Java Data Mining API (JSR-73)

Lukáš Zapletal

22. května 2006

1. Úvod

Oblast informatiky zvaná data mining (získávání znalostí, dolování dat) je dnes v popředí zájmů mnoha jednotlivců i firem. Avšak zatím zde nebyla snaha formáty výměny dat určených pro data mining jakož i použité algoritmy a postupy nijak standardizovat.

Proto byla zřízena pracovní skupina (Oracle, SPSS, SAS, Sun, Kxen, IBM a další) v rámci JSR, která se tímto problémem zabývala, což vyústilo ve finální specifikaci JDM Specification, která je volně ke stažení na webovém sídle www.jsr.org ([1]). Na tomto sídle se shromažďují pracovní skupiny, které se snaží rozšířit samotnou platformu Jav, nebo vytvořit různé standardy s tímto široce používaným jazykem související.

Hned na začátku je nutno poznamenat, že se jedná o balík rozhraní a abstraktních tříd, které spolu s dokumentací posloží implementátorům (dodavatelům technologií pro data mining) i vývojářům cílových aplikací (firmám implementujícím produkty pro data mining) úspěšně spolupracovat. Součástí JSR-73 je také referenční implementace, což je jakási demoverze vracející stále stejné výsledky, nad kterou mohou vývojáři testovat své produkty (například uživatelská rozhraní). Stejně tak je součástí také sada testů, které usnadní implementátorům jednotlivých algoritmů a metod otestovat, zda jejich produkty splňují přísné požadavky dané specifikací.

Kromě samotné specifikace (160 stran A4) je přítomno také několik úkázkových vzorků dat a konečně také WDSL specifikace nabzející používat toto API jako webovou službu. Zájemci o tuto specifikaci by se měli jistě podívat také na [2] (Java API for OLAP), což je specifikace doplňující tuto problematiku.

Tato zpráva se zabývá verzí specifikace 1.0. Java Data Mining API prochází pochopitelně vývojem, takže se může stát, že tento dokument po čase nebude zcela aktuální. Nicméně jako všechny rozhraní jazyka Java bude zřejmě i u JDM zachována zpětná kompatibilita.

1.1. Architektura

Architektura JDM je složena ze tří částí:

- API – toto rozhraní úplně odstíní vlastní implementace, takže programátoři pracující s JDM nemusejí studovat různé přístupy pro různé algoritmy;
- DME – data mining engine – vlastní jádro, kde se odehrává činnost (dolování dat), je postavena na architektuře klient-server;

- MOR – mining object repository – stará se o perzistenci dat a odděluje a zapouzřuje tuto činnost tak, aby bylo snadné používat obyčejné soubory, XML soubory či relační a objektové databáze.

2. Možnosti

JDM zapouzdřuje následující metody a jejich kombinace: klasifikace, regrese, důležitost atributů, klastrování a asociace.

2.1. Úlohy

JDM umožňuje vytvářet jednotlivé úlohy: vytvoření modelu, testování modelu, aplikování modelu na data, vytvoření statistik, import a export dat.

Modelem je v JDM chápán výsledek algoritmu, vytvořená znalost například v podobě pravidel pro rozhodovací strom nebo výsledky algoritmu hledajícího asociace v datech. K vytvoření modelu je nutno zvolit jednak funkci (typ), a pak také vlastní algoritmus. Jakmile je vlastní model pomocí jádra DME vytvořen, je možno jej uložit pomocí vrstvy MOR.

Stejně jako v programování hraje velkou roli testování, tak i v oblasti data miningu je nutno vytvořený model otestovat. K tomu existuje v JDM aparát určený pro testování, je schopen určit, nakolik je model přesný.

Pro metody klasifikace, regrese nebo klastrování je nutno mít možnost vytvořený model aplikovat. I k tomu nabízí JDM sadu rozhraní. Například software pro banku, který bude rozhodovat, zda klientovi udělit či neudělit úvěr na základě Bayesovy metody, použije předem vytvořený model (ten by se mohl automaticky generovat každou sobotu v noci).

