SPEECHWORKS INTERNATIONAL
Technical Report

**Algonquin: Statistical Inference of Clean Speech From Noisy Speech**

Alex Park

Friday, August 31, 2002

---

# Introduction and Overview

In this section, we give a brief overview of the theoretical aspects of Algonquin and summarize the papers discussing the algorithm.

## Background

Algonquin is a feature domain approach for robust speech recognition which attempts to improve recognition accuracy in noisy environments by modifying the incoming speech frames before they are passed on to the recognizer. Other examples of feature domain approaches include Parallel Model Combination (PMC), spectral subtraction, and cepstral mean normalization.

Algonquin makes use of statistical inference using Laplace's Method (or vector Taylor series) to estimate the log-spectra of clean speech given the log-spectra of the noise-distorted speech. The distinguishing feature of the Algonquin algorithm versus previous work with vector Taylor series (see Moreno) is the modelling of noise as a random process rather than using just a point estimate.

In order to be effective, Algonquin requires the following:

- Dual channel data - This is required during training in order to generate the error covariance matrix between the computed noisy speech (from clean speech, noise, and channel vectors) and the actual noisy speech.
- Reliable Noise estimate - This is required during training in order to create the computed noisy speech vector. Noise estimation is also re-

quired during recognition in order to provide an initial set of vectors to train the noise model.

- Clean speech model - This is a GMM that is trained over all clean speech training data. This is required during recognition in order to provide a starting point for the clean speech estimate.

## Mathematical Framework

Given the spectrum of a frame of noisy speech, $Y(f)$, the relationship between this observation and $X(f)$, $N(f)$, and $H(f)$, the clean speech, noise, and channel, respectively, is

$$Y(f) = H(f)X(f) + N(f)$$

If we assume that the phases of the noise and speech are uncorrelated, the relationship between the power spectra then becomes

$$|Y(f)|^2 \approx |H(f)|^2|X(f)|^2 + |N(f)|^2$$

if we let $\mathbf{y}$ be the $M$ dimensional vector containing the log-spectra, $\log |Y(f)|^2$ (likewise for $\mathbf{x}$, $\mathbf{n}$, and $\mathbf{h}$), then we get the relation

$$\begin{aligned} \exp(\mathbf{y}) &\approx \exp(\mathbf{h})\exp(\mathbf{x}) + \exp(\mathbf{n}) \\ \Rightarrow \quad \mathbf{y} &\approx \mathbf{h} + \mathbf{x} + \log(\mathbf{1} + \exp(\mathbf{n} - \mathbf{h} - \mathbf{x})) \end{aligned}$$

We denote the function on the right hand side to be the vector function $\mathbf{g}([\mathbf{x} \;\; \mathbf{n} \;\; \mathbf{h}]^T) : \mathcal{R}^{3M} \rightarrow \mathcal{R}^M$.

$$\Rightarrow \mathbf{y} \approx \mathbf{g}([\mathbf{x} \;\; \mathbf{n} \;\; \mathbf{h}]^T)$$

The errors introduced by approximation in the above equation are accounted for by modelling the observation likelihood as a Gaussian with mean $\mathbf{g}([\mathbf{x} \;\; \mathbf{n} \;\; \mathbf{h}]^T)$.

$$p(\mathbf{y}|\mathbf{x}, \mathbf{n}, \mathbf{h}) = \mathcal{N}(\mathbf{y}; \mathbf{g}([\mathbf{x} \;\; \mathbf{n} \;\; \mathbf{h}]^T), \mathbf{\Psi})$$

Here, $\mathbf{\Psi}$ is the covariance matrix of errors that is computed during training using the dual channel data to compare the a priori computed value of $\mathbf{g}([\mathbf{x} \;\; \mathbf{n} \;\; \mathbf{h}]^T)$ from the clean data with $\mathbf{y}$ from the noisy data.

Next, a clean speech GMM is used to model the prior, $p(\mathbf{x}, \mathbf{n}, \mathbf{h})$. This is done by taking the $M$-dimensional GMM for clean speech, and appending

the $M$ dimensional noise and channel GMMs. Therefore, starting from $N_x$ clean speech mixtures, $N_n$ noise mixtures, and $N_h$ channel mixtures, each with $M$ dimensions, we get a combined GMM of $N_s = N_x N_n N_h$ mixtures, which each have $3M$ dimensions. Given a mixture index, $s$, which has the conditional prior is

$$\begin{aligned} p(\mathbf{x}, \mathbf{n}, \mathbf{h}|s) &= \mathcal{N}([\mathbf{x} \;\; \mathbf{n} \;\; \mathbf{h}]^T; \mu_s, \mathbf{\Sigma}_s) \\ \Rightarrow p(\mathbf{y}, \mathbf{x}, \mathbf{n}, \mathbf{h}|s) &= \mathcal{N}(\mathbf{y}; \mathbf{g}([\mathbf{x} \;\; \mathbf{n} \;\; \mathbf{h}]^T), \mathbf{\Psi}) \mathcal{N}([\mathbf{x} \;\; \mathbf{n} \;\; \mathbf{h}]^T; \mu_s, \mathbf{\Sigma}_s) \\ \Rightarrow p(\mathbf{x}, \mathbf{n}, \mathbf{h}|s, \mathbf{y}) &= \frac{1}{C} \left( \mathcal{N}(\mathbf{y}; \mathbf{g}([\mathbf{x} \;\; \mathbf{n} \;\; \mathbf{h}]^T), \mathbf{\Psi}) \mathcal{N}([\mathbf{x} \;\; \mathbf{n} \;\; \mathbf{h}]^T; \mu_s, \mathbf{\Sigma}_s) \right) \end{aligned}$$

where $C$ is a constant representing $p(y)$ that can be accounted for later by normalization. The key approach utilized by Algonquin occurs at this step, which is to approximate this product of Gaussians by a single Gaussian at each mixture

$$p(\mathbf{x}, \mathbf{n}, \mathbf{h}|s, \mathbf{y}) \approx \mathcal{N}([\mathbf{x} \;\; \mathbf{n} \;\; \mathbf{h}]^T; \eta_s, \mathbf{\Phi}_s)$$

Here, $\eta_s$ and $\mathbf{\Phi}_s$ are determined via an iterative approach starting at the mean and variance of the clean speech mixture, $\mu_s$, and $\mathbf{\Sigma}_s$. By neglecting constants for now and taking logarithms, we get

$$\begin{aligned} (\eta_s - [\mathbf{x} \;\; \mathbf{n} \;\; \mathbf{h}])^T \mathbf{\Phi}_s^{-1} (\eta_s - [\mathbf{x} \;\; \mathbf{n} \;\; \mathbf{h}]) &= (\mathbf{y} - \mathbf{g}([\mathbf{x} \;\; \mathbf{n} \;\; \mathbf{h}]^T))^T \mathbf{\Psi}^{-1} (\mathbf{y} - \mathbf{g}([\mathbf{x} \;\; \mathbf{n} \;\; \mathbf{h}]^T)) \\ &\quad + ([\mathbf{x} \;\; \mathbf{n} \;\; \mathbf{h}] - \mu_s)^T \mathbf{\Sigma}_s^{-1} ([\mathbf{x} \;\; \mathbf{n} \;\; \mathbf{h}] - \mu_s) \end{aligned}$$

