XIII Konferencja PLOUG
Kościelisko
Październik 2007

Wykorzystanie interfejsu JAVA API do budowy narzędzi eksploracyjnych z wykorzystaniem Oracle Data Mining na platformie Oracle 10.2.0.1.

Robert Stępniak, Adam Pelikant
Instytut Mechatroniki i Systemów Informatycznych Politechniki Łódzkiej

rstepniak@gmail.com, apelikan@p.lodz.pl

Abstrakt. Artykuł zawiera informacje o sposobach tworzenia obiektów eksploracyjnych udostępnianych przez serwer bazy danych Oracle 10g z wykorzystaniem metod i klas języka programowania JAVA. Jest to jeden z dwóch dostępnych interfejsów API o analogicznych możliwościach jak interfejs PL/SQL API. Przedstawione również będą sposoby rozwiązania problemów konfiguracyjnych między systemami: Oracle i operacyjnym Windows XP w wersji polskiej.

Informacja o autorach. Profesor nadzw. dr hab. inż. Adam Pelikant: jedna z najbardziej doświadczonych osób w administracji i ob.-słudze systemów bazodanowych na Politechnice Łódzkiej. Przewodniczący i założyciel koła naukowego o tematyce baz danych, zrzeszającego studentów chcących rozwijać swoje umiejętności i zainteresowania na tym polu. Bardzo życzliwa osoba, która często bardziej wierzy w umiejętności swych podopiecznych niż oni sami. Zawsze skory do pomocy przy rozwiązywaniu nowych problemów studentów związanych z bazami danych. Zawsze przedstawia jasno swoje cele i oczekiwania względem danej pracy, pomimo że często są to trudne zagadnienia. Nastawiony na praktyczne stosowanie nowych narzędzi i technologii informatycznych. Jego dewizą są słowa Alberta Einsteina "Rozwiązuj wszystko prosto, ale nie prościej".
Mgr. inż. Robert Stępniak: absolwent Informatyki na wydziale Elektroniki, Elektrotechniki, Informatyki i Automatyki Politechniki Łódzkiej. Ukończył specjalizację Bazy danych i systemy ekspertowe w Instytucie Mechatroniki i Systemów Informatycznych.
Należał do koła naukowego Baz danych prowadzonego przez Profesora Adama Pelikanta, gdzie zajmował się rozwiązywaniem problemów związanych z tworzeniem "klientów" różnych baz danych – zwłaszcza Oracle i MSSQL 2000. Do jego osiągnięć na tym polu należy zbudowanie aplikacji do uploadu danych między systemami Excel, Access (dowolna wersja), MSSQL i Oracle (udostępnia tylko trudny w obsłudze SQL Loader, a dane muszą być w postaci tekstowej).
Pod koniec działalności w kole został przez Profesora Pelikanta zainteresowany narzędziem Data Mining, zaimplementowanym w Oracle 10g do eksploracji danych zgromadzonych w dużych bazach wykorzystujących algorytmy hybrydowe. Zaowocowało to stworzeniem za pomocą interfejsu Java API własnej końcówki klienckiej do budowy obiektów wyszukujących ukryte zależności między danymi z wykorzystaniem Oracle Data Mining.
Jego głównym zainteresowaniem na polu programistycznym jest tworzenie algorytmów rozwiązujących skomplikowane zagadnienia matematyczne, jak np. zalgorytmizowanie szybkiej transformaty Fourier'a, mającej bardzo wiele zastosowań w przetwarzaniu sygnałów i przy budowie filtrów.

1. Wstęp

Data Mining (DM) jest zbiorem metod analiz opierających się na oryginalnych rozwiązaniach pozwalających na eksploracje dużych baz danych odbiegających od zasady działania metod statystycznych. Powodem coraz prężniejszego rozwijania się Data Mining'u są przede wszystkim:

1.1. Gromadzenie bardzo dużych zbiorów danych, których analiza przekracza możliwości dotychczas stosowanych narzędzi (głównie statystycznych).

1.2. Rozwój mikroelektroniki i technologii informatycznych umożliwiających szybkie gromadzenie, porządkowanie i przetwarzanie danych za pomocą skomplikowanych i operacjochłonnych algorytmów.

W procesach Data Mining można wyróżnić 4 zasadnicze kroki, które nie mają charakteru linowego (często na jednym z kolejnych etapów proces eksploracji musi wrócić do poprzedniego):

1. **Przygotowanie danych** – ustalenie źródła danych do eksploracji, sprawdzenie ich poprawności (aby zapewnić zgodność danych z różnych źródeł) i dokonanie odpowiednich przekształceń.
2. **Eksploracyjna analiza danych** – poznanie ogólnych właściwości analizowanych danych w tym rozkładu cech charakterystycznych i podstawowych związków między nimi. Bardzo często dokonuje się tutaj wstępnej selekcji zmiennych uwzględniając tylko te cechy, które są istotne, dla przyśpieszenia budowy i działania modeli. Można tutaj filtrować dane z użyciem wyznaczonych wartości minimalnej i maksymalnej, usuwać brakujące dane lub zastępować je wartościami średnimi bądź modalnymi, eliminować i dobierać kolumny, które prawdopodobnie będą najlepszymi predyktorami dla bieżącego zbioru danych i nie wprowadzą „szumów" do tworzonych modeli. Możliwość wstępnego przetworzenia danych w narzędziach DM zapewnia wysoką spójność i jakość danych stosowanych na potrzeby modelu. Transformacja danych jest tak samo istotna jak wybór samej analizy, dając pewność prawidłowego i pełnego wykorzystania modelu. W DM wiersze tabel (lub widoków) źródłowych, często nazywane są *przypadkami* (ang. **Case** lub **Examle**), natomiast kolumny - *atrybutami*.
3. **Właściwa analiza danych** (budowa i ocena modelu lub odkrywanie wiedzy) – rozpoczyna się od wyboru właściwej metody odpowiedniej do: rozwiązywanego problemu, wielkości zbioru danych źródłowych, dopuszczalnej złożoności budowanego modelu i możliwości interpretacyjnych modelu. Po zbudowaniu modelu testuje się go sprawdzając czy prawidłowo klasyfikuje nowe przypadki. Standardową procedurą przy zagadnieniach klasyfikacyjnych i regresyjnych jest korzystanie z algorytmów podziału zbioru danych źródłowych na próbę uczącą i testową. Dane z części uczącej służą do wytrenowania modelu, natomiast na części testowej sprawdzana jest poprawność i trafność modelu.
4. **Zaaplikowanie i stosowanie modelu dla nowych danych.**

