Politechnika Poznańska
Wydział Informatyki
Instytut Informatyki

Dokumentacja projektu

IMPLEMENTACJA ALGORYTMU PREDYKCJI OPARTEGO O DRZEWO DECYZYJNE W ŚRODOWISKU SQL HURTOWNI DANYCH NETEZZA

Arkadiusz Kowalski, 122070
Jędrzej Klorek, 117321
Jakub Malczewski, 117332

Prowadzący
dr hab. inż. Robert Wrembel, prof. nadzw.

Poznań, 2019 r.

Spis treści

1. Wstęp

1.1. Wprowadzenie

W ramach projektu zostało opracowane rozszerzenie zestawu funkcji SQL hurtowni danych Netezza udostępniające użytkownikowi funkcje pozwalające na budowę modelu predykcyjnego oraz jego wykorzystanie.

System Netezza jest wysokowydajnym, specjalistycznym urządzeniem realizującym funkcjonalności hurtowni danych. Stosowany jest w przedsiębiorstwach do analiz biznesowych oraz analiz predykcyjnych i planowania.

Funkcjonalności predykcyjne systemu Netezza są bardzo rozbudowane dzięki narzędziom analitycznym, które czerpią dane z Relacyjnego Systemu Bazy Danych (RSBD), który jest wbudowanym element systemu. Podstawowa konfiguracja systemu Netezza nie umożliwia przetwarzania predykcyjnego z poziomu RSBD. Przetwarzanie tego typu jest możliwe po rozszerzeniu systemu o funkcje analityczne. Opis funkcjonalności dostępny jest w pozycji literaturowej [NIAG13].

1.2. Cel i założenia projektowe

Celem projektu jest udostępnienie w środowisku SQL systemu Netezza wybranej implementacji drzewa decyzyjnego.

Przyjęte założenia:

1. Jeżeli to będzie możliwe należy wykorzystać język Python.
2. Implementacja odbędzie się w środowisku Netezza Software Emulator 7.2.1.

1.3. Konstrukcja pracy

Praca składa się z rozdziałów przedstawiających opis środowiska uruchomieniowego, projektu oraz interfejsu udostępnianego przez SQL w postaci funkcji. Dla powtarzalności uruchomienia opisy są opatrzone licznymi instrukcjami w formie listy kroków. Przykładowa lista kroków wygląda następująco:

Krok 1. To jest krok pierwszy, który składa się z mniejszego kroku.

Krok 1.1. Mniejszy krok 1.

Krok 2. To jest krok drugi.

Dla pełnego zrozumienia podawane są polecenia powłoki systemowej. Przykładowe polecenie wygląda następująco:

```
$ echo „Hello world”;
```

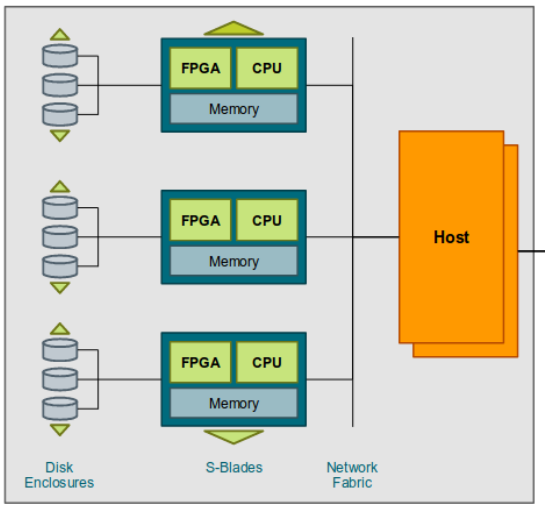
Praca zawiera również przykłady w języku SQL wykonywane z pomocą interfejsu programu nzsql. Przykładowe polecenie SQL wygląda następująco:

```
sql=> SELECT * FROM pracownicy;
```

Niekiedy w pracy podawane są oczekiwane wyniki działania programów. Są one sformatowane podobnie do dwóch powyższych:

```
To jest oczekiwany wynik działania programu.
```

2. Opis środowiska Netezza



Rysunek 1: Architektura systemu [IPS14]

System Netezza jest zbudowany z serwera zwanego Hostem, jednostek S-Blades oraz dysków twardych.

Host - serwer wyposażony w procesor o typowej architekturze x86-64. Zadaniem Hosta jest zarządzanie całym systemem oraz przetwarzaniem danych. Utrzymuje on podstawowe oprogramowanie bazodanowe oraz programy narzędziowe dla administratora jak serwer SSH lub programy nz* np. nzsql. Dodatkowym zadaniem hosta jest przetwarzanie niektórych podprogramów bazy danych oraz zdalnych programów analitycznych. W przypadku środowiska Netezza Software Emulator 7.2.1 zwanym dalej NSE utrzymywany jest system Red Hat Enterpriese Linux Server release 6.4 [nhos16] bez dostępu do zdalnych repozytoriów pakietów systemu.

S-Blade(SPU) [sbs16] - jest to komputer wyposażony w dwa procesory przy czym, na rzecz realizacji zapytań bazy danych przeznaczony jest procesor w architekturze FPGA. S-Blade utrzymuje własny system całkowicie zarządzany przez Host. System S-Blade jest całkowicie transparentny dla programisty. Współdzieli z Hostem fragment systemu plików umożliwiając w ten sposób łatwe dostarczanie bibliotek oraz podprogramów potrzebnych do realizacji zapytań SQL.

Dyski twarde [mnd16] - tworzą nadmiarową macierz dyskową w celu zwiększenia odporności na awarie. Służą one do składowania danych bazy danych. Dla zwiększenia wydajności systemu dane są partycjonowane na przestrzeni dysków oraz zorganizowane logicznie w tak zwane data slice [mds16]. **Data slice** jest logiczną reprezentacją partycji danych. Za organizację danych oraz ich przetwarzanie odpowiadają S-Blades.

