Politechnika Wrocławska

Wydział Matematyki
Kierunek studiów: Matematyka stosowana
Specjalność: –

Praca dyplomowa – inżynierska

ALGORYTMY ROZPOZNAWANIA MOWY OBECNOŚCI ZAKŁÓCEŃ.

Rafał Sokołowski

słowa kluczowe:
Ukryte łańcuchy Markowa, MFCC, rozpoznawanie mowy, szumy gaussowskie, analiza sygnałów.

krótkie streszczenie:
Rozpoznawanie mowy jest problemem identyfikacji słów z sygnałów dźwiękowych. Wykorzystując odpowiednie narzędzia do przetwarzania sygnałów i ukryte łańcuchy Markowa zaprezentujemy przepływ danych dla problemu rozpoznawania izolowanych słów. Skupimy się zwłaszcza na analizie danych w obecności zakłóceń i zaproponujemy metodę do polepszenia wyników modelu na zaszumionych obserwacjach.

Opiekun pracy dyplomowej	dr Marek Skarupski	.....	.....
	Tytuł/stopień naukowy/imię i nazwisko	ocena	podpis

*Do celów archiwalnych pracę dyplomową zakwalifikowano do:**

a) kategorii A (akta wieczyste)

b) kategorii BE 50 (po 50 latach podlegające ekspertyzie)

** niepotrzebne skreślić*

pieczątka wydziałowa

Wrocław, rok 2022

Wrocław University
of Science and Technology

Faculty of Pure and Applied Mathematics
Field of study: Applied Mathematics
Specialty:

Engineering Thesis

SPEECH RECOGNITION ALGORITHMS IN THE PRESENCE OF INTERFERENCE.

Rafał Sokołowski

keywords:
Hidden Markov chains, MFCC, speech recognition, gaussian noises, signal analysis.

short summary:
Speech recognition is the problem of identification words from audio signals. Using the proper tools for signal processing with hidden Markov chains we will introduce data workflow for the problem of isolated word recognition. We shall focus on data analysis in the presence of noise and present a method to boost performance of model on noised observations.

Supervisor	dr Marek Skarupski	.....	.....
	Title/degree/name and surname	grade	signature

*For the purposes of archival thesis qualified to:**

a) category A (perpetual files)

b) category BE 50 (subject to expertise after 50 years)

** delete as appropriate*

stamp of the faculty

Wrocław, 2022

Spis treści

Rozdział 1

Wstęp

Rozpoznawanie mowy jest problemem z gałęzi uczenia maszynowego, w którym modele wykorzystują do predykcji dane dźwiękowe. Istnieje bardzo szerokie spektrum zastosowań, które możemy zaobserwować nawet w życiu codziennym, gdzie rozpoznawanie mowy ma zastosowanie. Automatyczne generowanie napisów pod filmami w mediach społecznościowych, konwersja mowy na tekst w telefonach komórkowych lub rozpoznawanie konkretnych komend, uruchamiających funkcje przez różne urządzenia elektroniczne (np. telefony komórkowe) to tylko jedne z nielicznych przykładów gdzie możemy wykorzystać algorytmy rozpoznawania mowy.

Opisywanym tematem interesowano się już w poprzednim stuleciu, gdzie wielką popularnością cieszyły się modele bazujące na ukrytych łańcuchach Markowa (por. [1, 16]), a pierwsze systemy wykorzystujące je pojawiały się już w 1975 roku (p. [11]). Od tego czasu rozwiązano już problem rozpoznawania cyfr z dokładnością do 2–3% błędu tekstowego, gdzie cyfry mówione były w naturalny sposób przez niezależnych lektorów [16]. Wykorzystywano je również w problemach, w których słowniki osiągały nawet 1000 słów przy odpowiednich ograniczeniach gramatycznych. Dokładność rozpoznanych słów szacowano wtedy na 96% (p. [11]). Częste było wykorzystanie mieszanin rozkładów Gaussa (p. [1]) do estymacji prawdopodobieństw emisji stanów (por. [12]).

Wyzwanie rozpoznawania mowy, w przypadku wielkich słowników wypowiadanych przez niezależnych lektorów w sposób ciągły jest jednak bardzo zaawansowanym problemem i zaobserwowano, że same ukryte łańcuchy Markowa nie radzą sobie z nimi za dobrze [12, 4]. Obecnie najpopularniejsze systemy rozpoznawania mowy tj. Google API i Microsoft API wykorzystują rozbudowane modele akustyczne, bazujące na głębokich sieciach neuronowych do radzenia sobie z tym zadaniem. Co ciekawe Microsoft API w swoich rozwiązaniach również wykorzystuje ukryte łańcuchy Markowa w fazie dekodowania wyników [13].

W rozpatrywanym problemie możemy wyszczególnić przypadek, w którym dane zawierają zakłócenia. Obecność szumów w znaczący sposób może pogorszyć wyniki modelu. Wykorzystywano tu różne podejścia m. in. w 1990 roku A. P. Varga i R. K. Moore stosowali dekompozycje sygnału dźwiękowego na osobno, mowę i szum [19]. Lepsze podejście zaproponowali Gales i Young [5] przetwarzając sygnał za pomocą współczynników cepstralnych MFCC [9].

W pracy zajmiemy się zagadnieniem rozpoznawania izolowanych słów (ang. *Isolated Word Recognition*). Polega ono na rozpoznaniu i klasyfikacji sygnałów, w których pojawia się pojedyncza fraza. Istnieją publiczne zbiory do modelowania tego typu zadań, m. in. *Speech Command Dataset*, czyli zbiór danych oferowanych przez Google (p. [20]). W czołówce najlepszych modeli trenowanych na danym zbiorze dominują algorytmy uczenia głębokiego

(p. [14]), których wyniki potrafią osiągać ponad 95% dokładności.

Szczególnie będzie nas interesowało rozpoznawanie mowy w obecności szumów. Publiczne zbiory danych charakteryzują się jednak naturalnymi zakłóceniami wynikającymi często z czynników ludzkich. Chcąc przeprowadzić analizę w sposób bardziej kontrolowany stworzymy własny pomniejszony zbiór danych przy pomocy interfejsu gTTS (ang. *Google Text-to-Speech*, p. [7]) na wzór *Speech Command Dataset*. Po wygenerowaniu obserwacji będziemy mogli w dowolny sposób wprowadzać do nich zakłócenia.

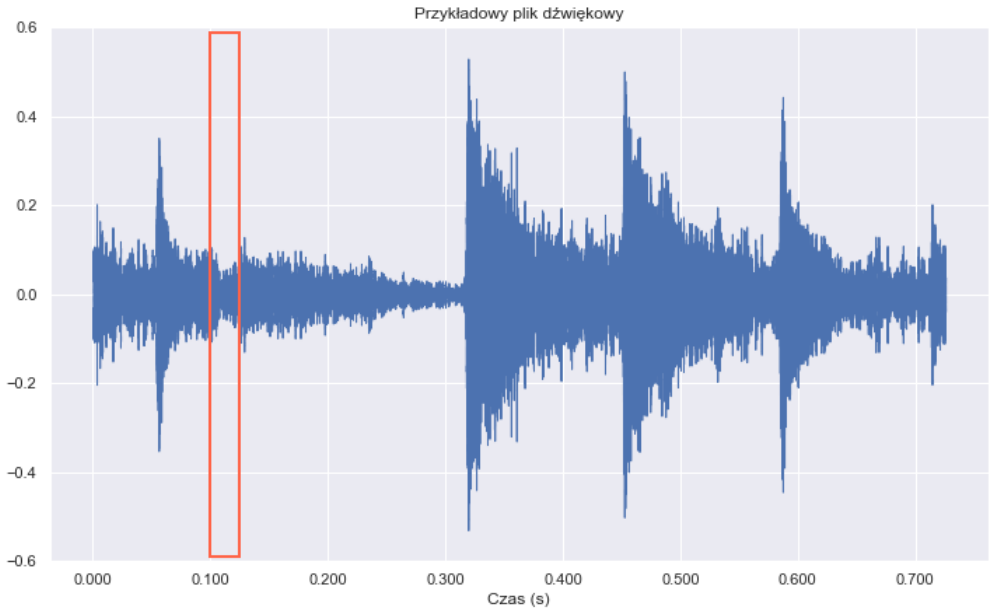
W rozdziale 2 przedstawimy wszystkie niezbędne narzędzia matematyczne potrzebne do zrozumienia działania naszego modelu i implementacji przepływu danych. Następnie w rozdziale 3 omówimy szczegóły implementacji, dla jakich parametrów były przeprowadzane symulacje i jak prezentują się wyniki w obecności zakłóceń. Przedstawimy również własną metodę, która powinna zwiększyć ogólnie dokładność modelu Markowskiego w przypadku analizy danych z różnymi rodzajami zakłóceń.

Rozdział 2

Teoria

2.1 Analiza sygnału dźwiękowego

Chcemy wykorzystać zdyskretyzowany sygnał dźwiękowy, czyli ciąg wartości $\{x_k\}_{k=0}^{n'-1}$, o którym wiadomo, że częstotliwość próbkowania wynosi sr. Będziemy chcieli przesuwać okno o szerokości S co krok R. Dokonujemy tego, aby wyliczać i analizować informacje o sygnale w konkretnym przedziale czasu.

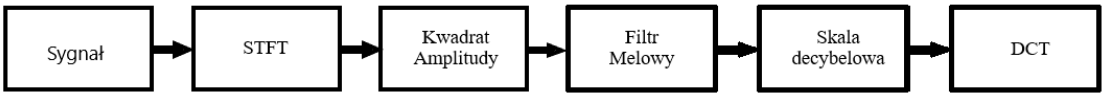


Rysunek 2.1: Przykład sygnału dźwiękowego zdyskretyzowanego z częstotliwością próbkowania $sr = 22050$ Hz. Czerwonym kolorem zaznaczamy przykładowe okno. Źródło: Opracowanie własne

W rozpoznawaniu mowy często przyjmuje się, że szerokość okna $S = 25$ms, z krokiem $R = 10$ms. Wartość S wynika, z faktu że w 25ms powinna mieścić się informacja o wypowiadanej zgłosce. Warto zwrócić uwagę, że poszczególne okna będą na siebie nachodzić.

2.1.1 Współczynniki MFCC

Współczynniki MFCC (ang. *Mel-frequency Cepstral Coefficients*) są nie tylko bardzo popularnym rozwiązaniem stosowanym do przetwarzania dźwięku, ale również pokazano, że dobrze radzą sobie w obecności zakłóceń (p. [5]). Metoda ta wykorzystuje zalety skali melowej, w której odległości między częstotliwościami są percepcyjnie równe. Ponadto korzystając z własności cepstrum otrzymujemy najistotniejsze informacje o tzw. *obwiedni sygnału*. Wyliczenie MFCC możemy sprowadzić do następujących etapów (p. Rysunek 2.2).



Rysunek 2.2: Etapy procesu wyliczenia współczynników MFCC.
Źródło: Opracowanie własne

Dyskretna Transformata Fouriera

Pierwszym etapem jest wyliczenie dyskretnej transformaty Fouriera (ang. *Discrete Fourier Transform*, DFT) dla każdego okna naszego sygnału $\{x_n\}$. Oznacza to, że będziemy wyliczać tzw. krótkoczasową transformatę Fouriera (ang. *Short-time Fourier Transform*, STFT). Uzyskamy w ten sposób macierz

$$\bar{\mathbf{X}}(f) = \left[\bar{X}_1(f)\, \bar{X}_2(f) \dots\, \bar{X}_k(f)\right], \tag{2.1}$$

gdzie transformata m-tego okna będzie określona wzorem:

$$\bar{X}_m(f) = \sum_{n=0}^{n'-1} x_n g(n - mR) \cdot e^{-2\pi i f n}, \tag{2.2}$$

gdzie:

f - częstotliwość

R - wielkość kroku okna

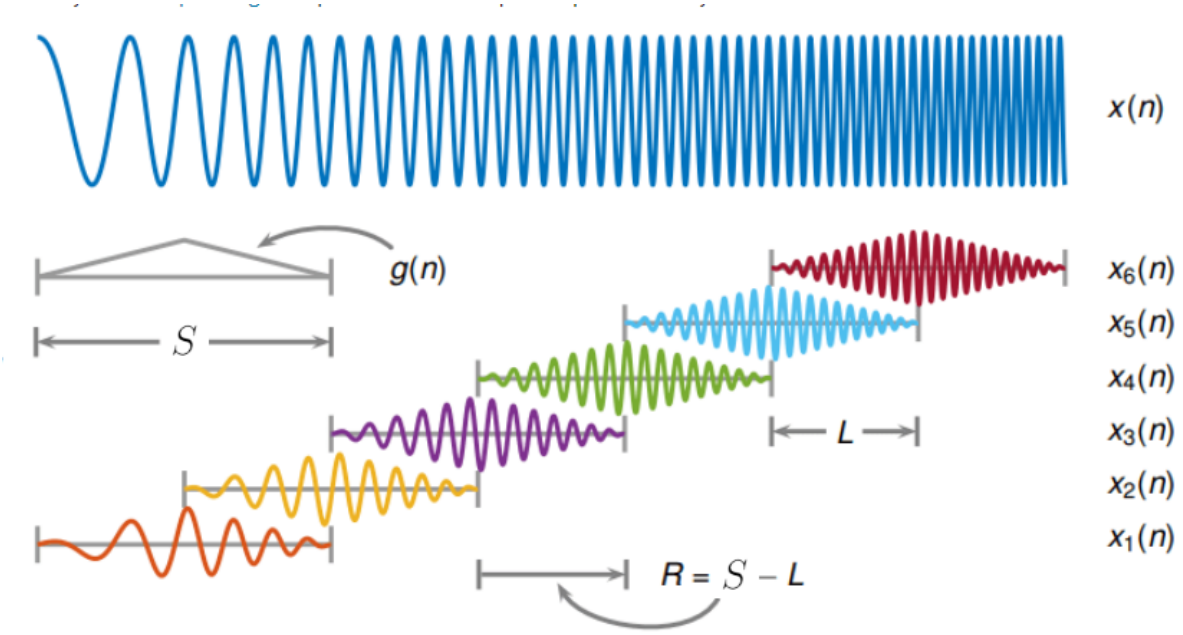
n' - liczba próbek

$g(n)$ - funkcja okienkowa

Będziemy zakładać, że otrzymana w ten sposób macierz $\bar{\mathbf{X}}$ będzie miała wymiary $p \times k$. Jak widać na rysunku (2.3) $g(n)$ może być oknem trójkątnym, określony wzorem:

$$g(n) = 1 - \left| \frac{n - \frac{S-1}{2}}{\frac{S-1}{2}} \right|. \tag{2.3}$$

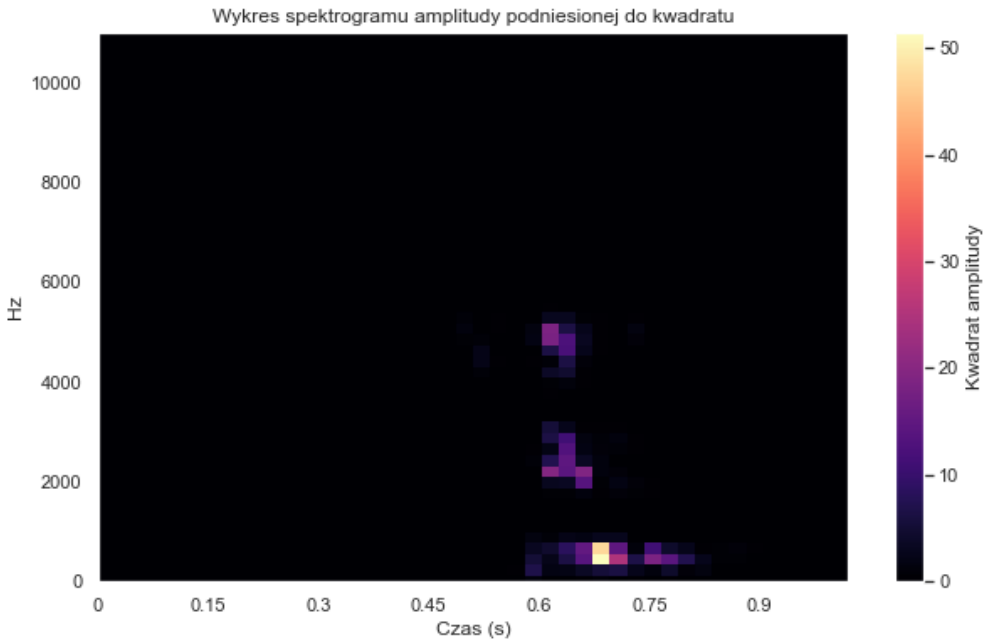
$g(n - mR)$ odpowiada za przesunięcie centrum okna o mR jednostek w prawo, tak by odfiltrowane zostało m-te okno. Następnie w zależności od wartości f będą wyznaczane kolejne wartości dyskretnej transformaty Fouriera.



Rysunek 2.3: Wizualizacja działania krótkoczasowej transformaty Fouriera.
Źródło: `https://www.mathworks.com/help/signal/ref/stft.html`

Widmo kwadratu amplitudy

Pojedynczą wartością macierzy $\bar{\mathbf{X}}$ jest liczba zespolona. Jeśli na wszystkie wartości nałożymy moduł i podniesiemy do kwadratu uzyskamy w ten sposób widmo kwadratu amplitudy nazywane również spektrogramem.



Rysunek 2.4: Spektrogram sygnału.
Źródło: Opracowanie własne

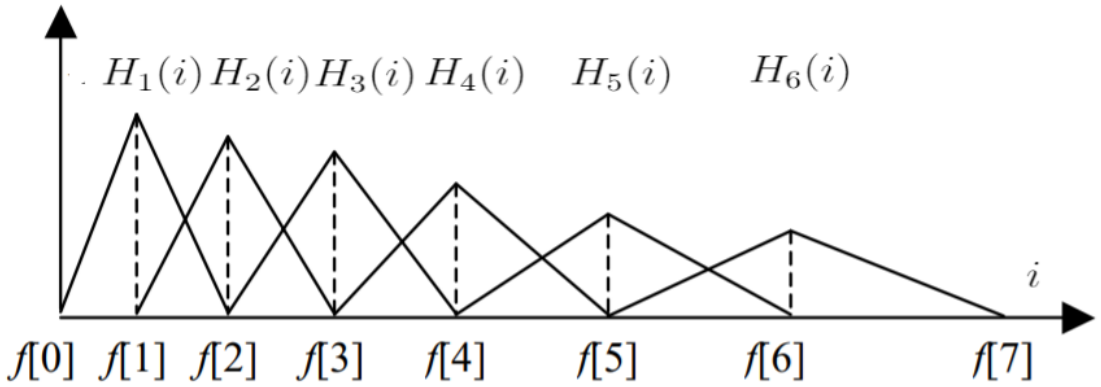
Jak widzimy na rysunku (2.4), możemy przedstawić wykres spektrogramu krótkoczasowej transformaty Fouriera umieszczając na osi poziomej czas, wzdłuż osi pionowej częstotliwość, a na ich przecięciu wartość kwadratu amplitudy uzyskanej w tym punkcie. Wysoka wartość amplitudy oznacza, że odpowiednia częstotliwość sygnału w danym momencie odgrywa dużą rolę.

Filtry melowe

Odczuwalnie ludzie słyszą dźwięki logarytmicznie. Znaczy to tyle, że dużo lepiej potrafimy usłyszeć różnice w dźwięku o częstotliwości 300 Hz i 400 Hz, niż w dźwięku o częstotliwości 10 100 Hz i 10 200 Hz, mimo że pomiędzy pierwszą i drugą parą jest różnica 100 Hz. W skali melowej odległości między częstotliwościami dużo lepiej oddają rzeczywistą wartość. Niech funkcja $B(f)$ oznacza wartość częstotliwości f w skali melowej wtedy:

$$B(f) = 1125 \ln\left(1 + \frac{f}{700}\right). \tag{2.4}$$

Wykorzystując skalę melową skonstruujemy d filtrów trójkątnych, odpowiedzialnych za „rozciągnięcie" wartości wzdłuż osi częstotliwości, aby uzyskać widmo odpowiadające skali melowej. Na początku musimy określić f_l, czyli minimalną częstotliwość i f_h – maksymalną częstotliwość. Często przyjmuje się $f_l = 0$ oraz $f_h = \frac{sr}{2}$. Intuicja jest taka, że generujemy d równo oddalonych od siebie punktów między $B(f_l)$ i $B(f_h)$, czyli w skali melowej. Nazwijmy je $f[i]$. Pomiędzy każdymi trzema punktami $f[i]$, $f[i+1]$ oraz $f[i+2]$, $i = 0, 1, \ldots, d-1$ konstruujemy okna trójkątne, przy czym $f[0] = B(f_l)$ i $f[d+1] = B(f_h)$. Na koniec dla każdej wartości $B(f) \in [B(f_l), B(f_h)]$ wracamy do naszej częstotliwości. Czyli nakładamy odwrotną transformację B^{-1}.



Rysunek 2.5: Zespół filtrów trójkątnych dla $d = 6$ wykorzystywanych do wyliczenia cepstrum melowego.
Źródło: Spoken Language Processing [9]

Na rysunku (2.5) widzimy filtry trójkątne, których wartości są przetransformowane do częstotliwości w Hz. Jak łatwo zauważyć wraz ze wzrostem częstotliwości i przedziały filtrów zwiększają się. Z tego powodu odległości pomiędzy wysokimi częstotliwościami będą się „rozciągać". Punkty graniczne $f[i]$ są określone wzorem:

$$f[i] = B^{-1}\left(B(f_l) + i\frac{B(f_h) - B(f_l)}{d+1}\right) \quad i = 0, 1, \ldots, d+1. \tag{2.5}$$

Chcemy konstruować macierz filtrów trójkątnych:

$$\mathbf{H}(i) = [H_1(i)\, H_2(i) \dots\ H_d(i)]' \quad i = 1, 2, \dots, p; \tag{2.6}$$

gdzie m-ty filtr będzie określony wzorem:

$$H_m(i) = \begin{cases} \dfrac{i - f[m-1]}{f[m] - f[m-1]}, & f[m-1] \leq i \leq f[m] \\ \dfrac{f[m+1] - i}{f[m+1] - f[m]}, & f[m] < i \leq f[m+1] \\ 0, \quad \text{poza tym} \end{cases} \tag{2.7}$$

Wtedy macierz $\mathbf{H}$ będzie miała wymiary $d \times p$. Stąd macierz:

$$\bar{Y} = \mathbf{H} \cdot \bar{\mathbf{X}} \tag{2.8}$$

ma wymiary $d \times k$ i reprezentuje tzw. *mel spektrogram*, czyli widmo po filtracji melowej. W tym momencie wiersze odpowiadają za konkretne wartości filtru $H_m(i)$, a kolumny to kolejne okna czasowe.

Skala decybelowa

Nasz mel spektrogram $\bar{Y}$ przekształcamy do skali decybelowej tzn.

$$Y = 10 \cdot \log_{10} \bar{Y}. \tag{2.9}$$

Możemy rzucić okiem jak teraz prezentuje się nasze widmo.

Jak widzimy na rysunku (2.6) wartości amplitud są bardziej rozciągnięte wzdłuż osi pionowej na dolnym wykresie. Skala decybelowa skutecznie poprawia obraz sygnału. Im mniejsza wartość w decybelach, tym mniejsza amplituda danej częstotliwości.

Dyskretna Transformata Kosinusowa

Ostatnim krokiem jest wyznaczenie dyskretnej transformaty kosinusowej (ang. *Discrete Cosine Transform*, DCT), a dokładnie krótkoczasowej transformaty kosinusowej. Uzyskujemy macierz:

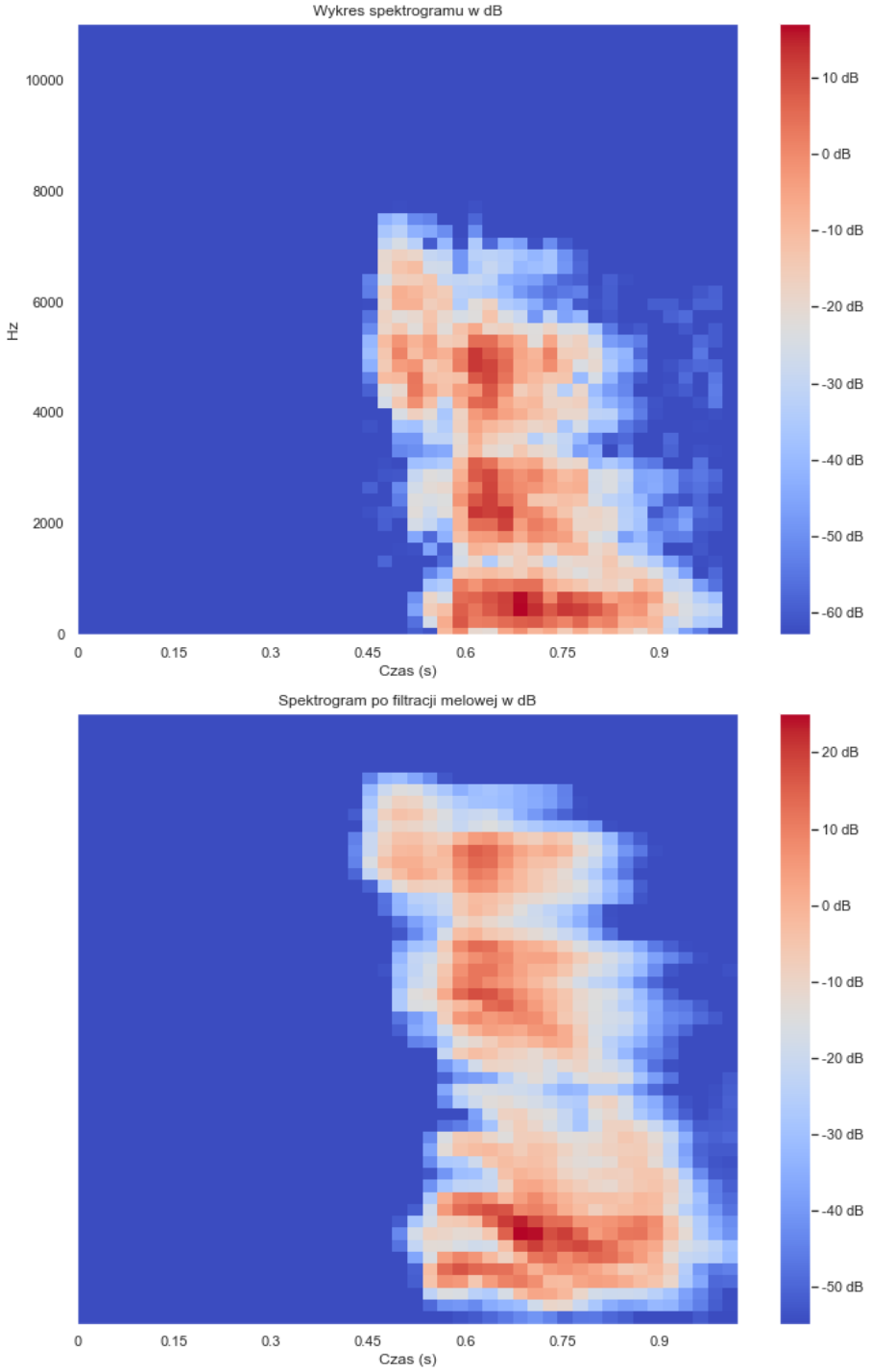
$$\mathbf{X}(i) = [X_1(i)\, X_2(i)\ \dots\ X_k(i)] \quad i = 1, 2, \dots, d;$$

gdzie m-ta transformata wynosi:

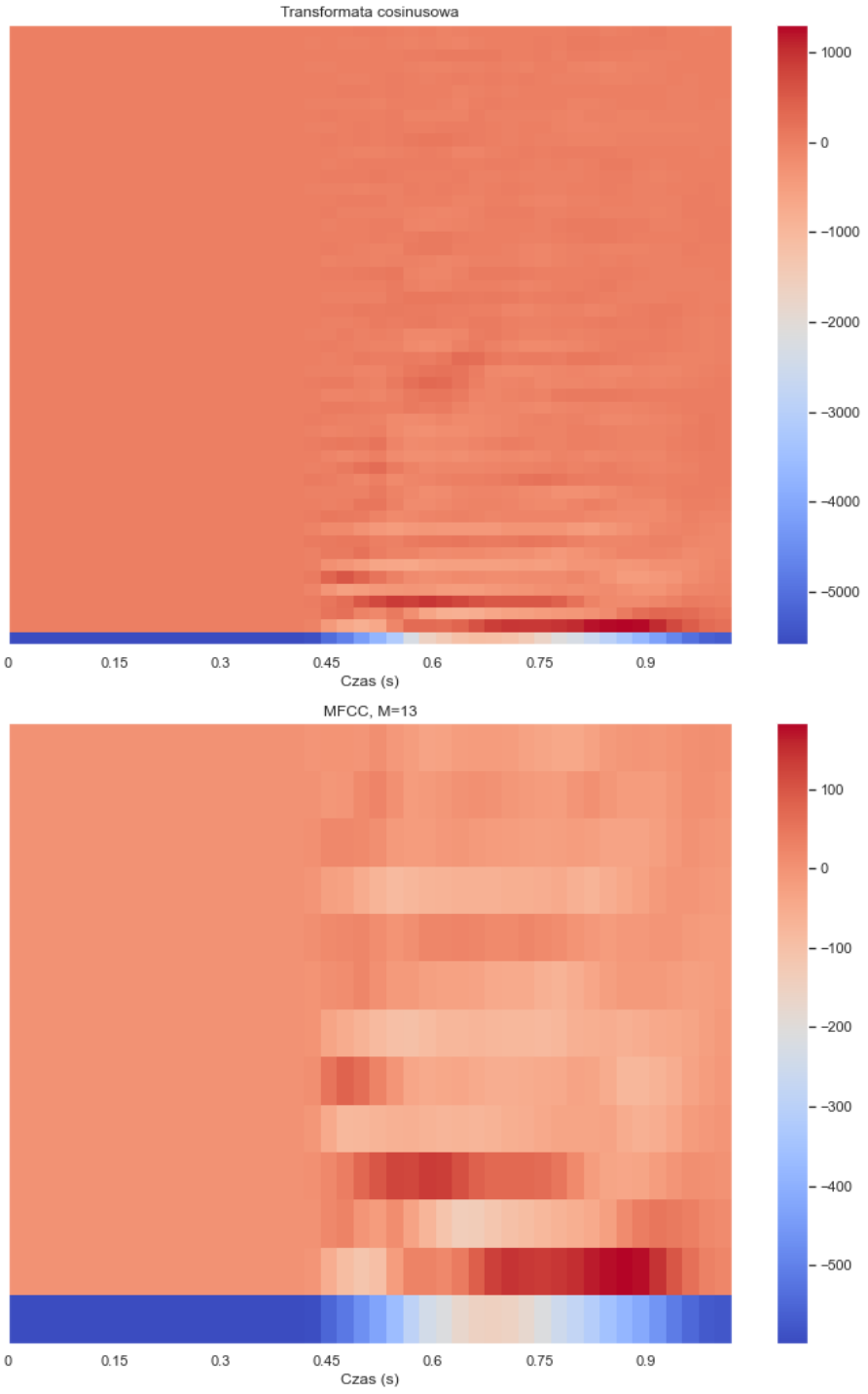
$$X_m(i) = \sum_{j=1}^{k} y_{ij} \cos\left[\frac{\pi}{k} i \left(j - \frac{1}{2}\right)\right] \quad i = 1, 2, \dots, d; \tag{2.10}$$

gdzie $Y = (y_{ij})$ jest naszym mel spektrogramem w skali decybelowej. Z uzyskanej macierzy wybieramy pierwsze M wierszy, które będą stanowiły współczynniki MFCC. Wybieramy M pierwszych cech, ponieważ niosą ze sobą najwięcej istotnych informacji. Możemy to zauważyć na rysunku (2.7). Widmo z prawej strony zarysem przypomina te z rysunku (2.6). Jednak wraz ze wzrostem wartości wzdłuż osi pionowej widmo coraz mniej się zmienia. Dlatego też dla ułatwienia obliczeń i pozbycia się szumu wybiera się pierwsze, najistotniejsze współczynniki. W zagadnieniach rozpoznawania mowy często przyjmuje się M pomiędzy 13, a 39. Na rysunku (2.7) z prawej strony widzimy przykład dla $M = 13$. Tutaj każdy indeks w czasie odpowiada wektorowi cech spektralnych.

Tak wyznaczone współczynniki MFCC będą stanowiły dane wejściowe dla naszego modelu.



Rysunek 2.6: Porównanie spektrogramu i mel spektrogramu w skali decybelowej. Źródło: Opracowanie własne



Rysunek 2.7: U góry wizualizacja macierzy $\mathbf{X}$ czyli widma po DCT. Na dole pierwsze M istotnych cech.
Źródło: Opracowanie własne

2.2 Modele Markowa

Niech $(\Omega, \mathcal{F}, P)$ będzie przestrzenią probabilistyczną oraz Q będzie przeliczalnym zbiorem stanów, tj. $Q \in \{1, \ldots, N\}$. Niech $\{q_t, t \in \mathcal{T}\}$, gdzie $\mathcal{T}$ to przeliczalny zbiór indeksów, jest dyskretnym procesem stochastycznym przyjmującym wartości z Q.

Definicja 2.1. Jeśli proces $\{q_t, t \in \mathcal{T}\}$ spełnia tzw. *własność Markowa*, tzn.

$$P(q_{t+1}|q_t, q_{t-1}, \ldots, q_0) = P(q_{t+1}|q_t), \tag{2.11}$$

to mówimy, że proces ten jest *łańcuchem Markowa* pierwszego rzędu.

Z równania (2.11) wnioskujemy, że w łańcuchach Markowa stan przyszły zależy tylko i wyłącznie od teraźniejszości. Chcąc opisywać powyższą klasę modeli wprowadzimy następujące oznaczenia.

Definicja 2.2. *Macierzą przejścia* $A = (a_{ij})\, \forall i, j \in Q$, gdzie

$$a_{ij} \geq 0 \quad \text{oraz} \quad \sum_{j \in Q} a_{ij} = 1 \quad \forall i, j \in Q,$$

nazywamy macierz, której elementy a_{ij} oznaczają prawdopodobieństwo przejścia ze stanu i do stanu j, czyli:

$$a_{ij} = P(q_{t+1} = j | q_t = i) \qquad \forall t \in \mathcal{T},\ \forall i, j \in Q. \tag{2.12}$$

Definicja 2.3. Niech π będzie *wektorem rozkładu początkowego*, takim że jego elementy $\pi_i, i \in Q$:

$$\pi_i = P(q_0 = i). \tag{2.13}$$

Możemy definiować łańcuch Markowa jako krotkę $\theta = (Q, A, \pi)$.

Wiedząc już czym są łańcuchy Markowa możemy rozszerzyć nasze zagadnienie definiując *Ukryte Łańcuchy Markowa*.

2.3 Ukryte Modele Markowa

Mając łańcuch Markowa zdefiniujemy model ukrytych łańcuchów Markowa, który będzie nam służył w problemie rozpoznawania mowy do analizy danych i generowania predykcji. Weźmy łańcuch Markowa $\theta = (Q, A, \pi)$. Będziemy zakładać, że w chwili t nie możemy bezpośrednio zaobserwować stanów z przestrzeni Q. Dlatego będziemy mówić, że proces stochastyczny q_t będzie *ukryty*. Zakładamy jednak, że w chwili t emitowany jest z pewnym prawdopodobieństwem proces stochastyczny $\{O_t, t \in \mathcal{T}\}$ zdefiniowany na przestrzeni stanów $\mathcal{O}$, którego stany są jawne, czyli widoczne dla obserwatora. Będzie to nasz ciąg obserwacji. Mówimy, że $\mathcal{O}$ jest przestrzenią stanów jawnych, a Q – przestrzenią stanów ukrytych.

Definicja 2.4. Niech proces stochastyczny q_t będzie zdefiniowany na przestrzeni stanów ukrytych Q. Proces stochastyczny O_t zdefiniowany na przestrzeni stanów jawnych $\mathcal{O}$ spełnia tzw. zależność stanu jawnego w chwili t od stanu ukrytego w chwili t, tzn:

$$P(O_t|q_t, q_{t-1}, \ldots, q_0, O_{t-1}, O_{t-2}, \ldots, O_1) = P(O_t|q_t) \tag{2.14}$$

dla każdego $t \in \mathcal{T}$.

Własność (2.14) będzie determinowała, tzw. prawdopodobieństwa emisji.

Definicja 2.5. Niech $\mathcal{O}, Q$ – przeliczalne przestrzenie stanów. Prawdopodobieństwo emisji b_{ij} będziemy określać jako:

$$b_{ij} = P(O_t = j|q_t = i) \qquad \forall i \in Q, \forall j \in \mathcal{O}. \tag{2.15}$$

Uzyskane prawdopodobieństwa możemy opisać za pomocą macierzy emisji:

$$B = (b_{ij}) \qquad \forall i \in Q, \forall j \in \mathcal{O}. \tag{2.16}$$

Z definicji wynika bezpośrednio:

$$\sum_{j \in \mathcal{O}} b_{ij} = 1 \quad \forall i \in Q. \tag{2.17}$$

Uwaga 2.6. Oznaczmy wektor obserwacji w chwili t jako o_t. Zauważmy, że są one cechami spektralnymi, które są ciągłe. W tym przypadku proces stochastyczny O_t jest zdefiniowany na zbiorze nieprzeliczalnym $\mathcal{O}$, a prawdopodobieństwo z równania (2.15) staję się funkcją zależną od wartości o_t. Będziemy wtedy pisać:

$$b_i(o_t) = P(O_t = o_t|q_t = i) \qquad \forall i \in Q, \forall o_t \in \mathcal{O}. \tag{2.18}$$

Definicja 2.7. Niech proces stochastyczny q_t zdefiniowany na przestrzeni stanów ukrytych Q spełnia własność Markowa i proces stochastyczny O_t spełnia zależność stanu jawnego w chwili t od stanu ukrytego w chwili t. Wtedy krotka:

$$\lambda = (Q, A, \pi, \mathcal{O}, B), \tag{2.19}$$

gdzie:

A - macierz przejścia procesu q_t

π - wektor rozkładu początkowego procesu q_t

B - macierz emisji

W jednoznaczny sposób definiuje ukryty łańcuch Markowa.

Jak pokazano w pracach naukowych (p. [15], [11]) ukryte łańcuchy Markowa mają duży potencjał w analizie danych sekwencyjnych, czyli m.in. rozpoznawaniu mowy.

2.4 Procedura forward–backward

Niech $O = [O_1, O_2, \ldots, O_T]$ oraz $q = [q_1, q_2, \ldots, q_T]$. Interesuje nas wyznaczenie $P(O|\lambda)$, czyli prawdopodobieństwa, że zajdzie sekwencja O znając parametry ukrytego łańcucha Markowa λ. Zaprezentujmy jak mogłoby wyglądać podejście naiwne, czyli z twierdzenia o prawdopodobieństwie całkowitym:

$$P(O|\lambda) = \sum_{q_1, q_2, \ldots, q_T} P(O|q, \lambda) P(q|\lambda). \tag{2.20}$$

Proces stochastyczny q_t może przyjąć wartości od 1 do N. Łatwo możemy zauważyć, że równanie (2.20) ma złożoność obliczeniową $O(N^T)$, czyli rosnącą w sposób eksponencjalny. Chcemy zaradzić temu problemowi wykorzystując algorytm forward–backward, wykorzystujący programowanie dynamiczne.

2.4.1 Algorytm forward

Wprowadźmy oznaczenie $O_{i:j} = [O_i, O_{i+1}, \ldots, O_j]$. W szczególności zauważmy, że $O = O_{1:T} = [O_1, O_2, \ldots, O_T]$. Interesuje nas policzenie prawdopodobieństwa łącznego:

$$P(q_t = j, O_{1:t}|\lambda) \quad \forall t = 1, 2, \ldots, T, \ \forall j \in Q; \tag{2.21}$$

czyli prawdopodobieństwo jakie determinuje nasz ciąg obserwacji do chwili t, zakładając, że w chwili t stan ukryty przyjmuje wartość j, znając przy tym parametry łańcucha λ. Wprowadzimy oznaczenie, tzw. *prawdopodobieństwa forward*:

$$\alpha_t(j) = P(O_{1:t}, q_t = j|\lambda) = \tag{2.22}$$

$$= \left[\sum_{i=1}^{N} \alpha_{t-1}(i)a_{ij}\right] b_j(o_t) \quad \forall t = 2, 3, \ldots T, \ \forall j \in Q, \tag{2.23}$$

z warunkiem początkowym:

$$\alpha_1(j) = \pi_j b_j(o_1). \tag{2.24}$$

Zauważmy, że teraz możemy zapisać:

$$P(O|\lambda) = \sum_{j=1}^{N} \alpha_T(j). \tag{2.25}$$

Da się pokazać, że algorytm ma złożoność obliczeniową $O(TN^2)$. W celu pokazania skali różnicy jaką uzyskaliśmy przyjmijmy dla przykładu, że długość sekwencji $T = 100$ i ilość ukrytych stanów $N = 10$. Wtedy naiwna metoda ma złożoność obliczeniową $O(10^{100})$, w tym samym momencie za pomocą algorytmu forward otrzymujemy $O(100 \cdot 10^2)$, czyli $O(10000)$.

2.4.2 Algorytm backward

Na potrzeby algorytmu Bauma–Welcha, który będzie wykorzystywany do estymacji parametrów modelu zdefiniujemy sobie następujące prawdopodobieństwo:

$$\beta_t(i) = P(O_{t+1:T}|q_t = i, \lambda). \tag{2.26}$$

Można pokazać, że wynosi ono:

$$\beta_t(i) = \sum_{j=1}^{N} a_{ij} b_j(o_{t+1})\beta_{t+1}(j) \quad \forall t = 1, 2, \ldots, T-1, \ \forall i \in Q, \tag{2.27}$$

z warunkiem początkowym:

$$\beta_T(i) = 1 \quad \forall i \in Q. \tag{2.28}$$

Czyli cofamy się z naszą wiedzą od obserwacji T do obserwacji $t+1$. Chcemy wiedzieć ile będzie wynosiło prawdopodobieństwo tego ciągu zdarzeń gdyby $q_t = i$.
Warto zauważyć również, że:

$$P(O|\lambda) = \sum_{i=1}^{N} \alpha_T(i) = \sum_{i=1}^{N} \pi_i b_i(o_1)\beta_1(i). \tag{2.29}$$

Poprzez wprowadzenie procedury forward–backward jesteśmy w stanie przedstawić działanie algorytmu Bauma–Welcha.

