Politechnika Krakowska
im. Tadeusza Kościuszki
Wydział Informatyki i Telekomunikacji

Piotr Rusin

Numer albumu: 115481

Biblioteka do tworzenia czatbotów w języku Python

Library for creating chatbots in Python language

Praca inżynierska
na kierunku INFORMATYKA

Praca wykonana pod kierunkiem:
dr Radosław Kycia

Streszczenie

W pracy opisano proces utworzenia biblioteki do tworzenia chatbotów w języku Python. Porównane zostały dwa popularne rozwiązania open-source Will i Chatterbot. Wynikowa

Kraków, 2023

umożliwia przetwarzanie zapytań użytkownika na podstawie wyrażeń regularnych oraz przez obliczenie podobieństw do predefiniowanego korpusu pytań i odpowiedzi. Wspierane jest również wyszukiwanie semantyczne słów-kluczy w lokalnym korpusie stron z portalu Wikipedia. Biblioteka w obliczeniach stopnia podobieństw dokumentów do korpusu opiera się na wektoryzacji TF-IDF i podobieństwie cosinusowym. Do wyszukiwania semantycznego wykorzystana została metoda PCA.

Abstract

The work describes the process of creating a library for chatbots in Python. Two popular open-source solutions Will and Chatterbot were compared. The resulting library is an intersection of the functionality of both solutions which allows processing user queries based on regular expressions and by calculating similarities to a predefined corpus of questions and answers. Semantic keyword search in the local Wikipedia corpus is also supported. The library is using TF-IDF vectorization, and cosine similarity for calculation of similarities. PCA method was used for semantic search.

Spis treści

Rozdział 1

1. Wstęp

1.1. Cel pracy

Tworzenie chatbotów przy pomocy istniejących rozwiązań open-source jest czasochłonne. Biblioteki w różny sposób podchodzą do przetwarzania języka naturalnego, często skupiając się na jego pojedynczych aspektach, a pomijając pozostałe. W ekosystemie brakuje uniwersalnego rozwiązania, które połączyłoby wspólne, pozytywne aspekty istniejących bibliotek, umożliwiając jednocześnie łatwe rozszerzenie ich o dodatkowe funkcjonalności. Autorzy istniejących bibliotek decydują się na architekturę, która umożliwia rozszerzalność, jednak realizuje to w sposób, który prowadzi do mało czytelnego wynikowego kodu projektu.

Celem pracy jest stworzenie biblioteki do chatbotów zapewniającej funkcjonalności z istniejących rozwiązań open-source takich jak wyszukiwanie poprzez wyrażenia regularne, jak i wyszukiwanie poprzez podobieństwo, jednocześnie ułatwiając przy tym rozszerzalność aplikacji o dodatkowe funkcjonalności z dziedziny przetwarzania języka naturalnego.

1.2. Zakres pracy

W zakres pracy wchodzi porównanie istniejących rozwiązań open-source wraz z wyciągnięciem wniosków o sposobie ich działania. Następnie przedstawiony zostanie projekt biblioteki, która umożliwia niezależne odwzorowanie funkcjonalności z tych rozwiązań w ustruktyryzowany sposób, wspierając przy tym rozszerzalność i zapewniając elastyczność implementacji. Przedstawiono też przykładową implementację biblioteki w projekcie. Zweryfikowane zostanie, w jaki sposób struktura biblioteki umożliwia rozszerzanie chatbota o dodatkowe funkcjonalności oraz zbadana zostanie efektywność stworzonego chatbota.

1.3. Metodyka pracy

Proces tworzenia biblioteki został podzielony na trzy części. Na początku przedstawiona została analiza istniejących rozwiązań open-source do tworzenia chatbotów *Will* oraz *Chatterbot*. Opierając się na pomysłach z tych bibliotek, została od podstaw zaprojektowana nowa biblioteka. Biblioteka ta została zaimplementowana w przykładowych chatbotach demonstrujących jej działanie. W dalszej części porównano możliwości biblioteki względem rozwiązań *Will* i *Chatterbot* oraz wyciągnięte zostały wnioski.

Część I

Część teoretyczna

Rozdział 2

2. Podstawy teoretyczne

2.1. Prawo Zipfa

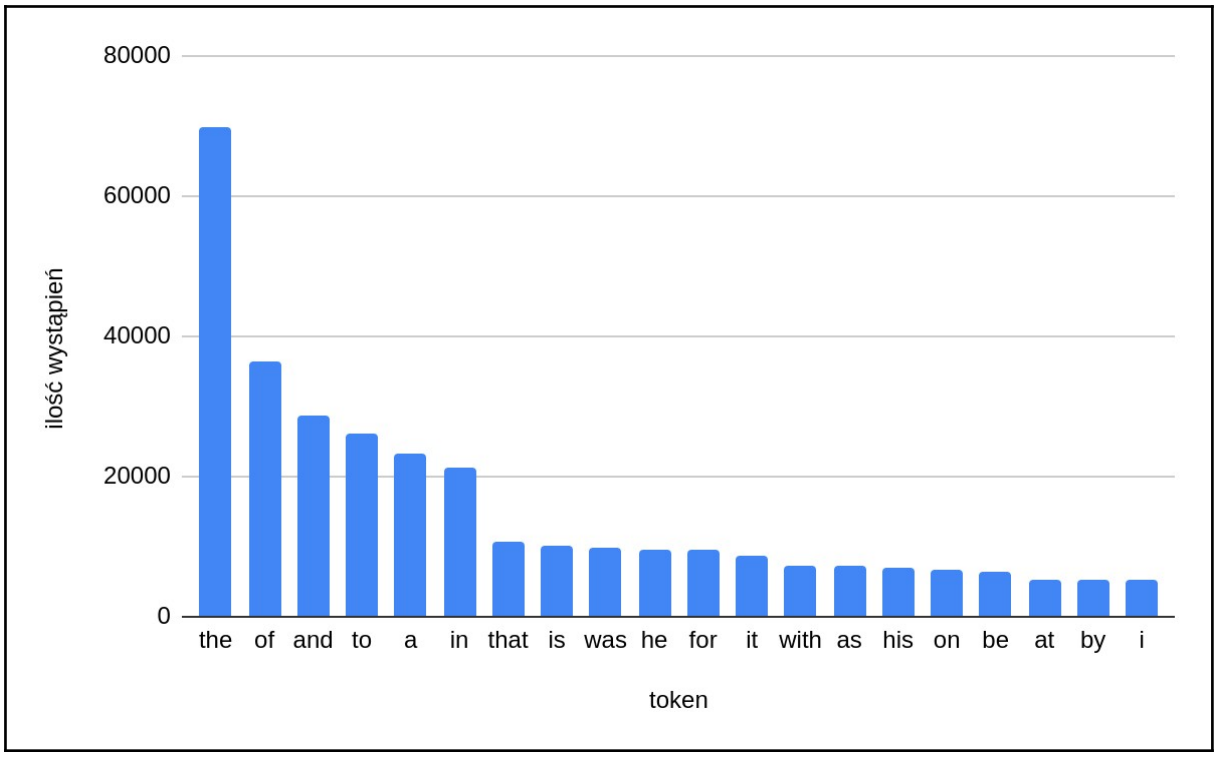
Zależność zaobserwowana przez George'a Kingsleya Zipfa, która mówi o tym, że częstotliwość słów i innych jednostek lingwistycznych ma tendencję do rozkładu skośnego zawierającego dużą liczbę rzadkich zdarzeń [1]. Prawo Zipfa sformułowane dla rozkładu częstotliwości danego słowa twierdzi, że jeżeli słowa zostaną ułożone według częstotliwości i zostanie im przydzielona ranga z to:

$$f_z(z, N) = 1/z^a, \tag{2.1.1}$$

gdzie $f_z(z, N)$ jest częstotliwością wystąpień rangi z dla próbki o rozmiarze N, a a to wykładnik bliski 1.

Oznacza to, że w przybliżeniu drugie słowo występuje w tekście dwa razy rzadziej (½), trzecie trzy razy rzadziej (⅓), itd.

Dla pierwszego elektronicznego korpusu "NLTK Brown Corpus" [2] utworzonego na Uniwersytecie Browna i zawierającego teksty z 500 źródeł podzielonych na kategorie, częstość słów po wykluczeniu znaków interpunkcyjnych i znaków specjalnych to [3]: the - 69971, of - 36412, and - 28853, to - 26158, a - 23195, in - 21337, that - 10594, is 10109, was - 9815, he - 9548, for - 9489, it - 8760, with - 7289, as - 7253, his - 6996, on - 6741, be - 6377, at - 5372, by - 5306, i - 5164



Rys. 1. Prawo Zipfa na przykładzie korpusu "Brown Corpus" Uniwersytetu Browna

Wykres (rys. 1.) obrazuje zależność prawa Zipfa. Każde kolejne słowo pojawia się w przybliżeniu dwa razy rzadziej od poprzedniego: "the" - 69971 razy, "of" - 36412 razy itd. Zależność ta jest wykorzystywana w przetwarzaniu języka naturalnego do zbudowania wektorów częstości tokenów (TF) oraz odwrotnej częstości w dokumentach (IDF).

2.2. Tokenizacja, tokeny i tokenizatory

Tokenizacja (ang. tokenization) oznacza proces przekształcenia zdań zawartych w dokumencie na niepodzielne elementy nazywane tokenami [4] i jest jednym z początkowych kroków w potoku przetwarzania języka naturalnego. Tokenem może być wyraz, znak, lub n-gram, który jest ciągiem złożonym z n elementów (np. "ice cream" to 2-gram, a "Johann Sebastian Bach" to 3-gram) [3]. W procesie tokenizacji wykorzystywane są narzędzia nazywane tokenizatorami.

Biblioteka do tokenizacji pytań i odpowiedzi wykorzystuje tokenizator *nltk.tokenize.casual_tokenize* [5].

2.3. Stop lista

Stop lista (ang. stop list) [6] jest zbiorem tokenów które zostaną wykluczone podczas procesu tokenizacji. W przetwarzaniu języka naturalnego są to tokeny które występują często ("a", "the", "is", itp), oraz takie które niewiele wnoszą do znaczenia tekstu [3].

Biblioteka wykorzystuje stoplistę *nltk.corpus.stopwords.words('english')*. [7]

2.4. Częstotliwość tokenów (TF)

Częstotliwość tokenów (ang. term frequency) jest to stosunek liczby wystąpień danego tokenu do liczby wszystkich tokenów [3]:

$$TF(t) = n_t / n, \tag{2.4.1}$$

gdzie: t to token, n_t to liczba wystąpień tokenu w dokumencie, a n to liczba wszystkich tokenów.

2.5. Odwrotna częstotliwość wystąpienia tokenu w dokumentach (IDF)

Odwrotna częstotliwość wystąpienia tokenu w dokumentach (ang. inverse document frequency) jest logarytmem ze stosunku ogólnej liczby dokumentów do liczby dokumentów zawierających token [3]:

$$IDF(t) = \log(n / df(t)), \tag{2.5.1}$$

gdzie: t to token, n to liczba wszystkich dokumentów, a $df(t)$ jest liczbą dokumentów zawierających token.

Odwrotna częstość w dokumentach pokazuje, w jakim stopniu unikalna jest obecność danego tokenu w danym dokumencie. Jeżeli token pojawia się kilkukrotnie w obrębie jednego dokumentu, a rzadko w obrębie całego korpusu, to wyrażenie jest znaczące dla danego dokumentu.

