Optimalizace řešení analytických úloh

Semestrální práce na kurz MI-KDD (zimní semestr 2015/2016)

Vypracoval: T. Frýda, J. Mráček

Zadání

Cílem této semestrální práce je automatizace nalezení požadovaného počtu hypotéz na analytické otázky procedury 4ft-Miner zadané uživatelem. Konečné zadání této práce bylo přesně definováno ve 3. verzi Úvodní zprávy k semestrální práci a případné drobné odlišnosti jsou uvedeny v popisu příslušných částí.

V následujících odstavcích je detailně popsáno především nastavení konfiguračního souboru, což je jediná možná spojnice mezi skriptem a uživatelem, který ho bude spouštět. Dále jsou také detailně rozebrány možnosti optimalizace a algoritmus binárního půlení, nastíněn výstup a také další možná vylepšení samotného skriptu.

Instalace

V příloze této zprávy lze najít archiv, který obsahuje složku LMOptimization. Tu je třeba po vybalení z archivu nakopírovat do adresáře Exec, který je umístěn ve složce s nainstalovaným programem LispMiner.

Struktura programu

Samotný skript je rozdělen do několika dílčích *.lua souborů, které se společně nachází ve složce LMOptimization:

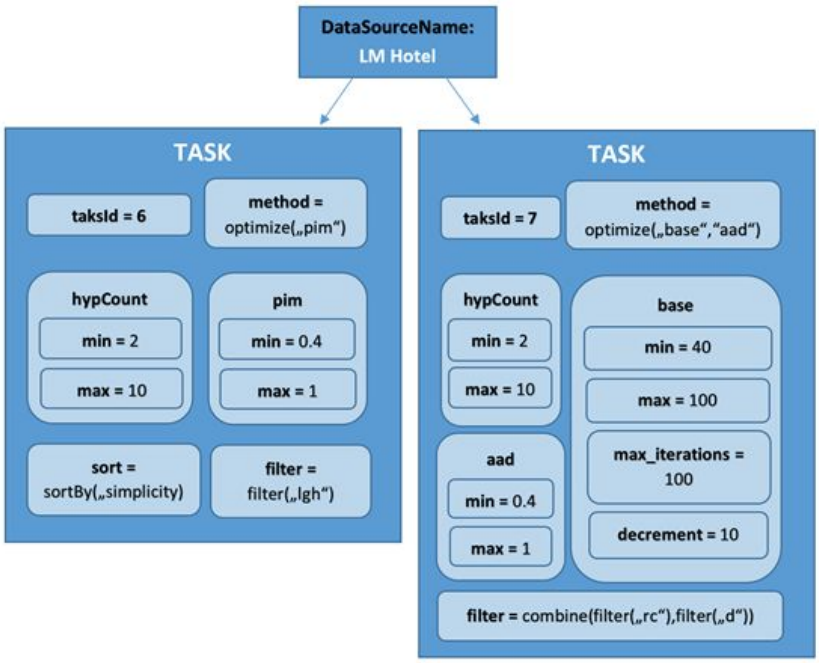
- **_LMOptimization.lua** Vstupní bod skriptu (spouští se v LMExec.exe)
- **Params.lua** Definice uživatelských vstupů (popsáno níže)
- **ParamsDefault.lua** Výchozí parametry (při jejich absenci v předchozím souboru)
- **OptimizationManager.lua** Řídí celý proces hledání hypotéz
- **Functions.lua** Funkce pro řazení a filtrování hypotéz
- **Output.lua** Generuje výstupní zprávu ve formátu HTML

Vstup

Vstupem programu jsou data a metabáze, nad kterými bude probíhat hledání hypotéz. Uživatel si tak před spuštěním skriptu jednoduše nastaví v LM Workspacu analytické otázky pro 4ft-Miner, ke kterým se budou hledat nějaké hypotézy. U těchto otázek budou uživatelem vyplněné cedenty včetně jejich rozmezí (minimálního a maximálního počtu) a také kvantifikátory včetně jejich hodnot, které budou skriptem použity jako spodní mez *a*. Dalším vstupem bude konfigurační soubor ve formátu lua, který je popsán níže.

Struktura konfiguračního souboru:

Konfigurační soubor obsahuje v úvodu jméno datového zdroje (z LM Workspacu), ve kterém jsou definované úlohy, ke kterým se budou následně tímto skriptem vyhledávat hypotézy. Následuje zadání jedné, nebo více samostatných úloh:



Parametr:	Možné hodnoty:	Popis:
taskId		identifikátor úlohy z LM Workplacu
method	“base”, “pim”, “aad”	jaký kvantifikátor bude optimalizován
hypCount	min = ..., max = ...	požadovaný počet hypotéz
base	min = …, max = …, max_iterations = …, decrement = ...	nastavení minima a maxima nastavení maximálního počtu iterací hodnota znížení base v každé iteraci
pim	min = …, max = ...	minimální a maximální hranice p-Implikace
aad	min = …, max = ...	minimální a maximální hranice Above Average Dependence
sort	“simplicity”	nastavení způsobu řazení hypotéz
filter	“top1”, “top2”, …, “top10” “rc”, “d”, “lgh”	prvních 1, 2, …, 10 rc = odstranění nadbytečné podmínky d = odstranění duplicitních hypotéz lgh = odstranění méně obecných hypotéz