Na platformie Oracle 10grid zaimplementowano cały pakiet funkcji i procedur obsługujących szereg zadań związanych z Data Mining. Na uwagę zasługuje fakt, że nie trzeba mieć dodatkowych zewnętrznych narzędzi do analizowania danych przechowywanych w repozytorium Oracle, ponieważ całą funkcjonalność eksploracji danych zaimplementowana jest wewnątrz tego systemu bazo-danowego - Oracle Data Mining. Powoduje to optymalne skrócenie czasu dostępu do danych i budowę obiektów DM przy jednoczesnej zwiększonej poprawności eksploracji.

Oracle Data Mining (ODM) jest całkowicie dedykowany obsługą baz serwera Oracle, przy której się znajduje. Nie można za jego pomocą analizować zewnętrznych źródeł danych i muszą być one wcześniej zaimportowane do bazy Oracle.

2. Modele i algorytmy dostępne w Oracle Data Mining:

1. **Attribute Importance** (Modele ważnych atrybutów) – pełni zadanie „pre-modelu" wyznaczając z danych źródłowych kolumny, które mają największy wpływ na określoną kolumnę celu przypisując wszystkim odpowiednie wagi. Umożliwia to odpowiednie dobranie najważniejszych kolumn przy budowaniu innych „właściwych" modeli dla procesów DM jak np. klasyfikacyjnych czy regresyjnych. Model ten buduje się w ODM algorytmem **Minimum Description Length** - MDL**.**
2. **Classifiaction –** modele typu klasyfikacyjnego grupują dane źródłowe w dyskretne klasy. Można wygenerować je jednym z algorytmów:
   - **Adaptive Bayes Network** (pol. Adaptatywna sieć Bayes'a – ABN)
   - **Decision Tree** (pol. Drzewo Decyzyjne – DT) – od wersji *Oracle 10.1.0.2.*
   - **Naive Bayes** (pol. Algorytm Naiwnego Bayes'a – NB)
   - **Support Vector Machine** (SVM)
3. **Regression –** modele regresyjne działają podobnie do modeli klasyfikacyjnych z wyjątkiem tego, że wartości kolumny celu muszą być numeryczne i ciągłe (mogą być zmiennoprzecinkowe).Różnica polega również na tym, że generowane klasy nie są „dyskretne", tzn. algorytm ten nie daje wyników w postaci określonej liczby klas, do których mają być zaliczane analizowane nowe przypadki, a jedynie nieczytelny dla użytkowników przepis, umożliwiający na podstawie kolumn predykcyjnych określić najlepszą wartość kolumny celu. Algorytmem generującym jest **Support Vector Machine** (SVM).
4. **Clustering –** modele klasteryzacji odpowiedzialne są za znajdowanie naturalnych grup w analizowanych danych. Są dostępne tutaj 2 algorytmy budujące:
   - **K-Means** (KM) i dodany od wersji *10.1.0.2*. algorytm **O_Cluster** (OC).
5. **Association Rules –** modele reguł asocjacyjnych poszukują ścisłych zależności między określonymi wartościami analizowanej kolumny danych źródłowych. Postać danych budujących model wskazuje na jego odrębność w stosunku do pozostałych typów, ponieważ dane te powinny być zapisane w tabeli (bądź widoku) transakcyjnej, w której jeden przypadek charakteryzowany przez określoną kolumnę może obejmować wiele rekordów. Modele te działają według „analizy koszykowej" i są generowane algorytmem **Apriori**.
6. **Feature Extraction –** dodane od wersji *Oracle 10.1.0.2* **-** modele wyszukiwania cech służą do generowania nowych cech charakterystycznych (Feature) utworzonych z kombinacji już istniejących.Generowane są algorytmem Non-Negative **Matrix Factorization** (NMF).
7. **Anomaly Detection –** modele detekcji anomalii mają za zadanie wyszukiwanie anomalii w danych (przydzielonych do klasy **0**) odbiegających określonymi cechami od prawidłowych danych (klasa **1**). Modele tego typu generuje algorytm **One-Class Support Vector Machine** (OCSVM).

ODM umożliwia budowanie modeli zgłębiania danych udostępniając programistom dwa rodzaje interfejsu **API** pozwalające na uzyskanie analogicznych możliwości i rezultatów eksploracji danych:

- **ODM PL/SQL API** – zestaw funkcji i procedur umożliwiających tworzenie obiektów eksploracyjnych w języku PL/SQL.

— **ODM JAVA API** - zestaw klas i metod zgromadzonych w pakietach pozwalających napisać w języku programowania Java własną końcówkę kliencką do ODM. Implementacja interfejsu API Java została całkowicie zmieniona od wersji Oracle 10.2 opierając się o standard **Java Data Mining 1.0**.

Między możliwościami tych interfejsów można dostrzec niewielkie różnice, które jednak nie mają wpływu na zasadniczy wyniki procesów eksploracyjnych.

Tworzone obiekty eksploracyjne (wyniki przygotowania danych dla modeli i same modele, ich testowania, wyniki klasyfikacji nowych danych, tabele i widoki pomocnicze) zarówno interfejsem API: PL/SQL jak i JAVA są przechowywane w bazie danych Oracle w strukturach stabelaryzowanych, które mogą być dalej przetwarzane lub prezentowane na ekranie.

Cały proces eksploracji danych: poczynając od przechowywania danych wejściowych (w tabelach lub widokach), ich transformacji na potrzeby modeli, budowy modeli, ich testowania i stosowania, jako klasyfikatory dla nowych danych, a kończąc na przechowywaniu wyników spoczywa na serwerze Oracle.

