**Chmura Obliczeniowa i Technologie Big Data**

*Moduł „PyPork” realizujący mechanizmy logiki rozmytej dla platformy Apache Pig*

|  |  |
| --- | --- |
| Kierownik | dr hab. inż. Dariusz Mrozek, Prof. PŚ |
| Autorzy | Jan Skowronek  Zuzanna Lempa  Paweł Kudzia  Wojciech Wątroba |
| Kierunek | Informatyka SSM |
| Specjalizacja | IDSI |
| Technologia | Apache Pig |

Spis treści

[1. Wstęp 4](#_Toc77016314)

[2.1. Cel projektu 4](#_Toc77016315)

[2.2. Założenia projektu 4](#_Toc77016316)

[2. Stos technologiczny 4](#_Toc77016317)

[2.1. Przygotowanie środowiska 5](#_Toc77016318)

[2.2. Przykład użycia biblioteki „PyPork” 7](#_Toc77016319)

[2.3. Architektura biblioteki „PyPork” 8](#_Toc77016320)

[3. Interfejs Pig Latin 9](#_Toc77016321)

[3.1. Ładowanie danych z pliku CSV 9](#_Toc77016322)

[3.2. Zmiana wartości na zmienną lingwistyczną 10](#_Toc77016323)

[3.3. Grupowanie danych po zmiennej lingwistycznej 10](#_Toc77016324)

[3.4. Złącznie danych po zmiennej lingwistycznej 11](#_Toc77016325)

[3.5. Zastosowanie funkcji fuzzy\_level 11](#_Toc77016326)

[3.6. Zastosowanie funkcji F\_AND, F\_OR, F\_NOT 11](#_Toc77016327)

[3.7. Zastosowanie funkcji \*\_membership („około”) 12](#_Toc77016328)

[3.8. Zastosowanie funkcji fequal 13](#_Toc77016329)

[4. Interfejs Python 13](#_Toc77016330)

[4.1. Rozmyte operatory logiczne 13](#_Toc77016331)

[4.1.1. Funkcja F\_OR 13](#_Toc77016332)

[4.1.2. Funkcja F\_AND 14](#_Toc77016333)

[4.1.3. Funkcja F\_NOT 14](#_Toc77016334)

[4.2. Funkcje przynależności 14](#_Toc77016335)

[4.2.1. Funkcja triangle\_membership 14](#_Toc77016336)

[4.2.2. Funkcja trapezoid\_membership 15](#_Toc77016337)

[4.2.3. Funkcja l\_class\_membership 15](#_Toc77016338)

[4.2.4. Funkcja y\_class\_membership 16](#_Toc77016339)

[4.2.5. Funkcja gaussian\_membership 16](#_Toc77016340)

[4.3. Funkcje pomocnicze 16](#_Toc77016341)

[4.3.1. Funkcja \_callProperFun 17](#_Toc77016342)

[4.3.2. Funkcja \_line\_intersection 17](#_Toc77016343)

[4.4. Funkcje operacji rozmytych 18](#_Toc77016344)

[4.4.1. Funkcja one\_to\_lingustic 18](#_Toc77016345)

[4.4.2. Funkcja list\_to\_lingustic 19](#_Toc77016346)

[4.4.3. Funkcja fuzzy\_level 19](#_Toc77016347)

[4.4.4. Funkcja fequal 19](#_Toc77016348)

[4.4.5. Funkcja add\_new\_pattern 20](#_Toc77016349)

[4.4.6. Funkcja get\_patterns\_names 20](#_Toc77016350)

[5. Podsumowanie 21](#_Toc77016351)

# 1. Wstęp

Big Data to termin odnoszący się do dużych i różnorodnych zbiorów danych. Przetwarzanie tego typu danych jest zadaniem wyjątkowo trudnym i wymagającym, ale z drugiej strony analiza ogromnych zbiorów okazuje się być wartościowa, ponieważ pozwala zdobyć nowy rodzaj wiedzy. Z tego też powodu na przestrzeni lat powstały narzędzia i rozbudowane platformy programistyczne umożliwiające m.in. efektywne gromadzenie, przechowywanie i przetwarzanie dużych ilości informacji. Warto zaznaczyć, że dane mogą pochodzić z wielu różnych źródeł, co niesie za sobą konieczność użycia specjalistycznego oprogramowania, za pomocą którego można usprawnić proces przetwarzania. Korzyści z używania narzędzi Big Data są niezaprzeczalne, dlatego istotne jest, aby istniała możliwość rozszerzania o nowe funkcjonalności i tak już zaawansowanych platform.

W ramach kursu *„Chmura Obliczeniowa i Technologie Big Data”* należy zrealizować własny moduł rozszerzający dla jednej z przykładowych platform programistycznych stosowanych w dziedzinie Big Data.

## 2.1. Cel projektu

Celem projektu jest wzbogacenie przykładowej platformy programistycznej o moduł, który umożliwi realizację rozmytego przetwarzania danych. Moduł ten należy opracować dla technologii Apache Pig będącej częścią ekosystemu Hadoop.

## 2.2. Założenia projektu

Nowy moduł dla Apache Pig powinien dostarczać podstawowe funkcjonalności rozmytego przetwarzania danych, do których należy zaliczyć: rozmyte filtrowanie z wykorzystaniem zmiennych lingwistycznych, grupowanie rozmyte względem zmiennej lingwistycznej, czy złączenia. Bibliotekę należy zaimplementować wykorzystując do tego UDF (*ang. User-defined function*), czyli funkcje rozszerzające dany program lub środowisko.

