Speech emotion recognition Documentation

Wydanie 1.0

Elżbieta Plaszczyk

15 sty 2018

Contents:

ROZDZIAŁ 1

speech_emotion_recognition

1.1 HMM module

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

Klasy bazowe: `object`

Klasa implementująca algorytm Ukryte Modle Markova 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 przejsć z stanu początkowego dostanu każdego z ukrytych stanów.

__init__(*transition_ppb*, *states_num*, *observations*)

Konstruktor klasy HMM.

Tworzy tablicę przejść pomiędzy stanami, transition_ppb. Dla dowolnych 2 stanów s_i i s_j, prawdopodobnieństwo przejścia z stanu s_i do stanu s_j: p(s_i, s_j) jest równe:

- transition_ppb[i][0] jeżeli i == j

- transition_ppb[i][1] jeżeli i == j-1
- 0 w przeciwnym przypadku

Tworzy tablicę emisji: emission_ppb. Na początku dla każdego stanu S i każdej obserwacji O prawdopodobieństwo przejścia emisji O w stanie S jest równe.

Tworzy tablicę: initial_ppb. Na początku dla każdego stanu S prawdopodobieństwo przejścia ze stanu początkowego do stanu S jest równe.

Parametry

- **transition_ppb** (*matrix[states_num][2]*) – macierz prawdopodobieństw przejść pomiędzy stanami.
  - matrix[i][0] - zawiera pradopodobieńśwo przejśćia z stanu i do stanu i,
  - matrix[i][1] - zawiera pradopodobieńśwo przejśćia z stanu i do stanu (i+1)
- **states_num** (*int*) – liczba ukrytych stanow
- **observations** (*list*) – lista wszystkich możliwych obserwacji

backward_algorithm(*ob_sequence*)

Implementacja algorytmu sufiksowego (backward algorithm)

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

:return 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

Return 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

:return 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)

:return 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.

Param list obs_sequence: lista obserwacji

forward_algorithm(*observation_seq*)
Implementacja algorytmu prefiksowego (forward algorithm).

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

:return 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

:return

- 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

1.2 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

__init__(*train_set*)
Konstruktor klasy. Normalizuje i zapisuje zbiór uczący

Parametry **train_set** (*list*) – zbiór uczący dany model KNN -> lista wektorów postaci [wektor_cech, emocja jaką reprezentuje]

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.

:return 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.

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

static normalize(*feature_vector_set*)
Normalizuje zbiór listę wektórów postaci [wektor_cecg, emocja]

Param feature_vector_set: Zbiór wektorów cech do znormalizowania

Zwraca

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

1.3 hanning_window module

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

__init__(*size*)

plot(*signal*)

1.4 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.**connect_to_database**(*db_name*, *db_password*)
Funckja łączy się z bazą danych jako root :param db_name nazwa bazy danych :type str :param db_password hasło do bazy danych :type str :return

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

helper_file.**create_summary_table**(*emotions_list*)

helper_file.**euclidean_distance**(*vec1*, *vec2*)
Funckja oblicza dystans euklidesowy pomiędzy wektorami :param vec1: wektor cech :param vec2: wektor cech :return: Dystans pomiedzy dwoma wektorami

helper_file.**get_most_frequently_occurring**(*emotions_set*)
Zwraca najczęściej występujacy element w liście :param emotions_set: lista elementów :type emotions_set: list :return element który występuje najczęściej

helper_file.**normalize_vector**(*feature_vector*, *min_features*, *max_features*)
Normalizuje wektor testowy wartościami podanymi jako argumenty :param feature_vector: wektor cech do znormalizowania :param min_features: wektor najmniejszych wartości z każdej cechy, którymi należy znormalizować podany wektor :param max_features: wektor największych wartości z każdej cechy, którymi należy znormalizować podany wektor

helper_file.**print_debug**(*text*)

helper_file.**print_progress_bar**(*iteration*, *total*, *prefix=''*, *suffix=''*, *decimals=1*, *length=100*, *fill='*')
FUnckja rysuje pasek postępu :param iteration: Obecna iteracja :type iteration: int :param total: Liczba wszystkich operacji :type total: int :param prefix: Tekst na początku paska postępu :type prefix: str :param suffix: Tekst na końcu paska postępu :type suffix: str

helper_file.**print_summary**(*summary_table*, *emotions_list*)
function illustrating how to document python source code