Network fabric - jest siecią łączącą S-Blades z Hostem. Jest ona całkowicie transparentna z punktu widzenia programisty.

SQL [nsi16] - dostęp do bazy danych Netezza jest realizowany z pomocą języka ANSI SQL-92 z wybranymi rozszerzeniami SQL-1999 oraz SQL-2003. Język ten rozszerzać można z pomocą UDX.

UDX [inuf16] - są to podprogramy zarejestrowane w bazie danych jako funkcje z implementacją w postaci programu C++. W zależności od sposobu wykorzystania, rozróżniamy UDF (Funkcje skalarne), UDTF (Funkcje tablicowe), UDA (Funkcje agregujące), UDSL (Biblioteka współdzielona). System Netezza udostępnia specjalne kompilatory oraz narzędzia sterujące kompilacją. Ponieważ zapytanie SQL docelowo wykonane zostanie w SPU, system kompiluje programy dla architektury x86-64 (na rzecz hosta) oraz wymaganej przez S-Blade.

Sposób implementacji oraz API opisuje IBM Netezza User-Defined Functions Developer's Guide. [UDFG12]

Domyślnie UDX wykonywany jest równolegle na każdym z S-Blade. Wymusza to na programiście specjalne podejście podczas projektowania rozszerzenia a zwłaszcza założenie natury równoległej programu. Jedynym wyjątkiem są UDTF, które mają możliwość uruchomienia w trybie "noparallel". W tym trybie UDTF w całości program jest realizowany na hoście.

AE (Analytic Executables) - jest to koncepcja rozszerzenia języka SQL Netezza będąca ewolucją UDX. Stworzona została w celu łatwiejszej implementacji funkcji analitycznych w tym i algorytmów Data-mining. System Netezza może zostać wyposażony w dodatkowy pakiet programów analitycznych wykorzystujący mechanizm AE. Proces analizy danych wymaga implementacji algorytmu łatwego w modyfikowaniu oraz testowaniu. W tym celu AE udostępnia adaptery różnych języków programowania (C, C++, Java, Fortran, Python, Perl, R), api AE w wybranych językach, interpretery oraz biblioteki charakterystyczne dla wybranych języków skompilowane w architekturze x86-64 oraz FPGA. Program AE, aby poprawnie komunikować się z bazą danych oraz dla poprawnej realizacji cyklu życia programu musi korzystać z api AE, które udostępnia podstawowe funkcje opisujące programowi środowisko, dostarcza przetwarzanych danych oraz (co najważniejsze) realizuje podział UDX na UDF, UDTF, UDSL, UDA. Program AE może zostać wykonywany lokalnie (baza danych zarządza wykonaniem oraz dostarczaniem danych programu na S-Blade) oraz zdalnie (odpowiedzialność zarządzania wykonaniem spada na administratora systemu). W przypadku programu AE typu UDTF również występuje dodatkowy tryb noparallel.

Opis koncepcji wraz z przykładami znajduje się w dokumencie IBM Netezza Analytics User-Defined Analytic Process Developer's Guide. [UDAG14]

Domyślnie mechanizm AE nie jest dostępny w systemie. Należy go zainstalować. Dostępny jest w ramach pakietu INZA.

W ramach mechanizmu AE dostępny jest adapter języka Python wraz z interpreterem w wersji 2.6.6 wraz z podstawowymi bibliotekami. Niestety NSE zawiera uszkodzoną wersję interpretera. Opis problemu oraz rozwiązania jest dostępny w punkcie 6.3. oraz 6.4.

3. Opis implementacji

3.1. Rozwój projektu

Pierwotnym planem realizacji projektu było wykorzystanie pakietu modułów SciKit-Learn popularnego w języku Python. Nie jest on jednak dostępny w ramach interpretera dystrybuowanego w NSE. Aby zainstalować bibliotekę SciKit-Learn należy zainstalować pakiety SciPy oraz NumPy. Pakiety do działania wymagają instalacji w systemie biblioteki LAPACK. W celu skompilowania LAPACK należy skompilować bibliotekę BLAS. LAPACK oraz BLAS są częściowo napisane w języku Fortran. Pomocny okazał się zestaw narzędzi dostępny w systemie Netezza gdzie znajduje się przygotowany kompilator zarówno dla Host jak i SPU. W trakcie realizacji instalacji bibliotek rozwiązaliśmy problemy z pkt. 6.3 oraz 6.4 oraz opracowaliśmy proces instalacji przedstawiony w pkt 4.1.1 oraz 4.1.2. Cały proces instalacji bibliotek oraz implementacji rozwiązań przytoczonych problemów zawarliśmy w rozbudowanym skrypcie języka BASH, który przeprowadza instalację samodzielnie, sterowaną decyzjami użytkownika o instalacji poszczególnych elementów.

Algorytmy zaimplementowane w ramach pakietu SciKit-Learn okazały się, niestety, nieprzydatne do realizacji projektu. Drzewa decyzyjne nie są przystosowane do pracy w aplikacjach równoległych, co więcej przetrzymują one cały zbiór trenujący w pamięci operacyjnej co w przypadku systemów hurtowni danych może stanowić wąskie gardło systemu. Podobnym sposobem odrzuciliśmy pozostałe dostępne algorytmy takie jak np. sieci neuronowe. Jedynym algorytmem z pakietu, który jest odpowiedni dla środowiska Netezza okazuje się algorytm regresji liniowej jednak ten nie spełnia założeń projektu.

