Rozpoznawanie emocji z głosu Documentation

Wydanie

Elżbieta Plaszczyk

06 lut 2018

Contents:

ROZDZIAŁ 1

speech_emotion_recognition

1.1 FeatureImportance module

FeatureImportance.**feature_importance**(*path_pattern*, *emotions*)

Funkcja tworzy wektor cech z plików podanych w path_pattern i wypisuje ważność każdej z cechy

Parametry

- **path_pattern** – Ścieżka do plików, z których mają być obliczone wektory cech
- **emotions** – Lista emocji które mają być uwzględnione

Zwraca None

1.2 HMM module

class HMM.**HMM**(*transition_ppb*, *states_num*, *observations*)

Klasy bazowe: object

Klasa implementująca algorytm Ukryte Modele Markowa dla problemu rozpoznawania emocji z głosu.

Dany jest zbiór uczący zawieracjący obserwację (wektory cech), z któżych każda ma przypisaną emocję jaką dany wektor

Parametry

- **hidden_states_num** (*int*) – liczba ukrytych modeli Markowa
- **observations_num** (*int*) – liczba wszystkich możliwych obserwacji
- **observation_dict** (*dict*) – słownik zawierający dla każdej obserawacji jej index w tablicy emission_ppb
- **emission_ppb** (*matrix[hidden_states_num][observations_num]*) – tablica zawierająca dla każdego stanu S i każdej obserawcji O prawdopodobieństwo wygenerowanie B w stanie O

- **transition_ppb** (*matrix[hidden_states_num][hidden_states_num]*) – tablica prawdopodobieństw przejść pomiedzy stanami. matrix[i][j] - prawdopodobieństwo przejśćia z stanu i do stanu j
- **initial_ppb** (*list[hidden_states_num]*) – lista prawdopodobieństw przejść z stanu początkowego dostanu każdego z ukrytych stanów.

backward_algorithm(*ob_sequence*)

Implementacja algorytmu sufiksowego (backward algorithm)

Parametry **ob_sequence** (*list*) – sekwencja obserwacji

Zwraca

matrix[hidden_states_num][len(observation_seq)], matrix[i][t]

Opis algorytmu:

Dane: Y = [y_0, y_1, … , y_n] - observation_seq X = [x_1, x_1, … , x_k] - ukryte stany markowa

Cel:

macierz beta[[hidden_states_num][n]) taka, że: beta[i][t] = P(Y[t+1] = y_t+1, Y[t+1] = y_t+1, …, Y[n] = y_n | X_t = i) - prawdopodobienstwo zaobserwowania obserwacji y(t+1:n) zaczynając w punkcie i w czasie t.

Algorytm:

- beta[i][n] = 1
- beta[i][t] = [sum_{j=1}^{k} (emission_ppb[j][y_t+1] * beta[j][t+1] * transition_ppb[i][j]]

baum_welch_algorithm(*observations*, *observations_num*, *obs_seq_len*, *laplance_smoothing=0.001*)

Implementacja algorytmu bauma-welcha z użyciem równania Levinsona, dla N niezależnych sekwencji obserwacji. Algorytm służy to reestymacji parametrów ukrytych modeli Markowa

Parametry

- **observations** (*list*) – lista sekwencji obserwacji
- **observations_num** (*int*) – liczba sekwencji obserwacji
- **obs_seq_len** (*int*) – długość każdej z sekwencji obserwacji
- **laplance_smoothing** – minimalne pradopodobieństwo wyrzucenia obserwacji

create_emission_ppb()

Funckcja dla każdej obseracji i każdego stanu tworzy tablicę prawodopodobieństw wyrzucenia obserwacji w danym stanie

Zwraca matrix[state_num][observation_num] - macierz emisji prawdopodobieństw obserwacji

create_initial_ppb(*states_num*)

Funkcja tworzy wektor prawdopodobieństw przejść ze stanu początkowe do każdego z ukrytych stanów

Parametry **states_num** (*int*) – liczba stanów modelu

Zwraca list[states_num]

create_observation_dict(*observations*)

Funkcja tworzy słownik obserwacji. Każdą obserwacje zamienia na string i przypisuje unikatowy numer z przedziału [0, len(observations)-1].