Ogromną zaletą ekosystemu Hadoop jest to, iż funkcje mogą zostać zaimplementowane w wybranych językach programowania, wspieranych przez aplikację Apache Pig. Jednym z nich jest Python i to właśnie język ten posłużył do wykonania projektu.

# 2. Stos technologiczny

W niniejszym rozdziale przedstawiony został stos technologiczny, który posłużył do realizacji projektu. Opisane zostały zagadnienia związane z przygotowaniem środowiska testowego, jak i również architektura autorskiej biblioteki *„PyPork”*.

## 2.1. Przygotowanie środowiska

W celu rozpoczęcia pracy z platformą Apache Pig należy przygotować środowisko, które zazwyczaj przyjmuje postać jakiegoś systemu operacyjnego z rodziny Linux. Z reguły wiele narzędzi programistycznych jest tworzonych właśnie pod ten system. Nie inaczej jest w przypadku technologii Hadoop i Apache Pig, w związku z czym pierwszym krokiem do rozpoczęcia pracy było skonfigurowanie przykładowej maszyny wirtualnej. Autorzy projektu zdecydowali, że użyją jedną z popularnych dystrybucji Linuksa, którą niewątpliwie jest Ubuntu. Instalacja systemu jest prosta i tak właściwie sprowadza się do postępowania według wskazówek wyświetlanych na ekranie. W przypadku, gdy zdecydowano się wybrać wersję desktop Ubuntu, podczas instalacji systemu warto skorzystać z opcji *„minimal”*, dzięki której finalnie zainstalowane zostaną tylko niezbędne pakiety i narzędzia.

Po prawidłowym wdrożeniu systemu na maszynie wirtualnej można przejść do kolejnego kroku, który dotyczy instalacji Hadoop i Apache Pig. Aby zminimalizować mogące się pojawić problemy i przy okazji uprościć ręczną konfigurację, autorzy projektu utworzyli przykładowy skrypt powłoki Bash. Skrypt ten realizuje następujące zadania:

* Instalacja środowiska uruchomieniowego Java,
* Pobranie i rozpakowanie paczek Hadoop i Apache Pig,
* Przeniesienie plików Hadoop i Apache Pig do odpowiednich katalogów,
* Utworzenie zmiennych środowiskowych,
* Sprawdzenie poprawności wykonanej instalacji za pomocą polecenia wypisującego informacje o zainstalowanej wersji danego oprogramowania.

Na listingu 3.1.1. zaprezentowany został opisywany skrypt instalacyjny. Należy pamiętać o tym, aby uruchomić skrypt z uprawnieniami administratora systemu.

Listing 2.1.1. Skrypt instalacyjny

#!/bin/bash

sudo apt install openjdk-8-jdk

sudo apt install openjdk-8-jre

java -version

# main directory where Hadoop and Pig will be installed

MAIN\_DIRECTORY=/usr/local

cd $MAIN\_DIRECTORY

# download hadoop

sudo wget http://apache.claz.org/hadoop/common/hadoop-2.10.1/hadoop-2.10.1.tar.gz

sudo tar xzf hadoop-2.10.1.tar.gz

sudo mkdir hadoop

sudo mv hadoop-2.10.1/\* hadoop

sudo rm -r hadoop-2.10.1

# download pig

sudo wget http://apache.claz.org/pig/pig-0.17.0/pig-0.17.0.tar.gz

sudo tar xzf pig-0.17.0.tar.gz

sudo mkdir pig

sudo mv pig-0.17.0/\* pig

sudo rm -r pig-0.17.0

# home directories for Hadoop and Pig

MY\_HADOOP\_HOME=$MAIN\_DIRECTORY/hadoop

MY\_PIG\_HOME=$MAIN\_DIRECTORY/pig

# create environmental variables

echo export HADOOP\_HOME=$MY\_HADOOP\_HOME >> ~/.bashrc

echo export HADOOP\_INSTALL=$MY\_HADOOP\_HOME >> ~/.bashrc

echo export HADOOP\_MAPRED\_HOME=$MY\_HADOOP\_HOME >> ~/.bashrc

echo export HADOOP\_COMMON\_HOME=$MY\_HADOOP\_HOME >> ~/.bashrc

echo export HADOOP\_HDFS\_HOME=$MY\_HADOOP\_HOME >> ~/.bashrc

echo export HADOOP\_COMMON\_LIB\_NATIVE\_DIR=$MY\_HADOOP\_HOME/lib/native >> ~/.bashrc

echo export HADOOP\_OPTS="-Djava.library.path=$MY\_HADOOP\_HOME/lib/nativ" >> ~/.bashrc

echo export YARN\_HOME=$MY\_HADOOP\_HOME >> ~/.bashrc

# if you installed java 7 change name here

echo export JAVA\_HOME=/usr/lib/jvm/java-1.8.0-openjdk-amd64 >> ~/.bashrc

# add Hadoop and Pig to PATH

HADOOP\_PATH=$MY\_HADOOP\_HOME/sbin:$MY\_HADOOP\_HOME/bin

PIG\_PATH=$MY\_PIG\_HOME/bin

echo export PATH=$PATH:$HADOOP\_PATH:$PIG\_PATH >> ~/.bashrc

source ~/.bashrc

hadoop version

pig -version

echo "Script completed."

