TECHNICKÁ UNIVERZITA V LIBERCI
Fakulta mechatroniky, informatiky
a mezioborových studií

VIZUALIZACE KLASIFIKAČNÍ ÚLOHY PRO E-LEARNINGOVÝ PORTÁL ALS

Diplomová práce

Studijní program: N2612 – Elektrotechnika a informatika
Studijní obor: 1802T007 – Informační technologie

Autor práce: **Bc. Jaromír Šída**
Vedoucí práce: RNDr. Klára Císařová, Ph.D.

Liberec 2015

TECHNICAL UNIVERSITY OF LIBEREC
Faculty of Mechatronics, Informatics
and Interdisciplinary Studies

VISUALIZATION OF A CLASSIFICATION TASK FOR E-LEARNING PORTAL ALS

Diploma thesis

Study programme: N2612 – Electrical Engineering and Informatics
Study branch: 1802T007 – Information Technology

Author: **Bc. Jaromír Šída**
Supervisor: RNDr. Klára Císařová, Ph.D.

Liberec 2015

TECHNICKÁ UNIVERZITA V LIBERCI
Fakulta mechatroniky, informatiky a mezioborových studií
Akademický rok: **2014/2015**

ZADÁNÍ DIPLOMOVÉ PRÁCE

(PROJEKTU, UMĚLECKÉHO DÍLA, UMĚLECKÉHO VÝKONU)

Jméno a příjmení:	**Bc. Jaromír Šída**
Osobní číslo:	**M13000202**
Studijní program:	**N2612 Elektrotechnika a informatika**
Studijní obor:	**Informační technologie**
Název tématu:	**Vizualizace klasifikační úlohy pro e-learningový portál ALS**
Zadávající katedra:	**Ústav mechatroniky a technické informatiky**

Zásady pro vypracování:

1. Prostudujte jeden ze způsobů automatického budování stromů pro potřeby klasifikace.
2. Prostudujte redukci kategorií u kategoriálních dat, naprogramujte ji tak, aby na vzorových datech bylo vysvětlené její použití.
3. Navrhněte výkladový způsob vizualizace vybraného algoritmu pro výuku DM a naprogramujte jej jako počítačový experiment s parametrickými vstupy a přiměřeným stupněm obecnosti.
4. Téma zpracujte pro e-learningový kurz na portále ALS.

Rozsah grafických prací: **dle potřeby dokumentace**

Rozsah pracovní zprávy: **40–50 stran**

Forma zpracování diplomové práce: **tištěná/elektronická**

Seznam odborné literatury:

[1] **Yong Yin, Ikou Kaku, Jiafu Tang: Data Mining, Springer London Ltd, 2011.**

[2] **Steve McConnell: Dokonalý kód, Computer Press, 2006.**

[3] **Kotler Philip: Marketing management, Grada, 2010.**

[4] **Hendl J.: Přehled statistických metod zpracování dat, Portál, 2006.**

[5] **Olivia Parr Rud: Datamining, Computer Press, 2006.**

[6] **http://www.msps.cz/data-mining/**

[7] **http://www.spss.cz/pasw_modeler.htm**

[8] **Tutoriály k SAP a další materiály na WWW stránkách.**

Vedoucí diplomové práce: **RNDr. Klára Císařová, Ph.D.**
Ústav mechatroniky a technické informatiky

Datum zadání diplomové práce: **10. října 2014**

Termín odevzdání diplomové práce: **15. května 2015**

prof. Ing. Václav Kopecký, CSc.
děkan

Technická univerzita v Liberci - 14 -

L.S.

doc. Ing. Milan Kolář, CSc.
vedoucí ústavu

V Liberci dne 10. října 2014

Prohlášení

Byl jsem seznámen s tím, že na mou diplomovou práci se plně vztahuje zákon č. 121/2000 Sb., o právu autorském, zejména § 60 – školní dílo.

Beru na vědomí, že Technická univerzita v Liberci (TUL) nezasahuje do mých autorských práv užitím mé diplomové práce pro vnitřní potřebu TUL.

Užiji-li diplomovou práci nebo poskytnu-li licenci k jejímu využití, jsem si vědom povinnosti informovat o této skutečnosti TUL; v tomto případě má TUL právo ode mne požadovat úhradu nákladů, které vynaložila na vytvoření díla, až do jejich skutečné výše.

Diplomovou práci jsem vypracoval samostatně s použitím uvedené literatury a na základě konzultací s vedoucím mé diplomové práce a konzultantem.

Současně čestně prohlašuji, že tištěná verze práce se shoduje s elektronickou verzí, vloženou do IS STAG.

Datum: 14.05.2015

Podpis: 

Poděkování

Děkuji své vedoucí diplomové práce RNDr. Kláře Císařové, Ph.D. za odborné vedení, za pomoc a rady při zpracování této práce.

Rovněž děkuji rodině, především rodičům, za neustálou podporu po dobu celého mého studia.

Abstrakt

Tato práce se zabývá tématikou z oblasti klasifikačních úloh v data miningu. Řeší výukovou implementaci algoritmu CHAID v modifikaci Exhaustive. Jedná se o algoritmus vytvářející rozhodovací stromy. Před samotným popisem implementace je čtenář seznámen se základními pojmy data miningu, statistickými postupy použitými v algoritmu CHAID a následně je samotný algoritmus detailně popsán.

Zadání bylo řešeno formou modulárního systému, který odděluje implementaci algoritmu od prezentační vrstvy. Prezentační vrstva byla realizována formou WPF aplikace za dodržení návrhového vzoru MVVM. V této práci je popsán proces optimalizace algoritmu a jeho paralelizace. Výsledkem optimalizace a paralelizace je téměř trojnásobné zlepšení oproti prvotní implementaci. Dále jsou v diplomové práci popsány způsoby, jakými bylo dosaženo zmiňované výukovosti.

Přínos výsledku této diplomové práce je v navržení možného způsobu, jak vytvářet podobné výukové implementace algoritmů z oblasti data miningu.

Klíčová slova

CHIAD, Chi square automatic interaction detector, EXHAUSTIVE CHAID, rozhodovací stromy, data mining, Big Data

Abstract

This thesis focuses on the area of classification tasks in Data mining. It solves the implementation of the EXHAUSTIVE CHAID algorithm in an educational form. It is an algorithm that creates decision trees. Prior the description of the implementation, the thesis introduces reader to the fundamental terms of Data mining, statistical approaches used in the algorithm and provides detailed description of the algorithm itself.

Solution of the task is a modular system, where the implementation of the algorithm and the presentation layer is separated. The presentation layer is implemented as WPF application. MVVM design pattern is used. In the thesis the optimization and parallelization process is described. The result of this process is that the algorithm has been improved almost three times in the terms of computing time from the initial implementation. The thesis also discuss the approaches chosen to meet the educational goal.

Acquisition of the thesis is the prototype of modular system that can be used on similar tasks that target on the education of students.

Keywords

CHIAD, Chi square automatic interaction detector, EXHAUSTIVE CHAID, decision trees, data mining, Big Data

Obsah

1 Úvod 10
2 Základní popis problematiky a seznámení s pojmy 11
2.1 Data mining a získávání znalostí z databází 11
2.2 Nástroje DM 11
2.3 Typy úloh 11
2.3.1 Klasifikace / predikce 11
2.3.2 Deskripce 12
2.4 Metodiky v data miningu 12
2.5 CRISP-DM 13
2.6 Základní pojmy oblasti data miningu 14
2.6.1 Typ atributu 15
2.6.2 Role atributu 15
3 Rozhodovací stromy 16
3.1 Vybraný rozhodovací strom – CHAID 18
3.2 Modifikace Exhaustive CHAID 18
4 Statistické přístupy použité v algoritmu 20
4.1 Testování statistických hypotéz 20
4.2 Statistické testy 22
4.2.1 Pearsonův chí kvadrát test (Test dobré shody) 22
4.2.2 ANOVA F-Test 24
4.3 Využití p-hodnot v algoritmu 24
5 Konkrétní popis implementovaného algoritmu 25
5.1 Parametry vstupující do implementace 25
5.2 Algoritmus 26
5.2.1 Diagram 26
5.2.2 Popis algoritmu 27
6 Návrh aplikace 29
6.1 Náplň praktické části 29
6.2 Klíčová hlediska při návrhu systému 29
6.3 Použité technologie 30

6.3.1 Windows Presentation Foundation (WPF) 30
6.3.2 Model View ViewModel (MVVM) 31
6.3.3 Použité externí knihovny 32
6.4 Moduly aplikace 33
7 Realizovaná aplikace 35
7.1 Interfaces 35
7.2 Statistické výpočty 36
7.3 Implementace CHAID 36
7.3.1 Vybrané datové struktury 36
7.3.2 Způsob růstu stromu (běhu algoritmu) 37
7.3.3 Optimalizace růstu stromu 37
7.3.4 Záznam z běhu algoritmu 40
7.4 Prezentační aplikace 41
7.4.1 Krok načítání dat 42
7.4.2 Krok nastavení parametrů modelu 42
7.4.3 Krok s vytvořenými stromy 43
7.4.4 Vlastnosti aplikace za účelem výkladu při studiu algoritmu CHAID 46
8 Závěr 48
9 Seznam použité literatury 49

Seznam ilustrací

Ilustrace 1: Posloupnost jednotlivých činností CRISP-DM (zdroj: en.wikipedia.org) 14
Ilustrace 2 Primitivní datová struktura 15
Ilustrace 3 Rozhodovací strom 16
Ilustrace 4 Demonstrace p-hodnoty 21
Ilustrace 5 Algoritmus CHAID 26
Ilustrace 6 Role v MVVM (zdroj: http://msdn.microsoft.com) 32
Ilustrace 7 Moduly v systému 33
Ilustrace 8 Vybraná rozhraní 35
Ilustrace 9 Sekvenční algoritmus 38
Ilustrace 10 Využití prostředků při sekvenční implem. (výstup z programu dotTrace) 38
Ilustrace 11 Paralelní algoritmus 39
Ilustrace 12 Využití prostředků při paralelní implem. (výstup z programu dotTrace) 39
Ilustrace 13 Rozložení prezentační aplikace 41
Ilustrace 14Krok nastavení parametrů modelu 43
Ilustrace 15 Krok zobrazující vygenerovaný strom 44
Ilustrace 16 Samostatné okno s informacemi o uzlu 45
Ilustrace 17 Záznam z běhu algoritmu 46
Ilustrace 18 Upravený tooltip 47

Seznam tabulek

Tabulka 1 Přehled základních vlastností vybraných algoritmů 18
Tabulka 2 Varianty rozhodnutí pro testované hypotézy 20
Tabulka 3 Kontingenční tabulka pro demonstraci chí kvadrát testu – empirické četnosti. 22
Tabulka 4 Kontingenční tabulka - očekávané četnosti 23
Tabulka 5 Přehled dosažených výsledků optimalizace algoritmu 40

1 Úvod

Měl-li bych vyjmenovat zajímavé trendy a otázky dnešní doby v oblasti informačních technologií, určitě bych v jednu chvíli zmínil způsoby využívání dat, které každým dnem poskytujeme různým společnostem. Skutečnost je taková, že jsou sledována a ukládána data téměř všeho druhu. Od dat, která jsou intuitivně užitečná, jako třeba adresy osob, odpovědi v dotaznících či zaslané emaily, až po data, která by na první pohled nemusela najít žádná další uplatnění. Například pohyb myši po webové stránce, zvuková charakteristika hlasu během telefonního hovoru nebo třeba fráze vyhledávané v internetových vyhledávačích. Je zřejmé, že množství takových dat je velice objemné a vyžaduje velikou kapacitu datových úložišť, stejně tak jako několik dalších technických požadavků. Těmto požadavkům je však dnes možné dostát díky technologickému pokroku minulých desetiletí. Zbývá tedy otázka, jak s těmito daty nakládat dál, máme-li potřebné prostředky. Tím se dostávám k fenoménu *Big Data*

Pojem *Big Data* obsahuje několik dílčích činností, jež se vztahují k nakládání s velkým množstvím dat. Jednou z těchto činností je analýza dat. V prvním odstavci jsem zmínil několik typů uložených informací, které, ačkoli se to nemusí na první pohled zdát zřejmé, mají svá sekundární využití. Například evidování vyhledávaných frází na internetovém vyhledávači Google. Lze odhadovat, že primární účel uchovávání takových dat byl vylepšovat takzvané vyhledávací „našeptávače“. Avšak společnost Google tato stejná data analyzovala a využila pro predikci šíření epidemie chřipky v USA, a to s velice dobrým výsledkem [1]. Povedlo se jim ze stovek milionů frází z minulosti vytipovat desítky, které měly vysokou korelaci s výskytem nemocí v daných letech. Důvody, proč se jednalo právě o tyto fráze, však nebyly podstatné – informace byla nalezena.

