UNIVERZITA KOMENSKÉHO V BRATISLAVE
FAKULTA MATEMATIKY, FYZIKY A INFORMATIKY

Porovnanie dataminingových metód MS analytického servera s knižnicami štatistického jazyka R

DIPLOMOVÁ PRÁCA

2017 **Bc. Matej MALÝ**

UNIVERZITA KOMENSKÉHO V BRATISLAVE
FAKULTA MATEMATIKY, FYZIKY A INFORMATIKY

Porovnanie dataminingových metód MS analytického servera s knižnicami štatistického jazyka R

DIPLOMOVÁ PRÁCA

Študijný program: Ekonomická a finančná matematika a modelovanie
Študijný odbor: 1114 Aplikovaná matematika
Školiace pracovisko: Katedra aplikovanej matematiky a štatistiky
Vedúci práce: RNDr. Igor Odrobina, CSc.

Bratislava 2017

Bc. Matej MALÝ

Univerzita Komenského v Bratislave
Fakulta matematiky, fyziky a informatiky

ZADANIE ZÁVEREČNEJ PRÁCE

Meno a priezvisko študenta: Bc. Matej Malý
Študijný program: ekonomicko-finančná matematika a modelovanie (Jednoodborové štúdium, magisterský II. st., denná forma)
Študijný odbor: 9.1.9. aplikovaná matematika
Typ záverečnej práce: diplomová
Jazyk záverečnej práce: slovenský
Sekundárny jazyk: anglický

Názov: Porovnanie dataminingových metód MS analytického servera s knižnicami štatistického jazyka R.
Comparison of the datamining methods in MS Analytic Server and methods in libraries of the language R.

Cieľ: Študent na vybraných dátach porovná výhody a nevýhody dvoch alternatívnych prístupov k analýze dát. Predpoklady: Absolvovanie predmetov Databázy a Databázy SQL, zvládnutie základov práce s MS SQL serverom poskytnutom v študentskom balíku DreamSpark.

Vedúci: RNDr. Igor Odrobina, CSc.
Katedra: FMFI.KAMŠ - Katedra aplikovanej matematiky a štatistiky
Vedúci katedry: prof. RNDr. Daniel Ševčovič, CSc.

Dátum zadania: 21.01.2016

Dátum schválenia: 25.01.2016

prof. RNDr. Daniel Ševčovič, CSc.
garant študijného programu

....

študent vedúci práce

Poďakovanie Touto cestou by som sa chcel poďakovať môjmu školiteľovi RNDr. Igorovi Odrobinovi, CSc. za jeho ochotu pri konzultovaní tejto práce. Veľká vďaka patrí mojej rodine a kolegom, ktorí mi pomohli a motivovali ma k čo najlepšej práci.

Abstrakt

MALÝ, Matej: *Porovnanie dataminingových metód MS analytického servera s knižnicami štatistického jazyka R* [Diplomová práca], Univerzita Komenského v Bratislave, Fakulta matematiky, fyziky a informatiky, Katedra aplikovanej matematiky a štatistiky; školiteľ: RNDr. Igor Odrobina, CSc., Bratislava, 2017, 60s.

Predmetom tejto práce je predstaviť vybrané štatistické metódy a ich realizáciu použitím viacerých prístupov. Tradičný spôsob predstavuje štatistický softvér R a jeho knižnice. Novinkou je jeho integrácia do jazyka SQL prostredníctvom SQL Serveru 2016, ktorý ponúka doplnok R Services. Tento prístup umožňuje realizáciu externého R skriptu a tým zefektívni spracovanie a prístup k dátam. To znamená, že sa môžeme vyhnúť ich zbytočnému prenášaniu a načítavaniu. V ďalších častiach budeme porovnávať výpočtový čas realizácie vybraných štatistických metód. Keďže druhý spomínaný spôsob ponúka oveľa rýchlejšiu realizáciu, pokúsime sa k týmto výpočtovým časom priblížiť aj prostredníctvom tradičného prístupu, teda softvérom R, využitím paralelného programovania.

Kľúčové slová: datamining, SQL server, softvér R

Abstract

MALÝ, Matej: Comparison of the datamining methods in MS Analytic Server and methods in libraries of the language R [Master Thesis], Comenius University in Bratislava, Faculty of Mathematics, Physics and Informatics, Department of Applied Mathematics and Statistics; Supervisor: RNDr. Igor Odrobina, CSc., PhD., Bratislava, 2017, 60p.

The subject of this thesis is to introduce selected statistical methods and their implementation through more approaches. The traditional way is the statistical software R and its libraries. The new feature is its integration to SQL through SQL Server 2016, which offers the addition R Services. This approach makes it possible to implement an external R script and provide effective data access. This means that we can avoid unnecessary data transfer and its loading. In other sections, we will compare the process time of realization on selected statistical methods. Since the second mentioned method offers a much faster realization, we will try to reach this process time through the traditional approach, thus software R, using parallel programming.

Keywords: datamining, SQL server, software R

Obsah

Zoznam obrázkov 9

Úvod 11

1 Dataminingové metódy 13

2 Klasifikačné algoritmy 15

2.1 Rozhodovacie stromy 15

2.1.1 Základný algoritmus vytvárajúci rozhodujúci strom 16

2.1.2 Výber atribútu pre deliace kritérium 18

2.1.3 Orezávanie stromu 21

2.2 Klasifikačné lesy 22

2.2.1 Bagging 23

2.2.2 Boosting a arcing 23

2.2.3 Náhodné lesy 24

2.3 Logistická regresia 24

2.3.1 Algoritmus prostredníctvom Newtonovej metódy 25

2.4 K najbližších susedov 27

3 Integrácia R do SQL 28

3.1 Využitie integrácií a popis postupov data scientistov 28

3.1.1 Externý R skript v Transact-SQL 30

3.1.2 Prostredie R použitím knižnice RevoScaleR 32

4 Návody na zrýchlenie výpočtového času v R 35

4.1 Jedno R prostredie, jedno jadro 36

4.2 Paralelné programovanie v R 40

4.2.1 Jeden program pre každé jadro 40

4.2.2 Jeden program pre viac jadier 42

5 Porovnanie prístupov 49

5.1 Výsledky pre logistickú regresiu 49

5.2 Výsledky pre náhodný les 52

5.3 Výsledky pre k najbližších susedov 54
5.4 Zhrnutie výsledkov 55

Záver **57**

Zoznam použitej literatúry **59**

Zoznam obrázkov

1 Proces dataminingu 13
2 Príklad rozhodovacieho stromu 15
3 Scenáre vetvenia pre rôzne typy atribútov 17
4 Porovnanie orezaného a neorezaného stromu 21
5 Zobrazenie pripojenia a pracovania s použítím servera 29
6 Spúštanie uložených procedúr na serveri 30
7 Výstup z príkladu č. 1 32
8 Výstup z príkladu č. 2 32
9 Porovnanie výpočtového času funkcie $f(x) = x^2$ pre vopred zadefinovaný vektor a vektor postupne narastajúci 37
10 Porovnanie výpočtového času funkcie $f(x) = 2\pi sin(x)$ pre výpočet 2π vo vnútri a po vykonaní cyklu 38
11 Porovnanie výpočtového času funkcie $f(x) = 2\pi sin(x)$ pre výpočet pomocou cyklu a bez neho 39
12 Zobrazenie R prostredí vytvorených na pozadí pri zadefinovanom paralelnom rozhraní 43
13 Percentuálne znázornenie využitia procesora pri spustení programu R . 44
14 Percentuálne znázornenie využitia procesora pri spustení paralelného rozhrania v programe R 44
15 Porovnanie výpočtového času `sum(a)` pre rôzne dĺžky vektora a sekvenčne a paralelne 47
16 Porovnanie výpočtového času `sum(a)` vzhľadom na operáciu pre rôzne dĺžky vektora a sekvenčne a paralelne 47
17 Porovnanie výpočtového času `sum(sin(a))` pre rôzne dĺžky vektora a sekvenčne a paralelne 48
18 Porovnanie výpočtového času `sum(sin(a))` vzhľadom na operáciu pre rôzne dĺžky vektora a sekvenčne a paralelne 48
19 Porovnanie výpočtového času trénovania modelu logistickej regresie pre jednotlivé prístupy 51

20 Porovnanie výpočtového času trénovania modelu náhodný les pre jednotlivé prístupy 53
21 Porovnanie výpočtového času klasifikácie metódou k najbližších susedov pre jednotlivé prístupy 55

Úvod

V auguste 2016 ponúkla firma Miscrosoft verziu SQL Server 2016, ktorá obsahuje integráciu jazyka R do tohto systému. Túto možnosť umožňuje doplnok R Services, ktorý je potrebný aktivovať pri inštalácii SQL Serveru. Prepojenie pomáha zlepšovať prácu s veľkými dátovými súbormi, pretože môžeme využívať rýchlosť SQL a zároveň pracovať so štatistickými metódami, ktoré by bolo zložité vytvoriť v jazyku SQL. Firma Microsoft taktiež ponúka balík RevoScaleR, ktorý obsahuje viacero štatistických metód a lepšiu prácu s dátami. Tieto priamo nenačítava do operačnej pamäte, ale zachováva v pamäti iba prístupovú cestu k nim. Pri práci s dátami ich potom načítava po častiach. Taktiež sme si všimli, že na pozadí otvára ďalšie prostredia, ktoré zrýchľujú výpočtový čas. Tieto dva prístupy teda pracujú oveľa efektívnejšie ako klasické knižnice v softvéri R. My sa preto v našej práci pokúsime priblížiť k týmto časom paralelnou realizáciou výpočtov dostupnými knižnicami. Porovnáme preto štyri prístupy: externý skript v programe MS SQL Server Management Studio, balík RevoScaleR, klasické štatistické balíky v softvéri R, paralelnú realizáciu tiež prostredníctvom R.

V prvých kapitolách si predstavíme ideu a konštrukciu vybraných metód, na ktorých budeme dané porovnania sledovať. Uvedieme metódy ako rozhodovacie stromy, náhodné lesy, logistická regresia a k najbližších susedov. Kvôli vypracovaniu paralelného výpočtu týchto metód bude veľmi dôležité pochopenie ich konštrukcie.

V tretej kapitole ukážeme ako pracovať s balíkom RevoScaleR. Táto časť ponúkne ukážky ako narábať s dátami od prvých krovov cez prihlásenie sa do databázy, vytvorenie výpočtového prostredia a zapísanie výsledkov do nových tabuliek v databázach. Budeme pracovať aj s Microsoft SQL Server Management Studiom, v ktorom ukážeme integráciu R do jazyka SQL. Uvedieme postupy ako pracovať s externým skriptom, ako do neho vybrať potrebné dáta a správne pracovať s výstupmi, ktorých výslednú formu musíme poznať vopred. Preto je potrebné zadefinovať vektor výstupov, prípadne tabuliek.

V predposlednej kapitole ukážeme ako zrýchliť niektoré časti kódu pri práci so štatistickým softvérom R. Tieto rady a tipy budeme potom využívať pri ďalšej práci. Spravíme aj úvod do paralelného programovania a knižníc, ktoré ho umožňujú. Tiež vykonáme simulácie na pochopenie tejto problematiky a overíme niektoré skutočnosti,

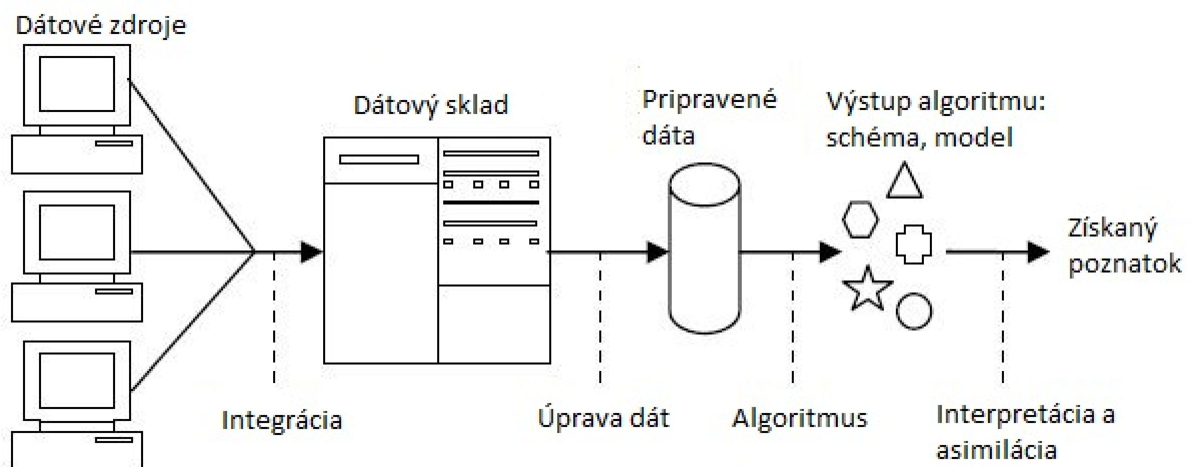
kedy je paralelné programovanie užitočné a kedy naopak zbytočné.

V poslednej kapitole spracujeme postupy ako paralelne vykonávať vybrané štatistické metódy na základe informácií uvedených v prvej kapitole. Po vykonaní simulácií pre rôzne veľkosti dát graficky znázorníme porovnanie výpočtových časov pre vyššie uvedené prístupy. Z daných výsledkov vytvoríme závery ohľadom najrýchlejšieho prístupu, vyhodnotíme ako sa naša paralelná implementácia metód priblížila najrýchlejšej, a či je efektívnejšia ako štandardný prístup pomocou štatistických balíkov dostupných v softvéri R.

1 Dataminingové metódy

Datamining ako termín, je skúmanie a analýza veľkého rozsahu dát za účelom objavenia zmysluplných súvislostí a pravidiel. Cieľom tejto oblasti je umožnenie korporáciám zlepšenie svojho marketingu, predaja, a prevádzky zákazníckej podpory prostredníctvom lepšieho pochopenia ich zákazníkov. Metódy dataminingu sú taktiež použiteľné aj v iných oblastiach ako medicína, kontrola priemyselných procesov a podobne.

Proces dataminingu pozostáva z viacerých krokov zobrazených na obrázku 1.



Obr. 1: Proces dataminingu

Prvým krokom tohto procesu je zber dát z rôznych zdrojov. Tie sú integrované a uložené do nejakého dátového skladu, kde sa dáta upravujú od rôznych nečistôt a outlierov. Takto upravené dáta môžeme následne použiť v dataminingových algoritmoch, ktoré produkujú výstupy vo forme modelov a rôznych schém. Interpretáciou a asimiláciou výstupov sa získajú rôzne poznatky dôležité pri ďalšom rozhodovaní. Z obrázku 1 vyplýva, že kľúčovým prvkom sú algoritmy dôležité na získavanie poznatkov z ich výstupov.

Algoritmy v dataminingu sú množinou heuristiky a výpočtov, ktoré vytvárajú z dát modely. Na vytvorenie modelu algoritmus najprv analyzuje zadané vstupné dáta. Algoritmus používa výsledky analýzy viacerých iterácií, za účelom nájdenia optimálnych parametrov pre dataminingový model. Aplikovaním týchto parametrov na celý súbor dát získavame hľadané vzory a detailné štatistiky.

Dôležitou úlohou pri analytických úlohách je výber správneho algoritmu. Hoci mô-

žeme použiť viaceré dataminingové algoritmy na vykonanie rovnakých úloh, každý algoritmus pracuje inak a môže vyprodukovať odlišné výsledky. Napríklad algoritmus rozhodovacie stromy môžeme použiť nie len na predikciu, ale tiež ako spôsob redukcie dátového setu, pretože rozhodovacie stromy môžu identifikovať stĺpce, ktoré neovplyvňujú konečný dataminingový model.

