Analysis of knowledge requirements for text alignment problem

Bartosz Kalińczuk

August 30, 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

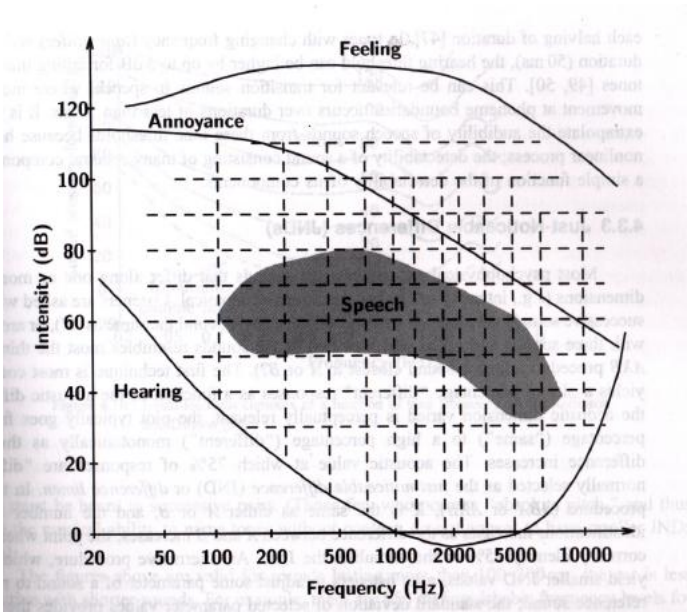
2.1 Human factor

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

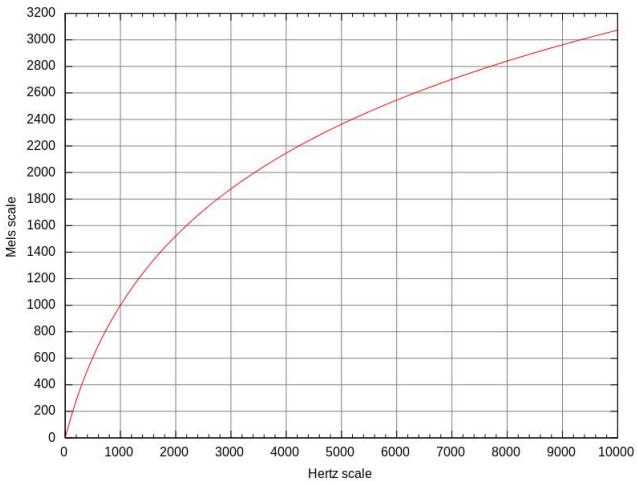
How we perceive sound, that is completely different matter and topic for long philosophical discussion. However we can help ourselves with some subjective experiments. For example Stevens, Volkman and Newman conducted an experiment on a number of listeners to measure, what do we perceive as equally distanced 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]

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

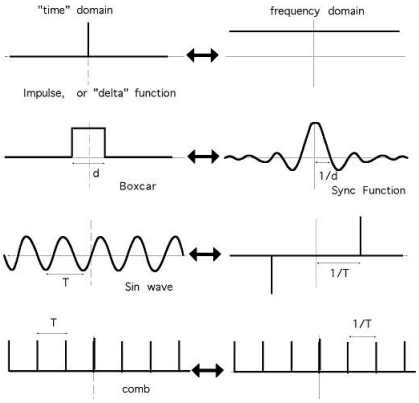
or for the discrete case:

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

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:

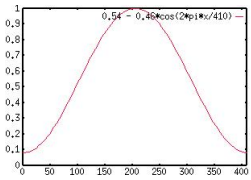


What size of the window should we use? First we have to notice, that in order to capture certain frequency, the window needs to be large enough. We would like to examine signals of frequency ranged from 100Hz (see speech frequencies ranges in chapter 2.1), which is a period of 100th of the second, so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

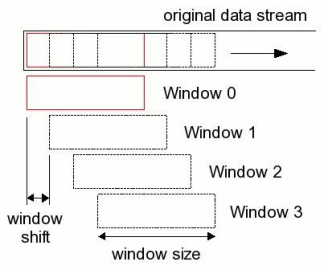
Also windows with abrupt signal discontinuities may cause result with spectral artefacts, so a windowing function is usually applied. Popular choice is a Hamming window function: [8]

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

Which's plot is:

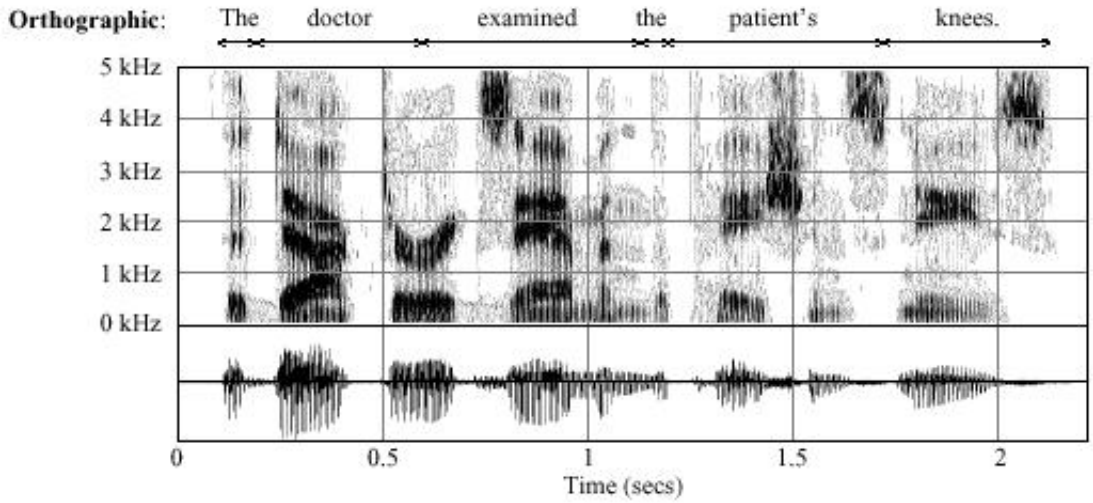


Note that it emphasises values in the middle of the window, so our actual windows should overlap to cover whole time domain. For example by shifting a window by a percentage of it actual width:

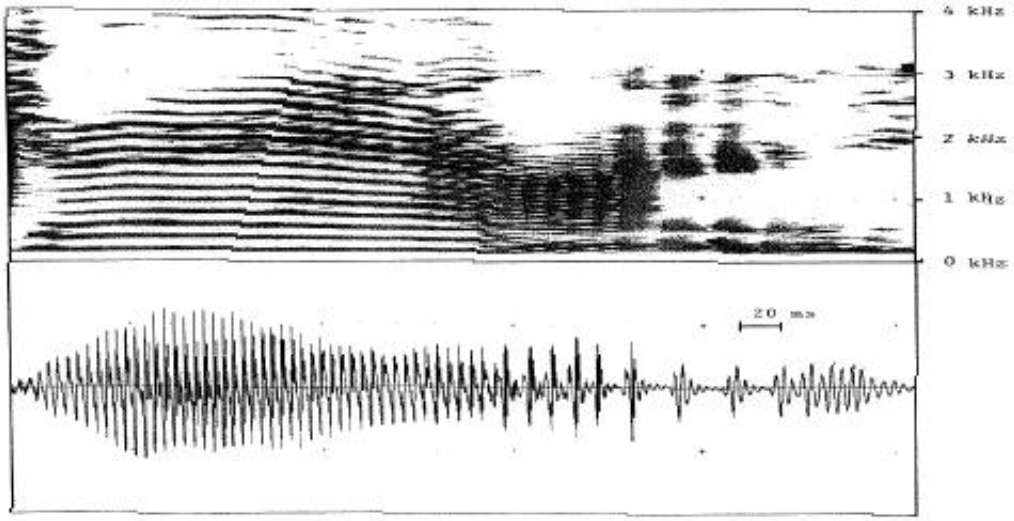


A human speech signal in frequency and time domain: [3]

