AKADEMIA GÓRNICZO-HUTNICZA IM. STANISŁAWA STASZICA W KRAKOWIE

WYDZIAŁ ELEKTROTECHNIKI, AUTOMATYKI, INFORMATYKI I INŻYNIERII BIOMEDYCZNEJ

KATEDRA AUTOMATYKI I INŻYNIERII BIOMEDYCZNEJ

Systemy głosowe
rok IV

Dokumentacja projektu – Rozpoznawanie komend głosowych

Spis treści:

1. Informacje wstępne

W systemach głosowych komunikacja między operatorem a systemem odbywa się za pomocą komend głosowych. Najczęściej są to proste hasła słowne. Aby system mógł zarejestrować, przeanalizować, rozpoznać i zinterpretować komendę próbka głosowa musi przejść wiele transformacji. Zaprojektowanie systemu w taki sposób aby komendy wydawane przez operatora były w większości poprawnie zinterpretowane jest nie lada wyzwaniem, o czym świadczyć może stosunkowo niska skuteczność wśród systemów rozpoznawania komend (np. poprzez kontrolery, ruch, dotyk). Fale dźwiękowe otaczają nas zewsząd i tylko dzięki potężnemu narzędziu jakim jest ludzki mózg jesteśmy w stanie pośród ogólnego hałasu wychwycić te sygnały, które nas interesują. Mózg przez wiele lat życia uczy się rozróżniać dźwięki stanowiące tło, lub zakłócenia i nie przepuszczać ich do naszej świadomości. Np. podczas rozmowy na przystanku przy bardzo zatłoczonych alejach, jesteśmy w stanie wychwycić sens wypowiedzi rozmówcy ,pomimo że przejeżdżające samochody generują dźwięki o zdecydowanie większym natężeniu. Podczas projektowania systemów głosowych jednak jesteśmy w dużo gorszej sytuacji. Nie poznaliśmy jeszcze większości sposobów w jaki działa ludzki mózg. Dodatkowo otrzymując próbkę dźwięku pozbawieni jesteśmy kontekstu. Nie znamy mimiki twarzy, odległości od źródła dźwięku, nie widzimy oraz nie doświadczamy warunków otoczenia. Nie widzimy zbliżającej się ciężarówki i nie wiemy, że za moment będziemy z próbki głosu musieli wyciąć warkot silnika o dużym natężeniu. Pomimo wszystkich tych trudności i problemów, udaje się stworzyć systemy głosowe wykorzystywane do komunikacji człowiek-maszyna. Odpowiednio ograniczając funkcjonalność systemu jesteśmy w stanie znacznie uprościć komunikację i tym samym zwiększyć skuteczność rozpoznawania komend. Systemy głosowe wypełniają lukę wszędzie tam gdzie nie możemy wydawać komend za pomocą rąk. Istnieje wiele systemów będących udogodnieniem dla kierowców prowadzących samochody, pozwalających np. zmieniać muzykę, przełączać mapy, wyszukiwać trasę przejazdu, odbierać połączenia. Z tego samego powodu systemy głosowe znacznie rozwinęły się w branży logistycznej, na halach magazynowych. Pracownik mogący bez użycia rąk odebrać polecenie z centrali, potwierdzić je i poinformować o bieżącym stanie, jest bardziej wydajny, co bezpośrednio przekłada się na zyski firm.

Celem naszej pracy było zaprojektowanie i realizacja systemu głosowego. W tym: implementacja algorytmu rozpoznającego polecenia wydawane za pomocą mowy, wykonanie aplikacji Web pozwalającej na komunikację użytkownika z systemem, testowanie utworzonego środowiska i prezentacja wyników.

2. Opis Algorytmu

DTW - Dynamic Time Warping

Algorytm pozwala na porównanie dwóch sygnałów głosowych zapisanych w postaci ciągów wektorów cech. Ze względu na to, że owe ciągi mogą mieć różną długość, ze względu na przeciąganie lub skracanie sylab w trakcie wypowiadania słów, potrzebny jest specjalny algorytm, który pozwoli na porównanie rozpoznawanego słowa ze wzorcem i określi który wzorzec odpowiada mu najlepiej. DTW w wielkim skrócie "marszczy" czas, tzn. rozciąga sylaby, bądź skraca je w taki sposób aby otrzymać najlepsze dopasowanie do wzorca. Dopasowanie wykonuje się dla każdego wzorca w słowniku, a wynikiem dopasowania jest odległość sygnału pomarszczonego od wzorca. Algorytm wskazuje wzorzec najbliżej sygnału i określa go jako wynik rozpoznawania. Implementacja opiera się na rozwiązaniu zadania programowania dynamicznego. Złożoność obliczeniowa ze względu na ograniczenia wprowadzone do algorytmu jest w przybliżeniu liniowa.

Algorytm DTW jako argument przyjmuje macierz będącą wektorem wektorów cech w kolejnych chwilach czasu. To jaką metodą otrzymane zostały cechy nie ma wpływu na przebieg algorytmu, natomiast może mieć wpływ na jakość rozpoznania. Jeżeli chodzi o końcową klasyfikację, to ze względu na zmienny rozmiar macierzy wejściowej nie ma dużego pola do wyboru jeżeli chodzi o algorytm klasyfikujący. Zaimplementowana została metoda najbliższego sąsiada. Można uogólnić metodę klasyfikacji do K-nearest-neighbours - metody k najbliższych sąsiadów. Działanie systemu powinno być wtedy mniej wrażliwe na cechy osobnicze wzorca.