2.6. TF-IDF

TF-IDF czyli częstość tokenów - odwrotna częstość w dokumentach (ang. term frequency - inverse document frequency) [3] jest iloczynem częstotliwości tokenów i odwrotnej częstotliwości tokenu w dokumentach:

$$TF-IDF(t) = TF(t) * IDF(t), \tag{2.6.1}$$

gdzie t oznacza token.

TF rośnie wraz we wzrostem częstotliwości konkretnych słów w dokumencie (TF-IDF również wtedy rośnie). IDF maleje wraz ze wzrostem liczby dokumentów zawierających token (TF-IDF również wtedy maleje).

Biblioteka do wektoryzacji TF-IDF wykorzystuje *sklearn.feature_extraction.text.TfidfVectorizer [8]*.

2.7. Implementacja TF-IDF w bibliotece sklearn

W dalszej części pracy do obliczenia wektorów TF-IDF została wykorzystana biblioteka sklearn i jej pomocnicza klasa *sklearn.feature_extraction.text.TfidfVectorizer [8]*. Implementacja obliczeń wektorów TF-IDF w tej klasie różni się od teoretycznej.

Biblioteka scikit-learn jako częstotliwość tokenu traktuje ilość wystąpień danego tokenu w dokumencie, a nie współczynnik ilości wystąpień tokenu w dokumencie do łącznej ilości tokenów [9]:

$$TF(t) = n_t, \tag{2.7.1}$$

gdzie: n_t to liczba wystąpień tokenu w dokumencie.

Wektor IDF może zostać obliczony według poniższych wzorów, w zależności od przekazanego parametru *smooth_idf*. Gdy parametr zostanie ustawiony na *smooth_idf = True* (domyślna wartość), wzór przybiera postać [9]:

$$IDF(t) = \log[(1+n)/(1+df(t))] + 1, \tag{2.7.2}$$

gdzie: t to token, n oznacza liczbę wszystkich dokumentów, a $df(t)$ to liczba dokumentów zawierających token.

Gdy *smooth_idf = False* [11]*:*

$$IDF(t) = \log[n/df(t)] + 1, \tag{2.7.3}$$

gdzie: t to token, n jest liczbą wszystkich dokumentów, a $df(t)$ liczbą dokumentów zawierających token.

Wektoryzer TF-IDF sklearn wspiera również normalizację wektora TF w zależności od parametru *norm*. Gdy parametr zostanie ustawiony na *norm = 'l2'* (domyślna wartość) to obliczana jest norma Euklidesowa [9] [10] każdego wiersza z macierzy:

$$¿ \vee X \vee ¿_2 = \sqrt{\square}, \tag{2.7.4}$$

a następnie macierz została znormalizowana poprzez podzielenie każdej składowej poprzez normę Euklidesową wiersza $¿ \vee X \vee ¿_2$ w której się znajduje:

$$Y_{n,m} = X_{n,m} / (¿ \vee X_1 \vee ¿_2 ... \vee ¿ X_n \vee ¿_2) \tag{2.7.5}$$

gdzie *n* to liczba wierszy, *m* to liczba kolumn, $X_{n,m}$ jest macierzą *n x m*, a ¿∨X_n∨¿$_2$ jest normą Euklidesową n-tego wiersza.

Dla macierzy X:

$$\mathrm{X}_{n,m} = \begin{pmatrix} x_{11} & \cdots & x_{1m} \\ \cdots & \cdots & \cdots \\ x_{n1} & \cdots & x_{nm} \end{pmatrix} \tag{2.7.6}$$

scikit-learn oblicza sumę z wiersza wykorzystując notację sumacyjną Einsteina, w rezultacie powstaje wektor z sumami składowych wierszy:

$$\begin{pmatrix} \sum_{j=1}^{m} a_{1j}^2 & \cdots & \sum_{j=1}^{m} a_{nj}^2 \end{pmatrix} \tag{2.7.7}$$

składowe wektoru są następnie pierwiastkowane aby obliczyć normy Euklidesowe każdego wiersza:

$$\begin{pmatrix} \sqrt{\sum_{j=1}^{m} a_{1j}^2} & \cdots & \sqrt{\sum_{j=1}^{m} a_{nj}^2} \end{pmatrix} = \begin{pmatrix} ||X_1||_2 & \cdots & ||X_n||_2 \end{pmatrix} \tag{2.7.8}$$

następnie wektor jest transponowany i obliczana jest znormalizowana macierz Y poprzez podzielenie macierzy X przez transponowany wektor norm wierszy z macierzy X co daje wynikową, znormalizowaną macierz Y:

$$\mathrm{Y}_{n,m} = \begin{pmatrix} \frac{x_{11}}{||X_1||_2} & \cdots & \frac{x_{1m}}{||X_1||_2} \\ \cdots & \cdots & \cdots \\ \frac{x_{n1}}{||X_n||_2} & \cdots & \frac{x_{nm}}{||X_n||_2} \end{pmatrix} \tag{2.7.9}$$

Gdy przekazany zostanie parametr *norm = 'l1'* normalizacja opiera się na normie pierwszej wierszy:

$$\mathrm{Y}_{n,m} = \begin{pmatrix} \frac{x_{11}}{\sum_{j=1}^{m} |a_{1j}|} & \cdots & \frac{x_{1m}}{\sum_{j=1}^{m} |a_{1j}|} \\ \cdots & \cdots & \cdots \\ \frac{x_{n1}}{\sum_{j=1}^{m} |a_{nj}|} & \cdots & \frac{x_{nm}}{\sum_{j=1}^{m} |a_{nj}|} \end{pmatrix} \tag{2.7.10}$$

W przypadku *norm = None* normalizacja zostaje pominięta.

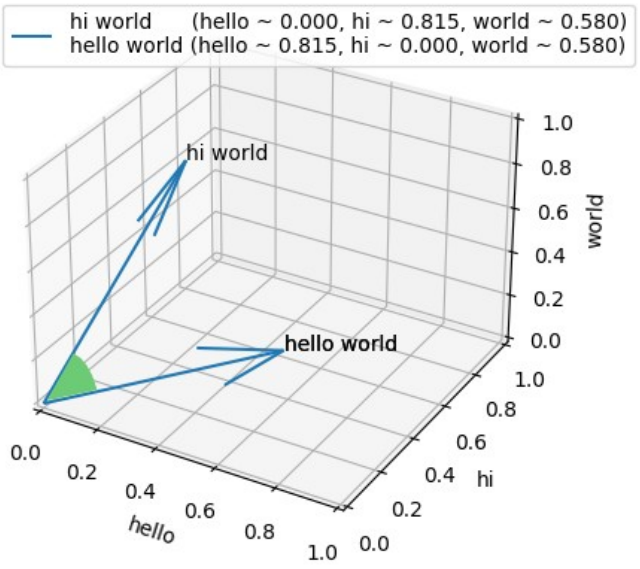
Biblioteka do normalizacji wykorzystuje normy Euklidesowe wierszy (2.7.9), a do obliczeń IDF wariant (2.7.2) aby uniknąć błędów dzielenia przez 0. Wektor TF-IDF niezależnie od wybranego sposobu obliczeń IDF zawsze jest obliczany według (2.6.1).

2.8. Podobieństwo cosinusowe

Podobieństwo cosinusowe jest miarą podobieństwa pomiędzy dwoma wektorami [12]. Podobieństwo określa się jako iloraz iloczynu skalarnego wektorów przez iloczyn ich długości:

$$cossimilarity = \cos(\theta) = \frac{A \cdot B}{¿ \vee A \vee ¿ \vee ¿ B \vee ¿} \tag{2.8.1}$$

Podobieństwo cosinusowe jest zależne od kąta pomiędzy wektorami i jest niezależne od długości wektorów [11].



Rys. 2. Reprezentacja wektorowa słów

Dokument \ Token	**hello**	**hi**	**world**	**cos similarity ("hi world")**	**cos similarity ("hello world")**
hi world	0	0.815	0.580	1	0.3361
hello world	0.815	0	0.580	0.3361	1

Tabela 1. Wektor TF-IDF i podobieństwo cosinusowe dokumentów "hi world", "hello world"

2.9. Wyrażenie regularne

Wyrażenie regularne są sposobem definiowania wzorców wykorzystywanych do wyszukiwania i przetwarzania tekstu [12] i opierają się na znakach specjalnych do budowania wzorców.

Biblioteka wykorzystuje wbudowane wsparcie wyrażeń regularnych w języku Python dostarczone przez moduł *re* [13].

2.10. SVD

SVD (ang. Singular Value Decomposition) jest rozkładem macierzy według wartości osobliwych [14] na iloczyn trzech macierzy.

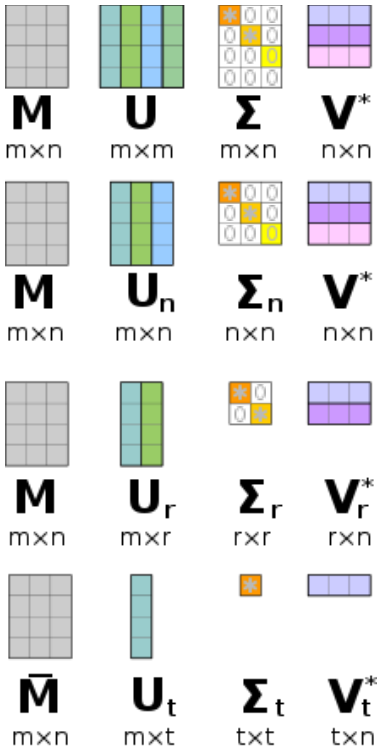
SVD macierzy M o wymiarach m x n to [15]:

$$M = U\Sigma V^T, \tag{2.10.1}$$

gdzie U i V są macierzami ortogonalnymi, kolumny U są ortonormalnymi wektorami własnymi MM^T, kolumny V są ortonormalnymi wektorami własnymi $M^T M$, a Σ jest macierzą m x n o przekątej z elementami będącymi liczbami rzeczywistymi σ_i, uporządkowanymi w taki sposób, że:

$$\sigma_1 \geq \sigma_2 \geq \ldots \geq \sigma_{min(m,n)?} \geq 0 \ , \tag{2.10.2}$$

gdzie σ_i to wartości osobliwe macierzy M.



Rys. 3. Wizualizacja operacji wykonanych na macierzach [16]

2.11. PCA

PCA (ang. Principal Component Analysis) to statystyczna metoda analizy głównych składowych, której celem jest redukcja wymiarowości. W przetwarzaniu języka naturalnego taka redukcja pozwala na ujawnienie ukrytych semantycznych cech korpusu [3]. PCA jest w bezpośredniej relacji z SVD w przypadku gdy główne czynniki są obliczane na podstawie macierzy kowariancji [19].

Biblioteka wykorzystuje sklearn.decomposition.PCA [18] do analizy głównych składowych i wyszukiwania semantycznego w częściowym korpusie z Wikipedii.

2.12. Implementacja PCA w bibliotece scikit-learn

W scikit-learn algorytm PCA został zaimplementowany jako obiekt transformatora, który uczy się komponentów w swojej metodzie dopasowania i może być używany na nowych danych do rzutowania ich na te komponenty [18].