Typy algoritmov:

- **Klasifikačné:** predikcia jednej alebo viacerých diskrétnych premenných.
- **Regresné:** predikcia jednej alebo viacerých spojitých numerických premenných, ako zisk alebo strata.
- **Segmentačné:** rozdelenie dát na skupiny alebo klastre prvkov, ktoré majú podobné vlastnosti.
- **Asociačné:** hľadanie korelácie medzi rôznymi atribútmi v dátovom sete

My sa v našej práci zameriame na klasifikačné algoritmy, ktoré budeme vzhľadom na skúmané dáta využívať najviac. Preto je ďalšia kapitola venovaná princípom a konštrukcii týchto metód.

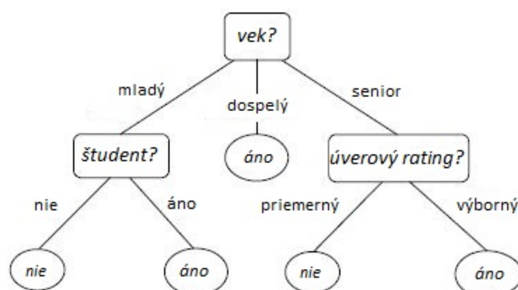
2 Klasifikačné algoritmy

Klasifikácia je forma analýzy dát, ktorá extrahuje modely popisujúce dátové triedy. Takéto modely predikujú kategorické (diskrétne) označenia tried. Veľa klasifikačných modelov bolo navrhnutých odborníkmi z oblasti strojového učenia a štatistiky. Klasifikácia má mnoho aplikácií vrátane odhaľovania podvodov, cieleného marketingu, predikcií výkonnosti, výroby a lekárskej diagnostiky. V tejto kapitole predstavíme hlavnú myšlienku klasifikácie, základné techniky na budovanie rozhodovacích stromov, náhodných lesov, logistickej regresie a k najbližších susedov.

Prvým krokom klasifikačných algoritmov je učenie alebo trénovacia fáza, kde algoritmus buduje klasifikátor analyzovaním trénovacej množiny zloženej z pozorovaných parametrov a k nim prislúchajúcej triedy. Pozorované parametre reprezentuje p-rozmerný vektor nazývaný **vektor atribútov:** $x = (x_1, \ldots, x_p)$. Každý prvok v trénovacej množine má priradenú triedu y z množiny C. Tento parameter nazývame **klasifikačný atribút**.

2.1 Rozhodovacie stromy

Na základe knihy [1] uvedieme teoretický úvod k tejto metóde. Rozhodovacie stromy sú dataminingové modely vytvárajúci rad vetvení a uzlov. Tieto uzly delíme na terminálne a neterminálne. Každý neterminálny uzol obsahuje test na atribút. Z týchto uzlov vedú vetvy, ktoré na základe testu v neterminálnom uzle smerujú buď do ďalšieho neterminálneho uzlu, alebo do uzlu terminálneho nazývaného aj ako list, ktorý obsahuje označenie triedy. Vrchný uzol stromu sa nazýva koreň.



Obr. 2: Príklad rozhodovacieho stromu

Na obrázku 2 môžeme vidieť ukážku rozhodovacieho stromu, ktorý znázorňuje, či si klient kúpi, respektíve nekúpi počítač. V hranatých políčkach sa nachádzajú netermínované uzly obsahujúce test atribútov. V oválnych políčkach sú termínované uzly reprezentujúce triedy *áno* a *nie*, ktoré identifikujú triedu klienta.

2.1.1 Základný algoritmus vytvárajúci rozhodujúci strom

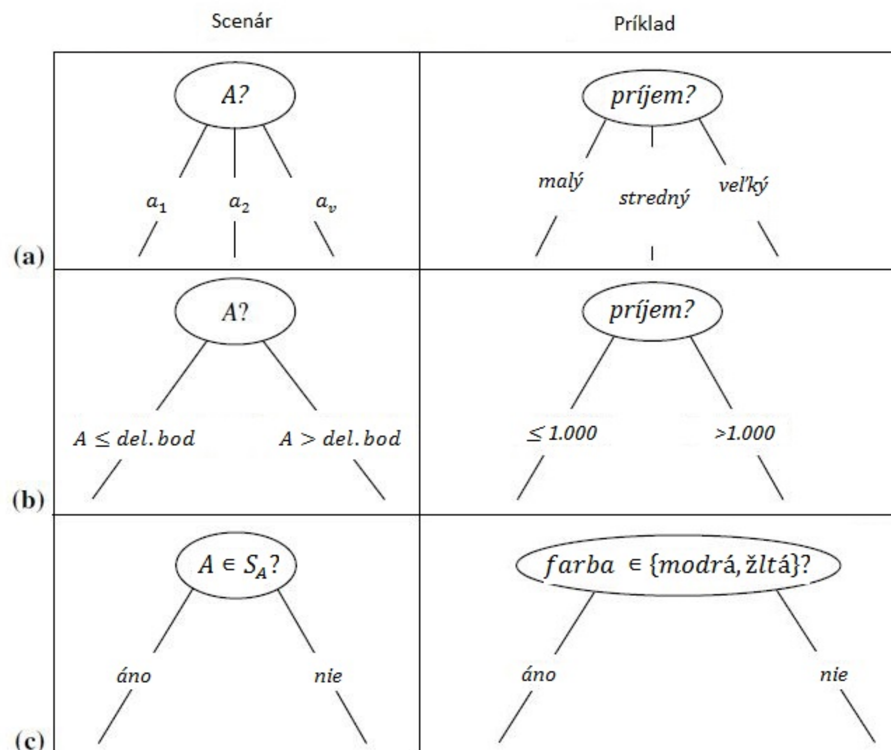
Rozhodovacie stromy sú konštruované rekurzívne spôsobom zhora nadol. Algoritmus obsahuje vstupné parametre: zoznam atribútov *attribute list*, metóda výberu delenia *selection method* , dátový set D pozorovaní s prislúchajúcou triedou, na ktorom sa bude strom trénovať. Algoritmus:

- Vytvorí sa uzol N. Potom sa overia 2 podmienky:
    1. ak sú vzorky v trénovacom dátovom D sete rovnakej triedy C, potom sa uzol N stáva listom a je označený touto triedou
    2. ak je *attribute list* prázdny, potom sa uzol N stáva listom označeným majoritnou triedou z množiny D
- V opačnom prípade sa aplikuje *selection method* k určeniu deliaceho kritéria. Deliace kritériu nám hovorí, ktorý atribút bol určený *selection method* ako najlepšia možnosť delenia vzoriek z D. Tiež nám hovorí o vetvách vychádzajúcich z uzlu N s ohľadom na výsledky zvoleného testu. Presnejšie povedané, deliace kritérium nám určuje atribút testu a tiež indikuje buď deliaci bod alebo deliacu podmnožinu. Kritérium je určené tak, aby výsledné delenie v každej vetve bolo čo najčistejšie. Delenie je čisté, ak všetky vzorky v ňom patria do rovnakej triedy.
- Uzlu N je priradené deliace kritérium, ktoré slúži ako test v uzle. Pre všetky výsledky z delenia vyrastá z uzlu vetva. Sú tri rôzne scenáre podľa typu deliaceho atribútu A zobrazené na obrázku 3.
    1. A je diskrétna premenná. V tomto prípade výstupy z uzla N priamo zodpovedajú známym hodnotám. Vetva je vytvorená pre každú hodnotu a_j z A, a označená touto hodnotou. Delenie D_j je podmnožina vzoriek s vlastnosťou

a_j. Pretože všetky vzorky vložené do vetvenia majú hodnotu atribútu A, môžeme tento atribút vyhodiť z množiny atribútov pre ďalšie delenia.

2. A je spojitá premenná. V tomto prípade test v uzle N má dva možné výstupy zodpovedajúce podmienke $A \leq deliacibod$ a $A > deliacibod$, kde je deliaci bod súčasť deliaceho kritéria. Z uzlu N vychádzajú dve vetvy, ktoré rozdeľujú pozorovania na podmnožiny D_1, pre ktoré je podmienka $A \leq deliacibod$ splnená a D_2, ktorá obsahuje doplnok.

3. A je diskrétna premenná a musí tvoriť binárnu vetvu. Test má tvar $A \in S_A$, kde S_A je podmnožina A. Ak atribút A nadobúda hodnotu a_j z A a ak $a_j \in S_A$, potom je podmienka v teste splnená. Z uzlu N sa vytvoria dve vetvy. Ľavá vetva je označená ako *áno* a zodpovedá podmnožine D_1, do ktorej spadajú pozorovania spĺňajúce podmienku v uzle N. Pravá vetva je označená *nie* a zodpovedá podmnožine D_2, ktorej prvky nespĺňajú test.



Obr. 3: Scenáre vetvenia pre rôzne typy atribútov

- Algoristmus používa rovnaký proces rekurzívne na formovanie rozhodovacieho stromu pre každé výsledné delenie.

- Rekurzívne delenie sa končí, ak je splnená jedna z nasledujúcich podmienok:
    1. Všetky vzorky z D v danom uzle majú rovnakú triedu.
    2. Nezostali žiadne ďalšie atribúty, na základe ktorých by mohli byť vzorky ďalej delené. V tomto prípade sa použije majoritná trieda, čiže z uzlu bude list označený triedou s najväčším zastúpením.
    3. Nezostali žiadne vzorky pre danú vetvu, to znamená, že delenie D_j je prázdne. V tomto prípade vytvoríme list s majoritnou triedou v D.

2.1.2 Výber atribútu pre deliace kritérium

Najťažšou úlohou pri tvorbe rozhodovacích stromov je výber správnej podmienky do deliaceho kritéria. Úlohou je nájsť atribút delenia, ktorý by najlepšie rozdelil vzorky z dátového setu D na čo najčistejšie podmnožiny. Túto vlastnosť popisuje miera deliaceho atribútu. Atribút s najlepšou hodnotu miery je zvolený ako deliaci atribút. Najznámejšie miery popisujúce tieto vlastnosti sú: *information gain, gain ratio, Gini index.* V tejto časti budeme používať značenie, kde m je počet tried, C_i pre $i = 1, \ldots, m$ je množina vzoriek $i-$tej triedy z D, $|D|$ je počet vzoriek v D, $|C_{i,D}|$ je počet vzoriek triedy C_i v D.

Information Gain

Information Gain alebo informačný zisk je miera založená na informačnej teórii, pri ktorej sa daný atribút snaží minimalizovať informáciu potrebnú na klasifikáciu vzoriek do daných delení s najmenšou nečistotou. Tento prístup minimalizuje očakávaný počet testov za účelom nájdenia najjednoduchšieho rozhodovacieho stromu.

Očakávaná informácia potrebná na klasifikáciu vzoriek je daná ako:

$$Info(D) = -\sum_{i=1}^{m} p_i \log_2(p_i), \tag{1}$$

kde p_i je nenulová pravdepodobnosť, že ľubovoľná vzorka z D patrí do C_i. Jej odhad je určený ako $|C_{i,D}|/|D|$. Logaritmus so základom 2 je dôsledkom kódovania v bitoch. $Info(D)$ je priemerné množstvo informácie potrebnej k identifikácii označenia triedy. $Info(D)$ sa tiež nazýva aj ako **entropia**.

Predpokladajme teraz delenie vzoriek podľa atribútu A, ktorý nadobúda rôzne hodnoty $a_1, a_2, \ldots, a_v$. A môže rozdeliť množinu D na podmnožiny $D_1, D_2, \ldots, D_v$, kde D_j obsahuje tie vzorky, ktoré nadobúdajú hodnotu a_j v A. Na určenie informácie, ktorú potrebujeme na presnú klasifikáciu definujeme:

$$Info_A(D) = \sum_{j=1}^{v} \frac{|D_j|}{|D|} \times Info(D_j). \tag{2}$$

Podiel $\frac{|D_j|}{|D|}$ vyjadruje váhu j-teho delenia. $Info_A(D)$ je očakávaná informácia, ktorú potrebujeme ku klasifikácii vzoriek z D založenej na delení prostredníctvom A.

Informačný zisk je potom definovaný ako:

$$Gain(A) = Info(D) - Info_A(D).$$

Inak povedané, *Gain* je očakávaná zmena informačnej požiadavky spôsobená definovaním atribútu A. A s najväčším informačným ziskom, $Gain(A)$, je vybraný ako deliaci atribút v uzle N.

Pre atribút, ktorý nadobúda spojité hodnoty musíme určiť deliaci bod. Najprv usporiadame hodnoty A vzostupne. Každý stredný bod medzi dvoma hodnotami je považovaný za možný deliaci bod. Preto ak má A počet hodnôt v, dostaneme $v - 1$ možných delení, ktoré získame:

$$\frac{a_i + a_{i+1}}{2}.$$

Pre každý takýto bod určíme $Info_A(D)$ pomocou rovnice 2, kde $v = 2$. Bod s najmenšou očakávanou informačnou požiadavkou je vybraný ako deliaci bod. Podmnožina D_1 obsahuje vzorky, ktoré spĺňajú $A \leq deliacibod$, D_2 zase spĺňa $A > deliacibod$.

Gain Ratio

Informačný zisk je je miera, ktorá je vychýlená k testom s viacerými výstupmi. To znamená, že preferuje výber atribútu s veľkým počtom hodnôt. Napríklad si predstavme atribút, ktorý je jedinečný identifikátor. Výsledkom delenia takýmto atribútom je veľké množstvo delení, kde každé obsahuje práve jednu vzorku. Pretože každé delenie je čisté, informačná požiadavka na klasifikáciu dát z D založená na takomto delení je $Info_{ID}(D) = 0$, kde ID je značenie pre jedinečný klasifikátor. Preto je informačný zisk pre tento atribút maximálny.

Rozšírenie, *Gain Ratio*, sa pokúša odstrániť tento nedostatok aplikáciou istej normalizácie na informačný zisk. Táto normalizácia sa nazýva *split information* a je definovaná ako:

$$SplitInfo_A(D) = -\sum_{j=1}^{v} \frac{|D_j|}{|D|} \times \log_2\left(\frac{|D_j|}{|D|}\right).$$

Hodnota *Gain Ratio* je potom definovaná:

$$GainRatio(A) = \frac{Gain(A)}{SplitInfo_A(D)}.$$

Atribút s najväčou hodnotou je potom vybraný ako deliaci atribút. Treba však poznamenať jeden nedostatok tejto miery, ktorý nastáva keď je sa hodnota *split information* blíži k nule. Preto informačný zisk testu musí byť aspoň tak veľký, aby bol väčší ako priemerný zisk zo všetkých možných testov.

Gini index

Gini index meria nečistotu dátového setu rozdeleného na podmnožiny. Je definovaný nasledovne:

$$Gini(D) = 1 - \sum_{i=1}^{m} p_i^2,$$

kde p_i je pravdepodobnosť definovaná ako vo vzorci 1. *Gini index* delí každý atribút binárne.

Predpokladajme prípad, keď je A diskrétna premenná s v odlišnými hodnotami $a_1, a_2, \ldots, a_v$. Na určenie najlepšieho binárneho delenia skúmame všetky možné kombinácie podmnožín, ktoré môžu byť formované z hodnôt nadobúdaných A. Každá podmnožina S_a môže byť uvažovaná ako binárny test vo forme $A \in S_a$?. Ak A nadobúda v možných hodnôt, potom existuje 2^v možných podmnožín. Ak vylúčime možnosť prázdnej a celej podmnožiny dostaneme $2^v - 2$ možných delení do dvoch podmnožín. Uvažujeme teda binárne delenie, pre ktoré chceme vyrátať váženú sumu nečistoty každého výsledného delenia. *Gini index* pre delenie D na D_1 a D_2 podľa A získame:

$$Gini_A(D) = \frac{|D_1|}{|D|} Gini(D_1) + \frac{|D_2|}{|D|} Gini(D_2).$$

Podmnožina, pre ktorú je *Gini index* najmenší, je zvolená ako deliaca podmnožina.

