Analysis of knowledge requirements for text alignment problem

Bartosz Kalińczuk

August 25, 2013

Abstract

The purpose of this final master degree project was to experiment with various algorithms for speech and text alignment either with granularity of sentences, single words or even single phonemes. The output of this study was expected to find out how little data is necessary to compute a proper alignment. This project focuses mainly on Polish language, however it can be quite easily generalized for different languages. It also focuses solely on a audio with quite low level of noise, since it introduces a lot of problems, and is out of the scope of this project.

Contents

1 Introduction

2 Speech signal

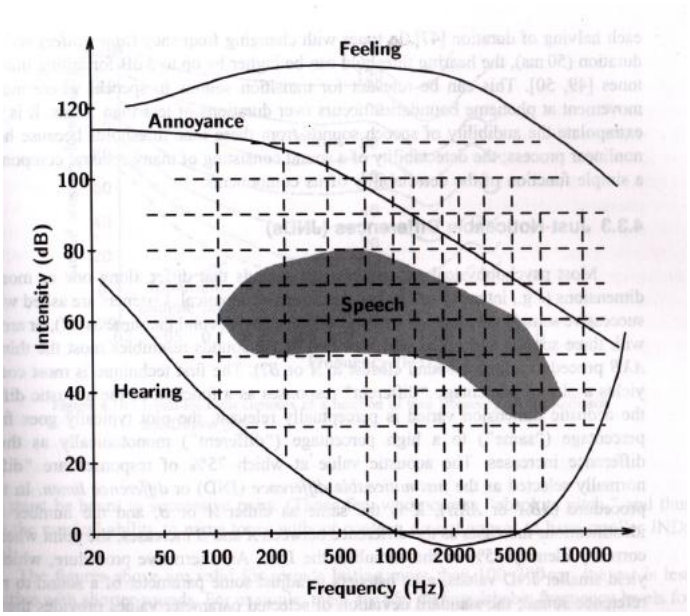
2.1 Human factor

Speech is a most efficient way the human communicate. For generations this process was refined by evolution, so we can easily exchange messages even in hard situations. For this purpose our vocal mechanisms must well cooperate with our hearing ability. There is a certain set of sounds we can produce and our ears evolved to hear them as well as possible.

What is sound? According to dictionary: "Vibrations transmitted through an elastic solid or liquid or gas, with frequencies in the approximate range of 20 to 20000 hertz, capable of being detected by human organs of hearing". [1]

How do we hear? Human ear consist about 30000 hair-cells, which can convert mechanical wave of the sound into electromagnetic wave inside auditory nerves [2]. Each of these cell is excited by different frequency of mechanical wave of internal ear fluids, so it is no surprise, that people can hear only a certain range of frequencies, as stated in definition. These we expect to be finely tuned to the range of the sounds we can produce. Although it seems, that we can hear a bit more, but as we don't need that, it happens, that as we grow older, our hearing range is getting smaller, because our hear cells fail sometimes, but mostly those responsible for high frequencies, which we don't use too often.

Humans can hear frequencies, that begins as low as 12Hz (under laboratory conditions) to 20kHz (for adults usually much lower). However speech range is a little bit smaller than that [3]:



2.2 Mel scale

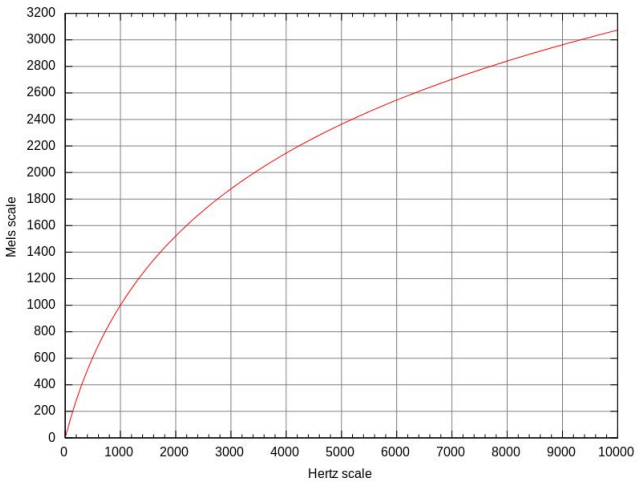
How we perceive sound, that is completely different matter and topic for long philosophical discussion. However we can help ourselves with some subjective experiments. For example Stevens, Volkman and Newman conducted an experiment on a number of listeners to measure, what do we perceive as equally distanced pitches. In this experiment, the participants of the experiment were asked to judge if given pitches were in equal distances. The output was, that humans don't experience sound linearly respectively to the frequency scale, but a perceptual scale was closer to logarithmic one. [5]

Certain formulas were conceived to translate frequency scale to one, that is closer to how human actually perceive sound.

One popular is mel scale, where mel comes from melody: [6]

$$m = 2595 log_{10}(1 + \frac{f}{700}) \tag{1}$$

What looks like that:



Another popular formula of so called bark scale, which is based on perception of loudness of the sound and proposed by Eberhard Zwicker in 1961. [7]

$$Bark = 13 atan(\frac{0.76f}{1000}) + 3.5 atan(\frac{f^2}{7500^2}) \tag{2}$$

In this project we use mel scale implemented in sphinx library, although bark scale is becoming more popular recently.

2.3 Frequency spectrum

The conclusion from the anatomy of human ear is, that frequencies of the sound are important. How can we obtain frequency spectrum from a digitized sound, so we can proceed further?
The obvious tool for conversion of discrete function to frequencies is Discrete Fourier Transform, named after Jean Baptiste Joseph Fourier it is one of the most often used techniques of modern times.

It all started from the postulate, that a heat equation can be satisfied by function of form: [11]

$$f(x) = \sum_{n=0}^{N} (A_n cos(nx) + B_n sin(nx)) \tag{3}$$

or in complex form:

$$f(\theta) = \sum_{n=-\infty}^{\infty} C_n e^{in\theta} \tag{4}$$

Basically we convert our function's domain to frequency domain or to domain of sinusoidal functions. C_n coefficients are complex values that encode both amplitude and phase of the converted signal/function at each frequency.