Macierz współczynników które otrzymuje DWT w naszym przypadku zawiera współczynniki cepstralne, których obliczenie opiera się na współczynnikach liniowej predykcji LPC. Wyciągnięcie wektora cech z sygnału mowy opiera się na heurystykach i nie ma jednego dobrego zestawu cech. Najbardziej popularnymi zestawami cech są współczynniki: MFCC - melowo-częstotliwościowe współczynniki cepstralne, LPC - liniowej predykcji, formanty (główne składowe częstotliwościowe).

Założenia algorytmu

Niemożliwe jest stworzenie algorytmu, który byłby całkowicie niewrażliwy na szumy, zmianę długości porównywanych sygnałów mowy i zmianę głosu osoby nagrywającej.

Aby poradzić sobie z pierwszym i drugim problemem stosuje się przetwarzanie wstępne sygnału wejściowego. W trakcie testów algorytmu nagrywaliśmy sygnały głosowe o długości 3 sekund. Algorytm usuwa ciszę na początku i końcu nagrania, lecz nie zostaną usunięte części sygnału, w których obecny jest tylko szum.

To samo słowo może być jednak wymawiane w różnym tempie. Aby poradzić sobie z tym problemem używamy algorytmu 'Dynamic Time Warping' do porównywania współczynników cepstralnych. Algorytm więc skonstruowany jest tak, by nie był wrażliwy na długość wymawiania tych samych słów oraz różną długość ciszy na początku i końcu bazy wzorców i sygnałów wejściowych dla nagrań nie zawierających szumów.

Użyty algorytm nie gwarantuje niewrażliwości na zmianę pobudzenia traktu głosowego (a więc na zmianę osoby wypowiadającej komendę). Widmo sygnału jest logarytmowane, a następnie jest obliczana odwrotna transformata Fouriera tego logarytmu. Dzięki zlogarytmowaniu widmo zamieniamy iloczyn w dziedzinie częstotliwości na sumę. A więc współczynnik jest sumą części odpowiadających za chwilową charakterystykę traktu głosowego i pobudzenie. Z tego względu algorytm jest zależny od mówcy. Możemy jednak założyć, że dla nawet dla dwóch różnych mówców wartości współczynników są podobne, gdyż część wynikająca z traktu głosowego w teorii nie zmienia się. Będziemy więc mogli rozróżnić nadal komendy, które mają bardzo różne części wynikające z traktu głosowego. Jeśli jednak części te będą podobne, pobudzenie będzie miało duży wpływ na wynik algorytmu. Algorytm zawiera również filtrację wstępną (preemfazę) sygnału, która zwiększa składowe dużych częstotliwości (filtr górnoprzepustowy) w celu zmniejszenia składowych szumu kwantyzacji.

W celu testowania algorytmu nagrane zostało próbki w różnych środowiskach, dla różnych osób. Badaliśmy skuteczność algorytmu gdy jedna osoba wprowadzała słowa do bazy i wydawała komendy oraz jedna osoba wgrywała słowa do bazy, a inna wydawała komendy. Oprócz tego badaliśmy wpływ otoczenia na skuteczność algorytmu – część sygnałów nagrana została w warunkach korzystnych (ciche miejsce), a część w warunkach niekorzystnych (kilkanaście osób w pomieszczeniu). Oprócz tego część sygnałów nagrana została innym mikrofonem, aby określić wpływ jakości urządzenia wejściowego na działanie algorytmu.

Implementacja algorytmu

Algorytm został zaimplementowany obiektowo w języku Python na podstawie algorytmu dostarczonego przez Data Enginner'a. Data Scientist pomagał w zrozumienu poszczególnych kroków algorytmu. Dodatkowo algorytm wykorzystywał podane poniżej biblioteki:

- math – wykorzystanie funkcji matematycznych np. sin(), max(), min()
- numpy – umożliwia tworzenie wektorów i macierzy oraz definiuje wszystkie operacje na nich
- scipy.io – umożliwia wczytywanie z plików danych w postaci wektorów, których format jest obsługiwany przez bibliotekę numpy

Wszystkie te biblioteki są dostępne w pakiecie Anaconda. Program został i przetestowny wykorzystując Pythona w wersji 3.5.2 z pakiem Anaconda w wersji 4.2.0. Klasa wraz z algorytmem zawiera się w pliku cepsrtalne.py

Opis klasy w której został zaimplementowany algorytm

Klasa nosi nazwę Alg. Składa się ona z dziewięciu pól oraz trzech metod które służą do wstępnego przetworzenia sygnału, segmentacji danych, ekstrakcji cech i klasyfikacji.
Opis pól klasy:

- *fpr* – częstotliwość próbkowania sygnału. Przeliśmy, że wszystkie sygnały, które są wczytywane przez są próbkowane z standardową częstotliwością 44kHz.
- *Np* – liczba biegunów w filtrze predykcji, przyjęta wartość 10.
- *Nc* – liczba wyznaczanych współczynników cepstralnych, w literaturze podaje się że minimalna wartość to 8, jednak powinno się przyjmować je nad miarowo. W naszym przypadku ich wartość wynosi 12.
- *twind* – długość okna obserwacji (ramki danych) w milisekundach, przyjmuje się że sygnał mowy jest stacjonarny w przedziale od 5ms do 30ms. Przyjęliśmy wartość wynoszącą 30ms. Powoduje to zmniejszenie ilości obliczeń przez co algorytm działa szybciej.
- *tstep* – przesuniecie pomiędzy kolejnymi położeniami okna w milisekundach. Dla naszego algorytmu wartość ta wynosi 10ms.
- *slowa* – jest to lista słów rozpoznawanych przez algorytm. Przyjęliśmy cztery słowa:
  - lewo
  - prawo
  - start
  - stop

- *Mlen* – jest wyznaczany na podstawie twind i fpr i służy do wyliczenia ilości próbek przypadających na jedną ramkę.
- *Mstep* – jest wyznaczmy na podstawie tstep i fpr i służy do wyliczenia ilości próbek pomiędzy kolejnymi położeniami ramki
- *M* – określa ilość elementów w zmiennej slowa.