Na uwagę zasługuje również fakt, że dopiero w wersji Oracle **10.2** modele tworzone w API Java i API PL/SQL są **interaktywne**, czyli modele utworzone w PL/SQL są dostępne przez interfejs Java i odwrotnie. Wcześniej nie było takiej możliwości i modele zbudowane w jednym interfejsie nie były widoczne dla drugiego.

Poszczególne zadania budowy obiektów eksploracyjnych wykonywane są w Java asynchronicznie. Wynika z tego, że w programie nie trzeba od razu przeprowadzać wszystkie etapy budowy, ponieważ poszczególne kroki zapisuje się w bazie danych Oracle i można się do nich odwołać w późniejszym czasie jak np. przetransformowane dane, tabelę z ustawieniami modelu, a w końcu same modele wyniki klasyfikacji nowych danych czy testowania ich poprawności.

ODM nie respektuje wszystkich typów danych, jakie mogą przyjmować kolumny w Oracle i obsługuje tylko poniższe:

Varchar2, Char, Number, Clob, Blob, Bfile, Xmltype, Uritype.

Wymagane minimalne uprawnienia użytkownika Oracle łączącego się i korzystającego z funkcjonalności serwera Oracle Data Mining

Role:

CONNECT

OLAP_USER

Uprawnienia systemowe:

CREATE TABLE

CREATE VIEW

UNLIMITED TABLESPACE

3. Budowa interfejsu JAVA dla potrzeb zgłębiania danych

3.1. Pakiety wymagane dla budowy interfejsu JAVA API

Dla prawidłowego działania ODM JAVA API wymagane jest, co najmniej środowisko **J2EE 1.4.2.** Dla skorzystania z obszernych możliwości ODM JAVA API przez korzystaniem z charakterystycznych obiektów, funkcji i procedur Data Mining'owych we własnym kodzie programu należy zaimportować wymagane pakiety. Dlatego do CLASSPATH (jest to wykaz katalogów i plików

zawierających biblioteki używane przez kompilator Java) włącza się następujące wymagane zarchiwizowane biblioteki **jar** dostępne w katalogach domowych zainstalowanego serwera Oracle 10.2 lub na stronie www.oracle.com:

%ORACLE_HOME%\oracle\product\10.2.0\db_1\RDBMS\jlib**jdm.jar**
%ORACLE_HOME%\oracle\product\10.2.0\db_1\RDBMS\jlib**ojdm_api.jar**
%ORACLE_HOME%\oracle\product\10.2.0\db_1\RDBMS\jlib**xdb.jar**
%ORACLE_HOME%\oracle\product\10.2.0\db_1\jdbc\lib**ojdbc14.jar**
%ORACLE_HOME%\oracle\product\10.2.0\db_1\oc4j\j2ee\home\lib**connector.jar**
%ORACLE_HOME%\oracle\product\10.2.0\db_1\jlib**orai18n.jar**
%ORACLE_HOME%\oracle\product\10.2.0\db_1\jlib**orai18n-mapping.jar**
%ORACLE_HOME%\oracle\product\10.2.0\db_1\LIB**xmlparserv2.jar**

Niektóre z tych bibliotek są dostępne również na zainstalowanym kliencie Oracle 10g, ale nie wszystkie.

W obrębie tych pakietów stworzonych jest wiele klas obiektów pośrednich wykonujących przygotowywanie danych, określenie parametrów modelu, budowę modelu, klasyfikację nowych danych i testowanie poprawności modelu.

3.2. Problemy z konfiguracją Oracle 10g w wersji polskiej w systemie Windows XP w wersji polskiej przy budowie modeli za pomocą interfejsu Java API

Przy standardowych ustawieniach systemu pojawia się błąd z interpretacją kropki, jako separatora części dziesiętnej od całkowitej w parametrach budowanego modelu:

ORA-40101: Błąd systemu eksploracji danych ORA-40101: Błąd systemu eksploracji danych ORA-06502: PL/SQL: błąd liczby lub wartości: błąd konwersji znaku do liczby

ORA-06512: przy "DMSYS.DBMS_DATA_MINING", linia 305

ORA-06512: przy "DMSYS.DBMS_JDM_INTERNAL", linia 157

ORA-06512: przy "SYS.DBMS_SYS_ERROR", linia 86

ORA-06512: przy "DMSYS.DBMS_JDM_INTERNAL", linia 179

ORA-06512: przy linia 1

Powodem są polskie ustawienia narodowe i dla prawidłowego działania aplikacji należy wykonać jedną z poniższych operacji:

A. **Zmiana ustawień regionalnych systemu operacyjnego na Angielski(Stany Zjednoczone).** Nie wystarczy zmienić separatora części dziesiętnej z przecinka na kropkę dla polskich ustawień.

B. W programie Java można załadować klasę **java.util.Locale** i zmienić ustawienia na język angielski dodając w kodzie programu instrukcję **Locale.setDefault(Locale. ENGLISH).** Instrukcję tą należy umieścić nie tylko przed procesem generacji samego modelu, ale już przed tworzeniem dla niego tabeli z ustawieniami. W przeciwnym razie budowa modelu może nie dojść do skutku.

Zmiana pliku konfiguracyjnego Oracle - SPFILE instrukcją:

ALTER SYSTEM SET NLS_NUMERIC_CHARACTERS = "," scope = **spfile**; **nie odnosi rezultatu.**

Ciekawostką jest to, że można wygenerować niektóre modele bez wykonania jednej z powyższych operacji jak np. modele Attribute Importance, dla których nie podaje się parametrów z wartościami dziesiętnymi.

3.3. Uzyskiwanie połączenia z serwerem Oracle za pomocą interfejsu Java API

Dla uzyskania połączenia z Oracle Data Mining należy wykorzystać klasy:

oracle.dmt.jdm.resource.OraConnectionFactory – „fabryka", z której buduje się właściwy obiekt przechowujący połączenie klienta Java z serwerem Oracle.

javax.datamining.resource.ConnectionSpec – klasa dla specyfikacji połączenia (wymaga podania rodzaju połączenia, loginu i hasła podłączającego się użytkownika)

javax.datamining.resource.Connection - klasa budująca obiekt tworzony z powyższej „fabryki" przechowujący właściwe połączenie z serwerem.