The coefficients for any integrable functions over an interval $[\frac{-T}{2}, \frac{T}{2}]$ can be obtain using formula: [11]

$$C_n = \int_{\frac{-T}{2}}^{\frac{T}{2}} f(x) e^{-2\pi i \frac{n}{T} x} dx \tag{5}$$

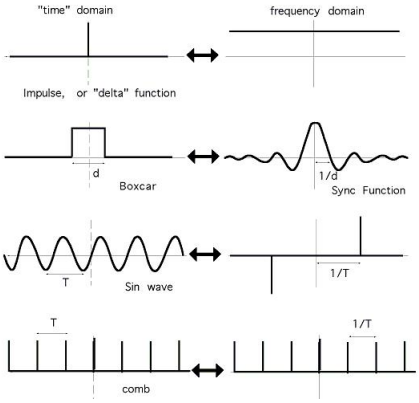
or for the discrete case:

$$C_k = \sum_{n=0}^{N-1} x_n e^{\frac{-2\pi i k n}{N}} dx \tag{6}$$

So far we haven't found any use in the speech recognition for phase part of the coefficients, however amplitude determines how powerful is signal at given frequency. The power value is given by:

$$|X_k|/N = \sqrt{\Re(X_k)^2 + \Im(X_k)^2}/N \tag{7}$$

A sample conversion:

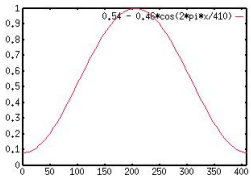


What size of the window should we use? First we have to notice, that in order to capture certain frequency, the window needs to be large enough. We would like to examine signals of frequency ranged from 100Hz (see speech frequencies ranges in chapter 2.1), which is a period of 100th of the second, so a 10millisecond window would be our bottom limit.

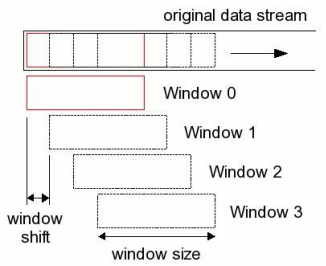
Also windows with abrupt signal discontinuities may cause result with spectral artefacts, so a windowing function is usually applied. Popular choice is a Hamming window function: [8]

$$w_j = 0.54 - 0.46cos(\frac{2\pi j}{W-1}) \tag{8}$$

Which's plot is:

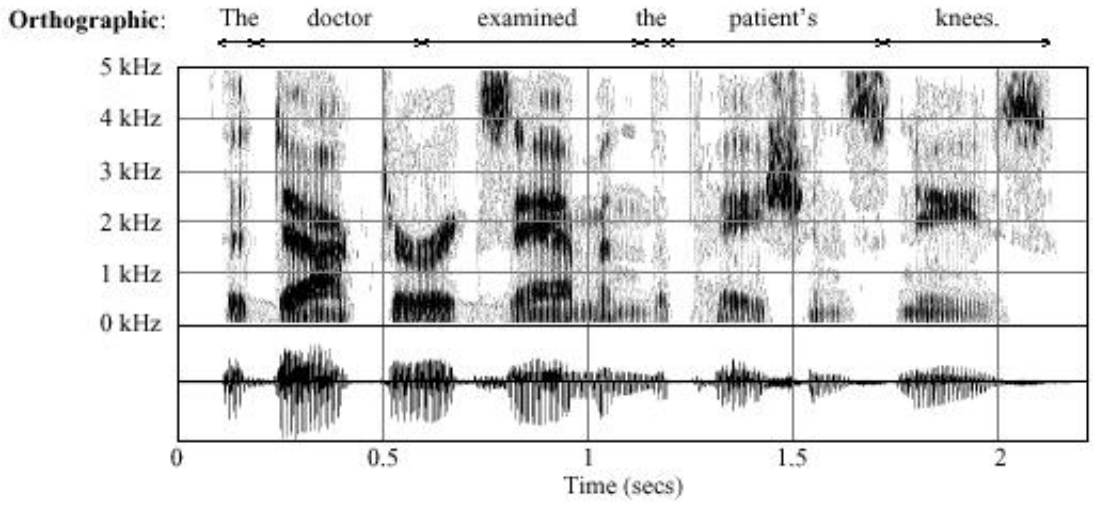


Note that it emphasises values in the middle of the window, so our actual windows should overlap to cover whole time domain. For example by shifting a window by a percentage of it actual width:

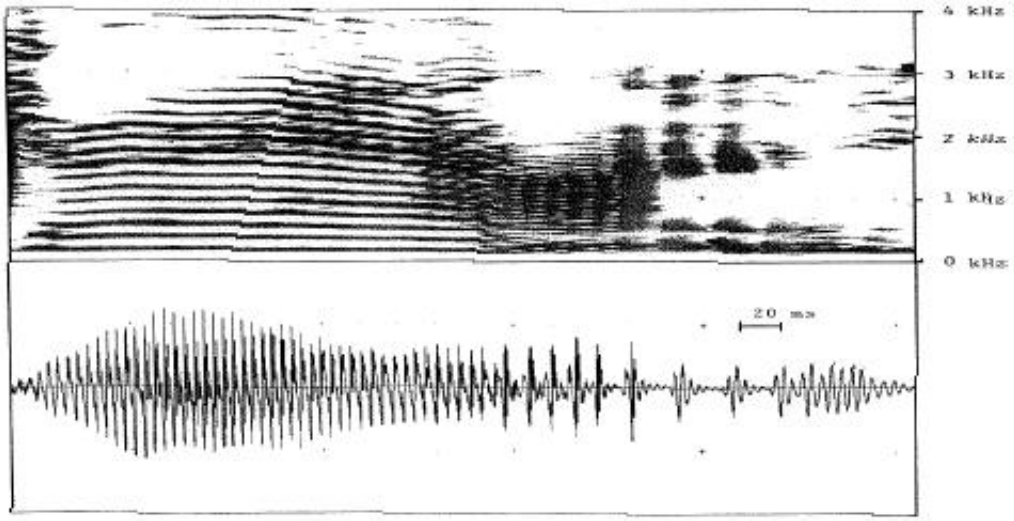


A human speech signal in frequency and time domain: [3]

Standard wideband spectrogram ($f_s = 10\ kHz$, $T_w = 6\ ms$):