Poza opisanymi polami algorytm składa się jeszcze z trzech metod: *cisza, cepstrum, dtw*

Metoda cisza

Każda zarejestrowaną komendę należy podać wstępnemu przetworzeniu. Metoda cisza odpowiada
za usunięcie ciszy z początku i końca sygnału i pozostawienie samej komendy. Sygnał jest normowany, a następnie jest wyliczana energia sygnału w dziesięco milisekundowych ramkach.
Eliminuje się n-początkowych oraz z-końcowych ramek dla których energia jest mniejsza od zadanego progu wynoszącego 0.25*fpr/8000.

Sposób użycia funkcji

*wyjscie=Nazwa_obiektu.**cisza**(sygnal)*

gdzie:

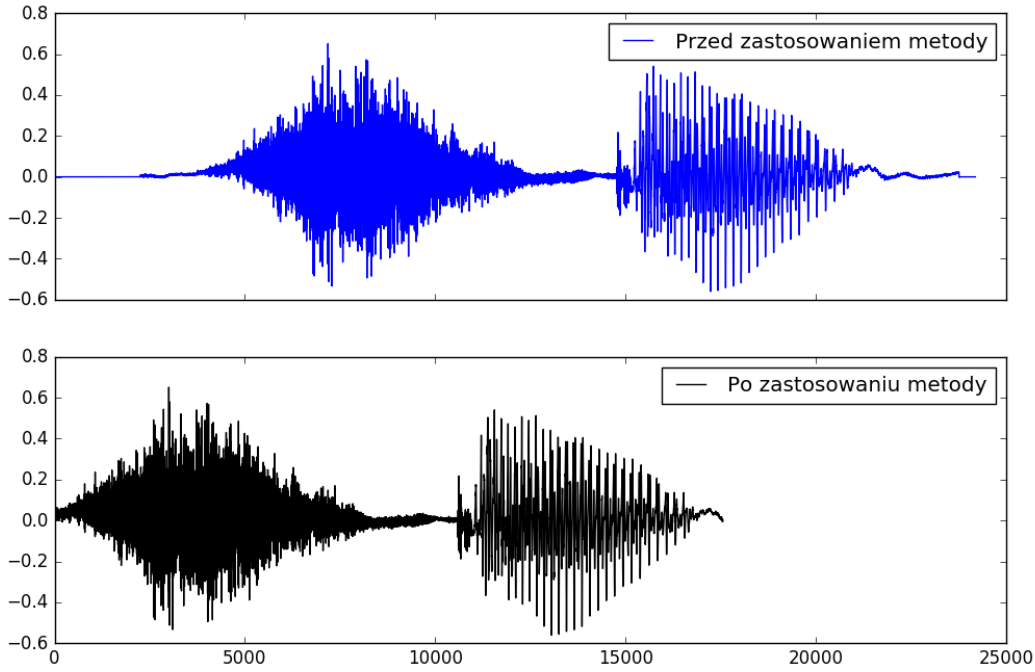
- *sygnal* to wektor pionowy zawierający nagraną komendę.
- *Nazwa_obiektu* jest nazwą obiektu klasy Algorytm.
- *wyjscie* to wektor pionowy zawierający komendę z usuniętą ciszą na początku i końcu przedziału.

Przykład użycia:

```
sygnal=sio.loadmat("Komenda_Prawy")["y"] # Wczytanie sygnalu
Algorytm=Alg()  # Stworzenie obiektu klasy Alg
wyjscie=Algorytm.cisza(sygnal) # wykonanie metody cisza
```

Wynik działania metody

Jako sygnał wejściowy została podana komenda start



Jak można zauważyć na rysunku przedstawiony powyżej metoda spełnia swoje zadanie usuwając fragmenty sygnału, które nie wnoszą żadnej informacji dotyczącej podanej komendy. Możemy również zobaczyć jak bardzo skomplikowany jest sygnał jednej prostej komendy.

Metoda cepstrum

Metoda ta służy do wyznaczenia macierzy współczynników cepstralnych na podstawie wcześniej już przetworzonego sygnału za pomocą metody cisza. Metoda cepstrum na początku dokonuje kolejnego przetworzenia sygnału dokonując preemfazy. Zabieg ten jest wykonywany w celu zmniejszenia szumu składowych kwantyzacji o częstotliwościach lezących w zakresie fonicznego sygnału analogowego. Zabieg ten polega za zwiększeniu składowych dużych częstotliwości fonicznego sygnału analogowego, wykorzystuje się do tego pochodną zadanego sygnału. W następnej fazie sygnał jest segmentowy na ramki czasowe zgodnie z parametrami klasy podanymi wcześniej. Następnie usuwana jest wartość średnia z ramki i stosowane jest okno czasowe Hamminga. W kolejnym etapie dla każdej ramki wyliczane są współczynniki cepstralne z współczynników filtra LPC. W efekcie otrzymujemy macierz współczynników cepstralnych.
Metoda zwraca dwuelementową listę.

Sposób użycia funkcji

*wyjscie=Nazwa_obiektu.***cepstrum***(sygnal)*

gdzie:

- *sygnal* to wektor pionowy zawierający komendę przetworzoną przez metodę cisza.
- *Nazwa_obiektu* jest nazwą obiektu klasy Algorytm.
- *wyjscie* to dwuelementową listę. Pierwszy elementem listy jest macierz współczynników, drugim zaś liczba ramek na który został podzielony sygnał.