Biblioteks scikit-learn w celu projekcji macierzy do przestrzeni o niższym wymiarze wykorzystuje liniową redukcję wymiarowości macierzy poprzez rozkład według własności osobliwych [18] [19].

W zależności od kształtu danych wejściowych i liczby składników do wyodrębnienia biblioteka wykorzystuje implementację LAPACK pełnego SVD [20] lub losowo przyciętego SVD metodą Halko et al. 2009 [21]. Możliwym jest również wykorzystanie przyciętego SVD z scipy.sparse.linalg ARPACK [22].

Część II

Część praktyczna

Rozdział 3

3. Implementacja

3.1. Analiza problemu

Postawionym problemem jest zaprojektowanie biblioteki do chatbotów zapewniającej funkcjonalności z istniejących rozwiązań open-source takich jak wyszukiwanie poprzez wyrażenia regularne i wyszukiwanie poprzez podobieństwo. Aby tego dokonać porównane zostaną dwie istniejące biblioteki do tworzenia chatbotów Will [23] i Chatterbot [24] które opierają się na tych funkcjonalnościach. Na tej podstawie zostanie zaplanowana struktura biblioteki z uwzględnieniem możliwości, które daje język programowania Python. Struktura biblioteki zostanie następnie rozpisana i przetestowana w praktyce przez przykładowego chatbota.

3.2. Porównanie istniejących rozwiązań open-source

Omówione zostały dwa rozwiązania open-source do tworzenia chatbotów - *Will* [23] i *Chatterbot [24]*. Biblioteki podchodzą do przetwarzania języka naturalnego w różny sposób.

3.2.1. Will

Will jest biblioteką do tworzenia chatbotów, która opiera się na wyrażeniach regularnych i wykorzystuje mechanizm dekoratorów [25] z języka Python do określenia wyrażeń regularnych, które po udanym sparowaniu z wiadomością wejściową uruchamiają powiązaną funkcję. W bibliotece Will zaimplementowane zostało wsparcie dla komunikatorów Slack, HipChat, Rocket.Chat oraz konsoli [26]. Chatbot komunikuje się z zewnętrznymi aplikacjami poprzez API.

Wspieranymi dekoratorami są respond_to, hear, periodic, route, i randomly [27]. Nazewnictwo dekorowanych funkcji może pozostać dowolne, jednak dekorowana funkcja musi być zawarta w klasie dziedziczącej po klasie WillPlugin:

```
from will.plugin import WillPlugin
```

3.2.1.1. Dekorator @respond_to

Dekorator *respond_to* [27] umożliwia zaprogramowanie reakcji bota na konkretną wiadomość. Warunkiem koniecznym jest oznaczenie chatbota w wiadomości przez tag (np. *@will hello*).

```
from will.decorators import respond_to
```

```
class HelloPlugin(WillPlugin):
  @respond_to("hello")
  def say_hello(self, message):
    self.reply("hello!")
```

Argumentami dekoratora *respond_to* są:

argument	**opis**
regex	wyrażenie regularne na podstawie którego wywołana zostanie dekorowana funkcja
include_me	zmienna określająca czy chatbot powinien też reagować na wiadomości napisane przez samego siebie
case_sensitive	zmienna określająca czy wyszukiwanie ma odbywać się bez rozróżnienia wielkości liter z wyrażenia regularnego
multiline	zmienna określająca czy chatbot ma wyszukiwać wyrażenie regularne w wielu liniach
admin_only	zmienna określająca czy chatbot ma reagować tylko na wiadomości od administratora
acl	lista uprawnień ACL [28]

Tabela 2. Argumenty dekoratora respond_to z biblioteki Will

3.2.1.2. Dekorator @hear

Dekorator *hear* [27] umożliwia nasłuchiwanie wiadomości. Dekorator nie wymaga oznaczenia bota i porównuje wszystkie otrzymane wiadomości ze zdefiniowanym wyrażeniem regularnym.

```
from will.decorators import hear
class Plugin(WillPlugin):
  @hear("(?:ran into )?a bug")
  def found_a_bug(self, message):
    self.reply("found a bug")
```

Argumenty dekoratora *hear* pokrywają się z argumentami dektoratora *respond_to*.

3.2.1.3. Dekorator @periodic

Dekorator *periodic* [27] określa przedział dni tygodnia oraz godzinę o której wykonana zostanie powiązana funkcja.

```
from will.decorators import periodic
class Plugin(WillPlugin):
  @periodic(hour='10', minute='0', day_of_week="mon-fri")
  def reminder(self, message):
    self.say("@all meeting reminder")
```

Argumentami dekoratora *periodic* są:

argument	**opis**
year	rok w którym ma zostać uruchomione cykliczne powiadomienie (czterocyfrowa liczba)
month	miesiąc w którym ma zostać uruchomione cykliczne powiadomienie (liczba z zakresu 1 - 12)
day	dzień w którym ma zostać uruchomione cykliczne powiadomienie (liczba z zakresu 1 - 31)
week	numer tygodnia w którym ma zostać uruchomione cykliczne powiadomienie (liczba z zakresu 1 - 53)
day_of_week	dzień tygodnia w którym ma zostać uruchomione cykliczne powiadomienie (liczba z zakresu 0 - 6 lub odpowiednik tekstowy mon, tue, wed, thu, fri, sat, sun)
hour	godzina w której ma zostać uruchomione cykliczne powiadomienie (liczba z zakresu 0 - 23)

Tabela 3. Argumenty dekoratora periodic z biblioteki Will

3.2.1.4. Dekorator @route

Dekorator *route* [27] jest dekoratorem który umożliwia deklarację funkcji generującej odpowiedź na zapytanie HTTP które kierowane jest do wbudowanego serwera biblioteki:

```
from will.decorators import route
class Plugin(WillPlugin):
  @route("/complex_page/<page_id:int>", method="POST")
  def complex_page(self, page_id):
    return "requested page id: {}".format(page_id)
```

3.2.1.5. Dekorator @randomly

Dekorator *randomly* [26] wykonuje dekorowaną funkcję w losowym momencie z określonego przedziału czasowego. Ilość wykonań funkcji w danym przedziale może zostać ograniczona przez argument *num_times_per_day:*

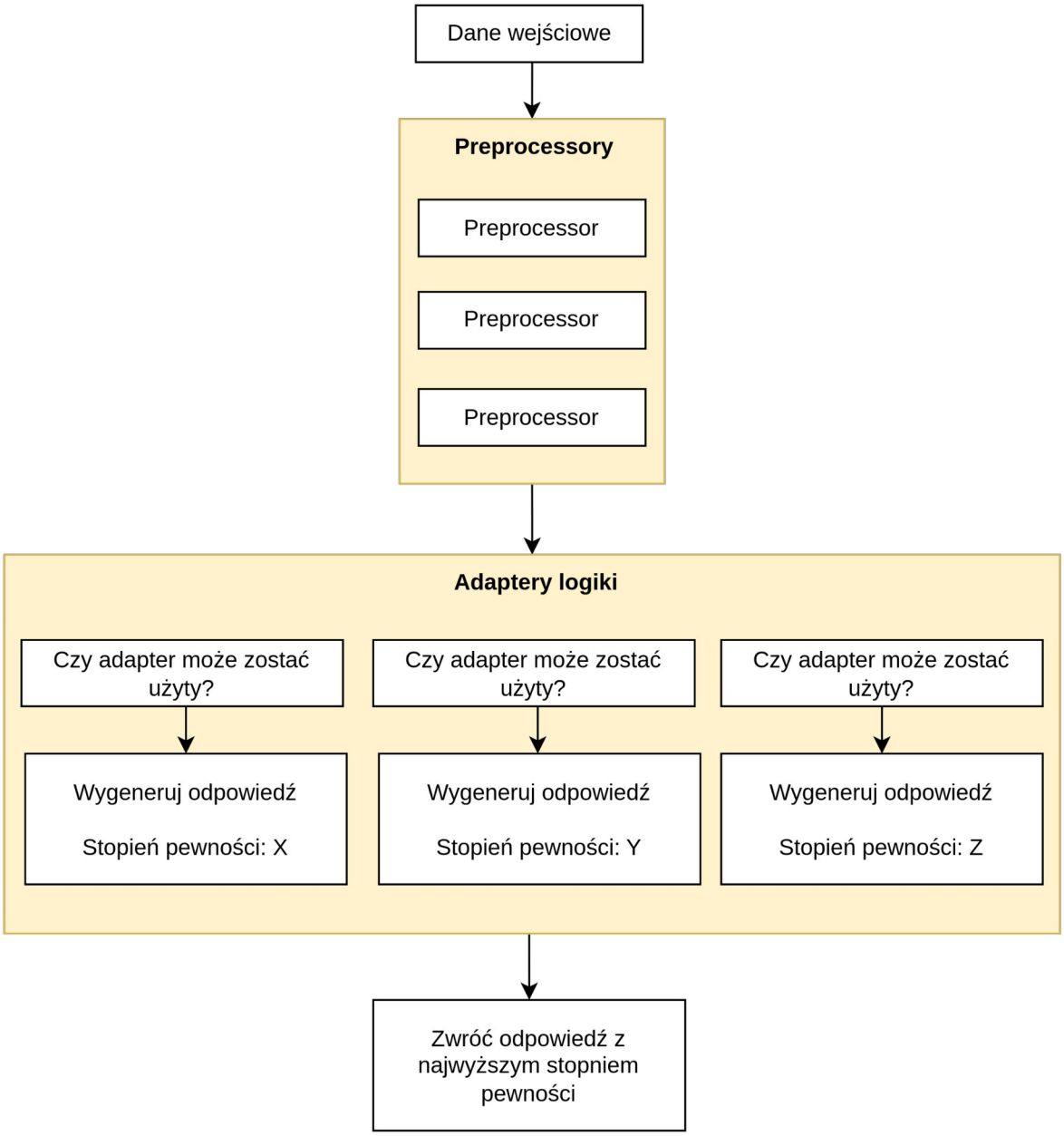
```
from will.decorators import randomly
class Plugin(WillPlugin):
    @randomly(
        start_hour='10',
        end_hour='17',
        day_of_week="mon-fri",
        num_times_per_day=1
    )
    def walkmaster(self):
        self.say("@all time for a walk!")
```

3.2.1.6. Wnioski

Biblioteka Will umożliwia tworzenie chatbotów we wspieranych komunikatorach tekstowych lub konsoli w oparciu o wyrażenia regularne. Wykorzystywane są również dekoratory z języka Python, które mogą zostać użyte tylko w funkcjach klas dziedziczących po klasie WillPlugin. Tworzenie nowych reguł wymaga utworzenia nowych klas lub rozszerzania tych istniejących o dodatkowe funckje i dekoratory, co utrudnia dynamiczną deklarację reguł na podstawie zewnętrznych danych (np. plik CSV, zapytanie API).

3.2.2. Chatterbot

Biblioteka Chatterbot podchodzi do zagadnienia inaczej. Chatterbot opiera się na predefiniowanej liście do budowy korpusu pytań i odpowiedzi. Do przetwarzania logiki wykorzystywane są adaptery logiki (ang. logic adapters) [29]. Przed przekazaniem danych wejściowych do adapterów logiki biblioteka iteruje po preprocesorach (ang. preprocessors) [30] czyli klasach odpowiedzialnych za początkową obróbkę danych wejściowych. Każdy z adapterów logiki przetwarza przygotowane dane wejściowe i określa odpowiedź. Wraz z odpowiedzią na zapytanie adapter określa zmiennoprzecinkową liczbę z zakresu <0, 1> która reprezentuje stopień pewności odpowiedzi. Odpowiedź zwrócona do użytkownika jest odpowiedzią z najwyższym stopniem pewności. W przypadku gdy dwa adaptery zwrócą równoważny stopień pewności, zwracana jest odpowiedź z adaptera, który został dodany do chatbota jako pierwszy [29].