Using a first order vector Taylor series approximation around $[\mathbf{x} \;\; \mathbf{n} \;\; \mathbf{h}] = \mathbf{z}_0$,

$$\mathbf{y} = \mathbf{g}(\mathbf{z}_0^T) + (\mathbf{z}_1 - \mathbf{z}_0) \mathbf{g}'(\mathbf{z}_0^T)$$

And this simplifies to

$$\begin{aligned} (\eta_s - \mathbf{z}_0)^T \mathbf{\Phi}_s^{-1} (\eta_s - \mathbf{z}_0) &= (\mathbf{y} - \mathbf{g}(\mathbf{z}_0^T))^T \mathbf{\Psi}^{-1} (\mathbf{y} - \mathbf{g}(\mathbf{z}_0^T)) + (\mathbf{z}_0 - \mu_s)^T \mathbf{\Sigma}_s^{-1} (\mathbf{z}_0 - \mu_s) \\ &= (\mathbf{z}_1 - \mathbf{z}_0)^T \mathbf{g}'(\mathbf{z}_0^T)^T \mathbf{\Psi}^{-1} \mathbf{g}'(\mathbf{z}_0^T) (\mathbf{z}_1 - \mathbf{z}_0) + (\mathbf{z}_0 - \mu_s)^T \mathbf{\Sigma}_s^{-1} (\mathbf{z}_0 - \mu_s) \end{aligned}$$

On each subsequent iteration, $i+1$, we can improve the estimate for $\mathbf{\Phi}_s$ and for $\eta_s$ based on the previous calculated values, $\mathbf{\Phi}_s^{(i)}$ and $\eta_s^{(i)}$.

**Updating Variance**

Initially, we start with $\mathbf{\Phi}_s^{(0)} = \mathbf{\Sigma}_s$. To update $\mathbf{\Phi}_s^{(i)}$, we can set $\mathbf{z}_1 = \eta_s = \mu_s$ and allow $\mathbf{z}_0 = \eta_s^{(i)}$.

$$\begin{aligned}(\eta_s^{(i)} - \mu_s)^T[\mathbf{\Phi}_s^{(i+1)}]^{-1}(\eta_s^{(i)} - \mu_s) &= (\mathbf{y} - \mathbf{g}(\eta_s^{(i)}))^T\mathbf{\Psi}^{-1}(\mathbf{y} - \mathbf{g}(\eta_s^{(i)})) \\ &\quad +(\eta_s^{(i)} - \mu_s)^T\mathbf{\Sigma}_s^{-1}(\eta_s^{(i)} - \mu_s) \\ &= (\mu_s - \eta_s^{(i)})^T\mathbf{g}'(\eta_s^{(i)})^T\mathbf{\Psi}^{-1}\mathbf{g}'(\eta_s^{(i)})(\mu_s - \eta_s^{(i)}) \\ &\quad + (\eta_s^{(i)} - \mu_s)^T\mathbf{\Sigma}_s^{-1}(\eta_s^{(i)} - \mu_s)\end{aligned}$$

$$\Rightarrow \quad [\mathbf{\Phi}_s^{(i+1)}]^{-1} = \mathbf{g}'(\eta_s^{(i)})^T\mathbf{\Psi}^{-1}\mathbf{g}'(\eta_s^{(i)}) + \mathbf{\Sigma}_s^{-1}$$

$$\therefore \mathbf{\Phi}_s^{(i+1)} = \left[\mathbf{\Sigma}_s^{-1} + \mathbf{g}'(\eta_s^{(i)})^T\mathbf{\Psi}^{-1}\mathbf{g}'(\eta_s^{(i)})\right]^{-1} \tag{1}$$

**Updating Mean**

Initially, we start with $\eta_s^{(0)} = \mu_s$. To update for $\eta_s^{(i)}$, we can set $\mathbf{z}_0 = \eta_s^{(i)}$, and $\mathbf{z}_1 = \eta_s = \eta_s^{(i+1)}$, then

$$\begin{aligned}(\eta_s^{(i+1)} - \eta_s^{(i)})^T\mathbf{\Phi}_s^{-1}(\eta_s^{(i+1)} - \eta_s^{(i)}) &= (\eta_s^{(i+1)} - \eta_s^{(i)})^T\mathbf{g}'(\eta_s^{(i)})^T\mathbf{\Psi}^{-1}(\mathbf{y} - \mathbf{g}(\eta_s^{(i)})) \\ &\quad + (\eta_s^{(i+1)} - \eta_s^{(i)})^T\mathbf{\Sigma}_s^{-1}(\mu_s - \eta_s^{(i)}) \\ &\quad - \underbrace{(\eta_s^{(i+1)} - \mu_s)^T\mathbf{\Sigma}_s^{-1}(\mu_s - \eta_s^{(i)})}_{\approx 0}\end{aligned}$$

**NOTE:** For reasons I can't explain, the original paper makes the assumption that the last term on the right hand side is negliable, in which case we get

$$(\eta_s^{(i+1)} - \eta_s^{(i)}) = \mathbf{\Phi}_s^{(i+1)}\left(\mathbf{g}'(\eta_s^{(i)})^T\mathbf{\Psi}^{-1}(\mathbf{y} - \mathbf{g}(\eta_s^{(i)})) + \mathbf{\Sigma}_s^{-1}(\mu_s - \eta_s^{(i)})\right)$$

$$\therefore \quad \eta_s^{(i+1)} = \eta_s^{(i)} + \mathbf{\Phi}_s^{(i+1)}\left(\mathbf{g}'(\eta_s^{(i)})^T\mathbf{\Psi}^{-1}(\mathbf{y} - \mathbf{g}(\eta_s^{(i)})) + \mathbf{\Sigma}_s^{-1}(\mu_s - \eta_s^{(i)})\right) \tag{2}$$

**Computing Posteriors**

Our goal is to ultimately determine $\hat{\mathbf{x}}$, the MMSE of the clean speech signal, which is given by

$$\begin{aligned}\hat{\mathbf{x}} &= \sum_{s_x}\int_{\mathbf{n},\mathbf{h}} E[\mathbf{x}|s_x,\mathbf{y},\mathbf{n},\mathbf{h}] \\ &= \sum_{s} E[\mathbf{x}|s,\mathbf{y}] \\ &= \sum_{s} \eta_s p(s|\mathbf{y})\end{aligned}$$

The posterior probability for a given mixture, $s$, can be computed using the final values for $\eta_s$ and $\mathbf{\Phi}_s$ and the mixture prior, $p(s) = \pi_s$. This comes out to

$$\begin{aligned}\rho_s^{(I)} = p(s|\mathbf{y}) &= \lambda\left(\pi_s\frac{|\mathbf{\Phi}_s^{(I)}|^{1/2}}{|\mathbf{\Sigma}_s|^{1/2}}\right)\cdot\exp\left(-\frac{1}{2}(\mathbf{y}-\mathbf{g}(\eta_s^{(I)}))^T\mathbf{\Psi}^{-1}(\mathbf{y}-\mathbf{g}(\eta_s^{(I)}))\right. \\ &\quad -\frac{1}{2}tr(\mathbf{\Sigma}_s^{-1}\mathbf{\Phi}_s^{(I)}) - \frac{1}{2}(\eta_s^{(I)}-\mu_s)^T\mathbf{\Sigma}_s^{-1}(\eta_s^{(I)}-\mu_s) \\ &\quad \left. -\frac{1}{2}tr(\mathbf{g}'(\eta_s^{(I)})^T\mathbf{\Psi}^{-1}\mathbf{g}'(\eta_s^{(I)})\mathbf{\Phi}_s^{(I)})\right)\end{aligned}$$

The $\lambda$ is a normalizing factor such that

$$\sum_s \rho_s^{(I)} = 1$$

The final estimate of the denoised speech frame is

$$\hat{\mathbf{x}} = \sum_s \rho_s^{(I)}\eta_s^{(I)}$$

Note that this process must be recomputed for each frame of speech in the test utterance.