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



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

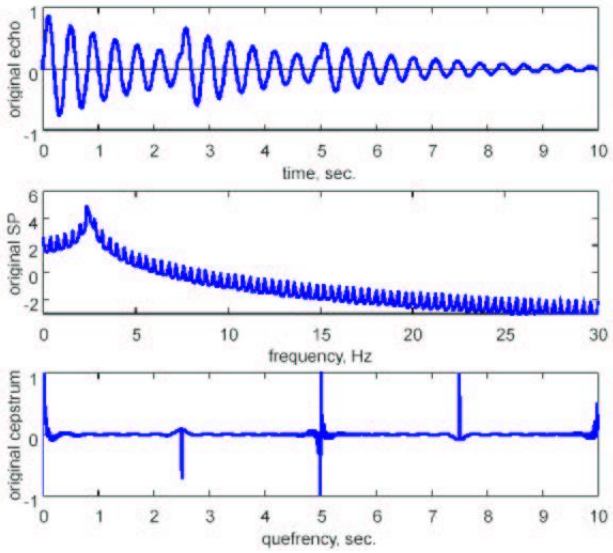


"Drown" (female)

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:

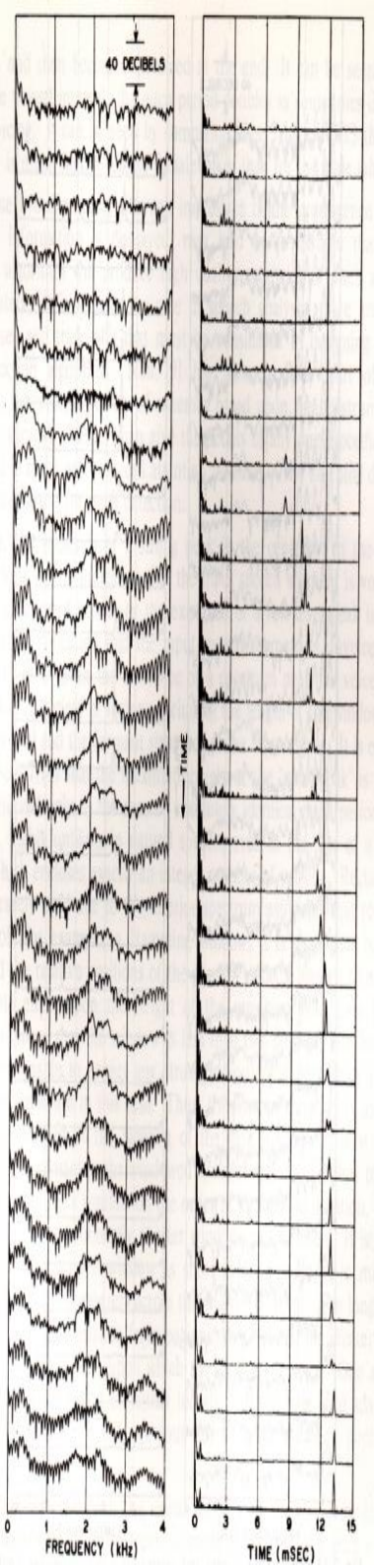
$$C = |F^{-1} log(|Ff(t)|^2)|^2 \tag{9}$$

,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.

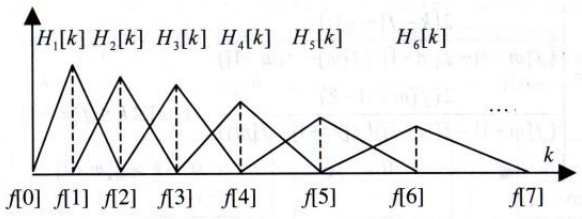
Conversion of frequency spectrum into a mel-spectrum using triangular overlapping windows defined as:

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

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.

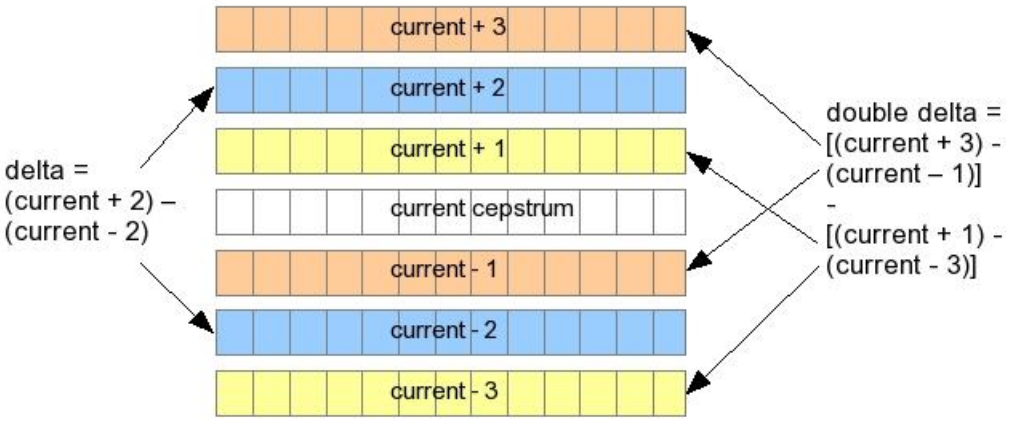
2.5.7 Cepstral Mean Normalization

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.

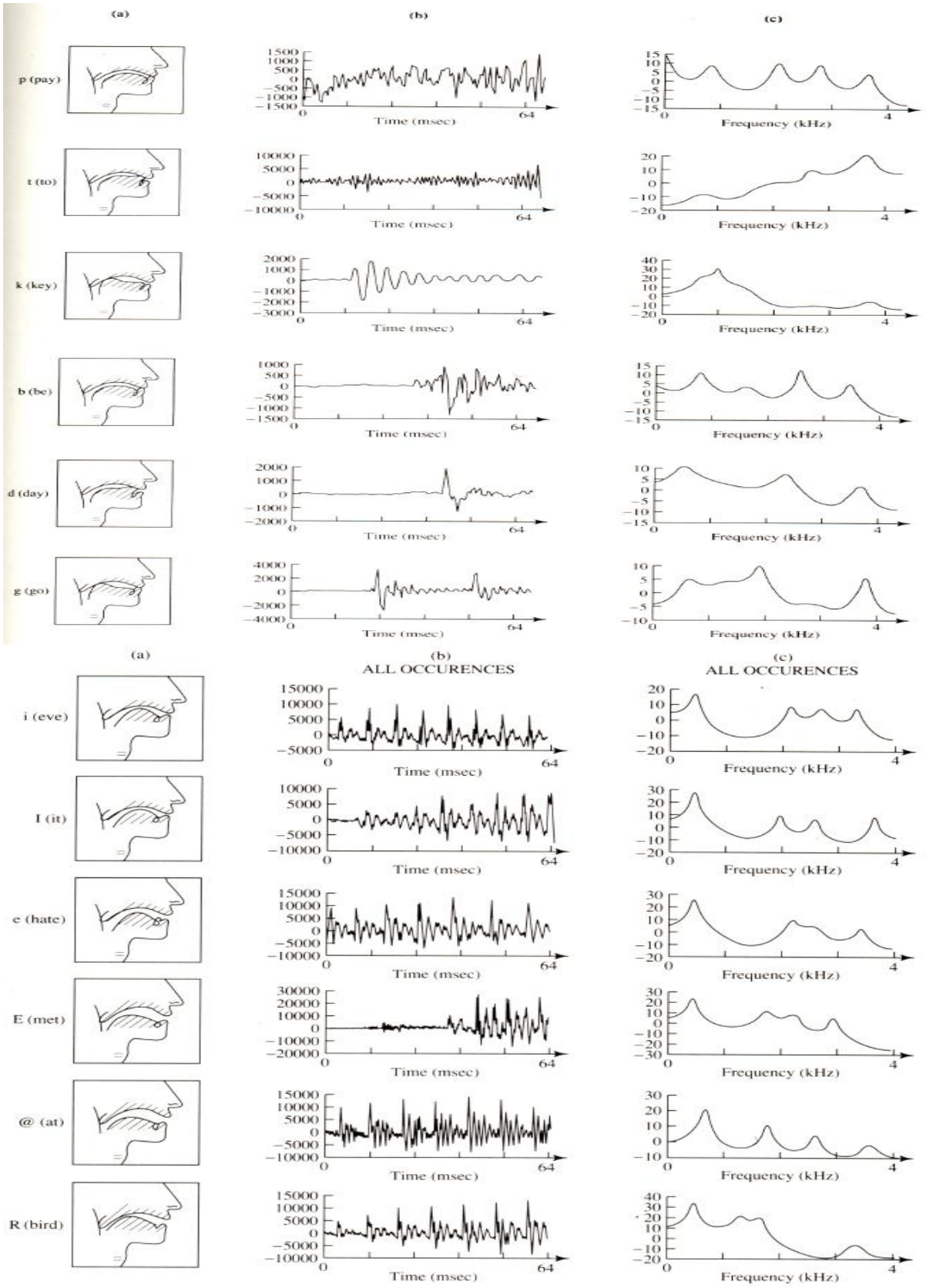
Language grapheme set usually differs quite substantially from it audio counterparts. Often it contains more characters than needed to represent every word from given language and at the same time it is much too small to represent all the nuances of human speech. What is more problematic, the word graphic 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:

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

what is a definition of a mean.