Tento přístup, při kterém kauzalita ustoupila do pozadí, je dle mého názoru přinejmenším zajímavý, a to je důvod, proč jsem se rozhodl zabývat se tématikou z oblasti Big Data. Konkrétně algoritmem ze sféry data miningu, který je jedním z nástrojů analýzy dat, a jeho implementací ve výukové formě.

2 Základní popis problematiky a seznámení s pojmy

2.1 Data mining a získávání znalostí z databází

Data mining je analytická část procesu, který nazýváme *získávání znalostí z databází*. Lze ho popsat jako *vyhledávání nových, neočekávaných informací v rozsáhlých datech* [2], přičemž výraz *rozsáhlá data* dostává každým rokem jiný význam, proto bych objem dat nepovažoval za stěžejní v této definici.

Procesu získávání znalostí z databází využívají lidé v mnoha profesních odvětvích. Využití ve zdravotnictví již bylo naznačeno v úvodu. Dalšími jsou například firmy, které za pomoci tohoto přístupu hledají možnosti k většímu profitu [3]. Využití není zdaleka jen v komerční sféře, ale také například u kriminalistů. Tyto skupiny budu v navazujících částech kapitoly nazývat *zadavateli*.

Zadavatel zná hrubé cíle, kterých chce dosáhnout. V příkladu z úvodu by to mohla být otázka, zda lze dát do souvislosti vyhledávané fráze a výskyt chřipky. V tuto chvíli se takto formulovaného problému chopí specialisté z oblasti získávání znalostí z databází a předzpracovávají data. Potom na již předzpracovaná data používají dostupné nástroje data miningu.

2.2 Nástroje DM

Metody data miningu jsou založené na algoritmech, nástrojích a přístupech z několika oblastí počítačové vědy. Těmito oblastmi jsou například umělá inteligence, strojové učení či statistika. Tyto algoritmy přitom lze zařadit do příbuzných skupin. Mezi příklady takových skupin se řadí například:

- Klasifikační a regresní stromy
- Metody shlukové analýzy
- Neuronové sítě

2.3 Typy úloh

2.3.1 Klasifikace / predikce

Algoritmům data miningu jsou předložena data obsahující historické záznamy a každý tento záznam je součástí nějaké třídy, nebo nabývá nějaké hodnoty. V klasifikačních / predikčních úlohách se pak snažíme najít způsob, jak pokud možno co

nejspolehlivěji přiřadit nové záznamy do daných tříd, popřípadě odhadnout hodnotu určitého atributu nového záznamu.

2.3.2 Deskripce

Úkolem je zde nahlížet na vztahy v datech. Pomocí algoritmů shlukové analýzy se tak mohou vyhledávat skupiny dat mající podobný charakter svých hodnot, takzvané shluky. Nebo naopak data, která se významně liší, takzvané anomálie. Pro změnu pomocí algoritmů regresní analýzy se vyhledávají vazby mezi jednotlivými proměnnými.

Nejdůležitějším prvkem data miningu v těchto úlohách je však výsledky správně interpretovat. Výstupu se musí přiřadit kontext dané problematiky a význam. Rovněž je třeba vyřadit výsledky, které již byly zjevné před použitím algoritmů, a nemají proto žádnou informativní hodnotu. Je také potřeba zvážit, zda daný konkrétní výsledek není jen důsledkem nedostatečného množství dat či nedostatečné kvality dat.

2.4 Metodiky v data miningu

V předchozích odstavcích jsem zmiňoval, že ještě předtím, než se v datech začnou vyhledávat nové informace, je třeba do nich *nahlédnout*. Dochází ke zhodnocení, zda jsou v dané podobě vhodná pro vybraný algoritmus. Další ze zmíněných činností byla interpretace výsledků a jejich zhodnocení. Toto jsou dva příklady procesů, které se během získávání znalostí z databází dějí vždy. To, v jakých fázích se tyto činnosti budou provádět, už záleží na konkrétní řešitelské skupině, respektive firmě.

Tím se dostávám k metodikám, neboli k posloupnosti těchto procesů během dolování. Jednotný rámcový přístup k problémům umožňuje efektivnější aplikaci na nová zadání. Stejně tak je možné mít specialisty na konkrétní proces. Řešení zadaného problému se totiž zpravidla účastní několik lidí. Kromě již zmíněného zadavatele to jsou například:

- Specialisté na daný *manažerský problém*, kteří chápou motivace zadavatele
- Specialisté na přípravu dat, kteří jednak upravují data z často složitých struktur do struktur plochých, a navíc vědí, v jaké podobě jsou data potřeba pro jednotlivé algoritmy
- Specialisté na algoritmy a vytváření modelů
- Lidé, kteří data dosadí do kontextu

Existují metodiky proprietární pro daný software a existují metodiky navržené pro prostředí nezávislé na konkrétním software.

2.5 CRISP-DM

Právě příkladem metodiky nezávislé na software a prostředí, ve kterém se aplikuje, je *CRoss-Industry Standard Process for Data Mining* (CRISP-DM), jejíž vytvoření mělo za cíl navrhnutí univerzálního postupu použitelného v nejrůznějších komerčních aplikacích [4].

Řešení zadaného problému se pak dělí na několik částí.

Porozumění problematice

V této části se tým zabývá převodem formulace úlohy z manažerského pojetí do pojetí z oblasti získávání znalostí z databází. Probíhá zde dialog mezi specialisty z oblasti zadavatele, což může být například zdravotnictví, a mezi specialisty na data mining. Prvotní plány pro postup v dalších částech jsou vytvořeny zde, popřípadě se zde konzultují dosažené výsledky.

Porozumění datům

Tento proces se zabývá tím, jaká data jsou k dispozici. Posuzuje se jejich množství, kvalita a dělají se první odhady, jaká data by mohla být zajímavá pro řešení zadaného požadavku.

Příprava dat

Data ve své surové podobě jsou většinou uchovávána ve složitých strukturách. Taková podoba není vhodná pro navazující část, a tak se transformují do mnohem jednodušších struktur. Rovněž se zde již bere v potaz algoritmus plánovaný k použití, a tak se data navíc musí upravit s ohledem na schopnosti daného algoritmu. Některé algoritmy si například neumí poradit s prázdnými hodnotami v datech či s daty určitého typu.

Modelování

Vytváření modelů aplikací jednoho nebo více analytických algoritmů. Možnost je vytvářet modely s různými nastaveními.

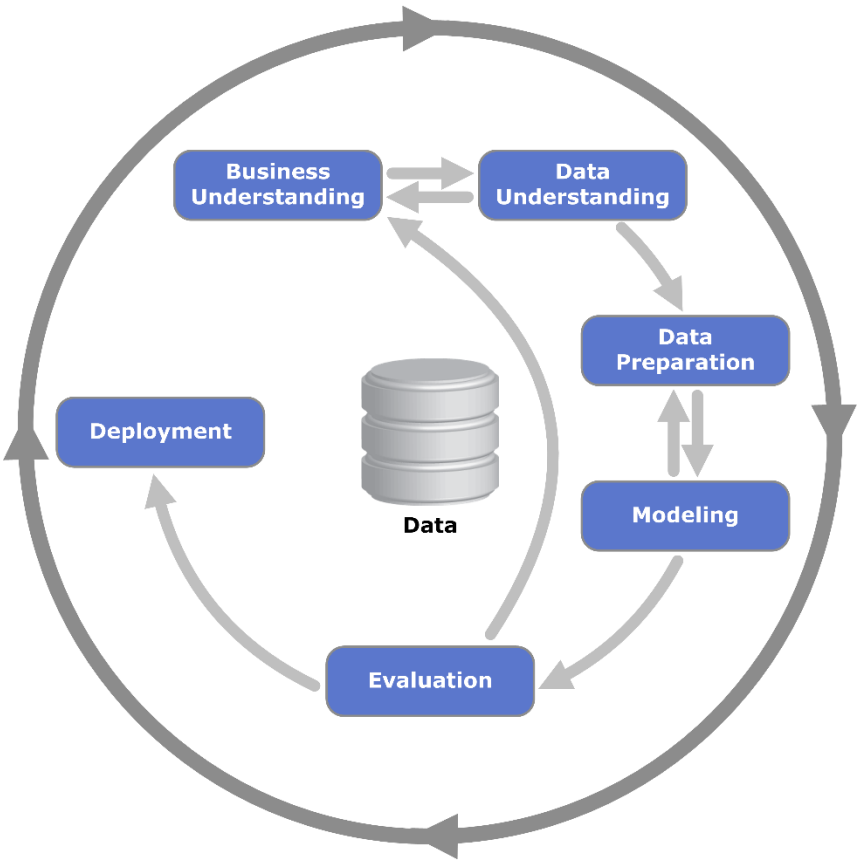
Vyhodnocení výsledků

Získání modelu ještě neznamená, že takový model přináší nějaké cenné informace. Proto je třeba daný model konzultovat s odborníky na doménu problematiky.

Využití výsledků

Jsou-li výsledky vyhodnoceny jako přínosné, zadavatel zareaguje na základě nalezených modelů.

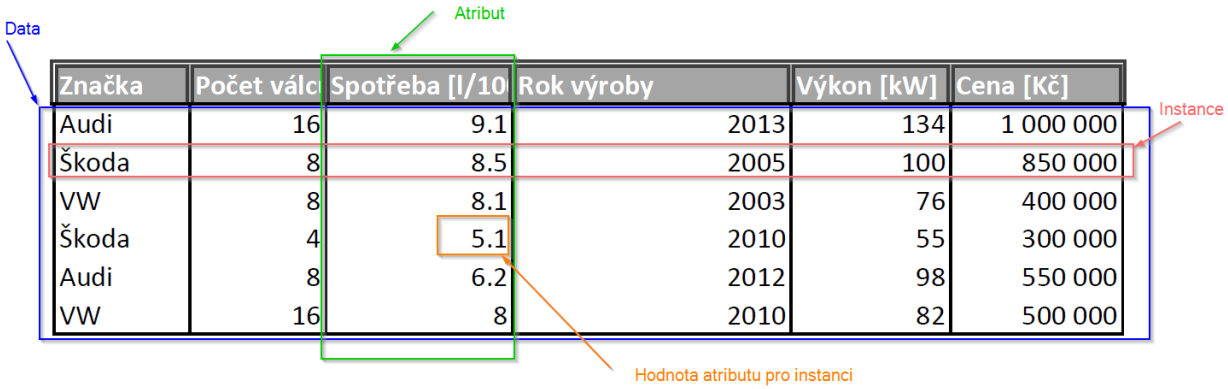
Zmíněné části na sebe navazují, avšak jde o iterativní proces, kdy na základě výsledků z části jedné upravujeme postup v části jiné. Toto je znázorněno na Ilustrace 1



Ilustrace 1: Posloupnost jednotlivých činností CRISP-DM (zdroj: en.wikipedia.org)

2.6 Základní pojmy oblasti data miningu

Vzhledem k tomu, že v následujících kapitolách již budu popisovat konkrétní algoritmy, zavedu zde terminologii pro popis dat, kterou hodlám po zbytek práce používat. Ilustrace 2 obsahuje velice jednoduchou a malou množinu dat, na které jsou vyznačeny ty naprosto základní pojmy, kterým se text věnovat nebude.