Przykład użycia:

```
sygnal = sio.loadmat("Komenda_Prawy")["y"] # Wczytanie sygnalu
Algorytm = Alg()  # Stworzenie obiektu klasy Alg
wyjscie = Algorytm.cisza(sygnal) # wykonanie metody cisza
Lista = Algorytm.cepstrum(wyjscie) # wykoanie metody cepstrum
```

Wynik działania metody dla komendy lewo:

W wyniku zadziałania metody komenda start została podzielona na 53 ramki. Powoduje to, że macierz współczynników cepstranych ma rozmiar 56x12. Każdej ramce został przypisany zestaw 12
współczynników cepstralnych co jest zgodne z przedstawianym wcześniej parametrem odpowiedzialnym za ilość współczynników cepstralnych. Podawanie tutaj całej macierzu współczynników cepstralnych mija się z celem, ponieważ nikt nie będzie analizował aż 636 elementów niej zawartych. Można natomiast przedstawić zestaw współczynników cepstralnych uzyskanych dla pierwszej ramki:

[3.14609251 1.5166825 2.32994712 -0.10022309 0.79164634 -0.94283064
0.61142632 -1.43954347 0.2615665 0.60342047 -0.09856935 0.14245478]

Metoda *cepstrum* jest wykorzystwana do tworzenia bazy wzorców, jaki i również do tworzenia macierzy współczynników cepstralnych, które później są wykorzystywane za pomocą metody *dtw*
do klasyfikowania zadanych komend

Metoda dtw

Metoda służy do rozpoznawania słowa poprzez porównanie macierzy jego współczynników cepstralnych z wzoracami za pomocą metody DTW (Dynamic Time Warping). DTW to klasa algorytmów które służą do porównywania szeregów czasowych. Metody DTW pozwalają na znalezienie najmniejszej odległości pomiędzy dwoma szeregami przy dopuszczeniu transformacji czasu dal obu szeregów. Miara odległości oparta na algorytmie DTW jest bardziej odpowiednia gdy porównujemy szeregi o podobnej strukturze lecz przesunięte lub rozciągnięte w czasie. W dużym uproszczeniu: dysponując dwoma sygnałami x i y o długościach M i N (w naszym przypadku ilość ramek sygnału) algorytm układa siatkę o wymiarach N x M, gdzie w każdym jej polu obliczana jest odległość pomiędzy próbkami (w naszym przypadku odległość euklidesowa pomiędzy wektorami współczynników cepstralnych dla odpowiednich ramek) sygnałów o numerach n i m, gdzie n = 1,2,...,N i m=1,2,...,M. Następnie poszukiwana jest ścieżka łącząca punkty (1,1) z (N,M). W taki sposób aby koszt ścieżki był jak najmniejszy. Ścieżka musi być ciągła, każdy punkt musi sąsiadować z poprzednim. Dla każdego kolejnego elementu ścieżki jego indeksy m i n nie mogą maleć, ścieżka musi być monotoniczna. Problem znalezienia najkrótszej ścieżki jest rozwiązywany za pomocą programowania dynamicznego (problem znajdowania najdłuższego wspólnego podciągu, LCS) . Złożoność algorytmu wynosi O(N·M). W efekcie działania algorytmu otrzymujemy wektor zawierający wartość najkrótszej ścieżki do każdego z wzorca oraz indeks najmniejszego elementu macierzy.

Sposób użycia funkcji

indeks, odleglosci=*Nazwa_obiektu.***dtw***(Cx,Cwr,Nramek)*

gdzie:

- *Cx* to macierz współczynników cepstralnych sygnału, który chcemy zidentyfikować uzyskany dzięki metodzie cepstrum.
- *C*wzr to lista macierzy wzorcowych współczynników cepstralnych, do stworzenia tej listy również można wykorzystać metodę cepstrum, podając jej wzorcowe sygnały.
- *Nramek* to lista ilości ramek odpowiadająca odpowiednim macierzą w Cwzr.
- *odleglosci* to wektor zawierający najkrótsze długości ścieżek pomiedzy Cx a każdą macierzą z listy Cwzr.
- *indeks* jest indeksem wektora *odleglosci,* dla którego wartość w tym wektorze jest najmniejsza. Innymi słowy zwraca on indeks najmniejszej wartości w wektorze *odleglosci.*

Przykład użycia

Zakładamy, że stworzyliśmy już listę macierzy *Cwzr* oraz odpowiadającą im listę ramek *Nramek.*

Kolejne macierze wzorcowe w liście odpowiadają słowom: lewo,prawo,start,stop. Tak samo jak w zmiennej *slowa* zdefiniowanej jako jedna z pól klasy.

```
Algorytm=Alg()
y = sio.loadmat('../SG/Wavelet/start_Karolina_2') # wczytanie pliku zawierajaco komende start
y = y['y'] # wczytanie komendy do zmiennej y
y = y[:, 0] # komenda głosowa ma dwa kanały więć wybiramy tylko jeden kanał
syg = Algorytm.cisza(y)  # odcięcia ciszy z sygnału
Cx = Algorytm.cepstrum(syg)  # wyznaczenie macierzy współczynników cepstralnych
nr, odleglosc = b.dtw(Cx[0], Cwzr, Nramek) # Klasyfikacja sygnalu
print('Wczytano start,\nRozpoznano {},\nWektor odległości {}'.format(Algorytm.slowa[nr], odleglosc))
```

Wynik działania powyżej przedstawionego algorytmu:

Wczytano start,
Rozpoznano start,
Wektor odległości [4.05555847 3.43467927 2.52675626 4.19724796]

Jak widzimy najmniejsza odległość jest do wzorca trzeciego, który jest wzorcem słowa start, tak więc rozpoznawanie komendy przebiegło prawidłowo.

