of the signal \( x(t) \): \( x(tT_s: (t+1)T_s) = (x(tT_s), x(tT_s + 1), \ldots, x((t+1)T_s - 1)) \). We refer to the segment \( x(tT_s: (t+1)T_s) \) as a “frame” throughout the rest of this work. Line 3 of \texttt{SPECTROGRAM} computes the number of frames used for computing the spectrogram. Line 5 explicitly breaks \( x(t) \) up into the overlapping spectral analysis frames. We weight the entries of the frame by the data taper \( h(t) \). In line 6 we compute the Fourier transform of each frame, but we pad the frames with zeros first in order to make the computation more efficient. This padding introduces redundancy into our spectral estimates since we have more Fourier coefficients than we have data points.

One of the inputs to the algorithm is a frequency cutoff quantity \( F_{\text{max}} \) that we pick to be \( 3kHz \) in our experiments. The Fourier transform performed in line 6 decomposes the signal into constituent frequency components. For every \( \tau \) and \( j \), entry \( \hat{V}_{\tau,j} \) of the vector \( \hat{V}_\tau \) (line 6) corresponds to the sinusoid \( e^{2\pi ij\tau\text{fft}} \) which has \( j \) cycles per \( T_{\text{fft}} \) samples so its frequency is \( jS_r/T_{\text{fft}}Hz \). We only consider coefficients whose associated frequency in \( Hz \) is less than \( F_{\text{max}} \). In line 7 we find \( j_{\text{max}} \) such that for any \( j \leq j_{\text{max}} \) we have \( \hat{V}_{\tau,j} \) is the coefficient for a sinusoid with frequency less than \( F_{\text{max}} \) so that only the low-frequency terms are retained. Line 10 of the algorithm takes the magnitude of the Fourier coefficients and constructs the array:

\[
S = \begin{bmatrix}
|\hat{V}_{0,0}| & |\hat{V}_{1,0}| & \cdots & \hat{V}_{T_v-1,0} \\
\hat{V}_{0,1} & \hat{V}_{1,1} & \cdots & \hat{V}_{T_v-1,1} \\
\vdots & \vdots & \ddots & \vdots \\
\hat{V}_{0,j_{\text{max}}-1} & \hat{V}_{1,j_{\text{max}}-1} & \cdots & \hat{V}_{T_v-1,j_{\text{max}}-1}
\end{bmatrix}.
\]

\( S \) is a function of time-frequency locations \((t, f) \in \{0, \ldots, T_v - 1\} \times \{0, \ldots, j_{\text{max}} - 1\} \) and is an estimate of the power-spectral density of the signal \( x(t) \).

### 3.3 Multitaper Signal Processing

Recall that for the multitaper spectrogram we compute \( K \) spectrograms \( \{S_k\}_{k=0}^{K-1} \) each with its own data taper \( \{h_k\}_{k=0}^{K-1} \) over the same signal \( x(t) \). In this work we use the Slepian sequences \([45, 31]\) for data tapers although there are many choices available. The Slepian sequences \( \{h_k\}_{k=0}^{K-1} \) are orthogonal so we achieve the desired reduction of variance in multitaper spectral estimates over naive spectral estimates. Our implementation \texttt{MULTITAPER\_SPECTROGRAM} is presented in Algorithm 2.
Input: $x(t)$ signal, $T_x \in \mathbb{Z}_{>0}$ signal length, $S_r \in \mathbb{Z}_{>0}$ sample rate, $\alpha \in \mathbb{R}$ preemphasis weight, $T_w \in \mathbb{Z}_{>0}$ frame length, $T_s \in \mathbb{Z}_{>0}$ frame hop length, $T_{fft} \in \mathbb{Z}_{>0}$ fft length, $F \in \mathbb{R}^j \rightarrow \mathbb{C}^j$ Fourier transform operator, $h(t) \in [0, 1] \rightarrow \mathbb{R}$ data taper, $F_{\text{max}} \in \mathbb{R}_{>0}$ cutoff frequency,

Output: $S(f, t)$ spectrogram

1: procedure SPECTROGRAM($x(t), T_x, S_r, \alpha, T_w, T_s, T_{fft}, h(t), F_{\text{max}}$)  \Comment{Compute a single-taper spectrogram over $x$}  
2: $x(t) \leftarrow x(t) - \alpha x(t - 1)$  \Comment{Preemphasis filter}  
3: $T_v \leftarrow \frac{T_x T_w}{T_s}$  \Comment{Get number of frames in spectrogram}  
4: for $\tau = 0 \ldots T_v$ do  
5: $v_\tau \leftarrow \begin{bmatrix} x(\tau T_s) h(0) \\ x(\tau T_s + 1) h(1/(T_W - 1)) \\ \vdots \\ x(\tau T_s + T_w - 1) h(1) \end{bmatrix}$  \Comment{Taper and window the signal}  
6: $\hat{V}_\tau \leftarrow F \left\{ v_\tau \right\} T_w$ frame entries \Comment{Pad with zeros, compute the Fourier transform}  
7: $f_{\text{max}} \leftarrow \max\{f = 0, \ldots, T_{fft} - 1 \mid \frac{f}{T_{fft}} S_r \leq F_{\text{max}}\}$  \Comment{Find cutoff for Fourier coefficients}  
8: $\hat{V}_\tau \leftarrow \begin{bmatrix} \hat{V}_{\tau,0} \\ \hat{V}_{\tau,1} \\ \vdots \\ \hat{V}_{\tau,f_{\text{max}}-1} \end{bmatrix}$  \Comment{Retain only low-frequency coefficients below cutoff}  
9: for $f = 0, \ldots, f_{\text{max}} - 1$ do  
10: $S(\tau, f) \leftarrow |\hat{V}_{\tau,f}|^2$  \Comment{Store entries in 2D spectrogram image}  
11: end for  
12: end for  
13: return $S$  
14: end procedure

Algorithm 1: Compute Spectrogram Algorithm
Input: \( x(t) \in \mathbb{Z}_+ \rightarrow \mathbb{R} \) signal, \( T_x \in \mathbb{Z}_{>0} \) signal length, \( S_r \in \mathbb{Z}_{>0} \) sample rate, \( \alpha \in \mathbb{R} \) preemphasis weight, \( \{h_k(t)\}_{k=1}^{K} \) orthogonal data tapers, \( T_w \in \mathbb{Z}_{>0} \) frame length, \( T_s \in \mathbb{Z}_{>0} \) frame hop length, \( T_{fft} \in \mathbb{Z}_{>0} \) fft length, \( \mathcal{F} \) Fourier transform operator, \( F_{\text{max}} \in \mathbb{Z}_{>0} \) cutoff frequency.

Output: \( S(f,t) \) multitaper spectrogram

1: \textbf{procedure} \textsc{MultitaperSpectrogram} \((x(t), T_x, S_r, \alpha, \{h_k(t)\}_{k=1}^{K-1}, T_w, T_s, T_{fft}, F_{\text{max}}, G, \mathcal{J})\)
    \(\triangleright\) Compute a multitaper spectrogram over \( x \)
2: \( S \leftarrow \sum_{k=0}^{K-1} \text{Spectrogram}(x(t), T_x, S_r, \alpha, T_w, T_s, T_{fft}, h_k(t), F_{\text{max}}) \)
3: \textbf{return} \( S \)
4: \textbf{end procedure}

Algorithm 2: Multi-Taper Spectrogram (MuT)

3.4 Visualizations

In the \textsc{Spectrogram} algorithm the data taper is the Hamming window as shown in Figure 1. Note how the values at the tails taper to zero and most of the weight of the curve is concentrated around the origin. In our implementation of \textsc{MultitaperSpectrogram} we use the first five Slepian sequences [31] as our data tapers: they are pictured in Figure 2.

We compare these spectral estimation methods on the problem of estimating the spectrum over a signal frame taken from someone saying /Ar/. The raw signal is shown in Figure 3.

The single taper Hamming based spectrum computed for that frame is shown in Figure 4. Note how there are many little peaks spread throughout the spectrum: those are caused by the glottal pulse and are called the pitch harmonics.

The multitaper spectrum estimate is shown in Figure 5. The smearing property of the spectrum has eliminated many of the pitch harmonics and one can more clearly see the presence of three distinct peaks in the spectrum.

3.5 Smoothed Log Spectrograms

Building off of the theoretical observations, we work in the logarithmic spectral domain as is common in signal processing algorithms for speech [34]. We have two algorithms for smoothing the logarithm of the spectrogram. The algorithm \textsc{GaussianSmoothing}, Algorithm 3 uses a Gaussian filter and \textsc{MelCepstralSmoothing} first mel-filters the spectrogram then applies a cepstrum-based lifting technique. “Lifter” refers to filtering on the log spectral domain, and “cepstrum” refers to the spectrum where the log spectrum is treated as a time-domain signal. In \textsc{GaussianSmoothing} we pick a Gaussian filter \( G \) and a subsampling operator \( \mathcal{J} \). The filter \( G \) will generally be an outer product between a one-dimensional filter in time \( g_{\text{time}} \) and a one-dimensional filter for frequency \( g_{\text{freq}} \) so

\[ G = g_{\text{time}} g_{\text{freq}}^\top. \]
Figure 1: Hamming window time-domain.

Figure 2: Slepian windows time-domain.
Figure 3: Single frame of speech

Figure 4: Single Taper Hamming Spectrum
Algorithm 3: Gaussian Smoothed Log Spectrogram

The subsampling operator $J$ will downsample the spectrogram along the frequency domain axis so that

$$S_{new}(t,f) = S_{old}(t,2f)$$

3.6 Cepstral-Smoothed Mel Spectrograms

MelCepstralSmoothing [Algorithm 4] applies a set of smoothing filters to a spectrogram. We may compute the input spectrogram using a single taper or a multi-taper approach. However, we found that this choice did not have material impact on detector performance most likely because the filtering operation averages the spectrum estimates over a small filter bandwidth and thus the estimate has similar properties to a downsampled multitaper estimate.

Aside from the input spectrogram $S$, MelCepstralSmoothing takes as input: a mel filter-bank $\{\phi_k\}_{k=0}^{K-1}$, the discrete cosine transform matrix $D$, and the cepstral liftering weights $W_C$. The
Input: \( S \in \mathbb{R}^{T \times F} \) spectrogram, \( \{ \phi_k(t) \}_{k=1}^K \) mel filters, \( \mathcal{D} \) discrete cosine transform matrix, \( W \) cepstral liftering coefficients

Output: \( S(f,t) \) smoothed log spectrogram

1: procedure MelCepstralSmoothing(\( S, \{ \phi_k \}_{k=0}^{K-1}, \mathcal{D}, W \)) \( \triangleright \) Compute a mel spectrogram over \( x \)

2: \hspace{1em} for \( t = 0, \ldots, T_v \) do

3: \hspace{2em} for \( k = 0, \ldots, K - 1 \) do

4: \hspace{3em} \( M(t,: \) \( \leftarrow \log \phi_k^\top S(t,:) \quad \triangleright \) Perform the mel-filtering and take the logarithm

5: \hspace{2em} end for

6: \hspace{1em} \( M(t,:) \leftarrow \mathcal{D}^{-1} \begin{bmatrix} W & 0 \\ 0 & 0 \end{bmatrix} \mathcal{D} M(t,:) \)

7: end for

8: return \( M \)

9: end procedure

Algorithm 4: Cepstral Smoothed Log Mel Spectrum Algorithm (CsLMel)

number of mel filters, \( K \), is usually 40 [32, 34] and each filter has the same dimension, \( f_{\text{max}} \), as \( S(t,:) \). A given mel filter \( \phi_k \) has a center frequency \( \omega_k \) and a support width \( l_k \) so that

\[
\phi_k = \begin{bmatrix}
0 \\
0 \\
\vdots \\
\phi_k(\omega_k - l_k/2) \\
\vdots \\
\phi_k(\omega_k + l_k/2) \\
0 \\
\vdots
\end{bmatrix}
\]

\( l_k \) entries.

\( \phi_k \) is an average over a length-\( l_k \) interval of frequencies centered at \( \omega_k \). So, we may also write \( \phi_k \) in terms of a kernel estimate of the frequency:

\[
\phi_k(f) = \psi \left( \frac{2\omega_k - f}{l_k} \right)
\]

and usually the kernel used for mel filtering is the triangular kernel

\[
\psi(f) = \max\{0, 1 - |f|\}.
\]

This means that

\[
\phi_k^\top \circ S(t,:) = \sum_f \psi \left( \frac{2\omega_k - f}{l_k} \right) S(t,f).
\]
Figure 6: MFCC filters $\phi_k$.

The center frequencies $\omega_k$ are spaced such that they put the frequency on a log-scale which more closely mirrors psychophysically-derived pitch scales [47].

We visualize the filters in Figure 6.

In line 6, MelCepstralSmoothing lifts the mel-filtered spectrum estimate. The matrix $D$ is the discrete cosine transform matrix arranged so that the lower-frequency components of the transform are at the top while the higher-frequency components are towards the bottom of the matrix. Hence $DS(t,:)$ will be a vector $(q_{0}, q_{1}, \ldots, q_{f_{\text{max}}})^\top$ where higher index entries correspond to higher quefrency coefficients. $W$ is a diagonal matrix whose dimension $N_w$ corresponds to the number of cepstral coefficients we are projecting the signal onto. We then transform the signal back into the log mel-spectral domain. A smaller $N_w$ dimension corresponds to more smoothing and a larger $N_w$ performs to less smoothing. 13 is commonly used in speech recognition systems [34, 32]. Often the diagonal entries of $W$ are just ones so that it is the identity matrix. Other times the entries will vary (usually being larger than one) and add weight to certain cepstral terms [24].

Given that the pitch harmonics correspond to rapid changes across the mel-spectrum, projecting the mel-spectrum onto its lower quefrency terms tends to remove those effects [34].

3.7 Visualization

Most of the experiments considered in this paper are constructed using three different signal-processing front ends. These main models are:

1. MultitaperSpectrogram with MelCepstralSmoothing
2. MultitaperSpectrogram with GaussianSmoothing where we smooth in time and frequency.
3. MultitaperSpectrogram with GaussianSmoothing where we smooth in time only.

We also ran a smaller number of experiments on:
1. Multitaper/Single taper mel spectrogram with no smoothing
2. Cepstral-smoothed single taper mel spectrogram
3. Single-taper spectrogram with Gaussian smoothing

We found that multitaper and single-taper mel spectrograms are very similar and that cepstral smoothing improves performance for the mel spectrogram. Also, the single-taper spectrogram with Gaussian smoothing performs very poorly.

We will visualize each of these different time-frequency representations by viewing them applied to an utterance by a male speaker saying “His sudden departure shocked the cast”. The first group of spectrograms is shown in Figure 8. We also note that the mel spectrograms are on a different scale and range than the other spectrograms we use. Notably, the mel spectrum axis we use ranges from 0 to 6873 Hz while the spectrogram axis only ranges from 0 to 3000 Hz. In Figure 7 we plot the bin center frequency against its index. Note that the y-axis is log-scale, for the spectrogram plot the bin frequency relates linearly to the index. These different frequency ranges mean that visual comparisons between mel spectrograms and the other spectrograms are not straightforward.

8a, 8b and 8 were generated using MultitaperSpectrogram, and MelCepstralSmoothing was used for smoothing in the mel spectrogram in Fig 8a while GaussianSmoothing was used for smoothing in Fig 8b and Fig 8c. The difference between Fig 8b and Fig 8c is that the
former was smoothed by \textsc{GaussianSmoothing} with a filter $W \in \mathbb{R}^{7 \times 7}$ which has extent in both time and frequency so frequencies are averaged, whereas the latter was smoothed using a filter $W \in \mathbb{R}^{7 \times 1}$ which only has time extent, so that frequencies are not averaged. We show some more plots in Figure 9 that display two mel spectrograms without cepstral smoothing and a single-taper spectrogram that used \textsc{GaussianSmoothing} for smoothing. Visually, the main difference is (9c) in Figure 9 retains harmonic artifacts (note the horizontal lines) which make the binary edge feature computations more difficult (section 4).

4 Edge Models

The classification algorithms analyzed in this paper use local binary features on the log-spectrogram. These were first introduced by \cite{2} in a similar detection context. Given a log spectrogram $S \in \mathbb{R}^{T \times F}$ the local binary edge features will consist in an array

$$E \in \{0, 1\}^{T \times F \times D}$$

where $T$ is the length of the support for the utterance in time, $F$ is the number of frequency channels, and $D$ is the number of binary features. For our edge models $D = 8$ corresponding to 8 orientations. Informally, we compute the edge map representation $E$ by performing a non-linear filtering operation on $S$ with eight different oriented filters (corresponding to the eight edges). The output of each filtering operation is then quantized into \{0, 1\} so that we have eight time-frequency maps $M_e \in \{0, 1\}^{T \times F}$ where $M_e(t, f) = 1$ indicates than we detected an edge of type $e$ at time-frequency location $(t, f)$. We stack the maps $M_0, \ldots, M_7$ to form the edge-map $E$. 

17
Figure 9: Spectrograms used for auxiliary experiments. Note that the mel spectrograms are on a logarithmic frequency scale than the single taper spectrogram in subfig (c).
Input: \( S \in \mathbb{R}^{T \times F} \) smoothed log spectrogram, \( O \in \{-1, 0, 1\}^{8 \times 2} \) edge orientation matrix, \( \alpha \in \mathbb{R}^8 \) difference thresholds, \( \tau \in [0, 1] \) threshold quantile, \( L_{\text{block}} \in \mathbb{Z}_{>0} \) block length, \( \Delta \in \mathbb{Z}_+ \) smoothing radius,

Output: \( E \in \{0, 1\}^{T \times F \times 8} \) edge map

1: \textbf{procedure} \text{Edgemap}(S, O, \alpha, \rho, L_{\text{block}}, \Delta) \quad \triangleright \text{Compute an edgemap over } S
2: \quad D \leftarrow \text{EdgeDifferences}(S, O) \quad \triangleright \text{Compute the candidate gradients}
3: \quad E \leftarrow \text{EdgeThresholding}(D, \alpha, \rho, L_{\text{block}}) \quad \triangleright \text{Binarize, leave only large, maximal gradients}
4: \quad E \leftarrow \text{EdgeSpreading}(E, O, \Delta) \quad \triangleright \text{Spread the edges}
5: \quad \text{return } E
6: \textbf{end procedure}

Algorithm 5: Edge Computation Algorithm (EC)

4.1 Edge Computation Algorithm

Our edge detector for spectrograms and other time-frequency displays is detailed below in Algorithm 5. The algorithm takes local maximal differences in the spectrogram along eight orientations in Algorithm 6 (visualized below in Figure 10). We retain only those local maximal differences that are greater than zero and are local maxima along their orientation. We then apply an adaptive and absolute threshold to the retain differences, and all differences larger than both thresholds are set to 1 and all other entries are set to zero in Algorithm 7. Finally, the edges are spread along their orientation in Algorithm 8 which is a form of max-pooling. The amount of spreading is controlled by a parameter \( \Delta \). Spreading achieves invariance to local deformations of the signal of the form

\[
S'(t, f) = S(t + \delta_{\text{time}}(t), f + \delta_{\text{freq}}(f))
\]

where \( \|\delta\| \) is small and we find that it improves classification performance. It has been noted that time-scaling the signal \( x'(t) = x((1 - \epsilon)t \) (which corresponds to pitch modification) results in a deformation to a mel spectrogram where \( \delta_{\text{freq}}(f) \) is constant across different frequencies, but for linear-frequency scale spectrograms \( \delta_{\text{freq}}(f) \propto f \) so that higher frequencies experience larger deformations \[4\]: this indicates spreading should confer a particular advantage for mel spectrograms although we did not find that to be the case in our experiments.