Parametry **observations** (*list*) – lista obserwacji (wektorów cech)

Zwraca słownik obserwacji

create_transition_ppb(*states_num*, *given_transition_ppb*)

Parametry

- **states_num** (*int*) – liczba stanów
- **given_transition_ppb** (*matrix[state_num][2]*) – tablica prawdopodobieństw przejść pomiedzy kolejnymi stanami

Return matrix[states_num][states_num] macierz prawdopodobieństw przejść pomiędzy stanami

evaluate(*obs_sequence*)

Funckcja oblicza prawdopodobieństwo, że dana sekwencja obserwacji została wyprodukowana przez ten model.

Parametry **obs_sequence** (*list*) – lista obserwacji

Zwraca prawdopodobieństwo wygenerowania podanej sekwencji przez ten model

forward_algorithm(*observation_seq*)

Implementacja algorytmu prefiksowego (forward algorithm).

Parametry **observation_seq** (*list*) – sekwencja obserwacji

Zwraca matrix[hidden_states_num][len(observation_seq)], matrix[i][t]

Opis algorytmu: Dane: Y = [y_0, y_1, . . . , y_n] - observation_seq X = [x_1, x_1, . . . , x_k] - ukryte stany markowa

Cel: macierz alfa[[hidden_states_num][n]) taka, że:

alfa[i][t] = P(Y[0] = y_0, Y[1] = y_1, . . . , Y[t] = y_t | X_t = i) - prawdopodobienstwo wygenerowania y(0:t) przy założeniu, że w czasie t byliśmy w stanie i.

Algorytm:

- alfa[i][0] = initial_ppb[i] * emission_ppb[i][y_0]
- alfa[j][t] = [sum_{i=1}^{k} (alfa[i][t-1])*transition_ppb[i][j]] * emission_ppb[j][y_t]

get_parameters()

Funkcja zwraca parametry obiektu

Zwraca

- transiton_ppb
- emission_ppb
- initial_ppb
- observation_dict

learn(*training_set*, *laplance_smoothing=0.001*)

Funkcja trenuje model HMM za pomocą podanego zbioru uczącego

Parametry

- **training_set** (*list*) – zbiór uczący postaci lista seqwencji obserwacji.
- **laplance_smoothing** (*float*) – minimalne prawdopodobieństwo wygenerowania obserwacji przez dany model

Ponieważ obserwacje modelu są typu string, najpierw zamienia każdą obserwację na elementy typu string. Następnie powtarza algorytm Bauma-Welcha na zbiorze uczącym, określoną ilość razy, lub dopóki różnica prawdopodobieństw wygenerowania zbioru uczącego w starym modelu i nowym będzie mniejsze niż epsilon.

print_params()
Funkcja wypisuje parametry modelu HMM

1.3 KNN module

class KNN.**KNN**(*train_set*)
Klasy bazowe: object

Klasa implementująca algorytm K najbliższych sąsiadów dla problemu rozpoznawania emocji z głosu

Dany jest zbiór uczący zawieracjący obserwację (wektory cech), z których każda ma przypisaną emocję jaką dany wektor reprezentuje. Zbiór uczący zostaje znormalizowany a zmienne użyte do normalizacje zapisane jako parametry obiektu.

Dany jest zbiór obserwacji C = (c_1, c_2 . . . c_k}. Celem jest na podstawie informacji z zbioru uczącego przewidzenie jaką emocję reprezentuje dany zbiór obserwacji.

S = [] - zbiór stanów wynikowych

Algorytm predycji: Dla każdej obserwacji c_i :

- c_i zostaje znormalizowane wartościami którymi znormalizowany został zbiór uczący.
- Obliczana jest odległość euklidesowa pomiedczy c_i a każdym wektorem z zbioru uczącego
- Z zbioru uczącego wybierane jest k wektorów, których odległość do c_i jest najmniejsza.
- Sumowane są stany które reprezentują zbiór k wektorów.
- Stany które wystąpiły najczęściej dodawane są do S

Stany które wystąpiły najczęściej w S są zwracane jako możliwe stany reprezentujace dany zbiór obserwacji

compute_emotion(*obs_sequence*, *num_of_nearest_neighbour*)
Funkcja dla każdego wektora z zbioru obserwacji, zlicza prawdopodobne stany jakie reprezentują.