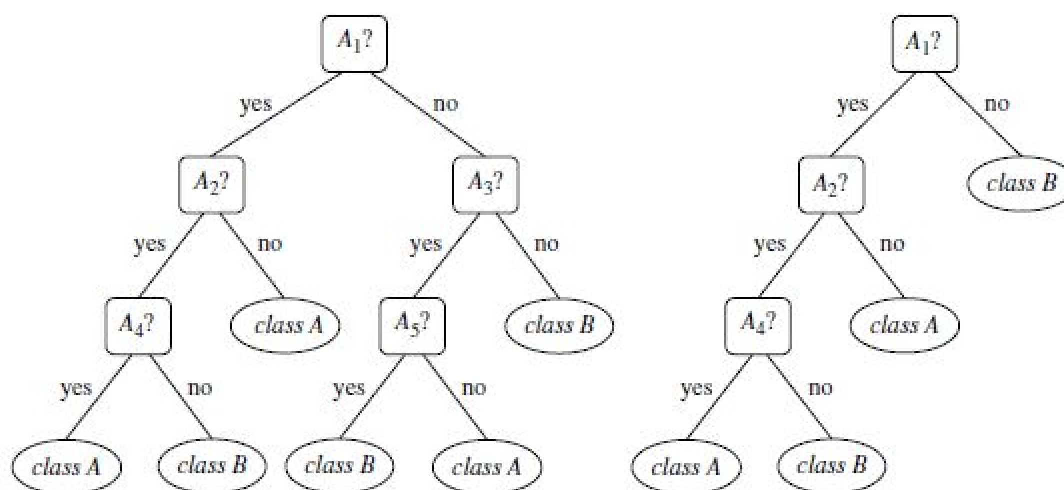
Pre spojité premenné musí byť braný do úvahy každý deliaci bod. Postup je potom podobný, ako sme ho opísali pri informačnom zisku. Za deliaci bod je potom vybraná hodnota, ktorá má najmenší *Gini index*.

Deliacim atribútom A potom bude ten, ktorý má najväčšiu redukciu nečistoty (ekvivalentné najmenšej hodnote *Gini index*):

$$\Delta Gini(A) = Gini(D) - Gini_A(D).$$

2.1.3 Orezávanie stromu

Po dobudovaní rozhodovacieho stromu veľa vetiev odráža anomálie v trénovacom dátovom sete, ktoré sú spôsobené outliermi. Tie môžu vytvoriť vetvy, ktoré obsahujú iba jednu vzorku. Metódy orezávania stromu tento problém odstraňujú. Takéto metódy zvyčajne používajú štatistické metódy na odstránenie najmenej spoľahlivých vetiev. Na obrázku 4 môžeme porovnať orezaný a neorezaný strom.



Obr. 4: Porovnanie orezaného a neorezaného stromu

Orezaný strom je menší a menej komplexnejší, ale ľahšie pochopiteľný. Sú tiež rýchlejšie a správnejšie klasifikujú nezávislé dáta, na ktorých sa strom natrénoval. Existujú dva bežné prístupy k orezávaniu stromov:

- **Redukcia nákladovej zložitosti**, pri ktorom je strom orezávaný zastavením jeho konštrukcie tým, že sa rozhoduje pre vetvenie alebo vytvorenie výsledného listu. Nákladová zložitosť je funkcia závislá od počtu listov a chybovosti (percento zle zaradených vzoriek). Orezávanie začína od dolnej časti. Pre každý neterminálny uzol vypočítame nákladovú zložitosť podstromu a nákladovú zložitosť ak by

bol orezaný. Tieto hodnoty porovnáme a vyberieme možnosť, pre ktorú je táto hodnota najmenšia. Na určenie tejto hodnoty sa používa orezávacia množina, ktorá je nezávislá od trénovacej množiny vzoriek. Algoritmus postupne generuje množinu orezaných stromov. Najmenší strom minimalizujúci nákladovú zložitosť je preferovaný ako výsledný.

- **Pesimistické orezávanie**, pri ktorom sa odstraňujú podstromy z výsledného stromu. Podstrom je potom nahradený listom. Táto metóda tiež využíva chybovosť, ale na rozdiel od prvej metódy, získanú z trénovacej sady. Preto je odhad presnosti a chybovosti silne vychýlený. Metóda ich preto upravuje pridaním penalizačného členu, ktorý zrátava odchýlky.

Druhá spomínaná metóda vyžaduje viac výpočtov, ale vo všeobecnosti poskytuje viac spoľahlivejšie výsledky. Doposiaľ nebola nájdená žiadna najlepšia metóda. V praxi sa preto využívajú aj ich kombinácie.

2.2 Klasifikačné lesy

V tejto časti predstavíme niekoľko metód na zlepšenie presnosti modelov. Hlavnou myšlienkou je namiesto jedného stromu vytvoriť súbor stromov. Preto sa tieto metódy nazývajú súborové. Predstavíme si nasledovné tri metódy:

- Bagging
- Boosting a arcing
- Náhodný les

Metódy kombinujú k získaných modelov nazývaných klasifikátory, $M_1, M_2, \ldots, M_k$, ktorých cieľom je vytvoriť lepší klasifikačný model M^*. Z množiny dát D vytvoríme k trénovacích podmnožín $D_1, D_2, \ldots, D_k$, použitých na generovanie klasifikátora M_i. Na predikciu novej vzorky sa použijú všetky modely M_i. Predikciou bude trieda, ktorá bola najpočetnejším výstupom z daných modelov. Kombinácia modelov týmto jednoduchým spôsobom sa nazýva väčšinové hlasovanie. Výhodou týchto modelov je väčšia presnosť a výpočet dobre paralelizovateľný na viac jadier procesora.

2.2.1 Bagging

Nech množina dát D obsahuje d pozorovaní, potom pre každú iteráciu $i, i = 1, 2, \ldots, k$ vytvoríme novú trénovaciu množinu D_i d pozorovaní, získanú náhodným výberom s opakovaním. Preto je Bagging skratka pre "bootstrap aggregating". Keďže každá trénovacia množina je získaná metódou bootstrap, niektoré vzorky z dátového setu D sa v nich nemusia nachádzať vôbec alebo sa môžu objaviť aj viac ako raz. Pre každú podmnožinu je potom vytvorený klasifikačný model M_i. Na predikciu sa potom aplikuje väčšinové hlasovanie.

2.2.2 Boosting a arcing

Táto metóda je pôvodne z teórie strojového učenia a v analýze dát sa označuje ako algoritmus *AdaBoost*. Predpokladajme trénovacie dáta $(x_1, y_1), \ldots, (x_n, y_n)$ a vektor $w = (w_1 \ldots, w_n)$, ktorý predstavuje váhy jednotlivých pozorovaní. V *AdaBoost* sa jednotlivé klasifikátory učia postupne, tak že sa podľa predchádzajúcich výsledkov upravujú váhy jednotlivých pozorovaní. Najprv sa použijú váhy vektora w_1, zadané užívateľom, a vytvorí sa model M_1. Pri konštrukcii ďalších modelov $M_i, i \geqq 2$ sa použije váhový vektor w_i získaný úpravou vektora w_{i-1} tak, že sa chybne klasifikovaným pozorovaniam modelom M_{i-1} priradí väčšia váha a správne klasifikovaným nižšia váha. Klasifikačná metóda sa tak stále viac sústreďuje na ťažko klasifikovateľné pozorovania, ktoré sa modelu nedarí správne klasifikovať.

Arcing predstavuje myšlienku spojenia baggingu a boostingu. Váhy sa postupne upravujú rovnakým spôsobom ako v *AdaBoost*, ale používajú sa inak. Namiesto toho, aby s týmito váhami vstupovali do analýzy všetky pozorovania, postupujeme ako v baggingu a vytvárame výbery s opakovaním, kde váhy slúžia ako pravdepodobnosti vytiahnutia.

Tieto dve metódy väčšinou znižujú chybu viac ako bagging. Napriek tomu existujú aj prípady, keď fungujú zle. Napríklad ak dáta obsahujú veľa outlierov, využitie týchto metód vedie k učeniu opakovaných chýb v dátach .

2.2.3 Náhodné lesy

V tomto prípade je každý klasifikátor rozhodovací strom, ktorého konštrukciu sme predstavili v časti 2.1. Spolu teda vytvárajú les. Táto metóda je založená na nasledovných postupoch:

- trénovacie súbory pre jednotlivé stromy sú bootstrapové výbery z množiny dát D
- pri voľbe vetvenia pre daný uzol sa z m prediktorov, ktoré sú k dispozícii, vyberie m_0, na ktorých sa potom klasicky vytvára najlepšie vetvenie, ale len z tých m_0 prediktorov, ktoré boli pretým vybrané
- stromy sa neorezávajú, zostávajú veľmi rozvetvené

Náhodný výber prediktorov výrazne skracuje výpočtový čas a výsledky klasifikačných úloh sú veľmi dobré. Zvlášť prínosná je táto metóda pre veľký počet prediktorov, z ktorých každý sám o sebe obsahuje len málo informácie o závislej premennej.

Voľba parametra m_0 závisí na úlohe a otvorenosti experimentovať. Odporúčané je použiť hodnotu blízko $log_2 m$.

Medzi baggingom, boostingom, arcingom a na druhej strane náhodnými lesmi je podstatný rozdiel. Prvé tri spomenuté sú nadstavbou klasickej metódy a snažia sa o čo najlepší a najpresnejší strom. V metóde náhodné lesy nie je podstatná kvalita jednotlivých stromov, ale hlavným cieľom je minimalizovať chybu celého lesa.

2.3 Logistická regresia

Druhou metódou, ktorú si predstavíme je logistická regresia. Budeme pritom vychádzať z článku [8]. Táto metóda sa zvyčajne používa na binárnu klasifikáciu tried, v ktorej vektor y môže obsahovať iba dve hodnoty: $C_1 = 0$ ak jav nenastal, $C_2 = 1$ ak jav nastal. Metóda meria vzťah medzi závislými premennými x a jednou alebo viac nezávislými premennými y odhadom pravdepodobnosti pomocou logistickej funkcie. Logistická regresia môže byť chápaná ako špeciálny prípad zovšeobecneného lineárneho modelu, a je teda analogická s lineárnou regresiou. Je však založená na odlišných predpokladoch o vzťahoch medzi závislými a nezávislými premennými. Hlavné rozdiely týchto dvoch modelov sú v nasledovných dvoch vlastnostiach logistickej regresie:

- Podmienené rozdelenie $y|x$ je Bernoulliho rozdelenie, pretože závislá premenná je binárna.
- Predikované hodnoty sú pravdepodobnosti, preto sú v rozmedzí $[0, 1]$. Tie získame z logistickej funkcie.

Model logistickej regresie je nasledovný:

$$log \frac{h(x)}{1 - h(x)} = \beta^\top x = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \ldots + \beta_p x_p, \tag{3}$$

kde x je vektor p nezávislých premenných a β vektor odhadovaných parametrov modelu. Potom pre $h(x)$ odvodením z 3 platí:

$$h(x) = \frac{1}{1 + \exp(-\beta^\top x)}. \tag{4}$$

2.3.1 Algoritmus prostredníctvom Newtonovej metódy

Keďže táto metóda predikuje pravdepodobnosti a nie len triedy, môžeme pre ňu vytvoriť vierohodnostnú funkciu. Pre každú vzorku máme vektor vlastností x a triedu pozorovania y. Nech:

$$P(y = 1|x, \beta) = h(x)$$
$$P(y = 0|x, \beta) = 1 - h(x).$$

Tieto rovnosti môžeme kompaktne zapísať nasledovne:

$$P(y|x, \beta) = h(x)^y \times (1 - h(x))^{1-y} \tag{5}$$

Potom vierohodnostná funkcia ma tvar:

$$L(\beta) = \prod_{i=1}^{n} h(x_i)^{y_i} \times (1 - h(x_i))^{1-y_i}, \tag{6}$$

logaritmovaním dostaneme:

$$\begin{aligned} l(\beta) &= \sum_{i=1}^{n} y_i \log h(x_i) + (1-y_i)\log(1-h(x_i)) \\ &= \sum_{i=1}^{n} \log(1-h(x_i)) + y_i \log \frac{h(x_i)}{1-h(x_i)} \\ &= \sum_{i=1}^{n} \log(1-h(x_i)) + y_i \beta^\top x_i \\ &= \sum_{i=1}^{n} \log \frac{1}{1+exp(\beta^\top x_i)} + y_i \beta^\top x_i \\ &= \sum_{i=1}^{n} -\log(1+exp(\beta^\top x_i)) + y_i \beta^\top x_i \end{aligned}$$

Ďalším krokom na nájdenie odhadu parametra metódou maximálnej vierohodnosti je derivovanie podľa tohto parametra a položenie derivácie rovnosti nule. Derivujme podľa parametra β_j, $j = 1, \ldots, p$:

$$\begin{aligned} \frac{\partial l}{\partial \beta_j} &= \sum_{i=1}^{n} -\frac{\partial}{\partial \beta_j} \log(1+exp(\beta^\top x_i)) + y_i \frac{\partial}{\partial \beta_j}(\beta^\top x_i) \\ &= \sum_{i=1}^{n} -\frac{1}{1+\exp(\beta^\top x_i)} \beta^\top x_i \frac{\partial}{\partial \beta_j}(\beta^\top x_i) + y_i \frac{\partial}{\partial \beta_j}(\beta^\top x_i) \\ &= \sum_{i=1}^{n} (y_i - h(x_i)) x_{i,j} \end{aligned} \tag{7}$$

Položením rovnosti nule nie sme schopný riešiť túto úlohu. Úlohu preto budeme riešiť numerickou optimalizáciou prostredníctvom Newtonovej metódy.

Predpokladajme, že $f(x)$ je hladká funkcia a chceme nájsť globálne minimum x^*. Nech x_0 je štartovací bod. Potom pomocou Newtonovej metódy aproximujeme minimum iteračným procesom:

$$x_{n+1} = x_n - \frac{f'(x_n)}{f''(x_n)}, \tag{8}$$

kde $f'(x)$ je gradient funkcie $f(x)$ a $f''(x)$ je Hessova matica druhých parciálnych derivácií. Na riešenie preto potrebujeme Hessovu maticu, ktorú získame nasledovne pre $j, k = 1, \ldots, p$:

$$\begin{aligned} \frac{\partial^2 l}{\partial \beta_j \partial \beta_k} &= \sum_{i=1}^{n} \Big(y_i - \frac{\partial}{\partial \beta_k} h(x_i)\Big) x_{i,j} \\ &= \sum_{i=1}^{n} -x_{i,j} \Big(\frac{1}{1+\exp(\beta^\top x_i)}\Big)^2 \exp(\beta^\top x_i) \frac{\partial}{\partial \beta_k}(\beta^\top x_i) \\ &= \sum_{i=1}^{n} -x_{i,j} h(x_i)(1-h(x_i)) x_{i,k} \end{aligned} \tag{9}$$

Zo získaných výsledkov vieme iteračným procesom aporoximovať odhady parametra β.

