**Rozpoznawanie człowieka metodami biometrii**

**Sprawozdanie 2**

Michalik Piotr

Spis treści

[Wstęp 3](#_Toc514442439)

[Dynamic Time Warping 3](#_Toc514442440)

[Weryfikacja 5](#_Toc514442441)

[ScreenShoty 6](#_Toc514442442)

[Opis ScreenShotów 9](#_Toc514442447)

[Wyniki 10](#_Toc514442448)

[Wnioski 10](#_Toc514442449)

[Źródła 10](#_Toc514442450)

# Wstęp

W poniższym dokumencie przedstawię środki użyte do realizacji drugiego projektu na przedmiot ‘Wykrywanie człowieka metodami biometrii’. Po kolei zostaną opisane i przedstawione metody zaimplementowane w tym projekcie. Później nastąpi części właściwa, to znaczy ocena i opis zaimplementowanych przeze mnie algorytmów. Na samym końcu zamieszczone zostanie krótkie podsumowanie etapu. Dla każdej opisanej metody zostaną zamieszczone screenshoty prezentujące działanie poszczególnej funkcjonalności.

W tym sprawozdaniu opisane, zaprezentowane i ocenione zostaną dwa algorytmy: algorytm DTW oraz algorytm służący do weryfikacji. Celem projektu jest weryfikacja osób na podstawie nagrań ich głosów.

# Dynamic Time Warping

## Teoria

Porównanie dwóch ścieżek dźwiękowych jest zadaniem trudnym, ze względu na wiele czynników. Przede wszystkim dźwięk jest reprezentowany jako pewna funkcja zależna od czasu. Co za tym idzie to samo słowo, wypowiedziane przez tą samą osobę może być zawarte w dwóch różnych wycinkach czasowych. (Potocznie mówiąc – te same słowa wypowiadamy szybciej, wolnej lub z różną częstotliwością). Po drugie w przeciwieństwie do innych cech biometrycznych (odciski palców, tęczówki) bardzo ciężko jest reprezentować mowę jakimkolwiek wektorem cech, przez to konkretne słowa trzeba porównywać w całości do siebie.

Dlatego użytą w tym projekcie metodą jest DTW. Dynamic Time Warping pozwala porównywać sekwencje sygnałów o różnych długościach. Zakładając że posiadamy dwa dźwięki (A oraz B) pod postacią dwóch tablic wartości rzeczywistych (o długościach kolejno *n* i *m*), gdzie kolejny index tabeli odpowiada kolejnemu momentowi czasowemu, algorytm prezentuje się następująco:

1. Tworzymy macierz lokalną Ml o wymiarach n x m. Następnie dla każdej pozycji w macierzy uzupełniamy koszt według wzoru:

To znaczy że dla każdej pary punktów z obu ścieżek zapamiętujemy różnicę miedzy nimi.

1. Tworzymy macierz globalną Mg również o wymiarach n x m. Następnie dla każdej pozycji w macierzy uzupełniamy koszt globalny według wzorów:

Jeżeli *x=1* i *y!=1*:

Jeżeli *x!=1* i *y=1*:

W pozostałych przypadkach:

1. Szukamy najtańszej ścieżki w macierzy kosztów globalnych. Przechodzimy od punktu *P(n,m)* do punktu *P(0,0)* w taki sposób, że możemy wykonać ruch tylko w 3 kierunkach: po przekątnej w stronę współrzędnej (0,0), w osi X w kierunku wartości 0 lub w osi Y w kierunku wartości 0. Zawsze wykonujemy ruch do pola o najniższym koszcie zapisanym w macierzy Mg.
2. Sumujemy całkowity koszty otrzymanej ścieżki, uzyskując koszt porównania dźwięków.