Narrowband Spectrogram ($f_s = 8\ kHz$, $T_w = 30\ ms$):

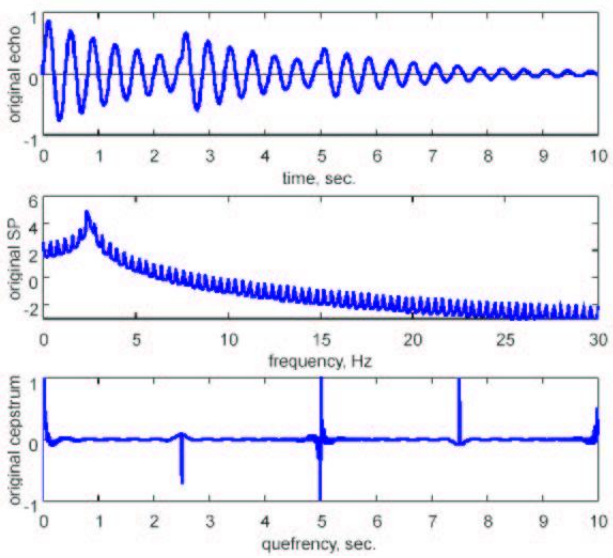


"Drown" (female)

2.4 Cepstrum

Looking at the frequency spectrum of human speech we see, that the signal in the frequency domain contain features that are quite periodic. As it is with converting initial signal with DFT, we would like to extract the information of periodicity in the spectrum. A cepstrum of the signal gives us this additional information.

The word is derived by reordering characters in the word spectrum to indicate switch of domains, similarly as word 'quefrency'. The cepstrum operates in the domain of time and the basic intuition is, that it reveals a rate of change in the different spectrum bands. For example a cepstrum of an echoed signal in the picture below shows clearly a three 'quefrencies' of the echo of the signal. [12]



Cepstrum definition is: "Inverse Fourier transform of the logarithm of the magnitude of the Fourier transform" or:

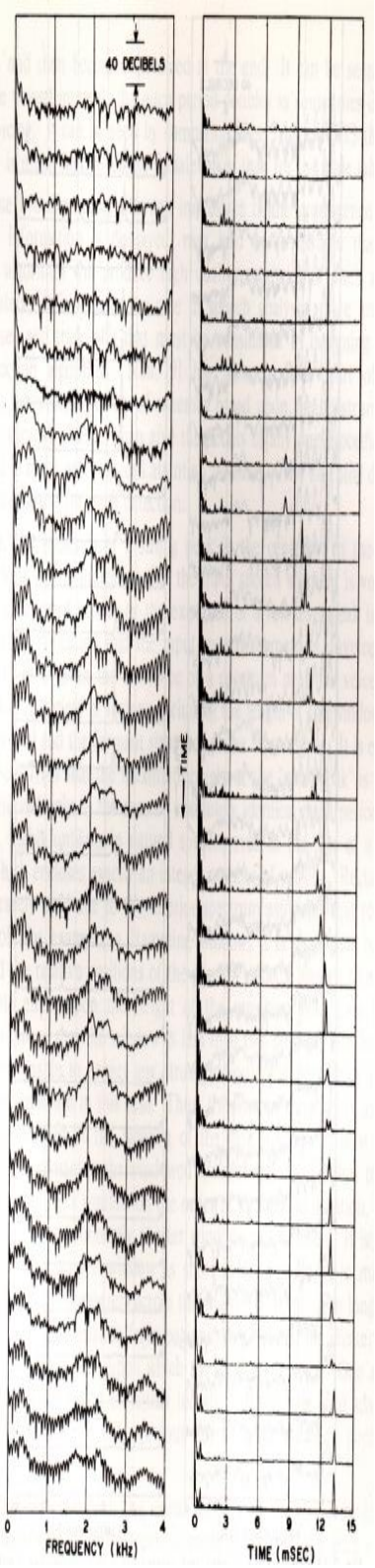
$$C = |F^{-1} log(|Ff(t)|^2)|^2 \tag{9}$$

,or:

$$c_x[n] = \frac{1}{2\pi} \int_{-\pi}^{\pi} log|X(e^{j\omega})|e^{j\omega n} d\omega \tag{10}$$

This is the definition of the power cepstrum, since it is calculated from the magnitude of each frequency band. However there also exists a complex, real and phase cepstrum depending on what part of initial Fourier transform it uses. In speech related problems a power cepstrum is usually used and I haven't see any reason to not focus only on this.

This is a typical cepstrum sequence of the vowel [3] computed every 10ms.



If the sound becomes periodic in the frequency domain it's quefrency domain contains a peak which is related to the periodicity of the sound.

Note that similar results can be obtained by taking just additional DFT of the signal. Inverse Fourier Transform is closely related to Fourier Transform and also performs a split of the function into periodic components.

After all IFT is defined:

$$f(x) = \int_{\mathbb{R}^n} e^{2i\pi x\zeta} \hat{f}(\zeta) d\zeta \tag{11}$$

while FT is defined:

$$\hat{f}(\zeta) = \int_{\mathbb{R}^n} f(x) e^{-2i\pi\zeta x} dx \tag{12}$$

Why taking logarithm of the magnitude? It serves as a normalization of power spectrum. In speech for example it happens, that low frequency components are usually more powerful than high frequncy components and by normalizing the signal, the periodicity becomes more apparent.

A bit different way of looking at the signal cepstrum is as a homomorphic transformation which changes convolution into sum. [3]

$$x(n) = e(n) * h(n) \tag{13}$$

$$\hat{x}(n) = \hat{e}(n) + \hat{h}(n) \tag{14}$$

Which on it's own can be seen as way of separating signals, since it is more easy to extract elements from a sum, than from a convolution.

In the example with echo, we could have used the cepstrum to separate echoed signal from initial signal, and it might be used to filter out an audio feedback.

2.5 Sphinx frontend

Sphinx is a speech recognition toolkit with a lot of useful functionalities for any speech related problem.
There is a certain common way to prepare a speech signal for the further processing. With slight variations in each step, the useful informations about speech are drawn from a cepstrum of the reduced signal (in the number of data dimensions), as presented in this chapter.