Ostatecznie zdecydowaliśmy się na algorytm Hoeffding Tree (VFDT - Very Fast Decision Tree) znany z pakietu analitycznego Weka. Ze względu na język zastosowana została implementacja znaleziona w portalu github.com [git00] będąca implementacją wzorowaną na pakiecie Weka. Implementacja została wykonana przez autora w Python3. Zawierała błąd uniemożliwiający poprawną serializację oraz deserializację drzewa decyzyjnego. Implementację przystosowaliśmy do uruchomienia przez Python w wersji 2.6 oraz poprawiliśmy wymienione problemy. Błąd polegał na porównywaniu referencji łańcuchów znaków zamiast ich wartości. W przypadku uruchomienia algorytmu w pojedynczej instancji programu interpreter nadaje ten sam identyfikator łańcuchom, co do wartości identycznym, stąd w przypadku wykonania w środowisku sekwencyjnym problem nie

występował. Różne identyfikatory pojawiają się w przypadku równoległego wykonywania operacji deserializacji/serializacji oraz wykonania szczególnych funkcji języka Python kreujących dane.

3.2. Opis funkcjonalności

Aplikacja w systemie Netezza udostępnia funkcjonalność predykcji klasy decyzyjnej, którego wytrenowanie również jest możliwe w systemie Netezza z pomocą języka SQL. Aplikacja umożliwia wyczyszczenie klasyfikatora, wykonanie próbkowania danych, wytrenowanie klasyfikatora oraz dokonanie predykcji z pomocą trzech różnych funkcji, do wyboru w zależności od potrzeb użytkownika.

Zastosowany algorytm Hoeffding Tree jest wariantem przyrostowych drzew decyzyjnych, inaczej zwanym VFDT. Wykorzystywany jest do budowania modeli predykcyjnych na dużych zbiorach danych, takich jak strumienie danych (które z założenia są nieskończone).

3.2.1. Czyszczenie klasyfikatora

Ze względu na konieczność składowania klasyfikatora, udostępniona została funkcjonalność czyszczenia klasyfikatora. W obecnej chwili aplikacja umożliwia istnienie jednego klasyfikatora w bazie danych. (/nz/export/ae/applications/model.pickle) Operacja czyszczenia jest uruchamiana również podczas próbkowania.

3.2.2. Próbkowanie danych

Celem operacji jest zmapowanie wartości nominalnych na liczbowe, akceptowalne przez algorytm VFDT. Próbka danych jest przechowywana w pamięci operacyjnej. Nie powinna więc być zbyt duża. Powinna ona zawierać wszystkie możliwe wartości każdego z atrybutów nominalnych, a przede wszystkim etykiet klas. Próbka danych zostanie dołączona do zbioru trenującego. Nie ma więc potrzeby włączać jej ręcznie.

3.2.3. Trenowanie klasyfikatora

W tym etapie następuje zbudowanie modelu w oparciu o zadany zbiór testujący powiększony o próbkę danych przechowywaną w pamięci operacyjnej.

3.2.4. Predykcja (sekwencyjna, równoległa UDTF, równoległa UDF)

Predykcja sekwencyjna - funkcjonalność predykcji zrealizowana z pomocą AE typu UDTF --noparallel, a więc całość odbywa się w pamięci operacyjnej Hosta. Funkcja może być przydatna podczas debugowania aplikacji.

Predykcja równoległa z wykorzystaniem UDTF - przetwarzanie odbywa się w SPU równolegle, co przyspiesza dodatkowo operację. W wyniku operacji dane są zorganizowane w tabeli (generowanej „w locie"), które można z łatwością grupować, sortować oraz przetwarzać z pomocą tych samych narzędzi, co standardowe tabele.

Predykcja równoległa z wykorzystaniem UDF - język SQL jest elastyczny. Predykcja z pomocą AE typu UDF może być użyteczna do aktualizowania pól tabel w bazie danych oraz generowania nowych danych, co zwiększa obszar zastosowań aplikacji.

3.2.5. Parametryzacja algorytmu

Parametry programu klasyfikującego zastosowanego w opracowanym rozwiązaniu są inicjowane poniższymi wartościami. Opis dostępny jest w pozycji literaturowej [weka01].

grace_period
Wartość: 50
Znaczenie: Liczba instancji liścia drzewa decyzyjnego do obserwacji pomiędzy próbami podziału.

h_tie_threshold
Wartość: 0.05
Znaczenie: Próg odcięcia, poniżej którego następuje przerwanie wiązania w drzewie decyzyjnym.

split_confidence
Wartość: 0.0001
Znaczenie: Dopuszczalny błąd w decyzji o podziale w drzewie decyzyjnym.

minimum_fraction_of_weight_info_gain
Wartość: 0.01
Znaczenie: Minimalny ułamek wag wymagany w rozgałęzieniach do podziału zysku informacyjnego.

3.4. Struktura projektu

Projekt udostępniony jest w repozytorium git w portalu github.com [git19-2]. Aplikacja składa się z następujących elementów.

testree.py – skrypt Python prezentujący użycie biblioteki HoeffdingTree w klasycznym programie sekwencyjnym Hosta. Jego uruchomienie pozwoli również na przetestowanie poprawnego działania.

testpickle.py – skrypt Python prezentujący użycie biblioteki cPickle w klasycznym programie sekwencyjnym Hosta. Jego uruchomienie pozwoli również na przetestowanie poprawnego działania serializacji/deserializacji.

compile.sh – skrypt BASH rejestrujący program AE w schemacie bazy danych.

put_ht.py – kod główny programu AE udostępniającego interfejs klasyfikatora w SQL.