2.5 Algorytm Bauma–Welcha

Algorytm Bauma–Welcha jest specjalnym przypadkiem algorytmu EM (ang. *Expectation–maximization*). Będziemy go używać do estymacji najlepszych parametrów ukrytego łańcucha Markowa $\hat{\lambda}$, którego $P(O|\hat{\lambda})$ będzie większe od $P(O|\lambda)$. Do tego zadania wykorzystamy prawdopodobieństwa forward i backward. Po ich wyliczeniu definiujemy zmienne:

$$\xi_t(i,j) = P(q_t = i, q_{t+1} = j|O,\lambda) = \frac{P(q_t = i, q_{t+1} = j, O|\lambda)}{P(O|\lambda)} \tag{2.30}$$

$$= \frac{\alpha_t(i)a_{ij}b_j(o_{t+1})\beta_{t+1}(j)}{P(O|\lambda)} \quad \forall i,j \in Q. \tag{2.31}$$

Jest to prawdopodobieństwo przejścia ze stanu i do stanu j po czasie t, pod warunkiem ciągu obserwacji O i łańcucha Markowa λ. Poza tym, niech

$$\gamma_t(i) = P(q_t = i|O,\lambda) = \frac{\alpha_t(i)\beta_t(i)}{P(O|\lambda)} = \sum_{j=1}^{N} \xi_t(i,j), \tag{2.32}$$

będzie prawdopodobieństwem, że w chwili t stan ukryty przyjmuje wartość i, pod warunkiem ciągu obserwacji O i łańcucha Markowa λ. Mając to i wykorzystując algorytm Bauma–Welcha wiemy, że zaaktualizowane parametry π_i^*, a_{ij}^* i $b_i^*(k)$ mają wartość:

$$\pi_i^* = \gamma_1(i) \quad 1 \leq i \leq N, \tag{2.33}$$

$$a_{ij}^* = \frac{\sum_{t=1}^{T-1} \xi_t(i,j)}{\sum_{t=1}^{T-1} \gamma_t(i)}, \tag{2.34}$$

$$b_i^*(k) = \frac{\sum_{t=1}^{T} \mathbb{1}(O_t = k)\gamma_t(i)}{\sum_{t=1}^{T} \gamma_t(i)}. \tag{2.35}$$

Jeśli zdefiniujemy ukryty łańcuch Markowa λ i będziemy reestymować go jako $\hat{\lambda}$ przy pomocy równań (2.33), (2.34) i (2.35), to można pokazać, że albo:

1. Początkowy model λ definiuje punkt krytyczny funkcji największej wiarygodności, w rezultacie $\hat{\lambda} = \lambda$.

2. Model $\hat{\lambda}$ uzyskuje większą wartość funkcji wiarygodności, tzn.

$$P(O|\hat{\lambda}) > P(O|\lambda), \tag{2.36}$$

   czyli znaleźliśmy lepszy model, który może zostać wykorzystany do naszego problemu.

Ponadto, jeśli iteracyjnie będziemy reestymować parametry λ, będziemy mogli poprawić wyniki $P(O|\lambda)$, dopóki nie osiągniemy pewnej granicy górnej.

W ten sposób dowiedzieliśmy się w jaki sposób model HMM będzie się uczył optymalnych parametrów wynikających z danych.

2.6 Gaussowskie Ukryte Modele Markowa

Powiedzieliśmy już, że w zagadnieniu rozpoznawania mowy proces stochastyczny O_t odpowiada za wektor cech spektralnych, którego zbiór wartości jest skupiony na nieprzeliczalnym

zbiorze $\mathcal{O}$. Z równania (2.6) wiemy, że prawdopodobieństwo emisji staje się wtedy funkcją zależną od wartości o_t, czyli przypomnijmy:

$$b_i(o_t) = P(O_t = o_t | q_t = i) \qquad \forall i \in Q, \forall o_t \in \mathcal{O}. \tag{2.37}$$

W gaussowskich ukrytych modelach Markowa będziemy zakładać, że $b_i(o_t)$ pochodzą z wielowymiarowego rozkładu normalnego $\mathcal{N}(\mu_i, \Sigma_i)$, gdzie:

μ_i - wektor średnich cech spektralnych, pochodzących ze stanu ukrytego i,

Σ_i - macierz kowariancji cech spektralnych, pochodzących ze stanu ukrytego i,

o_t - wektor cech spektralnych długości M, np. współczynniki MFCC,

$i = 1, 2, \ldots, N$.

Stąd prawdopodobieństwo emisji i-tego stanu ukrytego będzie miało postać (por. [12, s. 848] ,[16, 11]):

$$b_i(o_t) = \frac{1}{\sqrt{|2\pi\Sigma_i|}} \exp\left[(o_t - \mu_i)'\Sigma_i^{-1}(o_t - \mu_i)\right]. \tag{2.38}$$

Oczywiście wielkości μ_i i Σ_i będą determinowane przez długość wektora cech spektralnych. Korzystając z metody największej wiarygodności jesteśmy w stanie pokazać, że estymatory μ_i i Σ_i mają postać:

$$\hat{\mu_i} = \frac{1}{M}\sum_{t=1}^{M} o_t, \tag{2.39}$$

$$\hat{\Sigma_i} = \frac{1}{M}\sum_{t=1}^{M}[(o_t - \mu_i)'(o_t - \mu_i)], \tag{2.40}$$

Uwaga 2.8. Z własności dyskretnej transformaty kosinusowej współczynniki MFCC są ze sobą nieskorelowane. Zatem dla uproszczenia obliczeń możemy założyć, że Σ_i jest macierzą diagonalną.

Uwaga 2.9. Ciągłość obserwacji procesu O_t sprawia, że musimy wprowadzić korektę w algorytmie Bauma–Welcha, ponieważ w równaniu (2.35) użyty do reestymacji parametru $b_i^*(k)$ indykator nie miałby sensu. Tym razem w każdej iteracji tego algorytmu będziemy korygować μ_i i Σ_i w następujący sposób:

$$\hat{\mu_i} = \frac{\sum_{t=1}^{T}\gamma_t(i)o_t}{\sum_{t=1}^{T}\gamma_t(i)}, \tag{2.41}$$

$$\hat{\Sigma_i} = \frac{\sum_{t=1}^{T}\gamma_t(i)(o_t - \mu_i)'(o_t - \mu_i)}{\sum_{t=1}^{T}\gamma_t(i)}. \tag{2.42}$$

Wiemy już wszystko, co jest niezbędne do zdefiniowania modelu. Pozostaje zbudować odpowiednią teorię dotyczącą zakłóceń.

2.7 Szumy losowe

Rozpatrzmy sygnał $\{N_t : t = 0, \pm 1, \pm 2, \dots\}$ jako proces stochastyczny o średniej 0, tzn.:

$$E[N_t] = 0 \quad \forall t \in \mathbb{R}. \tag{2.43}$$

Definicja 2.10. Funkcję autokowariancji procesu stochastycznego N_t definiujemy następująco:

$$r(h) = Cov[N_t, N_{t-h}] = E[N_t \cdot N_{t-h}], \quad \forall t, h \in \mathbb{R} \tag{2.44}$$

i zakładamy, że jest ona zależna wyłącznie od przesunięcia h.

Uwaga 2.11. Proces N_t, który spełnia (2.43) oraz (2.44) jest procesem stacjonarnym w słabym sensie.

Definicja 2.12. Niech proces stochastyczny N_t spełnia (2.43) i (2.44). Widmowa gęstość mocy $\phi(f)$ (ang. *Power Spectrum Density*, PSD, [17]) zdefiniowana jest jako krótkoczasowa dyskretna transformata Fouriera funkcji autokowariancji $r(h)$, tzn.

$$\phi(f) = \sum_{h=-\infty}^{\infty} r(h) e^{-i \cdot f \cdot h}, \tag{2.45}$$

gdzie f to częstotliwość.

W rozważanej pracy będziemy generować gaussowskie szumy losowe, będące procesami stochastycznymi N_t, których widmowa gęstość mocy $\phi(f)$ będzie proporcjonalna do wartości $(1/f)^\beta$. Przy czym skupimy się na następujących szumach (p. [3], [21]):

Definicja 2.13.

- $\beta = -2$, $\phi(f) \propto f^2$ – fioletowy szum.
- $\beta = -1$, $\phi(f) \propto f$ – niebieski szum.
- $\beta = 0$, $\phi(f)$ jest stała – biały szum.
- $\beta = 1$, $\phi(f) \propto f^{-1}$ – różowy szum.
- $\beta = 2$, $\phi(f) \propto f^{-2}$ – czerwony szum, znany również jako szum brownowski.

Mając wygenerowane szumy dla odpowiednich parametrów $\beta = -2, -1, 0, 1, 2$ będziemy je nakładać na czysty sygnał, aby wprowadzić zakłócenia.

2.8 Współczynnik zaszumienia SNR

Definicja 2.14. Moc sygnału $\{S_t, t \in \mathcal{T}\}$, który jest procesem stochastycznym definiujemy jako:

$$P = E|S_t|^2 \quad \forall t \in \mathcal{T}, \tag{2.46}$$

czyli jest to drugi moment rozkładu S_t.

Chcemy zmierzyć poziom zaszumienia sygnału dźwiękowego. Wykorzystamy do tego współczynnik SNR (ang. *Signal-noise ratio*, [10]) zdefiniowany następująco:

$$SNR = \frac{P_S}{P_N}, \tag{2.47}$$

gdzie

P_S - moc sygnału dźwiękowego,

P_N - moc szumu.

W momencie, gdy szum $\{N_t, t \in \mathcal{T}\}$ jest procesem stochastycznym o średniej 0 i wariancji σ_N^2, dla każdego $t \in \mathcal{T}$, wtedy równanie (2.47) przybiera postać:

$$SNR = \frac{P_S}{\sigma_N^2}. \tag{2.48}$$

Często poziom zaszumienia podaje się w dB. Skalujemy współczynnik SNR i dostajemy:

$$SNR_{dB} = 10\log_{10}\left(\frac{P_S}{\sigma_N^2}\right) = 10\log_{10}(P_S) - 10\log_{10}(\sigma_N^2). \tag{2.49}$$

Zwróćmy uwagę, że im mniejsze SNR, tym większy poziom zaszumienia. Wiedząc jak wyliczać współczynnik SNR, jesteśmy w stanie manipulować mocą zakłóceń w naszych sygnałach.

2.9 Miary dopasowania

Model matematyczny definiowany przez nas będzie dokonywał predykcji słów. Przyjmijmy, że obserwacje x pochodzą z klas c. Wtedy predykcje jesteśmy w stanie podzielić na cztery unikalne części:

- PP – ilość prawdziwie pozytywnych. Model prawidłowo twierdzi, że x pochodzą z klas c,
- PN – ilość prawdziwie negatywnych. Model prawidłowo twierdzi, że x nie pochodzą z klas c,
- FP – ilość fałszywie pozytywnych. Model niepoprawnie twierdzi, że x pochodzą z klas c,
- FN – ilość fałszywie negatywnych. Model niepoprawnie twierdzi, że x nie pochodzą z klas c.

Mając zdefiniowane powyższe oznaczenia możemy opisać następujące miary dopasowania (p. [6]).

Definicja 2.15. Dokładność modelu (ang. *Accuracy*) opisana jest wzorem:

$$\text{ACC} = \frac{PP + PN}{PP + PN + FP + FN}, \tag{2.50}$$

czyli stosunek obserwacji sklasyfikowanych poprawnie do ilości wszystkich obserwacji.

Definicja 2.16. Pełność modelu (ang. *Recall*) opisana jest wzorem:

$$\text{Recall} = \frac{PP}{PP + FN}, \tag{2.51}$$

czyli odsetek obserwacji prawidłowo sklasyfikowanych przez model.

Definicja 2.17. Precyzja modelu (ang. *Precision*) opisana jest wzorem:

$$\text{Precision} = \frac{PP}{PP + FP}, \tag{2.52}$$

czyli odsetek prawidłowych obserwacji prawidłowo sklasyfikowanych przez model.

Definicja 2.18. Miara F_1 modelu (ang. F_1 *score*) opisana jest wzorem:

$$F_1 = \frac{2}{\frac{1}{\text{Precision}} + \frac{1}{\text{Recall}}}, \tag{2.53}$$

czyli jest to średnia harmoniczna precyzji i pełności.

Badając każdą z przytoczonych metryk, jesteśmy w stanie analizować poprawność predykcji naszego modelu.

Rozdział 3

Symulacje

Wszystkie symulacje oraz narzędzia programistyczne zostały zaimplementowane z użyciem języka Python. Szczegóły oraz link do repozytorium znajdują się w Dodatku.

3.1 Generowanie danych

Interesuje nas zagadnienie rozpoznawania izolowanych słów (ang. *Isolated word recognition*). Istnieje kilka publicznych zbiorów danych, na których można sprawdzać swoje rozwiązania, np. *Speech Command Dataset* (p. [20]) czy zbiór pojedynczych nazw owoców (p. [8]).

W obu przypadkach jednak obserwacje mogą posiadać naturalne zakłócenia wynikające z czynników ludzkich lub technologicznych. Może to być zarówno niewyraźne mówienie podczas nagrywania, wykorzystywanie sprzętu słabej jakości lub nagrywanie w głośnym otoczeniu. Chcemy zasymulować warunki idealne, czyli obserwacje bez zakłóceń, a w następnej kolejności dopiero rozszerzać je o szum. W tym celu wykorzystamy narzędzie udostępniane przez API firmy Google, tj. *Google Text-to-Speech*, do którego dostęp uzyskamy za pomocą biblioteki gTTS. Działa ona w ten sposób, że wybieramy odpowiedni generator mowy, w naszym przypadku będzie to generator języka angielskiego z akcentem amerykańskim. Wpisujemy słowo, które nas interesuje, a następnie używamy metody, która generuje plik dźwiękowy z wypowiedzianym przez syntezator mowy zadanym słowem. Definiujemy interesujące nas klasy słów:

id	Słowo
0	down
1	go
2	left
3	no
4	off
5	on
6	right
7	stop
8	to
9	yes

Tabela 3.1: Rozpatrywane klasy słów.
Źródło: Opracowanie własne

Następnie generujemy zbiór treningowy, gdzie dla każdej klasy będzie N_{train} obserwacji. Uzyskamy w ten sposób zbalansowany zbiór danych. Cechuje go to, że wyniki dokładności są miarodajne. Gdyby w zbiorze jedna klasa dominowałaby nad innymi, model mógłby nauczyć się wyróżniać tylko ją, uzyskując duże wyniki ACC.