In order to skip the reinvention of the wheel, I used the fronted part of the sphinx library in any experiment in this project. The sphinx fronted performs signal transformation and produces data composed of only 39 voice features, while actually only 13 are base ones and the rest is a derivation of these.

Sphinx frontend is a list of transformations executed on the result of the transformation placed higher in the list. In another words it is a transformation composition.

This Sphinx frontend pipeline includes:

- Data Blocker,
- Preemphasizer,
- Windower,
- Discrete Fourier Transform,
- Mel Frequency Filter Bank,
- Discrete Cosine Transform,
- Cepstral Mean Normalization,
- Deltas Feature Extractor.

2.5.1 Data Blocker

This initial transformation reads incoming double data read from audio source (file or microphone) and prepares blocks of the data to be used in later phases. In our case blocks contain 10ms of audio data.

2.5.2 Preemphasizer

The Preemphasizer applies a formula: $Y[i] = x[i] - (X[i-1] * preemphasizerFactor)$. The purpose of this transformation is to emphasize the high frequency components. It is kind of filter, which allows high frequency components to pass through, but weakens the low frequency ones.

2.5.3 Raised cosine windower

Creates windows from the incoming data. A windowing function

$$W(n) = (1 - \alpha) - \alpha cos(\frac{2\pi n}{N-1}) \tag{15}$$

is applied afterwards. Alpha coefficient set to 0.46 results with a mentioned before Hamming windowing function, which is a default setting and the one used by me.

2.5.4 Discrete Fourier Transform

Implementation of fast Fourier transform .The FFT can perform transformation with complexity $\Omega(Nlog(N))$, where N is the size of the input data. It can be perform on whole data, however in speech we would like to get an information of the frequencies of a small frame, that contains consistent speech signal, in particular a single phoneme. The output data is the power spectrum of input data window and the complex/phase information is lost. The number of FFT points is the closest power of 2 equal or larger to the number of samples in the incoming window of data. However the input signal is real, so resulting FFT is symmetric, so only half of the data is returned and the output size is $\frac{FFTpoints}{2} + 1$.

2.5.5 Mel frequency filter bank

This step is a part of calculating a Mel Frequency Cepstrum.

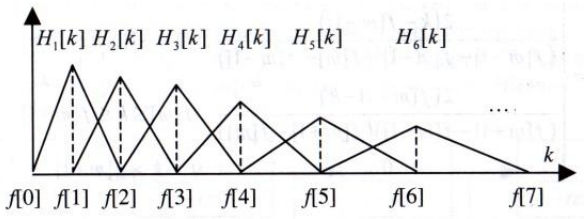
Conversion of frequency spectrum into a mel-spectrum using triangular overlapping windows defined as:

$$w(n) = 1 - |\frac{n - (N-1)/2}{(N+1)/2}| \tag{16}$$

The number of triangles/filters defined the size of mel-spectrum and the sphinx's default is 40.
The filters are chosen, so the result would simulate a mel-scale given by the formula:

$$melFreq = 2595log(1 + linearFrequency/700) \tag{17}$$

The given filters should look like in the picture:



Not all frequencies are covered by the filters. The chosen range of frequencies may differ for various audio encodings, but generally should cover only the speech ranges. The default values for 16kHz sample rate streams are 130Hz-6800Hz and are not changed in this project.

2.5.6 Discrete Cosine Transform

Another part of calculating Mel Frequency Cepstral Coefficient vector.

It applies a logarithm and the DCT type II to the input data.

A DCT type II (most common) coefficients are defined as:

$$C(u) = \alpha(u) \sum_{x=0}^{N-1} f(x) cos[\frac{\pi(2x+1)u}{2N}] \tag{18}$$

and it is quite tightly related to real part of the Fourier Transform. [18] The transform represents a function as a sum of cosine functions and it is equivalent to the DFT operating real data with even symmetry.
The number of dimensions returned is set by default to 13.

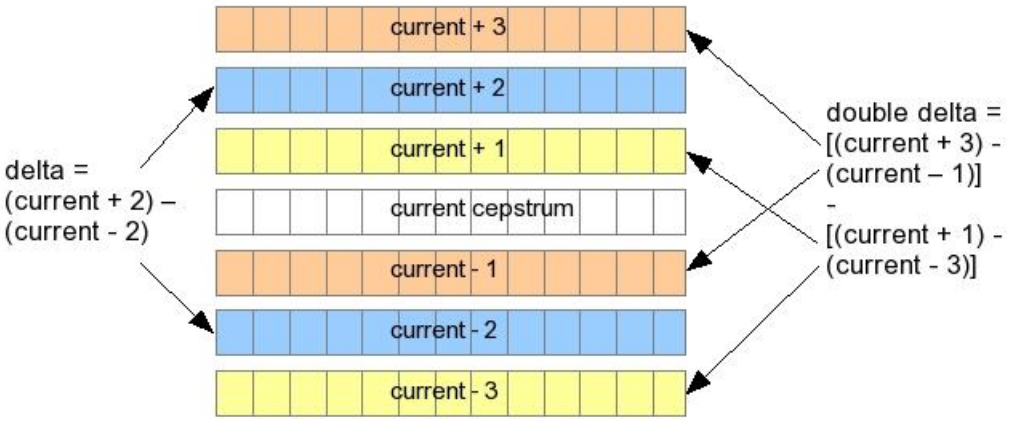
2.5.7 Cepstral Mean Normalization

Performs a normalization of MFCC vector by subtracting a mean of all the input. There are two versions of this step. One that calculates mean online and the other that reads all data before performing subtraction.

2.5.8 Deltas feature extractor

The final transformation in the sphinx frontend chain. It calculates first and second order derivative of the cepstrum as additional features of the speech signal. It improves noticeable speech processing algorithms by adding additional information about changes in the cepstrum data.

For the initial cepstrum data it adds additionally twice the size vector with first and second order differences, calculated as shown in the picture:



3 Speech Modelling

3.1 Phones, phonemes and graphemes

A phone is a unit of speech sound [20]. Phoneme's definition is: "The smallest contrastive linguistic unit, which may bring about a change of meaning" [21], so the phoneme is a classification unit of phones, which allow us to represent speech while preserving its meaning. While speech is being modelled using phonemes the graphical part of the language in form of text is modelled with characters or graphemes.