Następnie można zweryfikować poprawności instalacji Apache Pig. Najprostszy sposób polega na uruchomieniu aplikacji w trybie lokalnym korzystając z komendy przedstawionej na listingu 2.1.2.

pig -x local

Listing 2.1.2. Uruchomienie Apache Pig w trybie lokalnym

Jeśli instalacja odbyła się w sposób prawidłowy, aplikacja powinna uruchomić się w trybie interaktywnym, w którym możliwe jest wpisywanie poleceń języka Pig Latin. Aby zapoznać się z podstawowymi funkcjami można np. wczytać plik CSV i wyświetlić jego zawartość.

## 2.2. Przykład użycia biblioteki „PyPork”

W celu sprawdzenia poprawności opisywanej biblioteki należy skorzystać z dostarczonego pliku skryptowego w języku Pig Latin, który umieszczony został w katalogu pig\_examples. W przykładzie pokazano użycie operacji rozmytych na danych pochodzących z pliku CSV.

Ponadto należy pamiętać również o tym, aby na początku własnych skryptów zarejestrować główny plik biblioteki main\_fuzzy.py. Plik ten musi zostać umieszczony w miejscu, do którego jest łatwy dostęp, jak np. główny katalog użytkownika lub jego pulpit.

Rejestracja pliku biblioteki odbywa się w pliku o rozszerzeniu .pig za pomocą operatora REGISTER w podobny sposób, jak prezentuje to listing 2.2.1. Ponadto należy pamiętać o tym, aby w miejscu, w którym znajduje się skrypt Pig Latin umieszczony został również plik konfiguracyjny pattern.json oraz odpowiedni plik CSV przechowujące dane używane w skrypcie użytkownika.

Listing 2.2.1. Wskazanie lokalizacji skryptu main\_fuzzy.py

REGISTER '/home/tester/Desktop/PyPork/main\_fuzzy.py' using jython as fuzzy

Uruchomienie skryptu dla języka Pig Latin odbywa się z wykorzystaniem komendy z listingu 2.2.2. W oknie konsoli systemowej powinny zostać wypisane zarówno logi generowane przez narzędzie Hadoop oraz wyniki poszczególnych operacji zdefiniowanych w skrypcie języka Pig Latin.

Listing 2.2.2. Uruchomienie skryptu linguistic.pig

pig -x local linguistic.pig

## 2.3. Architektura biblioteki „PyPork”

Projekt składa się z kilku plików skryptowych języka Python, w których zdefiniowane zostały funkcje odpowiedzialne za realizację określonych zadań.

W katalogu pig\_examples umieszczono przykłady użycia rozszerzenia *„PyPork”*. Można tutaj znaleźć:

* linguistic.pig – zawiera kilka przykładów pokazujących działanie autorskich funkcji,
* user\_data.csv – przechowuje analizowane dane,
* pattern.json – plik konfiguracyjny, w którym zdefiniowane są zmienne lingwistyczne.

W głównym katalogu znajduje się skrypt main\_fuzzy.py zawierający wszystkie potrzebne funkcje do prawidłowego działania biblioteki. W pliku tym dostępne są m.in. funkcje operujące na wartościach lingwistycznych, definicje funkcji przynależności (np. trójkątna, trapezowa), czy funkcje realizujące działanie rozmytych operatorów AND, OR i NOT. Każda z funkcji została udokumentowana bezpośrednio w kodzie źródłowym (standardowe komentarze dokumentacji), dzięki czemu podczas pracy z kodem możliwe jest uzyskanie informacji pomocniczych bezpośrednio w edytorze kodu. Skrypt ten jest wymagany do pracy z Apache Pig, co zresztą zaprezentowano w poprzednim podrozdziale.

Ponadto główny katalog repozytorium zawiera kilka skryptów, które weryfikują poprawność działania poszczególnych funkcjonalności:

* fequal\_test.py – sprawdza działanie funkcji fequal,
* linguistic\_test.py – zawiera testy sprawdzające obsługę wartości rozmytych,
* membership\_test.py – przechowuje przykładowe testy, które weryfikują poprawność zaimplementowanych funkcji przynależności,
* operators\_test.py – sprawdza działanie rozmytych operatorów AND, OR i NOT.

Powyższe skrypty należy uruchomić z poziomu konsoli systemowej w podobnym sposób, jak pokazano na listingu 2.3.1.

Listing 2.3.1. Uruchamianie skryptu Python z poziomu konsoli systemowej

python operators\_test.py

Wynik uruchomienia pliku operators\_test.py zaprezentowany został na listingu 2.3.2.

Listing 2.3.2. Wynik działania testów ze skryptu operators\_test.py

Tests: F\_OR, F\_AND, F\_NOT operators

========== test 1 ==========

values[0]: 1

values[1]: 0.5

result: {'F\_OR': 1, 'F\_AND': 0.5, 'F\_NOT': {'values[0]': 0, 'values[1]': 0.5}}

========== test 2 ==========

values[0]: 0.75

values[1]: 0.1

result: {'F\_OR': 0.75, 'F\_AND': 0.1, 'F\_NOT': {'values[0]': 0.25, 'values[1]': 0.9}}

========== test 3 ==========

values[0]: 0.1

values[1]: 0.4

result: {'F\_OR': 0.4, 'F\_AND': 0.1, 'F\_NOT': {'values[0]': 0.9, 'values[1]': 0.6}}

========== test 4 ==========

values[0]: 0.6

values[1]: 1

result: {'F\_OR': 1, 'F\_AND': 0.6, 'F\_NOT': {'values[0]': 0.4, 'values[1]': 0}}

========== test 5 ==========

values[0]: 0.2

values[1]: 0.3

result: {'F\_OR': 0.3, 'F\_AND': 0.2, 'F\_NOT': {'values[0]': 0.8, 'values[1]': 0.7}}

Poniżej w sposób graficzny zaprezentowane zostało ułożenie modułu w stosunku do całego systemu Hadoop i Apache Pig.