## Implementacja

Cała aplikacja została zaimplementowana w języku C# w technologii WPF. Głowna część algorytmu została zaimplementowana zgodnie z opisem teoretycznym. W tej część omówię brakujący opis przygotowania danych:

Pre-processing danych dźwiękowych jest sprawą problematyczną, ponieważ przy porównywaniu dźwięków metodą DTW ciężko ustalić które fragmenty dźwięku będą miały kluczowy wpływ na porównanie, a które są tylko szumem, zakłóceniem lub są po prostu nieistotne. Dlatego w tym projekcie podejmuję tylko 3 proste działania w ramach obróbki wstępnej dźwięku.

Po pierwsze zapis udostępnionych nagrań zawierał dźwięki o bardzo małych wartościach (tysięczne części typu float), przez co w przypadku uśrednień, porównywania wartości czy zaokrągleń takie małe wartości tracą sens. Dlatego na wstępie każda wartość dźwięku jest mnożona przez 100, tak aby sygnał został wzmocniony i wartości stały się co najmniej wartościami dziesiętnymi.

Po drugie, każde nagranie zawiera parę sekund ciszy na końcu i na początku nagrania. Usuwamy tą cisze. Ustalamy wartość progową, która stanowi 2% maksymalnej wartości (różnica pomiędzy najwyższym i najniższym dźwiękiem) w danej ścieżce. Następnie od początku oraz od końca szukamy momentów czasowych w których nagranie po raz pierwszy przekracza tą wartość. W ten sposób ustalamy gdzie właściwe słowo się zaczyna a gdzie kończy. Usuwamy wszystko co nie znajduje się pomiędzy tymi dwiema wartościami.

Po trzecie, tablica wartości które reprezentuje dźwięk jest bardzo dużych rozmiarów. Przekracza to możliwości prostej implementacji w wybranej przeze mnie technologii ze względu na ograniczenia długości tablic. Dało by się to obejść, jednak nadal pozostał by problem czasu obliczeń, ponieważ danych jest bardzo dużo. Dlatego każdą ścieżkę dźwiękową o rozmiarze przekraczającym 8000 wartości liczbowych, kompresujemy do tej długości, wykorzystując średnią arytmetyczną. Po licznych testach okazało się, że mimo utraty części danych, sama „tendencja” i kształt funkcji zostają zachowane.

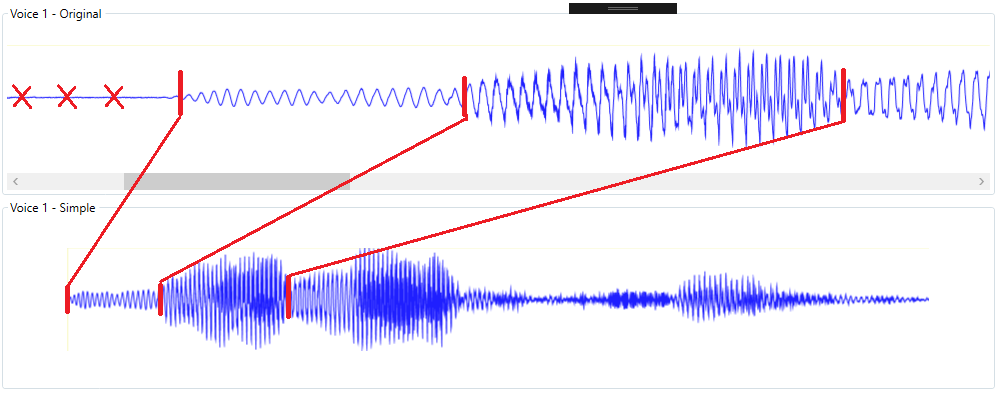
# Weryfikacja

## Proces porównania dwóch ścieżek, jako rezultat zostawia nam dwie uzupełnione macierze kosztów, oraz koszt całkowity najtańszej ścieżki. Weryfikować na tej podstawie możemy następujące rzeczy: czy na obu nagraniach znajduje się to samo słowo, lub czy oba nagrania zostały nagrane przez tą samą osobę. W obu przypadkach potrzebujemy jakiegoś punktu odniesienia w celu przyrównania do niego uzyskanej najlepszej ścieżki. W moim rozwiązaniu zdecydowałem przyrównać najlepiej uzyskaną ścieżkę do najgorszej uzyskanej ścieżki. Wiadomo, że dwa identyczne nagrania są porównane z całkowitym kosztem 0. W ten sposób posiadając wartość najlepszą oraz najgorszą jesteśmy w stanie ocenić jak podobne są do siebie nagrania.