Chcemy w pewien sposób zróżnicować obserwacje. Na początku zdefiniujmy parametr *silent_ratio*. Za jego pomocą dodamy bufor do sygnału w postaci ciszy na początku i końcu. Zakładając, że długość sygnału wynosi n', wtedy na początek i koniec zestawiamy wektor zer wielkości

$$silent\_ratio \cdot \frac{n'}{2}.$$

Co więcej, do każdego sygnału dźwiękowego losujemy liczbę z rozkładu jednostajnego $\mathcal{U}(-rhs, rhs)$. Parametr przesunięcia horyzontalnego rhs (*random_horizontal_shift*) definiuje przedział liczb, mówiących o jaki procent przesunąć sygnał w prawo lub lewo, przy czym wartości ujemne oznaczają przesunięcie w lewo, a dodatnie – w prawo. W ten sam sposób generujemy zbiór testowy, mający $N_{\text{test}} = 0.2 \cdot N_{\text{train}}$ obserwacji dla każdej klasy słów. Na koniec wszystkie pliki normalizujemy do tej samej długości dodając na końcu ciszę.

W ten sposób uzyskujemy zbiór danych gotowy do ewaluacji modelu.

3.2 Przygotowanie przepływu danych

Do przygotowania danych użyjemy pakietu *librosa*. Najpierw importujemy sygnał dźwiękowy, a następnie używamy funkcji *librosa.mfcc* do wyliczenia $M = 13$ współczynników MFCC. Ustawiamy również parametry *win_length* i *hop_length*, które oznaczają odpowiednio długość okna S i długość skoku okna R.

W zagadnieniu rozpoznawania izolowanych słów interesuje nas poprawne przypisanie klasy słów do obserwacji, co można sprowadzić do równania:

$$\hat{w} = \underset{w}{\operatorname{argmax}} \left\{ P(w|O) \right\}, \tag{3.1}$$

gdzie:

w - klasa słowa. W naszym przypadku przyjmuje wartości od 0 do 9.

O - $(O_1, \ldots, O_T)$, wektor obserwacji, czyli cech spektralnych.

Z twierdzenia Bayes'a możemy zapisać:

$$\hat{w} = \underset{w}{\operatorname{argmax}} \left\{ \frac{P(O|w)P(w)}{P(O)} \right\}, \tag{3.2}$$

ponieważ mianownik nie zależy od w piszemy:

$$\hat{w} = \underset{w}{\operatorname{argmax}} \left\{ P(O|w)P(w) \right\}. \tag{3.3}$$

Prawdopodobieństwo $P(w)$ zazwyczaj wyliczane jest wykorzystując tzw. *modele językowe*, mówiące o prawdopodobieństwie występowania słowa w w sekwencji słów. W zagadnieniu rozpoznawania izolowanych słów zakłada się, że prawdopodobieństwo występowania konkretnej klasy słów jest takie samo jak występowanie innej. Możemy zatem pominąć ten czynnik.

Dysponujemy skończoną liczbą klas, dlatego każde słowo w możemy zaprezentować jako unikalny ukryty łańcuch Markowa λ. Wtedy równanie (3.3) po pominięciu czynnika $P(w)$ możemy zapisać jako:

$$\hat{\lambda} = \underset{\lambda}{\operatorname{argmax}} \{P(O|\lambda)\}, \tag{3.4}$$

czyli sprowadziliśmy nasz problem do znalezienia takiego ukrytego łańcucha Markowa λ odpowiadającego za słowo w, który maksymalizuje $P(O|\lambda)$ otrzymane z procedury forward (por. [15], [2]).

Pojedynczy łańcuch Markowa λ definiujemy za pomocą zaprogramowanej klasy *model_setup.gmmhmm*, implementującej Gaussowski ukryty łańcuch Markowa o warunkach początkowych zależnych od danych treningowych. Taka klasa zależy od parametrów *n_components*, czyli ilości stanów ukrytych oraz *n_iter* odpowiadającej za ilość iteracji algorytmu Bauma–Welcha. W liście tworzymy N_c instancji klasy *model_setup.gmmhmm*, gdzie N_c oznacza liczbę klas. Następnie i-ty model w liście trenujemy za pomocą metody *gmmhmm.fit* na obserwacjach ze zbioru treningowego odpowiadających i-tej klasie, $i = 0, 1, \ldots, N_c - 1$.

Chcąc dokonać ewaluacji na zbiorze testowym każda obserwacja jest poddawana metodzie *gmmhmm.transform*, która wyznaczy dla nas wartość $P(O|\lambda)$ i-tego modelu $i = 0, 1, \ldots, N_c - 1$. Wyniki powstają z równania:

$$i = \underset{i}{\operatorname{argmax}} \{P(O|\lambda_i)\} \quad i = 0, 1, \ldots, N_c - 1, \tag{3.5}$$

gdzie λ_i jest ukrytym łańcuchem Markowa i-tej klasy słowa. Model, który składa się z listy λ_i, $i = 0, 1, \ldots, N_c - 1$ powstaje dzięki klasie *model_setup.IwrGaussianHMMModel*. Mając to możemy dokonywać ewaluacji za pomocą metody *IwrGaussianHMMModel.scores*, która na zadanym zbiorze wyliczy dokładność, pełność, precyzję i miarę F_1.

3.3 Wybór modelu

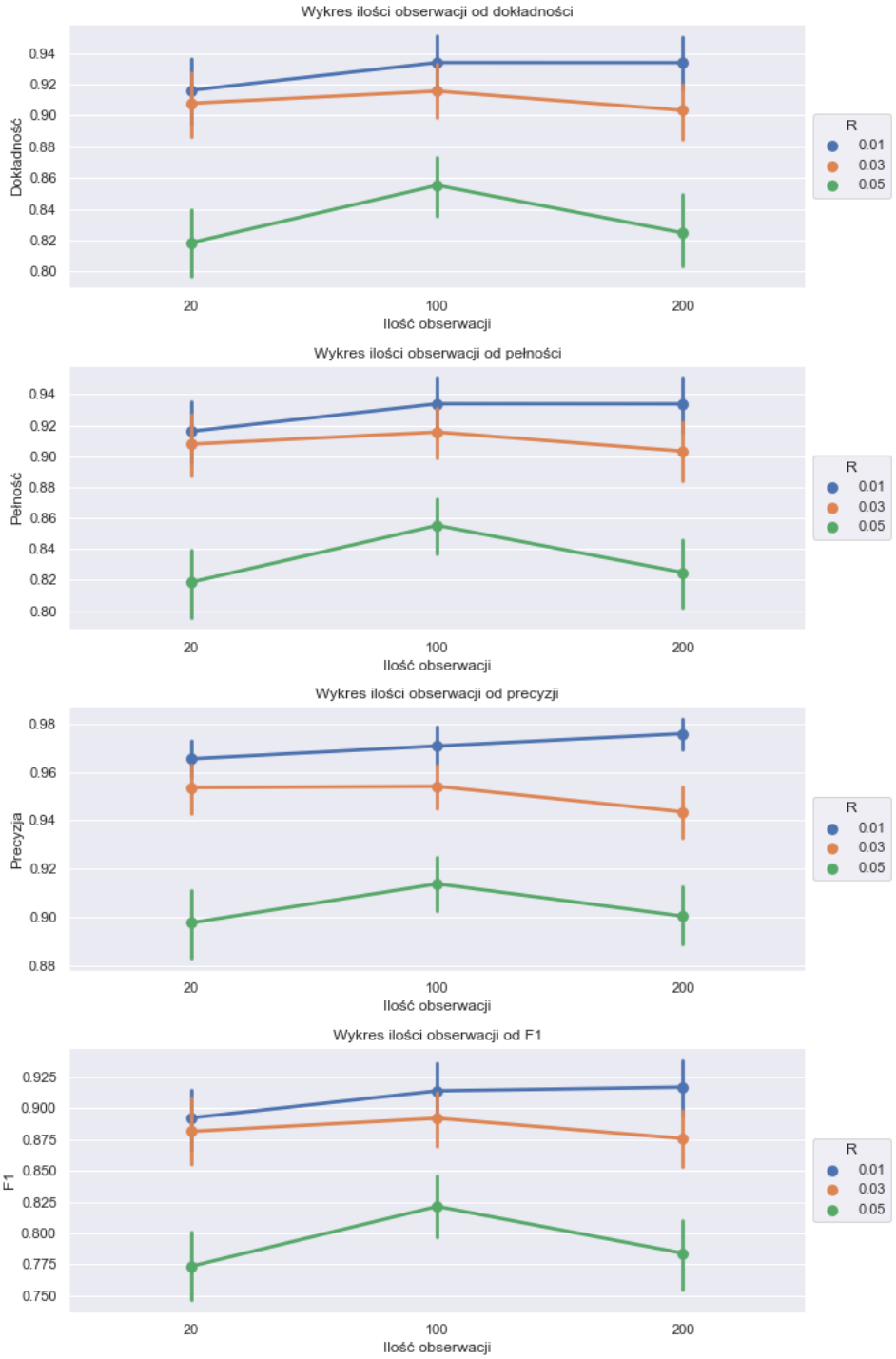
Przepływ danych składa się między innymi z procesowania plików dźwiękowych, wyliczenia współczynników MFCC, zdefiniowania i wytrenowania modelu *IwrGaussianHMMModel* oraz jego ewaluacji. Na wartość wyników wpływa wiele parametrów, którymi można manipulować na poszczególnych etapach. Chcąc wybrać najbardziej optymalne parametry będziemy sprawdzać ich różne kombinacje dla zmiennych:

- $N_{\text{train}} = 20, 100, 200$ - ilość obserwacji z jednej klasy w zbiorze treningowym. Wtedy wielkość zbioru treningowego wynosi $N_{\text{train}} \cdot N_c$.
- $R = 0.01\text{ms}, 0.03\text{ms}, 0.05\text{ms}$ - wielkość skoku okna czasowego.
- $S = 0.025\text{ms}, 0.04\text{ms}$ - szerokość okna czasowego.
- n_iter = 15, 30, 50 - ilość iteracji w algorytmie Bauma–Welcha.
- *n_components* = 4, 6, 12 - ilość ukrytych stanów.

Każdy model będzie poddawany zbalansowanej 5 foldowej walidacji krzyżowej implementowanej za pomocą funkcji *StratifiedKFold* z pakietu *scikit-learn*. Wykorzystujemy ten rodzaj walidacji, ponieważ zależy nam na zbalansowaniu każdego z 5 zbiorów walidacyjnych. W przypadku gdy zbiór walidacyjny nie posiadałby obserwacji z dowolnej klasy, model nie mógłby zwrócić wiarygodnych wyników.

3.3.1 Wybór N_{train}, R i S

Na początku możemy sprawdzić jak zmiana liczby obserwacji w zbiorze treningowym może wpłynąć na wyniki walidacji krzyżowej. Czyli dla różnych wartości N_{train} będziemy trenować modele z różnymi parametrami, a ich wyniki uśredniać.



Rysunek 3.1: Wykres punktowy średniej dokładności, pełności, precyzji i miary F_1 z walidacji krzyżowej w zależności od ilości obserwacji N_{train} z podziałem na wartość S. Źródło: Opracowanie własne

Na wykresie punktowym (3.1) widzimy wyniki dla N_{train}=20, 100, 200. Wprowadzamy

dodatkowy podział wyników w zależności od skoku okna R. Pierwszą istotną obserwacją jest, że dla R=0.01ms średnie wyniki wszystkich modeli są znacząco większe od przypadków R=0.03ms, 0.05ms. Im mniejsza jego wartość, tym więcej wektorów cech spektralnych $o_t, t = 1, \ldots, T$ otrzymujemy z pojedynczej obserwacji, a co za tym idzie, dane dotyczące sygnału są bardziej szczegółowe. Warto zauważyć, że dla N_{train}=20 uśrednione wyniki, włącznie z uwzględnieniem podziału na R są mniejsze bądź równe wynikom dla N_{train}=100. Oznacza to, że 20 obserwacji z każdej klasy w zbiorze treningowym jest niewystarczającą próbką do wyuczenia modelu HMM – jest on niedotrenowany. Kwestia ma się inaczej dla N_{train}=200. Wyniki są wtedy często gorsze od tych z N_{train}=100. Wyjątek stanowi przypadek z R=0.01ms, gdzie wyniki bywają minimalnie, ale nieznacząco wyższe. Dobrym wnioskiem jest to, że model nie wykazuje tendencji do przetrenowania się. Jednak, ponieważ wraz ze wzrostem N_{train} znacząco wzrasta czas trenowania, przy niewielkim wzroście dokładności, stąd w dalszej części pracy będziemy zakładać, że N_{train}=100.

Możemy w bardziej szczegółowy sposób zwrócić uwagę jak zmienia się wartość miar wyników w zależności od parametrów R i S. W tabeli (3.2) mamy średnie wyniki walidacji krzyżowej dla najlepszych modeli uwzględniając podział na R i S. Ponownie uzyskujemy wniosek, że zgodnie z literaturą najbardziej optymalnym wyborem jest R=0.01ms. Wraz ze wzrostem tego parametru, wyniki pogorszają się.