2.4 K najbližších susedov

Idea uvedená v knihe [2] pre algoritmus k najbližších susedov spočíva vo vybudovaní klasifikačnej metódy bez použitia predpokladov o funkcii $y = f(x_1, x_2, \ldots, x_p)$. Tá zodpovedá závislej premennej y od nezávislých premenných $x_1, x_2, \ldots, x_p$ nazývaných aj prediktory. Jediným predpokladom je, že funkcia f je hladká. Táto metóda je neparametrická, pretože jej výstupom nie je odhad parametrov ako napríklad v logistickej regresii.

Pri tejto metóde pracujeme s trénovacou množinou dát, v ktorej každé pozorovanie obsahuje zaradenie do triedy y. My budeme ako aj v predchádzajúcich metódach uvažovať iba binárnu klasifikáciu. Idea spočíva v identifikácii k pozorovaní z trénovacej množiny, ktoré sú podobné novému pozorovaniu $u_1, u_2, \ldots, u_p$, ktoré chceme klasifikovať. Odhad klasifikácie označme $\hat{v}$. Ak poznáme funkciu f, môžeme jednoducho tento odhad vypočítať ako $\hat{v} = f(u_1, u_2, \ldots, u_p)$. Ak teda predpokladáme, že f je hladká, rozumným postupom je hľadať pozorovania, ktoré sú najbližšie k novému pozorovaniu a z ich tried odhadnúť $\hat{v}$. Pojem susedia v názve naznačuje, že existuje nejaká miera vzdialenosti alebo rozdielnosti, ktorú môžeme na základe nezávislosti medzi pozorovaniami vypočítať. My sme sa pre jednoduchosť zamerali na najčastejšie používanú mieru: Euklidovskú. Euklidovská miera medzi bodmi $x_1, x_2, \ldots, x_p$ a $u_1, u_2, \ldots, u_p$ je

$$\sqrt{(x_1 - u_1)^2 + (x_2 - u_2)^2 + \ldots + (x_p - u_p)^2}.$$

Najjednoduchším prípadom je voľba $k = 1$, kde hľadáme iba najbližšie pozorovanie, podľa ktorého klasifikácie je odhadnuté $\hat{v}$. Voľba jedného najbližšieho suseda môže byť silná ak máme k dispozícii veľký počet dát na trénovanie.

Pre k susedov je idea rozšírená rovnako ako pre jedného suseda. V tomto prípade hľadáme k najbližších pozorovaní a väčšinovým hlasovaním určíme triedu nového pozorovania. Výhodou väčšej hodnoty k je odstránenie rizika pretrénovania dôsledkom outlierov v trénovacej vzorke. Parameter k zvyčajne v praxi nadobúda hodnoty jednotiek a desiatok. Ak by sa $k = n$, čiže počtu pozorovaní v trénovacej vzorke, tak predikovaná trieda by nadobudla hodnotu početnejšej skupiny bez ohľadu na vstupy $u_1, u_2, \ldots, u_p$ nového pozorovania.

3 Integrácia R do SQL

Možností na prepojenie softvéra R a jazyka SQL je viacero. Hoci integráciu R do SQL Serveru priniesla až verzia z augusta 2016, boli tu ešte možnosti pripojenia sa z R do databázy pomocou knižnice RODBC. Tento prístup má však viacero nevýhod:

- prenos dát môže byť pomalý, neefektívny, prípadne aj nebezpečný
- obmedzenia R na výkon a pamäť

Tieto nedostatky sa prejavujú hlavne pri prenášaní a analyzovaní veľkých dátových setov. Ďalšou možnosťou bolo využívanie R skriptov vo Visual Studiu pomocou doplnku *R tools for Visual Studio*. Takéto prepojenie samozrejme ponúka aj Oracle, najväčší konkurent firmy Microsoft. My vzhľadom na dostupnosť budeme využívať doplnok SQL Server R Services, ktorý ponúka dva prístupy tejto integrácie:

- externý R skript v Transact-SQL
- knižnica RevoScaleR v softvéri R

3.1 Využitie integrácií a popis postupov data scientistov

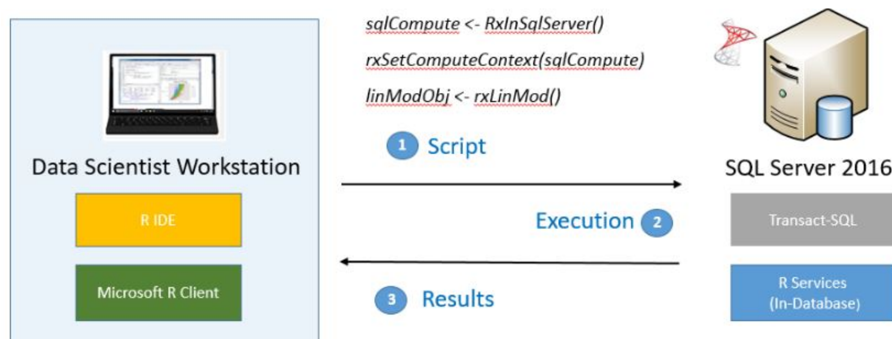
Štandardný pracovný postup pre budovanie vyspelých analytických riešení začína s prieskumom dát a pokračuje predikčným modelovaním. Data scientist vyvíja R skripty a modely tak, aby boli účinné a vhodné pre danú úlohu. Potom, keď sú skripty a modely pripravené, môžu byť nasadené a integrované v existujúcich alebo nových aplikáciach.

SQL Server R Services je navrhnutý tak, aby pomohol riešiť úlohy data scientistov. Prináša spôsob, ako pracovať súčasne s jazykom R a SQL nástrojmi, medzi ktorými vytvára prepojenie. Táto integrácia umožňuje využívať vlastné R kódy v procesoch bez toho, aby sme ich museli prepísať do iného jazyka. Tiež umožňuje použitie štatistických metód prístupných v známych balíkoch bez zbytočného prenášania dát a načítavania databáz do operačnej pamäte. Výhodou je tiež využitie sily SQL Serveru pre dosiahnutie maximálneho výkonu. Toto všetko zabezpečuje R Services, ktoré sa inštaluje počas inštalácie funkcií a doplnkov do SQL Serveru. Ten následne umožňuje vykonávať externé R skripty na SQL Serveri pomocou *SQL Server Trusted Launchpad*, ktorý riadi komunikáciu medzi jazykmi R a SQL.

Nasledujúce delenie poskytuje pohľad na typický analytický pracovný postup pri využití SQL Server R Services.

- Prvou časťou je **vývoj**, kde data scientisti používajú R na spracovanie údajov a budovanie predikčných modelov. Testovanie a ladenie modelu opakuje, kým nedostane model zodpovedajúci najmenšej chybovosti. R Services komponenty nám poskytujú všetky nástroje potrebné na experimentovanie a vývoj. Tie zahŕňajú knižnicu zlepšujúcu výkon štandardných operácií a sadu rozšírených balíčkov, ktoré podporujú vykonávanie R kódu v SQL Serveri.

- Ďalšou časťou je **optimalizácia** procesu. Pri analýze veľkých súborov dát v R často narazíme na problémy s výkonom a rozsahom pamäte, pretože bežná realizácia je *single-threaded*[1], ktorá môže načítať iba tie dátové súbory, ktoré sa zmestia do voľnej pamäte na lokálnom počítači. Ak chceme získať lepší výkon a pracovať s viacerými dátami, môžeme používať ScaleR API [2], ktoré je k dispozícii ako súčasť R Services. Tento doplnok umožňuje vykonať výpočty na počítači, kde je umiestnený server. Zadefinovaním výpočtového kontextu na serveri predchádzame premiestňovaniu dát, pretože výpočty prebiehajú priamo na ňom. Tento postup je znázornený na obrázku 5.

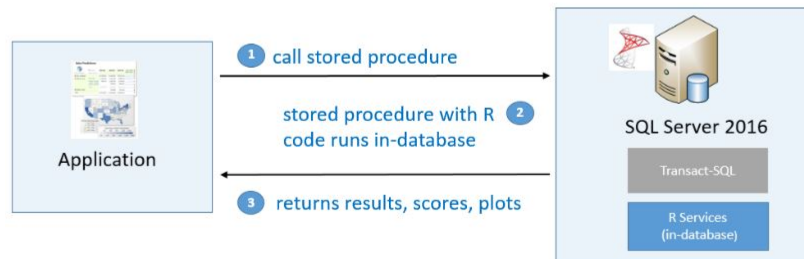


Obr. 5: Zobrazenie pripojenia a pracovania s použítím servera

[1]hovorovo jedno vláknitý, použitie threadov je spôsob, ako rozdeliť program na jedno alebo viac rovnocenne vykonávaných úloh

[2]skratka pre Application Programming Interface, v informatike označuje rozhranie pre programovanie aplikácií

- **Aplikácia** získaných modelov. Ak je R skript alebo model pripravený na použitie, databázový vývojár ho môže vložiť do uložených alebo nových procedúr. Môžeme teda vybrať konkrétne dáta pomocou jazyku Transact-SQL, ktoré spracujeme týmito procedúrami zapísanými v jazyku R. Pre dosiahnutie najlepšieho výkonu je potrebné opäť spúšťať tieto procedúry na serveri, ako to môžeme vidieť na obrázku 6.



Obr. 6: Spúštanie uložených procedúr na serveri

- Posledným krokom je **kontola** a rekalibrácia získaných parametrov do modelov uložených v procedúrach.

3.1.1 Externý R skript v Transact-SQL

V tejto časti si v krátkosti ukážeme syntax R v SQL Server R Services na základe návodov uvedených na stránke [4]. Prácu s jazykom R v prostredí SQL Serveru zabezpečuje uložená procedúra `sp_execute_external_script`. Takýto externý skript má nasledovný syntax:

```
sp_execute_external_script
@language = N'language' ,
@script = N'script',
@input_data_1 = ] 'input_data_1'
[ , @input_data_1_name = N 'input_data_1_name' ]
[ , @output_data_1_name = 'output_data_1_name' ]
[ , @parallel = 0 | 1 ]
[ , @params = ] N' @parameter_name data_type [ OUT | OUTPUT ] [ ,...n ]'
[ , @parameter1 = ]'value1' [ OUT | OUTPUT ] [ ,...n ]
```

```
[ , WITH <execute_option> ]
[;],
```

kde základnú časť tvoria:

- `@language`: definujeme jazyk, v ktorom bude písaný skript
- `@script`: definuje skript, ktorý sa má vykonať
- `@input_data_1`: definovanie dotazu k dátam, s ktorými chceme pracovať.
- `WITH RESULTS SETS`: príkaz definujúci schému výstupu

Ďalšie časti v zátvorkách [...] nemusia byť definované:

- `@input_data_1_name`: určuje názov dát. Ak táto časť nie definovaná, tak majú predefinovaný názov *InputDataSet*
- `@output_data_1_name`: určuje výstup zo skriptu. Ak táto časť nie definovaná, tak majú predefinovaný názov *OutputDataSet*
- `@parallel`: určuje paralelné vykonanie skriptu. Predefinovaná hodnota je 0, kedy neprebieha paralelné vykonanie. Táto vlastnosť je výhodná pri veľkých datasetoch.
- `@params`: deklarácia vstupných parametrov
- `@parameter1`: zoznam hodnôt pre vstupné parametre.

Na základe vyššie uvedeného popisu ukážeme pár jednoduchých aplikácií s ich výsledkami, ktoré budeme spúšťať v programe Microsoft SQL Server Management Studio.

Príklad č.1

```
execute sp_execute_external_script
@language = N'R'
, @script = N' mytextvariable <- c("Dobre rano", "Dobry den", "Dobry vecer");
OutputDataSet <- as.data.frame(mytextvariable);'
, @input_data_1 = N' SELECT 1 as Temp1'
WITH RESULT SETS (([col] char(20) NOT NULL));
```

Príklad č.2

```
execute sp_execute_external_script
@language = N'R'
, @script = N' OutputDataSet
<- data.frame(c("Dobre rano"),c("Dobry den"),c("Dobry vecer"));'
, @input_data_1 = N' '
WITH RESULT SETS (([col1] varchar(20), [col2] char(20), [col3] varchar(20)));
```

Na obrázkoch 7 a 8 môžeme vidieť výsledky týchto dvoch príkladov. Vidíme, že v oboch prípadoch sme si zadefinovali jazyk R. Vstup v prvom príklade je hodnota 1, v druhom je to prázdne pole. V skripte nám vstupy do príkazov nezasahujú, preto sme ich nemuseli ani v jednom prípade zadať. Výstupom pre oba príklady je vektor, ktorý musíme nahrať do preddefinovanej formuly *OutputDataSet*. Výstupy sú podobné a líšia sa len v tom, že druhý je transponovaný. Typ výstupu musíme vždy zadefinovať v klauzule `WITH RESULT SETS`. R a SQL využívajú rôzne dátové typy, preto musíme výstup vždy previesť na typ dátový rámec.

	col
1	Dobre rano
2	Dobry den
3	Dobry vecer

Obr. 7: Výstup z príkladu č. 1

	col1	col2	col3
1	Dobre rano	Dobry den	Dobry vecer

Obr. 8: Výstup z príkladu č. 2

3.1.2 Prostredie R použitím knižnice RevoScaleR

Praktický úvod práce s balíkom RevoScaleR je obsiahnutý v knihe [3]. RevoScaleR obsahuje implementáciu viacerých známych R funkcií, ktoré boli upravené tak, aby sa vykonávali paralelne a neboli obmedzené rozsahom pamäte. Najprv si ukážeme, ako získame prístup k dátam z SQL Serveru do prostredia R. Postup je nasledovný:

- vytvorenie pripojenia k SQL Serveru
- definovanie dotazu, ktorý vyselektuje potrebné dáta
- definovanie jedeného alebo viacerých výpočtových kontextov

Na získanie prístupu k databáze budeme potrebovať základné informácie o pripojení. Na základe toho si zadefinujeme reťazec *connectionString* s týmito údajmi.

```
connectionString<-"Driver; Server; Database; Uid; Pwd ".
```

Musíme tiež zadefinovať dáta, ktoré chceme použiť, prostredníctvom tabuľky alebo dotazu. Následne si môžeme vytvoriť pripojenie k dátam pomocou príkazu:

```
inDataSource<-RxSqlServerData(sqlQuery, connectionString, rowsPerRead).
```

Do premennej *inDataSource* si zadefinujeme dátový zdroj v SQL Serveri. Dáta špecifikujeme konkrétnym dotazom zadaným do parametra `sqlQuery`. Dôležitým pre manipuláciu s pamäťou a efektívnosť výpočtov je `rowsPerRead`. Tento parameter určuje, koľko riadkov z dát je načítaných v jednom kroku medzivýpočtov. Ak je jeho hodnota príliš veľká, môže spôsobiť spomalenie kvôli nedostatku operačnej pamäte.

K získaniu základných informácií o dátovom sete môžeme použiť príkaz, ktorého výstupom je tabuľka s názvom a typom premenných.

```
rxGetVarInfo(data = inDataSource)
```

Ak chceme aby výpočty prebiehali v mieste uloženia databázy, musíme vytvoriť výpočtový kontext pomocou príkazu `RxInSqlServer`. Tento príkaz vyžaduje informácie o zdieľanom adresári, konkrétne cestu k nemu. My sme si vytvorili lokálny výpočtový kontext s názvom *ShareDir*. Musíme tiež zadefinovať ako chceme manipulovať s výpočtom, a či chceme výsledky medzivýpočtov. Tieto vlastnosti nastavíme `sqlWait`$< -$`TRUE`, `sqlConsoleOutput`$< -$`FALSE`. Následne s týmito parametrami môžeme definovať výpočtový kontext.

```
RxInSqlServer(connectionString,shareDir, sqlWait, sqlConsoleOutput)
```

S takto zadefinovaným zdrojom už môžeme vykonávať rôzne príkazy, ktoré poznáme.