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

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:

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

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.

After mingling with above ideas, my best working solution is to calculate average for the whole frequency spectrum (not only sum of magnitudes), without any transformation to initial values (except for sphinx's preemphasizer transformation). Next step is to calculate background averages from the frames considered background, which is all frames where each frequency power is below average, so the background frames set B is:

$$B = \{F \in S \sum_{\sigma_i inF} sgn(max(0, \sigma_i - Ave_i)) = 0\} \tag{2}$$

where S is a set of all frames in audio stream, and frame F is a set of magnitudes of frequencies from the frame, and Ave is an average of all frames: $Ave_i = \frac{1}{|S|} \sum_{F \in S} \sigma_{F,i}$
Speech frames A are all remainging frames: $A = S \setminus B$.

That is not enough though, because there are often frames inside a speech, which are quite low on volume. These are pauses between phonemes, or some frames of quiet talking (at the end of sentence usually), which didn't not passed the above filter. The granularity is just too big for the purpose of alignment algorithm.

To deal with this granularity I marked as speech all the frames, which didn't pass through filter, but where surrounded by frames, that did, as a speech as well. The reason for that, is to remove all the short pauses, which probably meant nothing and might even be an actual speech. My choice was to mark as a speech all pauses that were shorter than 200ms.

Theoretically filling holes is similar to the algorithm, which calculates vector of volume averages of couple of neighbouring frames, and then use this average in the above formula instead. This introduces a certain inertia for pauses or speeches, but at this point I need just a proper longer pause detection with abrupt endings as quickly as speech starts/ends. Although I must admit, that a certain inaccuracies are not so important for the paused based alignment, if only because the algorithm is quite inefficient and produces only approximated results.

4.2 Text split

Before we can continue with alignment, we still would like to have text split to sentences. It is necessary to have a certain knowledge about punctuation in a given language. Many modern languages use similar punctuation symbols for marking sentence or sub-sentences, but we still need to know a given language alphabet.

This part is very simple. We treat all the alphabets characters as a part of speech, while every other character as punctuation mark, which separates parts of the text, and which may have some correlation with a pauses in a recording. Resulting chunks are those, that contain only characters from alphabet and blanks.

Couple of details are to be dealt with. Not only alphanumeric characters make a word, i.e. a ‘’‘ is also a character, that must be treated as part of speech character at this point, although in later phases it might be ignored.

Another is that, a sequence of white character might also be a separation. A title for example might not be separated by a dot mark, but only be a series of line breaks. Generally more than one line break is considered a pause indicator.

4.3 Estimating time

The problem of matching chunks of text with extracted speeches is a problem of matching duration time of speech recording and an estimated time of the chunks. Before we can continue, we need to estimated the time it takes to say words from each chunk.

Given:
$S = \{[s_i, e_i]\}$ – time intervals, where s_i is a time of beginning of i-th speech and e_i its ending time
$T = [[w_{1,1}, ...], ..., [w_{n,i}, ...]]$ – set of chunks, where each chunk is a word list

Let's rephrase this problem of estimating time it takes to say $[w_{i,1}, ..., w_{i,l_i}]$:

$$E(T_i) = \sum_{j=0}^{l_i} E(time\ to\ say\ w_{i,j}) \tag{3}$$

where l_i is number of words in i-th chunk.

My proposition is to estimated a time of single word with a formula:

$$E(time\ to\ say\ w) = \sum_{c \in w} 0.95\mu_{char} + 0.05\mu_{word} \tag{4}$$

where μ_{char} is an average time it takes to say a single character:

$$\mu_{char} = \frac{number\ of\ nonspace\ characters\ in\ text}{\sum_i e_i - s_i} \tag{5}$$

and μ_{word} is an average time it takes to say a single word:

$$\mu_{word} = \frac{number\ of\ words\ in\ text}{\sum_i e_i - s_i} \tag{6}$$

Although I'm not completely certain in what way my algorithm benefits from the second element, since I haven't run conclusive tests, by my intuition is, that it reduces variance of estimated values.
The proportions above I derived from purely empirical observations, but there were too many variables to be too attached to this exact coefficients.

4.4 Alignment

The assignment problem tries to minimize the difference between speech time and estimated time of matched text chunk. If A is a set of matched pairs of speech and text ($A = \{((s_i, e_i), t_i)\}$) then I want to minimize:

$$argmax_A \sum_{((s_i,e_i),t)\in A} ((e_i - s_i) - E(time\,to\,say\ \ t))^2 \tag{7}$$

This problem is easily solvable with dynamic programming.

The algorithm iterates over speeches.
At k-th iteration a vector R of partial results are kept. The i-th element of the vector contains a result of matching first k speeches and first i sentences.
Initial values of the vector is matching of first speech part with first i text chunks:

$$R(i) = ((e_0 - s_0) - \sum_{k=0}^{i} E(time\,it\,takes\,to\,say\ \ t_i)^2 \tag{8}$$

where s_j and e_j is start and end time of j-th speech,
and t_i is i-th text chunk.

The k-the iteration calculates next values from the formula:

$$R(i) = argmax_{din<0,i-1>}[P(i-l) + [(e_k - s_k) - \sum_{j=i-l}^{d} E(time\,to\,say\,t_j)]^2] \tag{9}$$

Estimated time it takes to say a text chunks $\{e_k, ..., e_{k+c}\}$ would be quite consuming to calculate at each iteration, but it can be precomputed.

Additionally a zero match was added as a way of adding a speech to a text chunk from previous iteration. Note however, that it won't produce a result, where everything is matched to everything, what would have a minimal score equal to 0, since a time it takes to say whole text is a total time of all speeches.
However matching from each iteration contribute to the score separately, so the previous match adds a difference and skipped speech will also add it own difference (assuming it was k-th speech, than added score is equal to $(e_k - s_k)^2$).
It does improve the algorithm though, because of some short speech leftovers, which actually are part of previously matched sentence.

At the end of algorithm the i-th element in R vector is equal to the best cumulative score of matching i chunks and all speeches. Obviously if there are n chunks, then the n-th element contains a score of best possible matching of all speeches and whole text.

To recreate the matching, one could iterate backwards over partial values, or as I did it, to keep additional vector which keeps track of chosen indexes (d in(9)) from all iterations. To produce the best matching, the algorithm traverses back through these indexes and for k-th speech it assigns all chunks between current and previous index. Empty assignment (index haven't changed) is considered to be merged with previous matching (time frame of previous label is updated with current speech).

4.5 Results

The efficiency of above method was tested versus a word alignment (obtained from different methods) in two ways:

- for a given label from the output find words and they testing labels,
  merge the found testing labels times to produce total time of the text chunk,
  give some statistics about time differences

- for a given label from the output find testing labels that are located within the time frame
  produce a text from found labels' words
  count the biggest word difference between the texts by formula below:

  $$len(output\_chunk) + len(testing\_chunk) - 2length(biggest\_subsequence) \quad (10)$$