Rys. 4. Przetwarzanie wiadomości przez bibliotekę Chatterbot

Domyślnym adapterem logiki jest adapter BestMatch, który oblicza odległość Levenshteina [3] do określenia podobieństw pomiędzy przetworzonymi danymi wejściowymi a wytrenowaną bazą pytań. Zwracana odpowiedź jest odpowiedzią, dla której powiązane pytanie jest najbardziej podobne do danych wejściowych.

3.2.2.1. Trenowanie

Aby poprawnie funkcjonować Chatterbot wymaga treningu [31]. Trening opiera się na zbudowaniu korpusu pytań i powiązanych odpowiedzi. Wspieranymi klasami służącymi do treningu są ListTrainer, który jako dane treningowe przymuje listę pytanie-odpowiedź,

UbuntuCorpusTrainer, który operuje na korpusie Ubuntu dialog [32], oraz ChatterBotCorpusTrainer, który bazuje na korpusie Chatterbot [33]

Przykładowy trening wygląda następująco:

```
from chatbot import chatbot
from chatterbot.trainers import ListTrainer
trainer = ListTrainer(chatbot)
trainer.train(["Hi there!", "Hello"])
trainer.train(["Greetings!", "Hello"])
```

Możliwym jest również przekazanie dłużej listy. W takim przypadku każdy element listy o parzystym indeksie (0, 2, 4, itd.) jest traktowany jako pytanie, a każdy element listy o nieparzystym indeksie (1, 3, 5, itd.) jest traktowany jako odpowiedź na poprzedzające pytanie.

3.2.2.2. Wnioski

Chatterbot buduje bazę pytań i odpowiedzi, na podstawie której dokonywane są porównania tekstów. Porównywany jest cały korpus pytań, a do porównań wykorzystywana jest domyślnie odległość Levenshteina. Baza danych jest uzupełniana na podstawie treningowych danych wejściowych. Dane te mogą być dowolną listą pytanie-odpowiedź (ListTrainer), mogą zostać uzupełnone na podstawie korpusu Ubuntu Dialog (UbuntuCorpusTrainer), oraz z korpusu Chatterbot Dialog (ChatterBotCorpusTrainer).

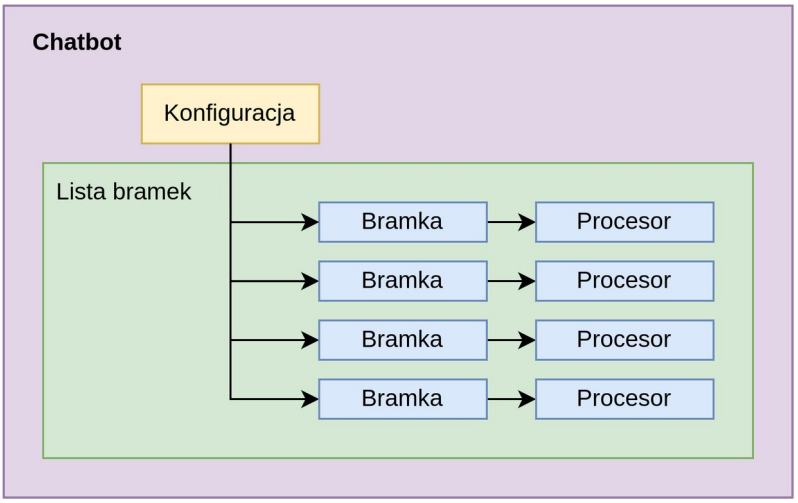
3.3. Planowanie struktury biblioteki

W celu zaplanowania struktury biblioteki wzięte pod uwagę zostały wnioski wyciągnięte z analizy bibliotek Will i Chatterbot. Will umożliwia tworzenie prostych chatbotów w oparciu o wyrażenia regularne, jednak wymaga czasochłonnych konfiguracji powiązanych pluginów i dekoratorów. Z drugiej strony Chatterbot umożliwia łatwiejsze tworzenie złożonych korpusów pytań i odpowiedzi jednak nie wspiera wyrażeń regularnych i opiera się na wektoryzacji i podobieństwach pytań do odpowiedzi, a do obliczeń podobieństw wykorzystuje cały korpus pytań.
Poniżej przedstawiono strukturę biblioteki, która próbuje połączyć prostotę Chatterbota w definiowaniu korpusu z elastycznością Willa przy definiowaniu powiązanych wyrażeń regularnych.

3.4. Struktura biblioteki

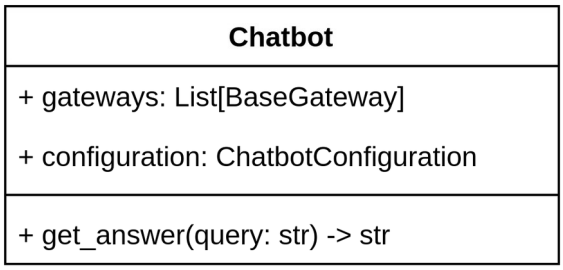
Biblioteka składa się z klas odpowiedzialnych za przetwarzanie wiadomości nazywanymi bramkami oraz na klasach, które są ich składowymi i służą do przetwarzania tekstu wejściowego lub jego fragmentu do wynikowej odpowiedzi, która jest zwracana użytkownikowi, nazywanymi procesorami.

Do utworzenia chatbota wymagany jest obiekt z globalną konfiguracją oraz lista z obiektami wspieranych bramek.



Rys. 5. Składowe chatbota

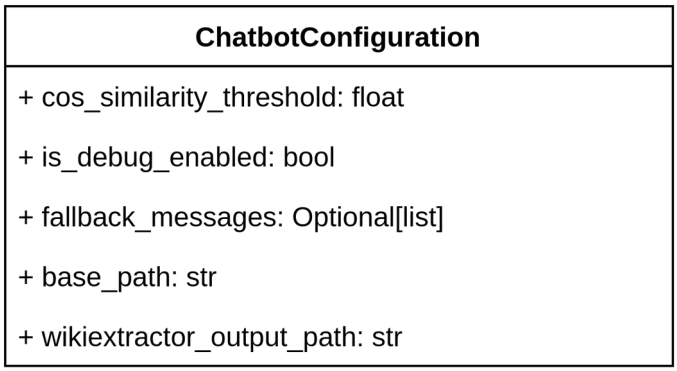
3.4.1. Chatbot



Rys. 6. Diagram UML klasy Chatbot

Chatbot jest klasą łączącą składowe biblioteki w całość, dzięki czemu jest możliwe przeprowadzenie rozmowy. Klasa jako wymagane argumenty przyjmuje listę bramek i instancję konfiguracji, która przy inicjalizacji zostaje jednorazowo przekazana do bramek. Jedyną funkcją tej klasy jest get_answer, której zadaniem jest określenie odpowiedzi na zadane pytanie.

3.4.2. Konfiguracja chatbota



Rys. 7. Diagram UML klasy ChatbotConfiguration

Konfiguracja chatbota jest reprezentowana przez klasę *ChatbotConfiguration*. Konfiguracja określa globalny poziom tolerancji podobieństwa cosinusowego dla bramek CosSimilarity, listę rezerwowych odpowiedzi, z których chatbot będzie losować jedną w przypadku żadna bramka nie zwróci odpowiedzi na zapytanie, ścieżkę do katalogu, w którym znajduje się biblioteka, oraz relatywną ścieżkę do danych wyjściowych z narzędzia *wikiextractor*.

Klasa ta może zostać rozszerzona w celu wsparcia dodatkowych atrybutów konifugracyjnych.

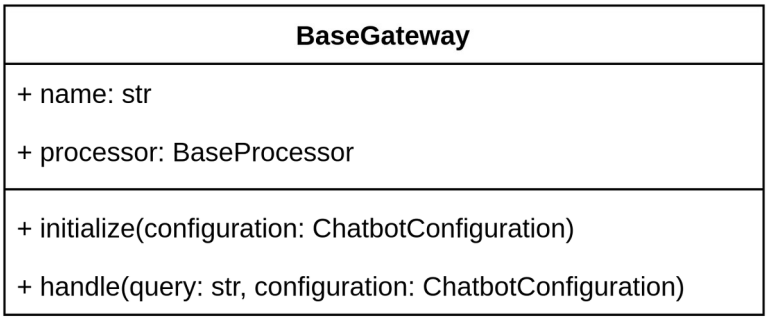
3.4.2.1. Domyślne wartości konfiguracyjne

Klucz konfiguracyjny	**Domyślna wartość**	**Rola**
cos_similarity_threshold	0.8	Domyślna wartość progu tolerancji bramki *CosSimilarity*.
is_debug_enabled	False	Flaga, która określa czy chatbot ma wypisywać informacje pomocne w diagnostyce takie jak wartości podobieństw cosinusowych, czy wektory TF-IDF.
fallback_messages	["Ooops. I did not understand."]	Lista wiadomości, z której zostanie wylosowany element zwrócony użytkownikowi w przypadku gdy żadna z bramek nie będzie w stanie przetworzyć zapytania.

base_path	""	Bazowy katalog chatbota. Ten atrybut jest wykorzystywany do określenia katalogu eksportu biblioteki *wikiextracor*.
wikiextractor_output_path	data/wikiextractor	Relatywna ścieżka katalogu eksportu biblioteki *wikiextractor*. Wykorzystywana przez procesor *WikiSearch*.

Tabela 4. Domyślne wartości konfiguracyjne

3.4.3. Bramka

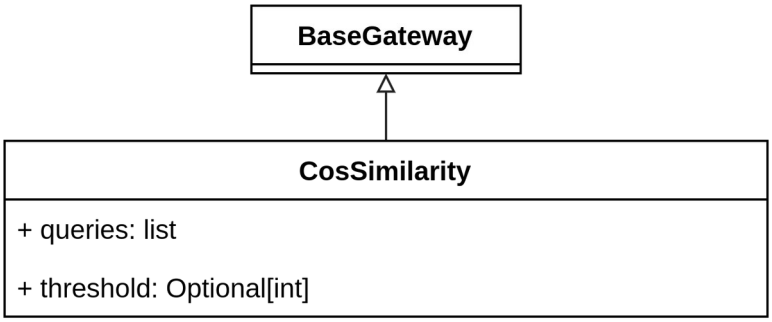


Rys. 8. Diagram UML klasy BaseGateway

Bramka jest odpowiedzialna za określenie czy zapytanie może zostać przetworzone oraz za przygotowanie danych wejściowych dla dalszej części potoku. W przypadku poprawności warunków i pozytywnej weryfikacji bramka przekazuje przetworzone dane wejściowe do procesora. W przypadku negatywnej weryfikacji procesor z bramki jest pomijany, a funkcja *handle* zwraca *None*. Każda z bramek bazuje na klasie *BaseGateway*. Funkcja *initialize* z tej klasy jest wywoływana jednorazowo podczas tworzenia chatbota, a funkcja *handle* jest wywoływana dla każdego nowego zapytania z danymi wejściowymi.