Značka	Počet válc	Spotřeba [l/10	Rok výroby	Výkon [kW]	Cena [Kč]
Audi	16	9.1	2013	134	1 000 000
Škoda	8	8.5	2005	100	850 000
VW	8	8.1	2003	76	400 000
Škoda	4	5.1	2010	55	300 000
Audi	8	6.2	2012	98	550 000
VW	16	8	2010	82	500 000

Ilustrace 2 Primitivní datová struktura

2.6.1 Typ atributu

1. **Kontinuální** – takový atribut nabývá hodnot z intervalu na číselné ose. Na Ilustrace 2 to je atribut *Cena*
2. **Kategorický** – jedná se o atribut, jehož hodnoty nejsou nijak porovnatelné a každá unikátní hodnota tvoří vlastní kategorii. Příkladem na Ilustrace 2 je atribut *Značka*
3. **Ordinální** – Podobně jako u kategorických typů tvoří každá unikátní hodnota vlastní kategorii s tím, že nyní je možné jednotlivé kategorie porovnávat. *Počet válců* je příkladem takového atributu na Ilustrace 2.

2.6.2 Role atributu

- **Vstupní atribut** – hodnoty tohoto atributu známe i pro nová data, ze kterých se model nevytvářel. Těmto atributům se také někdy říká nezávislá proměnná.
- **Výstupní atribut –** hodnoty těchto atributů známe pouze pro ta data, která slouží k vytvoření modelů. U nových dat se hodnoty snažíme určovat (predikce, klasifikace)

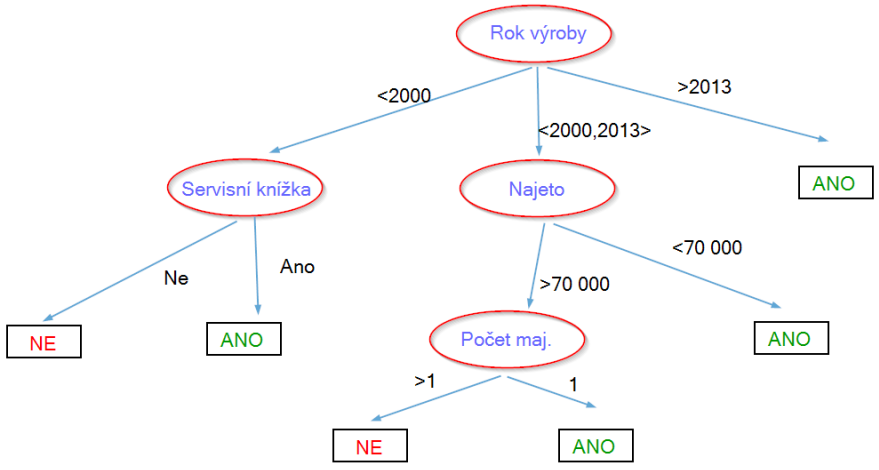
3 Rozhodovací stromy

Tato diplomová práce je zaměřená na implementaci algoritmu, který řeší klasifikační / predikční úlohy. Jedním ze způsobů, jak řešit tyto úlohy, jsou rozhodovací stromy.

Tato metoda je metodou učení s učitelem. Na počátku je tedy předložena trénovací matice dat, jejíž řádky tvoří jednotlivé instance a sloupce jsou tvořeny několika vstupními atributy a jedním výstupním atributem.

Obecný princip rozhodovacích stromů lze popsat následovně. Vstupní matice dat tvoří kořen stromu. V tomto kořeni se vyhledává takový vstupní atribut, který nejlépe dělí vstupní data vzhledem k výstupnímu atributu. Na základě tohoto atributu se kořen rozdělí zpravidla na takový počet uzlů, jaký je počet kategorií v daném atributu. Stejný princip se provádí i pro nově vytvořené uzly, dokud nenastane některá z podmínek ukončení růstu stromu. Listy stromu pak reprezentují výsledné zařazení nových případů. Příklad rozhodovacího stromu je zobrazen na Ilustrace 3:

Rok výroby	Počet majitelů	Servisní knížka	Najeto [tkm]	Dobrá prodejnost
Ordinální	Ordinální	Ano/Ne	Kontinuální	Ano/Ne



Ilustrace 3 Rozhodovací strom

Konkrétní algoritmy se pak liší především v těchto bodech:

1. Jakých typů může být výstupní atribut
2. Jakých typů mohou být jednotlivé vstupní atributy
3. Jak se určuje nejlepší štěpící atribut
4. Jaká jsou pravidla pro zastavení růstu stromu
5. Kolika n-ární stromy produkují

S prvním zmíněným bodem úzce souvisí následující základní dělení rozhodovacích stromů:

1. Klasifikační stromy
2. Regresní stromy

Klasifikační stromy předpokládají kategorický výstupní atribut a výsledkem je prohlášení, do jaké kategorie daný nově příchozí případ spadá. Při vyhodnocování vhodnosti atributu k štěpení se může využívat například Informační zisk či chí-kvadrát test.

Oproti tomu regresní stromy predikují hodnotu pro kontinuální výstupní atribut. Výběr vhodného štěpícího atributu je proveden například na základě zkoumání rozptylů výstupního atributu pro jednotlivé kategorie ve vstupních atributech. Výstupem pak může být konkrétní hodnota doplněná o možnou odchylku.

Výše zmíněné rozdělení se může stírat u takových algoritmů, které jsou uzpůsobené pro akceptování cílových atributů jak kategorického, tak spojitého typu. CHAID je toho příkladem a konkrétní řešení bude vysvětleno v jemu věnované části tohoto dokumentu.

V případě spojitých vstupních atributů mohou být hodnoty kategorizovány do určitého počtu kategorií, například tak, aby kategorie obsahovaly pokud možno stejný počet instancí. Výhodou rozhodovacích stromů je jejich snadná čitelnost, na rozdíl od jiných metod data miningu. Navíc strom lze snadno transformovat do rozhodovacích pravidel if – else.

Několik příkladů rozhodovacích stromů a jejich vlastností je uvedeno v Tabulce 1

Algoritmus	Binární strom	Typ vstupních atributů	Typ výstupních atributů
ID3	Ne	Kategorické	Kategorické
C&RT	Ano	Všechny	Všechny
CHAID	Ne	Všechny	Všechny

Tabulka 1 Přehled základních vlastností vybraných algoritmů

3.1 Vybraný rozhodovací strom – CHAID

Pro implementaci jsem si vybral algoritmus CHAID (Chi-squared Automatic Interaction Detector). Jeho návrh se datuje k roku 1980 a je připisován Gordonu V. Kassovi [5]. Ve své původní podobě uměl pracovat pouze s kategorickými atributy, avšak tento algoritmus doznával změn až do 90. let 20. století.

CHAID v dnešní podobě dokáže pracovat jak s kategorickými, tak reálnými hodnotami, a to jak v atributech vstupních, tak i výstupních. Z toho vyplývá, že CHAID je schopen vytvořit regresní i klasifikační stromy. Kategorie v jednotlivých atributech podléhají procesu slučování. To znamená, že dvě kategorie daného atributu jsou vyhodnoceny jako jedna, pokud splní jistá kritéria. Tyto dvě sloučené kategorie přitom již mohly vzniknout předcházejícím sloučením. Stromy, které jsou vytvářené, nemusí být binární, což znamená, že algoritmus není omezen nutností vyprodukovat pouhé dvě kategorie ze všech kategorií atributu.

Způsob, kterým CHAID rozhoduje o (ne)slučování kategorií daného atributu, je založen na aplikování statistických testů. Stejným způsobem se rozhoduje i o rozdělování uzlu do potomků na základě (ne)sloučených kategorií. V případě kontinuálního cílového atributu se jedná o F-Test a v případě kategorického cílového atributu se jedná o Pearsonův chí-kvadrát test či G-Test. Kritériem potom je p-hodnota, kterou ve spojení s těmito testy obdržíme. Testům, p-hodnotě a dalším pojmům z oblasti statistiky se v rámci potřeb pochopení tohoto algoritmu budu věnovat v nadcházejících kapitolách.

3.2 Modifikace Exhaustive CHAID

CHAID ve svém původním provedení skončí krok slučování kategorií v momentě, kdy p-hodnota nejlepší dvojce pro sloučení přesáhne mez, která určuje, zda kategorie *jsou podobné* vzhledem k výstupnímu atributu. Pokud existuje lepší rozdělení kategorií, které by vzniklo tím, že

by CHAID pokračoval ve slučování, tento přístup ho nenalezne. Právě tento nedostatek se snaží eliminovat modifikace algoritmu s názvem Exhaustive CHAID. Tato modifikace vznikla v roce 1991.

V Exhaustive CHAID slučování kategorií atributu pokračuje do té doby, dokud nezbývají pouze dvě kategorie. Následně se pro každý atribut vybere ta nejlepší sada kategorií. Její p-hodnota se modifikuje pomocí Bonferroniho korekce a porovná se stejně vzniklými p-hodnotami ostatních atributů. Ta *nejlepší* udává, podle kterého atributu se daný uzel rozštěpí. Počet potomků bude pochopitelně shodný s počtem nejlepší sady kategorií tohoto atributu.

Tato modifikace CHAIDu je výpočetně náročnější, a to právě proto, že slučování se pokaždé zastavuje, až když zbývají jen dvě dvojice. Použitě testy jsou shodné s původním CHAIDem. Stejně tak všechny ostatní vlastnosti. Detailní algoritmus bude představen ve vlastní kapitole poté, co uvedu statistické pojmy použité v předešlém textu.

4 Statistické přístupy použité v algoritmu

Myšlenka algoritmu CHAID spočívá ve využívání metod z oblasti testování hypotéz k tomu, aby rozhodnul, jak dané kategorie vstupních atributů určují hodnoty výstupního atributu. Uvedu zde základní pojmy z této oblasti a poté konkrétní testy použité v algoritmu CHAID.

4.1 Testování statistických hypotéz

Na počátku zformulujeme nějaké tvrzení, kterému budeme říkat **nulová hypotéza (H0)**. K němu sestavíme takzvanou **alternativní hypotézu (HA)**, která nulovou hypotézu popírá. Je-li nulová hypotéza formulována jako „Střední hodnota parametru pro danou populaci se rovná 100“, tedy $\mu = 100$, pak alternativní hypotéza může být formulována následujícími způsoby:

a. $\mu \neq 100$
b. $\mu > 100$
c. $\mu < 100$

V případě a) se jedná o oboustrannou alternativní hypotézu a ve zbylých případech se jedná o jednostrannou alternativní hypotézu. Úkolem testování statistických hypotéz je tedy buď zamítnout, nebo nezamítnout nulovou hypotézu. K tomu, abychom toto mohli učinit, potřebujeme testovací kritérium neboli statistický test. Je zpravidla nereálné pracovat s celou skutečnou populací při rozhodování o platnosti hypotézy. Proto se vždy pracuje s jistým výběrem populace, díky čemuž se během vyhodnocování testu můžeme dopustit dvou typů chyb, které jsou zobrazeny v Tabulka 2.