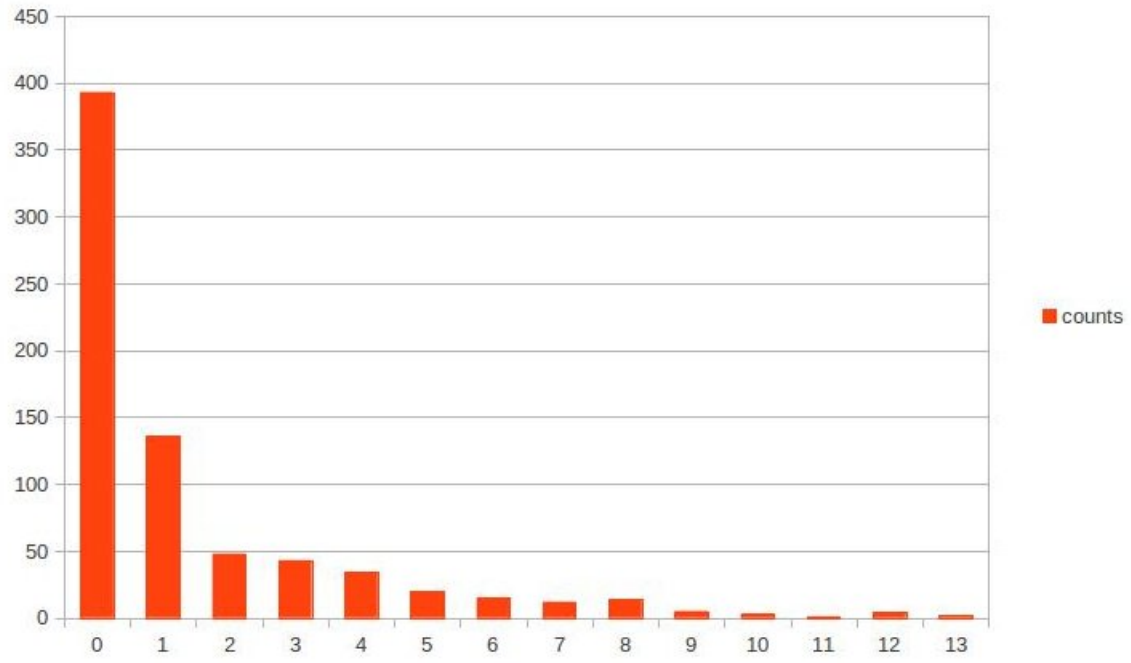
  give some statistics about the word differences

The testing recording is "Doktor Piotr" by Stefan Żeromski, which is 80 minutes long and consists **1886** sentences (or subsentences).

The first statistics are calculated for a variation of algorithm, where the allowed size of holes (or pauses), in speech detection algorithm part, was set to **200ms**. This version produced **730** labels.

The statistics are:

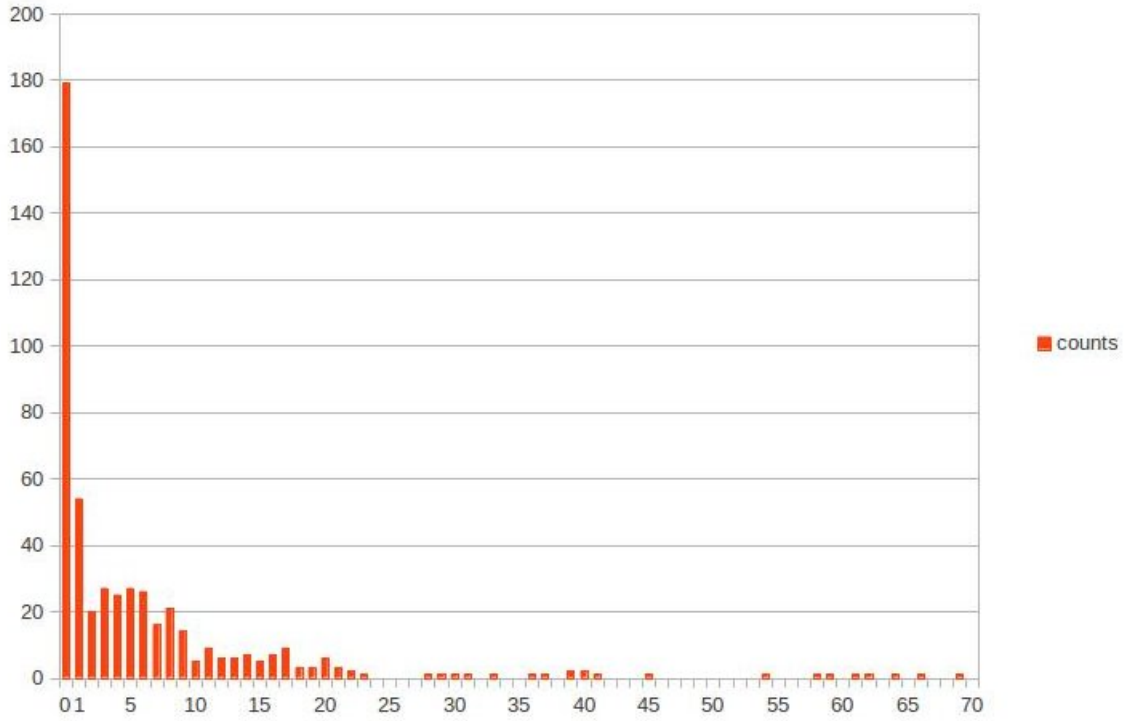
- time differences:
  - there were **370** chunks which time frame were within a **0.5s** difference,
  - average time difference was **0.77s**
  - standard deviation was **0.84s**
  - maximum time difference was **11.21s**



- word differences:
  - **393** chunks had **0** difference in words
  - **136** chunks where different by **1** word (missing or additional)
  - **48** different by **2** words
  - **56** with a difference over **5** words

For **503** speeches, with a pause longer than **300ms**:

- time differences:
  - there were **154** chunks which time frame were within a **0.5s** difference,
  - average time difference was **5.24s**
  - standard deviation was **5.23s**
  - maximum time difference was **37.9s**



- word differences:
  - **179** chunks had **0** difference in words
  - **54** chunks where different by **1** word (missing or additional)
  - **20** different by **2** words
  - **56** with a difference over **10** words

4.6 Conclusions

The time statistics are showing a bit worse results, because word labels are more precise than chunk time frames.

The results are within expectations, after all it is very crude algorithm. It probably could be improved though, however it is hard without introducing much more knowledge about language and possibly some additional training data (i.e. time of phonemes).

This algorithm is a bit sensitive to a chunks created. If only the longer pauses are allowed, by increasing the holes filling factor from 200ms to 300ms in speech detection algorithm, the results were considerably worse.
In this case detected speeches contained more silence frames and consequently the total time is different, and the estimated times might be a bit more wrong. Another reason why it produces worse results might be related to the fact, that it also causes the speeches to be on average longer. This kind of random variables follow a rule, that they differ from an estimated value by square root factor, ergo longer parts, worse estimations.

5 Audio based alignment

5.1 Speech recognition

Audio based alignment is simplified problem of speech recognition. Usually in speech recognition the language consist of millions of words in different conjugations, which may create many different sentences. In audio alignment we expect only one text and only one sentence at the time.

Usually speech recognition is based on trained Hidden Markov Models with Gaussian Phoneme models, although successful systems rarely rely on single state phoneme model. As I mentioned before (3.1) there are many different allophones of single phoneme. A variations are largely attributed to surrounding they are spoken in. For example in polish there are softening phonemes 'i' and 'j', that clearly change how preceding or succeeding phonemes are realized. At some contexts it may happen that certain phoneme will lose it voice are even get silent completely, at another they will be strengthened with additional voice, i.e.:

"staw płytki" → "w" loses it voice and starts to remind a phoneme "f"

"plac budowy" → "c" gains additional voice and reminds a phoneme "dz"

And almost never one can cleanly separate two consecutive phones.

In order to better model different realisations of phonemes, an audio modelling contains models of phonemes at different triphone contexts. For every possible neighbourhood a given phoneme might appear, a separate Gaussian model is trained, for example a WSJ model from sphinx database contains 40 different phonemes and additional silence phoneme, that gives 68921 different triphones, however WSJ model contains over 110k elements, because of additional information about location within a word.

It is not over yet, because transitioning from phoneme to phoneme should be modelled somehow as well. The obvious solution is to train a triple state HMM, which will better fit the changing phone.
That triples number of states.

This huge number of different models requires a lot of training data, which is on one hand not easy to come along with, on another requires a lot of time to train with.
Training data for speech recognition is one constant problem, which have to be dealt with. The problem is amplified, by the fact, that training samples should be small and very well described. That hardly can be done manually.
Another bad news is, that a model trained on news feeds, is expected to perform poorly as a spoken address recognition software.

Once we have our audio model, than speech recognition is an easy piece, isn't it?
Not really, since so many possible and often similar words make it a hard problem. Rarely a speech recognition software is based solely on audio model to recognize a sequence of phonemes.
Problem is, that the number of states is too huge in order to be processed exhaustively, even though we can use a Viterbi algorithm to find a best (most probable) sequence of states for given observation sequence and we keep results only for a present moment in time (a single frame), that still gives a huge number of kept states with calculated score.

The sphinx library deals with this problem by searching in a breadth first search manner (meaning that it will keep states only from a current moment) and to cut on number of states to process, a priority queue is used, which keeps only best scoring ones.

And this is the part of recognition software we would like to use for the alignment problem.
The difference from here is that, we do know what the text is spoken, while for recognition there is no such knowledge.
However let's dwell on this a little bit longer, so we can see what a huge simplification it makes.