Przykładowy kod połączenia z serwerem:

public static javax.datamining.resource.Connection poloczenie_API ; *//do przechowywania połączenia z serwerem Oracle*

OraConnectionFactory fabryka_oracle_poloczenie = new OraConnectionFactory();

ConnectionSpec poloczenie_specyfikacja = (javax.datamining.resource.ConnectionSpec)fabryka_oracle_poloczenie.getConnectionSpec (); *//tworzony w tyk kroku obiekt jest wymagany dla specyfikacji połączenia*

spoloczenie = "jdbc:oracle:thin:@"+host+":"+port+":"+sid; *// połączenie następuje przez „cienkiego" klienta, a zmienne typu string:* ***host*** *,* ***port*** *i* ***sid*** *określają odpowiednio host serwera Oracle , port nasłuchu i nazwę bazy danych*

Następnie należy skonfigurować połączenie dla danego użytkownika, korzystając z metod obiektu typu *ConnectionSpec:*

poloczenie_specyfikacja.setURI(spoloczenie); *// podanie tekstu URL połączeniu*

poloczenie_specyfikacja.setName(login); *// podanie loginu łączącego się użytkownika*

poloczenie_specyfikacja.setPassword(haslo); *// podanie hasła użytkownika*

Teraz już można próbować nawiązać połączenie z serwerem.

poloczenie_API = fabryka_oracle_poloczenie.getConnection(poloczenie_specyfikacja); *// stworzenie właściwego połączenia API między klientem java, a serwerem Oracle 10g*

Błędy łączenia się z serwerem podobnie jak błędy procesów tworzenia obiektów eksploracyjnych są obsługiwane przez wyjątek **JDMException** pochodzący z klasy **javax.datamining.JDMException**

Wymagane uprawnienia dla procesów DM nie są sprawdzane przy łączeniu się z serwerem. Dopiero przy próbach eksploracji danych przez nieuprawnionych użytkowników (bez stosowych uprawnień) serwer Oracle przez klienta Java może zgłaszać błędy w działaniu narzędzia.

Czasem w programie obsługującym DM wykonuje się typowe zapytania SQL, jak: usunięcie tabeli lub widoku bądź pobranie rekordów z tabeli bazy danych Oracle, wówczas należy zrzutować połączenie przechowywane w obiekcie typu javax.datamining.resource.Connection na „zwykłe" połączenie typu java.sql.Connection umożliwiające wykonywanie tych zapytań:

java.sql.Connection **poloczenieSQL**; *//zaimportowana klasa z java.sql.Connection*

poloczenieSQL = ((OraConnection) **poloczenie_API**).getDatabaseConnection();

gdzie obiekt **poloczenie_API** przechowuje nawiązane i otwarte połączenie z bazą w sposób opisany powyżej.

3.4. Wysyłanie poleceń budowy obiektów eksploracji na serwer Oracle

Zarówno przy transformacji danych, budowie modelu, testowaniu i klasyfikacji nowych danych, korzysta się z tej samej funkcji aktywacji serwera do wykonania tych procesów i zapisania wyników w bazie danych. Są to metody **saveObject** i **execute** opisanego wcześniej obiektu klasy *javax.datamining.resource.Connection*. Po parametrze typu ***javax.datamining.base.Task*** przeciążonej funkcji **saveObject** serwer wie jaki obiekt ma być wygenerowany, a metoda **execute** zwraca uchwyt do tego obiektu charakteryzowanego przez deskrypcję czyli opis dla tworzonego obiektu DM:

poloczenie_API.saveObject(deskrypcja, obiekt_DM, true); *// wstępne zapisanie obiektu bez przeprowadzenia procesów eksploracyjnych. Dzięki temu serwer wie, że z tym opisem związany jest dany obiekt.*

ExecutionHandle uchwyt_wyk =.poloczenie_API.execute(deskrypcja); *// zapamiętanie uchwytu do budowanego obiektu w typie klasy* ***javax.datamining.ExecutionHandle***

ExecutionStatus status = uchwyt_wyk.waitForCompletion(Integer.MAX_VALUE); **//** uruchomienie procesu Data Miningowego po stronie Oracle i ustalenie czasu budowy obiektu, zmienna ta pochodzi z klasy **javax.datamining.ExecutionState**

Po zakończeniu procesu przez Oracle można pobrać informacje typu boolean o poprawnym (**true**) lub błędnym (**false**) stworzeniu obiektu zwracaną przez metodę **getState** powyżej użytego obiektu typu javax.datamining.ExecutionState:

Boolean wyniki = status.**getState**().equals(ExecutionState.success);

3.5. Przygotowywanie danych – transformacja danych źródłowych

Dostępne klasy bezpośrednio związane z przygotowaniem obiektów transformacji można uzyskać z zaimportowanego pakietu **oracle.dmt.jdm.transform**. , który przechowuje klasy odpowiedzialne za grupowanie, normalizację i obcinanie wartości brzegowych. Wszystkie te procesy przeprowadza się za pomocą analogicznych kodów, różniących się jedynie wskazaniem na odpowiednią metodę Nasuwającą się różnicą w stosunku do interfejsu PL/SQL API jest **brak** w tym pakiecie klasy odpowiedzialnej za przygotowanie danych pod względem wypełniania luk w kolumnach źródłowych (ang.missing), której funkcjonalność można wygenerować samemu korzystając z instrukcji SQL: INSERT, UPDATE i DELETE wysyłanych z klienta JAVA. Ważne jest również istnienie w danych źródłowych kolumny identyfikującej każdy z przypadków, bez której nie można zbudować żadnego z obiektów eksploracji danych.

Transformacje danych przeprowadzonych na danych budujących model powinny być takie same jak przy transformowaniu nowych danych klasyfikowanych za pomocą tych modeli i je testujących, ponieważ od tego zależy dokładność i poprawność tych procesów.

Poszczególne transformacje umieszczone są w klasach:

oracle.dmt.jdm.transform.binning – klasa z metodami odpowiedzialnymi za grupowanie wartości kolumn znakowych i numerycznych

oracle.dmt.jdm.transform.normalize - klasa z metodami odpowiedzialnymi za normalizacje (dotyczy tylko kolumn numerycznych w tabeli źródłowej)

oracle.dmt.jdm.transform.clipping – klasa z metodami odpowiedzialna za przycinanie (dotyczy tylko kolumn numerycznych w tabeli źródłowej).