Balík *RevoScaleR* ponúka rôzne známe štatistické funkcie. Všetky funkcie z tohto balíka obsahujú kvôli lepšej identifikácii predponu *rx* alebo *Rx*. Teraz uvedieme pár príkladov z tohto balíka, z ktorých niektoré využijeme v ďalších častiach:

- `rxLinMod`: fituje lineárny model
- `rxLogit`: fituje logistickú regresiu na dáta
- `rxGlm`: fituje zovšeobecnený lineárny model
- `rxDTree`: vytvára klasifikačné stromy
- `rxDForest`: vytvára náhodný les
- `rxKmeans`: vykonáva k-means klastrovaciu metódu
- `rxNaiveBayes`: vykonáva Naive Bayse klasifikáciu.

Kompletný zoznam aj s opisom je k dispozícií na stránke [5].

4 Návody na zrýchlenie výpočtového času v R

Predtým, ako začneme porovnávať jednotlivé prístupy si ukážeme, ako zrýchliť výpočtový čas kódov v softvéri R. Na základe článku [7] uvedieme jednoduché príklady na zrýchlenie výpočtov, ale aj paralelné vykonanie opakujúcich sa cyklov. Jedným z riešení, síce dosť naivným, môže byť kúpa nového počítača s lepším hardvérovým vybavením. Nasledovné tri hardvérové časti počítača ovplyvňujú rýchlosť vykonania výpočtov:

- pevný disk (HDD)
- pamäť s voľným (náhodným, ľubovoľným) prístupom (RAM)
- procesor (CPU)

HDD

Pevný disk je miesto v počítači, kde sú uložené všetky dáta aj keď je počítač vypnutý. Sú tu uložené aj veľké dátové sety, ktoré chceme analyzovať pomocou R. Tým, že sú veľmi lacné, sú najlepšou možnosťou na ukladanie veľkého množstva dát. Nevýhodou pri čítaní a zapisovaní veľkých dát je to, že tento proces je príliš pomalý. Pre eliminovanie tohto problému je ideálne načítavať dáta len raz na začiatku kódu a zapisovať raz na konci. Opakované načítavanie a zapisovanie v tele kódu môže byť časovo náročné.

RAM

Pamäť s voľným prístupom je tiež používaná na ukladanie súborov. Pri načítaní dát uložených na pevnom disku sú dáta skopírované do RAM. Táto pamäť je rýchlejšia ako HDD a umožňuje prístup k dátam v reálnom čase. Tieto dáta sú však po vypnutí počítača stratené. Druhým problémom je, že táto technológia je príliš drahá. Ďalším problémom pre R, rovnako ako aj pre iné programy, je situácia, keď veľkosť dát presiahne presiahne kapacitu tejto pamäte. Načítanie dát veľkosti 10 GB z pevného disku do RAM s rozsahom 8 GB spôsobí spomalenie kódu. V tomto prípade počítač nemá inú možnosť a musí využiť časť pevného disku.

CPU

Procesor je základ každého počítača, kde sa vykonávajú všetky výpočty. V dnešnej

dobe majú procesory viac jadier. R však nie je schopný využiť procesor naplno. Napríklad ak máme procesor s 8 jadrami a chceme vykonať nejaký výpočet, jeho vykonanie nie je rozdelené po 1/8 medzi všetky jadrá. Všetky výpočty sú vykonávané iba na jednom jadre v rovnakom čase . To znamená, že využívame iba 12,5% z celkovej dostupnej výpočtovej sily. Ostatné jadrá zostávajú nevyužité.

4.1 Jedno R prostredie, jedno jadro

Ako bolo spomenuté vyššie, R používa iba jedno jadro aj keď je ich dostupných viac. V tejto časti si ukážeme pár trikov, ako aj napriek tejto skutočnosti zrýchliť kód. Najprv si ukážeme, ako budeme merať čas jednotlivých operácií:

```
zaciatok<-Sys.time()
## telo kódu ##
vysledny.cas<-Sys.time()-zaciatok
```

Pomocou tohto kódu získame čas vykonanej operácie v sekundách na 8 desatinných miest. Dôležité je však spomenúť, že výpočtový čas je vo všeobecnosti náhodný. Ak spustíme rovnaký kód niekoľko krát, môžeme očakávať odlišné dĺžky výpočtového času. Keď porovnávame dva odlišné kódy alebo funkcie, ktoré dávajú ten istý výsledok, za účelom vybrať tú rýchlejšiu, mali by sme merať výpočtový čas prvej funkcie vykonaním n_1 meraní a podobne pre druhej n_2. Na týchto dátach by sme potom mali vykonať t-test prípadne neparametrický test, ak nie je splnený predpoklad normality. Našťastie jedno meranie výpočtového času funkcií zvyčajne stačí na určenie rýchlejšej funkcie bez použitia nejakého štatistického nástroja.

Pri meraní rýchlosti nášho kódu je veľmi dôležité poznať, ako sa využíva pamäť. Každý objekt vytvorený v R je uložený v RAM. Napríklad generovanie 10^7 pozorovaní z normálneho rozdelenia vyžaduje 80MB kapacity. Pri práci s pamäťou je dôležité poznať nasledovné príkazy, ktoré môžu prečistiť pamäť a tým zrýchliť výpočet:

- `object.size()`: príkaz slúžiaci na získanie veľkosti objektu
- `ls()`: slúži na zobrazenie všetkých vytvorených objektov v prostredí R

- `rm()`: ak už niektorý z objektov v R nepotrebujeme, môžeme ho vyhodiť z RAM týmto príkazom

Iný spôsob odstránenie objektov z RAM je zatvorenie prostredia R. Tým sa stratí všetko, čo sme počas práce načítali do pamäte.

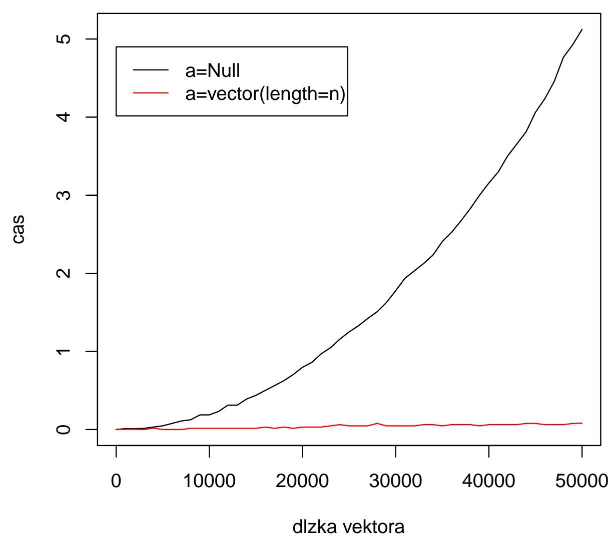
Typy na zrýchlenie

Ako prvé porovnáme dva odlišné *for* cykly, ktorých výstupom sú čísla $1^2, 2^2, \ldots, n^2$. V prvom postupe nebude vopred zadefinovaná dĺžka vektora, čiže vektor bude krok po kroku narastať.

```
a=NULL
for(i in 1:n) a=c(a,i^2)
```

Druhým spôsobom je zadefinovanie konečnej veľkosti vektora pred cyklom.

```
a=vector(length=n)
for(i in 1:n) a[i]=i^2
```



Obr. 9: Porovnanie výpočtového času funkcie $f(x) = x^2$ pre vopred zadefinovaný vektor a vektor postupne narastajúci

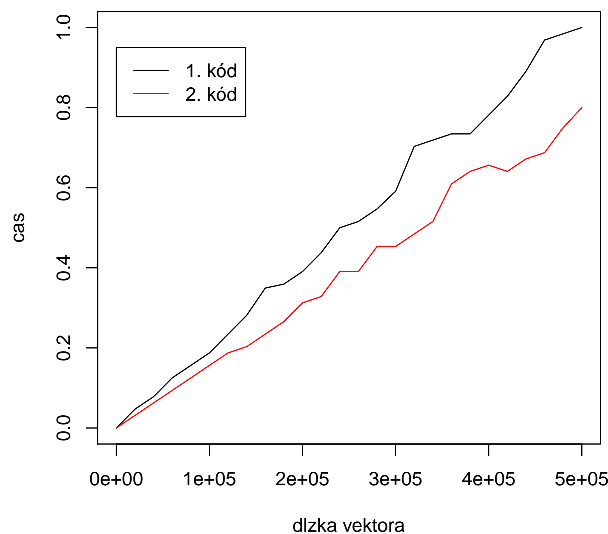
Na obrázku 9 môžeme pozorovať výšku výpočtového času pre vopred zadefinovaný vektor, zobrazený červenou krivkou, a postupne narastajúci vektor znázornený čiernou. Z grafu je zrejmé, že vopred zadefinovaný vektor znižuje dobu výpočtu.

Ďalšou častou chybou užívateľov je vykonávanie rovnakých výpočtov v cykle. Preto druhým zlým príkladom je výpočet funkcie $f(x) = 2\pi sin(x)$ nasledovným spôsobom:

```
a=vector(length=n)
for(i in 1:n) a[i]=2*pi*sin(i)
```

Opravou tohto príkazu je vykonanie nasledujúceho kódu, kde sa výpočet 2π nevykonáva zbytočne v každej iterácii cyklu, ale až po jeho skončení.

```
a=vector(length=n)
for(i in 1:n) a[i]=sin(i)
a=2*pi*a
```



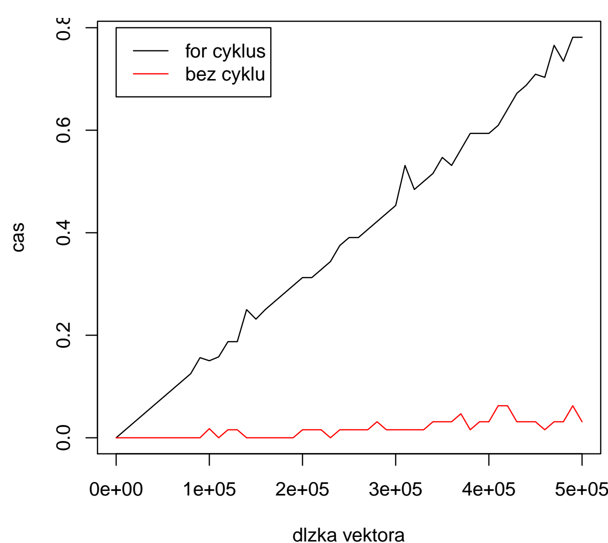
Obr. 10: Porovnanie výpočtového času funkcie $f(x) = 2\pi sin(x)$ pre výpočet 2π vo vnútri a po vykonaní cyklu

Na obrázku 10 môžeme pozorovať výšku výpočtového času pre vyššie spomínané 2 kódy. Prvý kód je znázornený čiernou farbou, druhý červenou. Výsledkom je opäť znázornenie zrýchlenia výpočtu jednoduchou úpravou tela programu.

Vo všeobecnosti R nie je stavaný na cykly. Preto ich používame iba v nevyhnutnom prípade. Na nasledujúcom príklade ukážeme, aký veľký rozdiel je pri využití alternatívnych postupov bez cyklu. Uvažujme opäť funkciu $f(x) = 2\pi sin(x)$, ktorú sme už použili predtým. Teraz ju porovnáme s jej ekvivalentom napísaným bez cyklu:

```
a=2*pi*sin(1:n)
```

Oba kódy nám dávajú rovnaké výsledky pre rovnaké vstupné hodnoty. Čierna krivka na grafe zobrazuje riešenie prostredníctvom cyklu, červená zase bez neho.



Obr. 11: Porovnanie výpočtového času funkcie $f(x) = 2\pi sin(x)$ pre výpočet pomocou cyklu a bez neho

Vidíme, že druhý postup je omnoho rýchlejší a tiež jednoduchší. Z týchto výsledkov môžeme usúdiť, že R pracuje lepšie s vektormi, maticami a tabuľkami.

Ďalším spôsobom ako zrýchliť výpočet je prepísanie ho ako funkciu jazyka C. Ten je hlavne pri cykloch rýchlejší ako R. Postup ako napísať a vytvoriť C funkciu v R je spomenutý na stránke [6].

4.2 Paralelné programovanie v R

4.2.1 Jeden program pre každé jadro

V dnešnej dobe majú počítače aspoň 2 jadrá, zvyčajne ale disponujú 4 alebo 8. Veľmi efektívna možnosť na využitie ďalších jadier je súbežné spustenie viacerých prostredí R. Operačný systém priradí iné jadro každému novému R prostrediu. Ak otvoríme viac R programov ako je jadier, niektoré prostredia sa budú byť o rovnaké jadro, čo spomalí výpočty. Musíme mať na mysli aj to, že na počítači bežia aj iné procesy, ktoré tiež využívajú určité percento procesora. Ak máme napríklad otvorený webový prehliadač a zároveň počet R prostredí rovný počtu jadier, aspoň jedno R začne súperiť s prehliadačom o to isté jadro, čo môže spomaliť prehliadač a aj výpočty vykonávané v danom prostredí.

Ak je počet dostupných jadier na počítači n_c, dobrým pravidlom je vždy nechať jedno jadro voľné a otvoriť maximálne $n_c - 1$ prostredí. Tým sa vyhneme interferenciám medzi R procesmi a ostatnými procesmi v počítači. Využitie $(n_c - 1)/n_c * 100\%$ z celkovej výpočtovej sily, ktorú poskytuje počítač, vytvorí veľa tepla a vyžaduje viac energie. Preto level batérie počítača, ktorý sa nenabíja začne rýchlo klesať.

Ak teda máme naraz otvorených niekoľko aktívnych prostredí R, môžeme vykonať cykly paralelne. Paralelne ich však môžeme vykonať iba vtedy, ak sú iterácie opakované v cykle nezávislé od predchádzajúcej a nasledujúcej iterácie. Aplikáciu ukážeme na príklade uvedenom v článku [7], ktorý v ďalšej časti rozšírime.

Odhad rozdelenia priemeru

Chceme odhadnúť rozdelenie výberového priemeru $\bar{X}_n$, kde $n = 100000$ a dáta sú i.i.d. z Poissonovho rozdelenia s parametrom λ.

Riešenie: vygenerujeme 100000 i.i.d. pozorovaní z Poissonovho rozdelenia, z ktorých vyrátame výberový priemer. Toto opakujeme napr. 90000 krát a vytvoríme histogram odhadovaného rozdelenia výberového priemeru $\bar{X}_n$. Z centrálnej limitnej vety môžeme očakávať, že rozdelenie $\bar{X}_n$ bude normálne so strednou hodnotou λ a disperziou λ/n. Na odhad použijeme nasledovný kód:

```
xbar=vector(length=90000)
n=100000
start<-Sys.time()
for(i in 1:90000) {
data=rpois(n ,lambda=1)
xbar[i]=mean(data)
}
time<-Sys.time()-start
hist(xbar)
```

Vykonanie tohto kódu trvalo 6 minút a 57 sekúnd. Keďže jednotlivé výpočty výberového priemeru sú nezávislé od predchádzajúcej a nasledujúcej operácie, môžeme tento výpočet vykonať paralelne. Znamená to, že čas približne 7 minút môžeme znížiť použitím viacerých jadier tak, že výpočet rozložíme medzi ne. Počítač, na ktorom vykonávame pozorovanie má k dispozícii 4 jadrá. Na základe pravidla spomenutého vyššie použijeme $n_c - 1 = 3$ jadier, to znamená, že si otvoríme tri R prostredia, kde dve sú chápané ako „pracovníci“ a jedno ako „šéf“. Medzi tieto prostredia rozdelíme generovanie a výpočet $\bar{X}_n$ na tri časti po 30000 opakovaní. Obidvaja pracovníci vykonajú tento cyklus a výsledky zapíšu na pevný disk. Šéf vykoná vlastné výpočty, načíta výsledky pracovníkov, spojí ich a vytvorí z nich histogram odhadovaného rozdelenia $\bar{X}_n$. Výpočet oboch pracovníkov trval približne 3 minúty a 15 sekúnd. Práca šéfa zabrala o 3 sekundy viac. Vykreslenie histogramu pri paralelnom rozdelení výpočtov bolo približne 2,2 krát rýchlejšie ako pri výpočte v jednom prostredí.