# Implementation

In this section, we discuss details related to our specific implementation of the algorithm. Although the previous section discussed the generalized Algonquin procedure which includes channel compensation, we only explored the noise compensation (i.e., we assumed $H(f)$ was flat).

There were two key differences between our implementation of Algonquin and the one originally described in the published papers. The first difference was in the front end feature sets used. Although we both used log-spectra, ours used $\log_{10}$ rather than $\log_e$. Moreover, it is likely that the number of dimensions and the filterbanks used were also slightly different. The second difference was in the method used to estimate noise frames from the training/test utterance. While the published paper used speech/silence detection to base their noise estimates on non-speech frames, the implementation detailed in this report used a different method known as quantile estimation [5].

## Training the Clean Speech model

In order to perform the denoising procedure during the recognition phase of Algonquin, a clean speech GMM is required to provide the initial means and variances. The clean speech GMM used for this task was trained globally over all speech frames from the nearfield/clean version of the training data. The training tools used consisted of three commands which were based closely upon the tools used for training diphone models from alignments:

- `get_single_gmm_cache_from_alignments`

  - This command goes through the alignments and extracts training vectors corresponding to the labels provided in a GMMlabels file. The training vectors are saved in a cache file.

- `train_single_gmm_from_cache`

  - This command goes through the cache file of extracted training vectors and trains an initial GMM model.

- `train_single_gmm_from_alignments`

  - This command goes through the entire set of alignments and trains a single GMM using training vectors corresponding to a specific set of labels. The model training can be initialized with the models obtained from cache training, or can be started from scratch.

These commands were combined into a perl script, `TrainSingleGMM.pl`, which creates a cache, initializes a set of models, then trains the models over all alignments.

For training, we trained two separate GMMs: one for speech, and one for silence. After these individual GMMs were trained, we combined them

into a single GMM. The purpose of this was to prevent the large number of silence frames in the training data from distorting the priors. The separate training was accomplished by running the `TrainSingleGMM.pl` script twice, with different label sets each time.

After training the speech and silence GMMs, they were combined into a single GMM using the `combine_single_gmm` tool, which takes the following arguments:

- `<in_model1>` The file containing the first single GMM model.
- `<in_model2>` The file containing the second single GMM model.
- `<out_model>` The file to save the combined model out to.
- `<weight1>` The weight to apply to the priors of the first model (between 0 and 1).
- `<weight2>` The weight to apply to the priors of the second model (should be equal to 1-weight1).

In order to follow the training procedures developed in the papers describing Algonquin, we used the following parameters in training the clean speech GMM.

| Training parameters | Clean |
|---|---|
| Feature | `swimfcc_filterbank` |
| Number of dimensions | 40 |
| Number of speech mixtures | 256 |
| Number of silence mixtures | 12 |
| Number of training iterations | 5 |
| Speech model weight | 0.9 |
| Silence model weight | 0.1 |

Table 1: Parameters used for training clean speech model

## Computing the Error Covariance Matrix, $\Psi$

Because we are using 40 dimensional log-spectra, which have been passed through a filterbank, we know that $\mathbf{y}$ is not exactly equal to $\mathbf{g}([\mathbf{x}\ \mathbf{n}\ \mathbf{h}])$. The

error covariance matrix, $\Psi$, attempts to model the error in this approximation. In order to compute the error covariance, it is important to have access to both the noisy and clean versions of the same training data. Since we did not attempt to compensate for channel effects (i.e., $H(f) = 1$, $\mathbf{h} = 0$), our main concern here was to come up with an accurate estimate of the noise for a given input frame.

We made the initial assumption that the error covariance matrix was diagonal, and began by creating three separate feature caches using the `compute_features` tool.

(i) $\mathbf{x}$ - This consisted of computing 40-dimensional `swimfcc_filterbank` features from the nearfield/clean version of the training data. For each utterance, log spectral frames were computed at 10 ms intervals, resulting in feature vectors for each frame in the utterance.

(ii) $\mathbf{y}$ - This consisted of computing the same `swimfcc_filterbank` features from the farfield/noisy version of the training data.

(iii) $\mathbf{n}$ - This consisted of computing `quantile_process` features from the farfield/noisy version of the training data. The `quantile_process` feature module takes as input a set of 120-dimensional power spectral frames, computes a quantile noise estimate in the power spectral domain, then converts this estimate to a 40-dimensional log-spectral estimate. This computation results in a single noise estimate vector for each utterance.

After computing feature caches for $\mathbf{x}$, $\mathbf{y}$, and $\mathbf{n}$, we call `compute_error_variance`, which calculates the error variance per dimension between $\mathbf{y}$ and $\mathbf{g}([\mathbf{x}\ \mathbf{n}\ 0])$. The error covariance matrix is then saved out as a 40-dimensional vector.

## Quantile Based Noise Estimation

In the previous section, we discussed noise estimation for the training phase. During training, it is sufficient to compute a point estimate of the noise for the purposes of computing the error variance. However, one of the main advantages of the Algonquin framework is that it models the noise probabilistically. In order to model the noise, a *set* of point estimates is required.

During recognition, a noise GMM is trained for the input utterance using a set of noise vector estimates which are computed by using quantile estimates

over a sliding window. In the `algonquin_fromwave` feature module, two variables are set which define the characteristics of the sliding window.

- `NOISE_WINDOW` - This is the size of the window (in seconds) over which to compute the quantile noise estimate. In our experiments, this was set to 0.5 seconds.
- `NOISE_OVERLAP` - This is the amount to overlap (in seconds) between subsequent windows. In our experiments, this was set to 0.1 seconds.

The number of noise mixtures to use is specified in the `feature.txt` file. Initially, we used a single noise mixture (`noise_mixtures = 1`), but also attempted different numbers of noise mixtures later. During recognition, noise vectors are computed over the entire utterance, and then used to train the noise model with `train_mdg`.