Statistiky dat je další oblastí, kterou specifikace pokrývá. Před zahájením některých algoritmů je dobré data normalizovat či předem upravit, a k tomu je dobré znát o datech různé průměry, minimální a maximální hodnoty, odchylky a mediány. Právě k tomu slouží tato část specifikace.

Velkou roli hlaje import a export (získaných) dat. Když si pracovní skupina kladla otázky, jak import a export navrhnout, padly podmínky, aby byla zachována kompatibilita se stávajícími aplikacemi pro dolování dat, aby formáty byly otevřené (dostupné všem) a snadno čitelné i mimo aplikace postavené nad JDM.

Jednotliví dodavatelé mají možnost jednak si naprogramovat vlastní import a export moduly, nebo využít vestavěnou podporu formátů PMML a CWM-DM (založené na XML). Formát PMML je široce rozšířen a podporuje jej většina nástrojů pro data mining, avšak není schopen pokrýt všechny vlastnosti nabízené specifikací JDM. Naproti tomu CWM-DM je dostatečně obecný, avšak jeho rozsáhlost zapřičinila, že mnoho dodavatelů jej nepodporuje vůbec, nebo jen z části.

Formát PMML je široce rozšířen a podporuje jej většina nástrojů pro data mining, avšak není schopen pokrýt všechny vlastnosti nabízené specifikací JDM. Naproti tomu CWM-DM je dostatečně obecný, avšak jeho rozsáhlost zapřičinila, že mnoho dodavatelů jej nepodporuje vůbec, nebo jen z části.

2.2. Základní objekty

Práce s JDM se velmi podobá architektuře JDBC pro připojení k relačním databázím. Je objektově navržena tak, aby bylo možné snadno (nebo přímo za běhu) vyměnit jádro

DME (tedy dodavatele data mining techlonogie). Základním objektem je Connection, které zpřístupňuje obě vrstvy, tedy samotné jádro DME i vlastní data ve vrstvě MOR.

Task (úloha) je objektem reprezentujícím operace sestavení (build), testování (test) a pro některé metody také aplikování (apply). Úlohy se mohou spouštět i asynchronně, což je vhodné zejména v distribuovaném a webovém prostředí.

Physical data set, physical attribute a physical data record jsou objekty reprezentující zdrojová data. Záleží na dodavateli, jaké poskytne metody pro čtení těchto dat, ale obvykle tyto třídy zapozdří spojení na databázi, XML soubory nebo OLAP krychle.

Build settings objekt nabízí podrobně obecné nastavovací mechanismy pro jednotlivé metody. Objekty Algorithm a Algorithm Settings pak nabízejí podrobnější nastavení pro jednotlivé algoritmy (tedy nejnižší úroveň).

Objekt Model je pak výsledkem činnosti metod (a algoritmů), může být použit k aplikaci, může být vyexportován nebo například testován. Jemnějším objektem je pak Model Detail, což je model pro specifickou metodu (například pro klastrování obsahuje metody jako getCluster a podobně).

Logical Attribute reprezentuje atributy pro data mining operace, může být numerický nebo kategoriální (případně pomocný – není použit algoritmem). Logical Data pak reprezentuje data, se kterými se přímo manipuluje při výpočtech.

Attribute Statistics pomáhá při generování statistik nad daty, například pro diskretizaci atributů a podobně.

Objekt Rule pak hraje roli při získávání asociačních pravidel z dat.

JDM obsahuje mnoho dalších pomochných objektů, například Category (kategorie atributů), Taxanomy (hiearchie kategorií) a další. Například problematika mapování (sdružení atributů fyzických dat a jejich logické reprezentace pro výpočet) obsahuje mnoho objektů umožňujících tyto operace.

3. Rozhraní JDM

Nyní se budu věnovat popisu vlastního API.