	Výsledek testu		
Skutečnost		**Nezamítnutí H0**	**Zamítnutí H0**
	Platí H0	Spolehlivost testu	Chyba I. typu
	Platí HA	Chyba II. typu	Síla testu

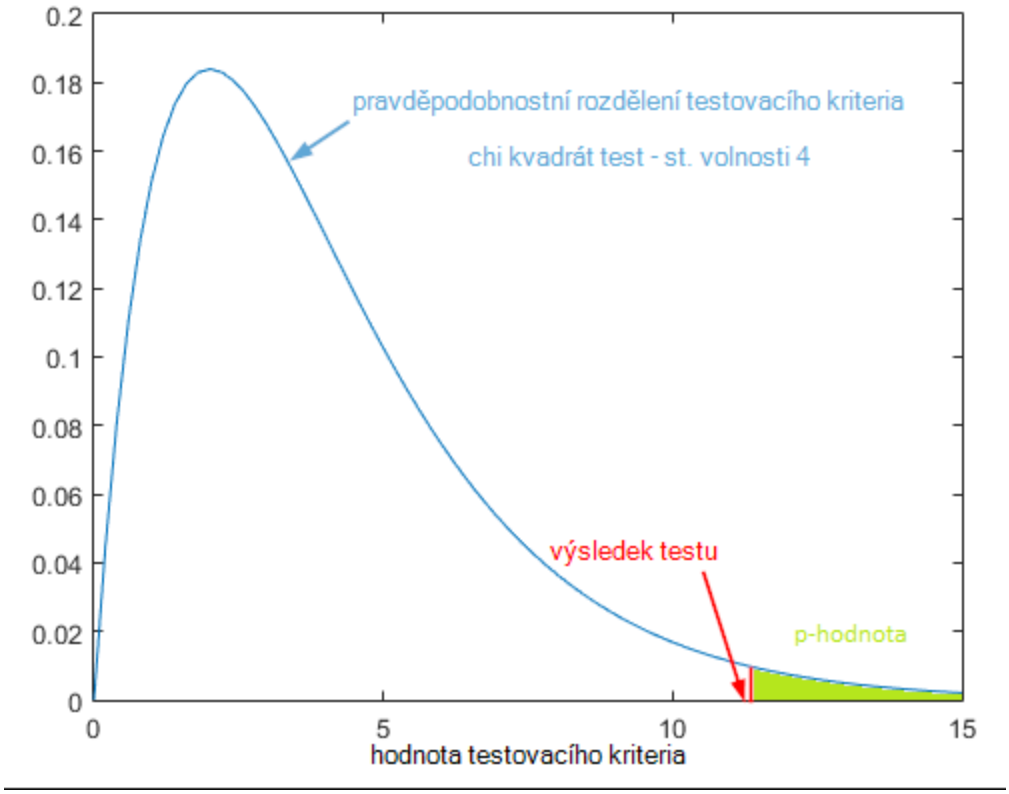
Tabulka 2 Varianty rozhodnutí pro testované hypotézy

Chybou typu I nazýváme situaci, kdy nulovou hypotézu zamítneme i přesto, že platí. Tato chyba také nese název **hladina významnosti**, jež značíme **α** a její hodnota udává, s jak velkou pravděpodobností se takové chyby dopustíme. **Chybou typu II** je pak situace, kdy nezamítneme nulovou hypotézu, v případě že je platná alternativní hypotéza. Značíme jí **β**.

Mezi správná rozhodnutí se pak řadí stav, kdy nezamítneme nulovou hypotézu v případě, že platí. Tomuto říkáme **spolehlivost testu** a je dána vztahem $1 - \alpha$. **Silou testu** je rozhodnutí o zamítnutí nulové hypotézy, pokud platí hypotéza alternativní, a zcela analogicky je vypočítána jako $1 - \beta$.

Samotný proces testování hypotéz může mít dva přístupy [6]. Jeden z nich je **klasický test**, kdy hladina významnosti je předem zvolena, a druhý z nich je **čistý test významnosti**, kdy na základě jeho výsledku můžeme rozhodnout, na jakých hladinách významnosti nulovou hypotézu (ne)zamítneme. Vzhledem k použití testů v algoritmu CHAID se budu věnovat pouze druhému typu.

Čistý test významnosti spočívá ve stanovení obou hypotéz, volbě testovacího kritéria (testu), výpočtu hodnoty tohoto kritéria a také **p-hodnoty**. P-hodnota udává, na jaké minimální hladině významnosti lze nulovou hypotézu zamítnout [7]. P-hodnota je v podstatě plocha pod křivkou, tvořenou pravděpodobnostním rozdělením daného testovacího kritéria a z jedné strany ohraničenou vypočítanou hodnotou testovacího kritéria. Viz Ilustrace 4:



Ilustrace 4 Demonstrace p-hodnoty

4.2 Statistické testy

Přestože existuje mnoho různých testů pro mnoho konkrétních aplikací, omezíme se pouze na ty testy, které vystupují v implementaci CHAID. Terminologie, použitá v popisech jednotlivých testů, bude v souladu s oblastí tématu této diplomové práce a použitým vzorcům bude dosazován význam z hlediska algoritmu CHAID.

4.2.1 Pearsonův chí kvadrát test (Test dobré shody)

Tento test se používá pro situace, ve kterých chceme rozhodnout, zda data pochází z určitého rozdělení. Konkrétně se dělí na dvě varianty:

a. Chceme ověřit, zda četnosti jednotlivých variant populace jsou rovny definovaným četnostem. Je tedy třeba rozdělit populaci do oněch variant podle nějakého znaku.
b. Chceme ověřit, zda populace má nějaké definované rozdělení. Například normální

Nás bude zajímat první varianta. V případě použití v algoritmu CHAID je znakem, podle kterého se dělí populace do variant, vybraný vstupní atribut. Vzhledem k tomu, že vstupní atributy jsou vždy kategorické, pak každou variantu zastupuje jedna kategorie. Chí-kvadrát testy je možné použít pouze v případě, kdy výstupní atribut je kategorický. Tento test se používá také při ověřování hypotéz v kontingenční tabulce, a právě touto formou ho budu dále demonstrovat.

Kontingenční tabulka bude tvořena kategoriemi z jednoho vstupního atributu a kategoriemi z výstupního atributu. Tabulka 3 je příkladem takto vytvořené struktury. Data jsou fiktivní, populace čítá 150 pozorování, za hodnoty vstupního atributu považujeme značky automobilů a za výstupní atribut potom hodnocení kvality jízdních vlastností.

Vlastnosti/Značka	**Škoda**	**Renault**
Dobré	30	20
Průměrné	50	20
Špatné	15	15

Tabulka 3 Kontingenční tabulka pro demonstraci chí kvadrát testu – empirické četnosti

Obsahem Tabulky 3 jsou empirické (skutečné) četnosti v dané populaci. K tomu, abychom mohli provést test, je třeba vypočítat očekávané četnosti jednotlivých variant.

Jejich výpočet vychází ze součinu příslušných marginálních relativních četností, a to proto, že testujeme nezávislost dvou náhodných veličin (nezávislost vstupního a výstupního atributu). Vztahy používané ve výpočtu očekávaných četností jsou uvedeny níže.

$$\widehat{m_{ij}} = \frac{n_{i.}\, n_{.j}}{n_{..}} \qquad (1)$$

$$n_{i.} = \sum_{j=1}^{J} n_{ij} \qquad (2)$$

$$n_{.j} = \sum_{i=1}^{I} n_{ij} \qquad (3)$$

$$n_{..} = \sum_{j=1}^{J}\sum_{i=1}^{I} n_{ij} \qquad (4)$$

Kde *I* je počet kategorií vstupního atributu, *J* je počet kategorií výstupního atributu a *i* a *j* jsou indexy jednotlivých kategorií. Aplikace vztahů (1), (2), (3) a (4) je obsahem Tabulka 4

Vlastnosti/Značka	**Škoda**	**Renault**	$n_{.j} = \sum_{i=1}^{I} n_{ij}$
Dobré	$\widehat{m_{00}} = \frac{95 * 50}{150} = 31{,}7$	$\widehat{m_{10}} = \frac{55 * 50}{150} = 18{,}3$	$n_{.0} = 50$
Průměrné	$\widehat{m_{01}} = \frac{95 * 70}{150} = 44{,}3$	$\widehat{m_{11}} = \frac{55 * 70}{150} = 25{,}7$	$n_{.1} = 70$
Špatné	$\widehat{m_{02}} = \frac{95 * 30}{150} = 19$	$\widehat{m_{12}} = \frac{55 * 30}{150} = 11$	$n_{.2} = 30$
$n_{i.} = \sum_{j=1}^{J} n_{ij}$	$n_{0.} = 95$	$n_{1.} = 55$	$n_{..} = \sum_{j=1}^{J}\sum_{i=1}^{I} n_{ij} = 150$

Tabulka 4 Kontingenční tabulka - očekávané četnosti

Označíme-li skutečné četnosti n_{ij}, pak výpočet hodnoty tohoto testovacího kritéria se řídí dle rovnice (5).

$$X^2 = \sum_{j=1}^{J}\sum_{i=1}^{I} \frac{(n_{ij} - \hat{m}_{ij})^2}{\hat{m}_{ij}} \qquad (5)$$

P-hodnotu získáme jakožto plochu pod křivkou v chí-kvadrát rozdělení se stupněm volnosti $(I-1)(J-1)$, z levé strany ohraničenou vypočtenou hodnotou X^2.

4.2.2 ANOVA F-Test

V případě, že výstupní atribut je kontinuálního typu, musíme použít jiný přístup k testování nezávislosti vstupního a výstupního atributu. Tímto přístupem je analýza rozptylu hodnot – ANOVA (ANalysis Of VAriance). Pro potřeby CHAIDu ve variantě, kdy nám stačí jeden faktor, tedy jeden vstupní atribut.

Myšlenka [8] spočívá v tom, že vybraná populace je rozdělena do určitého počtu skupin – počtu kategorií v daném vstupním atributu. Každá tato skupina má nějaký průměr / rozptyl vzhledem k hodnotám výstupního atributu. Opět se předpokládá, že výstupní atribut je nezávislý na vstupním atributu, a tak se očekávají malé rozdíly mezi odchylkami průměrů skupin od průměru celé populace. Konkrétní vztah pro vypočítání hodnoty testu je obsahem (6),

$$F = \frac{\sum_{i=1}^{I} \frac{(\bar{y}_i - \bar{y})^2}{I-1}}{\sum_{i=1}^{I}\sum_{n=1}^{N_i} \frac{(y_{in} - \bar{y}_i)^2}{(N-I)}} \tag{6}$$

kde I reprezentuje počet kategorií vstupního atributu, $\bar{y}_i$ je průměr hodnot výstupního atributu v i-té kategorii vstupního atributu, $\bar{y}$ je celkový průměr výstupního atributu, y_{in} je hodnota výstupního atributu pro n-tý prvek ze všech prvků i-té kategorie, N je celkový počet prvků a N_i je počet prvků i-té kategorie.

P-hodnotu získáme obdobně jako u chí-kvadrát testu s tím, že tentokrát používáme F-rozdělení o stupních volnosti $(I-1)$, $(N-I)$.

4.3 Využití p-hodnot v algoritmu

U obou testů jsem zmínil, že nulovou hypotézu tvoří předpoklad o nezávislosti vybraného vstupního atributu a výstupního atributu. P-hodnota, jež je výsledkem testu, svědčí o tom, na kolik je toto pravda pro vybrané kategorie daného atributu – obsahem testu totiž nemusí být všechny kategorie vybraného atributu.

Velikost p-hodnoty hraje roli při slučování kategorií atributu a následně při výběru atributu, podle kterého dojde k větvení stromu. Čím větší je p-hodnota, tím spíš by mělo dojít ke sloučení testovaných kategorií – kategorie a výstupní hodnota je nezávislá. Čím nižší je p-hodnota, tím spíš by mělo dojit k větvení stromu dle testovaného vstupního atributu – existuje zde závislost mezi kategorií a výstupní hodnotou.