Biblioteka definiuje dwa rodzaje bramek którymi są *CosSimilarity* oraz *Regex*.

3.4.3.1. Bramka CosSimilarity

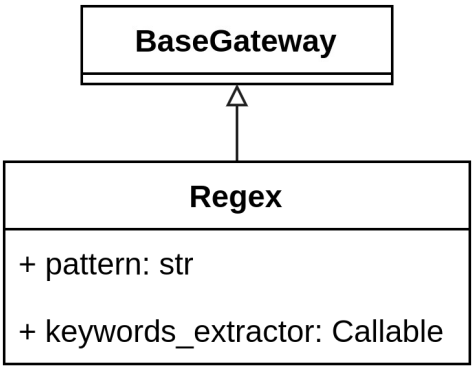


Rys. 9. *Diagram UML bramki CosSimilarity*

Bramka *CosSimilarity* opiera się na liście pytań *queries* i na progu tolerancji podobieństwa *threshold*, który w przypadku braku wartości określany jest na podstawie globalnej konfiguracji *cos_similarity_threshold*. W odróżnieniu od biblioteki Chatterbot do obliczeń odległości między wektorami wykorzystane zostało podobieństwo cosinusowe, a zakres porównywanych pytań jest ograniczony do pytań z bramki. Oznacza to, że bramka przy porównaniach odległości nie wykorzystuje kompletnego korpusu pytań, lecz tylko te pytania, które są w niej zawarte. Programista może zdefiniować wiele bramek tego rodzaju w obrębie jednej instancji chatbota i każda z nich będzie mieć niezależny korpus pytań.

Bramka podczas weryfikacji przekształca dane wejściowe i zdefiniowaną listę pytań na wektory TF-IDF. Następnie obliczane jest podobieństwo cosinusowe danych wejściowych do listy pytań. Bramka wywołuje powiązany procesor, gdy dla któregokolwiek z pytań wartość podobieństwa do danych wejściowych jest większa od określonego progu tolerancji. Wykonanie procesora jest pominięte w przypadku gdy wszystkie podobieństwa są poniżej progu tolerancji.

3.4.3.2. Bramka Regex

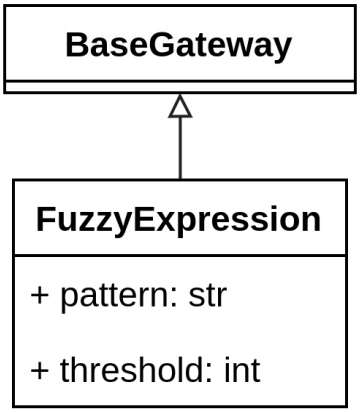


Rys. 10. *Diagram UML bramki Regex*

Zadaniem bramki *Regex* jest określenie czy dane wejściowe są dopasowane do wyrażenia regularnego *pattern*. W przypadku dopasowania, grupy dopasowań zostają przekazane do

anonimowej funkcji *keywords_extractor*. Funkcja ta przekształca grupy dopasowań do postaci słownika *kwargs*. Słownik zostaje następnie przekazany do procesora i może zostać przez niego wykorzystany do wygenerowania odpowiedzi. W przypadku pominięcia argumentu *keywords_extractor* do procesora jako *kwargs* zostaje przekazany pusty słownik *{}*. Procesor WikiSearch jest jedynym predefiniowanym procesorem, dla którego wymagana jest ta operacja, aby poprawnie rozpocząć proces wyszukiwania semantycznego.

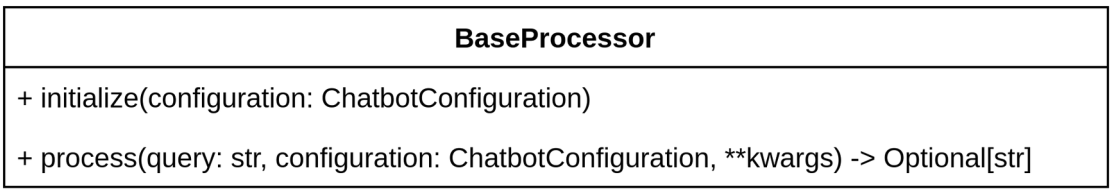
3.4.3.3. Bramka FuzzyExpression



Rys. 11. *Diagram UML bramki FuzzyRegex*

Bramka FuzzyExpression opiera się na bibliotece *RapidFuzz* [34] do określenia podobieństwa wzorca do tekstu. Do obliczenia stopnia podobieństwa *RapidFuzz* wykorzystuje odległość Levenshteina. Atrybut *pattern* to wyrażenie regularne, a *threshold* określa próg tolerancji podobieństwa, powyżej którego bramka przekaże zapytanie do powiązanego procesora. Do określenia podobieństwa wykorzystana została funkcja *fuzz.ratio(str, str)*.

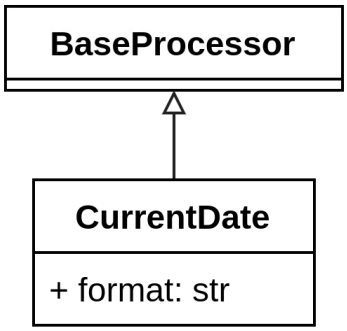
3.4.4. Procesor bramki



Rys. 12. *Diagram UML klasy BaseProcessor*

Zadaniem procesora jest przetworzenie danych wejściowych *query* na odpowiedź, która zostanie zwrócona użytkownikowi. Niektóre z procesorów opierają się na danych przekazanych z bramki przez słownik *kwargs* i w przypadku ich braku zostaną wraz z powiązaną bramką pominięte. Poskutkuje to przekazaniem danych wejściowych do kolejnej bramki z listy.

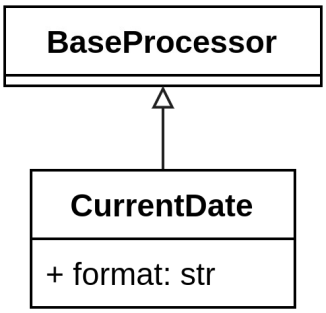
3.4.4.1. Procesor CurrentDate



Rys. 13. *Diagram UML procesora CurrentDate*

Procesor *CurrentDate* jako odpowiedź zwraca aktualną datę poprzez wykorzystanie wbudowanej biblioteki *datetime*. Aby określić w jaki sposób ma zostać zwrócona odpowiedź, można przekazać atrybut *format*. Domyślnie atrybut przyjmuje wartość *"Today is {}."*.

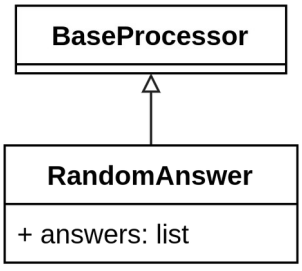
3.4.4.2. Procesor CurrentTime



Rys. 14. *Diagram UML procesora CurrentTime*

Procesor *CurrentTime* jako odpowiedź zwraca aktualną godzinę poprzez wykorzystanie wbudowanej biblioteki datetime. Aby określić w jaki sposób ma zostać zwrócona odpowiedź, można przekazać atrybut format. Domyślnie atrybut przyjmuje wartość *"Current time is {}."*.

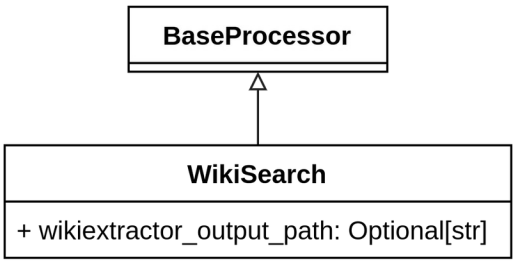
3.4.4.3. Procesor RandomAnswer



Rys. 15. *Diagram UML procesora RandomAnswer*

Procesor *RandomAnswer* jako odpowiedź zwraca losowy element z listy możliwych odpowiedzi przekazanych przy tworzeniu jako *answers*.

3.4.4.4. Procesor WikiSearch



Rys. 16. *Diagram UML procesora WikiSearch*

Procesor *WikiSearch* jako odpowiedź zwraca dwa pierwsze zdania z treści strony najlepiej dopasowanej podczas wyszukiwania semantycznego dokonanego na podstawie zapytania przekazanego przez bramkę. Wyszukiwanie semantyczne opiera się na lokalnym korpusie z portalu Wikipedia, który musi zostać wcześniej wyeksportowany do pliku XML i przekonwertowany do formatu JSON poprzez bibliotekę *wikiextractor* [35]. Do przekształcenia treści stron na reprezentację wektorową procesor wykorzystuje wektoryzację TF-IDF. Wyszukiwanie semantyczne opiera się na PCA, a obliczenia odległości na podobieństwie cosinusowym.

3.5. Python

3.5.1. Typowanie

Aby umożliwić analizę statystyczną kodu i zwiększyć czytelność biblioteki wykorzystane zostało typowanie (ang. type hint) [36] z języka Python.

Wszystkie predefiniowane funkcje z biblioteki są typowane. Do typowania wykorzystane zostały wbudowane typy (int, float, str), klasy z biblioteki (ChatbotConfiguration, BaseGateway, BaseProcessor) oraz wbudowana biblioteka języka Python o nazwie *typing* [37], która roszerza podstawowe typy między innymi o listę typów *List[int]*, oraz o typ opcjonalny *Optional*, który określa, że zwrócony zostanie określony typ lub *None: Optional[str]*.

Poniżej przedstawione zostało typowanie użyte w atrybutach i funkcjach składowych klas biblioteki.

Klasa	Atrybut	Typ

Chatbot	gateways	List[BaseGateway]
	configuration	ChatbotConfiguration
ChatbotConfiguration	cos_similarity_threshold	float
	fallback_messages	Optional[list]
	base_path	str
	wikiextractor_output_path	str
BaseGateway	name	str
	processor	Optional[BaseProcessor]
CosSimilarity	queries	list
	threshold	Optional[float]
Regex	pattern	str
	keywords_extractor	Optional[Callable]
FuzzyExpression	pattern	str
	threshold	float
RandomAnswer	answers	list
WikiSearch	wikiextractor_output_path	Optional[str]

Tabela 5. Typowanie atrybutów ze składowych klas biblioteki

Klasa	**Funkcja**	**Zwracany typ**	**Argument**	**Typ argumentu**
Chatbot	get_answer	str	query	str
BaseGateway	initialize	None	configuration	ChatbotConfiguration
	handle	Optional[str]	query	str
			configuration	ChatbotConfiguration
BaseProcessor	initialize	None	configuration	ChatbotConfiguration
	process	Optional[str]	query	str
			configuration	ChatbotConfiguration

Tabela 6. Typowanie funkcji ze składowych klas biblioteki

3.5.2. Wyjątki

Biblioteka wykorzystuje wyjątki z języka Python do komunikatów o krytycznych błędach w implementacji chatbota. Wyjątek jest rzucany w przypadku gdy nowo zdefiniowana bramka nie nadpisze funkcji *handle* lub w przypadku gdy nowo zdefiniowany procesor nie nadpisze funkcji *process*.

3.6. Aplikacja

Przedstawiona struktura biblioteki została zaimplementowana w języku Python i z jej wykorzystaniem został utworzony przykładowy chatbot.