## Iterative Update Computation

After training the noise means ($\mu_{n1}$, $\mu_{n2}$, ..., $\mu_{N_n}$) and variances ($\Sigma_{n1}$, $\Sigma_{n2}$, ...), they are concatenated with the clean speech GMM means ($\mu_{x1}$, $\mu_{x2}$, ..., $\mu_{N_x}$) and variances ($\Sigma_{x1}$, $\Sigma_{x2}$, ...) to create the global set, $(\mu_s, \Sigma_s)$.

$$\begin{array}{rclrclcrcl}
\mu_{s1} & = & [\mu_{x1} \;\; \mu_{n1}] & \mu_{s(N_x+1)} & = & [\mu_{x1} \;\; \mu_{n2}] & \cdots & \mu_{s(N_x(N_n-1)+1)} & = & [\mu_{x1} \;\; \mu_{N_n}] \\
\mu_{s2} & = & [\mu_{x2} \;\; \mu_{n1}] & \mu_{s(N_x+2)} & = & [\mu_{x2} \;\; \mu_{n2}] & \cdots & \mu_{s(N_x(N_n-1)+2)} & = & [\mu_{x2} \;\; \mu_{N_n}] \\
 & \vdots & & & \vdots & & & & \vdots & \\
\mu_{s(N_x)} & = & [\mu_{N_x} \;\; \mu_{n1}] & \mu_{s(2N_x)} & = & [\mu_{N_x} \;\; \mu_{n2}] & \cdots & \mu_{s(N_x N_n)} & = & [\mu_{N_x} \;\; \mu_{N_n}]
\end{array}$$

Likewise, the $M \times M$ covariance matrices are also combined to make $2M \times 2M$ covariance matrices. Since $\Sigma_x$ and $\Sigma_n$ are diagonal, the global covariance matrices, $\Sigma_s$, are represented as $2M$-dimensional vectors which are formed by concatenating the clean speech and noise covariance vectors.
For each $M$-dimensional input frame, $\mathbf{y}$, the following sequence of steps is performed for each global mixture, $(\mu_s, \Sigma_s)$.

0. Set $i = 0$, and $\eta_s^{(i)} = \mu_s$ and $\mathbf{\Phi}_s^{(i)} = \Sigma_s$.
1. Compute $\mathbf{g}(\eta_s^{(i)})$ and $\mathbf{g}'(\eta_s^{(i)})$. (`compute_gfunc_gprime()`)
2. Compute $\mathbf{\Phi}_s^{(i+1)}$. (`update_phi`())
3. Compute $\eta_s^{(i+1)}$. (`update_nu`())

4. Unless $i+1$ equals the number of desired iterations, increment $i$ by 1 and repeat steps 1-4.

Although the update equations and functional computations described in the previous section are written in matrix notation, the use of diagonal matrices for many of the covariance matrices simplifies many of the operations to vector operations.

### Computing $\mathbf{g}(\eta_s^{(i)})$, (`compute_gfunc_gprime()`)

The function $\mathbf{g}()$ maps a $2M$-dimensional vector into an $M$-dimensional vector. The implementation assumes that we are using $\log_{10}$ spectra, and that the first $M$ dimensions correspond to speech, and that the second $M$ dimensions correspond to noise. Therefore,

$$\begin{aligned} g(\eta_s^{(i)})[k] &= \log_{10}\left(10^{\eta_s^{(i)}[k]} + 10^{\eta_s^{(i)}[k+M]}\right) \\ &= \frac{\log\left(\exp(\eta_s^{(i)}[k]\log(10)) + \exp(\eta_s^{(i)}[k+M]\log(10))\right)}{\log(10)} \end{aligned}$$

### Computing $\mathbf{g}'(\eta_s^{(i)})$, (`compute_gfunc_gprime()`)

The function $\mathbf{g}'()$ maps the $2M$-dimensional vector into a $M \times 2M$-dimensional matrix. The derivative is defined as follows:

$$\mathbf{g}'(\eta_s^{(i)}) = \begin{pmatrix} \frac{\partial g[1]}{\partial \eta_s^{(i)}[1]} & \frac{\partial g[1]}{\partial \eta_s^{(i)}[2]} & \cdots & \frac{\partial g[1]}{\partial \eta_s^{(i)}[2M]} \\ \frac{\partial g[2]}{\partial \eta_s^{(i)}[1]} & \frac{\partial g[2]}{\partial \eta_s^{(i)}[2]} & \cdots & \frac{\partial g[2]}{\partial \eta_s^{(i)}[2M]} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial g[M]}{\partial \eta_s^{(i)}[1]} & \frac{\partial g[M]}{\partial \eta_s^{(i)}[2]} & \cdots & \frac{\partial g[M]}{\partial \eta_s^{(i)}[2M]} \end{pmatrix}$$

Since $g[k]$ is only a function of $\eta_s^{(i)}[k]$ and $\eta_s^{(i)}[k+M]$, we can see that most of the terms of this matrix are actually 0. Only the terms $\frac{\partial g[k]}{\partial \eta_s^{(i)}[k]}$ and $\frac{\partial g[k]}{\partial \eta_s^{(i)}[k+M]}$ have non-zero values.

$$\frac{\partial g[k]}{\partial \eta_s^{(i)}[k]} = \frac{10^{\eta_s^{(i)}[k]}}{\left(10^{\eta_s^{(i)}[k]} + 10^{\eta_s^{(i)}[k+M]}\right)}$$

And

$$\frac{\partial g[k]}{\partial \eta_s^{(i)}[k+M]} = \frac{10^{\eta_s^{(i)}[k+M]}}{\left(10^{\eta_s^{(i)}[k]} + 10^{\eta_s^{(i)}[k+M]}\right)}$$
$$= 1 - \frac{\partial g[k]}{\partial \eta_s^{(i)}[k]}$$

Since the resulting derivative matrix looks like

$$\mathbf{g}'(\eta_s^{(i)}) = \begin{pmatrix} \frac{\partial g[1]}{\partial \eta_s^{(i)}[1]} & 0 & \cdots & 0 & \frac{\partial g[1]}{\partial \eta_s^{(i)}[M+1]} & 0 & \cdots & 0 \\ 0 & \frac{\partial g[2]}{\partial \eta_s^{(i)}[2]} & \cdots & 0 & 0 & \frac{\partial g[2]}{\partial \eta_s^{(i)}[M+2]} & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & \frac{\partial g[M]}{\partial \eta_s^{(i)}[M]} & 0 & 0 & \cdots & \frac{\partial g[M]}{\partial \eta_s^{(i)}[2M]} \end{pmatrix}$$

We store $\mathbf{g}'(\eta_s^{(i)}$ in a collapsed $2M$-dimensional vector:

$$\mathbf{g}'(\eta_s^{(i)}) = \left(\frac{\partial g[1]}{\partial \eta_s^{(i)}[1]}, \frac{\partial g[2]}{\partial \eta_s^{(i)}[2]}, \ldots, \frac{\partial g[M]}{\partial \eta_s^{(i)}[M]}, \frac{\partial g[1]}{\partial \eta_s^{(i)}[M+1]}, \frac{\partial g[2]}{\partial \eta_s^{(i)}[M+2]}, \ldots, \frac{\partial g[M]}{\partial \eta_s^{(i)}[2M]}\right)$$