5 Konkrétní popis implementovaného algoritmu

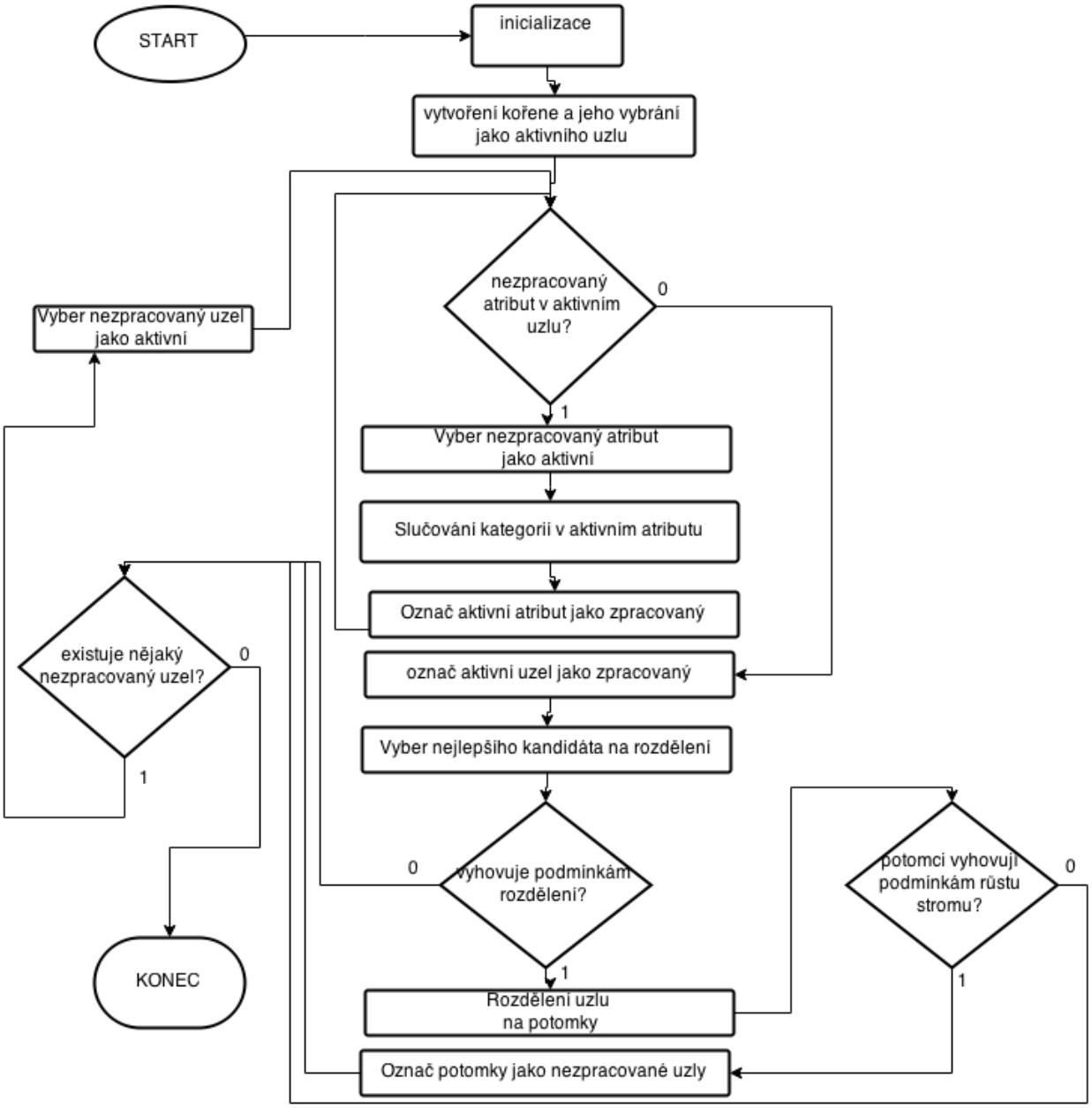
5.1 Parametry vstupující do implementace

- **Počet kategorií, do kterých se mají rozdělit kontinuální vstupní atributy**
  CHAID sice akceptuje kontinuální vstupní atributy, ale to jen proto, aby si je ještě před samotným během algoritmu rozdělil do určitého počtu skupin. Tento počet je jedním z uživatelsky definovaných parametrů.
- **Výběr chí-kvadrát testu**
  V této práci je popsán pouze Pearsonův chí-kvadrát test, nicméně implementovaná podoba umí navíc Likelihood-ratio Chi-squared test. Uživatel si může vybrat, který ze dvou zmíněných se má použít, je-li výstupní atribut kategoriální.
- **Hladina významnosti pro rozvětvení stromu**
  Uživatel má rovněž možnost nadefinovat si hladinu významnosti, kterou minimálně musí splňovat p-hodnota, aby mohl být daný uzel stromu dále větven.
- **Maximální hloubka stromu**
  V jistých případech by mohl růst stromu pokračovat do větší úrovně, než by uživatel potřeboval, proto je zde možnost zastavit růst na určité hloubce.
- **Minimální počet instancí v rodičovském uzlu**
  Algoritmu lze nastavit, aby nepokračoval s růstem v určitém uzlu, pokud tento uzel nemá více než definovaný počet instancí.
- **Minimální počet instancí v potomkovi**
  Pokud jeden z potomků by svým počtem klesl pod minimální počet instancí v uzlu, k větvení nedojde.
- **Definice skóre pro ordinální atributy**
  Je-li atribut ordinální, uživatel má možnost nadefinovat každé kategorii svoji váhu.

5.2 Algoritmus

5.2.1 Diagram

Ilustrace 5 zobrazuje diagram podoby algoritmu, ve které bude v následujících odstavcích prezentován. Text se bude vztahovat k této ilustraci. Pro jednoduchost je zde prezentován algoritmus v ryze synchronní variantě. V praktické části však byl implementován paralelně.



Ilustrace 5 Algoritmus CHAID

5.2.2 Popis algoritmu

1. Inicializace

Inicializací se zde myslí především transformace všech kontinuálních vstupních atributů do určitého počtu kategorií. Takový počet může být buď předdefinovaný, uživatelsky specifikovaný, nebo algoritmem vynucený. Vynucenost přichází v momentě, kdy není možné daný atribut kategorizovat do zamýšleného počtu skupin. Například pokud množství skupin je vyšší než počet unikátních hodnot, anebo pokud se jedna hodnota v datech vyskytuje vícekrát, než je podíl počtu všech instancí a počtu zamýšlených skupin. Další činností, jež lze v tomto kroku provádět, je validace vstupních dat. Například co se chybějících hodnot týče, správných hodnot pro kontinuální data, filtrace těch instancí, které nemají buď žádný vstupní atribut vyplněný, či jim chybí výstupní atribut.

2. Vytvoření kořene stromu a jeho vybrání jako aktivního uzlu

V tuto chvíli již všechna transformovaná a validovaná data utvoří počáteční uzel, ze kterého bude strom nadále růst.

3. Hledání nezpracovaného atributu a jeho případný výběr

Nachází-li se v aktivním uzlu nějaký doposud neslučovaný atribut, provede se jeho výběr a pokračuje se následujícím krokem. Jinak se pokračuje krokem 5.

4. Slučování kategorií v aktivním atributu aktivního uzlu

V první fázi tohoto kroku se nejprve hledá taková dvojice kategorií, která je sobě „nejvíce podobná“. K rozhodnutí o míry podobnosti se využívá již zmiňovaných statistických testů. V případě kategoriálního výstupního atributu je to jeden z chí-kvadrát testů a v případě kontinuálního výstupního atributu je to pak ANOVA F-Test.

Do testů tedy vstupuje pouze podmnožina instancí z daného uzlu, která je definovaná náležitostí instance do jedné ze dvou testovaných kategorií. Pro tuto dvojici je spočítána p-hodnota. Čím vyšší je výsledná p-hodnota, s o to větší pravděpodobností bychom se dopustili chyby, pokud bychom zamítli nulovou hypotézu neboli hypotézu, že výstupní atribut je nezávislý na vstupním atributu. Ze všech dvojic tak vybereme tu dvojici, která obdržela nejvyšší p-hodnotu jakožto výsledek testu.

Vzhledem k tomu, že se jedná o implementaci CHAID s modifikací EXHAUSTIVE, spočítáme p-hodnotu nového uskupení kategorií aktivního atributu. Tentokrát budou součástí příslušného statistického testu všechny kategorie. Tuto p-hodnotu si uchováme v paměti a pokračujeme ve slučování tohoto atributu až do doby, kdy zbývají pouze dvě kategorie, potencionálně vytvořené sloučením všech ostatních kategorií.

Nyní nastává chvíle, kdy pro aktivní atribut chceme vybrat takovou sadu kategorií, která je v jistém smyslu *nejlepší*. Kvalita sady kategorií je určena opět příslušnou p-hodnotou. Tentokrát nás však nezajímá maximální p-hodnota, nýbrž minimální. Je to pochopitelně proto, že v takové sadě kategorií, by měla být hodnota výstupního atributu pokud možno co nejlépe odhadnutelná z kategorie vstupního atributu stejné instance. Takovou nejlepší sadu kategorií a příslušnou p-hodnotu si zapamatujeme, atribut označíme jako zpracovaný a dále pokračujeme návratem do kroku 3.

5. Označení uzlu jako zpracovaného a vybrání nejlepšího kandidáta na rozdělení

Poté, co jsou všechny atributy v aktivním uzlu zpracovány, označí se aktivní uzel jako zpracovaný a začnou se porovnávat p-hodnoty nejlepších sad pro jednotlivé zpracované atributy. Ten atribut, který nabývá nejmenší p-hodnoty pro svou nejlepší sadu kategorií, je vybrán jako kandidát pro rozdělení.

6. Test, zda může být atribut použít k růstu stromu a případné rozdělení

Pokud vybraný kandidát pro růst stromu vyhovuje podmínkám, jako například minimálnímu počtu instancí v potomku, hladině významnosti pro rozdělení, z jednotlivých kategorií kandidáta se vytvoří další uzly a strom dále roste. Pokud ne, přeskakujeme do bodu 8.

7. Test vyhovění podmínkám dalšího růstu stromu v nových potomcích

Pokud i podmínky jako minimální počet instancí v rodičovském uzlu, maximální hloubka stromu, atd. jsou splněny, tak se nové uzly označí jako nezpracované.

8. Test existence nezpracovaného uzlu a jeho vybrání jako aktivního uzlu

V případě splnění tohoto testu se pokračuje krokem 3 s nově vybraným uzlem. Jinak algoritmus končí svojí činnost.

6 Návrh aplikace

6.1 Náplň praktické části

Součástí praktické části této diplomové práce bylo:

1. Implementovat algoritmus CHAID v modifikaci Exhaustive
2. Navrhnout systém, který provede uživatele algoritmem ve výukové formě zpracování
3. Zvolit technologie pro vývoj systému
4. Téma zpracovat jako součást univerzitního e-learningového portálu ALS

6.2 Klíčová hlediska při návrhu systému

Během návrhu a následné realizaci systému jsem se držel zásad, které jsem ve spolupráci s vedoucí práce stanovil již při zadání této práce.

- **Modularita**

Vzhledem k očekávané složitosti systému bylo již od počátku zřejmé, že nejlepší bude udělat systém modulární. Tímto jsem chtěl docílit možnosti, aby části výsledného systému byly v případě potřeby zaměnitelné, či aby je bylo možné použít v jiných, například navazujících, akademických pracích.

- **Testovatelnost**

Na právě zmíněnou modularitu navazuje požadavek mít možnost jednotlivé moduly testovat. Rozhodl jsem se, že tedy alespoň vybrané moduly, bude-li to možné a přínosné, pokryji takzvanými jednotkovými testy.

Jednotkové testy (unit testy) mají za úkol otestovat konkrétní části aplikace, například metody. Jednotkových testů v reálných aplikacích může být velmi mnoho a je to přímo úměrné rozsáhlosti aplikace. Naopak čím větší je rozsah systému, tím zásadnější je potřeba takovýchto testů, jelikož na rozsáhlých aplikacích zpravidla nepracuje pouze jeden člověk. Modifikací zdrojového kódu jiného vývojáře bychom mohli nevědomě porušit žádanou funkcionalitu. Jednotkový test by takový stav odhalil. Tyto testy mohou být, a z pravidla bývají, automatické a na sobě nezávislé.

- **Výukové zaměření aplikace**

Pravděpodobně nejdůležitějším aspektem vytvořeného systému je jeho výukovost. Výsledná aplikace má sloužit jako výuková pomůcka algoritmu CHAID na zdejší univerzitě. Doposud se k praktické ukázce tohoto algoritmu na naší škole využívá profesionální software IBM SPSS Modeler, který je cílen na komerční prostředí, respektive aplikace. Motivace v takovém prostředí je samozřejmě odstínit uživatele od detailů konkrétního algoritmu a poskytnout mu jakousi černou skříňku. Smyslem takové černé skříňky je potom danému uživateli poskytnout co nejlepší výsledek s minimem pro něj nepodstatných informací. Například pro manažera, jenž pomocí analýz v data miningu hledá odpovědi na to, jak zvýšit efektivitu svého podnikání, je průběh algoritmu jako takového nepodstatný. Pro výukové potřeby studentů IT je však naopak žádoucí poskytnout náhled do útrob algoritmu a jeho běhu.

- **Podpora pro starší operační systémy**

Posledním aspektem, na který byl brán zřetel již od počátku, byla podpora pro operační systémy, které se na univerzitě využívají. Ačkoli jsem se nevyhnul omezení pouze na operační systémy Windows, zvolené technologie alespoň umožnily provozovat aplikaci i na starších verzích, a to už od verze Windows XP.