Pri tomto postupe môže nastať problém, vtedy ak prostredie označené ako šéf začne načítavať výsledky od pracovníkov skôr ako ukončia prácu. V takomto prípade R vyhodí chybovú hlášku. Tento problém sa môže vyskytnúť aj keď spustíme kód pre šéfa ako posledný. Tento problém môžeme vyriešiť nastavením statusu v pre každého pracovníka. Šéf teda najprv prečíta status a až potom načíta výsledky.

Použitie viacerých R prostredí má preto svoje výhody:

- ľahké na porozumenie
- veľmi efektívne, použitie 3 prostredí je 2,2-3 krát rýchlejšie

Na druhej strane má ale aj isté nevýhody:

- riziko chyby ľudského faktora pri veľkom počte R prostredí
- pevný disk je využívaný na výmenu dát medzi pracovníkmi a šéfom. To spôsobuje spomalenie ak je veľké množstvo dát zapisované a načítané viac krát.

4.2.2 Jeden program pre viac jadier

Lepším spôsobom ako pracovať súčasne s viacerými R prostrediami je použitie viacerých jadier s jedným prostredím. Túto možnosť umožňujú viaceré balíky v R, ako napr.: pnmath, doSMP, doMC, snow, doParallel, foreach. My v našej práci budeme používať najmä doParallel [9] a foreach [10]. Preto si v tejto časti pomocou nich predstavíme pár príkladov a testov nášho zariadenia.

Aplikácia balíkov doParallel a foreach

Balík doParallel poskytuje paralelné pozadie pre balík foreach a spolu vytvárajú mechanizmus na vykonávanie cyklov paralelne. Preto sú tieto dva balíky prepojené. Pomocou prvého balíka používateľ zadefinuje paralelné rozhranie, na ktorom sa bude realizovať výpočet. Druhý balík je potrebný na zadanie nezávislých cyklov, ktoré je možné vykonať súbežne.

Po načítaní knižníc musíme zavolať funkciu `registerDoParallel`. Ak ju zavoláme bez žiadneho vstupného argumentu, operačný systém Windows vytvorí tri R prostredia. Iný počet prostredí môžeme zadefinovať argumentom `cores` alebo funkciou `makeCluster`. Počet prostredí by však nemal byť väčší ako počet jadier procesora, preto je potrebné preveriť, koľko ich máme k dispozícii. Na to nám slúži funkcia `detectCores`. Tento postup si uvedieme na nasledovných príkazoch.

```
library(doParallel)
cl <- makeCluster(n)
registerDoParallel(cl)
foreach(i=1,...) %do% {telo kódu}
stopCluster(cl)
```

Ako prvé sme načítali balík doParallel. Následne sme vytvorili n paralelne bežiacich R prostredí a pomocou príkazu `foreach` vykonáme cyklus. Operátor `%do%` však vykonáva iterácie sekvenčne. Aby `foreach` vykonal príkaz paralelne, musia byť splnené dve podmienky:

- zadefinované paralelné prostredie
- namiesto `%do%` použiť operátor `%dopar%`.

Ak si otvoríme správcu úloh na našom počítači a zadefinujeme paralelné rozhranie napríklad štyroch R prostredí, tak v záložke procesy môžeme vidieť nasledovnú situáciu.

Správca úloh

Súbor Možnosti Zobraziť

Procesy | Výkon | História aplikácií | Spustenie | Používatelia | Podrobnosti | Služby

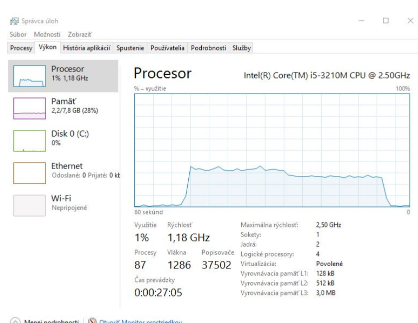
Názov	99% Procesor	36% Pamäť	2% Disk	0% Sieť
R for Windows front-end	24,0%	156,4 MB	0 MB/s	0 Mb/s
R for Windows front-end	23,5%	156,2 MB	0 MB/s	0 Mb/s
R for Windows front-end	23,4%	144,9 MB	0 MB/s	0 Mb/s
R for Windows front-end	23,4%	159,4 MB	0 MB/s	0 Mb/s
Shell Infrastructure Host	0,8%	3,9 MB	0 MB/s	0 Mb/s
Synaptics TouchPad 64-bit Enhancements	0,5%	4,0 MB	0 MB/s	0 Mb/s
System	0,5%	0,1 MB	0,1 MB/s	0 Mb/s
Task Manager	0,5%	16,4 MB	0 MB/s	0 Mb/s

Obr. 12: Zobrazenie R prostredí vytvorených na pozadí pri zadefinovanom paralelnom rozhraní

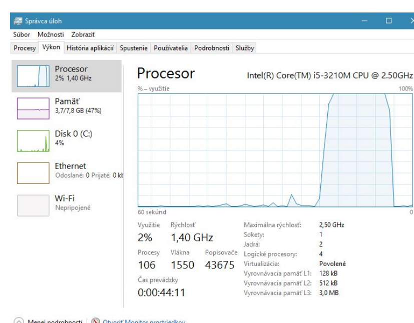
Tieto programy sa nám nezobrazia na lište, ale bežia iba na pozadí. Na obrázku 12 si tiež môžeme všimnúť, že každý program sa snaží naplno využiť kapacitu svojho jadra, čiže 25% (pracujme so 4 jadrovým procesorom). Ak už príkazy nepotrebujeme vykonávať paralelne, musíme zavrieť vytvorené prostredia. Na to slúži posledný z uvedených príkazov `stopCluster`.

Porovnajme si ešte využitie procesora pri práci s jedným prostredím a paralelným rozhraním. Ako už bolo spomenuté, R vie pracovať len s jedným jadrom. Nasledovná

dvojica obrázkov 13 a 14 nám dobre demonštruje rozdiel percentuálneho využitia procesora pri zložitejšej úlohe. Prvý obrázok zodpovedá práci so štandardným programom R. V tomto prípade si na grafe môžeme všimnúť, že príkaz sa vykonával dlhšie a využitie procesora nadobúdalo maximálne hodnoty len okolo 25-30%. Tieto skoky, respektíve rozdiely sú zapríčinené ostatnými procesmi. Na druhom obrázku, ktorý zodpovedá paralelnému rozhraniu so štyrmi R prostrediami, pozorujeme vykonanie tej istej úlohy kratšie, ale so 100% využitím výpočtovej sily počítača.



Obr. 13: Percentuálne znázornenie využitia procesora pri spustení programu R



Obr. 14: Percentuálne znázornenie využitia procesora pri spustení paralelného rozhrania v programe R

Parameter .combine

Pri použití paralelného programovania sú výsledky v R vypísané vo forme listu, respektíve zoznamu. Pomocou argumentu `.combine` môžeme výstup výsledkov upraviť. Môže byť špecifikovaný buď ako funkcia alebo názov funkcie. Zadefinovanie `.combine='c'` vykoná spojenie výsledkov do vektoru. Hodnoty `'cbind'` a `'rbind'` môžu kombinovať vektorové výsledky do matíc. Vstupy `.combine='+'` `.combine='*'` sčítajú, respektíve vynásobia výsledky z jednotlivých iterácií. Keďže tento parameter budeme využívať uvedieme niekoľko príkladov, ktoré nám lepšie ozrejmia jeho fungovanie. Uvažujme nasledovnú 3×3 maticu:

$$A= \begin{pmatrix} 2 & 2 & 2 \\ 3 & 3 & 3 \\ 4 & 4 & 4 \end{pmatrix}$$

Ako prvý príklad spravíme sčítanie všetkých prvkov matice:

```
foreach(i=1:3, .combine='+') %do% sum(A[,i])
```

Cyklus postupne sčíta prvky stĺpcov matice A a zadefinovaný parameter `.combine='+'` potom výsledky jednotlivých iterácii spočíta. Výsledkom je číslo 27, čiže súčet všetkých prvkov A. Na porovnanie vykonáme príkaz bez použitia `.combine`:

```
foreach(i=1:3) %do% sum(A),
```

ktorý sčíta prvky matice v každej iterácii, takže výsledkom bude list troch čísiel 27.

Ďalšími príkladmi sú sčítanie riadkov a stĺpcov matice:

```
foreach(i=1:3, .combine='+') %do% A[i,]
foreach(i=1:3, .combine='+') %do% A[,i]
```

V cykle sa postupne načítavajú riadky, respektíve stĺpce, ktoré sú parametrom `.combine='+'` sčítané po zložkách. Výsledkom sú teda vektory $(9, 9, 9)$ a $(6, 9, 12)$.

Posledný príkladom je použitie `.combine='c'` na riadky matice A:

```
foreach(i=1:3, .combine='c')%do% A[i,]
```

V cykle sa opäť len načítavajú riadky, ktoré sú prostredníctvom `.combine='c'` spojené do jedného výsledného vektora $(2, 2, 2, 3, 3, 3, 4, 4, 4)$

Späť k príkladu z predchádzajúcej kapitoly

Teraz sa vráťme k príkladu z kapitoly 4.2.1, v ktorom sme chceli odhadnúť rozdelenie výberového priemeru $\bar{X}_n$, kde $n = 100000$ a dáta sú i.i.d. z Poissonovho rozdelenia s parametrom λ. V riešení sme postupovali tak, že sme generovali n pozorovaní z daného rozdelenia, z ktorých sme odhadli výberový priemer. To sme opakovali 90000 krát. Keďže už vieme ako pracovať paralelne v R, môžeme tieto opakovania rozdeliť na viac

súčasne sa vykonávajúcich cyklov bez toho, aby sme museli manuálne otvárať viac prostredí. Použijeme na to nasledujúci kód, ktorý vytvorí j pracovníkov, respektíve R prostredí, medzi ktoré týchto 90000 opakovaní rozdelíme:

```
cl <- makeCluster(j)
registerDoParallel(cl)

xbar=vector(length=90000)
n=100000
start<-Sys.time()
xbar<-foreach(i=1:90000, .combine='c') %dopar% data=rpois(n ,lambda=1)
mean(data)
time<-Sys.time()-start
hist(xbar)
```

Výsledok každej iterácie je odhad $\bar{X}_n$, z ktorého pomocou parametra `.combine='c'` postupne vytvárame vektor odhadov. Z neho následne vytvoríme histogram. Pre `for` cysklus a rôzne voľby parametra j sme dostali nasledovné výpočtové časy:

j prostredí	for	2	3	4	5	6	7
čas	6:57	5:33	4:18	3:24	3:47	3:56	4:03

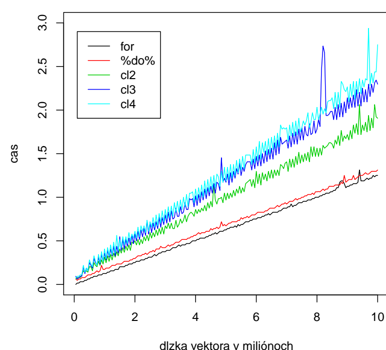
Tabuľka 1: Výpočtový čas odhadu rozdelenia vykonaný prostredníctvom `for` a `foreach` pre j prostredí

V tabuľke 1 môžeme pozorovať, že použitím príkazu `foreach` sa výpočtový čas znížil v závislosti od počtu vytvorených prostredí. Druhou skutočnosťou, ktorá stojí za povšimnutie, je nárast výpočtového času pre hodnoty 5, 6, 7. Je to spôsobené tým, že pracujeme so 4 jadrovým procesorom, dôsledkom čoho sa pri týchto výpočtoch jednotlivé prostredia bijú o dostupné jadrá. To v konečnom dôsledku spôsobuje spomalenie. Z tabuľky 1 vyplýva, že najoptimálnejšou voľbou je zvoliť $j = 4$. V kapitole 4.2.1 sme však uviedli, že dobrým pravidlom je zvoliť $n_c - 1$, kde n_c je počet jadier, čo znamená

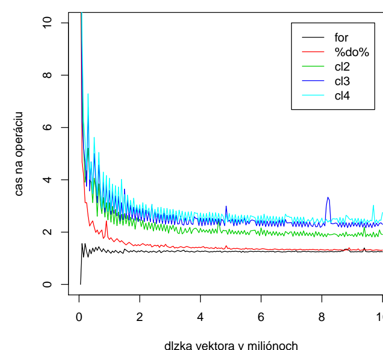
zvoliť $j = 3$. Keďže pre $j = 4$ sme dosiahli najrýchlejšie výsledky, budeme aj napriek uvedenému pravidlu pracovať s týmto počtom prostredí.

Porovnanie zložitosti opakovaného cyklu

V tejto časti si porovnáme dĺžku výpočtového času vzhľadom na náročnosť príkazu, ktorý opakujeme v cykle. Budeme porovnávať dva postupy. Ako prvé si pri oboch vygenerujeme vektor a z rovnomerného rozdelenia dĺžky *ArrayN*. Na neho budeme aplikovať súčet zložiek tohto vektora príkazom `sum(a)` a súčet sínusov zložiek v tvare `sum(sin(a))`. Tieto výpočty budeme vykonávať vo forme vektorov, pretože ako sme ukázali v časti 4.1 je to efektívnejšia cesta, ako ich sčítať po zložkách pomocou príkazu na cyklus `for`. Toto porovnanie vykonáme sekvenčne a paralelne pre rôzne počty pracovníkov tak, že budeme vyššie spomenuté funkcie opakovať 100 krát. Výsledkom teda bude list 100 rovnakých hodnôt. Postup vykonáme pre rôzne hodnoty dĺžky vektora *ArrayN*. Porovnajme najprv čas výpočtu `sum(a)` a `sum(sin(a))` pre vektor a dĺžky 1 milión. V prvom prípade realizácia výpočtu trvala 0.15 sekundy, zatiaľ čo v druhom až 6.72 sekundy. Rozdiel v zložitosti a trvaní výpočtu je dosť výrazný. Výsledky pre ďalšie hodnoty *ArrayN* a spôsob realizácie môžeme sledovať na nasledujúcich grafoch:



Obr. 15: Porovnanie výpočtového času `sum(a)` pre rôzne dĺžky vektora a sekvenčne a paralelne

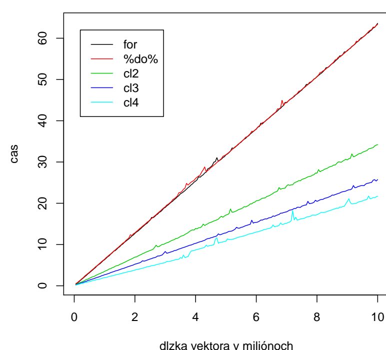


Obr. 16: Porovnanie výpočtového času `sum(a)` vzhľadom na operáciu pre rôzne dĺžky vektora a sekvenčne a paralelne

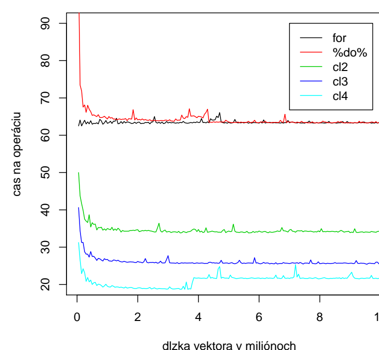
Na grafoch 15 a 16 sme znázornili realizáciu `sum(a)` pre hodnoty *ArrayN* od 50 tisíc až 10 miliónov s krokom 50 tisíc. Výsledky ukazujú, že paralelná realizácia je pomalšia pre všetky zhluky prostredí „cl2“, „cl3“, „cl4“, ktoré znázorňujú rozloženie výpočtu medzi 2, 3, 4 prostredia. Výpočty sekvenčnými cyklami sú v legende grafu označené

„for“, „%do%“. Zaujímavým faktom je aj výsledok väčšieho času vzhľadom na použitie väčšieho počtu prostredí.