Testowanie poprawności implementacji algorytmu

Najprościej sprawdzić działanie algorytmu na danych testowych. Wynikiem testu były zdania w formie:

<Nazwa wczytanej komendy> to <nazwa rozpoznanej komendy>, <wektor odległości>

Wyniki testu:

```
lewo to lewo, [ 0.          2.29052189  4.13228302  5.81446397]
lewo to lewo, [ 2.40757516  2.51416631  4.33090219  5.63726275]
lewo to lewo, [ 2.09952197  2.71061027  4.05510047  5.65013557]
lewo to lewo, [ 2.3842971   2.6831393   4.14077782  6.14104141]
prawo to prawo, [ 2.57421985  0.          4.56611576  7.42960373]
prawo to prawo, [ 2.78624717  2.30245791  4.20277071  6.95717671]
prawo to prawo, [ 2.73850642  2.18775838  4.69807601  7.67753523]
prawo to prawo, [ 2.44346479  2.12889944  4.54110114  7.68231522]
start to start, [ 3.86584299  3.29568316  0.          4.36473288]
start to start, [ 4.05555847  3.43467927  2.52675626  4.19724796]
start to start, [ 4.02033715  3.74007073  2.8279758   4.24157431]
start to start, [ 4.30684637  3.67073155  3.31277276  3.61667766]
stop to stop, [ 6.08212694  7.03294449  4.72430429  0.        ]
stop to stop, [ 4.27812437  4.40001481  3.55378665  3.24485962]
stop to stop, [ 4.8773367   4.38359242  3.89180511  2.46823679]
stop to stop, [ 4.4823141   4.4754866   3.66997733  3.27118922]
```

Wszystkie próbki sygnału pochodzą od tej samej osoby. Tam gdzie w wektorze odległości występują zera wczytywana komenda jest sygnałem wzorcowy. Wynika z tego, że porównanie dwóch tych samych komend daje nam odległość zero pomiędzy ich macierzami współczynników cepstralnych . Jest to pierwsza przesłanką, że algorytm został dobrze zaimplementowany. Drugą przesłanką wskazująca na poprawność działania algorytmu jest jego 100% skuteczność w przypadku tej próbki testowej. Należy jednak zauważyć, że na przykład dla algorytmu wczytana komenda prawo znajduje się blisko wzorców sygnałów prawo i lewo, różnice są bardzo niewielkie jednak algorytm poprawnie interpretuje wczytaną komendę. Mimo wszystko, jeżeli jakość sygnału, który będzie porównywany, będzie niska może to powodować znaczy spadek skuteczności rozpoznawania słów.
Podobny zjawisko możemy również zarejestrować w przypadku sygnału stop, który znajduje się blisko wzorców sygnałów stop i start. Prawdopodobnie bardzo częstym zjawiskiem będzie mylenie komend: lewej z prawej i start z stop.

3. Web aplikacja

Wymagania aplikacji

Do uruchomienia aplikacji na własnym komputerze wymagane są:
- zainstalowany interpreter **Pythona 2.7**
- moduły Pythona do zastowonań naukowych takie jak: **scipy, numpy**
- moduł **Django** – framework do tworzenia aplikacji webowych w Pythonie
- moduł **sounddevice** biblioteka Pythona umożliwiająca nagrywanie dźwięku

Instalację wyżej wymienionych modułów można przeprowadzać pojedynczo jednak podczas instalacji modułu **scipy** mogą wystąpić problemy. Aby tego uniknąć należy rozważyć instalacje paczki modułów Anaconda która zawiera moduły scipy i numpy.

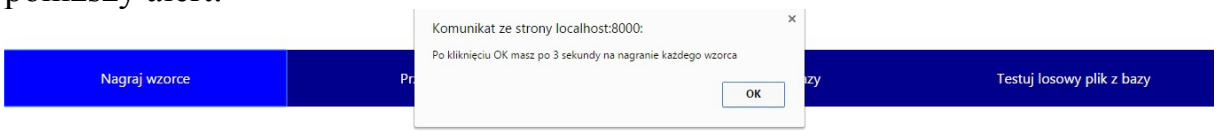
Jak uruchomić aplikację ?

1. Przejście do katalogu w którym stworzyliśmy aplikację (powinien być w nim plik mange.py)
2. W przypadku gdy korzystamy ze standardowego interpretera Pythona należy wpisać w konsoli polecenia **python manage.py runserver** w aplikacja będzie dostępna pod adresem http://127.0.0.1:8000/wizualizacja/
3. Gdy korzystamy z interpretera Pythona z pakietu Anaconda należy wpisać **(ścieżka do pakietu anaconda)/Anaconda2/python manage.py runserver**

Instrukcja obsługi interfejsu webowego

By uzyskać dostęp do interfejsu webowego aplikacji uruchomionej w poprzednim rozdziale, należy w przeglądarce wejść na stronę http://localhost:8000/wizualizacja/ . Po jej załadowaniu wyświetli się nam pasek tytułowy oraz cztery polecenia do wyboru.