\text{EdgeDifferences} (Algorithm 6) takes as input a spectrogram \( S \) and a matrix of orientations \( O \). The computation of \( S \) is discussed in subsection 3.2. The orientations \( O \) we use may be written
Figure 10: Eight binary edge masks

\[
O = \begin{bmatrix}
1 & 0 \\
-1 & 0 \\
0 & 1 \\
0 & -1 \\
1 & 1 \\
-1 & 1 \\
1 & -1 \\
-1 & -1 \\
\end{bmatrix} \begin{cases}
\text{Time Edges} \\
\text{Frequency Edges} \\
\text{Time-Frequency (Diagonal) Edges} \\
\end{cases}
\] (16)

In line 5 the difference at time frequency location \((t, f)\) at orientation \(o = O(e, :)\) is \(S(t + o_0, f + o_1) - S(t, f)\). In our notation \(O(e, :)\) refers to the \(e\)th row of \(O\). So if \(e \in \{0, 1\}\) then \(o\) will be a time edge since \(o_1 = 0\) so the difference is just \(\max\{0, S(t + o_0, f) - S(t, f)\}\) where \(o_0 \in \{-1, 1\}\). If \(e \in \{2, 3\}\) then \(o\) will be a frequency edge since \(o_0 = 0\) so the difference is just \(\max\{0, S(t, f + o_1) - S(t, f)\}\). Similar computations can be performed for the joint time-frequency diagonal edges. The eight varieties of edge masks used in \texttt{EdgeDifferences} (Algorithm 6) are visualized in Figure 10. These correspond to eight local gradients between neighboring pixels on a binary grid as shown in Fig 11. The next stage in Algorithm 6 is line 11 where we check whether the difference computed at time-frequency location \((t, f)\) along orientation \(e\) is a local maximum. Only local maxima are retained.

\texttt{EdgeThresholding} (Algorithm 7) binarizes the difference map with an absolute and adaptive threshold. The adaptive threshold varies over time. We split difference map \(D \in \mathbb{R}^{T \times F \times 8}\) along the time axis into \(N_B\) blocks

\[
\left\{D_0, \ldots, D_{N_B-1}\right\} \in \mathbb{R}^{L_{\text{block}} \times F \times 8}
\]

so that

\[
D = \begin{bmatrix}
D_0 \\
D_1 \\
\vdots \\
D_{N_B-1}
\end{bmatrix}.
\]

For each of these blocks \(D_b\) and each edge type \(e\) we estimate the \(\rho\)-quantile \(\lambda_{b,e}\) over the non-zero entries of \(D_b(:, :, e)\). We threshold every entry of \(D_b(:, :, e)\) to zero unless the entry is larger
Input: \( S \in \mathbb{R}^{T \times F} \) smoothed log spectrogram, \( O \in \{-1, 0, 1\}^{8 \times 2} \) edge orientation matrix,
Output: \( D \in \mathbb{R}^{T \times F \times 8} \) difference map

1: procedure \textsc{EdgeDifferences}(S, O) \>
  \( \triangleright \) Compute a difference map over \( S \)
  \( \triangleright \) Compute Differences
2: \hspace{1em} for \( e = 0, \ldots, 7 \) do
3: \hspace{2em} \( o \leftarrow O(e, :) \) \>
4: \hspace{2em} for all \( (t, f) \in [T] \times [F] \) do \>
5: \hspace{3em} \( D(t, f, e) \leftarrow \max \{0, S(t + o_0, f + o_1) - S(t, f)\} \)
6: \hspace{2em} end for
7: \hspace{1em} end for
8: \hspace{1em} for \( e = 0, \ldots, 7 \) do \>
9: \hspace{2em} \( o \leftarrow O(e, :) \) \>
10: \hspace{2em} for all \( (t, f) \in [T] \times [F] \) do \>
11: \hspace{3em} if \( D(t, f, e) < \max \{D(t + o_0, f + o_1, e), D(t - o_0, f - o_1, e)\} \) then \>
12: \hspace{3em} \( D(t, f, e) \leftarrow 0 \)
13: \hspace{3em} end if
14: \hspace{2em} end for
15: \hspace{2em} end for
16: \hspace{1em} return \( D \)
17: \hspace{1em} end procedure

Algorithm 6: Edge Differences Algorithm (ED)
than both $\lambda_{b,e}$ and the absolute threshold $\alpha(e)$ for a given edge type $e$. The threshold $\alpha(e)$ is a hyper-parameter that we set based on visual inspection of the edge-map, we intend to develop more rigorous procedures for selecting $\alpha$ in future work. The length of the signal, $T$, will often not be a multiple of the block length, $L_{\text{block}}$, which creates a tricky boundary problem since the blocks of that length will not evenly divide the signal. We handle this problem by letting the last two blocks overlap potentially: if

$$[(N_B - 2)L_{\text{block}}, (N_B - 1)L_{\text{block}} - 1]$$

is the second to last block interval and if $N_B L_{\text{block}} > T$ we let

$$[T - L_{\text{block}}, T - 1]$$

be the interval for the last block where $T - L_{\text{block}} < (N_B - 1)L_{\text{block}}$. We perform the thresholding operation on the second to last block as usual which means that entries in

$$[T - L_{\text{block}}, (N_B - 1)L_{\text{block}}]$$

are thresholded, and for the last block although we estimate the threshold $\lambda_{N_B,e}$ over the whole block

$$[T - L_{\text{block}}, T - 1],$$

but we only use $\lambda_{N_B,e}$ to threshold entries in the sub-block

$$[(N_B - 1)L_{\text{block}}, T - 1].$$

Thus, we always use a large window to estimate the threshold, but the segment thresholded can be shorter than the entire window. This block-based algorithm is an approximation to a procedure where one uses a sliding-window to estimate a threshold for each time point, and the local
Input: $D \in \mathbb{R}^{T \times F \times 8}$ difference map, $\alpha \in \mathbb{R}^8$ difference thresholds, $\rho \in [0, 1]$ threshold quantile, $L_{\text{block}} \in \mathbb{Z}_{>0}$ block length
Output: $E \in \{0, 1\}^{T \times F \times 8}$ binarized difference map

1: procedure EdgeThresholding($D, \alpha, \rho, L_{\text{block}}$)  
   $\triangleright$ Binarize a difference map $D$
2: $N_B = \text{ceiling}(T/L_{\text{block}})$
3: for $e = 0, \ldots, 7$ do
4:   for $b = 0, \ldots, N_B - 1$ do
5:     $T_{\text{end}} \leftarrow \min\{(b + 1)L_{\text{block}}, T\}$  
   $\triangleright$ Get the last index of the block, handling boundary effects
6:     $T_{\text{start}} \leftarrow T_{\text{end}} - L_{\text{block}}$  
   $\triangleright$ Start for the estimating block
7:     $V \leftarrow \{D(t,:,e) > 0 \mid T_{\text{start}} \leq t < T_{\text{end}}\}$
8:     $\lambda_{b,e} = \text{percentile}_\rho(V)$
9:     for all $(t,f) | bL_{\text{block}} \leq t < T_{\text{end}}, f \in [F]$ do
10:        $\triangleright$ Find the local maxima for the given orientation
11:        if $D(t,f,e) \geq \max\{\lambda_{b,e}, \alpha(e)\}$ then
12:           $D(t,f,e) \leftarrow 1$
13:        else
14:           $D(t,f,e) \leftarrow 0$
15:        end if
16:     end for
17: end for
18: end for
19: return $E$

Algorithm 7: Edge Thresholding Algorithm (ET)
max differences are thresholded at a given time point are thresholded by the sliding window estimate. Preliminary results indicated that the block-based and sliding-window based give very similar results.

**EDGESPREADING** [Algorithm 8](#) performs a “spreading” or max-filtering operation of the thresholded max-difference map. We input a $\Delta$ radius and then for the given edge type $e$ we apply a 1-dimensional max-filter of length $2\Delta + 1$ with orientation $O(e, \cdot)$ to the binary features. This spreading operation smears edges along their orientation axis.

### 4.2 Visualization

We wish to visualize the behavior of **EDGEMAP** [Algorithm 5](#) on the different time-frequency representations considered in **section 3**. The output of **EDGEMAP** is an array the same size as the time-frequency representation it operates on with a one or zero at each time-frequency location corresponding to whether an edge was detected there or not. For visualization purposes, we display a spectrogram in gray-scale and mark every time-frequency location where **EDGEMAP** detected an edge in that spectrogram (e.g. Figure 12). We find is that the best performing signal processing algorithm, **MULTITAPERSPECTROGRAM** with **GAUSSIANSMOOTHING** (as we will show in **subsection 6.4**), has edges that cluster into groups that are identifiably formants, formant transitions, and onsets. For each of the eight edge orientations $e$ we plot the locations where **EDGEMAP** detected edge $e$ on five different time-frequency representations: mel-spectrogram without cepstral smoothing, mel-spectrogram with cepstral smoothing, multitaper spectrogram with time-only smoothing, multitaper spectrogram with time-frequency smoothing, and a single taper spectrogram with time-frequency smoothing. In these plots we find that edges are usually detected along formants, formant transitions, and onsets, so the apparent false negative rate (chance of missing an important edge) is
low, however some of the plots have many false positive edges: i.e. edges that do not correspond to any obviously meaningful structure. In the case of mel spectrograms, frequency edges cluster cleanly along formants but the time edges are noisy with some occurring along obvious onsets while others appear to be randomly distributed. The single taper spectrogram also captures some structure but it has many “noise” edges that do not appear to cluster into any meaningful groups.

Figure 14 and 15 display the detected frequency edges and we see that for the The multitaper spectrogram (middle plot) appears to track most of the formants that are apparent in the spectrogram while the other spectrograms do not do as well. The bottom plot, in particular, appears to have many “noise” edges that do not belong to any obvious spatial cluster. has edges that track formants in, which are the plots displaying detected frequency edges. In Figure 16 and 17 the multitaper spectrogram cleanly clusters time-edges in lines corresponding to onsets. The diagonal edge displays in Figure 18, 19, 20, and 21.

In addition to the max-differences that are just in time or just in frequency there are the joint time frequency edges as shown in Fig 13.

After applying the max-differencing algorithm we then threshold the edges and spread them. We display the results of complete EDGEMAP algorithm in Fig 14, 15, 16, 17, 18, 19, 20, 21. We note that the multitaper spectrogram representations appear to be sparser than the other representations. The Hamming taper spectrogram has a lot of noise in each plot and the time-edges in the mel spectrograms are also very noisy Fig 16, 17. Time-smoothing the mel-spectrograms could potentially alleviate these problems.
Figure 12: Spectrograms and four difference maps demonstrating the \textsc{EdgeDifferences} algorithm.
Figure 13: Spectrograms and four difference maps demonstrating the EDGEDIFFERENCES algorithm.
Figure 14: Frequency Edges: Bottom to Top. These look for points along the frequency axis where lower frequencies have less energy and higher frequencies have more energy so $S(t, f + 1) > S(t, f)$. All three plots detect edges along the formants. The clusters in the mel spectrograms (top two plots) are mostly lines whereas in the multitaper spectrograms (middle and second from the bottom) the clusters are thicker. There appear to be a significant number of “noise” edges in the single-taper spectrogram. We note that there are some faint formant transitions that only the time-only-smoothed multitaper spectrogram picks up on, and that the edges do not seem to capture all of the relevant structures.
Figure 15: Frequency Edges: Top to Bottom. These are edges which detect a frequency location where the frequencies above have less energy than frequencies below so $S(t, f + 1) < S(t, f)$. These edges show a similar pattern to the edges in Figure 14 except that there are fewer noise edges and the different spectral features are better resolved. It is unclear why there is an apparent asymmetry between the edge detection algorithms.
Figure 16: Time Edges: Right to Left. These edges correspond to offsets $S(t, f) > S(t + 1, f)$. The main observation to make is that the mel spectrograms (top two plots) show many noise edges, the single taper spectrogram (bottom plot) shows a smaller number, and the multitaper plots (middle and second from the bottom) show very few. All plots appear to pick up on the same offset structures (i.e., a change from a high-energy region to a lower-energy region in time). The offsets picked up in this plot complement the onsets in Figure 17.
Figure 17: Time Edges: Left to Right. These are onsets which are time points where the energy sharply increases so that \( S(t, f) < S(t + 1, f) \). The mel spectrograms (top two plots) have many noise edges whereas the other three plots (multitaper and single-taper spectrograms) appear to have very few. The onsets picked up complement the offsets in Figure 16.

The appears to be some asymmetries in the edge detection problem: namely detecting an onset appears to be easier than detecting an offset and detecting frequency edges where the higher frequency has more energy and the lower frequency has less energy is easier than detecting edges of the opposite polarity. We do not know exactly what causes these differences, but, potentially these asymmetries indicate that specialized processing should be used for the different types of edges. We also note that with the except of the time-edges, the clusters in the mel spectrograms are line segments: consider the frequency edges and other diagonal edges in Figure 14, 15, 18, 19, 20, and 21. For each time point spanned by one of the line-segment clusters in the mel spectrogram there is at most one detected edge. The clusters in the multitaper spectrograms, however, are “thicker” and often have multiple detected edges across several frequency bands for each time point.
Figure 18: Bottom left to top right, these correspond to $S(t, f) < S(t + 1, f + 1)$. These features pick up on both onsets and formants. The single taper spectrogram appears to have many noise edges.
Figure 19: Top right to bottom left, these correspond to $S(t, f) > S(t+1, f+1)$. We note that the multitaper spectrogram shows “thicker” clusters than the mel spectrograms which most has clusters that are lines.
Figure 20: Bottom right to top left, these correspond to $S(t, f + 1) > S(t + 1, f)$, compared to the other plots the single taper spectrogram appears to have many fewer noise edges.
5 Statistical Models of Binary Data

The statistical workhorse in this thesis is the mixture of multivariate Bernoulli models. We will use this distribution to construct novel features and speech detectors.

5.1 Bernoulli Models

Our setting is that we have multiple observations \( \{E_0, E_1, \ldots, E_{n-1}\} \) of edge-map data where \( E_i \in \{0, 1\}^{T \times F \times D} \) where \( D \) is the number of edges (in our case eight). We assume our data is independently and identically distributed. We first consider a simple model (used in [2]) known as multivariate Bernoulli distributions that treats each coordinate \( E_i(t, f, e) \) of datum \( E_i \) as an independent, biased coin flip: the model assigns a probability \( P(t, f, e) \) to the event that \( E(t, f, e) = 1 \), a probability \( 1 - P(t, f, e) \) that \( E(t, f, e) = 0 \). The independence constraint means that

\[
\mathbb{P}(E = E') = \prod_{t,f,e} P(t, f, e)
\]
where $E' \in \{0,1\}^{T \times F \times D}$. For each coordinate $(t,f,e)$ of the $E_i$ in the set $[T] \times [F] \times [D]$ we have a parameter $P(t,f,e)$ in this model. The data then have a joint likelihood

$$L(E_0, \ldots, E_{n-1} | p) = \prod_{i=0}^{n-1} \prod_{t,f,e} P(t,f,e)^{E_i(t,f,e)} \left(1 - P(t,f,e)\right)^{1-E_i(t,f,e)}.$$  

The log-likelihood functions takes a linear form:

$$\log L(E_0, \ldots, E_{n-1} | P) = \sum_{i=0}^{n-1} \sum_{t,f,e} E_i(t,f,e) \log P(t,f,e) + (1 - E_i(t,f,e)) \log(1 - P(t,f,e)).$$  

In the above examples our parameters $P \in (0,1)^{T \times F \times E}$ (we exclude the possibility of $P(t,f,e) = 1$ since we can just eliminate that coordinate without changing the distribution) are arrays of the same dimension as the edge features $E_i$. We will refer to these parameter arrays as “templates”. Coordinates $(t,f,e)$ with a larger value of $P(t,f,e)$ indicates a greater likelihood of observing an edge, smaller values indicate that an edge is unlikely, and a middle value indicates uncertainty.

### 5.1.1 The Background Model

As an example application of Bernoulli models we present the background model from [2], which will be of great use for detection in section 6. The background model, denoted by $P_{bgd}$ can be used to construct a simple model of an edge map $E \in \{0,1\}^{T \times F \times D}$. We can think of $E$ as consisting of $T$ frames $E(t) \in \{0,1\}^{F \times D}$:

$$E = \left[ \begin{array}{c} E(0;,:,:) \\ E(1;,:,:) \\ \vdots \\ E(T-1;,:,:) \end{array} \right]$$

and $P_{bgd}$ models frames $E(t)$ as independent draws from a product of Bernoulli distributions (henceforth called a multivariate Bernoulli distribution) parameterized by a template $P_{bgd} \in (0,1)^{F \times D}$ which we note does not depend on time only on frequency $f$ and edge type $e$. The log-likelihood for a dataset of edge-maps $\{E_0, \ldots, E_{n-1}\}$ under this model is

$$P_{bgd}(\{E_i\}_{i=0}^{n-1}) = \prod_{i=0}^{n-1} \prod_{t=0}^{T_i-1} \prod_{f,p} P_{bgd}(f,p)^{E_i(t,f,p)} \left(1 - P_{bgd}(f,p)\right)^{1-E_i(t,f,p)}$$  

where, for each $i$, $T_i$ is the time extent of edge-map $E_i$.

### 5.1.2 Mixture Models

However, there is great variation in the way a given phone or speech sound is pronounced. The formants (high-energy lines in the spectrogram created by vocal tract resonances) of male speakers tend to be in a lower frequency than the formants for female speakers. At detection time we are not told whether the speaker is male or female, so we would like to have a model that accommodates such variation. This motivates a mixture-model approach.

Rather than having a single template $P_{bgd}$ as in the case of the background model, we parameterize the mixture model $P_{mix}$ with several templates $P_0, P_1, \ldots, P_{C-1}$ and a set of weights
The indices $c \in [C]$ are the latent classes or components of the mixture model, $P_c$ is the component template, and $w_c$ is the component weight. In the mixture model setting observations $E_i$ are treated as incomplete observations of a pair $(E_i, z_i)$ where the latent variable $z_i \in \{0, 1\}^C$ is a vector of $C - 1$ zeros and a single one. There are $[C]$ possibilities for $z_i$ and the weights $(w_0, \ldots, w_C)^\top$ form a categorical distribution over these $C$ possibilities for $z_i$. The likelihood in the fully observed case is

$$P(E, z) = P(z)P(E \mid z) = \prod_{c=0}^{C-1} \left( w_c \prod_{t,f,e} P_c(t,f,e)^{E(t,f,e)}(1 - P_c(t,f,e))^{1-E(t,f,e)} \right)^{z(c)}. \quad (18)$$

Thus, we have the conditional likelihood of $X$ conditioned on $z$ is

$$P(E \mid z) = \prod_{c} \left( \prod_{t,f,e} P_c(t,f,e)^{E(t,f,e)}(1 - P_c(t,f,e))^{1-E(t,f,e)} \right)^{z(c)}. \quad (19)$$

Using the law of total expectation the data can be written as a linear combination of these conditional models:

$$P(X) = \sum_{c=0}^{C-1} w_c \prod_{t,f,e} P_c(t,f,e)^{E(t,f,e)}(1 - P_c(t,f,e))^{1-E(t,f,e)}. \quad (19)$$

The full likelihood of the sample will then be a product over the individual likelihoods:

$$P(E_0, \ldots, E_{n-1}) = \prod_{i=0}^{n-1} P(E_i).$$

Noting that $w_c = P(z(c) = 1)$ and $\sum_{c=0}^{C-1} w_c = 1$, we add the restriction $0 < w_c < 1$ since if any component weight is zero we can simply drop that component from the model without changing the likelihood. Similarly, if a component has weight one so $w_c = 1$ then all the other components must be zero so we would drop them.