R	**S**	**Dokładność**	**Pełność**	**Precyzja**	**Miara F_1**
0.01	0.025	1.0	1.0	1.0	1.0
	0.04	1.0	1.0	1.0	1.0
0.03	0.025	0.984	0.984	0.988	0.983
	0.04	0.98	0.98	0.99	0.973
0.05	0.025	0.909	0.909	0.936	0.895
	0.04	0.954	0.954	0.971	0.944

Tabela 3.2: Wartości średniej dokładności, pełności, precyzji i miary F_1 z walidacji krzyżowej dla modeli trenowanych na $N_{\text{train}} = 100$ obserwacjach w zależności od R i S. Źródło: Opracowanie własne

Patrząc na wyniki z tabeli (3.2) nie jesteśmy w stanie rozstrzygnąć, czy przy R=0.01ms lepiej będzie dobrać parametr S=0.025ms, czy S=0.04ms. Patrząc po innych przypadkach i uwzględniając wyniki z tabeli (3.3), gdzie wyliczone zostały odchylenia standardowe dla wyników walidacji krzyżowej, możemy uznać, że im mniejsze S, tym bardziej stabilne wyniki osiągamy. Z tego powodu ponownie najlepszym wyborem okazuje się książkowe S=0.025ms.

Podsumowując, w dalszych obliczeniach będziemy wykorzystywać modele trenowane na $N_{\text{train}} = 100$ obserwacjach, w których sygnały są poddawane filtrom o długości okna S=0.025ms i skoku okna czasowego R=0.01ms.

3.3.2 Wybór *n_components* i n_iter

Chcąc zaimplementować najbardziej optymalny model bazując na symulacji pozostaje wybrać odpowiednie parametry *n_components* i n_iter. Uwzględniając na ten moment tylko modele trenowane na $N_{\text{train}} = 100$ obserwacjach.

Z tabeli (3.4) widzimy, że najlepsze wyniki osiąga wybór *n_components*=6. Ciężko jednak wybrać odpowiadający mu parametr n_iter. Rzućmy okiem na wykresy pudełkowe.

R	S	Dokładność	Pełność	Precyzja	Miara F_1
0.01	0.025	0.0	0.0	0.0	0.0
	0.04	0.0	0.0	0.0	0.0
0.03	0.025	0.02	0.02	0.012	0.021
	0.04	0.002	0.002	0.007	0.005
0.05	0.025	0.036	0.036	0.02	0.049
	0.04	0.048	0.048	0.013	0.056

Tabela 3.3: Wartości odchylenia std. dokładności, pełności, precyzji i miary F_1 z walidacji krzyżowej dla modeli trenowanych na $N_{\text{train}} = 100$ obserwacjach w zależności od R i S.
Źródło: Opracowanie własne

n_components	*n_iter*	Dokładność	Pełność	Precyzja	Miara F_1
4	15	0.8	0.8	0.926	0.747
	30	0.78	0.78	0.91	0.717
	50	0.8	0.8	0.92	0.743
6	15	1.0	1.0	1.0	1.0
	30	1.0	1.0	1.0	1.0
	50	1.0	1.0	1.0	1.0
12	15	1.0	1.0	1.0	1.0
	30	0.999	0.999	0.999	0.999
	50	0.954	0.954	0.975	0.941

Tabela 3.4: Wartości średnich dokładności, pełności, precyzji i miary F_1 z walidacji krzyżowej dla najlepszych modeli trenowanych na $N_{\text{train}} = 100$ obserwacjach, $R = 0.01$ms, $S = 0.025$ms w zależności od *n_components* i *n_iter*.
Źródło: Opracowanie własne

Na rysunku (3.2) możemy zauważyć, że średnie wyniki w znaczący sposób są większe dla *n_components*=6. Co prawda, dla *n_components*=12 maksymalne wyniki również potrafią osiągnąć 100%. Jednak modele z tej klasy charakteryzują się dużo większym zróżnicowaniem wyników, niż w przypadku *n_components*=6. Zwracając uwagę na wykresy pudełkowe dla *n_components*=6. Spróbujemy wybrać optymalną wartość n_iter. Najbardziej stabilne są wyniki dla n_iter=30. Zarówno dla n_iter=15 jak i n_iter=50 średnie wyniki są gorsze od n_iter=30. Oznacza to, że dla n_iter=15 model jest niedotrenowany, natomiast dla n_iter=50 model się przetrenowuje. Optymalny będzie wybór n_iter po środku tych właśnie wartości. W naszym przypadku n_iter=30 jest najlepsze.

Podsumowując, model, jaki będziemy wykorzystywać w dalszej części pracy, będzie trenowany na $N_{\text{train}} = 100$ obserwacjach, w których sygnały są poddawane filtrom o długości okna S=0.025ms i skoku okna czasowego R=0.01ms. Ukryty łańcuch Markowa będzie miał *n_components*=6 ukrytych stanów oraz n_iter=30 ilości iteracji algorytmu Bauma–Welcha.

3.4 Wyniki w obecności zakłóceń

3.4.1 Generowanie zakłóceń

W celu wygenerowania próby losowej szumu skorzystamy z pakietu *colorednoise*, bazującej na metodzie generowania szumu z [18]. Korzystając z metody powerlaw_psd_gaussian

i ustawiając parametr *beta*=β, który odpowiada za współczynnik definiujący kolor szumu, możemy wygenerować określoną liczbę próbek zadanego szumu gaussowskiego. Następnie korzystając z własnej metody *preprocess.noising*, możemy wprowadzić do zadanego sygnału dźwiękowego szum gaussowski, którego widmowa gęstość mocy $\phi(f) \propto (1/f)^{\beta}$. Ponadto, wybierając odpowiedni parametr SNR_{dB}, możemy ustawić jak głośny będzie szum względem sygnału .

3.4.2 Ewaluacja

Wybraliśmy optymalny model z wykorzystaniem zbalansowanej walidacji krzyżowej. Interesuje nas jak jego wyniki będą się zachowywały na zbiorach testowych poddanych zakłóceniom. W symulacjach będziemy uwzględniać parametry:

- SNR_{dB} = -2 dB, -1dB, . . . , 8dB, 9dB - współczynniki SNR (signal-noise ratio) w skali decybelowej.
- β = -2, -1, 0, 1, 2 - współczynniki definiujące kolor szumu, których widmowa gęstość mocy $\phi(f) \propto (1/f)^{\beta}$.

W tabeli (3.5) możemy zaobserwować wyniki dla zbioru testowego, którego obserwacje były zaszumione z SNR_{dB} = 9dB. Jest to najmniejszy z rozważanych poziomów zakłóceń. Naturalnym zachowaniem jest obserwowany wzrost wyników wraz ze wzrostem parametru β. W momencie, gdy szum ma niską moc w stosunku do naszych obserwacji, algorytmy radzą sobie bezproblemowo.

β	**Dokładność**	**Pełność**	**Precyzja**	**Miara** F_1
-2	0.889	0.889	0.944	0.852
-1	0.889	0.889	0.944	0.852
0	0.889	0.889	0.944	0.852
1	1.0	1.0	1.0	1.0
2	1.0	1.0	1.0	1.0

Tabela 3.5: Wartości dokładności, pełności, precyzji i miary F_1 ze zbioru testowego dla współczynnika SNR_{dB}=9dB w zależności od parametru β.
Źródło: Opracowanie własne

Tabela (3.6) pokazuje z drugiej strony wyniki dla zbioru testowego, którego obserwacje były zaszumione z SNR_{dB}=-2dB. Czyli jest to przypadek najbardziej wymagający, w którym moc szumu jest znacząco większa od mocy sygnału. Wyniki w tym przypadku są o wiele mniej zadowalające, jednak wciąż lepsze od przypadku kompletnie losowych wyników (w którym model uznałby wszystkie obserwacje za należące do jednej i tej samej klasy). Co ciekawe, dla tego poziomu zaszumienia model charakteryzuje się stosunkowo dużą precyzją, co jednak jest mało przydatne w przypadku, gdy pełność jest niska. Ciekawa jest również utrata reguły mówiącej, że wraz ze wzrostem β rośnie dokładność. Wyniki z SNR_{dB}=-2dB są najlepsze przy jednoczesnym β=1.

Pełny zakres wyników dokładności znajduje się na rysunku (3.3). Widać, że jej wielkość jest mocno pozytywnie skorelowana z wartością SNR_{dB}. Trochę mniejsza, ale wciąż pozytywna korelacja występuje między dokładnością a β. Wynika to z tego, że model lepiej sobie radzi z rozpoznawaniem słów zaszumionych różowym szumem, niż szumem brownowskim. Łatwo odczytać, że model w sposób akceptowalny zachowuje się z szumami

β	**Dokładność**	**Pełność**	**Precyzja**	**Miara F_1**
-2	0.183	0.183	0.632	0.106
-1	0.211	0.211	0.661	0.141
0	0.2	0.2	0.654	0.129
1	0.472	0.472	0.801	0.363
2	0.344	0.344	0.77	0.32

Tabela 3.6: Wartości dokładności, pełności, precyzji i miary F_1 ze zbioru testowego dla współczynnika SNR_{dB}=-2dB w zależności od parametru β.
Źródło: Opracowanie własne

o SNR_{dB} nie mniejszym niż 3-4dB. Wtedy wyniki modelu cały czas przekraczają poziom 50%. Co ciekawe, najniższy wynik pojawił się dla białego szumu z SNR_{dB}=-1dB. Uzyskano wtedy 17%, co wciąż jest lepsze od całkowicie losowych predykcji.

Na rysunku (3.4) widzimy wykresy punktowe w zależności od SNR_{dB} oraz β. Widać, że przy większej obecności zakłóceń, czyli małych SNR_{dB}, model radzi sobie najgorzej z białym szumem, a najlepiej z różowym szumem. Dla wysokich SNR_{dB} wyniki nie są jednoznaczne i ciężko wskazać zarówno najlepszy, jak i najgorszy rodzaj szumu, z którym zmierzyłby się model.

Wiemy już jak nasz model zachowuje się w obecności zakłóceń. Będziemy go nazywać *modelem bazowym*. Chcielibyśmy jednak spróbować poprawić obecne wyniki.

3.5 Optymalne parametry zaszumienia

Chcielibyśmy zaproponować metodę do ogólnego poprawienia dokładności modelu wśród rozważanych dotychczas zakłóceń. Metoda będzie opierała się na założeniu, że istnieje taki rodzaj szumu, że model wytrenowany na obserwacjach, które zostały nim zaszumione, będzie w ogólności zwracał lepszą dokładność na zbiorze testowym, w którym będą obecne zakłócenia dowolnego rodzaju.

Nie wiemy jaki to może być szum. Będziemy rozpatrywać zakłócenia sygnału o parametrach:

- $\beta' =$ -2, -1, 0, 1, 2 - współczynniki definiujące kolor szumu, których widmowa gęstość mocy $\phi(f) \propto (1/f)^{\beta}$. Zostaną użyte do zaszumienia obserwacji wykorzystywanych do trenowania modelu.

- $\overline{SNR}_{dB} =$ 5dB, 6dB, 7dB, 8dB, 9dB - współczynniki SNR (signal-noise ratio) w skali decybelowej, które zostaną użyte do zaszumienia obserwacji wykorzystywanych do trenowania modelu.

Interesuje nas znalezienie takiej kombinacji parametrów $(\beta', \overline{SNR}_{dB})$, przy których wyniki dokładności na zbiorach testowych dla rozważanych wartości β oraz SNR_{dB} będą najczęściej lepsze od wyników modelu bazowego.

Niech $ACC_{\beta'}^{\overline{SNR}_{dB}}(\beta, SNR_{dB})$ oznacza wynik dokładności na zbiorze testowym obserwacji zaszumionych szumami o parametrach (β, SNR_{dB}) w zależności od użytych w trakcie trenowania zaszumionych obserwacji z parametrami $(\beta', \overline{SNR}_{dB})$.

Szukając optymalnych parametrów zdefiniujmy funkcję nagrody, czyli taką, której maksimum od szukanych parametrów będzie osiągane dla najlepszej kombinacji $(\beta', \overline{SNR}_{dB})$.

Interesuje nas, żeby dokładność była jak najczęściej lepsza od dokładności modelu bazowego. Stąd, zdefiniujmy funkcje nagrody Z jako:

$$Z(\beta', \overline{SNR}_{dB}) = \#\{(\beta, SNR_{dB}) : ACC_{\beta'}^{\overline{SNR}_{dB}}(\beta, SNR_{dB}) > ACC(\beta, SNR_{dB})\}, \quad (3.6)$$

gdzie:

$ACC(\beta, SNR_{dB})$ – dokładność modelu bazowego na zbiorze testowym z obserwacjami zakłóconymi szumem o parametrach (β, SNR_{dB}).

$\beta, \beta' \in \{-2, -1, 0, 1, 2\}$.

$SNR_{dB} \in$ {-2 dB, -1dB, ..., 8dB, 9dB}.

$\overline{SNR}_{dB} \in$ {5dB, 6dB, 7dB, 8dB, 9dB}.

Po przeprowadzeniu symulacji okazało się, że najczęściej model, który uzyskiwał na zbiorze testowym lepsze wyniki od modelu bazowego był trenowany na obserwacjach zakłóconych szumem o parametrach $\overline{SNR}_{dB} = 9$dB i $\beta' = -2$. Wyniki możemy zaobserwować w tabeli (3.7).