Parametry

- **obs_sequence** (*list*) – lista obserwacji (wektorów) reprezentujących wypowiedź, której stan emocjonalny trzeba rozpoznać
- **num_num_of_nearest_neighbour** (*int*) – liczba najbliższych sąsiadów.

Zwraca stany najczęściej występujące w podanej sekwencji obserwacji.

get_emotion(*test_vector*, *num_of_nearest_neighbour*)
Funkcja porównuje podany wektor emocji z każdym z zbioru trenującego i wybiera k najbliższych.

Parametry

- **test_vector** (*vector*) – wektor, którego stan należy odgadnąć
- **num_num_of_nearest_neighbour** (*int*) – liczba najbliższych sąsiadów, z których należy wziąć stan do porównania.

Zwraca lista stanów których wektory pojawiły sie najczęściej w grupie k najbliższych wektorów.

1.4 database_module module

database_module.**connect_to_database**(*db_name*)
Funckja łączy się z bazą danych jako root

Parametry **db_name** – nazwa bazy danych

:type str :param db_password" hasło do bazy danych :type str

Zwraca

- db, cursor - jeżeli połączenie zostało nawiązane
- None, None - w przeciwnym przypadku

database_module.**is_training_set_exists**(*cursor*, *table*)
Funkcja sprawdza, czy wszystkie tablice podane w table istnieją.

Parametry

- **cursor** – kursor
- **table** (*dictionary*) – tablica, której kluczami są nazwy tablic, których istnienie należy sprawdzić

:return True - jeżeli wszystkie tablice w table.kesy istnieją False - w.p.p.

database_module.**prepare_db_table**(*db*, *cursor*, *table_names*)
Tworzy w bazie danych tablice o podanych nazwach

Parametry

- **db** – obiekt połączenia, który reprezentuje bazę danych
- **cursor** – kursor
- **table_names** (*list*) – lista nazw tablic, do utworzenia

database_module.**save_in_dbtable**(*db*, *cursor*, *vect*, *tbname*)
Funkcja zapisuje w bazie danych w tablicy tbname, wartości z wektora vect

Parametry

- **db** – obiekt połączenia, który reprezentuje bazę danych
- **cursor** – kursor
- **vect** (*vector*) – wektor który ma być zapisany w bazie danych
- **tbname** (*string*) – nazwa tablicy w bazie danych, do ktorej mają być zapisane wartości z wektora

database_module.**select_all_from_db**(*cursor*, *table_name*)
Funckja pobiera z bazy danych z tablicy table_name wszystkie wartości i zwraca w postaci listy wektorów.

Parametry

- **cursor** – kursonr
- **table_name** (*string*) – Nazwa tablicy, z której mają być pobrane dane

1.5 hanning_window module

class hanning_window.**HanningWindow**(*size*)
Klasy bazowe: object

Klasa reprezentuje funckję Hanninga o konkrentej wartości

plot(*signal*)
Funkcja przemnaża podany jako argument sygnał przez okno Hanninga

Parametry **signal** – Fragment sygnału dźwiękowego

Zwraca sygnał będący wynikiem przemnożenia podanego sygnału przez okno Hanninga

1.6 helper_file module

helper_file.**build_file_set**(*path_pattern*)

Funkcja tworzy listę plików znajdujących sie w katalogu path_pattern z rozszerzeniem wav. dla pliku ../anger/file.wav, tworzy parę [../anger/file.wav, anger]

Parametry **pattern** (*basestring*) – Ścieżka do pliku z którego maja być wyodrębione pliki wav

Zwraca list

helper_file.**create_summary_table**(*emotions*)
Funkcja tworzy tablicę podsumowującą i zeruje jej elementy

Parametry **emotions** (*list*) – lista emocji, w summary table

Zwraca dictionary

helper_file.**euclidean_distance**(*vec1*, *vec2*)
Funckja oblicza dystans euklidesowy pomiędzy wektorami