6.3 Použité technologie

Již v předchozí sekci jsem předeslal, že zvolené technologie vedly k využití aplikace na operačních systémech Windows. Konkrétním nástrojem pro vývoj bylo *Microsoft Visual Studio 2013* a platforma *.Net framework*. Za účelem provozování aplikace na Windows XP jsem zvolil .Net framework ve verzi 4. Zvoleným programovacím jazykem byl C#. Během vývoje jsem použil technologie Windows Presentation Foundation (WPF), ve kterém jsem dodržoval návrhový vzor Model View ViewModel (MVVM).

6.3.1 Windows Presentation Foundation (WPF)

Windows Presentation Foundation je grafický subsystém, který přišel s .Net frameworkem ve verzi 3. Jedná se o přístup k programování desktopových aplikací zavedený společně s Windows Vista. Představuje alternativní styl vývoje aplikací k doposud využívané technologii WinForms s důrazem mimo jiné na grafické možnosti.

Grafické možnosti WPF spočívají například v rozsáhlých možnostech upravení vzhledu dobře známých vizuálních prvků, takzvaných Controls, pomocí vzhledových šablon. Další novinku WPF přineslo pomocí snadného definování animací přímo v kódu. Několik základních, jako například Fade-Out, Fade-In, změna pozic či velikostí, je již připravených a programátor je může okamžitě využívat.

Veškerá grafika, včetně zmíněných animací, je vykreslována a vypočítávána na grafické kartě, pomocí technologie DirectX. To je rozdíl oproti WinForms, kdy se k takovému vykreslování využíval procesor, konkrétně pak části User32 a GDI/GDI+ operačního systému Windows [9].

Ve WPF má programátor k vytvoření aplikace možnost použít XML Markup-u, respektive jeho rozšířené verze pojmenované XAML (Extensible Application Markup Language), ve které se popisuje grafické rozhraní aplikace. XAML si programátor může zvolit jako alternativu ke klasickému programování např. v C#. Dokonce je možné oba přístupy kombinovat.

Poslední zásadní vlastností, která bude zmíněna je takzvaný data binding, neboli přímé mapování atributů vizuálních prvků, nebo obsahu vizuálních prvků, na datové zdroje z aplikačního kódu. Díky tomu se programátor nemusí starat o aktualizace vizuálních komponent, pokud se změní obsah datových zdrojů. V případě oboustranného data bindingu se pak nemusí starat ani o aktualizaci datových zdrojů v případě uživatelské modifikace obsahu vizuálních prvků.

Data binding je možné aplikovat na celé datové struktury, neboli terminologií programování, na netriviální třídy. Ty mohou potencionálně obsahovat další netriviální třídy, což v kombinaci s použitím datových šablon v XAML markup-u umožňuje snadné vytvoření dynamických vizuálních datových *kontejnerů*. Jejich provázanost s konkrétními instancemi zbavuje programátora nutnosti psát nepodstatný kód z hlediska funkcionality aplikace.

6.3.2 Model View ViewModel (MVVM)

MVVM je návrhový vzor vzniklý pro WPF a zaměřený na uživatelské rozhraní. Jako mnoho jiných návrhových vzorů si klade za cíl segmentovat vývoj aplikace a umožnit dělení odpovědnosti mezi vývojáře. V tomto návrhovém vzoru by uživatelské rozhraní samo o sobě nemělo být zatíženo aplikační logikou, a aplikační logika by se neměla starat o vzhled aplikace.

Model

Model, stejně jako v případě jiných návrhových vzorů, představuje vrstvu zodpovědnou za data. Nijak se v ní nereflektuje stav aplikace a v podstatě o něm nepotřebuje nic vědět. Bývají v něm popsány datové struktury, se kterými pracuje ViewModel a jeho součástí je i aplikační logika.

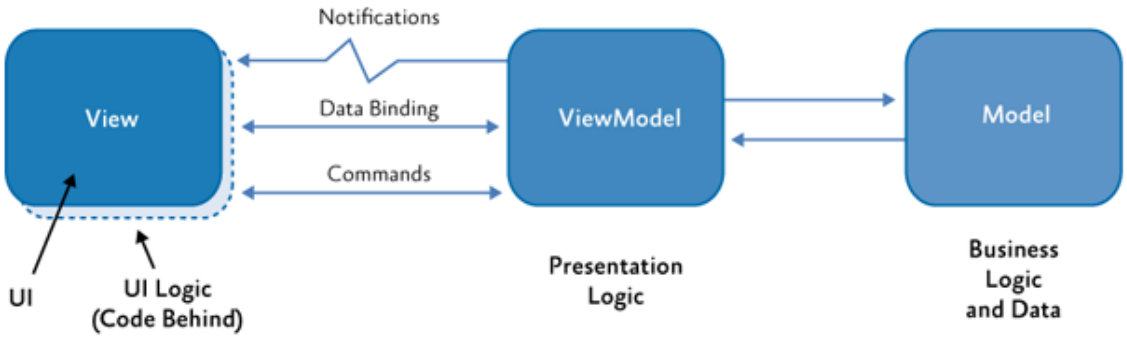
View

View, opět podobně jako u jiných návrhových zdrojů, je pro změnu odpovědné pouze za vzhled aplikace. Jediné, co potřebuje vědět, je jaké zdroje dat mu jsou vystavené z ViewModelu a jakou mají strukturu, aby na ně mohl případně mapovat (data bindovat) různé vizuální prvky. Uživatel komunikuje s ViewModelem buď pomocí data bindingu a nebo pomocí Command-ů, což je obdoba událostí z WinForms.

ViewModel

ViewModel je potom most mezi Modelem a View. Může Model transformovat do jiné podoby pro View, reaguje na Command-y zaslané z View, poskytuje zdroje dat pro View a odstiňuje prezentační logiku od View a od Modelu.

Schéma rozdělení rolí ve vzoru MVVM je zobrazeno na Ilustrace 6.



Ilustrace 6 Role v MVVM (zdroj: http://msdn.microsoft.com)

6.3.3 Použité externí knihovny

Alglib

První zmíněnou externí knihovnou je multiplatformní knihovna Alglib určená pro numerické analýzy a zpracování dat. Využití v aplikaci našla tak, že vypočítala p-hodnotu pro výsledek daného statistického testu.

MathNet

Ke stejnému účelu jako knihovna Alglib byla využita i knihovna MathNet, která je tímto druhou možností pro výpočet p-hodnoty.

Graph#

Knihovna Graph# poskytla mé aplikaci vizuální reprezentaci stromu, a to díky vlastní WPF Control. Navíc v ní bylo snadné upravit vzhled uzlů, takže o to uživatelsky přátelštější mohl být strom vytvořený algoritmem CHAID. Tato knihovna měla závislost na zahrnutí další knihovny, a to QuickGraph, která poskytuje zázemí pro strukturu stromu, neboli se v ní definují uzly a hrany.

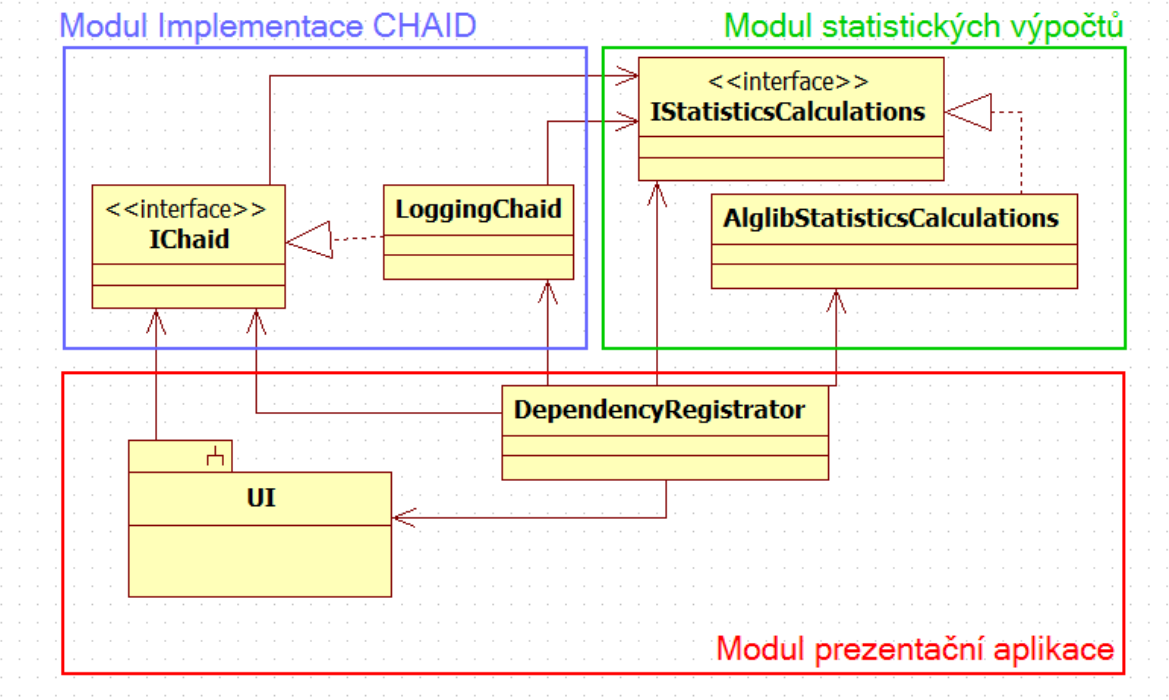
LumenWorks.Framework.IO CSV Parser

Vzhledem k tomu, že aplikace požaduje vstupní data ve formátu Comma Separated Values (CSV), použil jsem CSV Parser z namespace LumenWorks.IO.

NUnit

Pro zmíněné jednotkové testy jsem použil NUnit Framework, který původně vznikl z takzvaného *portu* implementace pro jazyk Java - JUnit frameworku do platformy .Net.

6.4 Moduly aplikace



Ilustrace 7 Moduly v systému

Vytvořený systém lze rozdělit do tří, respektive čtyř modulů. Čtvrtý modul, modul s rozhraními, však není součástí schématu zobrazeného na Ilustrace 7. Toto schéma je velice zjednodušeným UML diagramem tříd systému. Obsaženy v něm jsou pouze naprosto fundamentální třídy nejvyšší úrovně.

Modul s rozhraními

Tento modul je odkazován ze všech ostatních modulů, a to proto, že definuje rozhraní pro výměnu dat mezi moduly. Na schématu není zobrazen z důvodu přehlednosti.

Modul Implementace CHAID

Obsahem tohoto modulu je vlastní implementace algoritmu CHAID. Pracuje s modulem se statistickými výpočty, respektive s jeho rozhraním. Je používán modulem prezentační aplikace. Jeho úkolem je převzít data od prezentační aplikace a vytvořit strukturu obsahující rozhodovací strom.

Modul statistických výpočtů

Jediným modulem, který neodkazuje na žádné další moduly, je modul statistických výpočtů. Tento modul je používán implementací CHAID algoritmu. Jeho role je zpřístupňovat metody pro výpočet p-hodnoty na základě výsledku statistického testu. V mém konkrétním případě je výpočet jako takový prováděn jednou ze souvisejících externích knihoven, ale v principu lze tento výpočet implementovat vlastním způsobem.

Modul prezentační aplikace

Tento modul tvoří uživatelské rozhraní. Má dvě hlavní části.

První část, část se samotným UI, používá pouze jedno rozhraní, což je rozhraní CHAID algoritmu. Dále odkazuje na druhou část modulu, a to část nazvanou DependecyRegistrator.