**Updating $\mathbf{\Phi}_s^{(i)}$ (`update_phi()`)**

From the update equation for $\mathbf{\Phi}_s^{(i+1)}$, (Equation 1), we can see that $\mathbf{\Phi}_s^{(i)}$ is a $2M \times 2M$ dimensional matrix which is computed by taking the inverse of

$$\mathbf{\Sigma}_s^{-1} + \mathbf{g}'(\eta_s^{(i)})^T \mathbf{\Psi}^{-1} \mathbf{g}'(\eta_s^{(i)})$$

Note that both $\mathbf{\Sigma}_s^{-1}$ and $\mathbf{\Psi}^{-1}$ are square diagonal. However, since $\mathbf{g}'()$ is not, the structure of $\left(\mathbf{\Phi}_s^{(i+1)}\right)^{-1}$ becomes

$$\left(\mathbf{\Phi}_s^{(i+1)}\right)^{-1} = \begin{pmatrix} \left(\mathbf{\Phi}_s^{(i+1)}\right)_a^{-1} & \left(\mathbf{\Phi}_s^{(i+1)}\right)_b^{-1} \\ \left(\mathbf{\Phi}_s^{(i+1)}\right)_c^{-1} & \left(\mathbf{\Phi}_s^{(i+1)}\right)_d^{-1} \end{pmatrix}$$

where each of $\left(\mathbf{\Phi}_s^{(i+1)}\right)^{-1}_{\{a,b,c,d\}}$ are diagonal $M \times M$ dimensional matrices.

$$\left(\mathbf{\Phi}_s^{(i+1)}\right)^{-1}_a = \begin{pmatrix} \mathbf{\Sigma}_s^{-1}[1] + \left(\mathbf{g}'(\eta_s^{(i)})[1][1]\right)^2 \mathbf{\Psi}^{-1}[1] & \dots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \dots & \mathbf{\Sigma}_s^{-1}[M] + \left(\mathbf{g}'(\eta_s^{(i)})[M][M]\right)^2 \mathbf{\Psi}^{-1}[M] \end{pmatrix}$$

$$\begin{aligned} \left(\mathbf{\Phi}_s^{(i+1)}\right)^{-1}_b &= \begin{pmatrix} \mathbf{g}'(\eta_s^{(i)})[1][1]\mathbf{\Psi}^{-1}[1]\mathbf{g}'(\eta_s^{(i)})[1][M+1] & \dots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \dots & \mathbf{g}'(\eta_s^{(i)})[M][M]\mathbf{\Psi}^{-1}[M]\mathbf{g}'(\eta_s^{(i)})[M][2M] \end{pmatrix} \\ \left(\mathbf{\Phi}_s^{(i+1)}\right)^{-1}_c &= \left(\mathbf{\Phi}_s^{(i+1)}\right)^{-1}_b \end{aligned}$$

$$\left(\mathbf{\Phi}_s^{(i+1)}\right)^{-1}_d = \begin{pmatrix} \mathbf{\Sigma}_s^{-1}[M+1] + \left(\mathbf{g}'(\eta_s^{(i)})[1][1]\right)^2 \mathbf{\Psi}^{-1}[1] & \dots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \dots & \mathbf{\Sigma}_s^{-1}[2M] + \left(\mathbf{g}'(\eta_s^{(i)})[M][2M]\right)^2 \mathbf{\Psi}^{-1}[M] \end{pmatrix}$$

Since matrix inversion is generally an expensive operation, we utilize the special structure of $\left(\mathbf{\Phi}_s^{(i+1)}\right)^{-1}$ to perform inversion using Gaussian elimination. The general idea here is that given a matrix:

$$\mathcal{A} = \begin{pmatrix} a_{11} & \dots & 0 & b_{11} & \dots & 0 \\ \vdots & \ddots & 0 & \vdots & \ddots & 0 \\ 0 & \dots & a_{MM} & 0 & \dots & b_{MM} \\ c_{11} & \dots & 0 & d_{11} & \dots & 0 \\ \vdots & \ddots & 0 & \vdots & \ddots & 0 \\ 0 & \dots & c_{MM} & 0 & \dots & d_{MM} \end{pmatrix}$$

The inverse $\mathcal{A}^{-1}$ will be of the form

$$\mathcal{A}^{-1} = \begin{pmatrix} a'_{11} & \dots & 0 & b'_{11} & \dots & 0 \\ \vdots & \ddots & 0 & \vdots & \ddots & 0 \\ 0 & \dots & a'_{MM} & 0 & \dots & b'_{MM} \\ c'_{11} & \dots & 0 & d'_{11} & \dots & 0 \\ \vdots & \ddots & 0 & \vdots & \ddots & 0 \\ 0 & \dots & c'_{MM} & 0 & \dots & d'_{MM} \end{pmatrix}$$

where

$$\begin{pmatrix} a'_{kk} & b'_{kk} \\ c'_{kk} & d'_{kk} \end{pmatrix} = \frac{1}{a_{kk}d_{kk} - b_{kk}c_{kk}} \begin{pmatrix} d_{kk} & -b_{kk} \\ -c_{kk} & a_{kk} \end{pmatrix}$$

The procedure for calculating $\mathbf{\Phi}_s^{(i)}$, therefore is to first compute $\left(\mathbf{\Phi}_s^{(i)}\right)^{-1}$, then invert it using the fast inversion technique described above. Due to its diagonal structure, $\mathbf{\Phi}_s^{(i)}$ is stored as a $4M$-dimensional vector

$$\mathbf{\Phi}_s^{(i)} = \left[diag\left(\mathbf{\Phi}_s^{(i+1)}\right)_a \quad diag\left(\mathbf{\Phi}_s^{(i+1)}\right)_b \quad diag\left(\mathbf{\Phi}_s^{(i+1)}\right)_c \quad diag\left(\mathbf{\Phi}_s^{(i+1)}\right)_d\right]$$

### Updating $\eta_s^{(i)}$ (`update_nu()`)

Once $\mathbf{\Phi}_s^{(i)}$ has been updated, $\eta_s^{(i)}$ is simple to compute using standard matrix multiplication rules applied to Equation 2.

### Posterior Computation (`compute_posterior_probs()`)

After $\eta_s^{(I)}$ and $\mathbf{\Phi}_s^{(I)}$ have been computed for each mixture $s$, the posterior contributions of each mixture are calculated as described earlier, and used to weight the mixture means. This weighted sum of means is then used as the final estimate of the denoised frame, $\hat{\mathbf{x}}$.

### Exponential Table Lookup

In computing $\mathbf{g}()$ and $\mathbf{g}'()$, it is necessary to take exponentials and logarithms of both spectral values and probability values. Because they are called many times over the course of the algorithm, the exponential and logarithm functions bottleneck the overall speed of the algorithm. For the `exp` function, at least, this problem was ameliorated by using a table lookup. During initialization, an array of exponential values is precomputed and cached. Because two types of exponential values (`exp`$(\eta_s^{(i)})$ and `exp`$(\log(p(x)))$) are computed, we used two separate exponential lookup tables to account for the differences in scale.