1.5 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*, *db_password*)
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*)
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*, *db_password*, *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

1.6 knn_database module

knn_database.**is_training_set_exists**(*cursor*, *table*)

knn_database.**prepare_db_table**(*db*, *cursor*, *table*)

knn_database.**save_in_dbtable**(*db*, *cursor*, *vect*, *tbname*)

knn_database.**select_all_from_db**(*cursor*, *table_name*)

1.7 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*)
Funkcja dla każdego z dla każdego zbioru cech pobiera wektor cech oraz emocję jaką reprezentuje z bazy danych na którą wskazuje cursor

param path_pattern: kursor na bazę danych

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

rtype dictionary

knn_main.**knn_main**(*train_path_pattern*, *test_path_pattern*, *db_name*, *db_password*, *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

1.8 main module

main.**draw_energy_histogram**()

main.**draw_freq_histogram**()

1.9 voice_module module

voice_module.**get_energy_feature_vector**(*sample*, *window*)
Funkcja na podstawie podanej listy amplitude w domenie czasu oblicza wektor cech dla tych danych :param vector sample: lista zmian energii w domenie czasu :param window: funkcja okna :return: lista cech na podstawie wprowadzonych danych

voice_module.**get_feature_vectors**(*file*)
Funckja otwiera plik wav i dzieli go na kawałki o długości ~0,25s, biorąc sample co ~0,125s, czyli kawałki nachodzą na siebie - w celu zwiększenia liczby obserwacji. Dla każdego kawałka wypowiedzi oblicza na podstawie niego wektor cech częstotliwości i energii.

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

:return

- lista wektorów cech częstotliwości
- lista wektorów cech energii

voice_module.**get_file_info**(*filename*)

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

Zwraca parametry pliku

voice_module.**get_fundamental_freq**(*freq_domain_vect*, *sample_rate*, *sample_length*)
Funkcja oblicza częśtotliwość bazową dla danego funkcji w domenie częsttliwości

Parametry

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

Return float częstotliwość bazowa dla podanej funkcji

voice_module.**get_pitch_feature_vector**(*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 i oblicza wektor cech na podsawie tych danych.

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 Wektor cech częstliwośći dla podanego zbioru sampli

voice_module.**get_pitch_features**(*fundamental_freq_array*)
Funkcja na podstawie podanej listy częstotliwość bazowych oblicza wektor cech dla tych danych :param vector fundamental_freq_array: wektor częstotliowści bazowych :return: lista cech wektora

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

Parametry **filename** (*str*) – ścieżka do pliku

Zwraca częstotliwość samplowania

voice_module.**get_summary_pitch_feature_vector**(*pitch_feature_vectors*)
Funkcja na podstawie danych oblicza wektor cech częstotliwośći bazowych :param pitch_feature_vectors: lista wektorów cech częstotliowśći bazowych :return: wektor cech

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

:param wav_file wskaźnik na plik wav :param int length: liczba sampli jaką chcemy odczytać

:return [list] - lista sampli długości length, pochądzących z jednego channella.

Ponieważ rozmary sampli w pliku .wav mają różną długość należy odczytywać 1 próbkę należy odczytać określoną ilość bitów. DO tego służy tablica ftm_size

Sample w pliku wav są umieszczone następująco: s1_c1, s1_c2, s2_c1, s2_c2, gdzie s1 oznacza sample_1, a c1 channel_1.

Aby więc odczytać informację o długości n należy odczytać (fmt_size * length * wav_file.getnchannels()) bitów, a następnie wziąć co x-ty element x-ty element, gdzie x to liczba channeli.

ROZDZIAŁ 2

Indices and tables

- genindex
- modindex
- search

Indeks modułów pythona

h

k

m

p

v

Indeks

Symbols

B

C

D

E

F

G

H

I

K

L

M

N

P

R

S

V