Jeśli klikniemy pierwszy z przycisków, o nazwie „Nagraj wzorce”, ukaże się nam poniższy alert:



Po naciśnięciu „OK” użytkownik ma 3 sekundy na nagranie pierwszego wzorca („lewo”), a potem usłyszy on swoją próbkę:

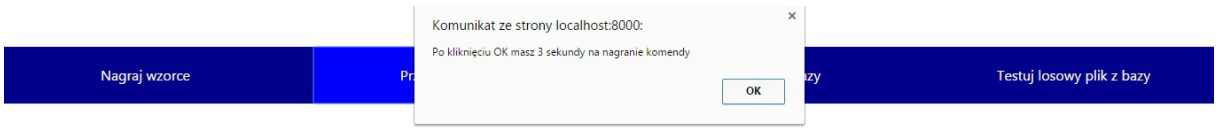
Gdy pokaże się napis „Powiedz: prawo” użytkownik ma 3 sekundy na nagranie następnego wzorca („prawo”), który też następnie usłyszy:

Gdy pokaże się napis „Powiedz: start” użytkownik ma 3 sekundy na nagranie trzeciego wzorca („start”), który też następnie usłyszy:

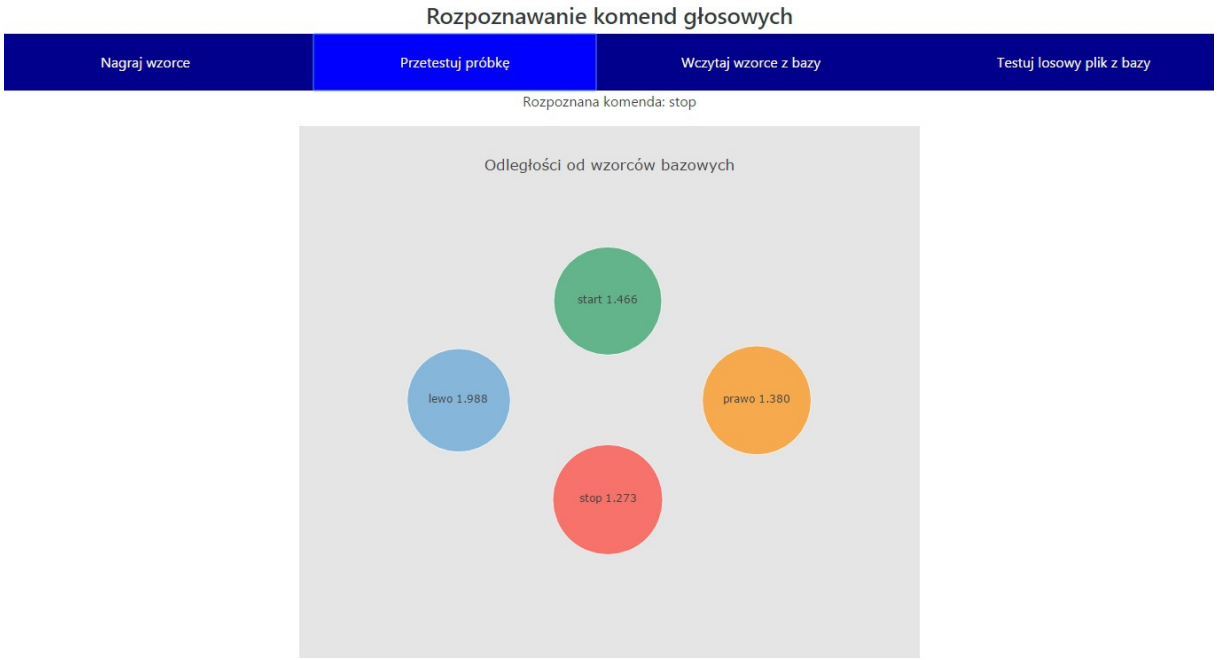
Gdy pokaże się napis „Powiedz: stop” użytkownik ma 3 sekundy na nagranie ostatniego wzorca („stop”), który też następnie usłyszy:

Jeśli „Nagraj wzorce” zostanie kliknięte po wykonaniu jakiejś procedury identyfikacyjnej, wykres z jej wynikiem zostanie usunięty, by nie wprowadzać w błąd użytkownika.

Następny przycisk to „Przetestuj próbkę”. Jego działanie będzie poprawne tylko w przypadku, gdy użytkownik stworzył już bazę wzorców za pomocą opcji „Nagraj wzorce” lub „Wczytaj wzorce z bazy”. Po jego kliknięciu ukazuje się poniższy alert:



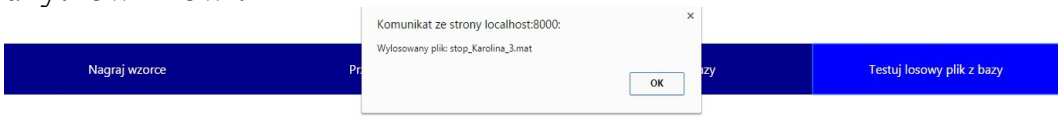
Po naciśnięciu „OK” użytkownik ma 3 sekundy na nagranie identyfikowanej próbki, a potem usłyszy on zarejestrowany dźwięk. Następnie wyświetli się napis informujący, jaki wzorzec został rozpoznany oraz wykres. Są na nim zawarte wyliczone przez algorytm odległości od wzorcowych wektorów cech oraz odpowiadające każdemu wzorcowi koło, którego wielkość maleje wraz z rosnącą odległością.