Let's start with a simple example by using WSJ dictionary:
phoneme sequence "W AH N", which on it's own can be recognized as word "one" or "won" is also a prefix for 36 another different words and subsequence of total 117 different words.
There is a sequence "W AH N S EH L F", which assigned to word "oneself", however it may also be from the end of word "penguin" ("W AH N") and beginning of word "cellphone" ("S EH L F").

Sphinx library provides two ways of language modelling to tackle sucha ambiguity:

- ngrams frequencies
- language grammar

For simple languages, like a sequence of digits or language containing one long sequence (like in case of text alignment), it is advisable to use language grammar.
For general recognition software like Apple Siri, ngrams are necessary, although I am not convinced that they haven't used a sort of grammar for typical queries.

Ngrams (in wsj n is maximum 3) are a way to assign a probability to word sequences. In above example it is expected, that "oneself" is more probable to occur in real word, than the later case, although obviously we can't judge so easily, since the probability of choosing these particular word depends on a probability of whole sentence.

Anyway I think that's enough about speech recognition.
For our purposes we need "only" audio model and word to phoneme sequence dictionary, in order to efficiently align text to speech.

5.2 Differences between english, russian and polish phonetics

The sets of phonemes for any language differ slightly between publications and they differ even more between different audio models. For that reason I'm not going to introduce here established phonology for any language, however I'll try to match a phoneme set used by WSJ model for English, VoxForge model for Russian, Corpora and mine for Polish.

In the table below I collected phonemes from four used sets and analysed differences, that may cause problem to represent some words using given phonetics and in result may cause problems with alignment. The table is organized so the similar phonemes are next to each other. If there is not alternative in a language, than the row remains empty.

English	example	Russian	example	Corpora Polish	Polish used by me	example	Notes on similarities
AA	adopt	a	арена	a	a	dwa	The most different from this set is English „AE", which also shows a similarity to Polish and Russian „e"
AE	act	aa	акт				
AH	acute						
AW	allow			a_ (ą)		tą	There is no such a vowel in Russian however it can be substituted with „oo l", similar for my Polish model with „o l"
OW	aero						
AY	bike						Simulated with Russian and Polish „a j"
B	bill	b	был	b	b	być	Quite similar
		bb	дебет				A softened version of „b", a bit different, but Polish and English „b" can also be slightly softened
		c	цвет	c	c	coś	In English it is something like „T S"
CH	jackovich	ch	девочка	ci (ć)	ć	czcić	English „CH" is actually used for slavian words containing „c", „ć", „cz" like phonemes. Russian „ch" is similar to softened Polish „cz", so it has to replace both „ć" and „cz"
				cz	cz	czego	
JH	just			drz		dżem	Simulated by „d zh" or „d ż", English „JH" is quite similar.
				dzi (dź)		kadź	Simulated by „d zh" or „d ź", English „JH" must simulate this one as well, although it also can be softened, it is not very similar.
D	dad	d	длина	d	d	duży	Except for Russian and Polish „d", all are a bit different. „dd" is again a softened version.
		dd	дитя				
DH	they						No counterparts, something between „d" and „z".

English	example	Russian	example	Corpora Polish	Polish used by me	example	Notes on similarities
				e_ (ę)		sęk	Like „ee l” or „e ł” or „EH W”
EH	thread	e	диване	e	e	ela	Similary as „a” alternatives, all sound quite alike, but a bit different. Russian differs in a stress, English „ER” is an “e” merged with a silent „r”
ER	thriller	ee	дне				
EY	thursday						Like „e j”.
F	film	f	фазы	f	f	film	Very alike, except of course a softened Russian version
		ff	филат				
G	eager	g	долго	g	g	gęś	As above
		gg	долге				
HH	who	h	домах	h	h	chata	As above
		hh	духи				
IH	picture	i	духами	i	i	igła	Russian and English variations differ in stress, but all are quite similar
IY	acree	ii	духи				
		ae	ранят				This is a quite like „i” phoneme, but not completely. In Russian many vowels can be shortened to unrecognized version, which sounds like „i”.
K	quote	k	кафедр	k	k	kto	Very alike, except of course a softened Russian version.
		kk	кефир				
Y	lawyer	j	раной	j	j	jak	Very similar.
L	lawyer	ll	алекс	l	l	lato	Quite similar, although Russian is a softened version of „l”, so it doesn’t cover all allophones of Polish „l”
W	work	l	лад	l_ (ł)	ł	łąka	Quite similar
M	mom	m	малы	m	m	mama	Very alike, except of course a softened Russian version
		mm	малыми				
N	nail	n	надеюсь	n	n	nos	Very alike.
		nn	надень	ni (ń)	ń	koń	No English alternatives, although it seems reasonable to use „N Y” sequence.
NG	thing						In Russian and Polish might be simulated with „n g” or just „n”, but no natural alternatives
AO	for	oo	поднёс	o	o	tok	Quite similar
		ay	погадай				Non stressed version of „oo”, however it sounds more like „a”
OY	foil						Can be simulated with „o i”.
P	pack	p	поезд	p	p	pas	Similar. Russian allophones cover Polish and English phonemes.
		pp	помпе				
R	race	r	рад	r	r	rura	As above
		rr	рюкзак				
S	sand	s	сцен	s	s	sen	As above
		ss	сегодня				

English	example	Russian	example	Corpora Polish	Polish used by me	example	Notes on similarities
SH	shop	sh	шёлка	si	ś	śliwka	Quite alike.
				sz	sz	szosa	It is a bit similar to "ś", but no real alternatives.
		sch	щека				No real counterpart. Can be simulated with „SH CH" and „ś ć' or „sz cz" and variations
T	tin	t	та	t	t	ten	Similar. Russian allophones cover Polish and English phonemes.
		tt	тёмный				
TH	thanks						No counterparts, something between „f" and „t".
UH	foot	u	ёлку	u	u	ósemka	All are quite alike, although subtle differences remain.
UW	fool	ur	юг				
		uu	абсолют				
V	visit	v	ваза	w	w	wiedza	Similar. Russian allophones cover Polish and English phonemes.
		vv	живьем				
		y	живы	y	y	dym	Russian allophones differ in stress, but there is a problem with English equivalent. The most alike phoneme is „IH".
		yy	живых				
Z	visor	z	ваза	z	z	zebra	Quite alike.
		zz	вазе	zi (ź)	ź	źle	Russian and Polish are very alike. No English equivalent, „ZH" is the closest.
ZH	visual	zh	жаба	rz (ż)	ż	rzeka	Quite alike.

I can try to draw some conclusions from the table alone.

In English there are no natural alternatives in 8 cases, while in Russian only in 2 cases, although there are some other dissimilarities, it is clearly visible now (if it wasn't before), that Russian is much more similar language to Polish, than English.
It might be a surprise though, that English isn't so different, and it seems possible to emulate every Polish word with English phonemes. Maybe not too surprising though, after all we are all humans.

5.3 Grapheme to phoneme conversion grammar

For the purpose of speech recognition we require very accurate dictionaries, that keep track of actual pronunciation of words with all possible variations.

Usually the dictionaries are created by analysing actual recordings.
I would like to propose a way of converting grapheme sequences into phonetic description, which might be good enough for many real life applications, like word alignment or even phoneme alignment.
Nevertheless it would still be applicable only to problems which are easier than speech recognition, but only because it is a hard problem, which tries to get as much accuracy from sub-problems as it is only possible.

Take a look at English word "one" and "tone". The differ with character, but a phonectics are: "W AH N" and "T OW N".
It doesn't seem to be trivial, and actually it might be necessary to study in more detail a language to find quite accurate grammar. However for Polish it might be easy enough to create such a grammar, that will give us enough accuracy to perform alignment task.

We need at least three of such a grammars, converting to English, Russian and Polish phonemes.
Here are example conversions:

Character sequence	English phoneme sequence	Russian phoneme sequence	Polish phoneme sequence
a	AA	aa	a
ą	AW	oo l	o ł
b	B	b	b
c	T S	c	c
ci	CH IH	ch ii	ć i
cia	CH Y AA	ch aa	ć j a
trz	T SH	t sh	t sz
dż	JH	d zh	d ż
dź	JH	d zh	d ź
ch	HH	h	h
h	HH	h	h
ij	IY	—	—
ó	UH	u	u
rz	ZH	zh	ż
ł	W	l	ł
dzia	JH Y AA	d zz aa	d ź j a
ź	ZH	zh	ź
ś	SH	sh	ś

The conversion table looks similar to description of lexer tokens and in general it is what we do here. We convert word into tokens standing for a phoneme sequence. It might be a good idea to use regular expression in the grammar, however I haven't found a need for these, hence my rules convert a character sequence only.