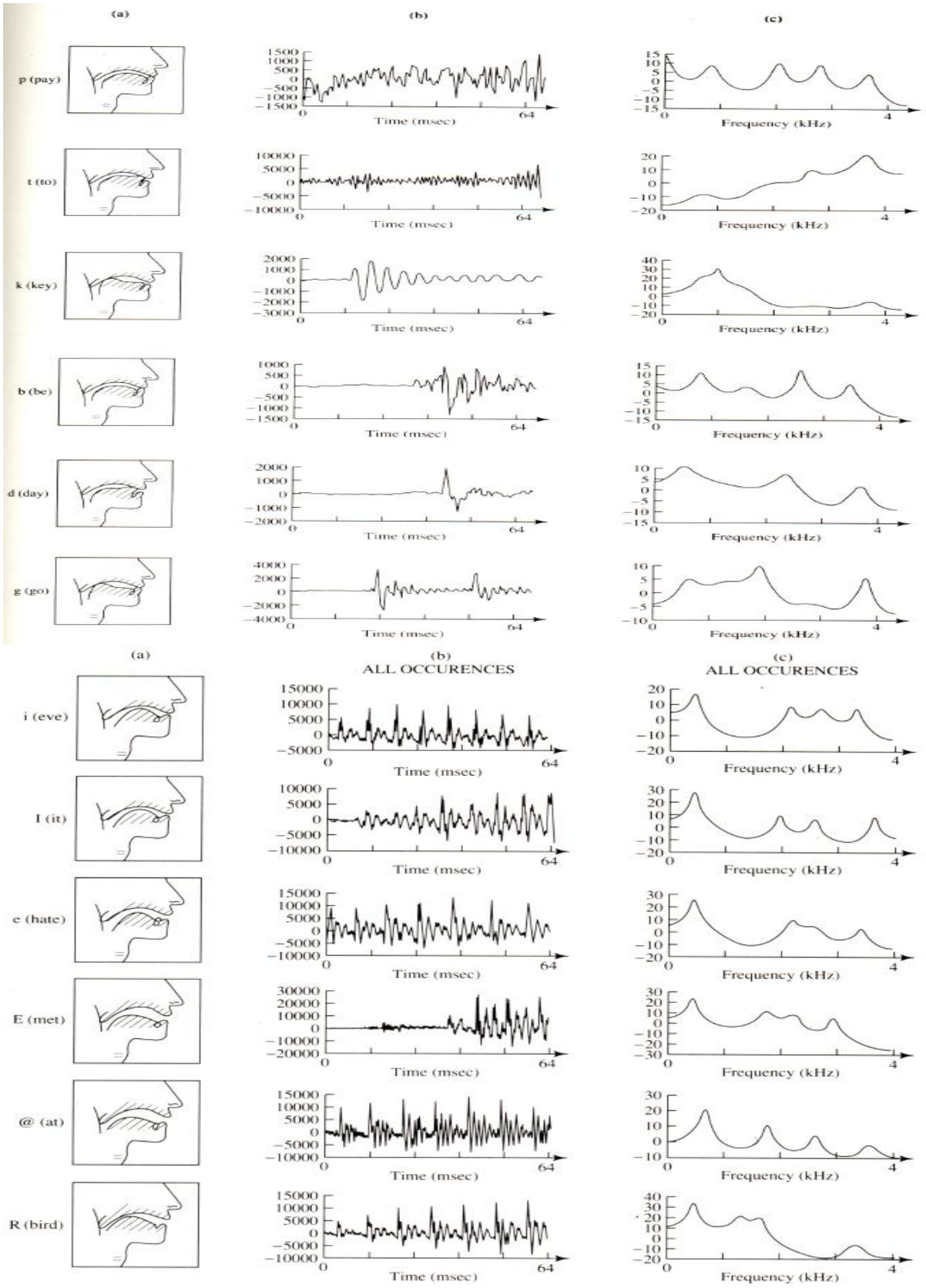
Language grapheme set usually differs quite substantially from it audio counterparts. Often it contains more characters than needed to represent every word from given language and at the same time it is much too small to represent all the nuances of human speech. What is more problematic, the word graphic representation often has very little to do with actual phones of the word. I.e. there are so called homographes: words, that are written the same yet they pronunciation differs ("zamarzać" from "marznąć" and "morzyć") or homophones, that look differently, but are pronounced similar ("może", "morze"). In polish though, the former is quite rare and this fact is actually used by me (chapter 5.3).

Actual phones that are classified under single phoneme create a diversified family. Different variants of a phoneme are called allophones. For example /l/ in English "leap" and "deal" or polish examples of allophones (/ł/ in "umysł" might be soundless contrary to "ławka") or vowels between soft consonants (/a/ in "jajko"). [22]

The phonemes can differ quite substantially depending on the surrounding phones. For example almost each phoneme in polish changes to softer version when put next to /i/ or /j/. Consecutive phones are not necessarily separated by clearly visible moment of silence. Often one phone is converting slowly into another. To model such a transitions a diphones or triphones are modelled for each sequence of two or three phones.

Phonemes are very important in the computational language modelling, either in speech recognition or alignment. The importance is derived directly from its definition. It is a unit of speech, which can't be switched to another without changing the meaning. This is the unit, that need to be modelled if we want to recognize and/or distinguish different words. Finer granularity of model is necessary only to make a better prediction where an observed phone belongs.

Some of English phonemes and example phones:



3.2 Audio distances

The simplest way to find similar audio sequence is to find a sequence which is nearby to another, that we know represents a certain sound, phoneme or word.

To calculate a distance we could use various norms:

$$||x||_1, ||x||_2, ..., ||x||_\infty \tag{1}$$

, where

$$||x||_k = (\sum_{i=0}^{N} x_i^k)^{\frac{1}{k}} \tag{2}$$

and they are all fine for uncorrelated vectors, which is not really our case.