- **javax.datamining** – Hlavní balíček obsahující celé rozhraní. Definuje několik pomocných tříd a výjimek společných celému API.
- **javax.datamining.base** – Obsahuje základní rozhraní popsané výše, tedy Model, ModelDetail, Task, BuildSettings a AlgorithmSettings. Rozhraní Model je tedy přetíženo například rozhraním AssociationModel z balíčku javax.datamining.association.
- **javax.datamining.resource** – Nabízí rozhraní a třídy pro připojení na jádro DME.
- **javax.datamining.data** – Balíček zastřešující veškeré rozhraní pro logickou a fyzickou reprezentaci dat.
- **javax.datamining.statistics** – Rozhraní pro získávání statistik atributů. Stejně jako ve všech ostatních případech je to jen sada rozhraní, abstraktních tříd a enum-tříd. Veškerá implementace je na dodavateli data mining jádra, protože je pochopitelně závislá od implementace logického uložení dat.

- **javax.datamining.rule** – Rozhraní pro predikáty a jejich skládání. Pomocí nich je možno vytvořit například predikát *věk ¿ 18 A ročník = 1 NEBO ročník = 2*. Finální verze specifikace obsahuje chybu, kdy popisuje balíček jako javax.datamining.rules, ve skutečnosti byl však nazván jednotným číslem.

- **javax.datamining.task** – Roznraní definující čtyři hlavní úlohy: BuildTask, ComputeStatisticsTask, ImportTask a ExportTask.

- **javax.datamining.association** – Rozhraní pro získávání asociačních pravidel. Definuje takové rozhraní, jako AssociationModel, AssociationRune, ItemSet nebo RuleFilter.

- **javax.datamining.clustering** – Rozhraní pro klasterizaci obsahující například ClusteringModel, Cluster nebo SimilarityMatrix.

- **javax.datamining.attributeimportance** – Rozhraní pro práci s důležitostmi atributů. Například při klasterizaci je vhodné vybrat ze zdrojových dat pouze ty nejdůležitější atributy, které nejvíce ovlivní výsledný model, a urychlit tak generování finálního modelu.

- **javax.datamining.supervised** – Obsahuje rozhraní pro poznávací (učící se) algoritmy. Balíček je dále rozdělen na dva s názvy *classification* a *regression*.

- **javax.datamining.algorithm** – Balíček pro nastavování jednotlivých algoritmů. Algoritmy dodají samozřejmě dodavatelé jádra, je však nutno umožnit uživateli pracujícím s JDM nějak nastavit parametry pro data mining algoritmy. Balíček obsahuje řadu rozhraní pro specifikaci a detailní nastavení algoritmů jako je naivní bayesův klasifikátor, k-means klasterizátor, neuronovu síť, algoritmus SVM nebo obecný stromový klasifikátor.

- **javax.datamining.modeldetail** – Nabízí několik doplňujících rozhraní pro modely, které jsou specifické některým algoritmům.

4. Příklad implementace

Nyní se budu zabývat rozhraním JDM z obou stran, to znamená z pohledu dodavatele data mining algoritmů a pohledu uživatele-programátora, používající toto API. Popíšu vytvoření praktického příkladu – aplikaci vyhovující standardu JDM sloužící k získávání asociačních pravidel ze záznamů webového serveru Apache. Tento program je schopen dolovat data nad záznamy webového serveru a získávat tím informace o chování uživatelů na určitém webovém sídle.

Rozhraní JDM je velmi rozsáhlé, obsahuje stovky tříd, každá má většinou několik desítek metod. Proto jsem implementoval jen nejnutnější součásti, aby ukázkový příklad fungoval. Jádro získávání pravidel jsem naprogramoval s přispěním knihovny Weka, ve které jsou algoritmy naprogramovány poměrně obecně. Ukázkový příklad funguje (v rozumné době několika vteřin) tedy jen s omezenými vstupy (řádově stovky transakcí nad necelou desítkou slouců-cílů).