3.6.1. Budowa chatbota z wykorzystaniem predefiniowanych bramek i procesorów

Aby utworzyć minimalną, funkcjonalną wersję chatbota należy stworzyć instancję konfiguracji, zdefiniować co najmniej jedną bramkę z procesorem i przekazać ją do instancji chatbota.

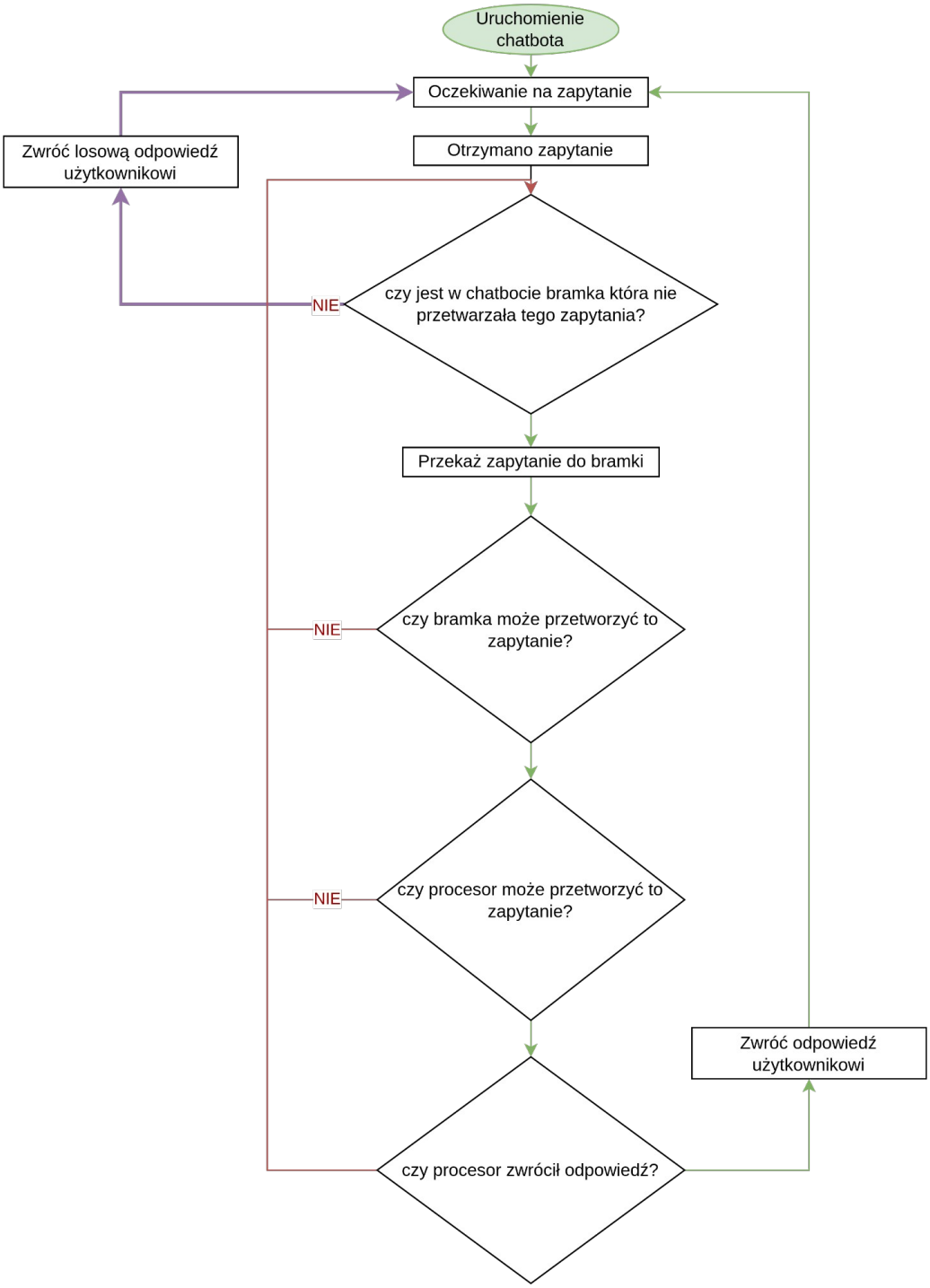
```
from library.chatbot import Chatbot, ChatbotConfiguration
from library.gateway.cos_similarity import CosSimilarity
from library.gateway.processor.random_answer import RandomAnswer

gateways = [
    CosSimilarity(
        queries=["hi", "hello"],
        processor=RandomAnswer(["hi!", "hello, dear", "hi there, how are you?"])
    )
]
configuration = ChatbotConfiguration(
    fallback_messages=["Ooops. I did not understand.", "Ask me something else."]
)
chatbot = Chatbot(gateways, configuration)
```

Z tak przygotowanym chatbotem możliwa już jest interakcja z wykorzystaniem nieskończonej pętli *while,* która oczekuje na wiadomość i przekazuje ją do chatbota.

```
while True:
    query = input('> ')
    print(chatbot.get_answer(query))
```

Funkcja *get_answer* iteruje po bramkach według kolejności dodania do listy. Zapytanie jest następnie przekazywane do bramki, a bramka weryfikuje czy może je przetworzyć i przekazuje je do procesora który zwraca odpowiedź. W niektórych przypadkach bramka lub procesor nie będą w stanie wygenerować odpowiedzi. Dobrym przykładem jest bramka

CosSimilarity, dla której warunkiem koniecznym przekazania zapytania do procesora jest podobieństwo cosinusowe do któregokolwiek z pytań większe od określonego progu tolerancji. Bramki w przypadku niepowodzenia są pomijane. Chatbot następnie wywołuje kolejną bramkę, a w przypadku jej braku odpowiada użytkownikowi losową wiadomością z konfiguracji *fallback_messages*.



Rys. 16. *Konwersja zapytania na odpowiedź*

Możliwym jest również rozszerzenie chatbota o własne bramki i procesory.

3.6.2. Deklaracja własnych bramek

Aby zadeklarować nową bramkę należy utworzyć klasę opierającą się na BaseGateway,

```
from library.chatbot import ChatbotConfiguration, BaseGateway
class OnlyCats(BaseGateway):
    def handle(self, query: str, configuration: ChatbotConfiguration):
        if 'cat' not in query:
            return None
        return self.processor.process(query, configuration, {'meow_sound': 'meow meow'})
```

a następnie dodać jej instancję wraz z procesorem do listy bramek bota.

```
from library.gateway.processor.random_answer import RandomAnswer
from gateway.only_cats import OnlyCats
gateways = [
    # … pozostałe bramki
    OnlyCats(processor=RandomAnswer(["meow"]))
]
# … chatbot
```

3.6.3. Deklaracja własnych procesorów

W podobny sposób może zostać utworzony nowy procesor,

```
from typing import Optional
from library.chatbot import ChatbotConfiguration, BaseProcessor
class Meow(BaseProcessor):
    def process(self, query: str, configuration: ChatbotConfiguration, **kwargs) ->
Optional[str]:
        sound = kwargs.get('meow_sound')
        return sound if sound else "meow"
```

gdzie słownik *kwargs* jest zależny od wartości przekazanych z bramki. Utworzony procesor może zostać wykorzystany w dowolnej bramce.

```
from gateway.only_cats import OnlyCats
from gateway.processor.meow import Meow
gateways = [
    # … pozostałe bramki
    OnlyCats(processor=Meow())
]
# … chatbot
```

3.6.4. Rozszerzenie konfiguracji

Predefiniowana konfiguracja biblioteki może zostać rozszerzona o dodatkowe atrybuty. Aby dodać dodatkowy atrybut do konfiguracji należy rozszerzyć klasę ChatbotConfiguration,

```
from library.chatbot import ChatbotConfiguration
class ExtendedConfiguration(ChatbotConfiguration):
    def __init__(self):
        super().__init__(fallback_messages=["Sorry. I was asleep."])
        self.foo = "bar"
```

a do chatbota przekazać instancję nowej klasy.

```
from extended_configuration import ExtendedConfiguration
# … bramki
configuration = ExtendedConfiguration()
chatbot = Chatbot(gateways, configuration)
```

3.7. Testy

W celu weryfikacji skuteczności biblioteki przeprowadzone zostały testy obrazujące wykorzystanie składowych biblioteki w praktyce przy tworzeniu chatbotów. Dla każdej składowej biblioteki przeprowadzono niezależne testy w dwóch językach - polskim oraz angielskim. Porównane zostały ich wyniki. Końcowym testem jest przykład chatbota złożonego z wielu bramek i procesorów, również takich, które nie należą do biblioteki i zostały utworzone w ramach przykładowej aplikacji.

3.7.1. Procesor RandomAnswer do generowania losowych odpowiedzi

Procesor *RandomAnswer* podczas wykonania funkcji *process* zwraca losową odpowiedź z przekazanej do niego listy *answers*.

Procesor	Zwrócona odpowiedź
RandomAnswer(['hi', 'hello', 'good morning'])	Losowa. “hi”, “hello”, lub “good m

Tabela 7. Przykład użycia procesora RandomAnswer

3.7.2. Procesory CurrentDate i CurrentTime dla aktualnej daty i czasu

Procesory CurrentDate i CurrentTime zwracają odpowiednio aktualną datę i aktualną godzinę. Możliwym jest przekazanie opcjonalnego atrybutu format który określa w jaki sposób wyświetlić powiązaną datę lub godzinę.

Procesor	Zwrócona odpowiedź

CurrentDate()	Aktualna data w formacie %Y-%m-%d,
CurrentTime()	Aktualny czas w formacie %H:%M:%S, np. 21:18:23

Tabela 8. Przykład użycia procesorów CurrentDate, CurrentTime

3.7.3. Procesor WikiSearch do wyszukiwania semantycznego w częściowym korpusie Wikipedii

Procesor	Zwrócona odpowiedź
WikiSearch()	Pierwsze dwa zdania z dokumentu najlep zapytania

Tabela 9. Przykład użycia procesora WikiSearch

Dla testów w języku polskim z portalu wyeksportowano 24 strony o kotach z kategorii **Kategoria:Słynne_koty**: Oscar_(kot), Muezza, Fred_(kot), Chico_(kot), Socks_(kot), CC_(kot), F.D.C._Willard, Wilberforce_(kot), Humphrey_(kot), Oscar_(maskotka_okrętowa), Lwy_z_Tsavo, Pantera_z_Rudraprayag, Tygrys_z_Champawat, Maru, Orangey, Creme_Puff, Grumpy_Cat, Filuś, Félicette, Umbriaga, Główny_Myszołap_w_służbie_Sekretariatu_Gabinetu, Larry_(kot), Lwica_Ryksa, Stepan_(kot)

Dla testów w języku angielskim z portalu wyeksportowano 31 stron o kotach z kategorii **Category:Individual_cats_in_the_United_States**: Browser_(cat), Colonel_Meow, Grumpy_Cat, Lil_Bub, Nora_(cat), Stewie_(cat), Stubbs_(cat), Scarlett_(cat), Creme_Puff_(cat), Venus_(cat), Jorts_(cat), Hank_the_Cat, Willow_(calico_cat), Oscar_(therapy_cat), Prince_Chunk, Meow_(cat), Room_8, Fred_the_Undercover_Kitty, Lewis_(cat), Homer_the_Blind_Wonder_Cat, Dusty_the_Klepto_Kitty, Henri,_le_Chat_Noir, Dewey_Readmore_Books, Lorenzo_the_cat, Mr._Nuts, Orangey, Little_Nicky_(cat), Sockington, Tara_(cat), Scarlett's_Magic, F._D._C._Willard, Jack_(cat)