Przykładowy kod transformacji grupowania kolumn znakowych i numerycznych w tabeli (widoku) źródłowej

OraTransformationTaskFactory fabryka_transform; *// klasa budująca obiekty wykorzystywane dla wszystkich dostępnych rodzajów transformacji do określania ich parametrów*

OraTransformationTask transformacja; // dla *specyfikacji parametrów transformacji,* które posłużą serwerowi Oracle do zbudowania wynikowego widoku i zapisania go w swoim repozytorium.

OraBinningTransformFactory fabryka_grupowanie ; *// „fabryka" grupowania generująca obiekt przechowujący parametry tej transformacji, pochodzi z klasy oracle**.dmt.jdm.transform.binning.OraBinningTransformFactory***

OraBinningTransformImpl grupowanie; *// obiekt utworzony z klasy* oracle***.dmt.jdm.transform.binning.OraBinningTransformImpl*** *do przechowywania ustawianych parametrów transformacji–grupowania,*

fabryka_transform = (OraTransformationTaskFactory)poloczenie_API.getFactory("oracle.dmt.jdm.task.OraTransformationTask"); *// stworzenie obiektu fabryki transformacji*

fabryka_grupowanie = (OraBinningTransformFactory) polloczenie_API.getFactory("oracle.dmt.jdm.transform.binning.OraBinningTransform"); *//zainicjowanie „fabryki" grupowania metodami charakterystycznymi dla tej transformacji*

grupowanie =(OraBinningTransformImpl)fabryka_grupowanie.create(tabela_z, widok_w, true); *//pierwszy parametr to nazwa tabeli lub widoku źródłowego który chcę poddać transformacji , drugi parametr określa nazwę wynikowego widoku po transformacji*

Z każdego rodzaju transformacji można wyłączać kolumny np. w przypadku kolumny identyfikującej każdy z przypadków, kolumnę celu (ang. target) albo kolumny, które ze względu na odpowiednią zawartość nie zostaną przetransformowane umieszczając je w liście dodawanej metodą **setExcludeColumnList** do obiektu z parametrami transformacji. Tak naprawdę nie trzeba wiedzieć czy kolumna id i celu są numeryczne czy znakowe, bo można wyłączyć te kolumny z obydwu procesów grupowania numerycznego i znakowego - to nie będzie błąd. Oracle podczas procesu transformacji sam rozpoznaje typ przechowywany przez daną kolumnę. W tym miejscu należy wspomnieć o odrębności modelu regresyjnego od innych, którego numeryczną kolumnę celu również należy przetransformować (co najmniej normalizacja):

String [] kolumny_wylaczone = {kol_id,kol_cel*}; // lista kolumn które nie będą brały udziału w transformacji oczywiście trzeba wyłączyć kolumnę identyfikacji rekordów i celu.*

grupowanie.setExcludeColumnList(kolumny_wylaczone);

Następnie można przejść do ustawienia parametrów transformacji kolumn numerycznych i znakowych nadpisując w ten sposób ustawienia domyślne:

Parametry dla kolumn numerycznych:

grupowanie.setNumberOfBinsForNumerical(10); // określenie *na ile grup powinny być podzielone wartości w kolumnach numerycznych w tabeli źródłowej (domyślnie 10)*

grupowanie.setNumericalBinningType(OraNumericalBinningType. quantile); *//określenie typu grupowania wartości w kolumnach numerycznych –* ***quantile*** *, można również wybrać jedną z pozostałych metod:* ***equi_width*** *,* ***auto_ equi_width*** *, a także* ***systemDefault*** *– określający domyślny typ grupowania.*

Parametry dla kolumn znakowych ustawia się analogicznie:

grupowanie.setNumberOfBinsForCategorical(10);

grupowanie.setCategoricalBinningType(OraCategoricalBinningType.top_n); //jedyna dostępna metoda **top_n** przeprowadza proces grupowania dla kolumn numerycznych

transformacja =fabryka_transform.create(grupowanie); *//zapisanie ustawień grupowania*

3.6. Usuwanie obiektów Data Miningowych z repozytorium Oracle

Obiekty Data Miningowe jak modele czy wyniki testowania to nie są zwykłe obiekty bazy danych jak tabele czy widoki. Chociaż mają strukturę tabelaryczną , to nie można się do nich odwołać jak do tabeli czy widoku. Podobnie jest z usuwaniem tych obiektów. Do tego celu wykorzystuje się połączenie API z serwerem Oracle.

W procedurze usuwającej obiekty eksploracji **removeObject** należącej do klasy **javax.datamining.resource.Connection** określa się nazwę usuwanego obiektu i jego typ:

phisicalDataSet – obiekt z ustawieniami danych źródłowych,

buildSettings **- obiekt** (tabela) z ustawieniami modelu**,**

task –deskrypcja budowanego obiektu eksploracyjnego,

model – zbudowany model Data Mining,

testMetrics – zbudowany obiekt testowania modelu,

applySettings – ustawienia dla obiektu klasyfikacji nowych danych przez model DM.

Oto przykładowy kod usuwania modelu z repozytorium Oracle przy pomocy metody removeObject dostępnej dla obiektu typu javax.datamining.resource.Connection przechowującego połączenie z serwerem:

poloczenie_API.removeObject(Justawieni_tz, NamedObject.model); *// w funkcji usuwającej* ***removeObject*** *należącej do klasy* ***javax.datamining.resource.Connection*** *określa usuwany obiekt Data Miningowy: pierwszy parametr typu String określa jego nazwę w repozytorium , drugi typ usuwanego obiektu - wykorzystując przestrzeń nazw* ***NamedObject*** *pochodzącą z klasy* ***javax.datamining.NamedObject***

Jedynie obiekty wynikowe klasyfikacji nowych danych przez modele eksploracyjne są tabelami i można je usuwać wykonując zwykłe zapytania SQL - DROP.