Jelikož je knihovna JDM velmi rozsáhlá, implementoval jsem jen potřebné třídy a rozhraní, aby program fungoval. Toto řešení tedy nebude vyhovovat plně celému rozhraní, ale s tímto se

ve specifikaci počítá a tak je i postavena. Ne každý dodavatel se totiž zabývá širokým polem všech metod, takže stačí jen vyhovět pravidlům, které jsou součástí specifikace, aby produkt například vyhovoval pro dolování asociačních pravidel. Ve specifikaci je přesně popsáno, které rozhraní je nutno implementovat. Já jsem se omezil pouze asi na desítku tříd. Musel jsem implementovat následující rozhraní (jsou v balíčku cz.zapletal.webminer.jdm):

- Connection a ConnectionFactory – umožňují připojení k jádru,
- ApacheLogPhysicalDataSet a ApacheLogPhysicalDataSetFactory – vstupní data (tyto třídy jsou schopny parsovat www statistiky serveru Apache a vytvořit z nich data vhodná pro dolování pravidel);
- AssociationSettings a AssociationSettingsFactory – udržuje nastavení pro algoritmy dolování pravidel (support, confidence, počet pravidel);
- BuildTask a BuildTaskFactory – vlastní implementace algoritmu (volání metod z Weky);
- AssociationModel a AssociationModelFactory – výsledek algoritmu;
- další pomocné rozhraní (AssociationRule, ExecutionHandle…).

Vlastní použití mého ukázkového příkladu, který jsem nazval WebMiner, přes rozhraní JDM je pak následující (soubor JDMTest.java):

```
// získáme továrnu na spojení
ConnectionFactory connectionFactory = new WMConnectionFactory();

// a získáme vlastní spojení
// (v praxi by se přidaly parametry jako DME server,
// uživatelské jméno a heslo a podobně)
Connection connection = connectionFactory.getConnection();

// továrna pro fyzická data
PhysicalDataSetFactory pdsFactory = (PhysicalDataSetFactory) connection
.getFactory("cz.zapletal.webminer.jdm.WMApacheLogPhysicalDataSet");

// načtení fyzických dat (v praxi by se v URI nastavily další
// parametry, například "soubor?matrixwidth=40&sparse=true")
String URI = "data/access_log_7days";
PhysicalDataSet buildData = pdsFactory.create(URI, true);

// uložení fyzických dat pro pozdější využití
connection.saveObject("myPhysicalData", buildData, true);

// vytvoření objektu pro nastavení asociačních pravidel
AssociationSettingsFactory lRSFactory = (AssociationSettingsFactory) connection
.getFactory("cz.zapletal.webminer.jdm.WMAssociationSettings");
AssociationSettings settings = lRSFactory.create();
```

```
// nastavení algoritmu pro získávání asociačních pravidel
//settings.setLogicalDataName("myLogicalData");
settings.setMinConfidence(0.8);
settings.setMinSupport(0.1);
settings.setMaxNumberOfRules(15);

// uložení objektu do kontejneru
connection.saveObject("mySettings", settings, true);

BuildTaskFactory lBuildTaskFactory = (BuildTaskFactory) connection
.getFactory("cz.zapletal.webminer.jdm.WMBuildTask");
BuildTask lTask = lBuildTaskFactory.create("myPhysicalData",
"mySettings", "myAssociationRulesModel");
connection.saveObject("myTask", lTask, true);

System.out.println("Spouštím vytvoření modelu");
ExecutionHandle lHandle = connection.execute("myTask");

// jelikož spuštění akce je v JDM implementováno asynchronně,
// musíme čekat, než se operace dokončí
lHandle.waitForCompletion(Integer.MAX_VALUE);
System.out.println("Model vytvořen");

// zjistíme, s jakým výsledkem akce skončila
ExecutionState lState = lHandle.getLatestStatus().getState();
if (lState.equals(ExecutionState.success)) {
System.out.println("Úspěch!");
} else {
System.out.println("Vytvoření neúspěšné!");
System.exit(1);
}

// nyní získáme z vytvořeného modelu asociační pravidla
// AssociationModel nabízí mnoho metod pro získávání pravidel,
// itemsetů, a to všech, nebo některých na základě filtrů a podobně
// já se omezím na obyčejné získání všech pravidel a vypsání na
// konzoli
AssociationModel model = (AssociationModel) connection
.retrieveObject("myAssociationRulesModel");

Iterator rulesIterator = model.getRules().iterator();
int i = 1;
System.out.println("Získaná pravidla:");

// všechna nalezená pravidla vypíšeme na konzoli
while (rulesIterator.hasNext()) {
AssociationRule rule = (AssociationRule) rulesIterator.next();
```

```
String ante = (String) rule.getAntecedent().getItems()[0];
String cons = (String) rule.getConsequent().getItems()[0];
float conf = (float) rule.getConfidence();
System.out.println("" + i++ + ": " + ante.toLowerCase()
+ " => " + cons.toLowerCase() + " (" + conf + ")");
}

// uzavřeme spojení
connection.close();
```