Do eksportu stron wykorzystano dedykowaną ku temu stronę portalu, a do konwersji pliku z formatu XML do formatu JSON użyto biblioteki *wikiextractor* [36]. Następnie przygotowany został chatbot do którego przekazano pytania o kotach.

```
import os
from library.chatbot import Chatbot, ChatbotConfiguration
from library.gateway.regex import Regex
from library.gateway.processor.wiki_search import WikiSearch

gateways = [
```

```
    Regex(pattern='(.*)', processor=WikiSearch())
]
configuration = ChatbotConfiguration(base_path=os.path.dirname(__file__))
chatbot = Chatbot(gateways, configuration)
while True:
    query = input(">")
    print(chatbot.get_answer(query))
```

3.7.3.1. Test w języku polskim

Pytanie	**Odpowiedź chatbota**
który z kotów odbył udany lot kosmiczny	Félicette – jedyny kot, który odbył udany lo Wykonała lot suborbitalny 18 października
dachowiec	Fred (ur. w maju 2005, zm. 10 sierpnia 200 krótkowłosy kot domowy, który zasłynął ja nowojorskiej policji w dochodzeniu dotycz świadczenia usług weterynaryjnych.
najstarszy kot	Oscar – kot, maskotka okrętowa na niemiec liniowym „Bismarck”, zaokrętowany oficja "Bordkatze der Bismarck" („kot pokładowy
pantera ludojad	Umbriaga – kot towarzyszący szczecińskim połowie XX wieku; jego postać jest obecna folklorze morskim i żeglarskim.
szczeciński kot	Umbriaga – kot towarzyszący szczecińskim połowie XX wieku; jego postać jest obecna folklorze morskim i żeglarskim.

Tabela 10. Wyniki testów wyszukiwania semantycznego w języku polskim

3.7.3.2. Test w języku angielskim

Pytanie	**Odpowiedź chatbota**
oldest cat ever	Creme Puff (August 3, 1967 – August owned by Jake Perry of Austin, Texas, ever recorded, according to the 2010 e Records", when she died aged 38 years
longest cat	Stewie (2005 – February 4, 2013) was domestic cat, according to the "Guinne

	Stewie was measured at on August 28,
cat that died because of lung failure	Meow (– May 5, 2012), also known as male domestic cat who attracted intern animal shelter publicized efforts to slir have him adopted. However, Meow di after entering the animal shelter, on M
famous library cat	Stubbs (April 12, 1997 – July 21, 2017 honorary mayor of Talkeetna, Alaska, death. Stubbs was described as a tourist attrac with cards and letters, and drawing 30 (most of whom were en route to other Denali) who hoped to meet "the mayor
cat responsible for burglary	Dusty the Klepto Kitty is a domestic S notoriety in early 2011 for his acts of " February 2011 appearance on the "Late Letterman", Dusty had stolen 16 car w dish towels, 7 wash cloths, 5 towels, 1 gloves, 1 pair of mittens, 3 aprons, 40 1 dog collar, 6 rubber toys, 1 blanket, golf club head cover, 1 safety mask, 2 balloons, 1 pair of pajama pants, 8 batl miscellaneous objects
grumpy cat	Henri, le Chat Noir (; French for "Henr series of short films on the existential written and directed by William Brade Henry (2003–2020), a male longhair tu

Tabela 11. Wynik testów wyszukiwania semantycznego w języku angielskim

Skuteczność powstałego chatbota nie pozostała stabilna podczas testów, zwracając dla niektórych zapytań (oznaczonych powyżej na czerwono) wyniki, które odstawały od przypuszczanych, biorąc pod uwagę kontekst wyeksportowanych stron.

Dla zapytania "*famous library cat"* trafniejszą odpowiedzią jest strona *"Dewey Readmore Books"* która opisuje jednego z najbardziej znanych bibliotecznych kotów. Podobnie dla zapytania *"grumpy cat"* trafniejszą odpowiedzią byłby *"Grumpy Cat"* czyli kotka która zyskała światową sławę przez swój marudny wyraz pyska. Zapytanie *"najstarszy kot"* powinno zwrócić analogicznie do wersji angielskiej korpusu kotkę *"Creme Puff"*, a do pantery ludojada bardziej pasuje *"Pantera z Rudraprayag"*, która w latach 20 XX wieku

zabiła ponad 120 osób w Indiach. W pozostałych testowych przypadkach zwrócone strony były najbardziej odpowiednimi dla danych zapytań.

3.7.4. Bramka CosSimilarity i wyszukiwanie poprzez podobieństwo cosinusowe

Bramka CosSimilarity wykorzystuje wektoryzację TF-IDF i podobieństwo cosinusowe do określenia podobieństwa zapytania *query* to określonych pytań *queries*. Możliwym jest określenie progu tolerancji po przekroczeniu którego bramka wykona powiązany procesor. W przypadku braku zdefiniowanego progu tolerancji wykorzystywana jest wartość domyślna z konfiguracji *configuration.cos_similarity_threshold = 0.8.*

3.7.4.1. Test w języku polskim

```
from library.chatbot import Chatbot, ChatbotConfiguration
from library.gateway.cos_similarity import CosSimilarity
from library.gateway.processor.random_answer import RandomAnswer
gateways = [
   CosSimilarity(
      queries=['hej', 'witaj', 'dzień dobry', 'dobry wieczór'],
      processor=RandomAnswer(['hej!']),
      threshold=<próg tolerancji>
   )
]
configuration = ChatbotConfiguration(fallback_messages=['Ups. Nie zrozumiałem o co ci chodzi'])
chatbot = Chatbot(gateways, configuration)
while True:
   query = input(">")
   print(chatbot.get_answer(query))
```

Pytanie	Wartości podobieństw cosinusowych	Próg tolerancji	Odpowiedź chatbota
hej	hej (1.0) witaj (0.0) dzień dobry (0.0) dobry wieczór (0.0)	0.8 (wartość domyślna)	hej!
		0.7	
		0.6	
dzień dobry	hej (0.0) witaj (0.0) dzień dobry (1.0)	0.8 (wartość domyślna)	hej!

	dobry wieczór (0.3833)		
		0.7	
		0.6	
dzień miałem dobry ale wieczór już nie	hej (0.0) witaj (0.0) dzień dobry (0.7865) dobry wieczór (0.7865)	0.8 (wartość domyślna)	Ups. Nie zrozumiałem o co ci chodzi
		0.7	hej!
		0.6	

Tabela 12. Wynik testów wyszukiwania przez podobieństwo cosinusowe w języku polskim

3.7.4.2. Test w języku angielskim

```
from library.chatbot import Chatbot, ChatbotConfiguration
from library.gateway.cos_similarity import CosSimilarity
from library.gateway.processor.random_answer import RandomAnswer
gateways = [
    CosSimilarity(
        queries=['hi', 'hello', 'good morning'],
        processor=RandomAnswer(['hi!']),
        threshold=<próg tolerancji>
    )
]
configuration = ChatbotConfiguration()
chatbot = Chatbot(gateways, configuration)
while True:
    query = input(">")
    print(chatbot.get_answer(query))
```

Pytanie	**Wartości podobieństw cosinusowych**	**Próg tolerancji**	**Odpowiedź chatbota**
hi	hi (1.0) hello (0.0) good morning (0.0)	0.8 (wartość domyślna)	
		0.7	
		0.6	

good day	hi (0.0) hello (0.0) good morning (0.7071)	0.8 (wartość domyślna)	Ooops. I did not understand.
		0.7	
		0.7	hi!
hey	hi (0) hello (0) good morning (0)	0.8 (wartość domyślna)	Ooops. I did not understand.
		0.7	
		0.6	

Tabela 13. Wynik testów wyszukiwania przez podobieństwo cosinusowe w języku angielskim

3.7.5. Bramka Regex i wyrażenia regularne

Bramka Regex opiera się na wyrażeniu regularnym *pattern* i umożliwia przetworzenie znalezionych grup do słownika przed przekazaniem ich do procesora przez *keywords_extractor*. Przetworzenie to jest użyteczne w przypadku gdy dla procesora interesujący jest tylko fragment wiadomości, a nie całość.

```
from typing import Optional
from library.chatbot import Chatbot, ChatbotConfiguration, BaseProcessor
from library.gateway.regex import Regex

class GreetByName(BaseProcessor):
    def process(self, query: str, configuration: ChatbotConfiguration, **kwargs) ->
Optional[str]:
        name = kwargs.get('name')
        return "Cześć, {}".format(name) if name else None

gateways = [
    Regex(
        pattern='nazywam się (.*)',
        keywords_extractor=lambda groups: {'name': groups[0]},
        processor=GreetByName()
    ),
]
configuration = ChatbotConfiguration(fallback_messages=['Ups. Nie zrozumiałem o co ci
chodzi'])
chatbot = Chatbot(gateways, configuration)
while True:
    query = input(">")
    print(chatbot.get_answer(query))
```

Pytanie	Odpowiedź chatbota
nazywam się Piotr	Cześć, Piotr
możesz mi coś opowiedzieć?	Ups. Nie zrozumiałem o co ci chod

Tabela 14. Wynik testów bramki Regex

3.7.6. Bramka FuzzyExpression i wyszukiwanie rozmyte

Bramka FuzzyExpression umożliwia wyszukiwanie rozmyte i podobnie do CosSimilarity zwraca wyniki, dla dopasowań których stopień podobieństwa przekracza próg tolerancji *threshold*. Zwrócone podobieństwo jest z zakresu 0 - 100 jednak jest normalizowane do zakresu 0 - 1 aby zachować spójny zakres atrybutu *threshold* dla obu bramek.

```
from typing import Optional
from library.chatbot import Chatbot, ChatbotConfiguration
from library.gateway.fuzzy_expression import FuzzyExpression
from library.gateway.processor.random_answer import RandomAnswer

gateways = [
    FuzzyExpression(
        pattern='michelangelo',
        processor=RandomAnswer(['Widziałeś już Pietę?', 'David jest olbrzymi!']),
        threshold=0.8
    )
]
configuration = ChatbotConfiguration(fallback_messages=['Ups. Nie zrozumiałem o co ci
chodzi'])
chatbot = Chatbot(gateways, configuration)
while True:
    query = input(">")
    print(chatbot.get_answer(query))
```

Pytanie	Znormalizowany stopień podobieństwa	Odpowiedź chatbota
micheaolangelo	0.9231	David jest oblrzymi!
machieololoangelo	0.7586	Ups. Nie zrozumiałem o co ci chodzi
michael angel	0.88	Widziałeś już Pietę?