Parametry

- **vec1** – wektor cech
- **vec2** – wektor cech

Zwraca Dystans pomiedzy dwoma wektorami

helper_file.**get_most_frequently_occurring**(*emotions_set*)
Zwraca najczęściej występujacy element w liście

Parametry **emotions_set** (*list*) – lista elementów

Zwraca element który występuje najczęściej

helper_file.**normalize**(*feature_vector_set*)
Normalizuje listę wektórów postaci [data, target], nie naruszając ich kolejności

Param feature_vector_set: Zbiór wektorów cech do znormalizowania

Zwraca

- wektor najmniejszych wartości z każdej cechy
- wektor największych wartości z każdej cechy

helper_file.**normalize_vector**(*feature_vector*, *min_features*, *max_features*)
Normalizuje wektor testowy wartościami podanymi jako argumenty

Parametry

- **feature_vector** – wektor cech do znormalizowania
- **min_features** – wektor najmniejszych wartości z każdej cechy, którymi należy znormalizować podany wektor
- **max_features** – wektor największych wartości z każdej cechy, którymi należy znormalizować podany wektor

helper_file.**print_progress_bar**(*iteration*, *total*, *prefix=''*, *suffix=''*, *decimals=1*, *length=100*, *fill='*')
Funckja rysuje pasek postępu

Parametry

- **iteration** (*int*) – Obecna iteracja
- **total** (*int*) – Liczba wszystkich operacji
- **prefix** (*str*) – Tekst na początku paska postępu
- **suffix** (*str*) – Tekst na końcu paska postępu

helper_file.**print_summary**(*summary_table*, *emotions*)
Funkacja wypisuje summary table dla emocji z listy

Parametry

- **summary_table** (*dictionary*) – Tablica podsumowujaca wyniki
- **emotion** (*list*) – tablica emocji, z summary table, które mają byc wypisane

1.7 histogram module

histogram.**show_subplot_histogram**(*file_set*)
Funkcja wyświetla przebieg częstotliwości bazowych, oraz wartości natężenia dla każdego z plików podanych jako argument

Parametry **file_set** – lista plików

Zwraca None

1.8 hmm_main module

hmm_main.**hmm_claster**(*feature_vector_set*)
Funkcja normalizuje i za pomocą algorytmu K-means klasteryzuje podany zestaw wektorów cech.

Parametry **feature_vector_set** (*list[vector]*) – zbiór wektoróch cech

Zwraca

- lista sklasteryzowancyh wektorów cech
- wektor najmniejszych wartości z każdej cechy
- wektor największych wartości z każdej cechy

hmm_main.**hmm_get_all_possible_observations**(*train_path_pattern*, *db_name*, *emotions*)

Funkcja dla każdego zestawu cech tworzy zbiór wszystkich możliwych wektorów cech, wyliczony z plików w katalogu train_path_pattern, oraz klasteryzuje je w celu uzyskania ograniczonego zbioru obserwacji

Parametry

- **train_path_pattern** (*basestring*) – ścieżka do katalogu z plikami, z których mają być wyliczone obserwacje
- **db** – baza danych do zapisu
- **cursor** – kursor na bazę danych
- **summary_table** – tablica podsumowująca wyniki
- **emotions** (*list*) – lista emocji do wykrycia

Zwraca

- dla każdego zestawu cech lista możliwych obserwacji
- dla każdego zestawu cech wektor najmniejszych i największych wartości każdej z cech

hmm_main.**hmm_get_features_vector_from_dir**(*path_pattern*, *emotions*)

Funkcja dla każdego z dla każdego zbioru cech tworzy wektor cech z plików w katalogu „path_pattern"

Parametry **path_pattern** (*basestring*) – ściażka do katalogu z plikami z których należy wygenerować wektory cech

Zwraca Dla każdego zbioru cech, lista wektorów cech

Typ zwracany dictionary

hmm_main.**hmm_get_features_vectors_from_db**(*cursor*)

Funkcja dla każdego z dla każdego zbioru cech pobiera wektor cech z bazy danych, którą wskazuje cursor

Param path_pattern: kursor na bazę danych

Zwraca Dla każdego zbioru cech, zbiór wektorów cech

Typ zwracany dictionary

hmm_main.**hmm_get_nearest_neighbour**(*vec*, *data*)

Funkcja porównuje dystans pomiędzy wektorem vec a każdym z wektórów „data".