For correlated vectors, we could try to introduce some weighting factor inside.
What factor should we use?
One approach is to tune the factors using external methods, which theoretically may give us some additional benefit of properly modelling phonemes, that we try to measure distance from, however this is a bit out of the scope of this chapter and most probably would in the end look similar to a different method. A simpler approach would be to calculate correlations and use them in a distance measure. If we had a correlation matrix (P) and than our distance could be:

$$dist_{usingcorrelations}(\vec{x}, \vec{p}) = (\vec{x} - \vec{p})^T P^{-1} (\vec{x} - \vec{p}) \tag{3}$$

Karl Pearson introduced such an idea [23] in form of correlation coefficient defined between two random populations:

$$\rho_X Y = \frac{cov(X, Y)}{\sigma_X \sigma_Y} = \frac{E[(X - \mu_X)(Y - \mu_Y)]}{\sigma_X \sigma_Y} \tag{4}$$

and in the form of matrices:

$$P = (\Sigma^{diagonal})^{-1/2} \Sigma (\Sigma^{diagonal})^{-1/2} \tag{5}$$

It should be noted, that in the denominator we have standard deviations which don't really bring any value to our measure, since this is a constant factor. By removing them we obtain so called Mahalanobis distance [16]:

$$distance_{Mahalanobis}(\vec{x}, \vec{p}) = (\vec{x} - \vec{p})^T \tilde{C}^{-1} (\vec{x} - \vec{p}) \tag{6}$$

If we knew elements belonging to any given phoneme, we could calculate a distance between this training sample and any encounter speech signal.

I conducted couple of experiments with different distances. Starting with a flawed alignment of larger portions of text I tried to:

- find the same word, that is quite lengthy and occurs multiple times just by searching for similar sequences,
- find a given sequence of three phonemes in a text, based on estimated location (time)

Mahalanobis distance in those experiments where not expected to any give significant results, since there were conducted without knowledge of phone classification and it behaved without a surprise.

The euclidean norm were performing the best.
In the second experiment it was able to find around 50% of all occurences of three phoneme sequence from beginning of the text and other found were quite similar (80% of the time they contained two out of three phonemes), although I was lucky, that in my testing recording, the starting three phonemes occurrences later in the text were quite far from each other.

Searching for whole word didn't give me any satisfying results. I could find a similar word when I pointed, which it should have being searched for, but without the intervention it always found a sequences which weren't similar at all (from a speech point of view) and at the same time the matching words, I was expected to find, were far in the list.

3.3 Gaussian Model

Given observation points belonging to a single class (a phoneme) with a given probability, we would like to model a distribution of emitting data point by the class.

A natural choice is a normal distribution, although we have to remember, that observation comes from a multidimensional universum, where populations are not independent. Luckily there is a definition of multivariate normal distribution, that considers correlations:

$$f_x(x_1, ..., x_k) = \frac{1}{\sqrt{(2\pi)^k |\Sigma|}} exp(-\frac{1}{2}(x-\mu)^T \Sigma^{-1}(x-\mu)) \tag{7}$$

where Σ is a covariance matrix and $|\Sigma|$ is its determinant.

Covariance matrix is always symmetric and positive-semidefinitive.
Symmetry comes directly from a definition: $cov(X) = E[(X - E(X))(X - E(X))^T]$, since outer product of a single vector gives always a symmetric matrix.

Positive-semidefinitive matrix is a matrix, where for any product $a^T Aa$ with any non-zero complex vector a is real and non-negative:

$$a^T Aa >= 0 \tag{8}$$

A product with any vector a and covariance matrix is also equal to:

$$a^T \Sigma a = a^T E(XX^T)a + a^T \mu\mu^T a = \frac{1}{N}(\sum_{i=1}^{N} a^T XX^T a) + a^T \mu\mu^T a \tag{9}$$

and each element of the sum is square of inner product of two vectors, so it is always positive (or equal to zero).

In order to prevent degenerate cases we can allow only positive-definite matrices.
Any such a matrix is guaranteed to be invertible.

If the number of dimensions is equal to one, then the formula reduces to single-variable normal distribution:

$$f(x) = \frac{1}{\sqrt{(2\pi)}\sigma} e^{-\frac{(x-\mu)^2}{2\sigma^2}} \tag{10}$$

A normal distribution of emitting signal frame have two free parameters: a mean vector and a covariance matrix, which needs to be calculated.
I assume, that an input is the list of observations with assigned probability.

If probability is actually a likelihood of emitting the signal (or its estimation), than we can calculate mean with a formula:

$$\vec{\mu} = \sum_{\vec{X}} Pr(\vec{X})\vec{X} \tag{11}$$

and a covariance matrix by:

$$\hat{C} = \sum_{vecX} Pr(\vec{X})(\vec{X} - \mu)(\vec{X} - \mu)^T \tag{12}$$

In the case, that probability is not a direct likelihood of given point, but a conditional probability of emitting the signal, (i.e. under the condition that it belongs to given sequence), the probabilities need to be normalized first.

We can assume, that conditional probability is the same for each observation, so the input probability is in the form of $Pr(\vec{X})Pr_{condition}$, then they have to be divided by a total sum to produce actual likelihood:

$$Pr(\vec{X}) = \frac{Pr_{input}(\vec{X})}{\sum_{\vec{X}} Pr_{input}(\vec{X})} = \frac{Pr(\vec{X})Pr_{condition}}{\sum_{\vec{X}} Pr(\vec{X})Pr_{condition}} = \frac{Pr(\vec{X})}{\sum_{\vec{X}} Pr(\vec{X})} \tag{13}$$

The denominator should sum to 1, after all it is a probability of emitting given point under a condition, that only $|X|$ points were emitted.