Na obrázkoch 17 a 18 sme graficky znázornili realizáciu `sum(sin(a))` pre rovnaké hodnoty *ArrayN* ako v predchádzajúcom prípade. Rovnaké značenie je sme použili aj pre sekvenčný a paralelný výpočet.



Obr. 17: Porovnanie výpočtového času `sum(sin(a))` pre rôzne dĺžky vektora a sekvenčne a paralelne



Obr. 18: Porovnanie výpočtového času `sum(sin(a))` vzhľadom na operáciu pre rôzne dĺžky vektora a sekvenčne a paralelne

Z výsledkov môžeme pozorovať rýchlejší výpočet použitím paralelného vykonania týchto cyklov. Grafy ukazujú, že čím viac prostredí na pozadí vytvoríme, tým je to rýchlejšie.

Porovnaním grafov 15 a 17 sme teda ukázali, že paralelné programovanie nemá význam pokiaľ opakované príkazy nie sú časovo a výpočtovo náročnejšie. Príčinou tohto faktu sú operácie, ktoré prebiehajú na pozadí paralelného programovania. Tým je myslené rozdelenie práce a dát medzi jednotlivé R prostredia a následné spojenie výsledkov do požadovanej formy. Samotný výpočet menšieho počtu cyklov u pracovníka môže byť rýchlejší, ale v dôsledku operácií na pozadí je vykonanie dlhšie. Z toho vyplýva, že operácie so `sum(a)` sú v softvéri R veľmi rýchle a ich rozloženie na súbežné vykonávanie nie je efektívne. Podobnú situáciu môžeme vidieť aj na obrázkoch 16 a 18, ktoré znázorňujú rýchlosť vzhľadom na operáciu. Môžeme si všimnúť, že pre malé hodnoty *ArrayN* dosahuje paralelný výpočet väčšie časy ako pre vektory obsahujúce viac zložiek. Paralelné programovanie preto môže niekedy viac uškodiť, ako pomôcť. Preto je dôležité poznať náročnosť iterácie, ktorú chceme vykonať.

5 Porovnanie prístupov

V tejto časti porovnáme výpočtové časy metód uvedených v kapitole 2 pomocou nasledovných prístupov:

- klasické štatistické balíky v softvéri R
- paralelná realizácia prostredníctvom R
- balík RevoScaleR
- SQL Server v programe Microsoft SQL Server Management Studio (MSSMS)

Na toto porovnanie použijeme konkrétne datasety *Fraud* a *Fraud Score* obsahujúce 10 miliónov záznamov o podvodoch s kreditnými kartami. Prvý obsahuje dáta pre odhad modelov a druhý je určený na skórovanie. Záznamy obsahujú stĺpce ako ID zákazníka *custID*, pohlavie *gender*, štát z USA *state*, typ držiteľa karty *cardholder*, zostatok *balance*, počet transakcií *numTrans*, počet medzinárodných transakcií *numIntlTrans*, kreditný limit karty *creditLine* a informáciu o tom, či podvod nastal *fraudRisk*. Premenné *gender*, *state*, *cardholder*, *fraudRisk* sú kategorické, zvyšné majú numerický charakter.

My budeme pracovať iba s 2 miliónmi dát kvôli problémom softvéra R s držaním dát v pamäti počas výpočtov. Predtým, ako uvedieme výsledky simulácií si musíme ujasniť, ako postupovať pri paralelnom programovaní pre každú z metód.

5.1 Výsledky pre logistickú regresiu

V časti 2.3.1 sme pre získanie modelu logistickej regresie uviedli algoritmus prostredníctvom Newtonovej metódy. Tá na odhad parametrov, ktoré sú výstupom z modelu, využíva gradient prvých parciálnych derivácií a Hessovu maticu druhých parciálnych derivácií vierohodnostnej funkcie. Tie sa vyrátavajú v každej iterácii optimalizačného cyklu. Paralelné programovanie v tejto metóde využijeme práve na výpočet týchto derivácií. Gradient uvedený v rovnici 7 tak získame podľa článku [8] nasledovne:

$$\nabla\Big(\frac{\partial l}{\partial \beta_1}, \frac{\partial l}{\partial \beta_2}, \dots, \frac{\partial l}{\partial \beta_p}\Big) = \begin{cases} \sum_{i=1}^{250}(y_i - h(x_i)) \times x_i \\ \sum_{i=251}^{500}(y_i - h(x_i)) \times x_i \\ \sum_{i=501}^{750}(y_i - h(x_i)) \times x_i \\ \sum_{i=751}^{1000}(y_i - h(x_i)) \times x_i \end{cases},$$

kde i predstavuje počet riadkov z celkového počtu dát $n = 1000$. Z násobenia $y_i - h(x_i)$ dostávame číslo, ktorým prenásobíme všetky prvky i-teho riadku z dát. Výpočet sme rozložili medzi 4 pracovníkov, z ktorých každý vykonáva časť výpočtov potrebných na získanie gradientu. Po ich skončení sú medzivýsledky sčítané po zložkách a konečným výsledkom je gradient prvých parciálnych derivácií pre všetkých p parametrov.

Rozšírením z [8] je paralelný výpočet Hesseho matice, ktorý sme si odvodili vo vzorci 9. Jeho výpočet sme si rozložili na dva kroky tak, aby sme priamo získali výslednú maticu. Prvým krokom je časť, kde vyčíslime pomocnú maticu M:

$$M = h(x_i)(1 - h(x_i)) \times x_i \qquad i = 1, \dots, n,$$

v ktorej sme všetky riadky x_i prenásobili číslom $h(x_i)(1 - h(x_i))$. Túto realizáciu sme tiež paralelne rozdelili medzi 4 prostredia.

Druhým krokom je výpočet výslednej Hesseho matice. Tú získame násobením transponovanej matice dát X s pomocnou maticou M. Najrýchlejším paralelným spôsobom násobenia matíc je produkt riadkov prvej matice s druhou maticou tak, že výpočet je pre každý riadok realizovaný samostatne.

$$H = X^\top \times M = \begin{cases} (\mathrm{X}_1)^\top \times M \\ (\mathrm{X}_2)^\top \times M \\ \vdots \\ (\mathrm{X}_p)^\top \times M \end{cases},$$

kde hodnoty $1, \dots, p$ reprezentujú i-ty riadok transponovanej matice X, p je teda počet jej stĺpcov. Medzivýsledkom sú vektory, ktoré sú následne po riadkoch spájané do konečnej matice H. Z týchto získaných hodnôt vieme vyrátať hodnotu novej iterácie postupom uvedeným vo vzorci 8. Uvedený paralelný výpočet takto prebieha v každej iterácii, kým nie je splnená podmienka ohľadom veľkosti posunu. Po splnení podmienok je výstupom zoznam hodnôt parametrov, ktoré vstupujú do modelu. Ak sú niektoré premenné kategorické, pred začatím optimalizačného cyklu sú upravené na vektory jednotiek a núl tak, že každá kategória má svoj vlastný vektor okrem prvej. Tá je zahrnutá do interceptu. Až takto rozšírené dáta vstupujú do tohto cyklu.

Na našich dátach *Fraud* vybudujeme nasledovný model logistickej regresie:

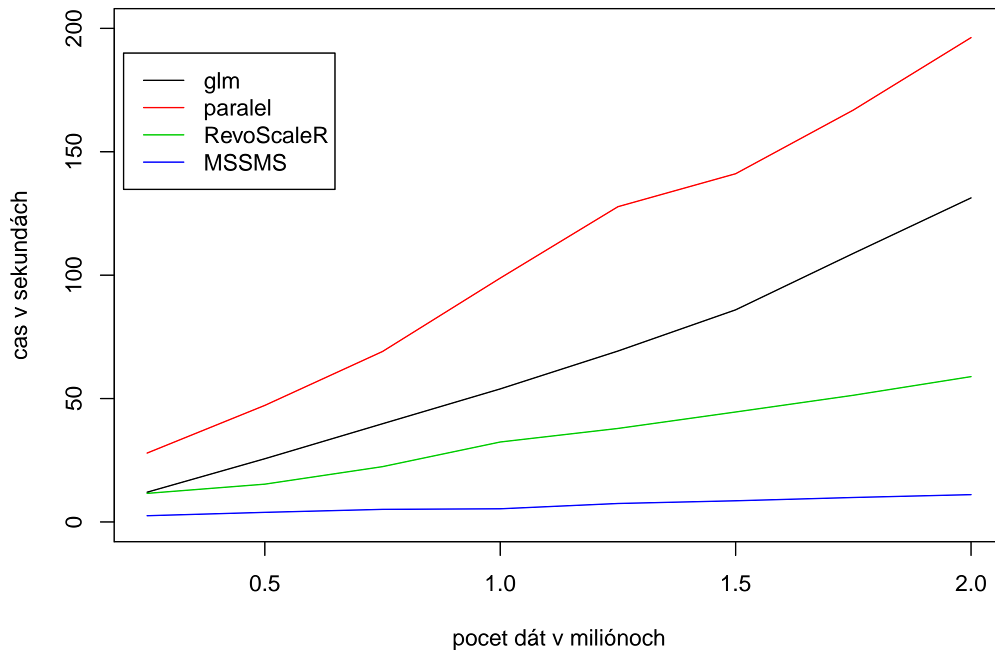
$$fraudRisk \sim gender + state + cardholder + balance \\ + numTrans + numIntlTrans + creditLine, \tag{10}$$

pre rôzne hodnoty počtu dát a pre všetky vyššie uvedené prístupy. Porovnanie výpočtových časov v minútach a sekundách môžeme sledovať v nasledujúcej tabuľke.

čas/počet dát [mil.]	0.25	0.5	0.75	1	1.25	1.5	1.75	2
R funkcia glm	0:12	0:25	0:39	0:59	1:09	1:26	1:49	2:11
Paralelná realizácia	0:28	0:47	1:09	1:38	2:08	2:21	2:47	3:16
rxGlm z RevoScaleR	0:12	0:15	0:22	0:32	0:38	0:44	0:51	0:59
Externý skript cez MSSMS	0:03	0:04	0:05	0:05	0:07	0:09	0:10	0:11

Tabuľka 2: Výpočtový čas trénovania modelu logistickej regresie

Hodnoty z tabuľky 2 sme graficky znázornili na obrázku 19.



Obr. 19: Porovnanie výpočtového času trénovania modelu logistickej regresie pre jednotlivé prístupy

Čierna krivka znázorňuje realizáciu modelu prostredníctvom štandardného prístupu softvérom R funkciou `glm` z balíku stats [11], červená označuje náš paralelný výpočet v tomto softvéri, zelená využitie balíka RevoScaleR a funkcie `rxGlm`, modrá externý R skript v programe MS SQL Server Management Studio, v ktorom si prostredníctvom dotazu zavoláme rovnaké dáta ako v prístupoch predtým a použitím funkcie `rxGlm` v externom R skripte dostaneme hodnoty parametrov modelu. Tie boli pre každý prístup a rôzne hodnoty počtu dát rovnaké.

Z grafu na obrázku 19 môžeme pozorovať, že najrýchlejším spôsobom je použitie externého skriptu. Druhým najrýchlejším spôsobom bolo trénovanie modelu prostredníctvom balíka RevoScaleR a funkcie `rxGlm`. Naopak najpomalšie dopadla paralelná realizácia, pretože sme ju aplikovali na funkcie súčtu a násobenia, ktoré sú v R veľmi rýchle. Tým sa proces viac spomalil. Nastal teda prípad, kedy je paralelné programovanie neefektívne. Druhým dôvodom dlhšieho výpočtu bola aj premenná *štát z USA*, pre ktorú sme museli vytvoriť vektory núl a jednotiek pre každý štát okrem jedného, ktorý bol zahrnutý v intercepte. Pre všetky hodnoty počtu dát sa preto muselo k dátam vytvoriť ďalších 50 vektorov. Tento proces prebiehal aj pre ostatné kategorické premenné. Tie však obsahovali iba 2 hodnoty. Takáto úprava trvala pre 2 milióny riadkov takmer 3 minúty.

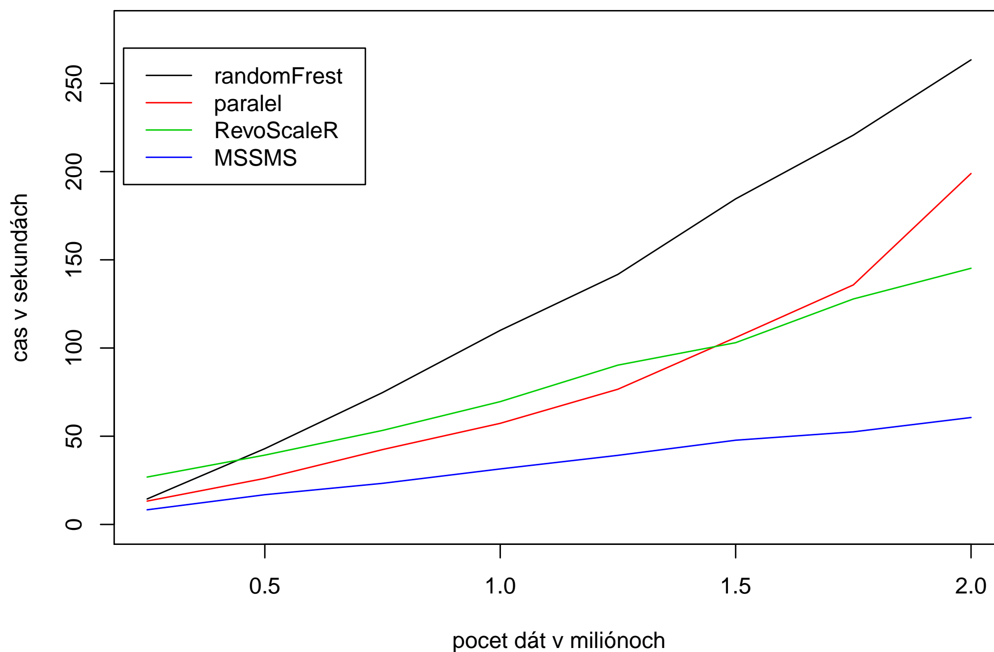
5.2 Výsledky pre náhodný les

V časti 2.2 sme uviedli ako sa budujú náhodné lesy. Princíp metódy spočíva vo vytvorení niekoľkých rozhodovacích stromov. Na predikciu sa potom používajú výstupy z každého jedného stromu a väčšinovým hlasovaním sa vyberie trieda nového pozorovania. Paralelná implementácia tejto metódy spočíva v rozložení počtu stromov, ktoré sa trénujú, medzi jednotlivé prostredia. Ak vytvárame les s m rozhodovacími stromami a k dispozícii máme n_c jadier procesora, tak budovanie modelu rozložíme na tieto jadrá tak, že každé trénuje m/n_c stromov. Výsledky takto rozdistribuovaných úloh sa následne spoja a vytvoria jeden model. V simuláciách sme budovali náhodný les podľa modelu 10 s 20 rozhodovacími stromami tak, že ich trénovanie rozložíme po 5 stromov medzi 4 R prostredia. V nasledujúcej tabuľke môžeme pozorovať výpočtové časy v minútach a sekundách pre rôzne hodnoty počtu dát.