Parametry

- **vec** (*vector*) – wektor cech
- **data** (*list[feature_vector]*) – wszystkie akceptowalne wektory cech

Zwraca Wektor z „data" dla którego dystans do wektora vec jest najmniejszy.

Typ zwracany vector

hmm_main.**hmm_get_observations_vectors**(*file*, *min_max_features_vec*, *all_possible_observations*)

Funkcja dla każdego zestawu cech tworzy zbiór wektorów cech wyliczonych z pliku „file". Każdy wektor normalizuje i przypisuje mu najbliższego sąsiada z wszystkich możliwych obserwacji.

Parametry

- **file** (*basestring*) – ścieżka do pliku z którego mają być pobrane zestawy cech
- **min_max_features** (*dictionary*) – Parametry potrzebne do normalizacji zbioru cech wektorów

- **all_possible_observations** (*vector[vector]*) – Dla każdej cechy wszystkie możliwe w HMM zbiory cech wektorów

Zwraca Słownik zawierający dla każdego zestawu cech listę sekwencję obserwacji wygenerowanych z pliku „file". Każda sekwencja obserwacji jest ok 1,5sek wypowiedzią i składa się z 6 wektorów cech, z których każdy reprezentuje 0,25s wypowiedzi.

Typ zwracany dictionary

hmm_main.**hmm_get_train_set**(*path_pattern*, *min_max_features*, *all_possible_observations*, *emotions*, *summary_table*)

Funkcja dla każdego zestawu cech i każdej emocji tworzy zbiór sekwencji obserwacji do trenowania obiektów HMM.

Parametry

- **path_pattern** (*string*) – Ścieżka do folderu zawierające pliki dźwiękowe, z których mają być wygenerowane dane trenujące
- **min_max_features** (*dictionary*) – Parametry potrzebne do normalizacji zbioru cech wektorów
- **all_possible_observations** (*string*) – Wszystkie możliwe zbiory cech wektorów
- **emotions** (*list*) – Lista emocji

Zwraca Słownik, zwierający dla każdego zestawu cech i każdej emocji, listę sekwencji obserwacji (wektorów cech) z wszystkich plików z katalogu „path_pattern".

Typ zwracany dictionary

hmm_main.**hmm_main**(*train_path_pattern*, *test_path_pattern*, *db_name*, *emotions*)

Główna funkcja hmm. Dla każdego zestawu cech i kazdej emocji tworzy model HMM i trenuje go wektorami obserwacji pobranymi z bazie danych db_name jeżeli istnieją, lub w przeciwnym wypadku obliczonymi z plików znajdujacych sie w katalogu train_path_pattern.

Następnie dla każdej wypowiedzi z katalogu test_path_pattern próbuje przewidzieć jaką emocję reprezentuje ta wypowiedź, w następujący sposób. Dla każdego pliku

1. Dla każdego zestawu cech oblicz listę sekwencji obserwacji
2. **Dla każdej sekwencji obserwacji:**

   2_1) Dla każdej emocji oblicza prawdopodobieństwo wygenerowania sekwencji obserwacji w modelu hmm reprezentującym emocje.

   2_2) Jako prawdopodobną emocję uznaje emocję reprezentującą przez model HMM, który zwrócił największe prawdopodobieństwo wygenerowania tej sekwencji obserwacji.
3. Za emocję reprezentującą ten plik uznaje emocję, która wystąpiła największą ilosć razy

Parametry

- **train_train_path_pattern** – Ścieżka do folderu zawierające pliki dźwiękowe, z których mają być wygenerowane dane trenujące
- **test_path_pattern** (*basestring*) – Ścieżka do folderu zawierające pliki dźwiękowe, z których mają być wygenerowane dane testujące
- **db_name** (*basestring*) – nazwa bazy danych
- **db_password** (*basestring*) – hasło do bazy danych
- **emotions** (*list*) – zbiór emocji do rozpoznania

hmm_main.**hmm_normalize**(*feature_vector_set*)
Funkcja normalizuje podany zbiór wektorów cech