3.4 Expected-Maximization algorithm

Expected-Maximization method is a technique for estimating parameters of any underlying distribution based on observed data. It tries to maximize the likelihood, that the data would be observed by the distribution:

$$argmax_\theta Pr(X|\theta) \tag{14}$$

For certain distributions, parameters, that maximize likelihood of the data, can be solved with analytic methods, i.e. calculated mean vector and variance are parameters to normal distribution, that do maximize the likelihood of observing the training data. We can calculate a derivative of normal density function to show it:

$$\begin{aligned} ln(\prod_{xinX}(\frac{1}{\sqrt{2\pi\sigma^2}}e^{\frac{-(x-\mu)^2}{2\sigma^2}}))d\mu &= \sum_{x\epsilon X}[\frac{-1}{2}ln(2\pi) - ln(\sigma) - \frac{(x-\mu)^2}{2\sigma^2}]d\mu = ... \\ ... &= \frac{-1}{2\sigma^2}\sum_{xinX}(x-\mu)^2 d\mu = \frac{1}{\sigma^2}\sum_{xinX}(x-\mu) = 0 \iff \\ &\iff \sum_{xinX}(x-\mu) = 0 \iff \mu = \frac{1}{|X|}\sum_{xinX} x \end{aligned} \tag{15}$$

what is a definition of a mean.

$$\begin{aligned} ln(\prod_{x\epsilon X} Pr(x|\sigma))d\sigma &= \sum_{x\epsilon X}[\frac{-1}{2}ln(2\pi) - ln(\sigma) - \frac{(x-\mu)^2}{2\sigma^2}]d\sigma = ... \\ ... &= \sum_{x\epsilon X}[\frac{-1}{\sigma} + (x-\mu)^2\sigma^{-3}] = \frac{-1}{\sigma}\sum_{x\epsilon X}[1 - (x-\mu)^2\sigma^{-2}] = ... \\ ... = 0 \iff \sigma^{-2}\sum_{x\epsilon X}(x-\mu)^2 - |X| = 0 &\iff \sigma^{-2}|X| = \sum_{x\epsilon X}(x-\mu)^2 \iff \sigma^2 = \frac{1}{|X|}\sum_{x\epsilon X}(x-\mu)^2 \end{aligned} \tag{16}$$

what is a definition of variance and a standard deviation is a square root of variance. And similarly can be done for many other distributions, including multivariate normal distribution.

It is not always the case, that one can calculate parameters so easily, i.e. mixture models of several populations may not give up so easily. Let's consider a mixture of Gaussian models of some populations. We have a random population, where each point is randomly drawn from each distribution. So the total likelihood of any point is: $\sum_{i=0}^{N} p_i f_i(x)$, where p_i is a probability of drawing a point from ith distribution and f_i is a density function of ith model.

Since each model from a mixture is easily solvable, if we knew to which model each point belonged, than estimating parameters would be easy, or at least if we knew what is the probability, that given point was drawn from each given class (see chapter about Gaussian model).
On the other hand it would be simple to calculate a probability, that a point was drawn from some distribution if we knew all the parameters of all models.

The Expected-Maximization technique deals with this problem, by finding better parameters using their previous estimation, and thus by iterating over series of converging estimates it is guaranteed to find some local maximum.

$$Q(\Theta^i, \Theta^{i-1}) = E[logPr(\chi, \Upsilon|\Theta)|\chi, \Theta^{i-1}] \tag{17}$$

, where Υ is an unknown data, which can be estimated using Θ^{i-1}, and when known, then Θ^i can be found, by find the parameters, which maximize log likelihood of observing random variables χ and Υ.

Thus the EM algorithm contains two steps in single iteration:
expectation step and maximization step.

- During E step, we find Υ data given previous estimate of Θ.
- During M step, we calculate Θ, that maximizes likelihood of observed data and expected hidden data.

In each iteration a likelihood $Q(\Theta^i, \Theta^{i-1})$ converges to some local maximum.

We are happy with only local maximum, because the problem resists our efforts to solve it analytically.

For example a mixture model can be trained using following steps:

- In E step we calculate a probability, that a observation is drawn from each class.
- In M step we use this probabilities to calculate a new parameters ($\{(\mu_i, \sigma_i, p_i)\}$), that maximize likelihood of our observed data, as well as an estimated probability of data classification.

3.5 Hidden Markov Model

We can describe an HMM by a triple:

$$\lambda = (A, B, \pi) \tag{18}$$

where A is a transition matrix $A = \{a_{ij}\} = p(Q_t = j | Q_{t-1} = i)$,
B is a observation probability function vector $B = \{b_i\}$, where each b_i is a function calculating likelihood that a given observation Q_t is produced by state i,
and π is an initial state distribution $\pi_i = P(Q_1 = i)$

For our purpose a B functions will be a Gaussian multivariate distribution of observations emitted by a state.

The most probable state sequence for given sequence of observations can be calculated using dynamic programming (i.e. Viterbi algorithm).
The algorithm iterates over over the discrete time indexes $t_1, ..., t_n$, where at each moment only one observation Q_{t_k} is emitted.

The k-th iteration produces a vector o probabilities $P_k = [p_1, ..., p_m]$ of the best state sequence ending at a state i at the moment t_k. For the initial moment the vector is equal to state intial probabilities π.

The P_{k+1} is calculated as follows:

$$P_{k+1,i} = max(P_{k,j} a_{j,i} b_i(Q_{k+1})) \tag{19}$$

where $a_{j,i}$ is a probability of transition from state j to state i,
and $b_i(Q_{k+1})$ is probability, that state i emitted observation Q_{k+1}.

At the end a maximum probability from elements of P_n gives us a probability of observing the sequence with the maximum likelihood for given sequence of observations.

To find actual sequence of states, we can keep a state for which a maximum was produced for each moment t_k and state i and recreate the maximum likelihood path.

3.6 Baum Welch algorithm

To train Hidden Markov Models we have to use a generalized version of EM algorithm, namely a Baum-Welch algorithm.
In the training of HMM we have observed data X and we want to find parameters set θ, which will maximize the probability of observing X, meaning:

$$argmax_\theta(Pr(X|\theta))) \tag{20}$$

If we knew what was the sequence of states in the HMM, we would be able to calculate optimal value of θ parameters. However we don't know, what the states of HMM were, hence hidden in the name. On the other hand if we knew θ, then we could easily calculate sequence of states, which would maximize the probability of emitting input observations (i.e. using Viterbi algorithm).

EM technique is meant for such a situations.

In the EM spirit, for each iteration we will perform two steps, bringing us to some local maximum:

expectation step Given previous estimation of parameters θ, we calculate the probabilities of being at any time t and at any state i: $Pr(s_i, t)$

maximization step Given probabilities of being at any state i, we calculate next estimation of θ parameters, which will maximize the likelihood of observing X: $argmax_\theta(Pr(X|\theta))$.