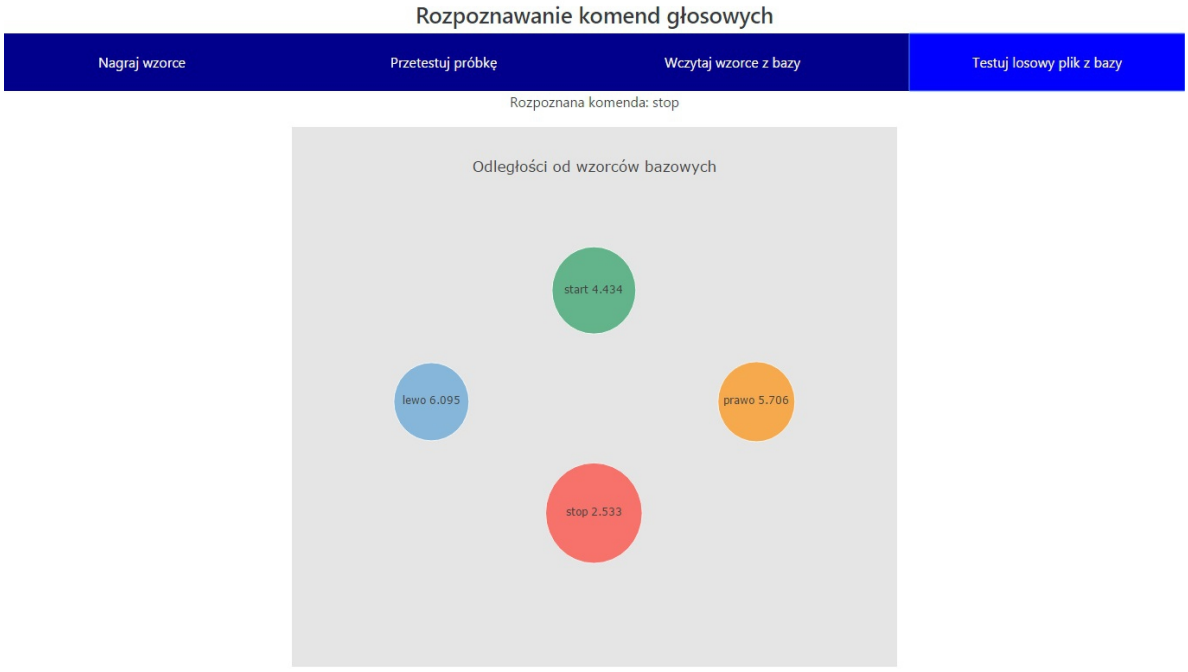
Jeśli „Przetestuj próbkę” zostanie kliknięte po wykonaniu jakiejś procedury identyfikacyjnej, wykres z jej wynikiem zostanie usunięty, by nie wprowadzać w błąd użytkownika.

Kolejnym przyciskiem jest „Wczytaj wzorce z bazy”. Jego naciśnięcie spowoduje, że Django jako wzorcowe wczyta wybrane pliki znajdujące się w bazie (lewo_Karolina_1.mat, prawo_Karolina_1.mat, start_Karolina_1.mat, stop_Karolina_1.mat). Jeśli „Wczytaj wzorce z bazy” zostanie kliknięte po wykonaniu jakiejś procedury identyfikacyjnej, wykres z jej wynikiem zostanie usunięty, by nie wprowadzać w błąd użytkownika.

Ostatnia komenda to „Testuj losowy plik z bazy”. Wylosuje ona jeden z plików umieszczonych przez nas w bazie próbek i wyświetli alert z jego nazwą użytkownikowi:



Przeprowadzi on jego identyfikację, a następnie wyświetli się napis informujący, jaki wzorzec został rozpoznany oraz wykres. Są na nim zawarte wyliczone przez algorytm odległości od wzorcowych wektorów cech oraz odpowiadające każdemu wzorcowi koło, którego wielkość maleje wraz z rosnącą odległością.



Jeśli „Testuj losowy plik z bazy” zostanie kliknięte po wykonaniu jakiejś procedury identyfikacyjnej, wykres z jej wynikiem zostanie usunięty, by nie wprowadzać w błąd użytkownika.

Dokumentacja backendu

Do obsługi części backendowej aplikacji wykorzystany został framework Django. Narzędzie to umożliwia tworzenie aplikacji webowych w Pythonie. Dzięki podzieleniu całego projektu na sekcje odpowiadające min. za obsługę szablonów, bazy danych praca nad rozwojem aplikacji jest usystematyzowana i w dużym stopniu zautomatyzowana. Django odpowiada za odpowiednią obsługę request- ów generowanych podczas działania aplikacji. Do ich obsługi wykorzystywane są następujące funkcje:

recordModel ()
Służy do utworzenia bazy próbek głosowych wykorzystywanych do rozpoznawania komend w późniejszym działaniu aplikacji. Po odebraniu odpowiedniego request-a następuje rozpoczęcie nagrywania głosu. Po jego zakończeniu dane są odpowiednio formatowane i dodawane do bazy.

record()
Odpowiada za nagranie komendy którą chcemy rozpoznać i wywołaniu na jej rzecz funkcji odpowiadającej za rozpoznawanie poleceń. Wynik działania algorytmu jest następnie konwertowany na odpowiedni typ danych i odsyłany z powrotem do przeglądarki.

fileModel()
Służy do utworzenia bazy danych na podstawie próbek zapisanych na dysku. Pliki z danymi są kolejno wczytywane i poddawane formatowaniu. Następnie dane dodawane są do bazy która będzie służyć jako baza nagrań wzorcowych w dalszej pracy aplikacji.

fileMatch()
Funkcja ta wywoływana jest z argumentem określającym nazwę pliku który chcemy przesłać do funkcji rozpoznawającej komendy. Po otworzeniu pliku i odczytaniu jego zawartość dane są formatowane analogicznie jak w przypadku poprzednich funkcji i przekazywane do rozpoznania. Wynik tej operacji jest następnie odsyłany z powrotem do przeglądarki.