### 5.2 Estimation via Maximum Likelihood

The estimation problem is to infer the parameters $P_0, P_1, \ldots, P_c$ and $w_0, \ldots, w_{C-1}$ given finitely many data. We use approximate maximum likelihood algorithms in this work.

Estimation is simple in the case of a single mixture component ($C = 1$). Our only weight parameter $w_0$ is equal to 1, hence we are only concerned with estimating the template $P_0$. An important case of this is the background model discussed in **subsection 5.1.1**. The full likelihood was given in **Equation 17**. To compute the maximum likelihood solution we find critical points of the log-likelihood function. Namely, those $P$ satisfying

$$\frac{\partial}{\partial P(f,p)} \sum_{i=0}^{n-1} \sum_{t=0}^{T_i-1} \sum_{f', p'} E_i(t,f',p') \log P(f',p') + (1 - E_i(t,f',p')) \log(1 - P(f',p')) = 0$$

for every frequency channel $f$ and edge-type $p$. This gives us a set of equations to solve for each coordinate $(f,p)$:
0 = \frac{\partial}{\partial P(f,p)} \sum_{i=0}^{n-1} T_i - 1 E_i(t, f, p) \log P(f, p) + (1 - E_i(t, f, p)) \log(1 - P(f', p'))

= \sum_{i=0}^{n-1} T_i - 1 E_i(t, f, p) - 1/E_i(t, f, p)

= \sum_{i=0}^{n-1} T_i - 1 E_i / P(f, p) (1 - P(f')) - (1 - E_i) / P(f, p) (1 - P(f'))

= \sum_{i=0}^{n-1} T_i - 1 E_i(t, f, p) - P(f, p)

\hat{P}(f, p) = \frac{1}{\sum_{i=0}^{n-1} T_i} \sum_{i=0}^{n-1} T_i - 1 E_i(t, f, p)

Which means that \( \hat{P}(f, p) \) is just the empirical frequency of the edges of type \( p \) detected in channel \( f \).

It is straightforward to generalize that result to the case where the model \( P \in \{0, 1\}^{T \times F \times D} \). The maximum likelihood estimator for \( P \) is simply an average over the data so that

\[
\hat{P}(t, f, e) = \frac{1}{n} \sum_{i=0}^{n-1} E(t, f, c). \tag{20}
\]

The log-likelihood function of the data (a linear function of \( \log P \) and \( \log(1 - P) \)) is very sensitive to when \( \hat{P}(t, f, e) \) is very close to zero or very close to one and small changes in that region results in large changes in the likelihood. In order to make the estimate robust we use a modified procedure where

\[
\hat{P}(t, f, e) = \begin{cases} 
1 - \epsilon & M(t, f, e) > 1 - \epsilon \\
M(t, f, e) & \epsilon \leq M(t, f, e) \leq 1 - \epsilon \\
\epsilon & M(t, f, e) 
\end{cases}
\]

where \( M(t, f, e) \) is the unadjusted average as in Equation 20.

### 5.3 Estimation via EM

In the case of a multi-component mixture, however, the optimization of the likelihood is no longer possible analytically. We instead use BERNOULLIMIXTUREEM an expectation-maximization algorithm \([9]\) for Bernoulli mixture data that alternates between:

1. Computing the expected sufficient statistics for the latent variables condition on the observed data (E-step)
2. Picking the parameters that maximize the likelihood using the sufficient statistics (M-step).

The setting is that we have data \( \{(E_i, z_i)\}_{i=0}^{n-1} \) and an initial model \( \theta = \{(P_c, w_c)\}_{c=0}^{C-1} \). The \( X_i \) are observed but the \( z_i \) are latent. BERNOULLIMIXTUREEM finds parameters \( \{(\hat{P}_c, \hat{w}_c)\}_{c=0}^{C-1} \) that increase the likelihood iteratively until convergence.

Estimation would be simple if we could observe the \( z_i \) since, in that case, we could just use empirical frequencies. The EM algorithm computes expected values for the \( z_i \) conditioned on the observed data and use those to compute frequencies.

In each iteration we compute the discrete distributions \( \{P_{\theta}(z_i | E_i)\}_{i=0}^{n-1} \). Each component \( c \) of the vector \( P_{\theta}(z_i | E_i) \in \{0, 1\}^c \) is the expected number of times \( z_i(c) = 1 \) conditioned on observing \( E_i \), hence \( P_{\theta}(z_i | E_i) \in \{0, 1\}^c \) are the expected sufficient statistics for the conditional distribution.

38
Input: \( \{E_i\}_{i=0}^{n-1} \subset \{0, 1\}^{T \times F \times 8} \) data, \( C \in \mathbb{Z} \) number of components, \( \epsilon \in \mathbb{R}_{>0} \) clipping factor, \( \tau \in \mathbb{R}_{>0} \) stopping criterion

Output: \( \{P_c\}_{c=0}^{C-1} \in (\epsilon, 1-\epsilon)^{T \times F \times 8} \) templates, \( \{w_c\}_{c=0}^{C-1} \in \Delta_{C-1} \) template weights

1: procedure \textsc{BernoulliMixtureEM}(\( \{E_i\}_{i=0}^{n-1}, C, \epsilon, \tau \)) \Comment{Cluster the data}
2: \( \{(P_c, w_c)\}_{c=0}^{C-1} \leftarrow \text{InitBernoulliEM}(\{E_i\}_{i=0}^{n-1}, C) \)
3: \( L_{\text{prev}} \leftarrow -\infty \)
4: \( L, A \leftarrow \text{Bernoulli-EStep}(\{E_i\}_{i=0}^{n-1}, \{(P_c, w_c)\}_{c=0}^{C-1}) \) \Comment{Algorithm 10}
5: while \( |(L - L_{\text{prev}})/L| > \tau \) do
6: \( \{(P_c, w_c)\}_{c=0}^{C-1} \leftarrow \text{Bernoulli-MStep}(E_i, \{(P_c, w_c)\}_{c=0}^{C-1}, A, \epsilon) \) \Comment{Algorithm 11}
7: \( L, A \leftarrow \text{Bernoulli-EStep}(\{E_i\}_{i=0}^{n-1}, \{(P_c, w_c)\}_{c=0}^{C-1}) \)
8: end while
9: return \( \{(P_c, w_c)\}_{c=0}^{C-1} \)
10: end procedure

Algorithm 9: Bernoulli Mixture EM

of the \( z_i \) under a model \( \theta \). That computes \textsc{Bernoulli-EStep} \Comment{Algorithm 10}. We then choose a new \( \theta' \) to maximize the likelihood of the fully observed data model where we use the expected sufficient statistics from the conditional model in lieu of actually observing the \( z_i \). We then find a new set of sufficient statistics \( \{P_{\theta'}(z_i | E_i)\}_{i=0}^{n-1} \) where those are computed under the model \( \theta' \).

\textsc{BernoulliMixtureEM} \Comment{Algorithm 9} implements the EM algorithm.

In line 4 we introduce a matrix \( A \in \Delta_{C-1}^n \) that contains the estimated sufficient statistics \( \{P_{\theta'}(z_i | E_i)\}_{i=0}^{n-1} \) where \( \theta' = \{(P_c, w_c)\}_{c=0}^{C-1}, \Delta_{C-1} \subset \mathbb{R}^C \) is the probability simplex that defines every categorical distribution with \( C \) categories.

Within \textsc{BernoulliMixtureEM} we call \textsc{Bernoulli-EStep} \Comment{Algorithm 10} to estimate the sufficient statistics and \textsc{Bernoulli-MStep} \Comment{Algorithm 11} to estimate the parameters.

Line 4 computes the sufficient statistics in the log-domain for numerical stability reasons. We recall that our estimation procedure means that

\[
P_c(t, f, e), 1 - P_c(t, f, e) \leq 1 - \epsilon
\]

which means that the raw likelihood of an observation \( E_i \) is

\[
\prod_{t, f, e} P_c(t, f, e)^{E(t, f, e)}(1 - P_c(t, f, e))^{1 - E(t, f, e)} \leq (1 - \epsilon)^{T \cdot F \cdot D}
\]

which will be too small to perform the normalization in line 13. By taking the logarithms we avoid this underflow issue. We need to compute the lower bound \( Q \) that matches the conditional expectation of \( z \), so we need to convert back out of the log-domain and normalize: i.e.

\[
Q_i(c) = \frac{\exp(A(i, c))}{\sum_{c'} \exp(A(i, c'))}.
\]

We observe that

\[
Q_i(c) = \frac{\exp(A(i, c)) - \max_{c'} A(i, c')}{\sum_{c'} \exp(A(i, c') - \max_{c'} A(i, c'))}
\]
Input: \( \{E_i\}_{i=0}^{n-1} \subset \{0,1\}^{T \times F \times 8} \) data, \( \{(P_c, w_c)\}_{c=0}^{C-1} \) mixture parameters
Output: \( L \) log-likelihood, \( A \) Estimates of affinities

1: procedure BERNOULLI-EStep(\( \{E_i\}_{i=0}^{n-1} \), \( \{(P_c, w_c)\}_{c=0}^{C-1} \)) \( \triangleright \) Compute Affinities
2: for \( i = 0, \ldots, n - 1 \) do
3:   for \( c = 0, \ldots, C \) do
4:     \( A(i, c) \leftarrow \log w_c \sum_{t,f,e} E_i(t, f, e) \log \frac{P_c(t, f, e)}{1 - P_c(t, f, e)} + \log (1 - P_c(t, f, e)) \)
5:   end for
6: end for
7: \( L \leftarrow \sum_{i,c} A(i, c) \)
8: for \( i = 0, \ldots, n - 1 \) do
9:   \( c_{\text{max}} \leftarrow \arg \max_{c} A(i, c) \)
10: \( A(i,:) \leftarrow \exp\{A(i,:) - A(i, c_{\text{max}}(i))\} \)
11: \( \text{row \_sum} \leftarrow \sum_{c} A(i, c) \)
12: for \( c = 0, \ldots, C \) do
13:   \( A(i, c) \leftarrow A(i, c) / \text{row \_sum} \)
14: end for
15: end for
16: return \( L, A \)
17: end procedure

Algorithm 10: Bernoulli Mixture E-step
and note that if we subtract by the maximum value over \( A(i,:) \) then \( \exp(A(i,:)) \) will have at least one entry 1 so normalizing the vector will not encounter underflow problems in line 13.

| Input: \( \{E_i\}_{i=0}^{n-1} \subset \{0,1\}^{T \times F \times 8} \) data, \( A \in \Delta_{C-1}^n \) affinities |
| Output: \( L \) log-likelihood, \( A \) Estimates of affinities |

1. **procedure** Bernoulli-MStep(\( \{E_i\}_{i=0}^{n-1}, A, \epsilon \)) > Compute Affinities
   2.   for \( c = 0, \ldots, C \) do
   3.     \( P_c \leftarrow \sum_{i=0}^{n-1} A(i,c) E_i \)
   4.     \( P_c \leftarrow \min\{\max\{P_c, \epsilon\}, 1 - \epsilon\} \)
   5.     \( w_c \leftarrow \frac{1}{n} \sum_{i=0}^{n-1} A(i,c) \)
   6.   end for
   7.   return \( \{(P_c, w_c)\}_{c=0}^{C-1} \)
   8. **end procedure**

**Algorithm 11:** Bernoulli Mixture M-Step

### 5.4 Part Models

We use the Bernoulli models above for classification, but also as a feature transform. Namely, they are used to cope with the redundancy of the edge-data. These models were developed in [6]. We model the edge map features using patches: each patch \( U \) is a contiguous set of binary features extracted from an input edgemap \( X \in \{0,1\}^{T \times F \times 8} \). We let \( L_t \) be the time length of the path and \( L_f \) be the frequency width of the patch so that \( U \in \{0,1\}^{L_t \times L_f \times 8} \) where \( U(\tau, \omega, e) = E(\tau + u_t, \omega + u_f, e) \) so that \( u_t, u_f \) is the time frequency location for the patch and the patch is a subwindow of \( X \).

We construct a Bernoulli model of the patches. We assume that we have edgemaps for utterances \( \{E_i\} \) and that we then densely sample patches \( \{U_j\}_{j=0}^{n_p-1} \) from those edgemaps. We then have \( n_p \) patches in total. If we have \( C_p \) latent patch classes then

\[
\mathbb{P}(U_0, \ldots, U_{n_p-1}) = \prod_{i=0}^{n_p-1} \sum_{c=0}^{C_p-1} w_c^{\text{patch}} \prod_{t,f,e} Q_c^{\text{patch}}(t, f, e) U(t,f,e) (1 - Q_c(t, f, e))^{1-U(t,f,e)}
\]

is our complete likelihood for the data with \( w_c \) being the weights for a class and \( Q_c \) being the patch class template. We estimate these parameters using the **BERNOULLIMIXTUREEM** (Alg 9) algorithms over unlabeled training data. We wish to capture interesting edge information so we only train on patches that have a minimum number of edges \( \tau_{\text{patch}} \) (in our experiments 13). For training the part templates we only sample patches from the training data that are a local maximum within a neighborhood. Namely, we construct an edge count map \( EC \in \{0,1\}^{T \times F} \) for each edgemap \( X \in \{0,1\}^{T \times F \times 8} \) where \( EC(t, f) \) is the number of edges in patch \( E(t : t + L_t, f : f + L_f,:) \). We use a patch \( E(t : t + L_t, f : f + L_f,:) \) only if

\[
EC(t, f) = \max_{|\delta_t|,|\delta_f| \leq \xi} EC(t + \delta_t, f + \delta_f)
\]

41
where $\xi$ is the neighborhood radius used to determine whether a patch has a locally maximal number of edges.

We summarize the part template training algorithm in Alg 12:

\begin{verbatim}
Input: $\{E_i\}_{i=0}^{n-1} \subset \{0,1\}^{T \times F \times 8}$ data, $\epsilon \in \mathbb{R}_{>0}$ edge probability clipping factor, $\tau_{EM} \in \mathbb{R}_{>0}$ EM stopping criterion, $L_t \in \mathbb{Z}$ patch time length, $L_f \in \mathbb{Z}$ patch frequency width, $C_p \in \mathbb{Z}$ number of patches to estimate, $\tau_{patch} \in \mathbb{Z}$ minimum edge threshold, $\xi \in \mathbb{Z}$ edge local max neighborhood
Output: $\{Q_c\}_{c=0}^{C_p-1}$ part models of patches

1: procedure PartModelTraining($\{E_i\}_{i=0}^{n-1}, \epsilon, \tau_{EM}, \tau_{patch}, L_t, L_f$) $\triangleright$ Estimate part model over patches
2: $\{U_j\}_{j=0}^{n_p-1} \leftarrow$ ExtractPatches($\{E_i\}_{i=0}^{n-1}, \tau_{patch}, \xi, L_t, L_f$)
3: $\{(Q_c, w_c)\}_{c=0}^{C_p-1} \leftarrow$ BernoulliMixtureEM($\{U_j\}_{j=0}^{n_p-1}, C_p, \epsilon, \tau_{EM}$)
4: return $\{Q_c\}_{c=0}^{C_p-1}$
5: end procedure

Algorithm 12: Part Training

Alg 12 relies on ExtractPatches which performs the patch selection process. That procedure, is summarized in Alg 13
\end{verbatim}
Input: \( \{E_i\}_{i=0}^{n-1} \subset \{0,1\}^{T \times F \times 8} \) data, \( L_t \in \mathbb{Z} \) patch time length, \( L_f \in \mathbb{Z} \) patch frequency width, \( \tau_{\text{patch}} \in \mathbb{Z} \) minimum edge threshold, \( \xi \in \mathbb{Z} \) edge local max neighborhood
Output: \( U = \{U_j\}_{j=0}^{n-1} \) extracted patches

Algorithm 13: ExtractPatches

1: procedure ExtractPatches\((\{E_i\}_{i=0}^{n-1}, \tau_{\text{patch}}, L_t, L_f)\)
2: \( U \leftarrow \emptyset \)
3: for all \((i,t,f)\) do
4: \( EC(i,t,f) \leftarrow \langle 1, E_i(t : t + L_t, f : f + L_f,:) \rangle \) \hspace{1cm} \( \triangleright \) Inner product with vector of ones counts the edges
5: end for
6: for all \((i,t,f)\) do
7: if \( \text{LocalMax}(EC,i,t,f,\xi) \) then
8: if \( EC(i,t,f) \geq \tau_{\text{patch}} \) then
9: \( U \leftarrow U \cup \{E_i(t : t + L_t, f : f + L_f,:\}) \)
10: end if
11: end if
12: end for
13: return \( U \)
14: end procedure

Where in line 4 we use the fact that we can take the inner product of a vector of ones, 1, and the patch to count the edges. In line 7 we call a \( \text{LocalMax} \) function which just returns a True or False depending on whether \( EC \) attains the local maximum at \( EC(i,t,f) \) within the \( \xi \)-ball \( N_i(t; f; i, \xi) \) of \((t,f)\) in \( EC(i,:, :) \) under the sup-norm:

\[
N(t,f; i, \xi) = \{EC(i, t + \delta_t, f + \delta_f) \mid |\delta_t|, |\delta_f| \leq \xi\}.
\]

5.5 Part Transform

We may then code an edgemap \( E \in \{0,1\}^{T \times F \times 8} \) as a part map \( Y \in \{0,1\}^{T \times F \times C_p} \) using the estimated \( \{Q_c\}_{c=0}^{C_p-1} \). The part transform is performed using a hypothesis test

\[
Y(t,f,i) = \begin{cases} 
1 & i = \arg \max_j \mathbb{P}(E(t : t + L_t, f : f + L_f,:; Q_i), \langle 1, E(t : t + L_t, f : f + L_f,:) \rangle) \geq \tau_{\text{patch}} \\
0 & \text{otherwise}
\end{cases}
\]

where \( E(t : t + L_t, f : f + L_f,:) \) refers to the patch within \( X \) that includes all \( \tau, \omega, e \) locations satisfying \( t \leq \tau < t + L_t \) and \( f \leq \omega < f + L_f \). 1 denotes a array of all ones (chosen to have the same dimension as \( E(t : t + L_t, f : f + L_f,:) \)) and \( \langle :, :) \rangle \) denotes the Euclidean inner product so that \( \langle 1, E(t : t + L_t, f : f + L_f,:) \rangle \) counts the number of edges in the patch. We add this last condition because the part template \( Q_c \) that matches a patch with a small number of edges is
meaningless. One can view this secondary condition as performing a test between the data and a uniform background model.

We want this part transform to be robust to local deformations so we employ spreading (a local max filter) to the part-map $Y$ where

$$Y^{\text{spread}}(t,f,i) = \max_{|\delta_t|,|\delta_f|<\xi} Y(t+\delta_t, f+\delta_f, i)$$

where $\xi$ is some integer indicating the size of the neighborhood we spread over. The parameters of the patch model $\{(w_c, Q_c)\}$ can be estimated by BERNOLLI-MIXTURE-EM (Algorithm 9). Given the templates from a part model $\theta = \{Q_c\}_{c=0}^{C-1}$ we wish to compute a part transform that codes an edge map $E \in \{0,1\}^{T \times F \times 8}$ into a parts representation $P \in \{0,1\}^{T \times F \times C}$ so that we associate a small number of parts with each time-frequency location (those are locations in $P$’s indices that have ones).

### Algorithm 14: Part Transform Algorithm

**Input:** $E \in \{0,1\}^{T \times F \times 8}$ binary thresholded edge map $\theta \in \Theta$ part model, $\Delta \in \mathbb{Z}_+$ spreading radius, $\eta \in \mathbb{Z}_+$ minimum edge threshold  

**Output:** $E \in \{0,1\}^{T \times F \times 8}$ spread edge map

1: **procedure** EDGE-SPREADING($E, O, \Delta$) 
2:  
3:  
4:  
5:  
6:  
7:  
8:  
9: **end procedure**

\[1: \text{Spread the edgemap } E \]

\[2: \text{Get an orientation} \]

\[3: \text{Perform max pooling} \]

### 5.6 Visualization of the Part Transform

We used 100,000 patches sampled from 100 utterances using the EXTRACT PATCHES Alg 13 algorithm. These were trained with PART MODEL TRAINING Alg 12. The parameters used were $L_t = 5$, $L_f = 5, \epsilon = 0.95, \tau_{\text{EM}} = 1e-6, \tau_{\text{patch}} = 13$, and $\xi = 2$. We estimated $C_p = 75$ parts but threw away a few degenerate parts to get a total of 73 parts.