Jedná se o příklad, který by měl ukázat, jak se pracuje s JDM. Referenční implementace totiž vrací jen předem definované (stejné) výsledky, a neumožňuje tak si skutečnou práci „osahat“ naostro.

Nejprve se získá připojení, v tomto případě předpokládám, že knihovna (soubor JAR) JDM implementace je přiložena přímo u programu. V praxi se pro vytvoření spojení využívá hlavně JNDI. Poté se vytvoří objekt reprezentující fyzický zdroj dat a nastaví se tak, aby tato data načetl ze souboru data/access_log_7days.

V dalším kroku se nastaví parametry získávání pravidel, vytvoří vlastní úloha (BuildTask) a tato úloha se nad vytvořeným spojením spustí. Daná úloha vytvoří v repozitáři nový objekt pojmenovaný myAssociationRulesModel, který obsahuje vlastní nalezená pravidla.

Jak je vidět, úloha je v JDM spouštěna asynchronně, to znamená, že se musí čekat na dokončení operace.

Ze získaného modelu se v závěru příkladu extrahují pravidla a vypíší na obrazovku. Výstup by mohl vypadat následovně:

```
Start aplikace Java Data Mining - ukázka
Spouštím vytvoření modelu
Model vytvořen
Úspěch!
Získaná pravidla:
1: /linuxexpres/index.html=0 => /linuxexpres/index.html=0 (0.8227848)
2: /linuxexpres/index.html=1 => /linuxexpres/index.html=1 (0.8227848)
3: /linuxexpres/aktualne.html=0 => /linuxexpres/aktualne.html=0 (0.8227848)
4: /linuxexpres/aktualne.html=1 => /linuxexpres/aktualne.html=1 (0.8227848)
5: /mandrake/index.html=0 => /mandrake/index.html=0 (0.8227848)
6: /mandrake/index.html=1 => /mandrake/index.html=1 (0.8227848)
7: /linux/index.html=0 => /linux/index.html=0 (0.8227848)
8: /linux/index.html=1 => /linux/index.html=1 (0.8227848)
9: /linux_info.html=0 => /linux_info.html=0 (0.8227848)
10: /linux_info.html=1 => /linux_info.html=1 (0.8227848)
Konec aplikace Java Data Mining - ukázka
```

Přiložená ukázková aplikace vyžaduje přítomnost Java JRE ve verzi 1.4 nebo vyšší. Spustit se dá příkazem jdmtest.sh (případně jdmtest.bat pro systém Windows). Přiloženy jsou veškeré knihovny nutné pro běh (adresář lib) a testovací data (www log serveru o linuxu za měsíc leden 2005).

5. Závěr

Java Data Mining API je výsledkem společného úsilí mnoha firem, které do stanoveného rozhraní vkládají naděje, že usnadní zformovat nehetoregení trh na poli data miningu. Rozhraní je to značně rozsáhlé, aby dostatečně pokrylo danou problematiku, takže implementátoři musejí počítat s jistou mírou pracnosti. Velkou výhodou však je záruka kvality pro své zákazníky v případě, že splní veškeré požadavky dané specifikací.

Cílem této práce bylo presentovat dané rozhraní a na několika stranách stručně ukázat základní principy a možnosti JDM API.

Reference

[1] Kolektiv autorů. *Java Data Mining API.* Elektronická publikace. http://www.jcp.org/en/jsr/detail?id=73

[2] Kolektiv autorů. *Java API for OLAP.* Elektronická publikace. http://www.jcp.org/en/jsr/detail?id=69