How can we calculate $\theta = \{A, B\}$ parameters?
Where:

A = transition probabilities (probability of transition between any two states)

B = observation probabilities (probability of observing any data at any given time)

To calculate observation probabilities, we need:

$$\alpha_i(t) = Pr(being\ after\ t\ steps\ at\ state\ i) \tag{21}$$

$$\beta_i(t) = Pr(ending\ sequence|being\ after\ t\ steps\ at\ state\ i) \tag{22}$$

Both of these values can be calculated using dynamic programming. One is calculated by Viterbi's algorithm in forward passage, the other can be calculated in similar manner by backward passage.

Combining these values, we can obtain:

$$Pr(being\ at\ time\ \boldsymbol{t}\ at state\ \boldsymbol{i}) = \frac{\alpha_i(t)\beta_i(t)}{\sum_{j=1}^{N}\alpha_j(t)\beta_j(t)} \tag{23}$$

and:

$$Pr(transition\ between\ states\ \boldsymbol{i}\ \boldsymbol{j}\ at\ time\ \boldsymbol{t}) = \frac{\alpha_i(t)a_{ij}\beta_j(t+1)b_j(o_{t+1})}{\sum_{i=1}^{N}\sum_{j=1}^{N}\alpha_i(t)a_{ij}\beta_j(t+1)b_j(o_{t+1})} \tag{24}$$

Probability of transition from states i to state j at any given time is:

$$Pr(transition\ \boldsymbol{i}\ \boldsymbol{j}) = \frac{\sum_{t=1}^{T-1} Pr(transition\ between\ states\ \boldsymbol{i}\ \boldsymbol{j}\ at\ time\ \boldsymbol{t})}{\sum_{t=1}^{T-1} Pr(being\ at\ time\ \boldsymbol{t}\ at\ state\ \boldsymbol{i})} \tag{25}$$

The probability of being at time t at state i can be used to calculated new probabilities of emitting observation o_t at time t by a state i, since it is emitted by the state with a probability of being at this state at this time.

New parameters of multivariate normal distribution would be:

$$\vec{\mu} = \frac{1}{T}\sum_{t=1}^{T} Pr(being\ at\ time\ \boldsymbol{t}\ at\ state\ \boldsymbol{i})\vec{o}_t \tag{26}$$

$$\tilde{C} = \frac{1}{T}\sum_{t=1}^{T} Pr(being\ at\ time\ \mathbf{t}\ at\ state\ \mathbf{i})(\vec{o}_t - \vec{\mu}) \cdot (\vec{o}_t - \vec{\mu})^T \tag{27}$$

4 Simple pause and length based alignment

The basic idea behind this approach is to match sentences with continuous sequence of speech. Humans rarely make pauses inside a sub-sentence and rarely continue to another sentence without a pause.
One simple approach to the alignment problem, which utilizes this fact, is to match part of speech with a portion of a text, which would take a similar time to say it.

4.1 Speech Detection

Before we can continue with an alignment, we need to detect pauses first or dually we need to detect speeches.

On its own in various environments the problem is quite hard, however we don't want to consider situations where extracting speech from background is too difficult. It is true, that humans are quite proficient at extracting speech from quite challenging situations like recognizing words of the song, or distinguishing speakers in a crowd. However even humans are not perfect, and are often prone for errors. Recognizing song lyrics is not always an easy task, and it remains an open question how much you can attribute the difficulty of this to the background music, how much to changed modulation of singer voice and how much to overabundance of signal in melody scale. On the other hand, humans can hear voices in white noise, or in the sounds of nature. Sound hallucinations are the most common among all. It's an easy test, where you try to hear something, where it is not there, but after couple of minutes, you'll start to imagine things. People are sometimes overfit to hear speech.

We leave this problematic cases and focus only on situations where noise to signal ratio is low and we can utilize statistics to detect speech.

Our signal contains speech and silence parts and we assume an environment where speech is clearly louder than silence/noise. Obviously it may also contain non-speech parts which are similar to actual spoken words, like i.e. inhales or other sounds that talking people can make during speech intervals. At this point I don't care about them and leave dealing with them to other approaches.
This is preprocessing of the speech signal required by every single approach to either speech recognition or alignment.

Speech parts are loud and silence is well, silent, almost at least. If we knew some threshold value, that splits a background noise from the speech, than algorithm of detecting the speech would be to find those parts that are consistently louder then the threshold.

The problem is to find the threshold and how to deal with consistency of the signal above it, since a small peak can always happen in the background, and a speech is sometimes quite quiet.
One should choose wisely how to deal with it, since in theory it is possible to figure out even small pauses in the speech signal, like i.e. between syllables. I found out, that on one hand it is quite difficult to find these pauses and at the same time to not ignore endings of the sentences, which often are slowly fading to silence (because speaker lacks of breath). On another an alignment using length estimations don't improve, when we detect too many pauses, because the algorithm works better when the chunks of signal are bigger, so there's a greater chance they align with punctuation marks at the text.

In the speech recognition systems a detection algorithms have to process the incoming data in an online fashion. Sphinx library implements Bent-Schmidt-Nielsen algorithm, which calculates background noise level and current average signal online, meaning it is updated with each incoming frame.
When signal average level of processed window (see 2.5) is larger than a signal threshold (input constant) and a background average, than the window is classified as a speech.

$$Ave(signal) - Ave(background) > threshold \tag{1}$$

Whetever the signal was marked as part of speech signal, the background average is always updated.

I found this algorithm to be too volatile at the beginning of the recording and it stops too easily at the middle of the longer sentence. I really needed an offline algorithm, which could detect quite reliably longer pauses, which are better aligned with punctuation marks.

Firstly my algorithm worked with a spectrum of a given window. It processed the window of the same length, but a volume was calculated as a sum of magnitudes of all frequencies. On it's own it doesn't give me additional gain, but I also experimented with different transformations of frequency spectrum:

- weighting frequencies depending on distance from normal distribution, what in theory should favour these bands, that are responsible for speech signal,
- applying logarithm or square root, to check how different band contribute to speech signal, by normalizing the power of each frequency
- counting how many times a magnitude of frequency exceeds an average value of a background

Distance from a normal distribution showed me, that lower frequencies have more irregular histogram, which agrees with the consensus, that most important speech data are located at lower frequency bands.
I also found out, that by increasing magnitude differences in favour of lower frequencies gave me better results, then decreasing them, what also agrees with above.