These parts can be visualized using a weighted average of spectrogram data. Namely, associated with the input edgemaps $\{E_i\}_{i=0}^{n-1}$ we have a set of spectrograms $\{S_i\}_{i=0}^{n-1}$ where $E_i$ is the edgemap generated from $S_i$. A patch within $E_i$ is just the set of edge responses for a collection of time-frequency locations $U_{u_t, u_f}$ indexed by base location $(u_t, u_f)$. Hence, for each patch $U_j$ we can find a spectral patch $V_j$ for the same utterance at the same time-frequency locations. $U_j \in \{0,1\}^{L_t \times L_f \times 8}$

44
is inherently hard to visualize given that it has three dimension, but $V_j \in \mathbb{R}^{L_t \times L_f}$ is much more straightforward to visualize.

To construct the visualization we note that PartModelTraining (Alg 12) in line 3 uses BernoulliMixtureEM (Alg 9), which is what returns the distribution parameters $\{Q_c\}_{c=0}^{C_p-1}$. In the writeup of BernoulliMixtureEM we throw away the matrix $A$ which has the property that $A(j,c)$ is an estimate of the likelihood that patch $U_j$ was generated by part template $Q_c$. Furthermore in the Bernoulli-MStep (Alg 11) the part templates are formed from a weighted sum of the data where the weights are given by the matrix $A$ so that

$$Q_c = \frac{\sum_{j=0}^{n_p-1} A(j,c)U_j}{\sum_{j=0}^{n_p-1} A(j,c)}.$$

This gives us an algorithm for constructing visualizations of the spectrogram: namely we average the spectrograms associated with the patches $U_j$ and we weight each spectrogram patch $V_j$ by $A(j,c)$ which gives us

$$Q_{c}^{spec} = \frac{\sum_{j=0}^{n_p-1} A(j,c)V_j}{\sum_{j=0}^{n_p-1} A(j,c)}.$$

With general speech data we would probably have to perform some normalization on $Q_c$ since some patches will be much louder than others which corresponds to an additive constant in the log spectral domain, however, we find that this does not make much of a difference for our purposes. Using this visualization technique we get the following picture for our parts Fig 22.

![Figure 22: Estimated parts: average of spectrogram data. Darker areas correspond to higher spectral energy and lighter regions correspond to lower spectral energy.](image)

We note that although there is a great deal of redundancy in the parts, in practice there appear to be benefits to using a larger number of parts that outweigh this disadvantage. The edge-map visualization of the parts is not terribly informative except that it demonstrates the degree of specialization of the parts. We chose eight different parts and displayed the part as an averaged spectrogram along with the eight edge orientations these are in Figure 23, Figure 24, Figure 25, Figure 26, Figure 27, Figure 28, Figure 29, Figure 30.

45
Figure 23: Top left is averaged spectrogram associated with the part. Each of the other plots is an edge map, darker areas are associated with a higher probability.

Figure 24: Top left is averaged spectrogram associated with the part. Each of the other plots is an edge map, darker areas are associated with a higher probability.

Figure 25: Top left is averaged spectrogram associated with the part. Each of the other plots is an edge map, darker areas are associated with a higher probability.
Figure 26: Top left is averaged spectrogram associated with the part. Each of the other plots is an edge map, darker areas are associated with a higher probability.

Figure 27: Top left is averaged spectrogram associated with the part. Each of the other plots is an edge map, darker areas are associated with a higher probability.

Figure 28: Top left is averaged spectrogram associated with the part. Each of the other plots is an edge map, darker areas are associated with a higher probability.
Figure 29: Top left is averaged spectrogram associated with the part. Each of the other plots is an edge map, darker areas are associated with a higher probability.

Figure 30: Top left is averaged spectrogram associated with the part. Each of the other plots is an edge map, darker areas are associated with a higher probability.
Another use for the EM algorithm is to cluster sounds and estimate probability distributions over their time-frequency representations. Our setting is that we have a set of training examples of a particular speech sound (e.g., a phone, diphone, word, etc.) \( \{ E_i \}_{i=0}^{n-1} \) and we wish to estimate a model for that sound. Generally, each example \( X \) has been extracted from an utterance \( E \) where \( E \) has a some sort of temporal labeling that indicates the support of the sound along the time axis. Formally, there will be times \( t_0 \) and \( t_1 \) such that for all times \( t \) with \( t_0 \leq t < t_1 \) the features in \( E \) are associated with the sound we are extracting. In that case the extracted example \( X = E(t_0 : t_1 ; : : ) \).

Given a collection of such extracted examples \( \{ E_i \}_{i=0}^{n-1} \), often different \( E_i \) will have different time extents. This means that there will be pairs of examples \( j \neq i \) such that \( E_i \in \{ 0, 1 \}^{L_i \times F \times D} \) and \( E_j \in \{ 0, 1 \}^{L_j \times F \times D} \) with \( L_i \neq L_j \) (here \( F \) is the number of frequency channels and \( D \) is the number of binary features at a time-frequency location: eight in the case of edges). If the examples have different lengths then we cannot apply \textsc{BernoulliMixtureEM} (Alg 9) directly to the set \( \{ E_i \}_{i=0}^{n-1} \) to estimate templates for the speech sound because the algorithm requires every example to have the same dimension.

To see the extent of the problem we illustrate the variability of duration for the diphone /ar/. We consider 1078 examples of /ar/ from the TIMIT database, a histogram for the different time lengths of /ar/ is in Figure 31.

![Figure 31: Histogram of the length distribution of /ar/](image)

BERNOULLIMIXTUREEM requires we adjust the data. A standard approach is time-normalization which has a history extending back to the earliest recognizers from the 1960s [50] (and a couple of references therein). In [2] the authors proposed a standard time-warping procedure for the utterances. In this paper we do not use time-warping. Timing cues are important for speech [11, 44] and could possibly be interfered with by warping the data. Our tactic is to pad each instance \( \{ E_i \} \) so that

\[
E_i^{\text{padded}} = [ E_i \ G_i ]
\]

and we choose \( G_i \) such that all the \( \{ E_i^{\text{padded}} \} \) have the same length. Let \( E_i \in \{ 0, 1 \}^{l_i \times F \times D} \) and \( l_{\text{max}} = \max_i l_i \). There are several potential choices for \( G_i \):
1. Zeros
2. $E(t_i + l_i : t_i + l_{\text{max}})$ where $E_i = E(t_i : t_i + l_i)$
3. Random pattern

In the second case $[X \ G]$ is a contiguous set of features from $E$. Alternatively, we could pick $G$ to be the zero array. For this paper we choose to let $G$ be random, where each frame of $G$ is drawn from the distribution defined by $P_{bgd}$. The rationale for this choice is explained in the following section.

Once the examples are all the same length $\{E_p^\text{padded}\}$ we can apply $\text{BERNOULLI MIXTURE EM}$ and obtain a set of templates $\{P_c\}_{c=0}^{C-1}$ where each $P_c$ models a latent class of the sound realized in the examples $\{E_i\}$.

### 5.7.1 Length Clustering in the Template Models

In fact, we find that the training procedure where we run $\text{BERNOULLI MIXTURE EM}$ on examples padded with random data produces detectors for examples of all lengths: long and short. There are two reasons for this:

1. $\text{BERNOULLI MIXTURE EM}$ with padded examples implicitly clusters examples by length
2. A likelihood ratio detector testing against background implicitly truncates the template to the average length of its cluster members

To illustrate the length clustering phenomenon we will visualize detectors learned using $\text{BERNOULLI MIXTURE EM}$. The detectors we consider were estimated to detect /ar/ in continuous speech. We have a collection of $n$ examples of /ar/ $\{E_i\}_{i=0}^{n-1}$ that have lengths $\{l_i\}_{i=0}^{n-1}$ and in order to use these in $\text{BERNOULLI MIXTURE EM}$ we use the padding procedure from Equation 22 in subsection 5.7 where $P_{bgd}$ is the background model. We set $C = 8$ so that we train an eight component mixture. We denote the learned template models as $\{P_c\}_{c=0}^{7}$. We construct detectors using log-likelihood ratio tests between sound models $\{P_c\}_{c=0}^{C-1}$ and the background model $P_{bgd}$. Namely, given query features $X \in \{0, 1\}^{L \times F \times D}$ where $L$ is the time length of the sound models $\{P_c\}_{c=0}^{C-1}$, our detector computes the following log ratio:

$$
\Lambda(X) = \log \frac{\prod_{t=0}^{L-1} \prod_{f,e} P_c(t, f, e) E(t, f, e)(1 - P_c(t, f, e))^{1 - E(t, f, e)}}{\prod_{t=0}^{L-1} \prod_{f,e} P_{bgd}(f, e) E(t, f, e)(1 - P_{bgd}(f, e))^{1 - E(t, f, e)}},
$$

where $W_c(t, f, e)$ and $b_c$ are appropriately defined. Hence the log-likelihood ratio detector can be expressed in terms of a linear classifier.

Note that $W_c \in \mathbb{R}^{L \times F \times D}$ and so we have an $L \times F$ array for each of the eight edge features. Figure 32 displays the linear filters learned for the eight latent classes $c$ for one of the frequency edges.
Figure 32: Linear filters \( \{W_c(\cdot,\cdot,2)\}_{c=0}^7 \) for /ar/. Green denotes a zero value. The line in the middle is the weighted mean length of the examples for a given template class. Time is the x-axis and frequency is along the y-axis.

The length clustering phenomenon is evident from Figure 32 as well as the templates for the other edge orientations (not shown). Each of the templates in Figure 32 has a green line running through the middle and we note that only the front segment of the classifier before the line has structure and the rest of the template is zero. We find that this template prefix-suffix phenomena is the same for each edge type and any diphone or phone model we estimate. In practice, if we truncate the templates to the vertical line there is no appreciable difference in the detector outputs.

The reason for why this happens requires us to consider the details of BernoulliMixtureEM. The templates \( P_c \) are computed as weighted averages in line 6 of Algorithm 9. The details are given in Bernoulli-Mstep (Algorithm 11):

\[
P_c = \frac{\sum_{i=0}^{n-1} A(i,c)E_i^{padded}}{\sum_{i=0}^{n-1} A(i,c)}
\]

where \( A(i,c) \) is an estimate of \( \mathbb{P}(z_i = c \mid E_i^{padded}) \) computed by Bernoulli-Estep (Algorithm 10). For each class \( c \) the vertical lines in Figure 32 mark the time-coordinates corresponding to a weighted average of the data lengths \( \{l_i\} \) where we use \( A \), again, for the weights. Formally,

\[
l_c = \frac{\sum_{i=0}^{n-1} A(i,c)l_i}{\sum_{i=0}^{n-1} A(i,c)}
\]

so that data points \( i \) with larger values of \( A(i,c) \) contribute more to the estimate \( l_c \). The sample variance over the lengths is given by

\[
V[l_c] = \frac{\sum_{i=0}^{n-1} A(i,c)(l_c - l_i)^2}{\sum_{i=0}^{n-1} A(i,c)}
\]

and these variances are plotted in Figure 33. We note that the length distribution for the shorter and longer templates has a larger variance than for the middle-length templates. Later we will see the shorter and longer templates perform worse than the middle templates in subsection 6.4.
Despite the considerable variance of the template lengths we note that the template values $W_c(t, f, e)$ are approximately zero for $t > l_c$ in Figure 32, which indicates that

$$P_c(t, f, e) \approx P_{bgd}(f, e)$$

for each $(f, e)$ and $t > l_c$:

$$0 \approx \frac{W_c(t, f, e)}{\log \frac{P_c(t, f, e)}{P_{bgd}(1-P_c(t, f, e))}} = \frac{1-P_{bgd}(f, e)}{1-P_c(t, f, e)}$$

Thus, empirically we see that shorter examples are clustered into one template and longer examples are clustered into other templates. Even though the algorithm BernoulliMixtureEM only learns models over examples with a fixed length, we have succeeded in inferring models with multiple different lengths where the lengths are picked in a data-driven manner. The large variance in the lengths is potentially a problem to address in future work. In this work we assume that the length of the template $W_c$ is approximately $l_c$, and in our experiments we find that for every $t \geq l_c$ we have that $W_c(t, f, e) \approx 0$ for each pair $(f, e)$, and in practice if we truncate $W_c$ to be of length $l_c$ in time then we get the same detection results. So, we truncate both the log-likelihood ratio test template $W_c$ and the probability template $P_c$ so that

$$W_c \in \mathbb{R}^{l_c \times F \times D} \quad P_c \in (0, 1)^{l_c \times F \times d}$$

so that they are constrained to be length $l_c$.

6 Detection

The main experiments considered in this paper are detection experiments. The setting is that we have a collection of training examples of a particular phone, diphone, triphone, or other sound and
we build a detector that, given an unlabeled query utterance \( U \), outputs a list of times \( t_0, \ldots, t_{n_D} \) where \( n_D \) is the number of detections and each time \( t_j \) is a hypothesized occurrence of the sound in \( U \). A time point \( t_j \) is classified as a “hit” or true positive if it is close to an actual occurrence of the sound in \( U \) and it is classified as a “miss” or false positive otherwise. Additionally, if there is some occurrence of the sound in \( U \) at time \( t' \) but there is no \( j \) such that \( |t_j - t'| < \varepsilon \) then the occurrence \( t' \) is called a false negative. Our goal is to build a detector for speech sounds that has a high true positive rate (or “recall” rate), and a low false positive rate. Our approach for doing so will be a cascaded hypothesis test that uses a likelihood-ratio test and a support vector machine-trained linear classifier.

Different speech recognition tasks often will have different costs for false positive mistakes and false negative mistakes. We wish to build a detector suitable for a variety of circumstances. Thus, the output we seek is an ROC curve that quantifies the best recall rate we can achieve given a fixed false positive rate. Formally, the ROC curve is the set of points

\[
\{ (FPR(\lambda), TPR(\lambda)) \mid \lambda \in \mathbb{R} \}
\]

where \( \lambda \) represents a detector threshold, \( FPR \) is the false positive rate (per second), and \( TPR \) is the fraction of objects detected—called the recall or true positive rate. We can define the curve more generally without dependence on an explicit threshold by viewing \( TPR \) and \( FPR \) as functions of a detection algorithm \( \mathcal{A} \). The ROC curve in this setting is formed by finding the algorithm \( \mathcal{A} \) that bounds the false positive rate below a level \( \alpha \) while getting the highest possible recall rate:

\[
\left\{ \left( \alpha, \max_{\mathcal{A} : FPR(\mathcal{A}) \leq \alpha} TPR(\mathcal{A}) \right) \mid \alpha \in [0, 1] \right\}.
\]

As mentioned above, for some applications having a very high recall rate with few false negatives may be more important than minimizing false positives. Such a case corresponds to being interested in the subset of the curve

\[
\left\{ \left( \alpha, \max_{\mathcal{A} : FPR(\mathcal{A}) \leq \alpha} TPR(\mathcal{A}) \right) \mid \alpha \in [1 - \xi, 1] \right\}.
\]

Other applications may require a very low false positive rate while tolerating a moderate number of false negatives. This low false positive rate case corresponds to

\[
\left\{ \left( \alpha, \max_{\mathcal{A} : FPR(\mathcal{A}) \leq \alpha} TPR(\mathcal{A}) \right) \mid \alpha \in [0, \xi] \right\}.
\]

We will discuss quantifying the performance of the algorithms in a later section.

For convenience of the discussion we will talk about sounds as “objects”. These are the acoustic patterns with finite support in time. We were inspired to use this term from [2], but the usage is also common in auditory scene analysis work (see [20] and the references contained therein). The objects we detect will always refer to short sequences of phones in this paper, but the framework is sufficiently general that it should apply to any “acoustic object” and we hope to address that problem in future work.

### 6.1 Cluster-Based Detection

Following the setting in [2, 38] we cast the detection problem as a hypothesis test between object and background. We have a library of object models \( \{P_c\} \subset (\mathbb{R}^{F \times D})^k \) indexed by \( \mathcal{C} \) with lengths
such that \( P_c \in \mathbb{R}^{L \times F \times D} \), an utterance with features \( E \in \{0,1\}^{T \times F \times D} \), and a background model \( P_{bgd} \in \mathbb{R}^{F \times D} \). \( T \) is the length of the utterance and the vector \( E(t) = E(t,:) \in \{0,1\}^{F \times D} \) is called a frame and it consists in the edge responses at time \( t \). We shorten \( E(t,:) \) to \( E(t) \). 

\( E(t_0 : t_1) \in \{0,1\}^{(t_1-t_0) \times F \times D} \) denotes a “slice”, which is the concatenation of every frame \( E(t) \) such that \( t_0 \leq t < t_1 \) for times \( t_0 \) and \( t_1 \) and we note that \( E(t_1 - 1) \) is the last frame of the slice so \( E(t_1) \) is not in the slice \( E(t_0 : t_1) \).

Equation 23, since the suffix of those templates is zero we ignore it. We note that unlike the object models produced by BERNOUILLIMIXTUREEM (Algorithm 9 in subsection 5.3) we do not have weights for the objects \( w_c \). The reason for this is that we assume the frequency with which different objects are observed in the testing set could be quite different from our training data.

In [2] the Amit et al. consider a single object model \( P_0 \) and they perform a test for that object at each time \( t \) with \( 0 \leq t < T \). Their detector computes a test statistic \( \mu(t) \) (in their notation \( \Lambda(t) \)) at each time point and they extract a subset of times that are “candidate detections” \( t \) where \( \mu \) attains a local maximum at \( t \) and \( \mu(t) \) is greater than some threshold. Intuitively, these correspond to those time points where the template matches the signal much better than the background. The authors find that there are generally many candidate detections clustered close together in time. A typical output for \( \mu(t) \) is shown in Figure 35 and one can see that many peaks are close together. The authors in [2] note that peaks cluster together because the detector output is essentially a convolution operation of the features with a smooth template so neighboring detections are highly correlated and tend to be close together. For example, in the task of detecting /ar/ the detector curve \( \mu(t) \) will often take moderately large values at an instance of /\textipa{ar}/, and for a single instance of /\textipa{ar}/ or /\textipa{ar}/ there will often be clusters of multiple false alarms given a detection threshold \( \lambda \). The authors propose, then, to explicitly search for clusters of detections from a candidate list computed from test statistics \( \mu(t) \).

![Figure 34: Raw detection scores for a single object hypothesis at all time locations](image)

We develop a framework for understanding the clustering in terms of a detection cascade. We then extend the detection framework to incorporate SVM-trained discriminative classifiers \( \{\varphi_c\} \).