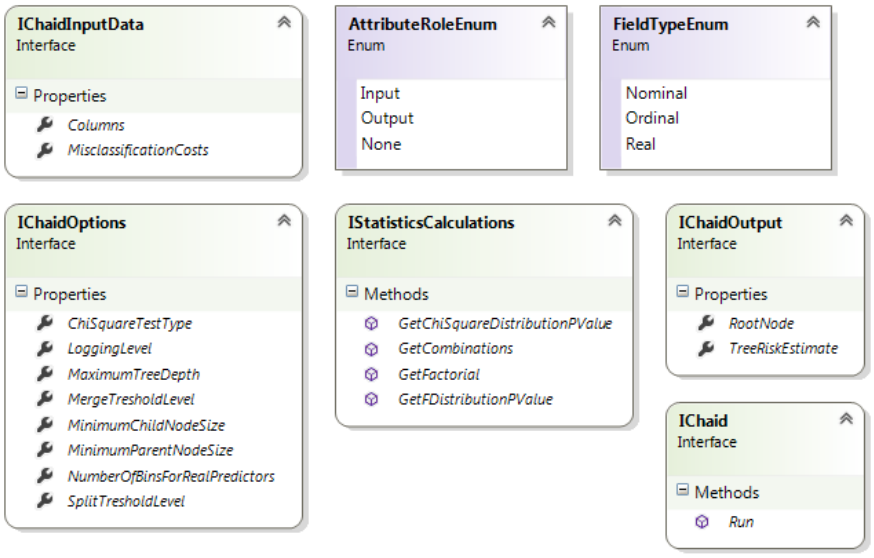
Jak je vidět ze schématu, DependecyRegistrator je jediná třída, která zná konkrétní implementace jednotlivých rozhraní. Jeho úkolem je tedy pomocí mechanismů Dependecy Injection (DI) poskytnout UI části již konkrétní instanci implementace CHAID, kterou vytvořil a které pro změnu poskytl konkrétní implementaci statistických výpočtů. Jedná se tedy o místo, kde se konfiguruje, jaké konkrétní implementace jednotlivých modulů se mají v aplikaci použít. Zde například můžeme rozhodnout, zda chceme využít pro statistické výpočty knihovnu Alglib nebo MathNet.

7 Realizovaná aplikace

7.1 Interfaces

V tomto modulu se nacházejí rozhraní pro třídy, které se používají či implementují i mimo jejich modul. Především se jedná o:

- Rozhraní pro implementaci CHAID, používané prezentační aplikací
- Rozhraní pro datovou strukturu, ve které se implementaci CHAID předávají vstupní data. Toto rozhraní je používáno v modulu s implementací CHAID a implementováno v modulu prezentační aplikace
- Rozhraní pro datovou strukturu, ve které je obsažen výsledek z algoritmu CHAID, tedy rozhodovací strom. Opačně k předchozímu rozhraní je toto rozhraní implementováno v implementaci CHAID a používáno v modulu prezentační aplikace.
- Rozhraní pro třídu se statistickými výpočty, které je využíváno pouze v modulu s implementací CHAIDu.
- Rozhraní pro třídu, která v sobě uchovává parametry pro vytváření rozhodovacího stromu, používané v implementaci algoritmu a implementované v prezentační aplikaci.
- Výčtové typy, které jsou sdíleny napříč moduly.



Ilustrace 8 Vybraná rozhraní

7.2 Statistické výpočty

Ve statistickém modulu je pouze jedna třída, která musí implementovat patřičné rozhraní. Zde mám na výběr ze dvou implementací. Implementace využívající knihovnu Alglib a implementaci využívající knihovnu MathNet. Rozhraní pro tento modul definuje signatury metod, kterou jiné moduly mohou používat. V obou implementacích taková metoda pouze zavolá metody daných externích knihoven a zprostředkuje výsledek. Jedná se tedy o návrhový vzor Adapter.

Samozřejmě lze vytvořit i vlastní implementace výpočtu p-hodnoty. Díky modularitě a *programování proti rozhraní* se z hlediska ostatních modulů vůbec nic nezmění.

Kromě zmíněných výpočtů p-hodnot k daným statistickým testům obsahuje tato třída i metody, které například počítají faktoriál či kombinace. Obě implementace jsou pokryty jednotkovými testy.

7.3 Implementace CHAID

7.3.1 Vybrané datové struktury

Struktura vstupních dat

Co se vstupních dat algoritmu týče, bylo řečeno, že existuje rozhraní pro data putující z prezentační aplikace do implementace algoritmu CHAID. Avšak záměrem tohoto rozhraní je především jeho jednoduchost a plochá struktura. Proto si má implementace interně transformuje data z definovaného rozhraní do vlastní datové struktury.

Struktura dat, se kterou se následně v algoritmu pracuje, má své specifikum v tom, že každá hodnota atributu je transformována do páru *hodnota:výstup*. Výstupem se v tomto případě myslí hodnota výstupního atributu pro danou instanci. Toto samozřejmě zavádí duplikaci dat, a to tak, že pokud má instance například 5 vstupních atributů, pak stejná hodnota výstupu se v transformovaných datech pro danou instanci objeví pětkrát. Na druhou stranu díky tomuto přístupu jsem ušetřil mnoho procházení hodnotami výstupního atributu.

Struktura použitá při běhu algoritmu

Vzhledem k tomu, že algoritmus CHAID vytváří strom, jako strukturu jsem se rozhodl použít návrhový vzor Composit, který spočívá v tom, že každá třída daného typu v sobě obsahuje kolekci tříd stejného typu.

Struktura výstupních dat

Po skončení algoritmu je třeba strukturu použitou při běhu algoritmu transformovat do jednodušší ploché struktury pro výstup. Výstup pak obsahuje v podstatě jen meta-data o daném uzlu a jeho potomcích. Takové zjednodušení zahrnuje například:

- Z kolekce vstupních atributů, jež obsahuje i jejich hodnoty, se vytvoří jednodušší kolekce obsahující pro daný atribut pouze jméno, seznam unikátních hodnot a četnosti těchto hodnot.
- Vypočítají se data, která pro běh algoritmu jako takový nejsou podstatná, ale pro uživatele přináší cenné informace, jako například:
    - Predikovaná hodnota pro daný uzel
    - Odhad riskantnosti klasifikace/predikce v daném uzlu, která vychází ze složení cílových atributů instancí daného uzlu
    - Podle jakého atributu pokračoval růst z tohoto uzlu
    - Jakou hodnotou jakého atributu byl tento uzel vytvořen z rodičovského uzlu

7.3.2 Způsob růstu stromu (běhu algoritmu)

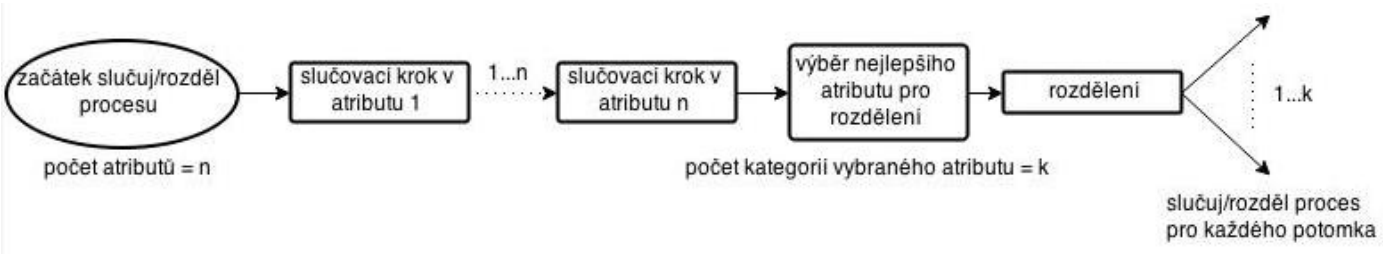
Již jsem zmínil, že struktura používaná pří běhu algoritmu měla návrhový vzor Composit. Na počátku samotného algoritmu byl tedy vždy jen jeden uzel – kořen stromu. Třídě reprezentující uzel jsem implementoval metody pro slučování kategorií a pro rozdělení uzlu na potomky. Poté, co obě metody vyprodukovaly výsledek, rozdělil jsem zpracovávaný uzel (z počátku jen kořen) na jeho potomky, tedy instance stejné třídy, a rekurzivně volal stejné metody nad každým potomkem, dokud nebylo vyhověno nějaké podmínce pro zastavení růstu stromu.

7.3.3 Optimalizace růstu stromu

Jednou z náplní praktické části bylo také optimalizovat algoritmy. Proto, po dokončení funkční verze implementace algoritmu, jsem začal tuto implementaci profilovat pomocí nástroje dotTrace od společnosti JetBrains. Během profilování jsem používal data, která měla 9000 instancí

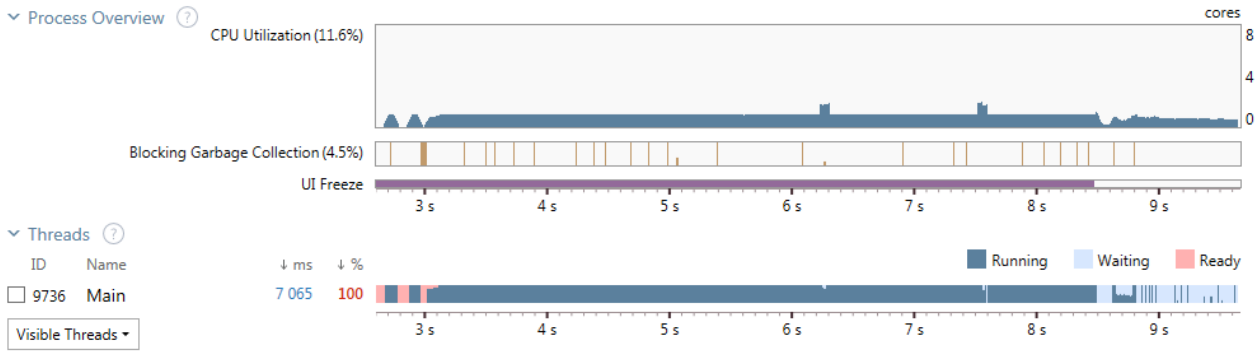
a zhruba 30 atributů. K profilování docházelo na počítači se čtyřmi jádry a hyperthreadingem, což dává dohromady osm logických vláken.

Nejprve byl algoritmus zcela sekvenční, bez zavedení paralelizace. Bez jakékoli optimalizace. Po profilování algoritmu jsem zjistil, že součet dob všech volání jedné konkrétní metody, která se stará o výběr nejlepšího páru kategorií ke sloučení, trvala 50 procent času běhu celého algoritmu. Díky tomu CHAID vytvářel strom 13 sekund. Po optimalizování této části jsem se dostal na 7 sekund pro stejnou úlohu. Diagram synchronního algoritmu je na Ilustrace 9.



Ilustrace 9 Sekvenční algoritmus

Ilustrace 10 zobrazuje využití systémových prostředků při sekvenční implementaci. Jak lze vidět, využito je pouze jedno vlákno a celkové využití procesoru je 11,6 procent.

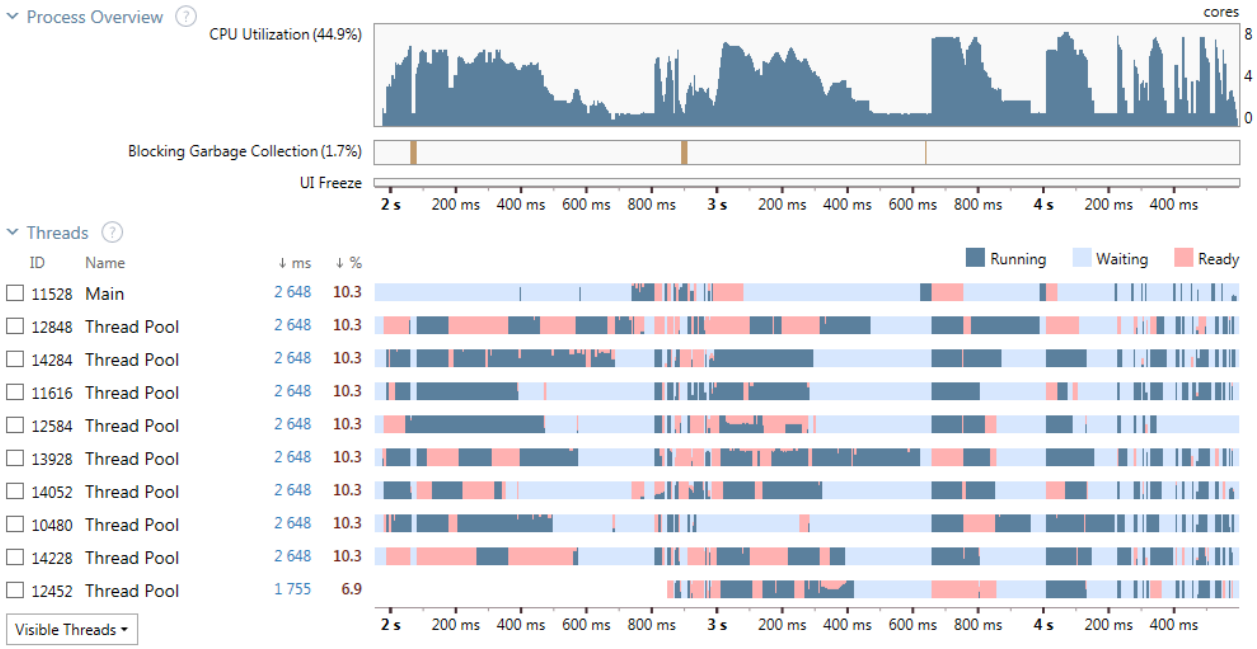


Ilustrace 10 Využití prostředků při sekvenční implem. (výstup z programu dotTrace)

Toto nízké využití systémových prostředků indikovalo, že je zde prostor pro optimalizaci zavedením paralelismu, jelikož charakter úlohy (vytváření stromu) paralelismus umožňuje. Paralelního zpracování jsem docílil pomocí třídy Task, která programátorovi nabízí vysokoúrovňové nástroje pro paralelizaci algoritmu. Po několika pokusech jsem jako ideální míru paralelismu zvolil stav, kdy se slučovací kroky provedou ve všech atributech naráz. Schéma takové implementace zobrazuje Ilustrace 11.