=obrazek=

Rysunek 2.3.1. Schemat architektury biblioteki „PyPork”

# 3. Interfejs Pig Latin

Rozdział zawiera opis interfejsu dostępnego w języku Pig Latin z wykorzystaniem kilku przykładów. Do każdego scenariusza użycia dołączone zostały instrukcje i otrzymane wyniki.

## 3.1. Ładowanie danych z pliku CSV

Listing 3.1.1. prezentuje sposób, w jaki można załadować przykładowe dane z pliku CSV. Za pomocą polecenia DUMP możliwe jest wypisanie wartości zmiennej w standardowym strumieniu wyjścia. Wyniki operacji znajdują się w komentarzu.

Listing 3.1.1. Ładowanie danych z pliku CSV

--Ładowanie danych z pliku user\_data:

users = LOAD 'user\_data.csv' USING PigStorage(',') AS (firstname: chararray, lastname:chararray,wzrost:int, wiek:int);

DUMP users;

-- (zosia,samosia,150,12)

-- (ala,makota,160,32)

-- (ALICJA,makota,160,18)

-- (harry,potter,194,89)

-- (ola,makota,175,8)

## 3.2. Zmiana wartości na zmienną lingwistyczną

Listing 3.2.1. prezentuje wykorzystanie funkcji one\_to\_linguistic, dzięki której można przeprowadzić konwersję wartości numerycznej na odpowiadającą jej wartość zmiennej lingwistycznej. Ponadto w przykładzie umieszczono stopień zgodności.

Listing 3.2.1. Wykorzystanie funkcji one\_to\_linguistic

--Zamiana wartości na zmienną lingwistyczną(one\_to\_lingustic) wraz z wyliczeniem stopnia zgodnośći(fuzzy\_level)

fuzzy\_users = FOREACH users GENERATE firstname, lastname, wzrost, fuzzy.one\_to\_lingustic('wzrost', wzrost) as fuzzy\_wzrost, fuzzy.fuzzy\_level('wzrost',wzrost,fuzzy.one\_to\_lingustic('wzrost', wzrost)) as level, wiek, fuzzy.one\_to\_lingustic('wiek', wiek) as fuzzy\_wiek;

DUMP fuzzy\_users;

-- (zosia,samosia,150,niski,0.8,12,mlody)

-- (ala,makota,160,niski,0.5,32,stary)

-- (ALICJA,makota,161,niski,0.4,18,mlody)

-- (harry,potter,194,gigant,0.7,89,emeryt)

-- (ola,makota,175,wysoki,0.75,8,dzieciak)

## 3.3. Grupowanie danych po zmiennej lingwistycznej

Na listingu 3.3.1. przedstawiono przykład obrazujący grupowanie danych po zmiennej lingwistycznej.

Listing 3.3.1. Grupowanie danych po zmiennej lingwistycznej

--Groupowanie danych po zmiennej lingwistycznej:

grouped\_fuzzy\_users = GROUP users BY fuzzy.one\_to\_lingustic('wzrost', wzrost) ;

DUMP grouped\_fuzzy\_users;

-- (niski,{(ALICJA,makota,160,niski,18,mlody),(ala,makota,160,niski,32,stary),(zosia,samosia,150,niski,12,mlody)})

-- (gigant,{(harry,potter,194,gigant,89,emeryt)})

-- (wysoki,{(ola,makota,175,wysoki,8,dzieciak)})

## 3.4. Złącznie danych po zmiennej lingwistycznej

Na listingu 3.4.1. zaprezentowane zostało wykonanie operacji złączenia po zmiennej lingwistycznej. Instrukcje zwracają ten sam wynik.

Listing 3.4.1. Złączenie danych po zmiennej lingwistycznej

users2 = LOAD 'user\_data.csv' USING PigStorage(',') AS (firstname: chararray, lastname:chararray,wzrost:int, wiek:int, zajecie:chararray);

fuzzy\_us2 = FOREACH users2 GENERATE fuzzy.one\_to\_lingustic('wzrost', wzrost) as fw, zajecie;

--złączenie dwóch PigStorage po zmiennej lingwistycznej

-- oba join daja ten sam wynik

-- self\_joined\_fuzzy\_users = JOIN fuzzy\_users BY fuzzy\_wzrost, fuzzy\_us2 BY fw;

self\_joined\_fuzzy\_users = JOIN users BY fuzzy.one\_to\_lingustic('wzrost', wzrost), users2 BY fuzzy.one\_to\_lingustic('wzrost', wzrost);

DUMP self\_joined\_fuzzy\_users;

-- (ALICJA,makota,160,niski,18,mlody,niski,prawnik)

-- (ALICJA,makota,160,niski,18,mlody,niski,prawnik)

-- (ALICJA,makota,160,niski,18,mlody,niski,uczen)

-- (ala,makota,160,niski,32,stary,niski,prawnik)

-- (ala,makota,160,niski,32,stary,niski,prawnik)

-- (ala,makota,160,niski,32,stary,niski,uczen)

-- (zosia,samosia,150,niski,12,mlody,niski,prawnik)

-- (zosia,samosia,150,niski,12,mlody,niski,prawnik)

-- (zosia,samosia,150,niski,12,mlody,niski,uczen)

-- (harry,potter,194,gigant,89,emeryt,gigant,nauczyciel)

-- (ola,makota,175,wysoki,8,dzieciak,wysoki,pacjent)

## 3.5. Zastosowanie funkcji fuzzy\_level

Listing 3.5.1. przedstawia użycie funkcji fuzzy\_level, która wylicza stopień zgodności dla podanej wartości do zmiennej lingwistycznej wywołując odpowiednią funkcję przynależności.