My rules were able to produce a set of possible phoneme representations, simply by adding a set of phoneme sequences to the right side of the rule. These however didn't prove to be quite beneficial, and after a short time experimenting with it, I reduced number of sets dramatically. After awhile I actually used a different variants only for the word alignment and only when I used sphinx with foreign audio model.

The algorithm looks as follows:

- variables:
  - **S** - a set of all rules in the grammar
  - **P** - a set of potential candidates
  - **c** - longest matched candidate so far
  - **R** - output set of phoneme sequences
- at the start we have:
  - $P = S$
  - c is unset
- iterate over a character sequence,
  - remove all candidates from P, that will never be a match after adding current character,
  - if exists such $p \in P$, that it is currently matched, set c with p
  - if P is empty
    - for each s phoneme sequence in c:
      - create R_s by adding s to each element of R
    - set new output $R = \cup_s R_s$
    - new set of potential candidates is $P = S$
    - move iteration pointer to the end of matched sequence c

I would leave here an open question if it is possible to create such a grammar for more accurate dictionaries.It might be, since phonemes are the effect of how comfortable it is to say a given sequence of phones and people are not really able to remember too complex conversion. It might be possible, that the grammar would need take into account some rare oddities, like word "one", but there is definitely a finite number of such. In order to find such a grammar, probably a study on existing and trained dictionaries should be done, but this is out of the scope of this project.

5.4 Audio model alignment and results.

This part is relatively easy, because sphinx library already support word alignment using any audio model. It requires only a phonetic dictionary, which we can create using conversion grammar from previous chapter. Whole conversion works similar to recognizing word, the same breadth first search algorithm with queue of best results is used, except that a HMM for whole language is created from simple grammar, which allows only one big sequence of words.

We would like to test word alignment using English (WSJ) model and Russian (VoxForge) and compare it to some manually aligned sample.

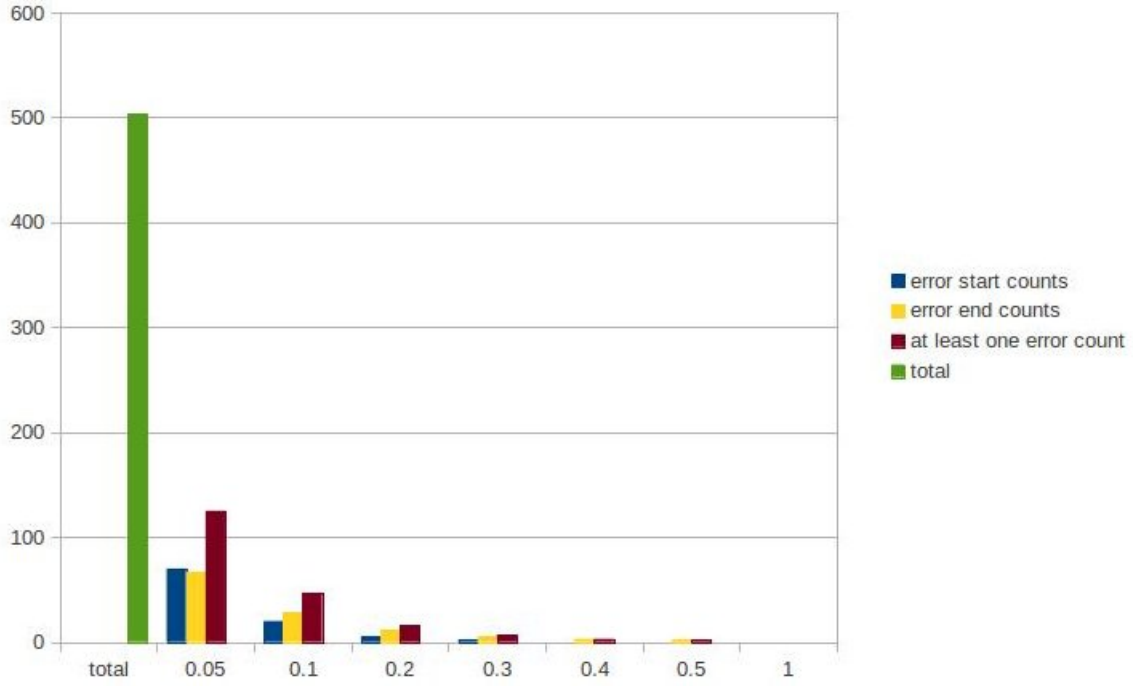
For my testing recording I used a 5 minute and 20 seconds of audiobook "Doktor Piotr" by Stefan Żeromski and 16 minutes and 20 seconds of audiobook "Boże Narodzenie" by Maria Dąbrowska.
They were aligned by two different, but generally normal people (no experts) with purpose to make alignment as accurately as possible without selecting too much or too little. However while the person aligning "Doktor Piotr" considered overlapping to be forbidden, the person aligning "Boże Narodzenie" didn't have this presumption. This small fact changes statistics a bit especially for smaller time differences.

Another difference between those texts, is that I have never seen any mistake in text for "Doktor Piotr", at least in those first 5 minutes, but I've seen wrong words in the "Boże Narodzenie", where the reader has replaced them with something different, or the text was incorrect, nevertheless there are some discrepancies.

With English model I haven't got too much luck. The problem with the algorithm for word alignment is that once it goes wrong, then it never goes back on track again. With English model after **38** seconds of "Doktor Piotr" it incorrectly assigned 3 seconds to word "części" after **62** words and it never recovered. For "Boże Narodzenie" it has gone wrong after **257** seconds on word "czarownicach" after **503** words.

The statistics for "Boże Narodzenie" however shows, that before it goes bad it actually aligns first **503** words quite nicely:

- Maximum difference (start or end): **0.559s**
- Maximum difference (start or end), if label was to short at one end: **0.559s**
- Average difference (start or end): **0.032s**
- Error counts depending on time thresholds:



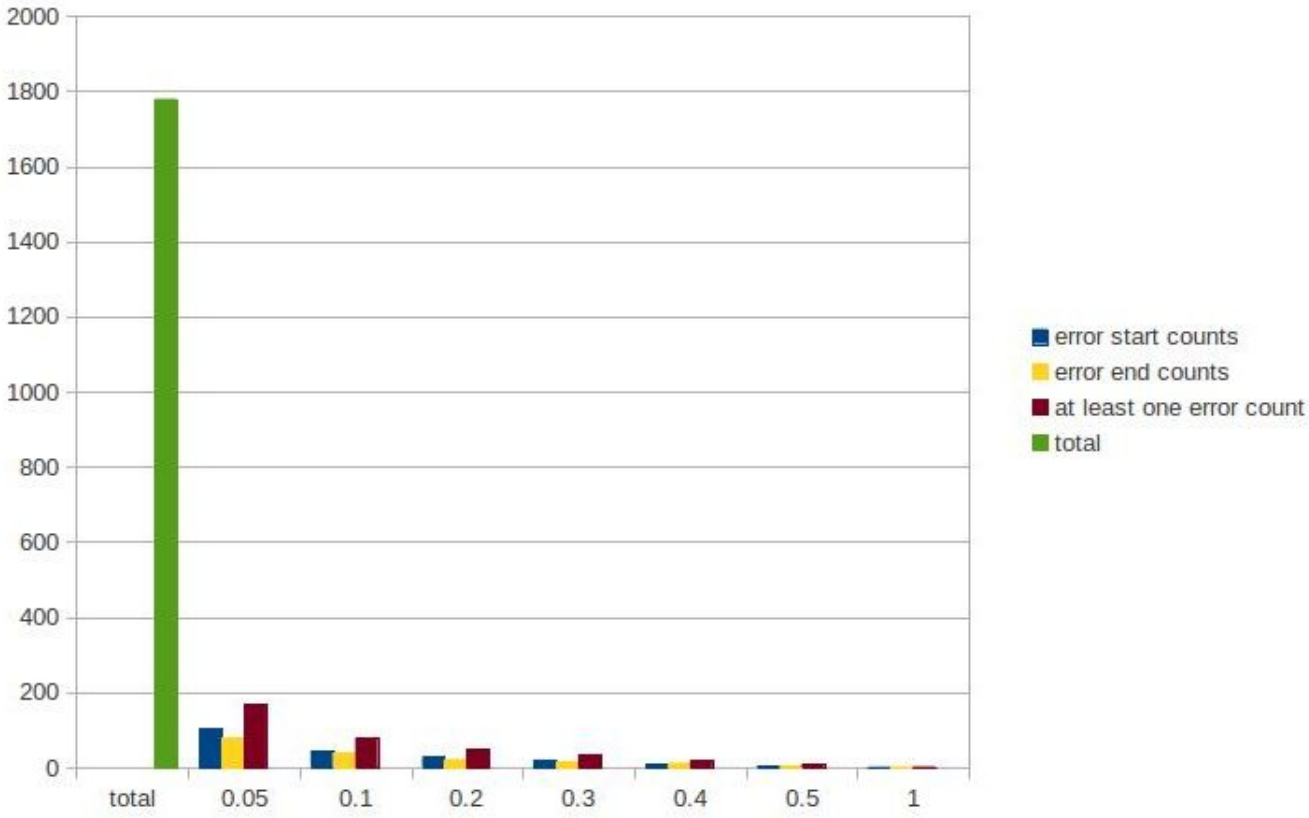
It seems that there weren't any substantial errors in the matching, although apparently the English audio model has some trouble with phoneme "cz", which I'll remind was replaced by English "T SH", which sounds more softened, than actual pronunciation. It is hard to figure out some other way of representing this one, and even though it looks like it could be used, some more elaborate method should be invented. Usually when word goes bad, the algorithm assigns too long selection to it, so maybe checking the time if it goes wrong and the to try some kind of recovery? I think it is possible, but I haven't explored this in more details.