β	**-2**	**-1**	**0**	**1**	**2**
$SNR_{\mathbf{dB}}$					
-2	0.356	0.161	0.294	0.017	-0.072
-1	0.445	0.339	0.539	0.028	0.016
0	0.366	0.461	0.578	0.122	0.083
1	0.494	0.567	0.7	0.178	0.077
2	0.4	0.556	0.628	0.1	0.056
3	0.333	0.35	0.4	0.028	-0.011
4	0.333	0.333	0.228	-0.039	-0.039
5	0.228	0.206	0.117	-0.083	-0.073
6	0.111	0.111	0.111	-0.133	-0.183
7	0.111	0.111	0.111	-0.156	-0.261
8	0.111	0.111	0.111	-0.188	-0.194
9	0.111	0.111	0.111	-0.194	-0.078

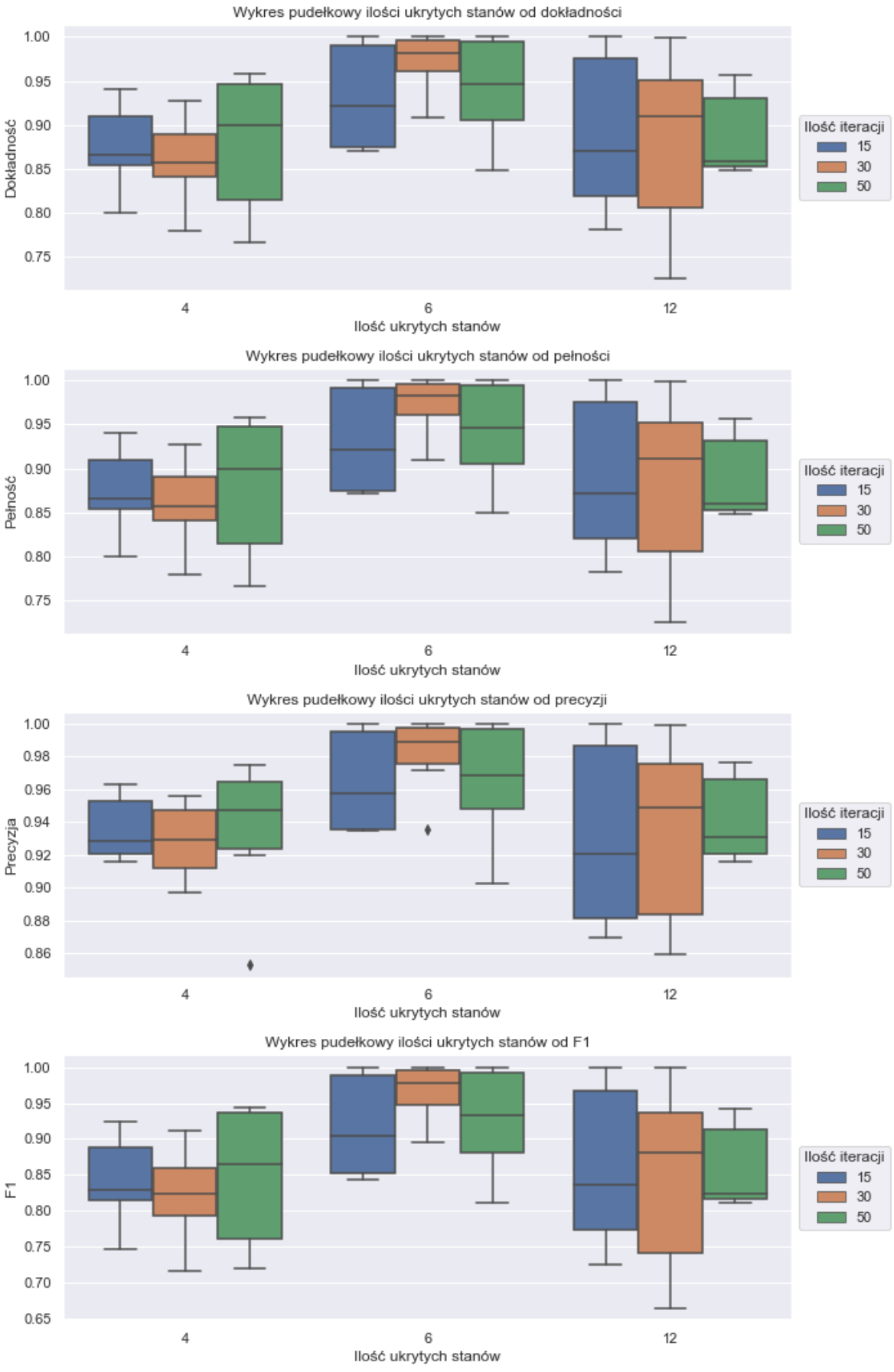
Tabela 3.7: Porównanie wyników dokładności modelu bazowego do modelu wytrenowanego na zaszumionych danych z parametrami $\overline{SNR}_{dB}$=9dB i $\beta' = -2$. Wartości tabeli przedstawiają zwroty, tzn. $ACC_{\beta'}^{\overline{SNR}_{dB}}(\beta, SNR_{dB}) - ACC(\beta, SNR_{dB})$.
Źródło: Opracowanie własne

Na zielono zostały zaznaczone poprawy w dokładności, a na czerwono pogorszenia. Jak możemy zaobserwować wyniki poprawiały się w niektórych przypadkach nawet o 70%. Największe pogorszenie wynosiło natomiast 19.4%. Znacząca poprawa w większości przypadków sprawia, że model trenowany na zaszumionych danych staje się bardziej stabilny przy predykcjach obserwacji w towarzystwie innych zakłóceń. Ciekawą obserwacją jest, że pogorszenie wyników występuje przy parametrach $\beta = 1$ i $\beta = 2$ i wysokich wartościach SNR_{dB}. Może to wynikać z doboru niskiej wartości dla β', przez co model nauczony na intensywnych zakłóceniach radzi sobie gorzej w obecności „delikatniejszych" szumów.

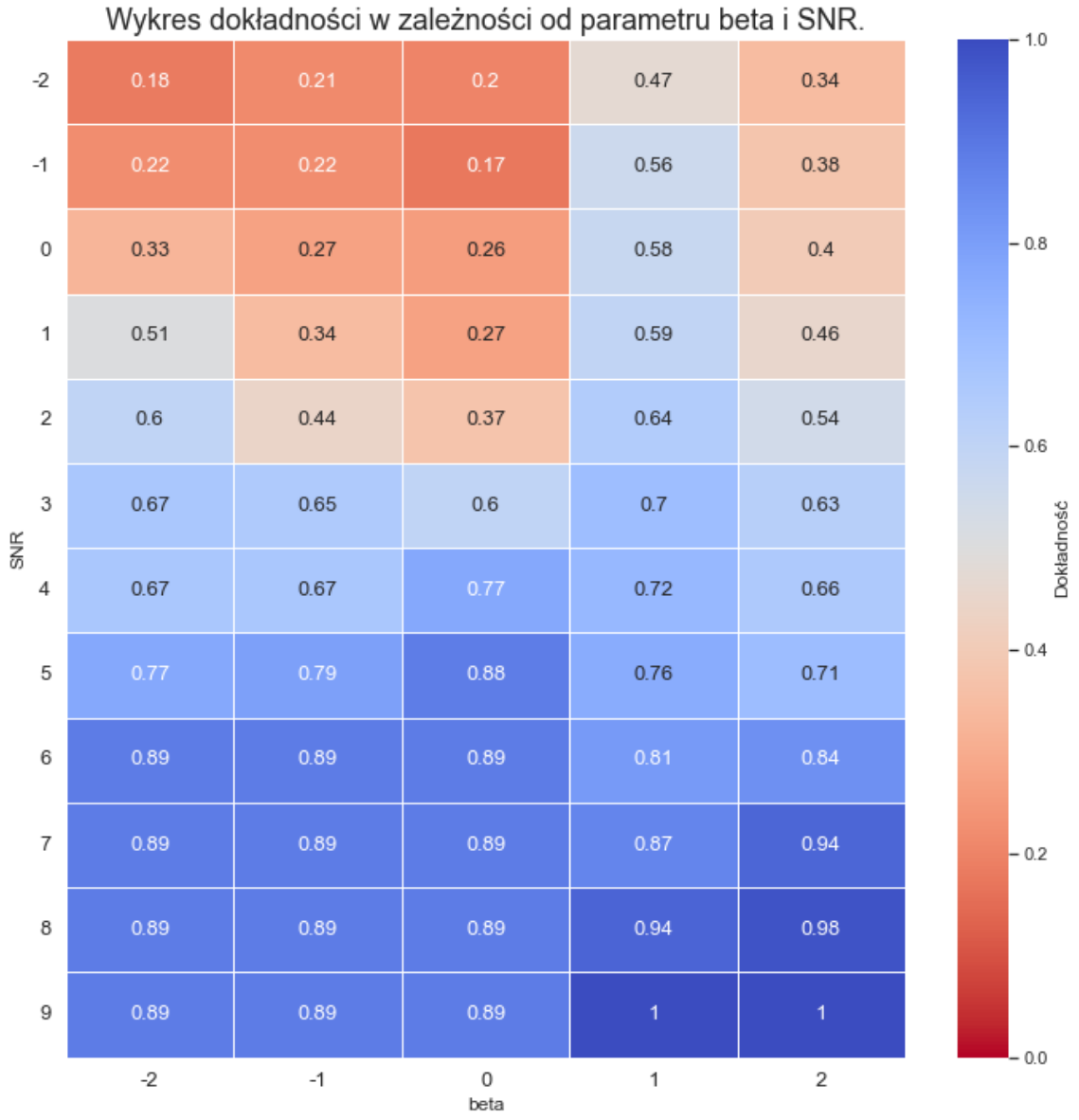
Możemy dokładniej przyjrzeć się wynikom wyznaczonego modelu. Na rysunku (3.5) widzimy, że nasz model znacząco poprawił dokładność dla niskich wartości β i wysokich SNR_{dB}. Przypomnijmy, że z rysunku (3.3) akceptowalna stosowalność modelu bazowego osiągała co najmniej SNR_{dB}=3db. Stosowalność naszego modelu poszerzyła zakres do co najmniej SNR_{dB}=0db, a w niektórych przypadkach nawet do co najmniej SNR_{dB}=-1db. Poprawa wynikająca z użycia w fazie treningu zaszumionych obserwacji jest znacząca.

Na rysunku (3.6) widzimy wykresy punktowe w zależności od SNR_{dB} oraz β. Widać, że przy większej obecności zakłóceń, czyli małych SNR_{dB}, model radzi sobie najlepiej z białym szumem, w przeciwieństwie do modelu bazowego. Wyniki natomiast są najgorsze dla szumu brownowskiego. Dla wysokich SNR_{dB} przy $\beta = -2, -1, 0$ wyniki osiągają 100%. Problemy wciąż utrzymują się z szumem brownowskim i różowym szumem.

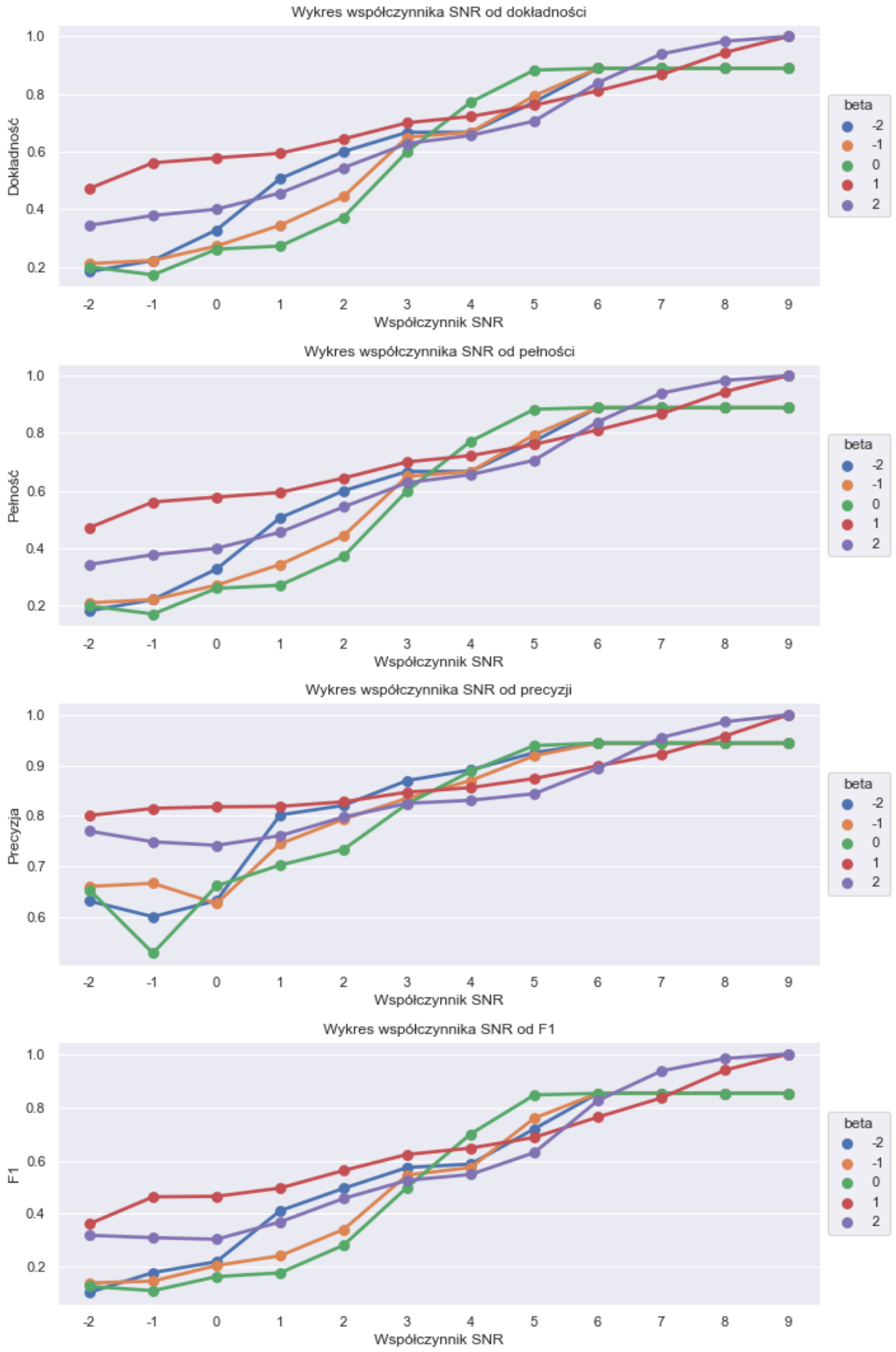
Jeśli wiemy, że przy naszym problemie będziemy mieli do czynienia z zakłóconymi sygnałami dźwiękowymi, to już na etapie trenowania modelu powinniśmy go przygotować do pracy z podobnymi obserwacjami. Dlatego też zaszumienie danych w sposób sztuczny może w wielu przypadkach poprawić wyniki predykcji na niewidzianych obserwacjach.