Ukázka konfiguračního souboru

```
require("ParamsDefault")
dataSourceName = "LM Hotel MB"
config = { -- kolekce konfiguraci pro n uloh
{ -- zacatek konfigurace ulohy
       taskId = 6, -- id ulohy (ve workspacu vpravo nahore pri otevrene uloze)
       method = optimize("pim"), -- zpusob optimalizace (viz zprava k semestralni praci)
       hypCount = { -- nastavuje rozmezi pro pocet hypotez
              min=2,
              max=10
       },
       pim = { -- nastavuje rozmezi testovane (fundovane) p-implikace
              min = 0.4,
              max = 1
       },
       sort=sortBy("simplicity"), -- razeni podle poctu cedentu
       filter=filter("lgh")
},{
       … NEXT TASK …
}}
```

Popis možností konfigurace

V konfiguraci máme tři parametry, které přijímají na svůj vstup funkci. První z nich je `method`, která přijímá binární funkci, jejichž argumenty jsou `task` a konfigurace pro danou úlohu. Výstupem této funkce je nanejvýš dvojice hodnot, která reprezentuje nalezené optimum. V případě, že nebylo optimum nalezeno, vrátí daná funkce `nil`. Druhým parametrem, který přijímá funkci, je parametr `sort`, který přijímá unární funkci. Tato funkce přijme tabulku hypotéz a vrátí nějakým způsobem seřazenou tabulku hypotéz. Posledním parametrem, který přijímá funkci je `filter`. Tento parametr přijímá unární funkci, která přijme tabulku hypotéz a z nich nějaké vybere. Ve výchozím stavu používáme u parametru `method` funkci `optimize`, která se postará o zvolení správné možnosti optimalizace kvantifikátorů a zároveň ošetřuje vstup od uživatele - například kvantifikátor p-implikace se dá zadat jako “pim”, “p-im”, “Pim” a podobně. Důležité je, že tato funkce vrací tabulku, ve které jsou klíče `first`, `second` a `fn`. Klíče `first` a `second` odpovídají kvantifikátorům, které se optimalizují. Klíč `fn` funkci, která provádí optimalizaci. Funkce `optimize` tuto funkci dodá. Tato dodaná funkce optimalizuje kvantifikátor v klíči `first` pomocí binárního půlení a pokud nějaký je, tak kvantifikátor v klíči `second` pomocí lineárního prohledávání od maxima k minimu.

Je možné použít i nějakou jinou funkci, která optimalizuje, ale k zjednodušení konfigurace jsme zvolili tento přístup. Je však nutné ji poskytnou v tabulce, která byla popsaná v předchozím odstavci (viz výstup funkce `optimize`). Podobně je to i u parametrů `sort` a `filter`. Ale vzhledem k tomu, že se dá očekávat, že i běžný uživatel bude chtít mít možnost přidat řadící či filtrovací funkci, tak svou funkci může zaregistrovat do tabulky `sortDispatcher` resp. `filterDispatcher` a následně volat přes unifikované rozhraní (unární funkce `sortBy` a `filter`). Pokročilejší uživatele by mohla potěšit jednoduchost přidání například multikriteriálního řazení: za předpokladu, že chci primárně řadit stabilním řazením sort1, dále pak sort2 až sortN, pak stačí do parametru sort napsat následující kód:

```
{...,
sort=foldl(combine,sort1,sort2,...,sortN),
...}
```

Mezi filtry, které jsme implementovali, patří:

- **filter(“d”)**
  odstranění duplicitních hypotéz (hypotéz, které mají stejnou textovou reprezentaci)
- **filter(“rc”)**
  odstranění hypotéz, které jsou až na podmínku stejné s jinou hypotézou bez podmínky
- **filter(“lgh”)**
  odstranění méně obecné hypotézy (obecnější hypotéza má stejný antecedent, succedent a podmínka musí být podmnožinou succedentu a podmínky méně obecné porovnávané hypotézy)

Filtry je také možné kombinovat mezi sebou, jak je ukázáno na následujícím příkladu:

```
{...,
filter=combine(filter(“rc”),filter(“d”))
...}
```

Důležité je také zmínit, že kromě parametrů `taskId, method` a `hypCount,` jsou všechny parametry volitelné. Výchozí hodnoty je možné nalézt na konci souboru `“VychoziParametry.lua”`. Během testování jsme zjistili, že získávání parametrů přímo z metabáze může být pro uživatele matoucí, tak jsme určili následující priority nastavení:

1. pokud je parametr nastaven v souboru `“Parametry.lua”`
2. pokud byl předchozí bod neúspěšný načteme hodnotu z metabáze
3. pokud byl předchozí bod neúspěšný načteme výchozí hodnotu proměnné

Pokud se jedná o parametry, které se v metabázi nenacházejí, přeskočíme druhý bod.