In general, $\eta_\mathbf{s}$ ranges in values from 10 to 40. On the other hand, $\log(p(x))$ ranges in values from 0 to $-\infty$. For $\eta_\mathbf{s}$, if the value falls within the range 0 to `EXP_HI`, then `exp`$(\eta_\mathbf{s})$ is found by looking in the table, otherwise, it is considered as a cache "miss", and the actual value is computed. For the probability scale, cache "hits" are looked up in the table, while cache

"misses" are assigned to values of 0 or 1 depending on which index boundary the operand crosses.

# Extensions

In this section, we discuss possible extensions to the use of the algonquin algorithm that may offer either improved accuracy, or faster computation.

## Dimensional Algonquin

This technique uses a set of dimensional mixture Gaussians trained on clean speech as the basis for a dimensionally uncoupled version of Algonquin. Instead of using a single GMM with $N_x$ mixtures for the clean speech, we build a separate GMM for each dimension of the feature vector. This allows for greater resolution while lowering the number of mixtures per dimension. The dimensional GMM can be trained using the tool
`train_single_gmm_from_cache_by_dim_fast`.

A major advantage of this approach is that it can potentially reduce the computation time of the posterior density estimation by reducing $N_x$. Moreover, since the overall framework of Algonquin (up to the final posterior calculation) assumes dimensional independence, the iterative update and matrix computation procedures do not have to be changed. The only required change is in the computation of $\rho_s^{(I)}$. Instead of computing a posterior probability for each vector mean, the posterior must be computed for each mixture mean of each dimension. A preliminary implementation of this module was completed in `algonquin_from_wave_by_dim`. There is currently a bug in this code which stores a value of `NaN` in the first dimension of the first output feature frame. Initial experimentation with this module did not seem very promising. Using a 36 mixture dimensional GMM for the clean speech with a single mixture noise model, the word error rate on the farfield `capri_050` data went from 59.7 % with no processing to 67.0 % with dimensional Algonquin processing.

## Staggered Algonquin

Staggering the frame level computations is another possible variation on Algonquin that was not implemented, but may also help to reduce computation

significantly. Although the algorithm described in its original form requires re-estimation of $\eta_s^{(I)}$ and $\mathbf{\Phi}_s^{(I)}$ on every frame, this is not always necessary. In particular, since the frame-level characteristics of speech can be slowly varying, with abrupt transitions, it would make sense to only re-estimate the mixture parameters when the input frame characteristics have "drifted" beyond a certain threshold from the last time the parameters were estimated. Using this type of strategy, $\eta_s^{(I)}$ and $\mathbf{\Phi}_s^{(I)}$ could be computed only when the input feature vector has changed significantly. For those feature vectors where the parameters are not re-estimated, the output feature vectors could be computed by re-calculating just the posterior $\rho_s^{(I)}$, which is fast compared to the iterative steps.

## Hi-Resolution Algonquin

Finally, a third variation of Algonquin is the use of higher resolution clean speech models. By increasing the number of mixtures in the clean speech GMM, the spectral resolution of the initial means, $\mu_s$, should be improved. Though this approach would likely help to improve the accuracy performance of the algorithm, the greater number of mixtures will also slow it down appreciably. Though larger GMMs were trained, this approach was not attempted due to time constraints.

# Results

In this section, we report on some of the results obtained using Algonquin and compare them against the baseline system. We also give the results of some results obtained by computing $\hat{\mathbf{x}}$ from the dimensional and vector single GMM models (quantized features).

## Baseline and Quantized Results

Table 2 shows the error rates achieved by the baseline recognizer on the nearfield and farfield versions of the three test sets (shown in increasing levels of background noise) without any type of additional preprocessing of the spectral features.

| Test set | Nearfield | Farfield |
|---|---|---|
| capri-low | 2.1 | 31.4 |
| capri-050 | 2.3 | 59.7 |
| capri-075 | 4.3 | 86.2 |

Table 2: Error rate - Testing without compensation of any kind

Using vector quantization with a clean speech GMM, the incoming spectral features, $\mathbf{x}$ were preprocessed to yield

$$\hat{\mathbf{x}} = \sum_{s} p(\mathbf{x}|s)\mu_s$$

where the $s$ are the mixtures of the clean speech GMM. The result of varying the total number of mixtures on the nearfield data are shown in Table 3. The purpose of this experiment was to verify the quality of the initial clean speech GMM and to observe that performance is improved by increasing the number of mixtures. Clearly, the use of vector quantized speech features results in significant quantization error when compared to the baseline. Moreover, the noise level in the farfield data leads to even greater error rates due to incorrect computation of the posteriors, $p(\mathbf{x}|s)$.

| Test set | Nearfield (M = 268) | Nearfield (M=524) | Nearfield (M=1036) | Farfield (M=268) |
|---|---|---|---|---|
| capri-low | 15.9 | 14.3 | 9.9 | 69.9 |
| capri-050 | 19.6 | 15.3 | 14.5 | 93.1 |
| capri-075 | 22.8 | 18.0 | 14.9 | 98.7 |

Table 3: Error rate - Testing using clean speech vector quantization

Using the dimensional clean speech model, the quantized output feature vector was computed as $\hat{\mathbf{x}} = [\hat{\mathbf{x}}_0 \hat{\mathbf{x}}_1 \ldots \hat{\mathbf{x}}_M]^T$, where $\hat{\mathbf{x}}_i$ is given by

$$\hat{\mathbf{x}}_i = \sum_{s_i} p(\mathbf{x}|s_i)\mu_{s_i}$$

In this equation, the $s_i$ are the mixtures for dimension $i$. The error rates achieved when using a 36-mixture dimensional clean speech GMM are shown

in Table 4. The results in this table are interesting because the error rates on the nearfield data are actually lower than those in the baseline for two of the test sets, suggesting that the dimensional preprocessing may reduce the quantization error between the training and test data when there are matched conditions. Figure 1 summarizes the results obtained by the different quantization methods on the nearfield test sets.

| Test set | Nearfield (36 Mixtures) | Farfield |
|---|---|---|
| capri-low | 2.3 | 40.9 |
| capri-050 | 2.1 | 77.1 |
| capri-075 | 3.1 | 96.8 |

Table 4: Error rate - Testing using clean speech dimensional quantization

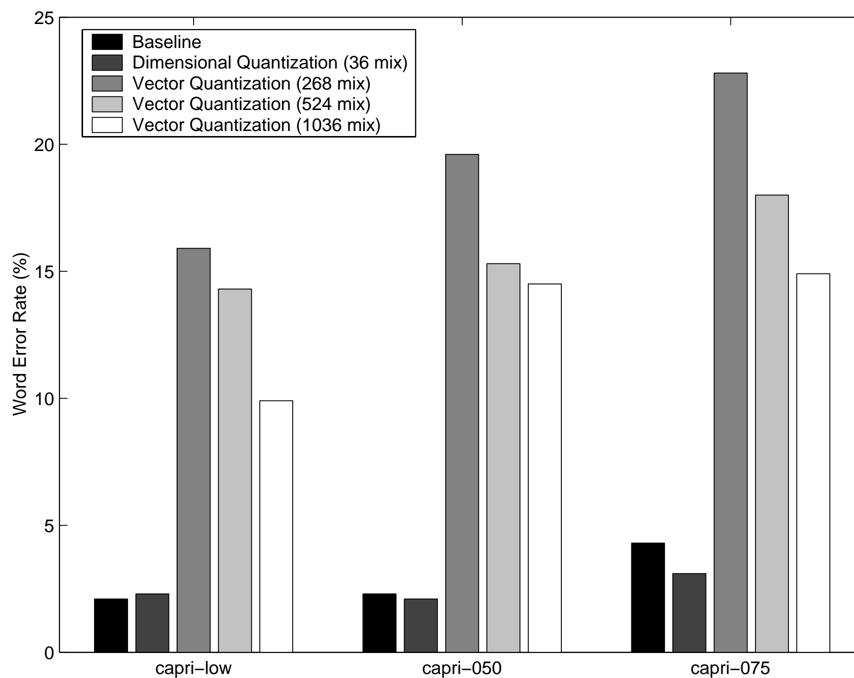


Figure 1: Word error rates of quantization methods on nearfield data.