Before we explain our extension we consider a more precise description of the algorithm sketched in [2]: CLUSTERBASEDDETECTION (Algorithm 15). The output is a list of time intervals \( S \): each corresponding to slices of \( E \) where the algorithm identifies a cluster of peaks in \( \mu(t) \) that exceed a
Input: \( \{ P_c \}_{c=0}^{C-1} \subset (\mathbb{R}^{F \times D})^* \) object models, \( E \in \{0,1\}^{T \times F \times D} \) utterance, \( P_{bgd} \in \mathbb{R}^{F \times D} \) background model, \( K \in \mathbb{Z}_+ \) maximum cluster size, \( \lambda \in \mathbb{R} \) base detector threshold

Output: \( S \in (\mathbb{Z} \times \mathbb{Z})^* \) list of clusters

1: procedure ClusterBasedDetection(\( \{ P_c \}_{c=0}^{C-1}, E, P_{bgd}, K, \lambda \))  
△ Find clusters of detections
2: \( \mu \leftarrow \text{DetectorCurve}(\{ P_c \}_{c=0}^{C-1}, E, P_{bgd}) \)
3: \( \mathcal{V} \leftarrow \text{DetectionCandidates}(\mu, \lambda) \)
4: \( S \leftarrow \text{ClusterCandidates}(\mathcal{V}, K) \)
5: return \( S \)
6: end procedure

Algorithm 15: Cluster-Based Detection Algorithm

We will call these time-intervals “clusters” since they are meant to capture the peak clusters in \( \mu(t) \). The size of the time-intervals or clusters is constrained by the hyper-parameter \( K \), which ensures that the output time-intervals are small enough to be meaningfully considered as detections. A cluster is a true positive if it overlaps significantly with a labeled example and it is a false positive otherwise (we explain the scoring in greater detail in subsubsection 6.1.6). We compute the ROC curve from the true positive rate and false positive rate as a function of the peak threshold \( \lambda \).

ClusterBasedDetection is built on three other algorithms:
1. DetectorCurve (Algorithm 16)
2. DetectionCandidates (Algorithm 17)
3. ClusterCandidates (Algorithm 18)

6.1.1 Detection Curve Computation

The algorithm DetectorCurve outputs a curve \( \mu \in \mathbb{R}^T \) where \( \mu(t) \), for every \( t \), \( \mu(t) \) is a generalized log-likelihood ratio test between these two hypotheses:

**Detector Hypothesis Testing**

\( \mathcal{H}_0 \) Null hypothesis that the utterance \( E \) was generated by the background model \( P_{bgd} \)

\( \mathcal{H}_A \) Alternative hypothesis that for some \( c \in [C] \), \( E(t:t+L_c) \) was generated by object model \( P_c \), and all other frames, \( \{ E(t') \mid t \neq t' \} \) were generated by \( P_{bgd} \)

\( \mu(t) \) is the log-likelihood ratio test comparing the hypothesis of just background against the hypothesis that a single object starts at time \( t \) with background surrounding the object. The likelihood of the data under the background model is
\[
\Pr(E; \theta_{bgd}) = \prod_{\tau=0}^{T-1} \prod_{f,e} P_{bgd}(f, e)^{E(\tau, f, e)} (1 - P_{bgd}(f, e))^{1 - E(\tau, f, e)}. 
\]

For the alternative hypothesis \( \theta(c, t) \), \( E(t : t + l_c) \) was generated by \( P_c \) while \( E(0 : t) \) and \( E(t + l_c : T) \) were generated by \( P_{bgd} \). The likelihood of the data under \( \theta(c, t) \) is

\[
\Pr(E; \theta(c, t)) = \underbrace{\Pr(E(0 : t); \theta_{bgd}) \cdot \Pr(E(t + l_c : T); \theta_{bgd})}_{\text{Background model for off-object data}} 
\underbrace{\prod_{\tau=t}^{t+l_c-1} \prod_{f,e} P_c(\tau - t, f, e)^{E(\tau, f, e)} (1 - P_c(\tau - t, f, e))^{1 - E(\tau, f, e)} }_{\text{model during hypothesized object support}}. 
\]

The form of \( \theta_{bgd} \) and \( \theta(c, t) \) is such that we get the following cancellation, as noted in [2], in the log-likelihood ratio test statistic:

\[
\log \frac{\Pr(E; \theta(c, t))}{\Pr(E; \theta_{bgd})} = \log \frac{\prod_{\tau=t}^{t+l_c-1} \prod_{f,e} P_c(\tau - t, f, e)^{E(\tau, f, e)} (1 - P_c(\tau - t, f, e))^{1 - E(\tau, f, e)} \prod_{\tau=0}^{T-1} \prod_{f,e} P_{bgd}(f, e)^{E(\tau, f, e)} (1 - P_{bgd}(f, e))^{1 - E(\tau, f, e)}}{\prod_{\tau=0}^{T-1} \prod_{f,e} P_{bgd}(f, e)^{E(\tau, f, e)} (1 - P_{bgd}(f, e))^{1 - E(\tau, f, e)}} 
= \sum_{\tau=t}^{t+l_c-1} \sum_{f,e} \log \frac{P_c(\tau - t, f, e)}{P_{bgd}(f, e)} (1 - P_{bgd}(f, e)) E(\tau, f, e) 
+ \sum_{\tau'=0}^{t+l_c-1} \sum_{f,e} \log \frac{1 - P_c(\tau, f, e)}{1 - P_{bgd}(f, e)} 
= b_c + \sum_{\tau=t}^{t+l_c-1} \sum_{f,e} W_c(\tau - t, f, e) E(\tau, f, e) 
\]

noting that \( W_c \) and \( b_c \) match the implicit definitions from [Equation 23] and they are independent of the choice of time \( t \) for hypothesis \( \theta(c, t) \). In the case of a single object, then, the response curve is determined by the formula

\[
\mu(t) = b_c + \sum_{\tau=t}^{t+l_c-1} \sum_{f,e} W_c(\tau - t, f, e) E(\tau, f, e) 
\]

which shows that computing \( \mu(t) \) for all times \( t \) is equivalent to a convolution of \( E \) with a linear filter. In the case where there are multiple objects we use a generalized likelihood ratio test and compare the background against the most likely alternative. Using the monotonicity of the logarithm function,

\[
\mu(t) = \log \max_c \frac{\Pr(E; \theta(c, t))}{\Pr(E; \theta_{bgd})} 
= \max_c \log \frac{\Pr(E; \theta(c, t))}{\Pr(E; \theta_{bgd})} 
= \max_c b_c + \langle W_c, E(t : t + l_c) \rangle 
= \max_c \mu_c(t) 
\]

56
where $ \mu_c(t) $ is the object $ c $ log-likelihood ratio test statistic, i.e. the test of $ \theta(c, t) $ against $ H_0 $. DETECTOR\textsc{Curve} (Algorithm 16) uses this linear-filtering formulation of hypothesis testing to compute $ \mu(t) $.

<table>
<thead>
<tr>
<th>Input: $ {P_c}<em>{c=0}^{C-1} \subset (\mathbb{R}^F \times \mathbb{D})^* $ object models, $ E \in {0, 1}^{T \times F \times D} $ utterance, $ P</em>{bgd} \in \mathbb{R}^F \times \mathbb{D} $ background model</th>
</tr>
</thead>
<tbody>
<tr>
<td>Output: $ S \in (\mathbb{Z} \times \mathbb{Z})^* $ list of clusters</td>
</tr>
</tbody>
</table>

1: procedure DETECTOR\textsc{Curve}(\{P_c\}_{c=0}^{C-1} , E, P_{bgd}) \> Perform GLRT at every time point
2: for $ c = 0, \ldots, C - 1 $ do
3: $ l_c \leftarrow \text{length}(P_c) $ \> Compute the bias term
4: for all $ t \in [l_c], f \in F, e \in [D] $ do
5: \> Construct the linear filter
6: $ W_c(t, f, e) \leftarrow \log \frac{P_c(t, f, e)(1 - P_{bgd}(f, e))}{P_{bgd}(f, e)(1 - P_c(t, f, e))} $ \> Perform GLRT
7: $ b_c \leftarrow \sum_{t, f, e} \log \frac{1 - P_c}{1 - P_{bgd}} $ \> Compute the bias term
8: end for
9: for $ t = 0, \ldots, T - 1 $ do
10: $ \mu(t) = \max_c b_c + \langle W_c, E(t : t + l_c) \rangle $ \> Perform GLRT
11: end for
12: return $ \mu $ \> Perform GLRT
13: end procedure

Algorithm 16: Detector Curve Algorithm

6.1.2 Multi-Object Configurations

In order to motivate the combinatorial algorithms DETECTION\textsc{Candidates} (Algorithm 17) and CLUSTER\textsc{Candidates} (Algorithm 18) we consider the problem of testing multiple-object hypotheses against background. The simplest formulation of a multi-object hypothesis would be a set

$$ \Theta = \{(c_0, t_0), \ldots, (c_{m-1}, t_{m-1})\} \in (\mathbb{C} \times \mathbb{T})^* $$

where we index $ (c_i, t_i) $ such that $ t_i \leq t_{i+1} $. Here $ c_i \in \mathbb{C} $ is the object identity and $ t_i \in \mathbb{T} $ is the start time in $ E $ for the object support.

We extend the model described by Equation 31 to the case of multiple objects. Consider the case where $ \theta \in \Theta $ is a multi-object hypothesis with no overlapping objects. Non-overlapping means that

$$ t_i + l_{c_i} < t_{i+1} $$

for every object $ i $ in $ \theta $. Associated with each object $ i $ in the hypothesis $ \theta $ is a start time $ t_i $ and a support length $ l_{c_i} $ so the interval $ [t_i : t_i + l_{c_i} - 1] $ is the hypothesized support. The simple extension is to model each support interval $ i $ by its object’s probability model $ P_{c_i} $ and define the likelihood of
all off-support frames in $E$ by the background model. If we let $t_m = T$ then

$$P(E; \theta) = P(E(0 : t_0); \theta_{bgd}) \prod_{i=0}^{m-1} \prod_{\tau = t_i}^{t_i + l_{c_i} - 1} \prod_{f,e} P_c(\tau - t_i, f, e) E(\tau, f, e) \left( 1 - P_c(\tau - t_i, f, e) \right)^{1 - E(\tau, f, e)}$$

which is well-defined because $t_i + l_{c_i} < t_i + l_{c_{i+1}}$ (although we may drop the first or last background model term depending on whether there is an object at the very beginning or very end of the utterance). Under this model we get a similar cancellation phenomenon as in Equation 32.

$$\log \frac{P(E; \theta)}{P(E; \theta_{bgd})} = \sum_{i=0}^{m-1} b_{c_i} + \sum_{\tau = t_i}^{t_i + l_{c_i} - 1} \sum_{f,e} W_{c_i}(\tau - t_i, f, e) E(\tau, f, e)$$

which shows that the more complex multi-object configuration can be computed just from using $\mu$.

A difficulty arises in the case where objects overlap. The overlap implies there is some object $i$ with $t_i + l_{c_i} \geq t_i + l_{c_{i+1}}$ which means that the frame $E(t_{i+1})$ would appear in multiple times if we naïvely use that formulation of the probability model. $E(t_{i+1})$ would appear in the term with $P_{c_i}$ and $P_{c_{i+1}}$.

We could specify that one model occludes the other model so that the support of the occluded model is reduced. Another approach is to model the region of the overlapping supports as generated by a mixture of the two models as in [3]. For the purposes of this paper we do not consider these and we make the assumption that the objects we are detecting are unlikely to appear adjacent to each other in time: namely, they will generally be components of a mixture distribution over a particular phone or diphone. The diphones we consider are not repeated in our data so we formulate the following assumption:

**Assumption**: The object supports are always disjoint

With this assumption in place we can now construct a simple procedure that uses the DETECTORCURVE (Algorithm 16) output $\mu$ to infer likely configurations of multiple objects.

### 6.1.3 Detection Candidates Computation

Following the procedure outlined in [2] the next stage is to compute a preliminary subset of potential locations.

A simple algorithm for selecting candidates is to use the set

$$V_{\lambda} = \{ \tau \mid \mu(\tau) \geq \lambda \}$$

where $\lambda$ is threshold chosen from the range of $\mu$. This simple rule has the defect that there is a strong dependence on the response $\mu(t)$ with respect to time. If the utterance contains an object sufficiently similar to the object we are detecting $\mu$ is very likely to take large values in a neighborhood of the object location. This happens because the linear filters $W_c$ are smooth as a function of $t$ and, as is
visible from the edge-map representations, the binary features are highly correlated in time (due to the presence of things like formants). Hence the convolution between the linear filters and the binary features will be a smooth function of time. This means that if \( \mu(t) > \lambda \) then it is very likely that \( \mu(t + 1), \mu(t - 1) > \lambda \) as is evidenced from the response curve in Figure 35.

Thus, we only consider detections that are a local maximum to remove some of these spurious detections. \textsc{DetectionCandidates} (Algorithm 17) finds local maxima in the detector response curve that exceed the input threshold \( \lambda \).

Line 2 restricts the set of detection scores to only those that are local maxima. In line 4 we construct a set \( V \) of detection candidates which is the set of times \( t \) such that \( \mu(t) > \lambda \) and \( \mu(t) \) is a local maximum.

\textsc{DetectorCandidates} returns the pairings of thresholds and sets of candidates.

### 6.1.4 Candidate Clustering Theory

Empirically, for the reasons mentioned in subsection 6.1, given a threshold \( \lambda \) with a high true positive rate there will be clusters of peaks near any speech sound in an utterance sufficiently similar to one of the object templates \( \{P_c\} \). Since each peak corresponds to a separate detection hypothesis if the cluster is not near a ground truth labeled example then each peak in the cluster will could as a false positive, but, intuitively, they should count for only one mistake.

One way to handle these clusters is to use a larger window for the local-max criterion when running \textsc{DetectionCandidates}. This is implemented by only accepting \( t \) as a detection if \( \mu(t + \delta) \leq \mu(t) \) if \( |\delta| < L_{\text{window}} \) where \( L_{\text{window}} > 1 \). However we find that this tends to reduce performance because the larger window means we have less information about the location of clusters. Spurious peaks at the boundary may dominate. This is an issue with a likelihood-ratio based detector because the detector is merely testing the syllable against background. Syllables that are very similar to the one we are detecting will often fire just as strongly as positive examples for the likelihood-ratio.
**Input:** $\mu \in \mathbb{R}^T$ detector curve, $\lambda \in \mathbb{R}$ detector threshold.
**Output:** $V \in \mathbb{Z}^*$ list of candidate detections

1: procedure DetectionCandidates($\mu, \lambda$) \Comment*{Get potential detection points}
2: $V_{\text{potential}} \leftarrow \{t \mid \mu(t) \geq \max\{\mu(t-1), \mu(t+1)\}\}$
3: for $j = 0, \ldots, J - 1$ do
4: $V \leftarrow \{t \mid \mu(t) \geq \lambda\} \cap V_{\text{potential}}$
5: end for
6: return $V$
7: end procedure

**Algorithm 17: Detection Candidate Finding Algorithm**

detector. This phenomenon is particularly prevalent when we set the threshold very low so that our recall rate is very high but the number of false positives is also large. Such a setting is of particular interest in developing a cascaded detector as we consider in subsection 6.2.

The authors in [2] propose an approach for clustering the candidate detections in $V$. The idea is that if there is a cluster of detections in DetectionCandidates near a particular time the clustering algorithm should output an interval that groups those detection candidates together. The algorithm ClusterCandidates takes as input the detection candidates $V$ and outputs a set of intervals, or, more precisely, outputs contiguous sequences of detections in $V$ that are sufficiently close to have been identified as a cluster. During scoring each of these output intervals is counted as a true-hit or a false positive and by changing the initial threshold passed to DetectorCurve we can generate an ROC curve.

ClusterCandidates (Algorithm 18) attempts to output a covering interval for each cluster of detections. Note that the outputs of ClusterCandidates are called “intervals” to distinguish them from the “clusters” of detection points that DetectionCandidates produces. The distinction is that an interval refers to something in the algorithm output, whereas a cluster is an indirectly observed characteristic of the dataset.

Our intuition is that each cluster corresponds to multiple peaks in the detector on the same underlying object in the data (be it a realization of the target object or a realization of a sufficiently similar object). We consider a statistical model of the detector output $\mu(t)$. Let $e = (e_0, e_1, \ldots, e_{T-1}) \in \{0, 1\}^T$ (35) be a sparse binary vector where the proportion of non-zero entries close to zero and with those non-zero entries spaced close to uniformly throughout the vector. We call $e$ our detection vector and it corresponds to the output of an idealized version DetectionCandidates that did not suffer from the peak clustering problem so that $e$ is sparse and its nonzero entries are well-spaced. We also have two sets of Gaussian random variables:

$g_{0, \mu_{\text{low}}}, g_{1, \mu_{\text{low}}}, \ldots, g_{T-1, \mu_{\text{low}}} \sim \mathcal{N}(\mu_{\text{low}}, \sigma)$

and

$g_{0, \mu_{\text{high}}}, g_{1, \mu_{\text{high}}}, \ldots, g_{T-1, \mu_{\text{high}}} \sim \mathcal{N}(\mu_{\text{high}}, \sigma)$,
which are independent and $\mu_{\text{low}} << \mu_{\text{high}}$. These random variables approximate the distribution of weighted sums of Bernoulli random variables (where the weights have bounded magnitude), i.e., the distribution of the cross-correlation between a template linear filter and the binary features of an utterance. The random variables indexed by “low” represent the distribution of the filter applied to non-object data and the random variables indexed by “high” represent the distribution of the filtering on similar-to-object data. We propose the following distribution for the detector output:

$$\mu(t) \sim g_t^{\text{low}} + \sum_{\tau=0}^{T-1} \kappa(t - \tau) g_{\tau}^{\text{high}} e_{\tau}$$

where $\kappa$ is some non-negative kernel whose bandwidth models the “smoothness” of the filter and binary acoustic features (since a larger bandwidth corresponds to greater contamination across time). Under this model, if there is a detection at time $t$ (i.e. $e_t = 1$) then $\mu(t)$ will likely have a value close to $\mu_{\text{low}} + \mu_{\text{high}}$ and so will $\mu(t \pm \delta)$ for $\delta$ comparable to the bandwidth of $\kappa$. In this setting we can think of the binary vector $(e_t)_{t=0}^{T-1}$ as representing where the detection cluster centers are since there will be many peaks above a threshold in a neighborhood of $t$ whenever $e_t = 1$. Running DetectionCandidates on a sequence $\mu$ generated in the above fashion will generate many peaks above a threshold centered near times $t$ where $e_t = 1$. Our task is a recovery of the $e_t$S from the noisy $\mu(t)$ output. Since $\kappa$ controls the time distance over which a single detection $e_t$ can contaminate the scores in $\mu$ our choice for the length of the intervals we consider should be guided by a guess of the bandwidth of $\kappa$.

Recall from DetectorCurve (Algorithm 16) that the detector curve computes local tests over object-length slices of the utterance features $E$. Our templates are often smooth in time (as is $E$ during vowels) so

$$\mu(t) \langle W_c, E(t : t + l_c) \rangle$$

can be very similar to

$$\mu(t + 1) = \langle W_c, E(t + 1 : t + l_c + 1) \rangle.$$ 

More generally, an two tests on overlapping segments of $E$ are correlated so a natural pick for the bandwidth of $\kappa$ is the object length. Additionally, if we assume that associated with a list of detection candidates there is an underlying set of “true clusters”, the cluster identification algorithm ClusterCandidates should avoid the following two types of mistakes:

1. Producing an interval that overlaps with multiple, distinct true clusters
2. Having two separate intervals whose union actually is a single true cluster

We can minimize the incidence of the first problem by ensuring the last point of a hypothesized cluster (one of the intervals output by ClusterCandidates) is far from the first point of the next hypothesized cluster. In particular, that distance should be at least the bandwidth of $\kappa$ or the object length. The situation we avoid with this requirement is where we have a hypothesized cluster $[t, t + 2l - 2]$ where $l$ is the object length and the candidate detection times are such that

$$\{t, t + l - 1, t + 2l - 2\} = \mathcal{V} \cap [t, t + l - 1]$$

so the interval has no detection candidates in $\mathcal{V}$ aside from at the end points. Th interval $[t, t + 2l - 2]$ probably does not correspond to a true cluster because a true cluster would probably have more detections than just the three points. The detections in $\mathcal{V}$ are unlikely if the interval were actually
close to a true cluster hence the algorithm should have a preference for splitting clusters that have points far away from each other into multiple clusters.