Bližší popis možnosti optimalizace kvantifikátorů

1. **BASE bude konstantní**
   hledání optima pro PIM nebo AAD (případně další) binárním půlením
2. **ostatní kvantifikátory budou pevně dané**
   optimalizace hodnoty BASE
3. **hledání optima pro BASE a jeden kvantifikátor** (PIM, AAD, apod..).
   V tomto případě budeme hledat optimum pro daný kvantifikátor pomocí binárního půlení a Base budeme snižovat o konstantu pokud nenalezneme řešení. Případně naopak.

Algoritmus

Binární půlení

1. provedení nalezení hypotéz tím se získá prahová hodnota a uloží se do proměnné *b*
2. nastavení hodnoty *a* na hodnotu definovanou v konfiguračním souboru (např. 0.4) pokud je vyšší než minimální prahová hodnota v opačném případě nastavíme hodnotu *a* na minimální prahovou hodnotu
3. pokud je pro *b* počet hypotéz větší nebo rovný požadovanému počtu hypotéz končíme
4. pokud se nalezl správný počet hypotéz nebo byl překročen počet iterací končíme
5. nastavíme hodnotu pro hledání hypotéz na *(a+b)/2*
6. pokud je počet hypotéz větší než byl požadovaný nastavíme *a* na *(a+b)/2* v opačném případě nastavíme *b* na *(a+b)/2*
7. přejdeme do kroku č. 4

Výstup

V ideálním případě je výstupem report s požadovaným počtem hypotéz. Zde se pokoušíme zajistit, aby ve výsledku nebyly víckrát obměny jedné hypotézy, což by mohlo uživatele plést - cílem tedy je ve výsledku mít pouze hypotézy logicky nezávislé. Když se podaří nalézt méně, případně více hypotéz, než uživatel požadoval, budou všechny zobrazeny s komentářem, že méně / více se jich nalézt nepodařilo.

Výsledný report lze nalézt pod názvem *LMOptimization_Result.html* v instalačním adresáři LispMineru v podsložce *Exec/Export*.

Potenciální důvody neúspěchu v hledání hypotéz:

1. příliš vysoká hodnota *a* (dolní mez binárního půlení)
2. hypotézy mohou být u sebe tak blízko, že ani při rozdělení původního intervalu na $2^{počet\ iterací}$ dílů nelze nalézt díl ve kterém byl požadovaný počet hypotéz
3. prahová hodnota *b* byla prvním během nastavena na zbytečně nízkou hodnotu analytická otázka neobsahuje žádnou hypotézu, která by byla pro uživatele zajímavá

Nedostatky a vylepšení

- možnost optimalizovat maximálně přes dva kvantifikátory s tím, že jeden z parametrů se optimalizuje binárním půlením a druhý postupným snižováním o konstantu; pro více proměnných by se už mohlo vyplatit použít jiný přístup (evoluční strategie,...)
- neschopnost spustit a nastavit vše na jednom místě. Je nutné část nastavit v LMWorkspace, část nastavit v konfiguračním souboru a následně celou úlohu spustit v LMExec, což může způsobit mírné zmatení
- chybí optimalizace parametrů zadaných cedentů
- optimalizace kvantifikátorů funguje pouze pro 4ft úlohy
- chybí úplná specifikace rozhraní, přes které by se mohly jednotlivé moduly dorozumívat.
- při práci s opravdu velkými daty by mohla chybět možnost provádění paralelních výpočtů, které by práci rozdělili do více vláken procesoru

Závěr

Hlavním smyslem této práce bylo pro náš dvoučlenný tým vyzkoušet si řešení analytických úloh také mimo LMWorkspace, pouze pomocí poskytnutého rozhraní pro jazyk LUA respektive jeho variantou pro LispMiner nazvanou LMCL (LispMiner Control Language).

Mnohdy jsme bojovali s nedostatkem informací, které jsme ale nakonec nalezli jako součást dokumentace, kde jsou vysvětlené jak základy práce s LMExec, tak na názorných diagramech představeny základní vztahy mezi jednotlivými součástmi systému. Dalším užitečným zdrojem byla vzorová ukázka, ze které jsme čerpali příklady k použití jednotlivých funkcí nalezených v dokumentaci.

Tato semestrální práce byla pro nás pro oba velice přínosnou zkušeností. Při vytváření skriptu bylo mnohdy nutné pochopení mnohem hlubších detailů řešení analytických úloh, než které musí řešit a zvažovat běžní uživatelé. Také seznámení, respektive zdokonalení se ve funkcionálním programování považujeme za velké plus. Věříme, že se úvodní cíle podařilo splnit a výsledný skript je funkční a případně v budoucnu dále rozšiřitelný.