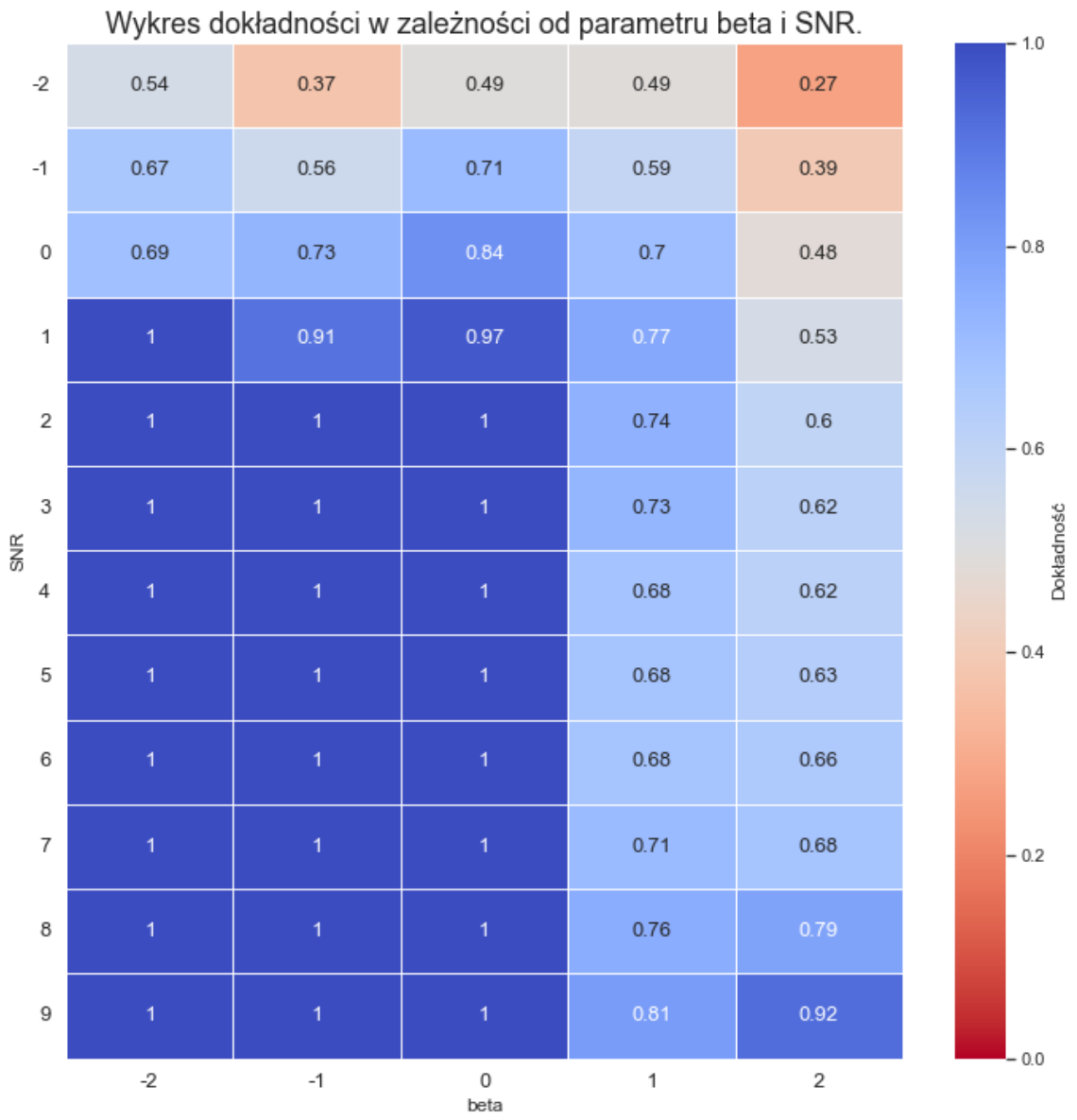
Rysunek 3.2: Wykres pudełkowy średniej dokładności, pełności, precyzji i miary F_1 z walidacji krzyżowej dla modeli trenowanych na $N_{\text{train}} = 100$ obserwacjach w zależności od *n_components* z podziałem na n_iter.
Źródło: Opracowanie własne



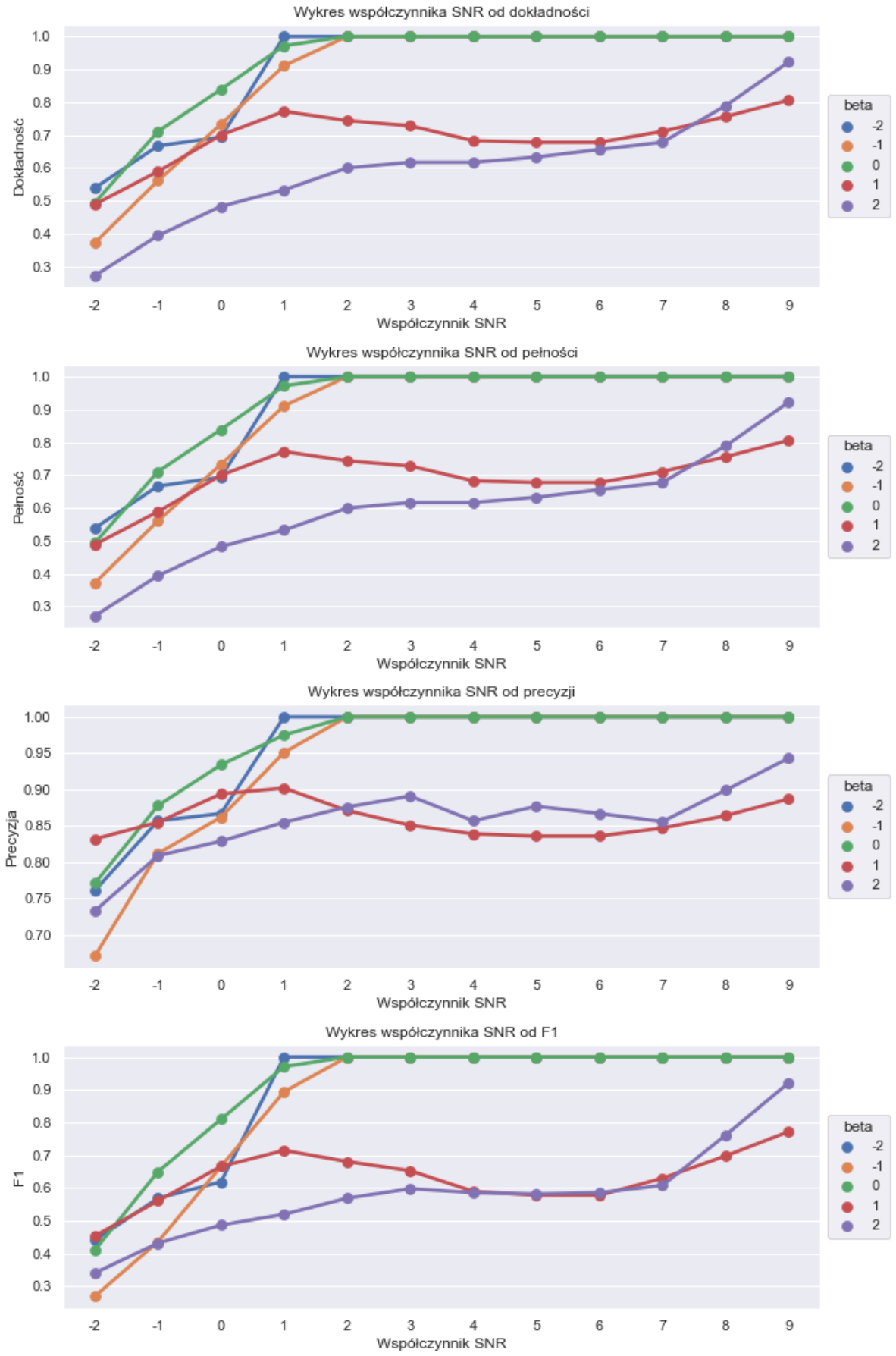
Rysunek 3.3: Wartości dokładności ze zbioru testowego w zależności od parametru β i współczynnika SNR_{dB}.
Źródło: Opracowanie własne



Rysunek 3.4: Wykres punktowy dokładności, pełności, precyzji i miary F_1 ze zbioru testowego w zależności od współczynnika SNR_{dB} z podziałem na parametr β.
Źródło: Opracowanie własne



Rysunek 3.5: Wartości dokładności ze zbioru testowego w zależności od parametru β i współczynnika SNR_{dB} dla modelu wytrenowanego na zaszumionych danych z parametrami $\overline{SNR}_{dB}$=9dB i $\beta' = -2$.
Źródło: Opracowanie własne



Rysunek 3.6: Wykres punktowy dokładności, pełności, precyzji i miary F_1 ze zbioru testowego w zależności od współczynnika SNR_{dB} z podziałem na parametr β dla modelu wytrenowanego na zaszumionych danych z parametrami $\overline{SNR}_{dB}$=9dB i $\beta' = -2$.
Źródło: Opracowanie własne

Rozdział 4

Podsumowanie

W rozdziale 2 zaczęliśmy od szczegółowego opisania procesowania sygnałów dźwiękowych za pomocą cech spektralnych, w szczególności współczynników MFCC. Wprowadziliśmy niezbędna teorię dotyczącą ukrytych łańcuchów Markowa i tego, w jaki za pomocą procedury forward–backward mogą się uczyć na danych. Zdefiniowaliśmy rozpatrywane w pracy szumy losowe i w jaki sposób, wykorzystując współczynnik SNR, jesteśmy w stanie kontrolować ich obecność w naszym sygnale docelowym.

W rozdziale 3 przedstawiliśmy proces przygotowania środowiska, w którym przeprowadzaliśmy symulacje. Skupiliśmy się na opisaniu najważniejszych narzędzi jak gTTS, ale również ograniczeń, do których się zastosowaliśmy, np. wybór parametrów użytych do znalezienia optymalnego modelu. Zdefiniowaliśmy sposób wyznaczania wyników. Wybraliśmy model bazowy z odpowiednimi parametrami i sprawdziliśmy jak będzie się zachowywał w obecności zaszumionych danych.

Zaproponowaliśmy metodę do ogólnego poprawienia rezultatów modelu na zaszumionych danych. HMM został wytrenowany na obserwacjach w obecności konkretnego rodzaju zakłóceń, tj. szumu gaussowskiego o parametrze β. Celem było poprawienie wyników modelu na niewidzianych sygnałach z szumem gaussowskim o dowolnym parametrze β i dowolnym parametrze SNR_{dB}. Pokazaliśmy, że dla zbioru treningowego zaszumionego zakłóceniami o parametrach $\beta' = -2$ i $\overline{SNR}_{dB} = 9\text{dB}$ model na nim nauczony osiągał wyższe wyniki dokładności od modelu bazowego, najczęściej z całej siatki parametrów $(\beta' \ , \ \overline{SNR}_{dB})$ i zrobił to w 46 na 60 rozpatrywanych przypadkach. Poprawa wyniku potrafiła osiągać nawet 70% ACC.

W pracy wybraliśmy do treningu modelu taki szum, który sprawiał, że najczęściej poprawiane były wyniki dokładności. Wybór analizy akurat tej metryki był w pełni subiektywny i w zależności od potrzeb związanych z problemem można wykorzystać równie dobrze inne metryki.

Dodatek

Całość kodu włącznie z symulacjami, wynikami i wizualizacjami opartymi na nich znajduje się na publicznym repozytorium w serwisie *Github* pod adresem `https://github.com/sokoly35/BSc_hmm_speech_recognition`.

Bibliografia

[1] Christopher M. Bishop. *Pattern Recognition and Machine Learning*. Springer, 2006.

[2] Giuseppe Ciaburro i Prateek Joshi. *Python Machine Learning Cookbook - Second Edition*. Packt Publishing, 2019.

[3] *Federal Standard 1037C*. Spraw. tech. Institute for Telecommunication Sciences. Institute for Telecommunication Sciences, National Telecommunications i Information Administration (ITS-NTIA), 2018.

[4] M. Gales i S. Young. *The Application of Hidden Markov Models in Speech Recognition. Foundations and Trends in Signal Processing*. 2007.

[5] M. J. F. Gales i S. J. Young. "An Improved Approach to the Hidden Markov Model Decomposition of Speech and Noise". W: *ICASSPE* (1992), s. 233–236.

[6] Aurélien Géron. *Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow. Concepts, Tools, and Techniques to Build Intelligent Systems. 2nd Edition*. O'Reilly Media, 2019.

[7] Google. *Google Text-to-Speech*. `https://gtts.readthedocs.io/en/latest/index.html`. Dostęp: 02.01.2022.

[8] *HMM Speech Recognition*. `https://code.google.com/archive/p/hmm-speech-recognition/downloads`. Dostęp: 02.01.2022.

[9] X. Huang, A. Acero i H. Hon. *Spoken Language Processing: A guide to theory, algorithm, and system development*. Prentice Hall, 2001, s. 314–316.

[10] D. H. Johnson. "Signal-to-noise ratio". W: *Scholarpedia* 1.12 (2006). revision #126771, s. 2088. DOI: `10.4249/scholarpedia.2088`.

[11] B. H. Juang i L. R. Rabiner. "Hidden Markov Models for Speech Recognition". W: *Technometrics* 33.3 (1991), s. 251–272.

[12] Daniel Jurafsky i James H. Martin. *Speech and Language Processing*. 1999.

[13] Veton Këpuska i Gamal Bohouta. "Comparing Speech Recognition Systems (Microsoft API, Google API And CMU Sphinx)". W: *Int. Journal of Engineering Research and Application2017* 7 (2017), s. 20–24.

[14] Byeonggeun Kim i in. "Broadcasted Residual Learning for Efficient Keyword Spotting". W: *ArXiv e-prints* (2021). arXiv: `2106.04140v2 [cs.SD]`.

[15] C. Kohlschein. "An introduction to Hidden Markov Models". W: (2007).

[16] B. H. Juang L. R. Rabiner. "An introduction to hidden Markov models". W: *IEEE ASSP Mag.* vol. 3, no. 1 (1986), s. 4–16.

[17] Petre Stoica i Randolph L. Moses. *Spectral Analysis of Signals*. Pearson Prentice Hall, 2005.

[18] J. Timmer i M. König. “On Generating Power Law Noise”. W: *AA* 300 (1995), s. 707–710.

[19] A. P. Varga i R. K. Moore. “Hidden Markov Model Decomposition of Speech and Noise”. W: *ICASSP90* (1990), s. 845–848.

[20] P. Warden. “Speech Commands: A Dataset for Limited-Vocabulary Speech Recognition”. W: *ArXiv e-prints* (kw. 2018). arXiv: 1804.03209 [cs.CL]. URL: https://arxiv.org/abs/1804.03209.

[21] Joseph S. Wisniewski. *Colors of noise pseudo FAQ, version 1.3*. https://web.archive.org/web/20110430151608/https://www.ptpart.co.uk/colors-of-noise. Dostęp: 03.01.2022.