We can minimize the second problem by leaving adjacent nodes in the same cluster if those two nodes are close. An extreme case is where the candidate detection points are
\[ V = \{t, t + 1, t + 2, t + 3\}. \]
In this case it would be prudent to have all the detection points in a single interval and it would be absurd to split them into multiple hypothesized clusters.

Thus we have three main requirements for the hypothesized cluster intervals output by \textsc{ClusterCandidates}:

1. Interval lengths should not exceed the object length
2. Candidate detections in an interval should be close together
3. Candidate detections in different intervals should be far apart

### 6.1.5 Candidate Clustering Algorithm

We now present a version of the algorithm (slightly different than the original formulation in [2]) that attempts to achieve those three goals from \textit{Item 6.1.4}. We present \textsc{ClusterCandidates} in \textbf{Algorithm 18}. We change notation slightly and refer to the intervals of the previous section as “clusters”. Clusters here refer to the intersection of an interval \( I \subset [T] \) with a set of candidate detection times \( V \). The length of a cluster \( I \cap V \) is the length of the smallest interval \( I_{\min} \) such that
\[ I_{\min} \cap V = I \cap V. \]

\begin{algorithm}
\begin{enumerate}
  \item \textbf{procedure} \textsc{ClusterCandidates}(\( V, K \)) \hfill \triangleright \text{Find an interval covering of } V
  \item \{\( I_j \)\}_{j=0}^{J-1} \leftarrow \textsc{InitClusters}(V, K) \hfill \triangleright \text{Find an interval covering of } V
  \item \( j_{\text{long}}, I_{j_{\text{long}}} \leftarrow \textsc{LongestCluster}(\{I_j\}_{j=0}^{J-1}) \hfill \triangleright \text{Find the longest cluster}
  \item \textbf{while} \( l_j > K \) \textbf{do}
    \item \( I_0^{\text{new}}, I_1^{\text{new}} \leftarrow \textsc{SplitCluster}(I_{j_{\text{long}}}) \)
    \item \( J \leftarrow J + 1 \)
    \item \{\( I_j \)\}_{j=0}^{J-1} \leftarrow \{I_0^{\text{new}}, I_1^{\text{new}}\} \cup \{I_j\}_{j=0}^{J-2} \setminus \{I_{j_{\text{long}}}\} \)
    \item \( j_{\text{long}}, I_{j_{\text{long}}} \leftarrow \textsc{LongestCluster}(\{I_j\}_{j=0}^{J-1}) \)
  \item \textbf{end while}
  \item \textbf{return} \{\( I_j \)\}_{j=0}^{J-1}
\end{enumerate}
\end{algorithm}

\textbf{Algorithm 18: Candidate Clustering Algorithm}

This procedure relies on the procedures \textsc{InitClusters} (described in \textbf{Algorithm 19}), \textsc{LongestCluster} which simply finds the cluster that spans the longest set of times, and \textsc{SplitCluster}. 

62
Input: \( V \subset [T] \) candidate detection times \( K \in \mathbb{Z}_+ \) maximum interval length (usually object length)
Output: \( \{I_j\}_{j=0}^{Z-1} \subset 2^V \) collection of intervals over \( V \)

1: procedure \textsc{InitClusters}(\{v_z\}_{z=0}^{Z-1}, K)  
\hspace{1em} \triangleright \text{Find an initial cluster covering of } V
2: \hspace{1em} \mathcal{I} \leftarrow \emptyset 
3: \hspace{1em} I \leftarrow \emptyset 
4: \hspace{1em} w \leftarrow v_0 
5: \hspace{1em} \text{for } z = 0, \ldots, Z - 1 \text{ do}
6: \hspace{2em} \text{if } v_z - w > K \text{ then}  
7: \hspace{3em} \mathcal{I} \leftarrow \mathcal{I} \cup \{I\} 
8: \hspace{3em} I \leftarrow \{v_z\} 
9: \hspace{2em} \text{else} 
10: \hspace{3em} I \leftarrow I \cup \{v_z\} 
11: \hspace{2em} \text{end if}
12: \hspace{1em} \text{end for}
13: \hspace{1em} \text{return } I 
14: \text{end procedure}

Algorithm 19: Candidate Clustering Initialization Algorithm

\textsc{SplitCluster} takes in a set of detection times \( (v_0, v_1, \ldots, v_p-1) \in [T]^p \), orders them, and splits them at the largest jump. Assuming a monotonic with respect to indices, the algorithm finds the index \( q \) such that \( v_q - v_{q-1} \) is maximal (the largest jump) and outputs the two intervals \( \{v_0, \ldots, v_{q-1}\} \) and \( \{v_q, \ldots, v_{p-1}\} \).

\textsc{InitClusters}, shown in Algorithm 19, works by finding the minimal linear partition of \( V \) such that \( K \) is greater than the distance between adjacent nodes in the clusters formed by the partition. \textsc{InitClusters} maintains a collection of clusters \( \mathcal{I} \) and a working cluster \( I \), both of which are initialized to be empty. In Algorithm 5, we loop through each candidate detection in time order and check the distance between the current candidate detection and the previous candidate detection. If the distance is less than the object length \( K \) the current candidate is placed in the working cluster, otherwise a jump is identified (since they are far apart) and the current working cluster is saved to the finished clusters and we initialize new working cluster with the current detection as its sole element.

The output of \textsc{InitClusters} gives an initial clustering of the candidate detection times. \textsc{ClusterCandidates} iteratively splits clusters that exceed the object length using \textsc{SplitCluster} until every cluster is no longer than the object length \( K \).

6.1.6 Cluster Performance Evaluation

The outputs of \textsc{ClusterBasedDetection} (Algorithm 15) are sets of clusters. \textsc{ClusterEvaluation} (Algorithm 20) evaluates a set of clusters based on how many ground-truth objects they
overlap with and how many clusters do not overlap with any ground-truth object. Let \( E \) be the utterance over which we have clustered the detection candidates. The ground-truth labels form a set \( \{(t_i^{gt}, l_i^{gt})\}_{i=0}^{I-1} \) where \( I \) is the number of objects, \( t_i^{gt} \) is the object length, and \( E(t_i^{gt} : t_i^{gt} + l_i) \) are the utterance features contained in the support of object \( i \). For each object \( i \) appearing in the utterance we associate a label-window

\[
G_i = [t_i^{gt} - l_i^{gt}/3, t_i^{gt} + l_i^{gt}/3]
\]

which we assume should contain all potential detections of the object. A cluster is considered a true hit for object \( i \) if it overlaps with at least half of the label-window. By design this means that at most one cluster can be classified as a true hit for \( i \). A cluster is a false positive if it is not a true hit. We can now compute a true positive rate and false positive rate given a list of clusters and ground truth labels. By varying the threshold \( \lambda \) in Algorithm 17 we can produce different sets of clusters and hence different true/false positive rates thereby forming an ROC curve.

In line 8 a procedure OVERLAP is invoked that simply checks whether two intervals have an overlap support greater than a particular length. This criterion checks to make sure that a given cluster only counts as a true detection if it overlaps sufficiently with the label-window of a ground-truth object.

6.2 Cascaded Detection

We propose to improve the detection results achieved by CLUSTERBASEDDETECTION by integrating a second classifier to form a cascade. We apply this second classifier to predict which of the clusters output by CLUSTERBASEDDETECTION are false positives and which clusters are true positives. Our strategy is to use a very low threshold \( \lambda \) in DETECTIONCANDIDATES [Algorithm 17] so that the set of clusters output by CLUSTERCANDIDATES contains nearly all of the ground truth objects. However, the lower threshold also means that the cluster set contains many false positives. These clusters \( S \) are then run through CLUSTERCLASSIFICATION [Algorithm 21] along with a set of classifiers \( \{\varphi_c\}_{c=0}^{C-1} \) (one classifier for each latent class of the object) and a threshold \( \zeta \).

```python
1: procedure CLUSTERCLASSIFICATION(\( S \), \( \{\varphi_c\}_{c=0}^{C-1}, \zeta \)) \( \triangleright \) Separate hits from false alarms
2: \( \mathcal{H} \leftarrow \emptyset \)
3: for \( S \in \mathcal{S} \) do
4: \( t, c, E_{\text{slice}} \leftarrow \text{GetMaxDetection}(S) \)
5: if \( \varphi_c(E_{\text{slice}}) \geq \zeta \) then \( \mathcal{H} \leftarrow \mathcal{H} \cup \{t\} \)
6: end if
7: end for
8: return \( \mathcal{H} \)
9: end procedure
```

Algorithm 21: Cluster Classification Algorithm

64
Input: $S \in \mathbb{R}^{T}$ detection clusters, $\{(t^{gt}_{i},l^{gt}_{i})\}_{i=0}^{I-1}$ ground truth labels, $T \in \mathbb{Z}$ number of time points.

Output: $TPR \in \mathbb{Z}$ true positive rate, $FPR \in \mathbb{Z}$ false positive rate.

1: procedure ClusterEvaluation($S, \{(t^{gt}_{i},l^{gt}_{i})\}_{i=0}^{I-1}, T$) \hspace{1em} \triangleright Separate hits from false alarms
2: \hspace{1em} $FP \leftarrow 0$ \hspace{1em} \triangleright counter for false positives
3: \hspace{1em} $TP \leftarrow 0$ \hspace{1em} \triangleright counter for true positives
4: \hspace{2em} for $S \in S$ do
5: \hspace{3em} $isFP \leftarrow True$ \hspace{1em} \triangleright Default is that a cluster is a false positive
6: \hspace{3em} for $i = 0, \ldots, I - 1$ do
7: \hspace{4em} $G_i \leftarrow [t^{gt}_{i} - l^{gt}_{i}/3, t^{gt}_{i} + l^{gt}_{i}/3]$ \hspace{1em} \triangleright Ground truth label-window
8: \hspace{4em} if OVERLAPS($S, G_i, l^{gt}_{i}/3$) then
9: \hspace{5em} $isFP \leftarrow False$
10: \hspace{4em} end if
11: \hspace{3em} end for
12: \hspace{2em} if $isFP$ then
13: \hspace{3em} $FP \leftarrow FP + 1$
14: \hspace{2em} else
15: \hspace{3em} $TP \leftarrow TP + 1$
16: \hspace{2em} end if
17: \hspace{2em} end for
18: \hspace{1em} return $TP, FP$
19: end procedure

Algorithm 20: Cluster Evaluation Algorithm
ClusterClassification takes as input a list of cluster $S$ output by ClusterCandidates (Algorithm 18) and a set of classifiers $\{\phi_c\}$. For each cluster $S \in S$ ClusterClassification predicts whether $S$ is a true positive cluster or a false positive cluster. Algorithm 4 calls a procedure GetMaxDetection for each cluster $S \in S$ extracts a time $t$, an object class $c$, and a slice $E_{\text{slice}} \in \{0, 1\}^{l_c \times F \times D}$ of the utterance $E$. The cluster set $S$ output by ClusterCandidates, as described in Algorithm 18 does not include information about the utterance or the latent class or even the candidate detections, however, in practice, we keep track of these data, so that GetMaxDetection can find the time $t$ in the cluster where $\mu(t')$ is maximal

$$t = \arg \max_{t' \in S} \mu(t')$$

and $c$—the maximal class at time $t$:

$$c = \arg \max_{c' \in [C]} \mu_{c'}(t).$$

We interpret $t$ as an estimate of the latent “true location” of the object associated with the cluster and $c$ is the latent object class. In the setting described in subsection 6.1.4 time $t$ is our best point estimate for a non-zero entry in the vector $e$ defined in Equation 35.

### 6.2.1 Cascaded Detection Performance Evaluation

Associated with the latent object class $c$ we have a classifier $\varphi_c$. To each cluster in $S$ we have a slice $E(t : t + l_c)$, which are the features in the hypothesized object’s support. We predict the cluster is a true positive if $\varphi_c(E(t : t + l_c))$ exceeds the threshold $\zeta$. Those time points that are classified as exceeding the threshold are the predicted set of times for the object. Due to the fact that there is at most one detection point output per cluster these output points are sparse and well-separated in time. Evaluation is performed by checking which among those output points are within the label-window $G$ of a ground truth object. We then compute a true positive rate (fraction of ground truth objects with a hit) and a false positive rate (number of proposed hits not near any object per unit time) so by varying the threshold $\zeta$ we can generate an ROC curve.

### 6.2.2 Full Cascaded Detection Algorithm

With these algorithms we can now formulate CompleteCascadedDetection (Algorithm 22) which takes as input the utterance $E$, ground-truth labels $\{t^g_i, l^g_i\}_{i=0}^{I-1}$, object models $\{P_c\}_{c=0}^{C-1}$, background model $P_{\text{bgd}}$, cluster classifiers $\{\varphi_c\}_{c=0}^{C-1}$, a maximum cluster-size hyperparameter $K$, first cascade layer detection threshold $\lambda$, and a final output threshold $\zeta$. The output is a recall rate and false positives per second rate. By running this algorithm with many different thresholds $\{\zeta_j\}_{j=0}^{J-1}$ we can construct an ROC curve and investigate the performance.

66
6.3 Training

In order to run \textsc{CompleteCascadedDetection} \textbf{[Algorithm 22]} we must estimate a collection of templates \( \{ P_c \}_{c=0}^{C-1} \), a background model \( P_b \), a base threshold \( \lambda \), cluster thresholds \( \{ \zeta_j \}_{j=0}^{J-1} \), and cluster classifiers \( \{ \varphi_c \}_{c=0}^{C-1} \). Training the object models \( \{ P_c \}_{c=0}^{C-1} \) and the background model \( P_b \) has already been discussed. We have to consider how to set the thresholds and how to train the cluster classifiers.

We have a training data set \( \{ E_i \}_{i=0}^{n-1} \) and using the models learned by \textsc{BernoulliMixtureEM} we construct linear classifiers
\[
\{(W_c, b_c)\}_{c=0}^{C-1}
\]
which perform log-likelihood ratio tests as discussed in \textbf{subsubsection 6.1.1}. Each data point \( E_i \) is a slice from an utterance \( E_{u_i}(t_{i}^{gt} : t_{i}^{gt} + l_{i}^{gt}) \) and has a label window
\[
G_i = [t_{i}^{gt} - l_{i}^{gt}/3, t_{i}^{gt} + l_{i}^{gt}/3].
\]

The threshold \( \lambda \) is derived from the maximal responses of a detector curve \( \mu \) over the label windows. Recall that \( \mu \) is computed with \textsc{DetectorCurve} \textbf{[Algorithm 16]} so for each ground truth object \( i \) define
\[
\lambda_i = \max_{t \in G_i} \mu(t)
\]

The base threshold \( \lambda \) is chosen to be a small percentile of the set \( \{ \lambda_i \}_{i=0}^{n-1} \) since we want the the base threshold to register peaks near most of the ground truth objects in the training set. We generally choose \( \lambda \) to be the third percentile in our experiments. In future work \( \lambda \) will be tuned on a development set.

\begin{algorithm}
\begin{algorithmic}[1]
\State \textbf{procedure} \textsc{CompleteCascadedDetection} \((\{ P_c \}_{c=0}^{C-1}, E, P_b, K, \lambda, \{ \varphi_c \}_{c=0}^{C-1}, \{ (t_{i}^{gt}, l_{i}^{gt}) \}_{i=0}^{I-1}, \{ \zeta_j \}_{j=0}^{J-1} )\) \Comment{Algorithm 15}
\State \( S \leftarrow \text{ClusterBasedDetection}(\{ P_c \}_{c=0}^{C-1}, E, P_b, K, \lambda) \) \Comment{ROC curve}
\State \( H_j \leftarrow \text{ClusterClassification}(S, \{ \varphi_c \}_{c=0}^{C-1}, \zeta_j) \) \Comment{Algorithm 21}
\State \( TPR_j, FPR_j \leftarrow \text{DetectionEvaluation}(H, \{ (t_{i}^{gt}, l_{i}^{gt}) \}_{i=0}^{I-1}) \) \Comment{subsubsection 6.2.1}
\State \( \mathcal{R} \leftarrow \mathcal{R} \cup \{ (TPR_j, FPR_j) \} \)
\State \textbf{end for}
\State \textbf{return} \( \mathcal{R} \)
\State \textbf{end procedure}
\end{algorithmic}
\end{algorithm}
Using the chosen \( \lambda \), we then run \textsc{DetectionCandidates} (Algorithm 17) over the detector curve \( \mu \) to output a candidate list of detections, which are then clustered with \textsc{ClusterCandidates} (Algorithm 18) producing a set of clusters \( S \). From each cluster \( S \in \mathcal{S} \) we have a time \( t_S \) and a latent class \( c_S \) such that \( \mu \) is maximal over the cluster at \( \mu(c,t) \)

\[
c_S, t_S = \arg \max_{t' \in S, c' \in C} \mu(c', t').
\]

\( E_S = E(t : t + l_c) \) is the slice of the utterance features \( E \) associated with the maximal response. If \( S \) is a true positive cluster then the example \( E_S \) will be close to a labeled example of the object we are detecting, although it will not necessarily be at the exact location. By taking the maximum response and not the true labeled location we handle some of the noise involved in labeling and segmenting a speech data set. Alternatively, if \( E_S \) is a false positive cluster then \( E_S \) will not be close to any ground truth instance of the object. These two groups of slices then present a binary discrimination problem.

We divide the clusters \( S \) into true positives \( S_{\text{true}} \) and false positives \( S_{\text{false}} \). For each class \( c \in [C] \) we can construct subsets \( S_{c_{\text{true}}} \) and \( S_{c_{\text{false}}} \) which correspond to those clusters \( S \) such that \( c_S = c \). Then we train a linear classifier \( \varphi_c \) using the support vector machine algorithm \[30\] to discriminate between positive examples \( \{ E_S : S \in S_{c_{\text{true}}} \} \) and negative examples \( \{ E_S : S \in S_{c_{\text{false}}} \} \) for class \( c \). We retain the scores of the classifiers \( \varphi_c \) on the positive examples and store them as the thresholds \( \{ \zeta_j \}_{j=0}^{J-1} \) where the \( \zeta_j \) combine the scores computed from classifiers \( \varphi_c \) for every class \( c \). Combining the thresholds for the different \( \varphi_c \) potentially hurts performance and more sophisticated strategies are of interest in future work.

6.4 Results

Our purpose in this work is to establish the proposed technique as a direction for research. We are primarily extending results from \[2\] in that we consider a larger number of syllables and multiple templates per syllable. We also compare some of our syllable-spotting techniques to the experiments in \[22\]. Based on the results we discuss below, we surmise:

1. Using multiple templates for a syllable helps
2. The SVM detectors \( \{ \varphi_c \} \) work best for making final detection decisions, while the likelihood ratio test detectors \( (W_c, b_c) \) are better suited for the initial detection decisions in the cascade
3. The longest and shortest templates perform the worst
4. Templates do not need to stretch to detect vowels

We use the TIMIT database \[18\] for our experiments, and we concentrate on the detection of diphones as in \[2\]. TIMIT is a database of sentences in text with the audio of those sentences read aloud. Each sentence in TIMIT is segmented into phonetic labels. The labels used in our experiments are in Table 1. Our detection task is to identify diphones, so we try to detect adjacent pairs of these labels.