Classifier.py – klasa wykorzystywana przez program AE do sterowania procesem budowania i wykorzystywania modelu predykcyjnego.

vendor/HoeffdingTree – biblioteka implementująca algorytm drzew decyzyjnych VFDT w języku Python na wzór analogicznej biblioteki programu Weka w języku Java [git00].

example_data/generate.py – skrypt Python generujący zbiory danych w formie pliku SQL gotowego do importu. Na tych danych można przetestować działanie programu.

4. Instrukcja użytkownika

4.1. Przygotowanie środowiska pracy

4.1.1. Instalacja pakietu INZA

Środowisko Netezza Software Emulator zawiera pliki instalacyjne pakietu INZA w katalogu /nz/inza_install. Instalacja pakietu odbywa się poprzez instalator inzaPackageInstaller.sh. Do poprawnej instalacji pakietu, skrypt instalacyjny wymaga odpowiedzi na wiele pytań konfiguracyjnych. Poniżej prezentujemy odpowiedzi zapewniające poprawne działanie projektu, które zostały opracowana na podstawie pozycji literaturowej [inas16].

Krok. 1. Wykonaj polecenia:

```
$ cd /nz/inza_install

$ ./inzaPackageInstaller.sh
```

Krok. 2. Na kolejno pojawiające się pytania udziel poniższe odpowiedzi:

Krok. 2.1. Install INZA packages? (y/n):
Odpowiedź: y (wymagane)

Krok. 2.2. Install Documentation packages? (y/n):
Odpowiedź: n

Krok. 2.3. Please review the packages to install:
1) INZA package: YES
2) INZA Documentation package: NO
Do you wish to install the above selections?
Enter "y" to continue, "x" to exit or any other key to be prompted to modify your selection:
Odpowiedź: y

Krok. 2.4. Installing INZA packages…

Available zipped installation file(s):

[0] /export/home/nz/inza/v2.5.4/inza-2.5.4.zip

[1] A zipped file in a different directory or with a non-standard name.

Enter your selection:

Odpowiedź: 0

Krok. 2.5. Would you like to run the INZA cartridge installer now? (y/n):

Odpowiedź: y (wymagane)

Krok. 2.6. Would you like to perform an Express (e) or Custom (c) install (e/c):

Odpowiedź: c

Krok. 2.7. Install MapReduce components? (y/n):

Odpowiedź: n

Krok. 2.8. Install Matrix components?

Note: Matrix components are also required for PCA, Kmeans, GLM and linear regression.

Odpowiedź: n

Krok. 2.9. Install IBM Netezza In-database Analytics components? (y/n):

Odpowiedź: n

Krok. 2.10. Install Spatial components? (y/n):

Odpowiedź: n

Krok. 2.11. Please review the components to install:

1) MapReduce: NO

2) Matrix: NO

3) IBM Netezza In-database Analytics: NO

4) Spatial: NO

Do you wish install the above selections?

Enter "y" to continue, "x" to exit or any other key to be prompted to modify your selection:

Odpowiedź: y (wymagane)

Krok. 2.12. Po udzieleniu odpowiedzi w poprzednim kroku skrypt rozpocznie instalację. Początkowo postęp nie jest sygnalizowany komunikatami.

Krok. 2.13. Would you like to re-enable all databases that are enabled for IBM Netezza Analytics? (y/n):

Odpowiedź: y

Krok. 2.14. Would you like to update all spatial databases that already have spatial registered with the current version? (y/n):

Odpowiedź: y

Krok. 3. Wykonaj polecenie

```
$ source ~/.bashrc
```

4.1.2. Instalacja bibliotek

W trakcie opracowywania rozwiązania przygotowany został instalator podstawowych bibliotek Python. Dodatkowo instalator naprawia problemy z biblioteką zlib oraz wariantami algorytmu SHA w dostępnym środowisku języka Python. W celu uniknięcia niespodziewanych problemów w pracy z aplikacją zalecana jest instalacja wszystkich bibliotek.

Lista instalowanych bibliotek

1. Lapack 3.8.0 (na rzecz instalacji NumPy).
2. Setuptools (biblioteka instalatorów pakietów Python).
3. PIP (narzędzie zarządzania pakietami Python).
4. NumPy 1.10.4.
5. SciPy 0.18.1 (prezentuje się w systemie jako wersja 1.13).
6. SciKit-Learn 1.16.1.

Uwagi

- Operację należy wykonać po instalacji INZA (patrz pkt. 4.1.1)
- Jeżeli bash poinformuje o nieznalezieniu "/bin/shell^...." należy skorygować znaki końca wierszy w skryptach.
- W podkatalogach instalatora dostępne są pliki z logami (katalogi ./logs oraz ./subscripts/logs)
- Instalację można wznowić uruchamiając ponownie instalator. Każdy krok, który został w poprzedniej próbie pomyślnie wykonany można pominąć.

- Skrypt może być obarczony drobnymi błędami zwłaszcza w obsłudze nieoczekiwanych błędów kompilacji itp.
- Wymagany jest dostęp do internetu w celu pobrania odpowiednich bibliotek. Pobieranie wykonywane jest automatycznie.
- Instalator przerwie pracę w przypadku wystąpienia przewidzianych błędów (np. brak skompilowanej biblioteki po operacji kompilacji).

Instrukcja instalacji

Krok. 1. Pobierz oraz wgraj pliki instalatora dostępne w [git19] do katalogu /export/home/nz/nz_scripts w systemie Netezza.

Krok. 2. Wykonaj poniższe polecenia w celu uruchomienia instalatora:

```
$ su root
<wpisz hasło użytkownika root - domyślnie “netezza”>
$ cd /export/home/nz/nz_scripts
$ chmod +x main_installer.sh
$ mkdir /export/home/nz/instalation
$ ./main_installer.sh install /export/home/nz/instalation
```

Krok. 3. Udziel odpowiedzi „y” na kolejno pojawiające się pytania.