Listing 3.5.1. Zastosowanie funkcji fuzzy\_level

--Funkcja fuzzy\_level wylicza stopień zgodności dla podanej wartości do zmiennej lingwistycznej wywołując odpowiednią funkcję przynależności.

filtered = FILTER users BY fuzzy.fuzzy\_level('wzrost',wzrost,'wysoki') > 0.5;

DUMP filtered;

-- (ola,makota,175,8)

## 3.6. Zastosowanie funkcji F\_AND, F\_OR, F\_NOT

Listingi 3.6.1. prezentuje sposób użycia funkcji realizujących działanie rozmytych operatorów logicznych F\_AND, F\_OR i F\_NOT.

Listing 3.6.1. Zastosowanie funkcji F\_AND, F\_OR, F\_NOT

-- Przykłady łączenia funkcji za pomocą F\_AND, F\_OR i F\_NOT.

filtered\_fand = FILTER users2 BY fuzzy.F\_AND(fuzzy.triangle\_membership(wzrost, 140.0, 150.0, 160.0), fuzzy.trapezoid\_membership(wiek,10.0,12.0,20.0,30.0)) > 0.5;

DUMP filtered\_fand;

-- -- (zosia,samosia,150,12,uczen)

filtered\_for = FILTER users2 BY fuzzy.F\_OR(fuzzy.triangle\_membership(wzrost, 140.0, 150.0, 160.0), fuzzy.trapezoid\_membership(wiek,10.0,12.0,20.0,30.0)) > 0.5;

DUMP filtered\_for;

-- (zosia,samosia,150,12,uczen)

-- (ALICJA,makota,160,18,prawnik)

filtered\_font = FILTER users2 BY fuzzy.F\_NOT(fuzzy.trapezoid\_membership(wiek,10.0,12.0,20.0,30.0)) > 0.5;

DUMP filtered\_font;

-- (ala,makota,160,32,prawnik)

-- (harry,potter,194,89,nauczyciel)

-- (ola,makota,175,8,pacjent)

filtered\_fand2 = FILTER users BY fuzzy.F\_AND(fuzzy.fuzzy\_level('wzrost',wzrost,'wysoki'), fuzzy.fuzzy\_level('wiek',wiek,'dzieciak')) > 0.1;

DUMP filtered\_fand2;

-- (ola,makota,175,8)

filtered\_for2 = FILTER users BY fuzzy.F\_OR(fuzzy.fuzzy\_level('wzrost',wzrost,'wysoki'), fuzzy.fuzzy\_level('wiek',wiek,'emeryt')) > 0.5;

DUMP filtered\_for2;

-- (harry,potter,194,89)

-- (ola,makota,175,8)

## 3.7. Zastosowanie funkcji \*\_membership („około”)

Na listingu 3.7.1. przedstawiono sposób użycia funkcji \*\_membership, która realizuje założenie funkcji około, gdyż zwraca stopień zgodności dla podanej funkcji przynależności. W bibliotece „PyPork” dostępnych jest kilka implementacji funkcji przynależności.

Listing 3.7.1. Zastosowanie funkcji około

--  Funkcja fuzzy.\*\_membership realizuje założenie funkcji około, gdyż zwraca stopień zgodności dla podanej funkcji przynależności.

-- Przykładem takiego filtrowania byłoby wywołanie fuzzy.\*\_membership(zmienna, parametry) > minimalne\_ktyterium\_zgodności).

-- Dostępne są funkcje trojkatna (triangle\_membership), trapezowa (trapezoid\_membership), l class (l\_class\_membership), y class (y\_class\_membership) oraz gaussowska (gaussian\_membership)

-- Poniżej przykład dla funkcji trójkątnej:

tmp = FILTER users2 BY fuzzy.triangle\_membership(wzrost, 140.0, 150.0, 160.0) >0.5;

DUMP tmp;

-- (zosia,samosia,150,12,uczen)

## 3.8. Zastosowanie funkcji fequal

Na listingu 3.8.1. zaprezentowano działanie funkcji fequal, która zwraca stopień zgodności obliczając punkty przecięcia rozmytych wartości.

Listing 3.8.1. Zastosowanie funkcji fequal

-- Funkcja fequal zwraca stopien zgodności obliczając punkty przecięcia rozmytych wartości.

-- Ze względu na ograniczenia Pig Latin dotyczące wywoływania udf(user defined functions) poprawne jej wykorzystanie do implementacji funkcji fjoin wydaje się niemożliwe.

tmp = FOREACH users2 GENERATE firstname, fuzzy.fequal(wzrost, 10, wzrost+5, 5);

DUMP tmp;

-- (zosia,0.6666666666666667)

-- (ala,0.6666666666666667)

-- (ALICJA,0.6666666666666667)

-- (harry,0.6666666666666667)

-- (ola,0.6666666666666667)

# 4. Interfejs Python

Niniejszy rozdział dotyczy dokumentacji technicznej interfejsu dostępnego w języku Python. Opisane zostały wszystkie funkcje z pliku main\_fuzzy.py.

## 4.1. Rozmyte operatory logiczne

Dostępne są trzy rozmyte operatory logiczne.

### 4.1.1. Funkcja F\_OR

Funkcja realizuje działanie rozmytego operatora OR.

Wejście: Wartości numeryczne, które mają zostać przetestowane. Każda wartość musi być z zakresu od 0 do 1 (włącznie).