Znalezienie ścieżki najgorszej odbywa się w sposób analogiczny co znalezienie ścieżki najlepszej, z tą różnicą że nie wybieramy sąsiadów o najmniejszym koszcie, a tych o koszcie największym.

Finalna opowiedz zatem jest wartością procentową, mówiącą jak bardzo podobne są do siebie oba nagrania (innymi słowy – odwrotność tego jak wielki był procentowo koszt najtańszej ścieżki).

# ScreenShoty



Rysunek - Pre-processing ścieżki

# 

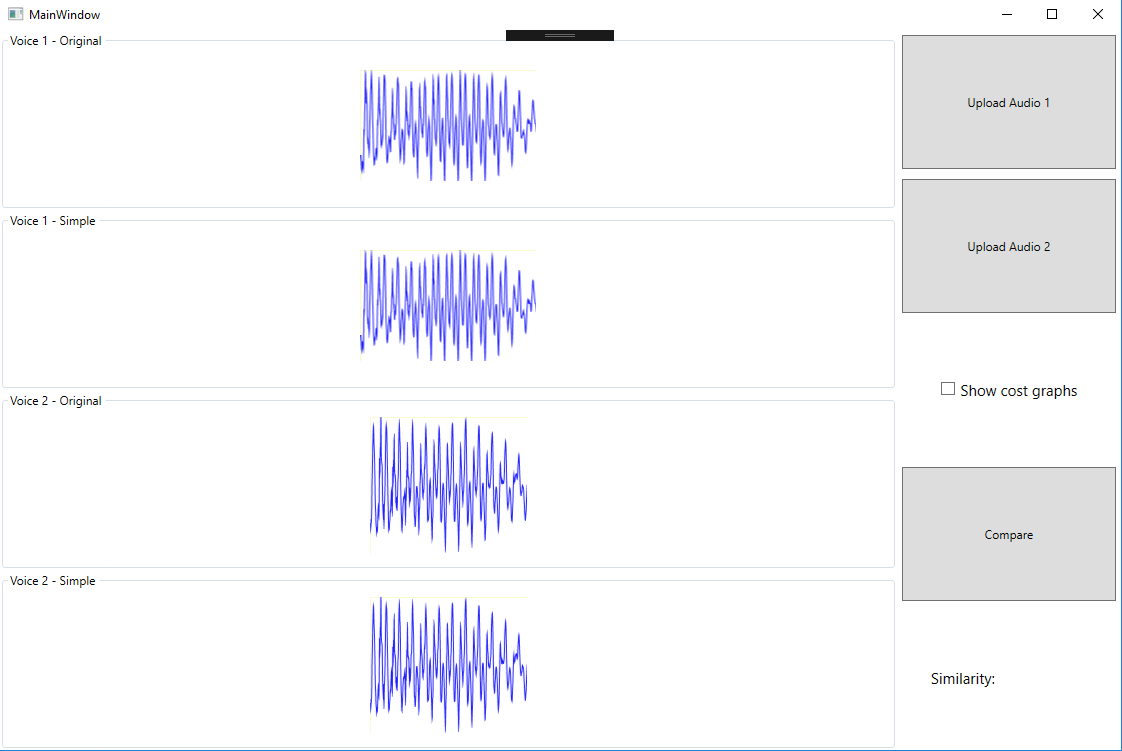
Rysunek - Interface aplikacji

# 

Rysunek - macierz kosztów lokalnych dla ścieżek z Rys.2



Rysunek - macierz kosztów globalnych dla ścieżek z Rys.2



Rysunek - Interface porównania krótszych ścieżek

# 

Rysunek - macierz kosztów lokalnych dla ścieżek z Rys. 5

# 

Rysunek - macierz kosztów globalnych dla ścieżek z Rys. 5

# Opis ScreenShotów

Rysunek 1 pokazuje przykładową kompresję ścieżki dźwiękowej. Widać wycięcie ciszy, oraz kompresję danych. Wzmocnienie jest wykonywane na samym początku obróbki dźwięku i jest wymagane do pozytywnej wizualizacji.