Do nagrywania dźwięku wykorzystana została biblioteka Pythona **sounddevice.** Umożliwia ona w prosty sposób nagrywanie dźwięków po uprzednim określeniu parametrów nagrania najważniejsze z nich to:
- częstotliwość próbkowania
- czas trwania nagrania

- liczba kanałów
- format w jakim będą zapisywane dane

Moduł ten można pobrać za pomocą standardowego narzędzia do pobierania modułów Pythona – PIP. Kompletna dokumentacja dotycząca obsługiwania biblioteki **sounddevice** dostępna jest pod adresem http://python-sounddevice.readthedocs.io/

Dokumentacja frontendu

Do wizualizacji wyników pracy algorytmu wykorzystaliśmy dwie biblioteki: Plotly.js i Bootstrap. Utworzyliśmy funkcje Javascriptu aktywowane po kliknięciu odpowiednich przycisków. Oto krótki ich opis.

fileBase()

Służy do rozpoznania przez Django wybranych plików znajdujących się w bazie (lewo_Karolina_1.mat, prawo_Karolina_1.mat, start_Karolina_1.mat, stop_Karolina_1.mat) jako wzorcowych. Wywoływana po naciśnięciu „Wczytaj wzorce z bazy". Usuwa wyniki poprzedniej identyfikacji i wysyła odpowiedni request do serwera.

recordAndMatch()

Służy do rozpoznania przez Django podanych po kolei przez użytkownika próbek dźwięku jako wzorcowych. Wywoływana po naciśnięciu „Nagraj wzorce". Usuwa wyniki poprzedniej identyfikacji, a następnie informuje użytkownika o przebiegu nagrania i wysyła odpowiedni request do serwera. Wyświetla na ekranie polecenia wypowiedzenia odpowiednch komend.

sendAndMatch()

Służy do porównania próbki wybranej losowo z bazy ze wzorcem (gotowym lub tym nagranym przez użytkownika). Wywoływana po naciśnięciu „Testuj losowy plik z bazy". Usuwa wyniki poprzedniej identyfikacji, losuje plik, podaje użytkownikowi jego nazwę i wysyła odpowiedni request do serwera, zaś po otrzymaniu odpowiedzi wyświetla wyniki identyfikacji.

recordTest()

Służy do porównania nagranej przez użytkownika próbki ze wzorcem (gotowym lub tym nagranym przez użytkownika). Wywoływana po naciśnięciu „Przetestuj próbkę". Usuwa wyniki poprzedniej identyfikacji, informuje użytkownika o przebiegu nagrania i wysyła odpowiedni request do serwera, zaś po otrzymaniu odpowiedzi wyświetla wyniki identyfikacji.

genChart()

Służy do wyświetlenia wykresu z wynikami identyfikacji. Są na nim zawarte wyliczone przez algorytm odległości od wzorcowych wektorów cech oraz odpowiadające każdemu wzorcowi koło, którego wielkość maleje wraz z rosnącą odległością.

4. Testy

Do testów użyta została baza składająca się z czterech komend:
-lewo,
-prawo,
-start,
-stop.

Pierwszy test został przeprowadzony w korzystnych warunkach. Ta sama osoba nagrywała bazę i nagrywała komendy.

Użyto 4 razy każde słowo, a więc w sumie 16 słów.
Skuteczność zaimplementowanego algorytmu wyniosła w tym teście 100%.
Warto jednak wskazać, że otrzymywano zbliżone wartości kosztów dla komend „lewo" i „prawo". Na tej podstawie, przypuszczaliśmy, że słowa te będą problematyczne dla próbek nagranych w niekorzystnych warunkach, lub dla różnych osób.

Następnie przeprowadzono test używając tej samej bazy, w korzystnych warunkach, lecz komendy mówiła inna osoba z użyciem mikrofonu dużo gorszej jakości (wbudowany w laptopie). Warto zauważyć, że osobą nagrywającą bazę była kobieta, a osobą wydającą komendy mężczyzna.

Wyniki:
-lewo – 100%,
-prawo – 60%,
-start – 100%,
-stop – 20%.

Największe problemy sprawiła komenda „stop". Algorytm zazwyczaj klasyfikował ją jako „start". Komenda „prawo" była rozpoznawana dobrze, lub klasyfikowana jako „lewo".

Postanowiliśmy również przetestować algorytm używając bazy i próbek nagranych mikrofonem gorszej jakości. Ta sama osoba tworzyła bazę i nagrywała komendy.

Otrzymaliśmy następujące wyniki:
-lewo – 60%,
-prawo – 80%,
-start – 70%,
-stop – 80%.
Na podstawie powyższych danych wnioskujemy, że bardzo ważna jest jakość użytych sygnałów.

Próbowano również zbadać wpływ otoczenia na wyniki algorytmu. W tym celu przeprowadzono testy w pomieszczeniu, w którym znajdowało się kilkanaście osób. Najpierw użyto bazy nagranej w korzystnych warunkach. Komendy wydawała ta sama osoba, która nagrywała bazę:
-lewo – 40%,
-prawo – 20%,
-start – 40%,
-stop – 0%.

Dla porównania ta sama osoba nagrała nową bazę w tym samym środowisku, w którym odbywały się testy:
-lewo – 80%,
-prawo – 30%,
-start – 70%,
-stop – 50%.

Na podstawie wyników dwóch powyższych testów wnioskujemy, że algorytm działa zdecydowanie lepiej, gdy baza nagrywana jest w tym samym środowisku, w którym wydawane będą komendy.

Skuteczność algorytmu w sytuacji, gdy inna osoba tworzy bazę, a inna wydaje komendy również uległa zdecydowanemu pogorszeniu.

Najważniejszym wnioskiem płynącym z testów jest to, że skuteczność algorytmu bardzo mocno zależy od ilości szumów podczas wydawania komend. Aby zwiększyć skuteczność algorytmu można więc popracować nad lepszym algorytmem eliminacji zakłóceń.

Oprócz tego dowiedzieliśmy się, że algorytm działa lepiej, kiedy baza nagrywana jest w tym samym środowisku, w którym będą wydawane komendy.