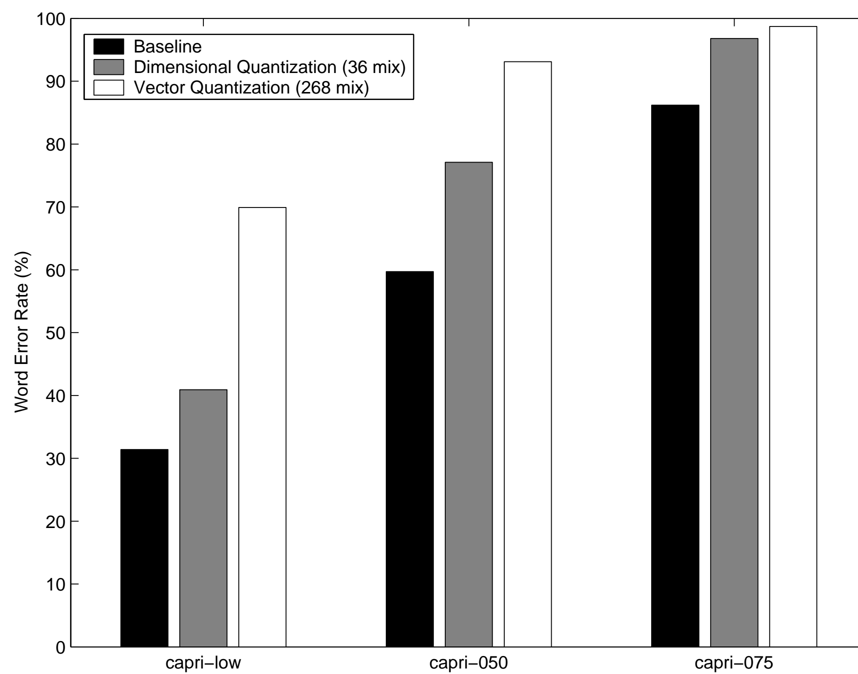


Figure 2: Word error rates of quantization methods on farfield data.

## Algonquin Results

In this section, we give the results obtained when the spectral features have been preprocessed using Algonquin. The word error rates for both nearfield and farfield test data are shown in Table 5. For the farfield data, we also varied the number of noise mixtures ($N_n$) used in the computation. These results are also shown in Figure 3. When compared to the baseline, we can see that preprocessing using Algonquin reduces the word error rate across all three farfield test sets by significant amounts. In particular, by using 2 noise mixtures, relative reductions in error rate of 27.7 %, 42.0 %, and 33.7 % are achieved on `capri_low`, `capri_050`, and `capri_075`, respectively. We note that while going from 1 to 2 noise mixtures improves accuracy, extending the number of mixtures to 4 actually hurts performance. This is likely due to the small number of noise vectors gathered for each utterance, which leads to inaccurate parameter estimation for the noise mixtures when $N_n$ gets too large.

Additionally, while it appears that Algonquin hurts performance on the nearfield data, these nearfield error rates are lower than those obtained by directly quantizing with the clean speech GMM. This suggests that Algonquin shifts the means of the clean speech GMM closer towards the actual means, but cannot completely compensate for the quantization error introduced by using only 268 mixtures. In Figure 4, the error rates obtained by the different preprocessing methods on the nearfield data are compared.

| Test set | Nearfield ($N_n$=1) | Farfield ($N_n$=1) | Farfield ($N_n$=2) | Farfield ($N_n$=4) |
|---|---|---|---|---|
| capri-low | 13.7 | 28.2 | 22.7 | 67.9 |
| capri-050 | 14.5 | 40.7 | 34.6 | 66.0 |
| capri-075 | 13.1 | 63.4 | 57.1 | 88.1 |

Table 5: Error rate - Testing using Algonquin vector estimation

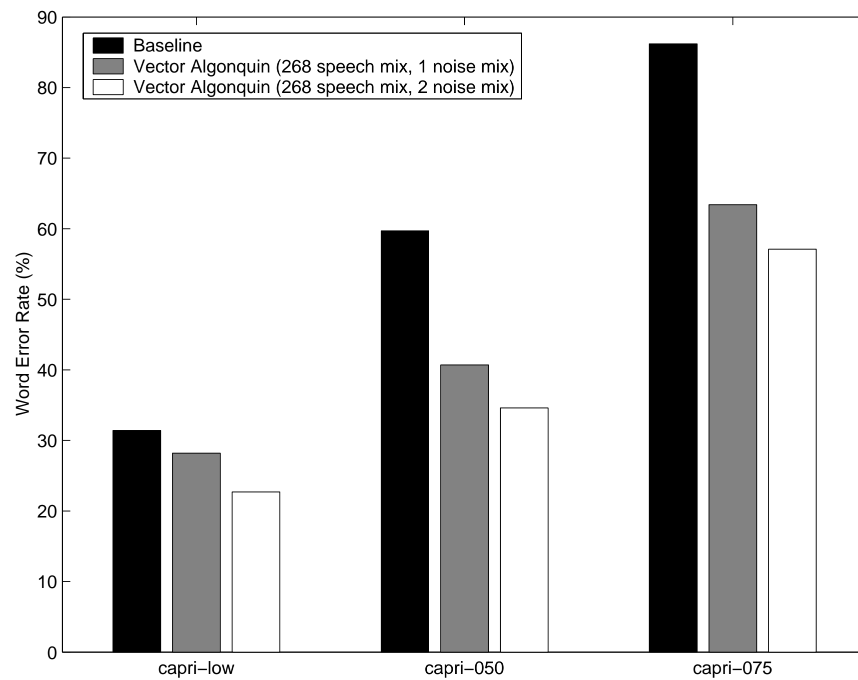


Figure 3: Word error rates of Algonquin methods on farfield data.