I also tried Russian model, which haven't got problems as above and I could have generate statistics for whole alignment:

The "Boże Narodzenie" statistics are:

- Total number of words: **1779**
- Maximum difference (start or end): **2.451s**
- Maximum difference (start or end), if label was to short at one end: **0.543s**
- Average difference (start or end): **0.016s**

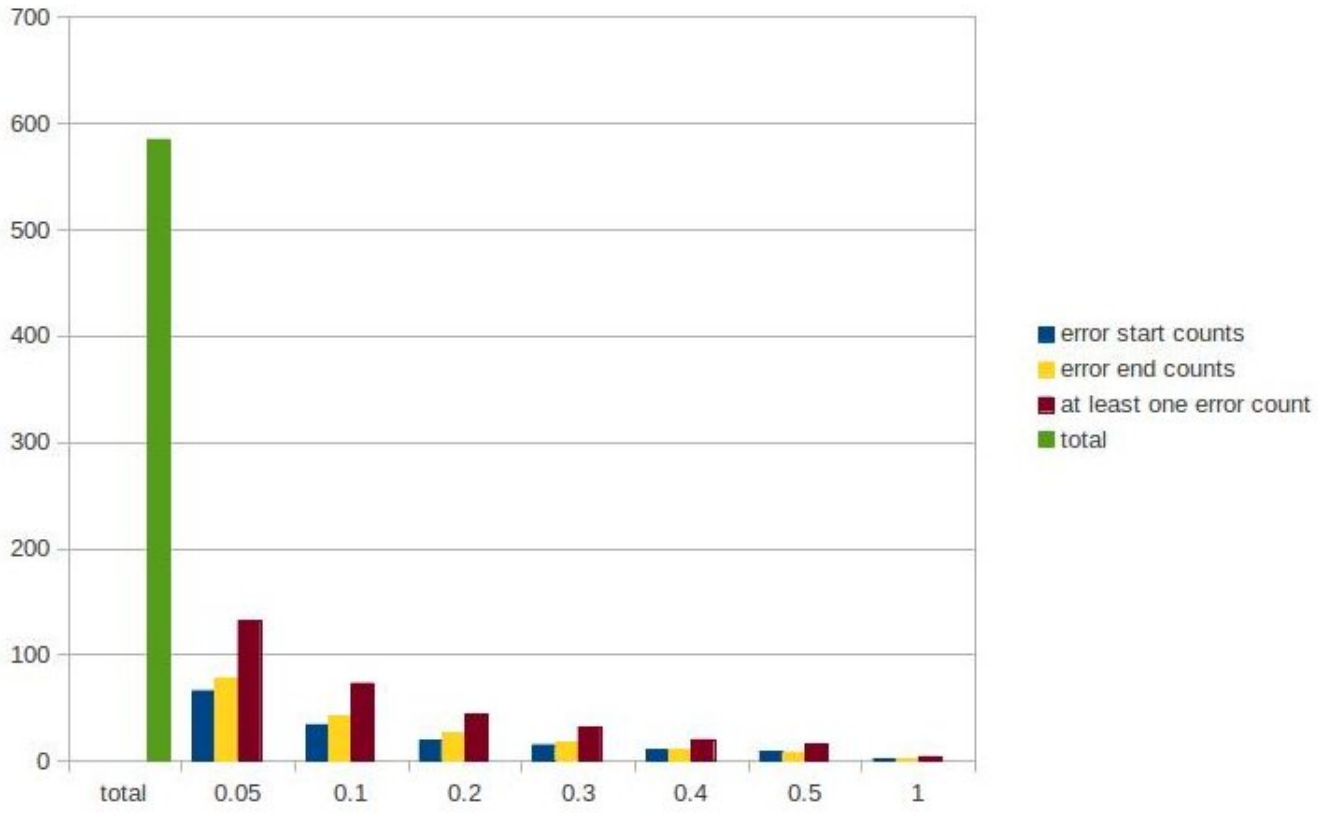
The error counts regarding different time thresholds:



The statistics for “Doktor Piotr” sample are:

- Total number of words **585**
- Maximum difference (start or end): **1.354s**
- Maximum difference (start or end), if label was to short at one end: **0.534s**
- Average difference (start or end): **0.033s**

The error counts regarding different time thresholds:



Analysing the above statistics it can be noticed, that even if English model didn't go wrong, it still seems to perform worse, than Russian model. The average difference is twice bigger than for the former. Also the counts of errors in relation to total number of aligned words shows, that Russian models outperforms the English one, what is quite understandable, since we knew apriori, that Russian is much more similar to Polish.

It is also noticeable, that they perform a bit differently. While English model finds borders of the word quite well, except for spectacular failures of course, the Russian on other hand frequently happens to include a pause after the word. This is a bit problematic for many different applications, except maybe for training, which can deal with that. Anyway since for English, the algorithm failed to finish the alignment, it may have happened, that it would show similar behaviour in later parts of the book.

The additional pause added to a word label isn't completely bad, since it means, that we are quite sure, that given selection contains an assigned word, and that there is very little total misses. Direct observation of collected data supported that conclusion. There were very few total misses and only related to alignment conjunctives ("w" especially). From autopsy of collected data we also see, that the biggest mismatches were committed, when accounting a silence as a part of word.

One could try to further improve above method, at least by meddling with a phoneme conversions or maybe even training dictionary from a corpus, to get a better phoneme representation. I can't be sure at this point how big impact has the simple conversion grammar on results, although it doesn't seem to be the most promising room for improvement.

My final conclusion is that, regardless of some mismatches, it is a quite nice method for word alignment, especially for the purpose of further improvements in speech recognition systems and audio model trainings.