Krok. 4. Po pomyślnej instalacji zakończ pracę poleceniem

```
$ exit
```

W przypadku potrzeby wznowienia instalacji w wyniku błędu odpowiedz „s” w etapach instalacji, które w poprzedniej próbie zakończyły się powodzeniem.

Instrukcja czyszczenia instalatora

Krok. 1. Wykonaj poniższe polecenia:

```
$ su root
<wpisz hasło użytkownika root - domyślnie “netezza”>
$ cd /export/home/nz/nz_scripts
$ ./main_installer.sh clean
```

Instrukcja uzyskiwania opisu instalatora

Krok. 1. Wykonaj poniższe polecenia:

```
$ su root
<wpisz hasło użytkownika root - domyślnie “netezza”>
$ cd /export/home/nz/nz_scripts
$ ./main_installer.sh help
```

4.1.3. Instalacja opracowanej aplikacji

Krok. 1. Pobierz oraz wgraj pliki aplikacji dostępne w [git19-2] do katalogu /export/home/nz/predykcja w systemie Netezza.

Krok. 2. Utwórz schemat bazy danych w którym zostanie zarejestrowana aplikacja

```
$ nzsql -c "CREATE DATABASE <nazwa schematu>;"
```

Krok. 3. Przejdź do katalogu projektu:

```
$ cd /export/home/nz/predykcja
```

Krok. 4. Nadaj uprawnienia wykonania skryptu rejestrującego aplikację w schemacie:

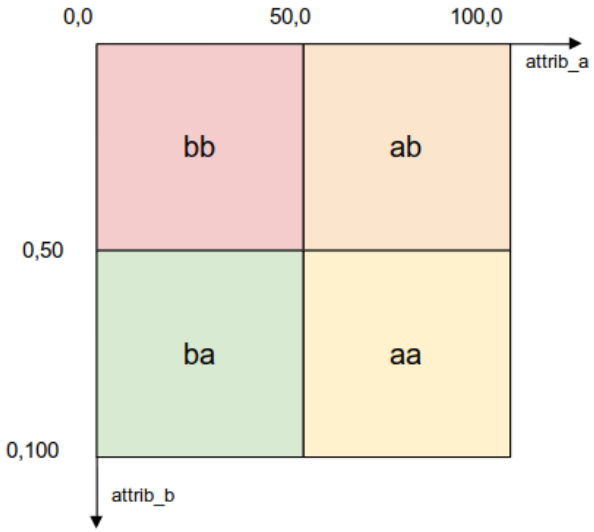
```
$ chmod +x compile.sh
```

Krok. 5. Uruchom skrypt. W schemacie zostaną zarejestrowane nowe funkcje predykcyjne wymienione w rozdziale 4.2.

```
$ ./compile.sh <nazwa schematu>
```

4.1.4. Przykładowe dane – generowanie oraz ładowanie do RSBD

Przykładowe dane są generowane przez dostarczony w projekcie program [git19-2, generate.py]. Zawierają zbiór próbkowany, zbiór trenujący oraz zbiór testujący (aby wykonać pomiar trafności klasyfikatora należy go zmodyfikować). Zbiory składają się z dwóch atrybutów numerycznych oraz etykiety będącej atrybutem nominalnym (poza zbiorem testującym). Etykiety tworzą 4 klasy. Atrybuty numeryczne to liczby pseudolosowe z zakresu 0..100. Przydział do klas jest zgodny z poniższym wykresem. Wartości na granicy są przydzielane do klasy powyżej progu.



Wykres 1: Podział przykładowych danych na klasy decyzyjne.

Składnia wywołania programu generującego dane do pliku ./example_data/data.sql:

```
python ./example_data/generate.py <DATABASE_SCHEMA_NAME>
<optional:probe set size> <optional:train set size> <optional:test set size>
```

Przykładowe wywołanie:

```
$ python generate.py IRIS 3000 6000 100
```

Ładowanie danych do RSBD. Dane zostaną załadowane do tabel IRIS.example_probe IRIS.example_train, IRIS.example_test

```
$ nzsql -f ./example_data/data.sql
```

4.2. Funkcje SQL udostępniane przez aplikację

4.2.1. Próbkowanie (UDTF)

Składnia: PUT_HT_CREATE_AND_PROBE(VARARGS)

Typ zwracany: TABLE(debug VARCHAR(1024))

Uwagi: VARARGS - atrybuty, które będą przetwarzane przy czym ostatni jest etykietą klasy. Zwracana jest informacja Probe, index=numer rekordu, który trafił do instancji modelu.

4.2.2. Trenowanie (UDTF)

Składnia: PUT_HT_TRAIN(VARARGS)

Typ zwracany: TABLE(debug VARCHAR(1024))

Uwagi: VARARGS - atrybuty, które będą przetwarzane przy czym ostatni jest etykietą klasy. Jeżeli predykcja nie została jeszcze uruchomiona można dokonać trenowania wielokrotnie na różnych zbiorach trenujących. Każdy kolejny będzie rozbudowywał model klasyfikatora. Zwracana jest informacja Train, index=numer rekordu, który trafił do instancji modelu.

4.2.3. Predykcja

Opis funkcjonalny poniższych metod został przedstawiony w rozdziale 3.2.

Predykcja sekwencyjna

Składnia: PUT_HT_PREDICT_SEQ(VARARGS)
Typ zwracany: TABLE(predicted_class VARCHAR(1024))
Uwagi: VARARGS - atrybuty, które będą przetwarzane. Zwracana jest przewidywana klasa.