čas/počet dát [mil.]	0.25	0.5	0.75	1	1.25	1.5	1.75	2
R funkcia randomForest	0:14	0:43	1:15	1:50	2:21	3:04	3:40	4:23
Paralelná realizácia	0:13	0:26	0:42	0:57	1:16	1:46	2:15	3:18
rxDForest z RevoScaleR	0:27	0:39	0:53	1:09	1:30	1:43	2:07	2:25
Externý skript cez MSSMS	0:08	0:17	0:23	0:31	0:39	0:48	0:52	1:01

Tabuľka 3: Výpočtový čas trénovania modelu náhodný les

Hodnoty z tabuľky 3 sme graficky znázornili na obrázku 20. Farebné označenie jednotlivých prístupov je rovnaké ako na obrázku 19. Pri štandardnom prístupe sme použili funkciu `randomForest` z balíka randomForest [12]. Paralelnú realizáciu vykonávame tiež prostredníctvom tejto funkcie tak, že generujeme 4 menšie náhodné lesy, ktoré nakoniec spojíme do jedného. V prístupe s balíkom RevoScaleR pracujeme s funkciou `rxDForest`. Tú využívame aj vo štvrtom zo spomínaných prístupov, kde ju aplikujme na rovnaké dáta zavolané do externého R skriptu.



Obr. 20: Porovnanie výpočtového času trénovania modelu náhodný les pre jednotlivé prístupy

Z grafu na obrázku 20 hodnotíme, že najrýchlejším spôsobom je opäť použitie externého

skriptu v MS SQL Server Management Studio, znázornené modrou krivkou. Červená a zelená krivka, ktoré znázorňujú paralelnú interpretáciu a použitie funkcie z balíka RevoScaleR sú blízko seba, a dokonca sa aj pretínajú. Paralelným programovaním sme sa teda priblížili k výpočtovým časom prístupu, ktorý ponúka balík RevoScaleR. Pre niektoré hodnoty veľkosti ich tento spôsob aj predbehol.

5.3 Výsledky pre k najbližších susedov

Metodologický postup tejto metódy sme predstavili v podkapitole 2.4. Algoritmus pracuje tak, že pre každú klasifikáciu nového pozorovania zráta Euklidovské vzdialenosti pre každé pozorovanie v trénovacej množine. Z nich potom vyberie k najbližších, z ktorých klasifikácie väčšinovým hlasovaním určí triedu pre nové pozorovanie. Paralelnú realizáciu sme vykonali rozdelením neklasifikovaných vzoriek medzi 4 prostredia.

Keďže táto metóda pracuje len s numerickými hodnotami, pri výpočte sme využili iba stĺpce *balance*, *numTrans*, *numIntlTrans*, *creditLine*, *fraudRisk*. Model bude preto nasledovný:

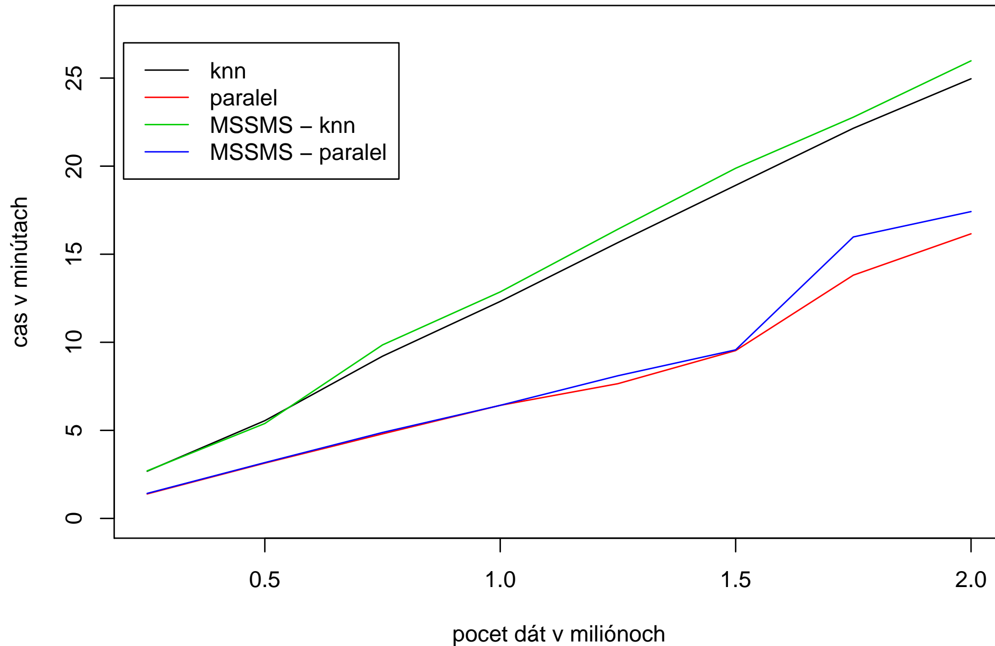
$$fraudRisk \sim balance + numTrans + numIntlTrans + creditLine.$$

Simulácie sme vykonávali na trénovacej množine s rozsahom 100 tisíc pozorovaní pre hodnotu $k = 3$. Hľadali sme teda 3 najbližšie body a klasifikovali rôzne veľké dátové sety. Keďže balík RevoScaleR neobsahuje funkciu pre k najbližších susedov, porovnali sme výpočtové časy prostredníctvom R funkcie `knn` z balíku class 13 s jej paralelnou realizáciou. Tie isté naprogramované kódy s klasickými balíkmi a funkciami sme spustili aj prostredníctvom MSSMS. Výsledky časov sú vypísané v nasledujúcej tabuľke.

čas/počet dát [mil.]	0.25	0.5	0.75	1	1.25	1.5	1.75	2
R funkcia knn	2:41	5:33	9:13	12:19	15:40	18:55	22:09	24:57
Paralelná realizácia	1:24	3:08	4:48	6:25	7:39	9:34	13:49	16:10
R funkcia knn v MSSMS	2:42	5:23	9:51	12:52	16:25	19:53	22:47	25:59
Paralelná realizácia v MSSMS	1:25	3:10	4:53	6:25	8:06	9:40	15:59	17:25

Tabuľka 4: Výpočtový čas klasifikácie metódou k najbližších susedov

Hodnoty z tabuľky 4 sme graficky znázornili na obrázku 21.



Obr. 21: Porovnanie výpočtového času klasifikácie metódou k najbližších susedov pre jednotlivé prístupy

Z obrázku 21 s identickým značením ako v predchádzajúcich metódach je vidno, že paralelné programovanie opäť zrýchlilo výpočtové časy oproti použitiu klasického prístupu. Na grafe môžeme tiež pozorovať, že vykonanie kódov v R alebo MSSMS má podobné hodnoty. Z toho môžeme usúdiť, že pokiaľ nepoužívame funkcie z balíku RevoScaleR, tak sa vykonanie výpočtov v programe MSSMS neurýchľuje.

5.4 Zhrnutie výsledkov

Zo získaných výsledkov môžeme usúdiť, že pokiaľ pracujeme s funkciami z balíku RevoScaleR v prostredí Microsoft SQL Server Management Studio, tak výpočtový čas trénovania modelov je najrýchlejší. To sa nám potvrdilo pri logistickej regresii a aj pri náhodných lesoch. Druhým najrýchlejším spôsobom bola práca s knižnicou RevoScaleR priamo v prostredí R. Pomocou neho si môžeme zadefinovať výpočtový kontext v mieste, kde je uložená databáza, čo zrýchľuje výpočet. Funkcie z tohto balíku sú upravené, tak aby ich realizácia bola rýchlejšia, čo sa nám aj potvrdilo. Budovanie modelov

bolo rýchlejšie ako pomocou funkcií zo známych štatistických balíkov. V tretej metóde, ktorou bola k najbližších susedov, sme porovnávali len jej vykonanie v softvéroch R a MSSMS prostredníctvom funkcie z balíka class, pretože balík RevoScaleR ju neobsahuje. V tomto prípade sú prístupy podobné, pretože dosahovali približne rovnaké výpočtové časy.

Záver

Štandardným prístupom k databázam v softvéri R je pripojenie pomocou knižnice RODBC. Nevýhodou tohto spôsobu sú nedostatky R v podobe obmedzení na výkon a pamäť, pretože R pracuje len s jedným jadrom procesora a umožňuje načítať len dáta veľkosti prístupnej operačnej pamäte. Tento problém pomáha odstrániť integrácia jazyka R do SQL Servera, ktorá so sebou priniesla veľa výhod. Umožňuje naďalej pracovať s obľúbenými štatistickými knižnicami a zároveň využívať robustnosť a silu SQL Servera.

Prvou úlohou, s ktorou sme sa museli popasovať, bola inštalácia požadovaného softvéra, pretože verzia 2016 vyžadovala ako minimálne požiadavky Windows 8 alebo 10. Taktiež bolo potrebné nainštalovať Visual Studio. Až potom sme mohli začať pracovať s inštaláciou SQL Servera 2016 a MS SQL Server Management Studia, v ktorom sme vykonávali naše testy externých R skriptov. Aby tieto skripty fungovali, museli sme pri inštalácii pridať doplnok SQL Server R Services, ktorý umožňuje komunikáciu medzi týmito dvoma jazykmi.

V práci sme najprv predstavili vybrané štatistické metódy, na ktorých budeme vykonávať naše porovnania. Vybrali sme si 3 klasifikačné metódy; náhodné lesy, logistickú regresiu a k najbližších susedov. Prvý kontakt s týmto novým prístupom k databázam nám ukázal, že pracuje oveľa rýchlejšie a efektívnejšie. Toto prepojenie nám umožnilo vykonávať externé R skripty priamo v MS SQL Server Management Studiu. Druhou možnosťou bola práca s balíkom RevoScaleR priamo v softvéri R, ktorý obsahuje viacero známych funkcií z tohto prostredia. Výhodou tohto prístupu je, že nenačítava dáta priamo do pamäte, ale zachováva si iba cestu k nim. Preto sme v ďalších častiach práce ukázali ako pracovať s externými R skriptami, balíkom RevoScaleR a ako optimálne zrýchliť výpočtové časy programov v softvéri R. Jednou z možností bolo aj paralelné programovanie, prostredníctvom ktorého sme sa chceli priblížiť k výpočtovým časom tohto nového prístupu. Porovnávali sme preto štyri prístupy: externý skript v programe MS SQL Server Management Studio, balík RevoScaleR, klasické štatistické balíky a ich paralelnú realizáciu prostredníctvom softvéra R.

Veľmi dôležitým krokom pri paralelnom programovaní bolo správne určiť, ktoré časti budovania modelu možno takto rozložiť. Na toto pochopenie nám pomohla podrobná

teoretická časť úvodných kapitol. Na porovnanie vyššie spomenutých prístupov sme mali k dispozícií dáta o podvodoch kreditných kariet, ktoré obsahovali 10 miliónov záznamov. Aby sme mohli vykonať primerané porovnanie pracovali sme len s 2 miliónmi, pretože softvér R mal pri budovaní modelov problém alokovať do pamäte väčšie množstvo dát. Z výsledkov simulácií uvedených v poslednej kapitole sme zistili, že najrýchlejším spôsobom práce je použitie funkcií balíka RevoScaleR v externom skripte. Druhým najefektívnejším spôsobom bolo použitie funkcií tohto balíka priamo v softvéri R. Paralelnou realizáciou sa nám k nim podarilo priblížiť iba pri metóde náhodných lesov. Pri logistickej regresii bolo paralelné programovanie neefektívne, pretože sme ho mohli aplikovať iba na funkcie sčítania a násobenia matíc, ktoré sú v R veľmi rýchle. Druhým dôvodom mohla byť aj štruktúra dát, ktorá spôsobila zdĺhavú prípravu dát do modelu. Pri k najbližších susedov sme sa stretli so situáciou, kedy balík RevoScaleR neobsahoval funkciu k tejto metóde. Preto sme testovali jej vykonanie štandardným a paralelným prístupom v R a Management Studiu. Pre oba prístupy sme dostali takmer rovnaké hodnoty výpočtových časov, bez ohľadu na použitý softvér.

Simulačnými výsledkami sme potvrdili, že integrácia R do SQL Servera priniesla veľa výhod nielen zrýchlením známych štatistických metód, ale aj odstránením zbytočného presunu dát, ktoré ja často pomalé a neefektívne.

Zoznam použitej literatúry

[1] Han, J., Kamber, M., Pei, J.:*Data Mining Concepts and Techniques*, Elsevier Inc., Waltham, 2012

[2] Bramer, M.:*Principles of Data Mining*, Springer-Verlag, Londýn, 2007

[3] Woody, B., Dean, D., GuhaThakurta, D., Bansal, G., Conners, M., Tok, W.: *Data Science with Microsoft SQL Server 2016*, Microsoft Corporation, Redmond, 2016

[4] Using R Code in Transact-SQL (SQL Server R Services), dostupné na internete (27.03.2017): https://docs.microsoft.com/en-us/sql/advanced-analytics/tutorials/rtsql-using-r-code-in-transact-sql-quickstart

[5] RevoScaleR Functions, dostupné na internete (27.03.2017): https://msdn.microsoft.com/en-us/microsoft-r/scaler/scaler

[6] Three ways to call C/C++ from R, dostupné na internete (27.03.2017): https://www.r-bloggers.com/three-ways-to-call-cc-from-r/

[7] Uyttendaele, N.: *How to speed up R code: an introduction*, Working Paper, Université catholique de Louvain, Leuven, 2015, dostupné na internete (27.03.2017): https://arxiv.org/abs/1503.00855

[8] Huang, R., Xu, W.: *Performance Evaluation of Enabling Logistic Regression for Big Data with R*, Conference Paper, Big Data IEEE International Conference, 2015, dostupné na internete (27.03.2017): https://www.researchgate.net/publication/303543502_Performance_Evaluation_of_Enabling_Logistic_Regression_for_Big_Data_with_R

[9] Weston, S.: *doParallel: Foreach Parallel Adaptor for the 'parallel' Package*, R package version 1.0.10, 2015, dostupné na internete (27.3.2017): http://CRAN.R-project.org/package=doParallel

[10] Weston, S.: *foreach: Provides Foreach Looping Construct for R*, R package version 1.4.3, 2015, dostupné na internete (27.3.2017): http://CRAN.R-project.org/package=foreach

[11] R Core Team: *R: A Language and Environment for Statistical Computing*, R Foundation for Statistical Computing, Viedeň, 2015, dostupné na internete (27.3.2017): https://www.R-project.org/

[12] Liaw, A., Wiener, M.: *Classification and Regression by randomForest*, R News(2), 2002, 18-22, dostupné na internete (27.3.2017): http://CRAN.R-project.org/doc/Rnews/

[13] Venables, W. N., Ripley, B. D.: *Modern Applied Statistics with S*, Springer, New York, 2002, dostupné na internete (27.3.2017):
http://www.stats.ox.ac.uk/pub/MASS4