4. Budowa modeli eksploracyjnych

Procesy tworzenia modeli eksploracyjnych w Oracle 10g różnią się między sobą parametrami budowy modelów, ustawień algorytmów, które mają je wygenerować czy też użytymi obiektami (np. przechowującymi poszczególne ustawienia). Jednak w poszczególnych etapach budowy tych modeli widoczna jest duża analogia, dlatego poniżej przedstawiony jest kod tych procesów dla jednego wybranego modelu Regresyjnego generowanego algorytmem Support Vector Machine.

W pierwszym kroku należy utworzyć obiekt do przechowywania ustawień budowanego modelu:

RegressionSettingsFactory fabryka_ustawienia_modelu_regresji *;// „fabryka" ustawień modelu wymagana do uzyskania obiektu typu* ***RegressionSettings***

RegressionSettings ustawienia_modelu_regresji ; *// obiekt do przechowywania ustawień modelu , które można zmieniać w określonych granicach*

SVMRegressionSettingsFactory fabryka_ustawienia_SVMR;*//„fabryka" ustawień modelu utworzona z klasy* ***javax.datamining.algorithm.****svm.regression.SVMRegressionSettingsFactory*

SVMRegressionSettings ustawienia_algorytm_SVMR; *// obiekt przechowujący ustawienia modelu, utworzony z klasy* ***javax.datamining.algorithm.****svm.regression.SVMRegressionSettings*

Po usunięciu starych obiektów eksploracji i przetransformowaniu danych na potrzeby danego obiektu DM można przystąpić do jego budowy.

Rozpoczyna się od zadeklarowania i zdefiniowania typów „fabryk" odpowiedzialnych za poszczególne etapy budowy obiektów eksploracji wykorzystywane do tych samych celów podczas budowy modeli, klasyfikacji nowych danych i testowania tych modeli:

PhysicalDataSetFactory fabryka_set = (PhysicalDataSetFactory) polocze-nie_API.getFactory("javax.datamining.data.PhysicalDataSet"); *// „fabryka" budowy ustawień dla tabeli z danymi źródłowymi modelu, jest typu* ***PhysicalDataSetFactory*** *zaimportowanego z klasy* ***javax.datamining.data.PhysicalDataSetFactory***

PhysicalAttributeFactory fabryka_atrybuty = (PhysicalAttributeFactory) polocze-nie_API.getFactory("javax.datamining.data.PhysicalAttribute"); *// „fabryka" ustawień danych źródłowych , jest typu* ***PhysicalAttributeFactory*** *zaimportowanego z klasy* ***javax.datamining.data.PhysicalAttributeFactory***

BuildTaskFactory fabryka_budowy = (BuildTaskFacto-ry)poloczenie_API.getFactory("javax.datamining.task.BuildTask"); */ „fabryka" budowy modelu używana przy wysyłaniu zadania budowy modelu na serwer Oracle , jest typu* ***BuildTaskFactory*** *zaimportowanego z klasy* ***javax.datamining.task.BuildTaskFactory***

Następnie można przejść do etapu określania źródła danych dla budowanego modelu. Dane te powinny być wcześniej przetransformowane w sposób odpowiedni dla danego modelu (dla omawianego modelu regresyjnego dane powinny być co najmniej znormalizowane):

PhysicalDataSet Dane_treningowe = Wybor_algo.fabryka_set.create(widok_danych, false); *// wskazanie na widok (widok_danych) utworzony z przetransformowanej tabeli z danymi źródłowymi dla tworzonego modelu i zapisanie do typu* ***PhysicalDataSet*** *zaimportowanego z klasy* ***javax.datamining.data.PhysicalDataSet***

Określa się w tych danych źródłowych kolumnę identyfikującą każdy z przypadków i jej typ:

PhysicalAttribute atrybuty_dane_treningowe = fabryka_atrybuty.create(kolumna ID,AttributeDataType.integerType, PhysicalAttributeRole.caseId); //*ustawienie wymaganej kolumny identyfikującej każdy przypadek z tabeli (widoku), która ma być typu* ***Integer*** *, ustawienia te zapisuje się do obiektu typu* ***PhysicalAttribute*** *zaimportowanego z klasy* ***javax.datamining.data.PhysicalAttribute***

Poniższy kod zapisuje ustawienia obiektu typu PhysicalAttribute z ustawieniami kolumny identyfikującej rekordy do obiektu typu PhysicalDataSet dla uzyskania jednoznacznych informacji o ustawieniach danych źródłowych:

Dane_treningowe.addAttribute(atrybuty_dane_treningowe); *//dodanie do obiektu* ***Dane_treningowe*** *ustawień obiektu* ***atrybuty_dane_treningowe***

Kolejnym krokiem budowy modeli jest zdefiniowanie ustawień algorytmu generującego:

fabryka_ustawienia_SVMR = (SVMRegressionSettingsFactory) poloczenie_ API.getFactory("javax.datamining.algorithm.svm.regression.SVMRegressionSettings"); *//fabryka ustawień wykorzystywana do uzyskiwania obiektu typu* ***SVMRegressionSettings***

ustawienia_algorytm_SVMR = fabryka_ustawienia_SVMR .create(); *// zapisanie ustawień dla algorytmu generującego model Regresyjny, można zmieniać lub pozostawić wartości domyślne tych parametrów.*

Z powyższego obiektu typu SVMRegressionSettings można odczytać za pomocą odpowiedniej funkcji get wartości domyślne parametrów algorytmu, który ma wygenerować model.

ustawienia_algorytm_SVMR.getKernelFunction().name()); *- zwraca domyślną funkcję jądra,* ***ustawienia_algorytm_SVMR.getTolerance());*** *- zwraca domyślną wartość parametru tolerancji*

Nie wszystkie parametry algorytmów mają wartości domyślne i nie można się do nich odwołać, jeżeli się ich wcześniej nie ustawi za pomocą określonej dla parametru funkcji set, która służy do ustawiania własnych wartości poszczególnych parametrów:

*ustawienia_algorytm_SVMR.**setEpsilon**(0.1f); - ustawienie własnej wartości dla parametru Epsilon ustawienia_algorytm_SVMR.**setKernelFunctio**n(**KernelFunction.kGaussian**); - ustawienie funkcji jądra na Gaussowską. Wymaga ona większych zasobów sprzętowych niż liniowa, jeżeli chodzi o prędkość procesora i rozmiar dostępnej pamięci.*

ustawienia_modelu_regresji = fabryka_ustawienia_modelu_regresji.create(); // *zapisanie ustawień modelu do obiektu typu* ***RegressionSettings****, można je zmieniać* ***ustawienia_modelu_regresji.setAlgorithmSettings(ustawienia_algorytm_SVMR); //****dodanie do obiektu typu* ***RegressionSettings*** *ustawień obiektu typu* ***PhysicalAttribute*** dla uzyskania jednego obiektu ze wszystkimi ustawieniami dla generowanego modelu

ustawienia_modelu_regresji.setTargetAttributeName(kolumnaCel); // *ustawienie kolumny celu w danych źródłowych dla budowy modelu, pobranej z łańcucha* ***kolumnaCel*** *przechowującego jej nazwę.*

Końcowy obiekt typu BuildTask z ustawieniami: danych źródłowych, algorytmu generującego, modelu zainicjowany funkcją create obiektu BuildTaskFactory jest przygotowany na wysłanie go na serwer w celu rzeczywistej budowy modelu DM:

BuildTask budowa_modelu = fabryka_budowy.create(

Justawienia_tabela_zrodlowa, // nazwa *wcześniej utworzony obiektu po stronie Oracle z obiektu typu* PhysicalDataSet*, charakteryzującego źródło danych*

Justawienia_algorytmu, *// nazwa wcześniej utworzonej tabeli z obiektu* typu SVMRegressionSettings *po stronie Oracle przechowującej ustawienia dla algorytmu i modelu*

nazwa_model *// nazwa pod jaka model ma zostać zbudowany i zapisany w repozytorium Oracle*);

5. Klasyfikacja nowych danych za pomocą wytrenowanych modeli

Za pomocą interfejsu Java API można klasyfikować nowe dane przez istniejące modele i analizować wyniki, które mogą być również przydatne przy późniejszym testowaniu tych modeli. Nie wszystkie modele zaimplementowane w Oracle 10g mogą klasyfikować nowe dane. Procesowi temu nie podlegają modele reguł asocjacyjnych i ważnych atrybutów. Dla wszystkich pozostałych modeli proces ten przebiega analogicznie. Poniższy przykład jest przygotowany dla modelu wygenerowanego dowolnym algorytmem klasyfikacyjnym.

Podobnie jak dla budowy modeli wymagane jest zadeklarowanie i zainicjowanie odpowiednich obiektów- „fabryk":

ClassificationApplySettingsFactory fabryka_ustawien_nowe_dane_Klasyfi ; // *fabryka ustawień dla nowych danych zaimportowana z klasy* ***javax.datamining.supervised.classification.ClassificationApplySettingsFactory***

fabryka_ustawien_nowe_dane_Klasyfi= (ClassificationApplySettingsFactory) polozenie_API.getFactory("javax.datamining.supervised.classification.ClassificationApplySettings"); *// odpowiednia do przypadku inicjalizacja fabryki ustawień*

DataSetApplyTaskFactory nowe_dane_fabryka_budowy = (DataSetApplyTaskFactory) poloczenie_API.getFactory("javax.datamining.task.apply.DataSetApplyTask"); // fabryka dodawania nowych danych typu DataSetApplyTaskFactory *z klasy* ***javax.datamining.task.apply.DataSetApplyTaskFactory****, używana przy wysyłaniu zadania klasyfikacji nowych danych przez model na serwer Oracle*

Obiekt typu ClassificationApplySettings jest używany do przechowywania ustawień dodawanych danych inicjowany jest z fabryki typu ClassificationApplySettingsFactory:

ClassificationApplySetting ustawienia_nowe_dane_Klasyfi = fabryka_ ustawien_nowe_dane_Klasyfi.create() ;

Jeżeli otrzymane wyniki mają posłużyć do testowania modelu klasyfikującego należy dodatkowo umieścić w kodzie 3 poniższe instrukcje dla identyfikacji kolumny celu w zbiorze nowych danych, które nie są wymagane dla prawidłowego otrzymania wyników klasyfikacji nowych danych. Wykorzystuje się tutaj procedurę **setSourceDestinationMap** z parametrem typu lista określającą kolumnę celu.

HashMap mapa_celu = new HashMap(); *// obiekt zaimportowany z klasy* ***java.util.HashMap***

mapa_celu.put(kolumnaCel, kolumnaCel);

ustawienia_nowe_dane_Klasyfi.setSourceDestinationMap(mapa_celu);

Końcowy obiekt typu DataSetApplyTask z ustawieniami: danych źródłowych, algorytmu generującego, modelu inicjuje się funkcją create obiektu DataSetApplyTaskFactory:

DataSetApplyTask nowe_dane_budowa = nowe_dane_fabryka_budowy.create(

ustawienia_tabela_nowe, //nazwa *obiektu utworzonego z typu* PhysicalDataSe**t** *po stronie Oracle przechowującego ustawienia tabeli z nowymi danymi*

model, *// nazwa modelu wcześniej utworzonego , za pomocą którego Oracle ma klasyfikować nowe dane*

Justawienia_dodawania, *// ustawienia dla klasyfikowanych nowych danych tworzone z obiektu typu* ClassificationApplySettings*, może również zawierać informacje potrzebne przy testowaniu modelu*

tabela_nowe_dane_wynik); *// nazwa obiektu eksploracyjnego (tabeli), pod jaką mają być przechowywane wyniki klasyfikacji.*

W strukturze funkcji **create** dla obiektu typu DataSetApplyTaskFactory można dostrzec analogię w stosunku do tej samej procedury wykonywanej dla obiektu typu BuildTaskFactory wykorzystywanego przy planowaniu procesu budowy modelu. Parametry tych funkcji szczegółowo określają dane źródłowe ich charakter, parametry budowy tych obiektów i nazwy, pod jakimi mają być zapisane wyniki. Podobnie będzie w procesach testowania opisanych poniżej.