Tabela 15. Wynik testów bramki FuzzyExpression

3.7.7. Złożony chatbot

Aby zobrazować połączenie kilku bramek z procesorami, został utworzony złożony chatbot, który składa się ze wszystkich predefiniowanych bramek: *Regex*, *CosSimilarity*, oraz *FuzzyExpression*. Składa się również ze wszystkich predefiniowanych procesorów: *RandomAnswer*, *WikiSearch*, *CurrentDate*, oraz *CurrentTime*. Definiuje też własny procesor *Math,* którego rolą jest wykonywanie prostych obliczeń matematycznych dodawania, odejmowania, mnożenia i dzielenia.

```
import os
from typing import Optional
from library.chatbot import Chatbot, ChatbotConfiguration, BaseProcessor
from library.gateway.regex import Regex
from library.gateway.cos_similarity import CosSimilarity
from library.gateway.fuzzy_expression import FuzzyExpression
from library.gateway.processor.random_answer import RandomAnswer
from library.gateway.processor.wiki_search import WikiSearch
from library.gateway.processor.current_date import CurrentDate
from library.gateway.processor.current_time import CurrentTime

class Math(BaseProcessor):
    def __init__(self, format: str = "{}."):
        self.format = format
    def process(self, query: str, configuration: ChatbotConfiguration, **kwargs) ->
Optional[str]:
        left = kwargs.get('left')
        op = kwargs.get('op')
        right = kwargs.get('right')
        if not left or not op or not right:
            return None
        return self.format.format(eval("{} {} {}".format(left, op, right)))

gateways = [
    CosSimilarity(
        queries=['hej', 'cześć', 'dzień dobry', 'dobry wieczór', 'witaj'],
        processor=RandomAnswer(['no hej :-)', 'cześć!']),
        threshold=0.85
    ),
    CosSimilarity(
        queries=['aktualna data', 'data dzisiaj'],
        processor=CurrentDate(format='Dzisiaj jest {}.')),
    CosSimilarity(
        queries=['która jest godzina'],
        processor=CurrentTime(format='Jak ten czas leci... jest już {}.')
    ),
    Regex(
        pattern='((?:czym jest|kim jest|chce wiedzieć)? ?(?:możesz powiedzieć mi|znasz|znasz
```

```
jakiegoś)? ?(?:więcej|coś|cokolwiek|czym jest)? ?(?:o)?) ?(.*)',
        keywords_extractor=lambda groups: {'search_query': groups[1] if groups[0] else
None},
        processor=WikiSearch()
    ),
    Regex(
        pattern='.*?(\d+)\s?(\+|-|\*|\/)\s?(\d+)',
        keywords_extractor=lambda groups: {
            'left': groups[0],
            'op': groups[1],
            'right': groups[2]
        } if len(groups) == 3 else None,
        processor=Math(format='Według mojego kalkulatora wynik to {}.')
    ),
    FuzzyExpression(
        pattern='salvador dalí',
        threshold=0.85,
        processor=RandomAnswer(['Salvador Dalí był hiszpańskim malarzem i jednym z
najbardziej znanych surrealistów.'])
    )
]
configuration = ChatbotConfiguration(base_path=os.path.dirname(__file__),
fallback_messages=['Nie zrozumiałem.'])
chatbot = Chatbot(gateways, configuration)
while True:
    query = input(">")
    print(chatbot.get_answer(query))
```

Przykładowa rozmowa z chatbotem:

```
> hej
no hej :-)

> jesteś w stanie obliczyć 21 * 1379?
Według mojego kalkulatora wynik to 28959.

> a wiesz może która jest godzina?
Jak ten czas leci... jest już 15:41.

> a jaka jest dzisiaj data?
Dzisiaj jest 2023-02-01.

> znasz jakiegoś kota ze szczecina?
Umbriaga – kot towarzyszący szczecińskim żeglarzom w połowie XX wieku; jego postać
jest obecna w miejscowym folklorze morskim i żeglarskim.

> powiedz mi jeszcze ile to jest 247512/123
Według mojego kalkulatora wynik to 2012.2926829268292.
```

> a 777+1234?
Według mojego kalkulatora wynik to 2011.

> może jakiś opwiesz mi o jakimś artyście
Nie zrozumiałem.

> salvador dale?
Salvador Dalí był hiszpańskim malarzem i jednym z najbardziej znanych surrealistów.

> opwiedz mi jakąś historię
Nie zrozumiałem.

> co dzisiaj robiłeś
Nie zrozumiałem.

Skuteczność chatbota jest bezpośrednio zależna od zadawanych pytań. Chatbot gubi się dla zapytań które nie zostały wzięte pod uwagi przy implementacji.

Rozdział 4

Wnioski

Celem pracy było stworzenie biblioteki do chatbotów zapewniającej sposoby dopasowania wykorzystane w istniejących rozwiązaniach *Will* i *Chatterbot* jednocześnie ułatwiając przy tym rozszerzalność aplikacji o dodatkowe funkcjonalności z dziedziny przetwarzania języka naturalnego. Cel ten został spełniony. Biblioteka wspiera dopasowanie odpowiedzi na podstawie podobieństwa cosinusowego danych wejściowcyh do pytań. Wspierane jest również dopasowanie przez wyrażenia regularne i wyrażenia rozmyte. W przeciwieństwie do biblioteki *Will* która wykorzystuje dekoratory z języka Python do określenia wzorców, opisana biblioteka realizuje dopasowania poprzez klasy-procesory, umożliwiając przy tym dopasowania przez wyrażeinia regularne przez bramkę *Regex* oraz dopasowania rozmyte przez bramkę *FuzzyExpression*. Możliwym jest też wyszukiwanie semantyczne w wyeksportowanym korpusie z portalu Wikipedia przy pomocy procesora *WikiSearch*. Testy wykazały częściową skuteczność tej wyszukiwarki. Skuteczność chatbota jest zależna od zapytania oraz tego, czy dany przypadek został wzięty pod uwagę w powiązanych bramkach. Chatbot nie jest w stanie odpowiedzieć na bardziej złożone zapytania. Przedstawiona biblioteka jest łatwo rozszerzalna poprzez nowe bramki i procesory, które mogą też być dowolnie ze sobą połączone w zależności od potrzeb.

Bibliografia

[1] Paul Deane, A nonparametric method for extraction of candidate phrasal terms. In Proceedings of the 43rd Annual Meeting on Association for Computational Linguistics, 2005, 605–613.

[2] Brown University corpus. http://icame.uib.no/brown/bcm-los.html. Data dostępu 14.01.2023

[3] H. Lane, C. Howard, H. Hapke, Przetwarzanie języka naturalnego w akcji, Helion, 2021

[4] Gregory Grefenstette, Tokenization, Syntactic Wordclass Tagging, 1999, 117–133

[5] nltk.tokenize.casual module. https://www.nltk.org/api/nltk.tokenize.casual.html#module-nltk.tokenize.casual. Data dostępu 25.01.2023

[6] Christopher Fox, A stop list for general text, ACM SIGIR Forum Volume 24, Issue 1-2, 1990, 19–21

[7] Word Lists and Lexicons. Sample usage for corpus. https://www.nltk.org/howto/corpus.html#word-lists-and-lexicons. Data dostępu 26.01.2023

[8] sklearn.feature_extraction.text.TfidfVectorizer. https://scikit-learn.org/stable/modules/generated/sklearn.feature_extraction.text.TfidfVectorizer.html. Data dostępu 25.01.2023

[9] Tf–idf term weighting. https://scikit-learn.org/stable/modules/feature_extraction.html#tfidf-term-weighting. Data dostępu 28.01.2023

[10] Vector Norm. https://mathworld.wolfram.com/VectorNorm.html. Data dostępu: 26.01.2023

[11] Sidorov, Grigori, Gelbukh, Soft Similarity and Soft Cosine Measure: Similarity of Features in Vector Space Model, Computación y Sistemas. 18, 2014, 491–504

[12] Jeffrey E. F. Friedl, Mastering Regular Expressions Third Edition, O'Reilly Media, 2006

[13] Regular expression operations. https://docs.python.org/3/library/re.html. Data dostępu 29.01.2023

[14] Sudipto Banerjee, Anindya Roy, Linear Algebra and Matrix Analysis for Statistics, Chapman and Hall/CRC, 2014

[15] J. Dongarra, M. Gates, A. Haidar, J. Kurzak, P. Luszczek, S. Tomov, I. Yamazaki, The Singular Value Decomposition: Anatomy of Optimizing an Algorithm for Extreme Scale, SIAM Review, Volume 60, Issue 4, 2018, 808–865

[16] Singular value decomposition. https://en.wikipedia.org/wiki/Singular_value_decomposition. Data dostępu 30.01.2023

[17] Rasmus Bro, Age K. Smilde, Principal component analysis, Analytical Methods, Issue 9, 2014

[18] sklearn.decomposition.PCA. https://scikit-learn.org/stable/modules/generated/sklearn.decomposition.PCA.html. Data dostępu 21.01.2023

[19] D.P. Berrar, W. Dubitzky, M. Granzow, Singular value decomposition and principal component analysis, A Practical Approach to Microarray Data Analysis, Springer, 2002

[20] Z. Drmac, K. Veselic, New fast and accurate Jacobi SVD algorithm: I, LAPACK Working Notes, 2005

[21] N. Halko, P. G. Martinsson, J. A. Tropp, Finding Structure with Randomness: Probabilistic Algorithms for Constructing Approximate Matrix Decompositions, SIAM Review, Volume 53, Issue 2, 2011, 217-288

[22] Sparse eigenvalue problems with ARPACK. https://docs.scipy.org/doc/scipy/tutorial/arpack.html. Data dostępu 29.01.2023

[23] Meet Will. http://skoczen.github.io/will. Data dostępu 21.01.2023

[24] About ChatterBot. https://chatterbot.readthedocs.io/en/stable. Data dostępu 23.01.2023

[25] K. D. Smith, J. J. Jewett, S. Montanaro, A. Baxter, PEP 318 – Decorators for Functions and Methods, 2003, https://peps.python.org/pep-0318. Data dostępu 28.01.2023

[26] IO Backends. http://skoczen.github.io/will/backends/io/. Data dostępu 28.01.2023

[27] What will can notice. http://skoczen.github.io/will/plugins/notice/. Data dostępu 29.01.2023

[28] Behrang QasemiZadeh, Anne-Kathrin Schumann, The ACL RD-TEC 2.0: A Language Resource for Evaluating Term Extraction and Entity Recognition Methods, Proceedings of the Tenth International Conference on Language Resources and Evaluation (LREC'16), 2016, 1862–1868

[29] Logic Adapters. https://chatterbot.readthedocs.io/en/stable/logic/index.html. Data dostępu 29.01.2023

[30] Preprocessors. https://chatterbot.readthedocs.io/en/stable/preprocessors.html. Data dostępu 29.01.2023

[31] Training. https://chatterbot.readthedocs.io/en/stable/tutorial.html#training-your-chat-bot. Data dostępu 29.01.2023

[32] Ubuntu Dialogue Corpus v1.0. http://dataset.cs.mcgill.ca/ubuntu-corpus-1.0/. Data dostępu 29.01.2023

[33] ChatterBot Corpus. https://chatterbot.readthedocs.io/en/stable/corpus.html. Data dostępu 29.01.2023

[34] RapidFuzz. https://maxbachmann.github.io/RapidFuzz/. Data dostepu 30.01.2023

[35] Wikiextractor - A tool for extracting plain text from Wikipedia dumps. https://attardi.github.io/wikiextractor/. Data dostępu 29.01.2023

[36] R. Gonzalez, P. House, I. Levkivskyi, L. Roach, G. van Rossum, PEP 526 – Syntax for Variable Annotations, https://peps.python.org/pep-0526/. Data dostępu 29.01.2023

[37] typing — Support for type hints. https://docs.python.org/3/library/typing.html. Data dostępu 29.01.2023