In our first set of results we build a detector for /ar/ and vary the signal-processing algorithms used to compute the time-frequency representation. We test Hamming-tapered spectrograms, multitaper spectrograms, and mel spectrograms. For each of these spectrograms, we use \textsc{EdgeMap} (Algorithm 5) to compute the edge representation (see section 4). To run \textsc{EdgeMap} four hyper-parameters must be set: \( \rho \) (an adaptive, relative threshold), \( L_{\text{block}} \) (neighborhood size for thresholding with \( \rho \)), \( \alpha \) (an absolute threshold), and \( \Delta \) (max-pooling neighborhood radius). In \textsc{EdgeThresholding} (Algorithm 7), \( \rho \) was chosen to be 70% for all experiments, and the block size
Table 1: Labels of the phones considered in our experiments. 'bcl', 'tcl', and 'dcl' are the closure symbols for the stops 'b', 't', and 'd', respectively.

\[
\begin{array}{|c|c|c|}
\hline
\text{Symbol} & \text{Example Word} & \text{Possible Phonetic Transcription} \\
\hline
b & bee & \text{bcl b iy} \\
t & tea & \text{tcl t iy} \\
s & sea & \text{s iy} \\
r & ray & \text{r ey} \\
iy & beet & \text{bcl b iy tcl t} \\
ih & bit & \text{bcl b ih tcl t} \\
ae & bat & \text{bcl b ae tcl t} \\
aa & bott & \text{bcl b aa tcl t} \\
ay & bite & \text{bcl b ay tcl t} \\
ux & suit & \text{s ux tcl t} \\
ix & debit & \text{dcl d eh bcl b ix tcl t} \\
eh & bet & \text{bcl b eh tcl t} \\
\hline
\end{array}
\]

\(L\)-block over which we compute the thresholds corresponding to the \(\rho\)-percentile was 40 frames or 215 milliseconds. The absolute thresholds \(\alpha \in \mathbb{R}^8\) specify a different absolute threshold for detecting an edge at different orientations:

\[
\begin{bmatrix}
1 & 0 \\
-1 & 0 \\
0 & 1 \\
0 & -1 \\
1 & 1 \\
-1 & 1 \\
1 & -1 \\
-1 & -1 \\
\end{bmatrix}
\]

\(\alpha = \begin{bmatrix}
.025 \\
.025 \\
.015 \\
.015 \\
.02 \\
.02 \\
.02 \\
.02 \\
\end{bmatrix}
\)

These hyper-parameter settings were chosen on the basis of inspection of the edge-maps and we have not performed rigorous parameter-tuning. The other parameter required by EDGE\textsc{map} is the amount of spreading radius \(\Delta\) used by EDGE\textsc{spreading} (Algorithm 8). We chose the radius to be one in all experiments.

Some of the signal processing parameters were the same across the different experiments: the signals used were sampled at 16kHz and we windowed the signal into frames with 320 samples (corresponding to 20 ms), every 80 samples (corresponding to 5 ms). A 512-point Fourier transform was computed on every segment, and all but the Fourier coefficients corresponding to frequencies less than 3kHz were discarded. When we used GAUSSIAN\textsc{smoothing} (Algorithm 3) the smoothing filter was always 7 pixels long (in time) and/or 7 pixels wide (in frequency) and had a standard deviation of 1.

Since our detector is a cascade composed of two detectors in serial we have to set two thresholds: one for the base likelihood-ratio test detector and one for the SVM-trained detector. In this work we do not employ a careful tuning of these two thresholds. For most of the experiments the likelihood ratio-test detector threshold is set so that the likelihood ratio test detector curve \(\mu\) will have a peak near at least 97% of the examples.
To evaluate the performance of the proposed detection algorithms we use receiver operating characteristic (ROC) curves. Our task is detecting short phone sequences in continuous speech. In the literature this task is closest to the problem of keyword spotting, which is normally evaluated by examining the recall rate where the number of false alarms is fewer than 10 per hour. Since, detecting syllables is inherently harder than words (due to the shorter length of syllables) we plot the region of the ROC curve where the detector makes fewer mistakes than 30 per hour. Additionally, we report a figure-of-merit score which is an area-under-the-curve measurement standard for keyword-spotting systems:

$$FOM(TPR, FPR) = \frac{1}{10} \sum_{m=1}^{10} TPR(\arg \max_{\lambda \in T : FPR(\lambda) \leq i} FPR(\lambda))$$

(37)

where $TPR$ is the recall, $FPR$ is the number of mistakes per hour, and $T$ is a collection of thresholds estimated for the detector. The main complexity in computing the $FOM$ (figure of merit) comes from the fact that we have true positive rates for a discrete subset of all possible false positive rates. Theoretically, the $FOM$ score is computed as an average of the best recall rate when the detector makes 1 mistake, 2 mistakes, \ldots, 10 mistakes per hour. Instead we find the best recall rate when we make no fewer than 1 mistake, 2 mistakes, \ldots, 10 mistakes per hour, and average over those recall rates.

### 6.4.1 /ar/ Experiments

Our detector differs from the one proposed in [2] because we use multi-component Bernoulli mixtures rather than a single template for each syllable and an SVM-trained cascaded classifier. Figure 36 and Table 2 demonstrate the impact of mixtures and the use of a cascade. The spectrogram used was the same as in [2]: MultiTaperSpectrogram (Algorithm 2) followed by GaussianSmoothing (Algorithm 3) where we smooth along the time and frequency axes. We see that mixtures do not make a great deal of difference for the non-cascaded detection algorithm and that using too many mixture components is detrimental. Note that the ROC curve for these baseline detection results is constructed from the thresholds $\{\lambda_j\}$ found by DetectionCandidates (Algorithm 17) on the training data and the error rates are computed using ClusterEvaluation (Algorithm 15) on testing data.

Figure 36 also demonstrates the importance of the $\zeta$ threshold in ClusterClassification (Algorithm 21), which determines the set of clusters that will be deemed candidates for true positives and tested by the SVM classifier. In Figure 36 the “3 thresh” and “25 thresh” labels indicate the respective $\zeta$-thresholds for those detectors—these are the thresholds that determine which candidate detection clusters will be tested by the SVM classifier. For the detector “3 thresh”, a cluster produced by ClusterCandidates (Algorithm 18) is only considered a potential true positive if the likelihood-based detector registers a peak within the cluster above the 3rd percentile among the scores for the ground truth examples in the training set. The detector “25 thresh”, only uses clusters such that the likelihood-based detector registers a peak above the 25th percentile. Using multiple object models (e.g. “cascaded 8 3 thresh” corresponds to using eight object models with a 3rd percentile $\zeta$ threshold) appears be superior to using a single object model (“1”, “cascaded 1 3 thresh”, and “cascaded 1 25 thresh” in Figure 36), and the single object model should use a more conservative $\zeta$ threshold for ClusterClassification as “cascaded 1 3 thresh” performs much worse than “cascaded 1 25 thresh”. Careful tuning of the thresholds will be potentially very important for the performance of these algorithms.
Figure 36: Comparison of different detectors for /ar/. The number refers to the number of mixture components. “cascaded” refers to using the cascaded detection algorithm (Algorithm 22), the other ROC curves are for the baseline no cascade algorithm. “thresh” refers to the threshold false negative rate chosen for the likelihood ratio test detector.

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>FOM</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 mixture, baseline</td>
<td>.29</td>
</tr>
<tr>
<td>4 mixtures, baseline</td>
<td>.32</td>
</tr>
<tr>
<td>6 mixtures, baseline</td>
<td>.29</td>
</tr>
<tr>
<td>8 mixtures, baseline</td>
<td>.64</td>
</tr>
<tr>
<td>1 mixture, SVM, 3% threshold</td>
<td>.02</td>
</tr>
<tr>
<td>1 mixture, SVM, 25% threshold</td>
<td>.50</td>
</tr>
<tr>
<td>4 mixtures, SVM, 3% threshold</td>
<td>.47</td>
</tr>
<tr>
<td>6 mixtures, SVM, 3% threshold</td>
<td>.59</td>
</tr>
<tr>
<td>8 mixtures, SVM</td>
<td>.64</td>
</tr>
</tbody>
</table>

Table 2: Results using the signal processing setup from [2] to detect /ar/. “Baseline” refers to using the system in [2] while “SVM” refers to using Algorithm 22 COMPLETECASCADEDDETECTION.
One possible explanation for the poor performance of the baseline likelihood-ratio-based detector is that the likelihoods computed by the different object models are not comparable because they cover objects of different lengths. Each of the templates defined by an object model collection \( \{ P_c \} \) has an estimate variance \( \hat{l}_c \) and length-variance \( \text{Var}[\hat{l}_c] \). Some of the templates (particularly the longer templates) have large variances \( \text{Var}[\hat{l}_c] \) (Figure 33) which suggests that the likelihoods computed under that template may have a different scale and range than the likelihoods computed by templates that are shorter since their distributions have a different scale. The (aa,r) detector degrades considerably with the longest and shortest templates in Figure 37 and in Table 3, we note that in both those figures we are using a detector over the multitaper spectrogram with no frequency smoothing. The large variation in error statistics for the different component templates suggest that we might improve performance by having different thresholds for the different mixture components, or that some kind of normalization could help.

Figure 37: Comparison of different detectors for /ar/. The number refers to the number of mixture components. “casc.” refers to using the cascaded detection algorithm (Algorithm 22), “freq. smooth” refers to whether we smoothed along the frequency axis of the spectrograms.
<table>
<thead>
<tr>
<th>Component</th>
<th>FOM</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0.48</td>
</tr>
<tr>
<td>1</td>
<td>0.69</td>
</tr>
<tr>
<td>2</td>
<td>0.91</td>
</tr>
<tr>
<td>3</td>
<td>0.99</td>
</tr>
<tr>
<td>4</td>
<td>0.84</td>
</tr>
<tr>
<td>5</td>
<td>0.96</td>
</tr>
<tr>
<td>6</td>
<td>0.77</td>
</tr>
<tr>
<td>7</td>
<td>0.18</td>
</tr>
</tbody>
</table>

Table 3: FOM scores for the component templates of the /ar/ detector.

We can improve recognition performance considerably by omitting the frequency-axis smoothing in \textsc{GaussianSmoothing}, indeed doing so is how we attain the performance in Table 3. We call these features time-only smoothed multitaper features. We show these improved results in Figure 38 and Table 4.

Figure 38: Comparison of different detectors for /ar/. The number refers to the number of mixture components. “casc.” refers to using the cascaded detection algorithm (Algorithm 22). “freq. smooth” refers to whether we smoothed along the frequency axis of the spectrograms. The 1 component cascaded detectors do poorly except “1 casc. no freq smooth thresh 25” where the base detector threshold is set to the 25th percentile in terms of responses on the training ground truth objects (so the false negative rate on the training data is 25%)—every other curve is generated with a 3rd percentile choice for the threshold.
### Table 4: Comparison of the cascaded detectors of /ar/ against non-cascaded detectors where we vary the number of mixture components.

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>FOM</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 mixture, baseline, no smoothing</td>
<td>.30</td>
</tr>
<tr>
<td>4 mixtures, baseline, no smoothing</td>
<td>.39</td>
</tr>
<tr>
<td>6 mixtures, baseline, no smoothing</td>
<td>.32</td>
</tr>
<tr>
<td>8 mixtures, baseline, no smoothing</td>
<td>.33</td>
</tr>
<tr>
<td>1 mixture, SVM, no smoothing</td>
<td>.50</td>
</tr>
<tr>
<td><strong>4 mixtures, SVM, no smoothing</strong></td>
<td><strong>.69</strong></td>
</tr>
<tr>
<td>6 mixtures, SVM, no smoothing</td>
<td>.63</td>
</tr>
<tr>
<td>8 mixtures, SVM, no smoothing</td>
<td>.66</td>
</tr>
</tbody>
</table>

We wish to compare these results against more standard signal processing approaches in the literature. We compare with edge-maps computed on standard spectrograms with a Hamming taper, and mel spectral features (Algorithm 4) against the multitaper, time-only smoothed multitaper features. In Figure 39 comparing them shows that the other features perform much worse.

Figure 39: Comparison of different detectors for /ar/. The number refers to the number of mixture components. “mel” refers to using the a cepstral-smoothed mel spectrogram (Algorithm 4), “no dpss” refers to using a single Hamming taper to compute the spectrogram (Algorithm 1). “no freq. smooth” refers to the time-only smoothed multitaper features.

The performance gap in Figure 39 provides evidence that the multitaper evolutionary spectrum
works better for speech recognition tasks. We suspect the main reason for the performance gap is that the Hamming-taper spectrogram has lower bias and higher variance—in particular, it preserves the harmonics which create many stray edges (see Fig 14, 15, 16, 17, 18, 19, 20, 21) that can have wildly-varying intensities.

Having considered different design parameters for the time-frequency representation, another question to ask is whether the edges are adding something useful. To answer this question we consider two further signal processing algorithms that are based on intensity. One of them is to use straight MFCCs (i.e. a discrete cosine transform applied to log mel spectral features), and the other is to construct an intensity-based binary representation. The intensity-based binary representation uses an absolute and an adaptive threshold to binarize the intensities of the spectrogram by passing the raw spectrogram to the algorithm \textsc{EdgeThresholding} (Algorithm 7) and then spread the binary features in a neighborhood rather than along a gradient. In order to perform detection with MFCCs we use a Gaussian mixture model (with a diagonal covariance matrix) for the templates and a single frame Gaussian background model (again with a diagonal covariance matrix), so the object models consist of independent Gaussian random variables.

The results are displayed in Figure 40. MFCCs perform fairly well but they are still inferior to the binary edge features on a multitaper spectrogram. Magnitude features perform much worse than either of the feature sets.

![Figure 40: Comparison of different detectors for /ar/. The number refers to the number of mixture components. “MFCC” refers to the Gaussian classifier over MFCC features, “mags” refers to using a Bernoulli classifier on thresholded magnitudes. “no freq. smooth” refers to the time-only smoothed multitaper features.](image-url)
We also see that if we add noise to the utterance by mixing in audio from another speaker, we only get mild degradation in performance as shown in Figure 41.

Figure 41: Comparison of /ar/ detectors in utterances corrupted by noise given by another talker. The noise level is given by the base 10 logarithm of the ratio between the energy in the target utterance and the energy in the corrupting utterance.

Another question whether the cascading detectors reduces the errors made. Cascaded detection is generally used to speed up inference, but we find that cascading detectors impact the error probabilities, and that the combination of the likelihood-based detector with the SVM classifier outperforms either detector alone. Previously, we showed that cascades outperform just using the likelihood-based detector, and we now show that the cascade outperforms just using the SVM classifier in Figure 42. The likelihood ratio-SVM cascade “NFS” is compared against a non-cascaded detector (CLUSTERBASEDDETECTION Algorithm 15) (SFNC) that uses the SVM-learned filter instead of the likelihood-ratio test, and the double SVM cascade (DSC) which uses the SVM-learned filter as the base detector for the cascade and also uses the SVM filter in the cascade. We see that although DSC outperforms SFNC, our original cascade NFS is the clear winner. Here, performance is measured by the ROC curve, and we are looking for an ROC curve that for a given false positive rate achieves the lowest false negative rate.

Another question to address is whether using the SVM algorithm to train the classifier in the last step of the cascade is important. In subsection 6.3 we mention that we have a training set of positive and negative examples. Conceivably any number of algorithms could be used. One interesting comparison is with respect to generative and discriminative models. We have given evidence that a cascade with an initial generative classifier and a later discriminative classifier outperforms purely
Figure 42: Comparison of the cascaded detector and variations on the SVM detector different detectors for /ar/. The number refers to the number of mixture components. “NFS” is our standard cascade. “SFNC” is the CLUSTERBASEDDETECTION algorithm used with an SVM-learned filter rather than a likelihood ratio test. “DSC” is a detection cascade where both filters were learned with the SVM.
discriminative arrangements. The natural question, then, is whether purely generatively-trained algorithms will outperform the proposed hybrid detector.

We train a generative classifier for the CLUSTERCLASSIFICATION step. As described in subsection 5.2, we can estimate simple multivariate bernoulli models using empirical frequencies. In the training procedure for the second stage cascade classifiers we have a set of positive examples and negative examples for each latent class \( c \in [C] \). We take an average of the positive examples to create the positive model for that class \( P_{c}^{pos} \in [0, 1]^{T \times F \times D} \), and we let \( P_{c}^{neg} \in [0, 1]^{T \times F \times D} \) be the negative model averaged over the negative examples. We then form a classifier as a log likelihood-ratio test between these two models:

\[
\varphi_{c}(X) = \sum_{t=0}^{l_{c}-1} \sum_{f,e} \log \frac{P_{c}^{pos}(1 - P_{c}^{neg})}{P_{c}^{neg}(1 - P_{c}^{pos})} X + \sum_{t=0}^{l_{c}-1} \log \frac{1 - P_{c}^{pos}}{1 - P_{c}^{neg}}
\] (38)

which is just a linear classifier. We compare this likelihood ratio-based classifier (LRC) with eight mixture components over edges generated from a multitaper log spectrogram that was smoothed only in time. This was compared to our best-performing model (cascade) in Figure 43.
Figure 44: Performance of the cascaded detector on several different phones. They are labeled by their TIMIT codes: refer to Table 5 for the pronunciation.

### 6.4.2 Syllable Detection Without Frequency Smoothing

We also conducted a limited number of experiments with the cascaded detector on other syllables. Eight mixture components were used for the different object models in all cases. The resultant ROC curves for these experiments are displayed in Figure 44 and we give the FOM scores in Table 5. We performed well on all of the syllables with the exception of /tI/, where performance is notably poor for detection thresholds that allow fewer than ten false alarams per hour.
<table>
<thead>
<tr>
<th>TIMIT annotation</th>
<th>FOM</th>
</tr>
</thead>
<tbody>
<tr>
<td>aa r</td>
<td>0.66</td>
</tr>
<tr>
<td>sh iy</td>
<td>0.95</td>
</tr>
<tr>
<td>s ux</td>
<td>0.85</td>
</tr>
<tr>
<td>b ay</td>
<td>0.29</td>
</tr>
<tr>
<td>y ih axr</td>
<td>0.67</td>
</tr>
<tr>
<td>k aa</td>
<td>0.28</td>
</tr>
<tr>
<td>p r</td>
<td>0.34</td>
</tr>
<tr>
<td>t ix</td>
<td>0.03</td>
</tr>
<tr>
<td>ix t</td>
<td>0.15</td>
</tr>
</tbody>
</table>

Table 5: Figure of merit (FOM) scores attained by 8-component mixture-based cascaded detectors for a variety of syllables

For (b,ay) and (y,ih,axr) [22] reported FOM rates of 0.09 and 0.33 respectively on the Boston University Radio Corpus. While not entirely comparable to our method, these results indicate that the proposed approach is able to capture something important that is missed by other detectors.

6.4.3 Syllable Detection With Parts

We now provide a preliminary evaluation of the part features. We have not implemented parts on the non-frequency smoothed features, nor have we investigated parts generally. These plots are considered in Figure 45, Figure 46, Figure 47, Figure 48, Figure 49.