Predykcja równoległa (UDTF)

Składnia: PUT_HT_PREDICT(VARARGS)
Typ zwracany: TABLE(predicted_class VARCHAR(1024))
Uwagi: VARARGS - atrybuty, które będą przetwarzane. Zwracana jest przewidywana klasa.

Predykcja równoległa (UDF)

Składnia: PUT_HT_PREDICT_S(VARARGS)
Typ zwracany: VARCHAR(1024)
Uwagi: VARARGS - atrybuty, które będą przetwarzane. Zwracana jest przewidywana klasa.

Usuwanie klasyfikatora (UDTF)

Składnia: PUT_HT_CLEAN(BOOL)
Typ zwracany: TABLE(debug VARCHAR(1024))
Uwagi: funkcja jest wykonywana jeżeli argumentem będzie wartość true. W przypadku powodzenia zwracana jest informacja “OK”.

4.3. Przykład użycia aplikacji

Poniższa instrukcja przedstawia wywołania poszczególnych funkcji udostępnianych w SQL RSBD systemu Netezza. Wykonanie kolejnych kroków doprowadzi do zbudowania modelu predykcyjnego oraz jego zastosowania na danych testowych, które można wygenerować zgodnie z instrukcją w rozdziale 4.1.4, w efekcie których powstają tabele example_probe (zbiór danych do próbkowania typów atrybutów), example_train (zbiór danych trenujących model), example_test

(zbiór danych testowych). Dla jasności przykładowych wywołań tabele zostały użyte w poniższej instrukcji.

Krok 1. Przejdź do narzędzia SQL Netezza:

```
$ nzsql
```

Krok 2. Przełącz aktywny schemat RSBD:

```
sql=> SET CATALOG <nazwa schematu>;
```

Krok 3. Zainicjuj tworzenie modelu predykcyjnego. Konstrukcja „TABLE WITH FINAL(...)" jest niezbędna w systemie Netezza do uruchomienia UDTF. Jako argument przyjmuje wywołanie UDTF zwracające dane typu „table" które następnie traktuje jako standardową relację RSBD tylko do odczytu. Iloczyn kartezjański tych dwóch relacji udostępnia dane z relacji example_probe w UDTF o nazwie PUT_HT_CREATE_AND_PROBE [UDFG12, s. 3-6].

```
sql=> SELECT * FROM example_probe, TABLE WITH FINAL
( PUT_HT_CREATE_AND_PROBE( attrib_a, attrib_b, class) ) ;
```

```
IRIS.ADMIN(ADMIN)=> SELECT * FROM example_probe, TABLE WITH FINAL ( PUT_HT_CREAT
E_AND_PROBE( attrib_a, attrib_b, class) ) ;
 ATTRIB_A | ATTRIB_B | CLASS |      DEBUG
----------+----------+-------+-----------------
       83 |       36 | ab    | Probe, index=1
       17 |       34 | bb    | Probe, index=2
       92 |       95 | aa    | Probe, index=3
        4 |       25 | bb    | Probe, index=4
       34 |       68 | ba    | Probe, index=5
       56 |       37 | ab    | Probe, index=6
       71 |        0 | ab    | Probe, index=7
       54 |       78 | aa    | Probe, index=8
       28 |       71 | ba    | Probe, index=9
       29 |       57 | ba    | Probe, index=10
       37 |       33 | bb    | Probe, index=11
```

Rysunek 2: Przykładowy rezultat wywołania funkcji PUT_HT_CREATE_AND_PROBE

Krok 4. Zbuduj model predykcyjny w oparciu o zbiór trenujący.

```
sql=>  SELECT * FROM example_train, TABLE WITH FINAL
( PUT_HT_TRAIN( attrib_a, attrib_b, class) );
```

```
IRIS.ADMIN(ADMIN)=> SELECT * FROM example_train, TABLE WITH FINAL ( PUT_HT_TRAIN
( attrib_a, attrib_b, class) );
 ATTRIB_A | ATTRIB_B | CLASS |       DEBUG
----------+----------+-------+-------------------
       42 |       91 | ba    | Train, index=3001
       70 |       26 | ab    | Train, index=3002
       67 |       67 | aa    | Train, index=3003
       34 |       47 | bb    | Train, index=3004
       79 |       71 | aa    | Train, index=3005
       33 |       60 | ba    | Train, index=3006
       63 |       48 | ab    | Train, index=3007
       36 |       30 | bb    | Train, index=3008
        0 |       80 | ba    | Train, index=3009
       26 |        2 | bb    | Train, index=3010
       11 |       92 | ba    | Train, index=3011
       10 |       59 | ba    | Train, index=3012
       53 |       59 | aa    | Train, index=3013
        1 |       35 | bb    | Train, index=3014
```

Rysunek 3: Przykładowy rezultat wywołania funkcji PUT_HT_TRAIN

Krok 5. W celu wykonania predykcji wykonaj jedno z poniższych poleceń.

Wersja sekwencyjna

```
sql=> SELECT * FROM example_test, TABLE WITH FINAL
( PUT_HT_PREDICT_SEQ( attrib_a, attrib_b) );
```

Wersja równoległa UDTF (Patrz rozdział 4.2.3. oraz 3.2.)

```
sql=> SELECT * FROM example_test, TABLE WITH FINAL
( PUT_HT_PREDICT( attrib_a, attrib_b) );
```

Wersja równoległa UDF (Patrz rozdział 4.2.3. oraz 3.2.)

```
sql=> SELECT *, PUT_HT_PREDICT_S( attrib_a, attrib_b) FROM
example_test;
```

```
IRIS.ADMIN(ADMIN)=> SELECT *,PUT_HT_PREDICT_S( attrib_a, attrib_b) FROM example_
test;
 ATTRIB_A | ATTRIB_B | PUT_HT_PREDICT_S
----------+----------+------------------
       60 |       48 | ab
       39 |       21 | bb
       63 |       74 | aa
       72 |       75 | aa
       10 |       24 | bb
       27 |       86 | ba
        7 |        9 | bb
       42 |       53 | ba
        0 |       67 | ba
       77 |       57 | aa
       67 |       70 | aa
       84 |       55 | aa
       71 |       82 | aa
       56 |       32 | ab
       99 |        9 | ab
       29 |       24 | bb
```

Rysunek 4: Przykładowy rezultat wywołania funkcji PUT_HT_PREDICT_S. Atrybut PUT_HT_PREDICT_S zawiera klasę predyktowaną.