Wyjście: Maksymalna wartość z podanych na wejściu wartości.

Przykład użycia w Python: F\_OR(1, 0.5) lub F\_OR(\*[1, 0.5]).

Listing 4.1.1.1. Funkcja F\_OR

def F\_OR(\*values):

    return max(values)

### 4.1.2. Funkcja F\_AND

Funkcja realizuje działanie rozmytego operatora AND.

Wejście: Wartości numeryczne, które mają zostać przetestowane. Każda wartość musi być z zakresu od 0 do 1 (włącznie).

Wyjście: Minimalna wartość z podanych na wejściu wartości.

Przykład użycia w Python: F\_AND(1, 0.5) lub F\_AND(\*[1, 0.5]).

Listing 4.1.2.1. Funkcja F\_AND

def F\_AND(\*values):

    return min(values)

### 4.1.3. Funkcja F\_NOT

Funkcja realizuje działanie rozmytego operatora NOT.

Wejście: Wartość numeryczna, która ma zostać przetestowana. Wartość musi być z zakresu od 0 do 1 (włącznie).

Wyjście: Wynik wykonania operacji rozmytego operatora NOT.

Przykład użycia w Python: F\_NOT(1).

Listing 4.1.3.1. Funkcja F\_NOT

def F\_NOT(value):

    return 1 - value

## 4.2. Funkcje przynależności

Dostępnych jest pięć funkcji przynależności.

### 4.2.1. Funkcja triangle\_membership

Funkcja realizuje działanie trójkątnej funkcji przynależności.

Wejście: Wartość numeryczna x oraz parametry funkcji: a, b, c.

Wyjście: Wartość numeryczna z zakresu od 0 do 1 (włącznie).

Przykład użycia w Python: triangle\_membership(4, 5, 10, 12).

Listing 4.2.1.1. Funkcja triangle\_membership

def triangle\_membership(x, a, b, c):

    if x <= a:

        return 0

    elif x <= b:

        return (x - a) / (b - a)

    elif x <= c:

        return (c - x) / (c - b)

    else:

        return 0

### 4.2.2. Funkcja trapezoid\_membership

Funkcja realizuje działanie trapezowej funkcji przynależności.

Wejście: Wartość numeryczna x oraz parametry funkcji: a, b, c, d.

Wyjście: Wartość numeryczna z zakresu od 0 do 1 (włącznie).

Przykład użycia w Python: trapezoid\_membership(0.5, 1, 11, 14, 15).

Listing 4.2.2.1. Funkcja trapezoid\_membership

def trapezoid\_membership(x, a, b, c, d):

    if x <= a:

        return 0

    elif x <= b:

        return (x - a) / (b - a)

    elif x <= c:

        return 1

    elif x <= d:

        return (d - x) / (d - c)

    else:

        return 0

### 4.2.3. Funkcja l\_class\_membership

Funkcja realizuje działanie funkcji przynależności klasy L.

Wejście: Wartość numeryczna x oraz parametry funkcji: a, b.

Wyjście: Wartość numeryczna z zakresu od 0 do 1 (włącznie).

Przykład użycia w Python: l\_class\_membership(5, 24, 30).

Listing 4.2.3.1. Funkcja l\_class\_membership

def l\_class\_membership(x, a, b):

    if x <= a:

        return 1

    elif x <= b:

        return (b - x) / (b - a)

    else:

        return 0

### 4.2.4. Funkcja y\_class\_membership

Funkcja realizuje działanie funkcji przynależności klasy Y.

Wejście: Wartość numeryczna x oraz parametry funkcji: a, b.

Wyjście: Wartość numeryczna z zakresu od 0 do 1 (włącznie).

Przykład użycia w Python: y\_class\_membership(5, 24, 30).

Listing 4.2.4.1. Funkcja y\_class\_membership

def y\_class\_membership(x, a, b):

    if x <= a:

        return 0

    elif x <= b:

        return (x - a) / (b - a)

    else:

        return 1

### 4.2.5. Funkcja gaussian\_membership

Funkcja realizuje działanie gaussowskiej funkcji przynależności.

Wejście: Wartość numeryczna x oraz parametry funkcji: mean, sd.

Wyjście: Wartość numeryczna z zakresu od 0 do 1 (włącznie).

Przykład użycia w Python: gaussian\_membership(1, 2, 5).

Listing 4.2.5.1. Funkcja gaussian\_membership

def gaussian\_membership(x, mean, sd):

    return exp(((x - mean) \*\* 2) / 2 \* (sd \*\* 2))

## 4.3. Funkcje pomocnicze

Dostępnych jest kilka funkcji pomocniczych, które używane są przez inne funkcje.

### 4.3.1. Funkcja \_callProperFun

Funkcja odpowiedzialna za wywołanie wskazanej funkcji przynależności. Jest to funkcja pomocnicza, która nie powinna być bezpośrednio używana.

Wejście: Wartość obiektu JSON z pliku konfiguracyjnego JSON. Obiekt JSON musi zawierać klucz „function” oraz inne odpowiednie klucze, które opisują parametry dla wskazanej funkcji przynależności. Przykładowo dla trójkątnej funkcji przynależności muszą zostać zdefiniowane klucze: „a”, „b” i „c”.

Wyjście: Wartość zwrócona przez wskazaną funkcję przynależności. Wartość numeryczna z zakresu od 0 do 1 (włącznie).

Przykład użycia w Python: \_callProperFun(v, value).