Rysunek 2 pokazuje jak się prezentuje interface użytkownika po wgraniu obu głosów do porównania. Jednocześnie widać uproszczenie obu porównywanych nagrań.

Rysunek 3 pokazuje macierz kosztów lokalnych dla wgranych w Rysunku 2 nagrań. Niestety ze względu na długość ścieżek, macierz wizualnie jest bardzo nieczytelna.

Rysunek 4 pokazuje macierz kosztów globalnych dla wgranych w Rysunku 2 nagrań. Niestety ze względu na długość ścieżek, macierz wizualnie jest bardzo nieczytelna.

Rysunek 5 pokazuje jak się prezentuje interface użytkownika po wgraniu dwóch krótkich wycinków nagrań, w celu lepszego pokazania macierzy kosztów.

Rysunek 6 pokazuje macierz kosztów lokalnych dla wgranych w Rysunku 5 nagrań. Wyraźnie widać miejsca tańsze (kolor czarny), którymi potencjalnie mogłaby być pociągnięta ścieżka najtańszego kosztu.

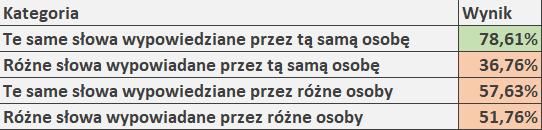
Rysunek 7 pokazuje macierz kosztów globalnych dla wgranych w Rysunku 5 nagrań. Wyraźnie widać całe czarniejsze obszary w których będzie biegła finalna ścieżka.

# Wyniki

Testy podzielone zostały na 4 tematyczne kategorie:

* Te same słowa wypowiedziane przez tą samą osobę
* Różne słowa wypowiadane przez tą samą osobę
* Te same słowa wypowiedziane przez różne osoby
* Różne słowa wypowiadane przez różne osoby

Dla każdej kategorii przeprowadzono 10 różnych testów i wyciągnięto z nich średni wynik.



# Wnioski

Zaproponowane i zaimplementowane rozwiązanie spełnia swoją role. Po wynikach widać, że rozwiązanie (z wyraźną różnicą 20 %) rozróżnia te same słowa wypowiadane na przez tą samą osobę, od całej reszty. Wydaje mi się że wyniki mogą być jeszcze lepsze. Problemem przede wszystkim jest brak porządnego pre-processingu. Dane należało by w jakiś sposób normalizować, oraz zastanowić się nad wygładzaniem szumów. Po drugie możliwa niedokładność może wynikać z utraty części danych w skutek kompresji. Rozwiązaniem mogła by być implementacja na platformach nisko poziomowych, które umożliwiają lepsze zarządzanie pamięcią. Ostatnim aspektem jest sama weryfikacja. Nie ma uniwersalnego narzędzia które pozwoliło by ocenić otrzymany wynik z algorytmu DTW. Prymitywne porównanie kosztu najtańszej ścieżki do potencjalnie najgorszej, jak widać po wynikach, spełnia swoje zadanie. Jednak można założyć że istnieje znacznie lepszy sposób.

Zatem wyniki zaprezentowanej pracy są satysfakcjonujące, natomiast zwiększając znacznie nakład pracy można prawdopodobnie polepszyć wyniki.

# Źródła

* <http://www.mini.pw.edu.pl/~rafalkoj/www/?download=DTW.pdf> – opis algorytmu dr inż. Janusz Rafałko, Mini PW 2018.