6.4.4 Visualizing the SVM

SVM filters may be interpreted as performing a hypothesis test between a template and an alternative. Given an object template \( P \in (0, 1)^{T \times F \times D} \) and an SVM filter \( W \in \mathbb{R}^{T \times F \times D} \) we can define another template \( Q \in (0, 1)^{T \times F \times D} \) by

\[
Q(t, f, e) = \frac{P(t, f, e)}{P(t, f, e) + (1 - P(t, f, e)) \exp(W(t, f, e))}
\]

then we have

\[
P(t, f, e)(1 - Q(t, f, e)) = P(t, f, e) \left(1 - \frac{P(t, f, e)}{P(t, f, e) + (1 - P(t, f, e)) \exp(W(t, f, e))}\right)
\]

\[
= \frac{P(t, f, e)(1 - P(t, f, e)) \exp(W(t, f, e))}{P(t, f, e) + (1 - P(t, f, e)) \exp(W(t, f, e))}
\]

\[
Q(t, f, e)(1 - P(t, f, e)) = \frac{P(t, f, e)(1 - P(t, f, e)) \exp(W(t, f, e))}{P(t, f, e) + (1 - P(t, f, e)) \exp(W(t, f, e))}
\]

\[
\log \frac{P(t, f, e)(1 - Q(t, f, e))}{Q(t, f, e)(1 - P(t, f, e))} = \log \frac{P(t, f, e)(1 - P(t, f, e)) \exp(W(t, f, e))}{P(t, f, e)(1 - P(t, f, e))}
\]

\[= W.\]
Figure 45: Comparison of the cascaded detector using parts, frequency smoothing, and time-only smoothing.
Figure 46: Comparison of the cascaded detector using parts, frequency smoothing, and time-only smoothing.
Figure 47: Comparison of the cascaded detector using parts, frequency smoothing, and time-only smoothing.
Figure 48: Comparison of the cascaded detector using parts, frequency smoothing, and time-only smoothing.
Figure 49: Comparison of the cascaded detector using parts, frequency smoothing, and time-only smoothing.
Given data \( X \in \{0, 1\}^{F \times T \times D} \), the log-likelihood ratio test of multivariate Bernoulli distributions with parameter \( P \) against parameter \( Q \) is (using the formulation from earlier):

\[
\log \frac{P(X; P)}{P(X; Q)} = \sum_{t,f,e} E(t, f, e) \log \frac{P(t, f, e)(1-Q(t, f, e))}{Q(t, f, e)(1-P(t, f, e))} + \frac{1-P(t, f, e)}{1-Q(t, f, e)} \tag{41}
\]

where \( \cdot \) is the vector dot product, and \( b \) is the same as in Equation 32.

There are no constraints on \( W \), so given any template model \( P \) and its associated SVM-learned linear classifier we may construct a \( Q \) such that \( W \) is essentially performing a log-likelihood ratio test between \( P \) and \( Q \). We call \( Q \) the SVM-alternative.

### 6.4.5 Visualization of Templates

Using the equations in the previous section we may construct some visualizations of the templates—as seen in Figure 50 and Figure 52. In each of the displays we have three columns: the right-most column consists of SVM-filters, the left column has the object models estimated with the EM algorithm, and the middle column is the SVM-alternative constructed using Equation 39. The left columns and middle columns are very similar and difficult to distinguish by visual inspection so the SVM is picking up on very subtle differences between the two models. Moreover, the SVM-filters are relatively sparse with small blobs of coefficients that are either negative or positive—those blobs correspond to features detected in the support vectors with the positive blobs corresponding to positive examples and negative blobs corresponding to negative examples. Interpreting these results is difficult, but it appears that very subtle differences at the template level are making large differences with respect to classification. One possible future line of research is to find audio features that would accentuate these differences.

### 7 Classification

Our ultimate goal in this work is to build a continuous speech recognizer which involves localizing when a speech sound occurs and handling competing hypotheses. Our work on detection is focused on the localization problem, and in this section we deal with the competing hypotheses problem. We utterances \( E \) that are composed of a sequence of phones \( X \). To test the classification power of our statistical models and features, we extract the phones \( X \) from the utterance and test the models of each phone against every other phone. We gauge our performance based on the number of mistakes made by the classification algorithm and we find that the absolute error rate we achieve is quite poor. The mistakes, however, are not completely random: and usually the classifier predicts a phone label that is similar to the ground-truth phone. We also have not implemented a cascade and we suspect recognition can be improved a great deal by using cascades.

#### 7.1 Classification Procedure

The detection setting is we have a query phone \( X \) with an unknown label, and for each phone \( i \) we have a collection of models \( \{P_{i,c}\}_{c=0}^{C_{i} - 1} \) and we wish to infer the phone model that generated the query. Our approach is to use a log-likelihood ratio test:

\[
\Lambda_i(X) = \log \frac{\max_{c \in [C_i]} P(X; P_{i,c})}{\max_{j,c} P(X; P_{j,c})} = \log \max_{c} P(X; P_{i,c}) - \log \max_{j,c} P(X; P_{j,c}) \tag{42}
\]
Figure 50: Example of /ka/. Left column is the template, middle column is the SVM-alternative, and the right column is the SVM-linear classifier. The rows correspond to the different edge types.
Figure 51: Templates for /oi/. Left column is the template, middle column is the SVM-alternative, and the right column is the SVM-linear classifier. The rows correspond to the different edge types.
which tests a class against all others. If $I$ is our set of phone models we note that there will be only one $i \in [I]$ such that

$\Lambda_i(X) \geq 0$

This follows because, except on a set of measure zero,

$$\max_{c \in [C_i]} P(X; P_{i,c}) \neq \max_{c \in [C_{i'}]} P(X; P_{i',c})$$

for $i \neq i'$ hence the set

$$\{ \max_{c \in [C_i]} P(X; P_{i,c}) \mid i \in [I] \}$$

has a unique maximum, say $i^*$, and only a maximal $i^*$ will satisfy $\Lambda_{i^*}(X) \geq 0$. Thus, we may design a classifier $\varphi(X)$ to find the maximum of

$$\varphi(X) = \arg \max_{i \in [I]} \max_{c \in [C_i]} P(X; P_{i,c}).$$

### 7.1.1 Classification with Background

However, if we use the models learned by \textsc{BernoulliMixtureEM} (Algorithm 9 in subsection 5.3) as in subsection 6.3 the models learned will all have different lengths so a direct comparison is impossible. Moreover, the query phones will also not necessarily have the same length in time. To handle this difficulty we use the approach from subsubsection 6.1.1 where each object hypothesis $\theta$ is not just the phone $i$ and phone class $c$ but also the position $\tau$ of the phone in the data $X$, and all frames not within the object support are modeled using a background model $P_{bgd}$. Since we are matching a template $P_{i,c}$ that is of potentially different length than $X$, the phone we want to classify, we need to think carefully about the extraction step. Thus, the hypothesis space $\Theta(X)$ for observation $X$ will be composed of triples $(i, c, \tau)$ where $i$ is the phone, $c$ is the latent class of the phone, and $\tau$ is the time.

The appropriate hypothesis space depends on the extracted data. Let $X \in \{0, 1\}^{L \times F \times D}$ where $X = E(t : t + L)$ for some utterance $E$ so that $t$, $t + L$ are the labeled start and end times for the phone. An object hypothesis triple $(i, c, \tau) \in \Theta(X)$ is constrained so that $i \in [I]$, $c \in [C_i]$, and $\tau \in [t - L/3, t + L/3]$. The final constraint means that $\tau$ must occur within an interval centered at the labeled start time of the phone, and the interval is two-thirds of the labeled length of the phone. This is the same label window employed in subsubsection 6.1.6. We also suspect that the end of the phone could potentially be at time $t + L + L/3$ so that it extends passed the labeled boundary. So, our first attempt at extracting a phone for computing likelihood ratio tests is:

$$X = E(t - L/3 : t + 4L/3).$$

One issue we run into with Equation 45 is $X$ being too short for the template. The phone /p/ can be as short as three frames long while /a/ can be seventy. However, since we do not know the identity of a phone at testing time we would like to be able to test /a/ templates against /p/ data. We handle this through padding.

An obvious choice for the padding is to pad with random data drawn from a background model as defined by Equation 22. We do not use this procedure because doing so makes our classifier randomized, whereas we want it to be deterministic. Instead we pad with the real-valued background template $P_{bgd}$. 

---

89
We demonstrate the usefulness of this approach with a simple example where the template has a longer time support than the example. We will consider two approaches: one where we pad with the model (model-padding) and one where we pad with random data from the model (random-padding). We will show that model-padding achieves the same goal as random-padding but with less variance.

Let $P_{i,c} \in (0,1)^{L_{i,c} \times F \times D}$ be an object template and $X \in \{0,1\}^{L \times F \times D}$ be our data with $L < L_{i,c}$. For the model-padding approach, pad $X$ using $P_{bgd}$ so

$$E_{\text{padded}} = \begin{bmatrix} X \\ P_{bgd} \\ \vdots \\ P_{bgd} \end{bmatrix} \text{ } \text{ } \text{ } L_{i,c} - L \text{ copies.}$$

hence $E_{\text{padded}}$ has length $L_{i,c}$. For the random-padding approach let $B$ be an $(L_{i,c} - L) \times F \times D$ array of Bernoulli random variables such that each frame $B(t)$ is distributed as a multivariate Bernoulli vector with parameter $P_{bgd}$. We then define

$$E_{\text{random}} = \begin{bmatrix} X \\ B \end{bmatrix}.$$ 

Let $\mathcal{L}(\theta; Z)$ be the log-likelihood function of model $\theta$ holding the data $Z$ fixed. Then

$$\mathbb{E}\mathcal{L}(P_{i,c}; E_{\text{random}}) = \mathbb{E}\sum_{t=0}^{L-1} \sum_{f,e} \log \frac{P_{i,c}(t,f,e)}{1 - P_{i,c}(t,f,e)} E(t,f,e) + \log(1 - P_{i,c}(t,f,e))$$

$$+ \sum_{t=L}^{L_{i,c}-1} \left[ \sum_{f,e} \log \frac{P_{i,c}(t,f,e)}{1 - P_{i,c}(t,f,e)} E B(t,f,e) + \log(1 - P_{i,c}(t,f,e)) \right]$$

$$= \mathbb{E}\mathcal{L}(P_{i,c}(0 : L); X)$$

$$+ \sum_{t=L}^{L_{i,c}-1} \left[ \sum_{f,e} P_{bgd}(f,e) \log \frac{P_{i,c}(t,f,e)}{1 - P_{i,c}(t,f,e)} + \log(1 - P_{i,c}(t,f,e)) \right]$$

$$= \mathbb{E}\mathcal{L}(P_{i,c}(0 : L); X) + \sum_{t=L}^{L_{i,c}-1} \mathcal{L}(P_{i,c}(t); P_{bgd})$$

$$= \mathbb{E}\mathcal{L}(P_{i,c}(0 : L); E_{\text{padded}})$$

where we abuse notation to allow the log-likelihood function $\mathcal{L}$ to compute a Bernoulli likelihood over real-valued data. Equation 47 shows that by padding with the background model we are implicitly computing an expectation over the log-likelihood ratio computed on a randomly padded example. In expectation the two approaches are the same except that by padding randomly we increase variance. So we improve upon our original extraction formula Equation 45 using model-padding:

$$X = \begin{bmatrix} E(t - L/3 : t + 4L/3) \\ P_{bgd} \\ \vdots \\ P_{bgd} \end{bmatrix} \text{ } \text{ } \text{ } L_{\text{max}} - L \text{ copies.}$$

where $L$ is the length of the phone and $L_{\text{max}} = \max_{i \in [I], c \in [C_i]} L_{i,c}$. This new $X$ has been padded such that the template with the longest time support, $L_{\text{max}}$, can be tested against the data over the interval $[\tau, \tau + L_{\text{max}}]$ for any $\tau \in [t - L/3, t + L/3]$, so that we can get compute a likelihood for any hypothesis $(i, c, \tau)$ where $\tau$ is within the label window.
We find it is convenient to work with likelihood ratio tests of a phone against background rather than directly with the phone likelihood. The likelihood defined [Equation 47] for a hypothesis \((i, c, \tau)\) and example \(X \in \{0, 1\}^{L \times F \times D}\) models \(E(0 : \tau)\) and \(E(\tau + L_{i,c} : L)\) with the background template \(P_{bgd}\) while \(E(\tau : \tau + L_{i,c})\) is modeled with the template \(P_{i,c}\). The null hypothesis is no object occurs, which means that every frame of \(X\) will be modeled with the background template \(P_{bgd}\). The likelihood ratio test in this case is

\[
\Xi((i, c, \tau); X) = \frac{\prod_{t=\tau}^{t+L_{i,c}} \prod_{f,e} P_{i,c}(t-\tau, f, e)E(t, f, e) \prod_{t=\tau}^{t+L_{i,c}} P_{bgd}(f, e)E(t, f, e) \prod_{t \in [0, \tau-1] \cup [\tau + L_{i,c}, L]} \prod_{f,e} P_{bgd}(f, e)E(t, f, e) \prod_{t \in [0, \tau-1] \cup [\tau + L_{i,c}, L]} \prod_{f,e} P_{bgd}(f, e)E(t, f, e)}{\prod_{t=\tau}^{t+L_{i,c}} \prod_{f,e} P_{i,c}(t-\tau, f, e)E(t, f, e) \prod_{t=\tau}^{t+L_{i,c}} P_{bgd}(f, e)E(t, f, e) \prod_{t \in [0, \tau-1] \cup [\tau + L_{i,c}, L]} \prod_{f,e} P_{bgd}(f, e)E(t, f, e) \prod_{t \in [0, \tau-1] \cup [\tau + L_{i,c}, L]} \prod_{f,e} P_{bgd}(f, e)E(t, f, e)}
\]

(49)

so we compute a linear function of a slice of the data just as in [Equation 32].

A simple generalized likelihood ratio test can be formulated using [Equation 49]. Observe that if the terms are expanded carefully then

\[
\log \frac{L((i, c, \tau); X)}{L((j, c', \tau'); X)} = W_{i,c} \cdot E(\tau : \tau + L_{i,c}) + b_{i,c} - [W_{j,c'} \cdot E(\tau' : \tau' + L_{j,c'}) + b_{j,c'}].
\]

(50)

Assuming that the linear functions all take different values on the data \(X\) there will be some \((i, c, \tau)\) that takes the maximal value

\[
W_{i,c} \cdot E(\tau : \tau + L_{i,c}) + b_{i,c}
\]

which will mean that

\[
\log \frac{L((i, c, \tau); X)}{\max_{(j, c', \tau') \neq (i, c, \tau)} L((j, c', \tau'); X)} = W_{i,c} \cdot E(\tau : \tau + L_{i,c}) + b_{i,c} - \max_{(j, c', \tau') \neq (i, c, \tau)} [W_{j,c'} \cdot E(\tau' : \tau' + L_{j,c'}) + b_{j,c'}]
\]

(51)

So our expanded classification function \(\varphi(X)\) may be defined

\[
\varphi(X) = \arg \max_{i, c, \tau} W_{i,c} \cdot E(\tau : \tau + L_{i,c}) + b_{i,c}.
\]

(52)

### 7.2 Algorithms

Having developed the theory we can now write down the training and testing algorithms for the classification procedure in [Algorithm 23](#). For [Classification](#) to work we require that the data \(X\) was extracted from the utterance and padded with the background model as in [Equation 48](#). The algorithm is basically a brute force search over the parameter space \(\Theta(X)\), which consists in all \((i, c, \tau)\) such that \(i \in [I], c \in [C_i]\), and \(\tau\) in the label-window. The algorithm requires \(\sum_i C_i\) convolutions to be performed between the phone and all the different models. Future work may consider strategies to prune the search space.
Input: \( X \in \{0,1\}^{L \times F \times D} \) extracted data, \( \{\{(W_{i,c}, b_{i,c})\}_{c=0}^{C_i-1}\}_{i=0}^{I-1} \) collection of object filters, \( P_{bgd} \in \mathbb{R}^{F \times D} \) background model

Output: \( v^* \in \mathbb{R} \) classification score, \( (i^*, c^*, \tau^*) \in \Theta \) estimated hypothesis

1: procedure \textsc{Classification}(X, \{\{(W_{i,c}, b_{i,c})\}_{c=0}^{C_i-1}\}_{i=0}^{I-1}, P_{bgd}, L)
2: \( v^* \leftarrow -\infty \)
3: for \( i = 0, \ldots, I-1 \) do
4: for \( c = 0, \ldots, C_i - 1 \) do
5: for \( t = 0, \ldots, L \) do
6: \( v \leftarrow W_{i,c} \cdot E(t : t + L_{i,c}) + b_{i,c} \)
7: if \( v \geq v^* \) then
8: \( v^* \leftarrow v \)
9: \( (i^*, c^*, \tau^*) \leftarrow (i, c, t) \)
10: end if
11: end for
12: end for
13: end for
14: return \( v^*, (i^*, c^*, \tau^*) \)
15: end procedure

Algorithm 23: Classification Algorithm
7.3 Results

The TIMIT dataset was used for this experiment. The models were trained on the training subdivision and tested on the testing subdivision. The features used were edge features, along with our standard multitaper spectrogram with no smoothing along the frequency dimension. The percentage of correct classifications was 42%, which is very poor compared to other systems. However, we present the confusion matrix below to show that the confusions are largely confined to sensible pairs of easily-confuseable phones. We note that cascades were not implemented for classification, and that we could potentially improve these results through the use of cascades.

8 Future Work

We conclude with some discussion of future work.

8.1 Cues for Automatic Speech Recognition

Work by [27, 28] presents a theory of speech perception driven by localized time-frequency cues. We are generally interested in building computational models of acoustic cues: these include localized detectors of signal energy, formant detection/pose estimation, and burst localization. It has long been hypothesized that these play a fundamental role in human speech perception, but no computational model of speech recognition has made use of all of these statistics.

8.2 Improved Features

One interesting result from this work was that multitaper spectrogram features improved upon MFCC features, which suggests that a certain degree of These methods based on binary features can be adapted to work on any signal processing architecture, so they could potentially be used on auditory-inspired features which tend to be faithfully represent formants and onsets more faithfully in the presence of noise.

8.3 Detection with a Patchwork of Phones

Another obvious extension is to use ideas from [3] to build a model for continuous speech recognition. In this model, we allow the object models to overlap in time so that hypothesized objects can occur at any time. More formally, the setting is that we have an utterance $E$ and we seek to find the hypothesis $\theta$ with maximal likelihood on $E$ among the hypotheses in $\Theta(E)$, which is the hypothesis space for the observation $E$. $\Theta(E)$ is constructed using a background model $P_{bgd} \in (0, 1)^{F \times D}$, a set of object models $\{P_c\}_{c=0}^{C-1} \subset (0, 1)^{T \times F \times D}$, and a set of kernels $\{\kappa_c\}_{c=0}^{C-1}$ (so that $\kappa_c$ controls the support of $P_c$ in the utterance ). An object hypothesis $\theta \in \Theta$ for an utterance $E$ is a set of object placements $\{(c_j, t_j)\}_{j=0}^{J-1}$ on $E$ where $c_j$ is the object identity and $t_j$ is the placement time. The likelihood for $E(t, f, e)$ under $\theta$ may be written

$$P(E(t, f, e); \theta) = \begin{cases} \frac{\sum_{j=0}^{J-1} \kappa_{c_j}(t_j - t, f, e)P_{c_j}(t, f, e)}{\sum_{j=0}^{J-1} \kappa_{c_j}(t_j - t, f, e)} & \text{if there is some } j \text{ such that } \kappa_{c_j}(t_j - t, f, e) > 0 \\ P_{bgd}(f, e) & \text{otherwise} \end{cases}$$

(53)
Figure 52: Confusion matrix for classification of TIMIT phones. Entry \((i,j)\) is the number of times ground-truth phone \(j\) was classified as phone \(i\). Darker cells correspond to more classifications. The idea matrix would be black exactly along the diagonal and completely white off the diagonal.
This model can be used to string together arbitrary word, syllable, or phone hypotheses. Additionally, the form of the kernels $\kappa_c$ allows us to specify object models $P_c$ that have support only over a narrow band of frequencies, and a further extension could allow frequency translation of object parts so that we can model pitch explicitly in the model. Formants and bursts could have specific object models that are then combined together to construct models of phones. One way to automatically infer $\kappa_c$ that are localized in the time-frequency plane is to look for spatial clusters of discriminative locations in the templates such as those visualized in Figure 50.

Another powerful feature of the patchwork model is that we can explicitly allow constrained forms of warping. For example, we could model the syllable /ka/ with a /k/ template and an /a/ template that can be placed closer or further part to model length variation. We also should consider more sophisticated model combination techniques beyond the linear form of Equation 53 so that co-articulatory effects could be incorporated explicitly into the model.

We can also treat the object hypotheses $\theta$ as being random variables so that an explicit language model can control the likelihood of different object placements.

References