Listing 4.3.1.1. Funkcja \_callProperFun

def \_callProperFun(v, value):

    if v['function'] == 'triangle':

        return triangle\_membership(value, float(v['a']), float(v['b']), float(v['c']))

    elif v['function'] == 'trapezoid':

        return trapezoid\_membership(value, float(v['a']), float(v['b']), float(v['c']), float(v['d']))

    elif v['function'] == 'l\_class':

        return l\_class\_membership(value, float(v['a']), float(v['b']))

    elif v['function'] == 'y\_class':

        return y\_class\_membership(value, float(v['a']), float(v['b']))

    elif v['function'] == 'gaussian':

        return gaussian\_membership(value, float(v['a']), float(v['b']))

### 4.3.2. Funkcja \_line\_intersection

Funkcja odpowiedzialna za sprawdzenie, czy podane na wejściu linie przecinają się. Jest to funkcja pomocnicza, która nie powinna być bezpośrednio używana.

Wejście: Pierwsza linia, druga linia.

Wyjście: Wartość określający, czy linie przecinają się.

Przykład użycia w Python: \_line\_intersection(line1\_1, line2\_1).

Listing 4.3.2.1. Funkcja \_line\_intersection

def \_line\_intersection(L1, L2):

    Ax1 = L1[0][0]

    Ay1 = L1[0][1]

    Ax2 = L1[1][0]

    Ay2 = L1[1][1]

    Bx1 = L2[0][0]

    By1 = L2[0][1]

    Bx2 = L2[1][0]

    By2 = L2[1][1]

    d = (By2 - By1) \* (Ax2 - Ax1) - (Bx2 - Bx1) \* (Ay2 - Ay1)

    if d:

        uA = ((Bx2 - Bx1) \* (Ay1 - By1) - (By2 - By1) \* (Ax1 - Bx1)) / d

        uB = ((Ax2 - Ax1) \* (Ay1 - By1) - (Ay2 - Ay1) \* (Ax1 - Bx1)) / d

    else:

        return 0

    if not(0 <= uA <= 1 and 0 <= uB <= 1):

        return 0

    x = Ax1 + uA \* (Ax2 - Ax1)

    y = Ay1 + uA \* (Ay2 - Ay1)

    return y

## 4.4. Funkcje operacji rozmytych

Dostępnych jest kilka funkcji dotyczących operacji rozmytych.

### 4.4.1. Funkcja one\_to\_lingustic

Funkcja odpowiedzialna za konwersję wartości numerycznej do wartości lingwistycznej.

Wejście: Zmienna lingwistyczna, wartość numeryczna.

Wyjście: Wartość lingwistyczna odpowiadającą wartości numerycznej.

Przykład użycia w Python: one\_to\_lingustic('wzrost', 170).

Listing 4.4.1.1. Funkcja one\_to\_lingustic

def one\_to\_lingustic(pattern\_name, value):

    value\_patter = pattern[pattern\_name]

    max\_val = 0

    max\_key = ''

    for k, v in value\_patter.items():

        res = \_callProperFun(v, value)

        if res > max\_val:

            max\_val = res

            max\_key = k

    return max\_key

### 4.4.2. Funkcja list\_to\_lingustic

Funkcja odpowiedzialna za konwersję wartości numerycznych do wartości lingwistycznych.

Wejście: Zmienna lingwistyczna, lista wartości numerycznych.

Wyjście: Lista wartości lingwistycznych odpowiadających wartościom numerycznym.

Przykład użycia w Python: list\_to\_lingustic('wzrost', [110, 170]).

Listing 4.4.2.1. Funkcja list\_to\_lingustic

def list\_to\_lingustic(pattern\_name, values):

    results = []

    for value in values:

        res = one\_to\_lingustic(pattern\_name, value)

        results.append(res)

    return results

### 4.4.3. Funkcja fuzzy\_level

Funkcja odpowiedzialna za wykonywanie operacji sprawdzającej stopień przynależności.

Wejście: Zmienna lingwistyczna, wartość numeryczna, wartość zmiennej lingwistycznej.

Wyjście: Wartość zwrócona przez wskazaną funkcję przynależności. Wartość numeryczna z zakresu od 0 do 1 (włącznie).

Przykład użycia w Python: fuzzy\_level('wzrost', 170, 'wysoki').

Listing 4.4.3.1. Funkcja fuzzy\_level

def fuzzy\_level(pattern\_name, value, key):

    v = pattern[pattern\_name][key]

    return \_callProperFun(v, value)

### 4.4.4. Funkcja fequal

Funkcja odpowiedzialna za wykonanie operacji rozmycia wartości. Zwraca stopień zgodności obliczając punkty przecięcia rozmytych wartości.

Wejście: Wartości numeryczne, poziom rozmycia dla wartości numerycznych.

Wyjście: Stopień zgodności dwóch zmiennych rozmytych.

Przykład użycia w Python: fequal(3, 2, 10, 2).