Krok 6. Krok opcjonalny. Czyszczenie modelu predykcyjnego. Model zostanie usunięty po jawnym podaniu true jako argumentu funkcji.

```
sql=> SELECT * FROM TABLE WITH FINAL ( PUT_HT_CLEAN(true) );
```

```
IRIS.ADMIN(ADMIN)=> SELECT * FROM TABLE WITH FINAL ( PUT_HT_CLEAN(true) );
 STATE
-------
 OK
(1 row)
```

Rysunek 5: Przykładowy rezultat wywołania funkcji PUT_HT_CLEAN.

5. Kierunki rozwoju

W aplikacji widzimy możliwe kierunki rozwoju:

Kierunek 1. Refaktoryzacja Classifier.py w celu zwiększenia łatwości zrozumienia aplikacji

Kierunek 2. Dodanie funkcji SQL umożliwiającej sparametryzowanie programu klasyfikującego w zakresie parametrów grace_period, h_tie_threshold, split_confidence, minimum_fraction_of_weight_info_gain opisanych w rozdziale 3.2.4.

Kierunek 3. Dodanie funkcjonalności egzystencji wielu klasyfikatorów w ramach jednego schematu bazy danych.

Kierunek 4. Przetestowanie aplikacji w przypadku awarii (np. odcięcie prądu w trakcie klasyfikacji) oraz wdrożenie ewentualnych poprawek.

Kierunek 5. Dodanie walidacji argumentów sterujących pracą programu.

Kierunek 6. Wyodrębnienie implementacji algorytmu predykcyjnego oraz umożliwienie wyboru wersji implementacji poprzez parametr. Dzięki tej zmianie będzie możliwe zastosowanie innych algorytmów w ramach tego samego interfejsu.

6. Rozwiązania typowych problemów

6.1. SPU nie przechodzi w stan Online, Netezza jest w stanie Discovering.

Problem z inicjalizacją SPU napotykaliśmy każdorazowo przy uruchamianiu nowej maszyny wirtualnej NSE. Problem występuje niezależnie od systemu operacyjnego gospodarza (Windows 10 / Linux) jak i rodziny procesorów (AMD/Intel). Po wykonaniu poniższej instrukcji dla bezpieczeństwa maszynę wirtualną należy zapisywać zamiast restartować.

W terminalu Hosta należy wykonać poniższe polecenia [ibms16]:

```
$ nzsystem pause
$ nzpush -id 1003 power cycle
$ nzpush -a status -s
```

Należy odczekać aż powyższe operacje zwrócą wynik podobny do poniższego:

```
spu0103: SPU: hx5 sn: <sn> net: bond0 (UP) uptime: 2 min nps: ready
spu0101: SPU: hx5 sn: <sn> net: bond0 (UP) uptime: 14 min nps: ready)
```

Następnie:

```
$ nzsystem resume
```

Id SPU można sprawdzić za pomocą polecenia:

```
$ nzhw
```

6.2. Parametr „--noparallel” jest ignorowany dla AE typu UDTF.

Programy AE rejestrowane wg schematu UDTF w języku Python, mimo wielokrotnego przywołania w dokumentacji IBM możliwości użycia --noparallel nie obsługują go! (testowane w wersji NSE 7.2.1) Błąd znajduje się w wewnętrznym szablonie mechanizmu AE rejestracji programu AE typu Python.

Szablon należy skopiować, tworząc tym samym nowy. W kopii należy wprowadzić poniższą poprawkę, a następnie nową wersję wykorzystywać podczas rejestracji programów w bazie danych z wykorzystaniem nowego szablonu.

Krok. 1. Kopiowanie

```
$ cp /nz/export/ae/adapters/python/3/templates/udtf
/nz/export/ae/adapters/python/3/templates/hudtf
```

Krok. 2. Edycja pliku hudtf

Pod wierszem “API VERSION 2” w nowym wierszu wstawić należy “[% ae_parallel %]”

6.3. Python - brak implementacji SHA

Błąd może pojawić się podczas korzystania z dodatkowych bibliotek, a zwłaszcza instalowanych przez menadżer pakietów PIP, które do kontroli wersji wykorzystują algorytm SHA. Python posiada funkcję ładującą biblioteki, która uwzględnia algorytmu SHA mimo, że wraz z interpreterem w systemie Netezza nie są one dystrybuowane. Rozwiązaniem jest zakomentowanie w funkcji ładującej algorytmu SHA.

Należy wyedytować plik /nz/export/ae/languages/python/2.6/spu/lib/python2.6/hashlib.py w funkcji __get_builtin_constructor (wiersze około 58-78) zakomentować należy wiersze 65-78.

Naprawę realizuje przygotowany przez skrypt instalacyjny bibliotek Pythona (patrz punkt 4.1.2).