Param feature_vector_set: Zbiór wektorów cech do znormalizowania

Type feature_vector_set: list[vector]

Zwraca

- wektor najmniejszych wartości każdej cechy
- wektor największych wartości każdej cechy

1.9 knn_main module

knn_main.**knn_compute_emotions**(*path_pattern*, *KNN_modules*, *summary_table*, *emotions*)
Funckja dla każdego pliku z path_pattern, pobiera wektory obserwacji, a nastepnie testuje nimi każdy z modelów KNN w celu odganięcia najbardziej prawdopodobnej emocji jaką reprezentuje plik

Parametry

- **path_pattern** (*basestring*) – ściażka do katalogu z plikami z których należy wygenerować wektory cech
- **KNN_modules** (*dictionary*) – Zbiór wytrenowanych obiektów KNN, dla każdej emocji jednej obiekt obiekt KNN
- **summary_table** (*list*) – Pomocnicza tablica do zapisywania wyników testów
- **summary_table** – Lista emocji do przetestowania

knn_main.**knn_get_training_feature_set_from_db**(*cursor*)
Funkcja dla każdego z dla każdego zbioru cech tworzy wektor cech z plików w katalogu „path_pattern”

Parametry **path_pattern** (*basestring*) – ściażka do katalogu z plikami z których należy wygenerować wektory cech

Zwraca Dla każdego zbioru cech lista [wektor cech , emocja]

Typ zwracany dicttionary

knn_main.**knn_get_training_feature_set_from_dir**(*path_pattern*, *emotions*)

Funkcja dla każdego z dla każdego zbioru cech pobiera wektor cech oraz emocję jaką reprezentuje z bazy danych na którą wskazuje cursor

Parametry **path_pattern** – kursor na bazę danych

Zwraca Dla każdego zbioru cech lista [wektor cech , emocja]

Typ zwracany dictionary

knn_main.**knn_main**(*train_path_pattern*, *test_path_pattern*, *db_name*, *emotions*)
Główna funckja knn. Dla każdej emocji tworzy model KNN i trenuje go wektorami obserwacji pobranymi z bazie danych db_name jeżeli istnieją, lub w przeciwnym wypadku obliczonymi z plików znajdujacych sie w katalogu train_path_pattern. Następnie testuje ich działanie wektorami obserwacji obliczonymi z plików znajdujących się w test_path_pattern

Parametry

- **train_path_pattern** (*basestring*) – Ścieżka do folderu zawierające pliki dźwiękowe, z których mają być wygenerowane dane trenujące

- **test_path_pattern** (*basestring*) – Ścieżka do folderu zawierające pliki dźwiękowe, z których mają być wygenerowane dane testujące
- **db_name** (*basestring*) – nazwa bazy danych
- **db_password** (*basestring*) – hasło do bazy danych
- **emotions** (*list*) – zbiór emocji do rozpoznania

1.10 main module

1.11 voice_module module

voice_module.**compute_rms_db**(*time_domain_signal*, *window*)

Funkcja mnoży podany sygnał przez podene okno i oblicza średnią wadratową wartości natężenia tego sygnału

Parametry

- **time_domain_singal** (*list*) – sygnał w domenie czasu
- **window** – objekt reprezentujący funkcję okna.

Zwraca średnia kwadratowa wartości natężenia

Typ zwracany double

voice_module.**get_energy_feature_vector**(*sample*, *window*)

Funkcja na podstawie podanej listy amplitude w domenie czasu oblicza wektor cech dla tych danych

Parametry

- **sample** (*vector*) – lista zmian energii w domenie czasu
- **window** – funkcja okna

Zwraca lista cech na podstawie wprowadzonych danych

Typ zwracany list

voice_module.**get_energy_history**(*file*)

Funkcja otwiera plik podany w ścieżce, oraz oblicza rozkład wartości nateżęnia w czasie w tym pliku.

Parametry **file** (*str*) – Ścieżka do pliku

Zwraca lista zawierająca wartości natężenia w czasie w podanym pliku

Typ zwracany list

voice_module.**get_feature_vectors**(*file*)

Funckja otwiera plik .wav a nastepnie co 0,125ms z pliku odczytuje próbkę dźwięku od długości ok 0,25 ms. Z każdej próbki oblicza wektor cech częstotliwości i energii, tworząc wektory cech.