Listing 4.4.4.1. Funkcja fequal

def fequal(value1, fuzzy\_level1, value2, fuzzy\_level2):

    value1 = float(value1)

    value2 = float(value2)

    fuzzy\_level1 = float(fuzzy\_level1)

    fuzzy\_level2 = float(fuzzy\_level2)

    line1\_1 = [[value1 - fuzzy\_level1, 0.0], [value1, 1.0]]

    line1\_2 = [[value1, 1.0], [value1 + fuzzy\_level1, 0.0]]

    line2\_1 = [[value2 - fuzzy\_level2, 0.0], [value2, 1.0]]

    line2\_2 = [[value2, 1.0], [value1 + fuzzy\_level2, 0.0]]

    max\_level = 0

    levels = [

        \_line\_intersection(line1\_1, line2\_1),

        \_line\_intersection(line1\_1, line2\_2),

        \_line\_intersection(line1\_2, line2\_1),

        \_line\_intersection(line1\_2, line2\_2)

    ]

    for lvl in levels:

        if lvl > max\_level and lvl <= 1:

            max\_level = lvl

    return max\_level

### 4.4.5. Funkcja add\_new\_pattern

Funkcja odpowiedzialna za dodawanie nowej zmiennej lingwistycznej. Zmienna zostaje dodawana do obiektu w aplikacji (przechowywanym w pamięci RAM) i nie jest dodawana do pliku konfiguracyjnego JSON, dzięki czemu możliwe jest dynamiczne zdefiniowanie zmiennych lingwistycznych w trakcie wykonywania skryptu.

Wejście: Zmienna lingwistyczna, słownik przechowujący definicję zmiennej lingwistycznej.

Wyjście: Funkcja nic nie zwraca.

Przykład użycia w Python: add\_new\_pattern('wzrost', {'niski': {'function': 'triangle', 'a': 130, 'b': 155, 'c': 165},'wysoki': {'function': 'triangle', 'a': 160, 'b': 180, 'c': 190}}).

Listing 4.4.5.1. Funkcja add\_new\_pattern

def add\_new\_pattern(pattern\_name, new\_pattern):

    pattern[pattern\_name] = new\_pattern

### 4.4.6. Funkcja get\_patterns\_names

Funkcja odpowiedzialna za zwrócenie wszystkich zmiennych lingwistycznych z pliku konfiguracyjnego JSON.

Wejście: Funkcja nie przyjmuje żadnych argumentów.

Wyjście: Lista zmiennych lingwistycznych.

Przykład użycia w Python: get\_patterns\_names().

Listing 4.4.6.1. Funkcja get\_patterns\_names

def get\_patterns\_names():

    patterns\_names = []

    for k, v in pattern.items():

        patterns\_names.append(k)

    return patterns\_names

# 5. Podsumowanie

Wykonanie projektu pt. *„PyPork”* pozwoliło autorom zapoznać się z platformą Apache Pig, która znajduje zastosowanie w analizie i przetwarzaniu dużych zbiorów danych. Platforma ta udostępnia proceduralny język Pig Latin, dzięki któremu możliwe jest pisanie zarówno bardzo prostych, jak i bardziej zaawansowanych skryptów umożliwiających przetwarzanie danych. Pig Latin oprócz udostępniania typowych mechanizmów charakterystycznych dla języków programowania tj. różne typy danych (tuple, bag, map), czy funkcje (np. LOAD, STORE), daje również możliwość definiowania własnych funkcji, co pozwala wprowadzić dodatkowe operacje w potoku przetwarzania. Opcja rozbudowy istniejących rozwiązań niewątpliwie jest jedną z kluczowych zalet platform Hadoop i Apache Pig, ponieważ dzięki temu standardowy zestaw narzędzi (który swoją drogą i tak jest obszerny) może być rozbudowany. W kontekście projektu *„PyPork”* Apache Pig wzbogacono o wybrane elementy logiki rozmytej, która jak się okazuje może być niezwykle przydatna w ciekawej dziedzinie informatyki jaką jest Big Data.

Zaletą technologii Apache Pig jest przede wszystkim to, iż do definiowanie UDF możliwe jest z wykorzystaniem kilku języków programowania. Implementacja biblioteki w języku Python okazała się bardzo wygodna, aczkolwiek ma swoje ograniczenia, ponieważ język ten nie jest głównym językiem wspieranym przez Apache Pig, o czym zresztą można było się przekonać podczas próby implementacji operacji fjoin. Okazuje się, że z poziomu języka Pig Latin niemożliwe jest przekazanie dwóch kolekcji, w związku z czym potrzebne jest obejście tego ograniczenia np. poprzez wywołanie skryptu .pig z poziomu języka Python.

Autorzy modułu są zadowoleni z uzyskanych wyników i osiągniętego celu, choć nie da się ukryć, że aktualne rozwiązanie mogłoby zostać bardziej usprawnione. Podczas pracy nad finalną implementacji pojawiły się spore trudności, co zresztą zasygnalizowane zostało na kilku spotkaniach konsultacyjnych z prowadzącym kurs. Niemniej jednak zarówno po wskazówkach, jak i kilku testach zrealizowanych podczas oddawania projektu udało się dojść do pozytywnych wniosków. Przy okazji jeśli dodatkowym celem projektu byłoby udostępnienie biblioteki *„PyPork”* szerzej publiczności, wtedy koniecznym krokiem jest usprawnienie publicznego repozytorium w serwisie GitHub. Ponadto konieczna byłaby rozbudowana istniejącej dokumentacji biblioteki. Tego typu zmiana z pewnością podniosłaby ogólną jakość projektu.

Podsumowując, realizacja projektu *„PyPork”* poruszającego dosyć niestandardową tematykę w ramach kursu *„Chmura Obliczeniowa i Technologie Big Data”*, zdecydowanie poszerzyła wiedzę dotyczącą technik pracy z językiem Python, zbiorów rozmytych i co najważniejsze specjalistycznych narzędzi służących do analizy, gromadzenia i przetwarzania dużych zbiorów danych. Nie da się ukryć, że wiedza o istnieniu narzędzi tj. Hadoop, czy Apache Pig jest niezwykle przydatna podczas projektowania komercyjnych produktów informatycznych, które główny nacisk kładą m.in. na sprawne przekazywanie i analizowanie dużych zbiorów informacji.