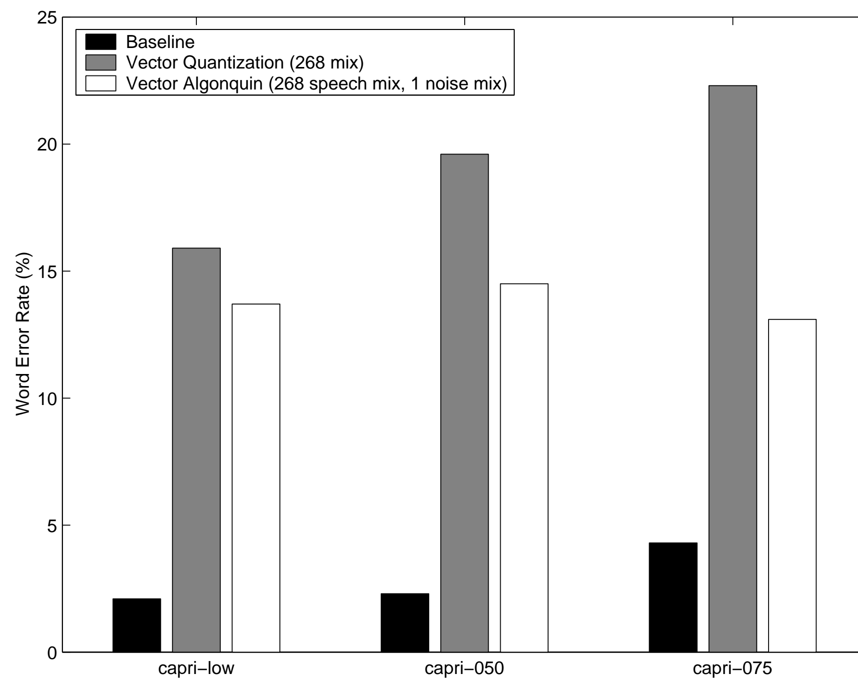


Figure 4: Word error rates of various methods on nearfield data.

# Conclusions

From the results, we observed that Algonquin can significantly reduce word error rates on the farfield data at all three noise levels. The level of degradation on the clean data is less than that observed for the vector quantization method with the same number of mixtures. When used with 2 noise mixtures, Algonquin reduces error rates by between 27 % and 42 %.

The primary disadvantage of Algonquin is the computation time required to denoise each utterance. In [4], Frey and Acero mention that their implementation takes 2 minutes to denoise a single utterance (from Wall Street Journal corpus with artificial noise) using a 256 mixture clean speech model, a single mixture noise model, and 5 iterations of Laplace's method. With our optimized implementation, we are able to denoise a single utterance in about 20 seconds using a 268 mixture clean speech model, a single mixture noise model, and 5 iterations of Laplace's method. The main computational bottleneck in the iterative update step is the logarithm computation.

A related disadvantage is that the computation times grows as the product of $N_n$ and $N_x$, the number of noise mixtures, and the number of speech mixtures. Thus, denoising using two noise mixtures takes twice as long as with a single mixture.

At present, the amount of time required to fully denoise each utterance makes it impractical to use this algorithm with a deployed system. However, using one of the variations of Algonquin mentioned in the previous section, the computation time may be reduced significantly (in particular, staggered computation may divide the computation time by the update step).

# Appendix - Listing of Code modules

## Feature module

### Algonquin Module

`/feature/src/module/algonquin_fromwave.c` (requires coretech)
This feature module performs takes in power spectral coefficients (`swimfcc_ps_powerspec`) and output filterbank coefficients that have been denoised with Algonquin. Due to the noise model training involved, code from `coretech` in the `train` model is required. In the feature.txt file, required options include

- `error_covariance` - The file containing the error covariance vector, $\Psi$.

  ```
  /users/apark/nearfield/near_far_evar_vector.x
  /users/apark/nearfield/near_near_evar_vector.x
  ```

- `speech_model` - The file containing the clean speech GMM.

  ```
  /users/apark/nearfield/models/nearfield.combined.268.mix
  /users/apark/nearfield/models/nearfield.combined.524.mix
  /users/apark/nearfield/models/nearfield.combined.1036.mix
  /users/apark/nearfield/models/dimensional_single_gmm.combined36.mix
  ```

- `num_iter` - The number of iterations to make in estimating $\eta_s^{(i)}$ and $\mathbf{\Phi}_s^{(i)}$. Due to a small glitch, this number must actually be one more than the real number of iterations desired. (i.e., if 5 iterations are desired, then `num_iter` = 6.

- `noise_mix` - The maximum number of noise mixtures to use in modelling the noise. Note that if `train_mdg` returns fewer mixtures than `noise_mix`, then the smaller number is used to avoid a segmentation fault.

### Quantization Module

`/feature/src/module/quantize_dim.c`
This feature module takes in filterbank coefficients (log-spectra) and computes the quantized MMSE using a dimensional or vector clean speech GMM model. This code is dependent on
`/feature/src/module/mdgd_functions_feature.h`
`/feature/src/module/mdgd_functions_feature.c`

### Quantile Noise Estimate

`/feature/src/module/quantile_process.c`
This feature module computes a point estimate of the noise on 120 power spectral coefficients for each utterance. This processing was only used for pre-computing the noise feature cache when calculating $\Psi$.

## rec_test module

`/rec_test/src/acc_test.h` ⟨modified⟩
`/rec_test/src/acc_test.c` ⟨modified⟩ We modified the `acc_test` code to allow for recognition tests from a feature cache. To utilize this option, the line

`feature_cache` ⟨`feature_cache_location/cache_filename`⟩

at the top of the recognition script. Currently, this module is coded to expect input features of the type `swimfcc_filterbank` only.

## Tools

For each of the following tools, command line options and arguments are explained in the usage statement for each tool.

### GMM Training Tools (Vector)

`/train/src/tools/get_cache_from_alignments/get_single_gmm_cache_from_alignments.c`
`/train/src/tools/train_from_cache/train_single_gmm_from_cache.c`
`/train/src/tools/train_from_alignments/train_single_gmm_from_alignments.c`
`/train/cmd/TrainSingleGMM.pl`

`/train/src/tools/train_from_alignments/compute_error_variance.c`
`/train/src/tools/train_from_alignments/combine_single_gmm.c`

### GMM Training Tools (Dimensional)

`/train/src/tools/train_from_cache/train_single_gmm_from_cache_by_dim_fast.c`
`/train/src/tools/train_from_cache/mdgd_functions.c`
`/train/src/tools/train_from_cache/mdgd_functions.h`

# References


[1] B.J. Frey, T. Kristjansson, L. Deng, and A. Acero. Learning dynamic noise models from noisy speech for robust speech recognition. In *Advances in Neural Information Processing Systems*, 2001.

[2] T. Kristjansson, B.J. Frey, L. Deng, and A. Acero. Joint estimation of noise and channel distortion in a generalized EM framework. In *Proc. ASRU*, 2001.

[3] A. Acero, L. Deng, T. Kristjansson, and J. Zhang. HMM adaptation using vector Taylor series for noisy speech recognition In *Proc. ICSLP*, Beijing, 2000.

[4] B.J. Frey, L. Deng, A. Acer, and T. Kristjansson. Algonquin: Iterating Laplace's method to remove multiple types of channel distortion from log-spectra used in robust speech recognition. In *Proc. Eurospeech*, Aalborg, 2001.

[5] V. Stahl, A. Fischer, and R. Bippus. Quantile Based Noise Estimation for Spectral Subtraction and Wiener Filter. In *Proc. ICASSP*, volume 3, pp. 1875-1878, 2000.