Na uwagę zasługuje tutaj fakt, że generowany obiekt przechowujący wyniki klasyfikacji danych przez odpowiedni model eksploracyjny jest tak naprawdę tabelą. Co upraszcza sposób odwoływania się do tych rezultatów za pomocą zwykłych „sql'owych" zapytań.

6. Testowanie modeli eksploracyjnych

Na platformie Oracle 10g można testować tylko modele Klasyfikacji i Regresji. Dla interfejsu JAVA API proces ten można oprzeć o tabelę z nowymi danymi- wymagane jest tutaj użycie testowanego modelu lub tabelę z wynikami klasyfikacji nowych danych przez testowany model sprawiając, że całe zadanie odbywa się bez bezpośredniego udziału modelu.

6.1. Testowanie na podstawie nowej tabeli z danymi

Podobnie jak przy poprzednich modelach eksploracyjnych przy testowaniu modelu należy posłużyć się charakterystycznymi „fabrykami" dzięki, którym użytkownik może zdefiniować min. źródło danych i rodzaj testowanego modelu:

ClassificationTestTaskFactory fabryka_budowy_Test = (ClassificationTestTaskFactory) poloczenie_API.getFactory

("javax.datamining.supervised.classification.ClassificationTestTask");

ClassificationTestTask budowa_Test = fabryka_budowy_Test.create(

Justawienia_tabela_Test, *// wcześniej utworzony przy pomocy typu* PhysicalDataSet *i przechowywany w bazie danych Oracle obiekt definiujący ustawienia tabeli z danymi testowymi dla modelu. Parametr wymagany tak jak przy budowie modeli i klasyfikacji nowych danych.*

model, · // nazwa*, pod jaką jest przechowywany w bazie Oracle testowany model*

wyniki_test // nazwa*, pod jaką Oracle ma zapisać obiekt wyników testowania*

);

W tym rodzaju testowania bierze udział bezpośrednio model.

6.2. Testowanie modelu na podstawie tabeli z wynikami klasyfikowanych nowych danych przez ten model

Należy zauważyć, że interfejs PL/SQL API nie udostępnia tego typu testowania ograniczając się tylko do wcześniej opisanego opartego o nowe dane testowe i wykorzystującego bezpośrednio testowany model.

Najważniejszą różnicą w stosunku do poprzedniego typu testowania to składnia procedury przeciążonej create obiektu typu ClassificationTestTask.

ClassificationTestTask budowa_Test = fabryka_budowy_Test.create(

Justawienia_tabela_Test, // *wcześniej utworzony przy pomocy typu PhysicalDataSet i przechowywany w bazie danych Oracle obiekt definiujący ustawienia tabeli z danymi testowymi dla modelu. Parametr wymagany tak jak przy budowie modeli i klasyfikacji nowych danych.*

kolumnaCel*, // nazwa kolumny celu w sklasyfikowanych nowych danych*

"PREDICTION"*, // nazwa kolumny celu w tabeli z wynikami klasyfikacji nowych danych przez model*

wyniki_test *// nazwa pod jaką Oracle ma zbudować obiekt testowania modelu*

);

Można również dla powyższego obiektu z ustawieniami testowania określić nazwę kolumny tabeli z wynikami klasyfikacji nowych danych przez model testowany, podającej poprawność klasyfikacji poszczególnych przypadków (domyślnie jest to „probability"). Nazwę tej kolumny nadaje Oracle podczas budowy wyników klasyfikacji nowych danych:

budowa_Test.**setPredictionRankingAttrName**("PROBABILITY*"); // tutaj określa się nazwę kolumny tabeli z wynikami klasyfikacji nowych danych przez model testowany, określającej poprawność klasyfikacji poszczególnych przypadków. Nazwę tej kolumny nadaje Oracle podczas klasyfikacji nowych danych.*

Przed wysłaniem zadania budowy obiektu testowania przez Oracle Data Mining użytkownik może selektywnie wybrać, jakie charakterystyki mają zostać wygenerowane (domyślnie wszystkie są generowane) korzystając z metody **computeMetric** obiektu typu ClassificationTestTask. Pierwszy parametr tej procedury definiuje typ budowanej charakterystyki, a drugi czy ma być ona zbudowana (**true**) czy nie (**false**) jak w poniższym przykładzie:

— dla zbudowania macierzy pomyłek testowanego modelu:

```
budowa_Test.computeMetric(ClassificationTestMetricOption.confusionMatrix, true);
```

— dla zbudowania charakterystyki LIFT:

```
budowa_Test.computeMetric(ClassificationTestMetricOption.lift , true);
```

— dla zbudowania charakterystyki ROC:

```
budowa_Test.computeMetric (ClassificationTestMetricOption.receiverOperating Char-
acteristics, true);
```

Ponadto można scharakteryzować dodatkowe parametry testowania jak:

— ustawić własną liczbę kwantyli dla charakterystyki LIFT

```
budowa_Test.setNumberOfLiftQuantiles([liczba_kwantyli]);
```

— ustawić własną macierz kosztów

```
budowa_Test.setCostMatrixName("Nazwa macierzy kosztów");
```

— można określić , która wartość kolumny celu jest najważniejsza przy testowaniu

```
budowa_Test.setPositiveTargetValue(" najważniejsza_wartość_w _kolumnie_celu");
```

7. Podsumowanie

Omówione w wielkim skrócie możliwości interfejsu JAVA API udostępnianego przez Oracle 10g pozwalają na budowę własnych kodów odwołujących się do funkcjonalności Oracle Data Mining. Sprawia to możliwość budowy autonomicznych aplikacji uruchamianych zdalnie z „wszystkimi" elementami, które w razie potrzeby będą odwoływać się do funkcji i procedur data miningowych udostępnianych przez serwer. Należy pamiętać, że wszystkie procesy związane z eksploracją danych odbywają się po stronie serwera Oracle, a wyniki pośrednie i końcowe tych zadań zapisywane są w obsługiwanej przez niego bazie danych.

W procesach budowy poszczególnych obiektów eksploracyjnych można dostrzec dużą analogię dla różnych typów modeli. Inaczej jest w przypadku wyświetlania ich wyników, gdzie większość z nich ma własną specyfikę przechowywania informacji i trzeba dobrze znać jej strukturę, aby można było uzyskać wymagane informacje.