6.4. Python - brak biblioteki zlib

Błąd może pojawić się podczas korzystania z niektórych bibliotek Pythona dostępnych w systemie Netezza podczas ich importowania jako modułów języka Python w programach wykonywanych przez SPU, np. programu AE. Błąd zgłaszany jest w postaci wyjątku:

```
ImportError: No module named zlib
```

Jak się okazuje interpreter dostępny w NSE udostępnia bibliotekę zlib dla programów Hosta , ale dla SPU już nie. Aby naprawić błąd można wykorzystać dostępną bibliotekę w wersji interpretera przeznaczonej dla hosta, przy okazji uzupełniając interpreter hosta o biblioteki współdzielone funkcji skrótu.

Naprawę można wykonać stosując następujące polecenia:

```
$ cp /nz/export/ae/languages/python/2.6/host/lib/python2.6/lib-dynload/
zlib.so /nz/export/ae/languages/python/2.6/spu/lib/python2.6/lib-dynload/
```

```
$ cp
/nz/export/ae/languages/python/2.6/spu/lib/python2.6/lib-dynload/_md5.so /nz/
export/ae/languages/python/2.6/host/lib/python2.6/lib-dynload/

$ cp
/nz/export/ae/languages/python/2.6/spu/lib/python2.6/lib-dynload/_sha.so /nz/
export/ae/languages/python/2.6/host/lib/python2.6/lib-dynload/
```

Literatura

[NIAG13] IBM Netezza In-Database Analytics Reference Guide. IBM, 2013. [on-line] https://www.ibm.com/developerworks/community/files/form/anonymous/api/library/1b6a2624-dc86-4856-b4ed-cdda6bfdecda/document/2651620f-4ce0-44f8-b558-d32300d16552/media/IBM_Netezza_In-Database_Analytics_Reference_Guide-3.0.1.pdf, 2013. Dostęp: 2019-01-20.

[IPS14] Phil Francisco. IBM PureData System for Analytics Architecture. IBM, 2014. [on-line] https://www.redbooks.ibm.com/redpapers/pdfs/redp4725.pdf, 2014. s.4. Dostęp: 2018-07-10.

[nhos16] Netezza host operating system. [on-line] https://www.ibm.com/support/knowledge center/en/SSULQD_7.2.1/com.ibm.nz.reln.doc/c_relnotes_host_os.html, 2016. Dostęp: 2018-07-10.

[sbs16] Snippet blades (S-Blades). [on-line] https://www.ibm.com/support/knowledgecenter/en/SSULQD_7.2.1/com.ibm.nz.dbu.doc/c_getstrt_s_blades.html, 2016. Dostęp: 2018-07-10.

[mnd16] Manage disks. [on-line] https://www.ibm.com/support/knowledgecenter/en/SSULQD_7.2.1/com.ibm.nz.dbu.doc/c_sysadm_managing_disks.html, 2016. Dostęp: 2018-07-10.

[mds16] Manage data slices. [on-line] https://www.ibm.com/support/knowledgecenter/en/SSULQD_7.2.1/com.ibm.nz.dbu.doc/c_dbuser_ntz_sql_introduction.html, 2016. Dostęp: 2018-07-10.

[nsi16] Netezza SQL introduction. [on-line] https://www.ibm.com/support/knowledgecenter/en/SSULQD_7.2.1/com.ibm.nz.dbu.doc/c_dbuser_ntz_sql_introduction.html, 2016. Dostęp: 2018-07-10.

[inuf16] IBM Netezza user-defined functions. [on-line] https://www.ibm.com/support/knowledgecenter/en/SSULQD_7.2.1/com.ibm.nz.udf.doc/c_udf_plg_overview.html, 2016. Dostęp: 2018-07-10.

[UDFG12] Netezza User-Defined Functions Developer's Guide. IBM, 2012. [on-line] https://www.ibm.com/developerworks/community/files/form/anonymous/api/library/1b6a2624-dc86-4856-b4ed-cdda6bfdecda/document/bae7dae4-8fc8-439d-a804-c9498dc09f18/media, 2012. Dostęp: 2018-07-10.

[UDAG14] IBM Netezza Analytics User-Defined Analytic Process Developer's Guide. IBM, 2014. [on-line] https://www.ibm.com/developerworks/community/files/form/anonymous/api/library/1b6a2624-dc86-4856-b4ed-cdda6bfdecda/document/13a30879-47ba-486f-864f-6a5b76c7421a/media, 2014. Dostęp: 2018-07-10.

[git00] https://github.com/vitords/HoeffdingTree. [on-line] , 2000. Dostęp: 2018-07-10.

[weka01] Class HoeffdingTree. [on-line] http://weka.sourceforge.net/doc.dev/weka/classifiers/trees/HoeffdingTree.html, . Dostęp: 2019-01-20.

[git19-2] Netezza InDatabase Prediction. [on-line] https://github.com/writ3it/HD_Netezza_in-database_prediction, 2019. Dostęp: 2019-01-27.

[inas16] Balaji Veeraraghavan, Ph.D.Installing NumPy and SciPy in IBM PDA. IBM,2013.

[git19] HD Python Libs Installation. [on-line] https://github.com/writ3it/HD_Python_Libs_Installation, 2019. Dostęp: 2019-01-27.

[git19-2, generate.py] Netezza InDatabase Prediction. [on-line] https://github.com/writ3it/HD_Netezza_in-database_prediction, 2019. Dostęp: 2019-01-27.

[UDFG12, s. 3-6] Netezza User-Defined Functions Developer's Guide. IBM, 2012. [on-line] https://www.ibm.com/developerworks/community/files/form/anonymous/api/library/1b6a2624-dc86-4856-b4ed-cdda6bfdecda/document/bae7dae4-8fc8-439d-a804-c9498dc09f18/media, 2012. Dostęp: 2018-07-10.

[ibms16] SPU crash when nzsystem resume after SPU power cycle. [on-line] http://www-01.ibm.com/support/docview.wss?uid=swg21695658, 2016. Dostęp: 2018-07-10.