Parametry **file** (*str*) – ścieżka do pliku z którego mają być wygenerowane wektory cech

Zwraca

- lista wektorów cech częstotliwości
- lista wektorów cech energii
- lista wektorów cech częstotliwości i energii

Typ zwracany dictionary

voice_module.**get_file_info**(*filename*)
Zwraca informacje o podanym pliku

Parametry **filename** (*str*) – Ścieżka do pliku z rozszerzeniem wav

Zwraca parametry pliku

Typ zwracany dictionary

voice_module.**get_freq_history**(*file*)
Funckja otwiera plik wav i dzieli go na kawałki o długości ~0,25s i z każdego fragmentu oblicza wartość tonu podstawowego

Parametry **file** (*str*) – ścieżka do pliku z którego mają być wygenerowane wektory cech

Zwraca

- lista zawierająca wartości tonu podstawego w czasie w podanym pliku

Typ zwracany list

voice_module.**get_fundamental_freq**(*freq_domain_vect*, *sample_rate*, *sample_length*)
Funkcja dla podanej jako argument funkcji w domenie częstotliwości, oblicza wartość tonu podstawowego

Parametry

- **freq_domain_vect** (*vector*) – wektor reprezentujacy funkcję w domenie częstotliwości
- **sample_rate** (*int*) – częstotliwość próbkowania dźwięku z którego pochodzi funkcja
- **sample_length** (*int*) – długosć ramki z której została wygenerowana funkcja

Zwraca wartość tonu podstawowego obliczonego z podanej funkcji

Typ zwracany float

voice_module.**get_fundamental_freq_form_time_domain**(*sample*, *frame_length*, *window*, *frame_rate*)
Funkcja dla każdej rammki z sample, dłguośći frame_length przygotowuje fft i oblicza z tego częstotliwość bazową. Następnie tworzy listę częstotliwości bazowych.

Parametry

- **sample** – lista sampli, z ktorych ma być obliczony wetor cech częstotliwości
- **frame_length** – Dłguosć ramki do fft
- **window** – Funkcja okna
- **frame_rate** – Czestotliwość samplowania

Zwraca Lista czestotliwości bazowych

Typ zwracany list

voice_module.**get_pitch_features**(*fundamental_freq_array*)
Funkcja na podstawie podanej listy wartości tonów podstawowych oblicza wektor cech dla tych danych

Parametry **fundamental_freq_array** (*vector*) – wektor częstotliowści bazowych

Zwraca lista cech obliczonych na podstawie podanej jako argument funckji

Typ zwracany list

voice_module.**get_sample_rate**(*filename*)

Parametry **filename** (*str*) – Ścieżka do pliku z rozszerzeniem wav

Zwraca częstotliwość samplowania

Typ zwracany int

voice_module.**get_summary_pitch_feature_vector**(*pitch_feature_vectors*)
Funkcja na podstawie danych oblicza wektor cech częstotliwośći bazowych

Parametry **pitch_feature_vectors** – lista wektorów cech częstotliowśći bazowych

Zwraca wektor cech

Typ zwracany list

voice_module.**read_from_wav_file**(*wav_file*, *length*)
Funkcja odczytuje z pliku wav określoną ilość próbek pochodzących z jednego kanału.

Parametry

- **wav_file** – wskaźnik na plik wav
- **length** (*int*) – liczba sampli jaką chcemy odczytać

Zwraca lista sampli długości length, pochodzących z jednego kanału.

Typ zwracany list

Opis:

Sample w plikach wav z więcej niż jednym kanałem są ustawione naprzemiennie. Najpierw jest pierszy sampel kanału 1, następnie pierwszy sampel kanału 2 itd. dopiero później są sample dla kanału 2.

Aby więc odczytać informację o długości n należy odczytać długość sampla * liczbę kanałów * rządana długość, a następnie z odczytanych danych wziąć elemnty z określonego kanału. Na przykład dla trzek kanałów należy wziać co trzeci element.

ROZDZIAŁ 2

Indices and tables

- genindex
- modindex
- search

Indeks modułów pythona