synthesized texts	Notes on erroneous parts in synthesized speech using words and phonemes initiated by pause based alignment
“Rosja przedwojenna była wymarzoną areną dorobku dla ludzi tego typu zwłaszcza pochodzących z Królestwa”	- „areną” – „a” is missing - jumps between phonemes are clearly visible, although they are not disturbing enough to cloud the meaning
“Wiadomości zaczerpnięte w klasach gimnazjalnych wrodzona inteligencja która wraz ze zdrowiem towarzyszyła poszukiwaczowi posady i na zawołanie zjawiała się nie siana i nie pielęgnowana wytrzymałość odwaga wesołość i pewna odrobina drwiny z Moskala u którego się służy lecz nad którym jednak panuje się mimo wszystko torowały drogę od niższej do wyższej pozycji”	- the phonemes of word „mimo” are quite short, so listener need to be really focused, to not miss a meaning of the word, - after „zjawiała się” there are two extra words „do wyjścia”, closer inspections showed, that „się” was included as whole, but was badly aligned,
“Trzeba przyznać że nie ostatnią rolę grała w tej operze protekcja cicha pokorna dobra wróżka prowadząca za rękę od niskiego do coraz wyższego rodaka tu i tam zaczepionego nogą lub łokciem na tej rosyjskiej drabinie”	- „przyznać” - „ć” is actually „ź” - „tej” is something else - „protekcja” - hearable „w” before „t” - „spokojna” - something extra after „k” - „zaczepionego” - „z” is missing - „drabinie” - „ie” is missing
“Chrząszcz brzmi w trzcinie w Szczebrzeszynie W szczękach chrząszcza trzeszczy miąższ Czcza szczypawka czka w Szczecinie”	- „brzmi” - „żm” phoneme sequence was not found in whole text so it was omitted in the synthesized audio, - „czcza” sounds like „cza”, - „Szczecinie” - „ie” missing
“Chrząszcza szczudłem przechrzcił wąż Strząsa skrzydła z dżdżu A trzmiel w puszczy, tuż przy Pszczynie Straszny wszczyna szum”	- „wąż” - „ą” is missing - „straszny” - „ny” is missing - „szum” - „m” is strange
“Litwo Ojczyzno moja ty jesteś jak zdrowie Ile cię trzeba cenić ten tylko się dowie Kto cię stracił Dziś piękność twą w całej ozdobie Widzę i opisuję bo tęsknię po tobie”	- „cenić” - „ć” is missing - „stracił” - extra „e” or even kind of „em” - „piękność” - very short „e” - „ozdobie” - first „o” is missing, second sounds like „ą” - „tobie” - short „e”

synthesized texts	Notes on erroneous parts in synthesized speech using words and phonemes initiated by pause based alignment
“Panno święta co Jasnej bronisz Częstochowy I w Ostrej świecisz Bramie Ty co gród zamkowy Nowogródzki ochraniasz z jego wiernym ludem”	- „świecisz” - „ś” sounds weird like “sz” through teeth, - „co” - hearable „t” at end, - „gród” - missing „g”, - „nowogródzki” - extra „na” at the beginnings and „dz” sounds like „jz”, - „ochraniasz” - first „o” is actually „w”, - „wiernym” - quite bad, at the beginning there is extra „pły”, and there is missing „r” making it completely unrecognizeable
“Jak mnie dziecko do zdrowia powróciłaś cudem Gdy od płaczącej matki, pod Twoją opiekę Ofiarowany martwą podniosłem powiekę I zaraz mogłem pieszo do Twych świątyń progu Iść za wrócone życie podziękować Bogu Tak nas powrócisz cudem na Ojczyzny łono”	- „mnie” - extra „u” at the beginning, - „ofiarowany” - first „o” sounds like „he”
“Tymczasem przenoś moją duszę utęsknioną Do tych pagórków leśnych, do tych łąk zielonych Szeroko nad błękitnym Niemnem rozciągnionych”	- „łąk” - extra short „a” at the beginning
“Do tych pól malowanych zbożem rozmaitem Wyzłacanych pszenicą, posrebrzanych żytem Gdzie bursztynowy świerzop, gryka jak śnieg biała Gdzie panieńskim rumieńcem dzięcielina pała”	- „do” - extra „łe” at the end, - „rozmaitem” - sounds like „smeitem”, - „gryka” - noticeable short „ł” at the end, - „panieńskim” - „ie” is short, - „rumieńcem” - a pause between „m” and „ie”
“A wszystko przepasane jakby wstęgą miedzą Zieloną na niej z rzadka ciche grusze siedzą”	- „wstęgą” - only „gą” is recognizeable
“na stole z powyłamywanymi nogami leżą śliwki czereśnie pomarańcze i ogórki”	- „leżą” - extra „na” up front, - „czereśnie” - missing „re”
“W czasie suszy szosa sucha”	
“Za górami za lasami znajduję się wysoka wieża strzeżona przez smoka”	

synthesized texts	Notes on erroneous parts in synthesized speech using words and phonemes initiated by pause based alignment
“Maksymalistyczny egzystencjalny program Mrożka polegał właśnie na stawianiu świata pod znakiem zapytania w świetle jak najbardziej trzeźwych zarzutów o jego niewystarczalność”	- „maksymalistyczny” - first „m” is actually an „s”, extra „lny” at the end, - „mrożka” - „first „m” is actually „z”, - „polegał” - extra short trailing „o”, - „o” - very short and faint
“Mrożek jako krytyk cywilizacji widział w niej nadto przemoc mechanizm mielenia jednostek na proszek W rewolucjach brzydził go fetor mierzwy w jaką zmieniają się górnolotne czyste ideały”	- „mechanizm”- sounds like „mechaniźmie”, because „z” is „ź”, - „jednostek” - short but loud extra „w” at the beginning, - „brzydził” - noticeable „g” between „y” and „dz”, - „fetor” - trailing „e”, - „ mierzwy” - missing „y”, - „jaką” - a bit distorted and unrecognizeable, - „ideały” - „i” is replaced by „u” and extra „j” between „e” and „a”
“Pomimo języka groteski którym tak chętnie się posługiwał był przede wszystkim piewcą głębi sztuki jej ocalającego kontemplacyjnego wymiaru”	- „pomimo” - some artefact in ending „o”, - „jej” - something extra at the beginning, - „ocalającego” - some „r” after „oca”
“Przy czym nigdy głośno o tym nie mówił Jednocześnie sceptycznie i ostrożnie podchodził do kwestii wyobraźni widząc w niej zasadę kierującą ludzkimi poczynaniami a co z kolei skutkuje każdorazowo totalitarną eksterminacją”	- „kwestii” - last „i” like „j”, - „wyobraźni” - missing „ob” changing the meaning of the word, - „totalitarną” - „ą” sounds like „ąu”

6 Bibliography

References

[1] Houghton Mifflin Company. *The American Heritage Dictionary of the English Language*, Fourth Edition, 2000.

[2] Prof. W. Alberti. *The anatomy and physiology of the ear and hearing.*

[3] Prof. Joseph Picone *Fundamentals of speech recognition*

[4] Olson Harry F. (1967) *Music, Physics and Engineering*

[5] Stevens Stanly Smith, Volkman John, Newman Edwin B. (1937) *A scale for the measurement of the psychological magnitude pitch Journal of the Acoustical Society of America*

[6] Douglas O'Shaughnessy (1987) *Speech communication: human and machine.*

[7] Zwicker E. (1961) *Subdivision of the audible frequency range into critical bands.*

[8] H.P. Combrinck and E.C. Botha *On The Mel-scaled Cepstrum*

[9] Welch P. (1967) *The use of fast Fourier Transform for the estimation of power spectra: A method based on time averaging over short, modified periodograms*

[10] Steven W. Smith *The scientist and engineer's guide to digital signal processing.*

[11] A. Pinsky *Introduction to Fourier analysis and wavelets*

[12] B.P. Bogert, M. J. R. Healy, J. W. Tukey *The Quefrency Alanysis of Time Series for Echoes: Cepstrum, Psuedo-Autocovariance, Cross-cepstrum and Saphe Cracking*

[13] Seyed Hamidreza Mohammadi, Hossein Sameti, Amirhossein Tavanaei, Ali Soltani-Farani *Filter-bank design based on dependencies between frequency components and phonem characteristic*

[14] Dr. James Glass, prof. Victor Zue *Automatic Speech Recognition* MIT course.

[15] Daniel Jurafsky, James H. Martin *Speech and language processing*

[16] B. Plannerer *An Introduction to Speech Recognition*

[17] Davis, Marmelstein (1980) *Comparison of parametric representation of monosyllable word recognition in continously spoken sentences*

[18] Ahmed N., Natarjan T., Rao K.R. (1974) *Discrete Cosine Transform*

[19] Syed Ali Khayam *The Discrete Cosine Transform (DCT): Theory and Application*

[20] Crystal David *Linguistic*

[21] Chomsky N., Halle M. *The sound pattern of English*

[22] Jagodziński G. *Gramatyka języka polskiego.*

[23] J.L. Rodgers, W.A.Nicewander *Thirteen ways to look at the correlation coefficient.*

[24] Jae Myung *Tutorial on maximum likelihood estimation*

[25] Prof. A. Moore *Clustering with Gaussian Mixtures*

[26] Jeff A. Bilmes *A Gentle Tutorial of the EM Algorithm and its Application to Parameter Estimation for Gaussian Mixture and Hidden Markov Models*

[27] M. Karaś, M. Madejowa *Słownik wymowy polskiej.*

[28] John-Paul Hosom *Speaker-Independent Phoneme Alignment Using Transition- Dependent States*