Ilustrace 11 Paralelní algoritmus

Díky této změně jsem dokázal potřebnou dobu zpracování zmíněné testovací úlohy snížit až na 4 sekundy. Na Ilustrace 12 je opět systémové využití při takto implementovaném algoritmu. Nyní zde již vidíme, že všech osm vláken bylo zaměstnáno během růstu stromu. Celkové vytížení procesoru se dostalo na 44,9 procent. Zároveň si lze všimnout momentů, kdy ze všech osmi vláken pracuje pouze jedno. To je způsobeno rozdílným počtem kategorií u jednotlivých atributů. Vlákno, kterému je přiřazen atribut s více kategoriemi, má pak více různých kombinací kategorií k otestování. Vzhledem k tomu, že k rozhodnutí o *nejlepším* atributu může dojít až poté, co je na všechny atributy aplikován slučovací krok, ostatní vlákna tedy musí čekat na to nejpomalejší. Poté program pokračuje do takzvané kritické (synchronizované) sekce zabývající se štěpením uzlu.



Ilustrace 12 Využití prostředků při paralelní implem. (výstup z programu dotTrace)

Tabulka 5 shrnuje výsledky dosažené během procesu optimalizace implementace algoritmu CHAID.

	Atributů	Instancí	Paralelizace	Čas [s]	Paměť [MB]
bez optim.	30	9000	Ne	13	30
s optim.	30	9000	Ne	7	30
s optim.	30	9000	Ano	4	30

Tabulka 5 Přehled dosažených výsledků optimalizace algoritmu

7.3.4 Záznam z běhu algoritmu

Jednou z vlastností, kterou jsem přidal mé implementaci algoritmu CHAID, byla podpora záznamů z běhu algoritmu. Toto bylo přidáno za účelem výukovosti aplikace, jednoho z cílů této diplomové práce.

Jedná se o seznam prováděných výpočtů, jejich parametrů a jejich výsledků během růstu stromu. Smyslem je, aby uživatel měl možnost vidět, proč výsledný strom vypadá právě tak, jak vypadá, proč v daném uzlu došlo k rozštěpení právě podle tohoto atributu, proč byla jako nejlepší sada kategorií vyhodnocena právě tato sada kategorií a tak dále. Seznam je vytvářen během vytváření stromu a zahrnut ve výstupní struktuře, aby mohl být zobrazen v prezentační aplikaci.

O tom, jak detailní tento seznam bude, rozhoduje jeden z parametrů, který uživatel má možnost zvolit. Existují tři úrovně záznamů:

1. **Basic** (základní)
    - Výsledek testování, zda se v uzlu má začít proces slučování kategorií
    - Vyhodnocení nejlepší sady kategorií pro každý atribut
    - Vyhodnocení nejlepšího atributu pro štěpení uzlu
    - Výsledek testování, zda lze uzel rozštěpit pomocí atributu, který byl vyhodnocen jako nejvhodnější pro štěpení.
    - Informace o rozštěpení uzlu
2. **Extended** (rozšířené)
    - Oznámení o aktuálně slučovaném atributu
    - Výsledek výpočtu p-hodnoty pro dosud neslučovaný atribut
    - Oznámení o počátku hledání nejlepšího páru v sadě kategorií
    - Nalezený nejlepší pár pro sloučení, včetně jeho p-hodnoty
    - Výsledek výpočtu p-hodnoty pro každou nově vzniklou sadu kategorií
3. **Detailed** (detailní)

- Výsledek výpočtu p-hodnoty pro každý slučovaný pár kategorií
- Informace o nalezení nové nejlepší sady kategorií pro daný atribut

7.4 Prezentační aplikace

Prezentační aplikace je ta část systému, která přichází do styku s uživatelem. Jejím úkolem je umožnit uživateli nahrát data, ze kterých se má vytvářet model, umožnit mu nastavit parametry pro vytvářený model, zobrazit mu výsledný rozhodovací strom a mnoho detailů, které se k němu vážou.

Navrhl jsem ji jako jakéhosi průvodce jednotlivými kroky tohoto procesu. Od zadání dat až po zobrazení vygenerovaného stromu. Struktura visuální části aplikace je ve stylu Master-Detail. To znamená, že hlavní (Master) okno v aplikaci je statické a obsahuje vždy jedno podřízené okno (detail), které reprezentuje aktuální krok, v němž se uživatel nachází.

Hlavní okno pak také zajišťuje navigaci mezi jednotlivými podřízenými okny pomocí navigačních tlačítek *Next* a *Previous*. Dále v sobě uchovávají data, která jsou pro všechna podřízená okna sdílená. Navíc je úkolem hlavního okna konstruovat okna podřízená a sdílená data jim předávat v jejich konstruktorech pomocí reference na instance sdílených dat. Zmíněné rozložení je obsahem Ilustrace 13.

Ilustrace 13 Rozložení prezentační aplikace

Kromě hlavních a podřízených oken jsou zde ještě *samostatná okna*. Jejich obsah zpravidla není součástí průvodce, takže v nich nelze využívat navigační tlačítka. Jedná se především o

pokročilá nastavení parametrů modelu, či například detailní informace o uzlech stromu. Důvodem vytvoření tohoto typu oken byla skutečnost, že jejich obsah je velice rozsáhlý a nebylo by možné udržet přehlednost aplikace, pokud by byla zakomponována do podřízených oken.

Vytvořená aplikace má následující kroky a s nimi spojená samostatná okna:

1. Krok načítání dat
2. Krok nastavení parametrů modelu
    - Okno s definicí skóre pro ordinální atributy
    - Okno s definicí postihů za špatnou klasifikaci výstupního atributu
3. Krok s vytvořenými stromy
    - Okno s detailem uzlu

Přechody mezi jednotlivými kroky jsou hlídány pomocí validačních pravidel pro aktuální krok.

7.4.1 Krok načítání dat

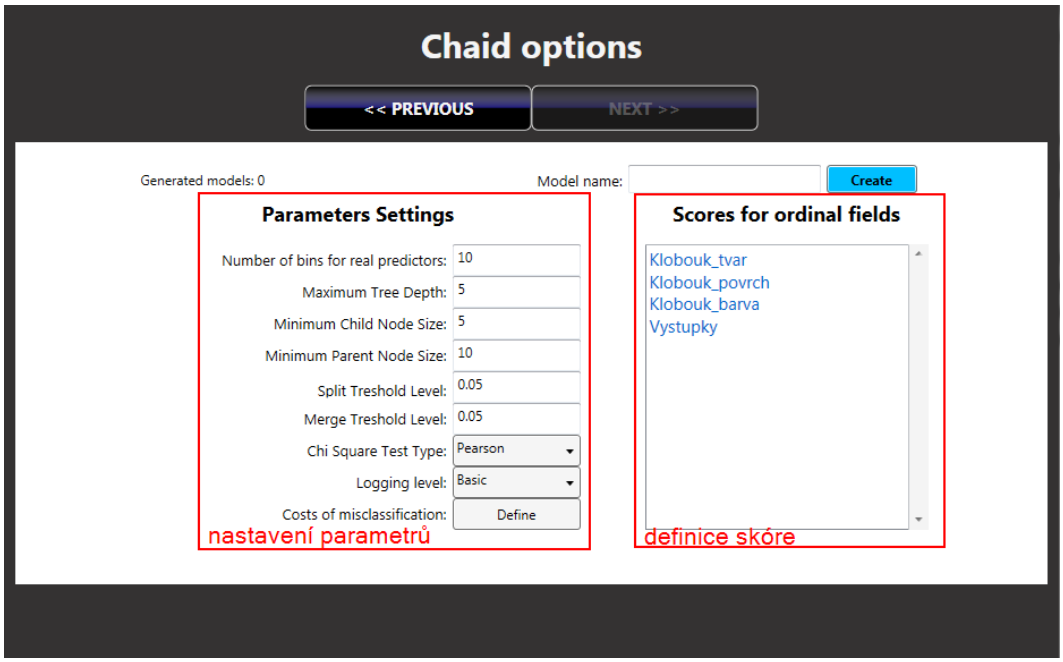
Jako první krok je uživateli představeno okno, ve kterém musí vybrat, z jakého souboru se mají načíst data. Jediný přijatelný formát souboru je formát comma separated values (CSV), u kterého se očekává přítomnost hlavičky. Poté, co se za pomoci externího CSV parseru zpracuje tento soubor, se data transformují do struktury, která je přijatelná jako vstupní parametr pro implementaci algoritmu CHAID.

V takto transformovaných datech uživatel zvolí typ a roli pro každý atribut. Pokud nastavení atributů vyhovuje validačním pravidlům tohoto kroku, je možné pokračovat dalším krokem. Validačními pravidly zde je kontrola, zda byl nahrán a úspěšně zpracován nějaký CSV soubor a zda v nastavení atributů je alespoň jeden vstupní a jeden výstupní atribut.

7.4.2 Krok nastavení parametrů modelu

Po nahrání dat a definici jednotlivých atributů má uživatel možnost změnit výchozí hodnoty nastavených parametrů pro generování stromu. Dále se zde definuje jméno modelu, který bude s těmito parametry vytvořen.

Kromě toho jsou mu zobrazeny všechny jím definované ordinální atributy, ke kterým má možnost pomocí samostatného okna definovat skóre. Rovněž samostatným oknem je řešen případ, pokud chce uživatel definovat také postihy za nesprávné klasifikace výstupního atributu, není-li výstupní atribut kontinuálního typu. Podřízené okno náležející tomuto kroku je zobrazeno na ilustraci 14



Ilustrace 14Krok nastavení parametrů modelu

Validačním pravidlem v tomto kroku je pouze skutečnost, zda byl nějaký model vytvořen. Pokud ano, pokračuje se krokem zobrazujícím vytvořený rozhodovací strom.

7.4.3 Krok s vytvořenými stromy

Okno pro tento krok lze rozdělit na čtyři části, které jsou vyznačeny na Ilustrace 15.

Rozhodovací strom

První částí je samotný rozhodovací strom. Pro vykreslení tohoto stromu jsem použil tak zvanou *User Control* z externí knihovny Graph# určenou speciálně pro WPF aplikace. Nicméně konkrétní vizuální reprezentaci jsem si upravil podle svých potřeb. Jako vhodnou reprezentaci zastoupení výstupního atributu v uzlu jsem zvolil koláčový diagram. Nad každým uzlem je zobrazeno, jakými kategoriemi rodičovského uzlu byl vytvořen. Pod ním je pro změnu název atributu, podle kterého se daný uzel dále štěpí.

Obarvení výstupních atributů

Každé kategorii výstupního atributu, je-li kategorický, je přiřazena náhodně vybraná barva, která se používá v koláčových diagramech napříč celým stromem. Takto přiřazené barvy jsou obsahem druhé části tohoto okna. Může se však stát, že náhodně vybrané barvy nejsou dostatečné

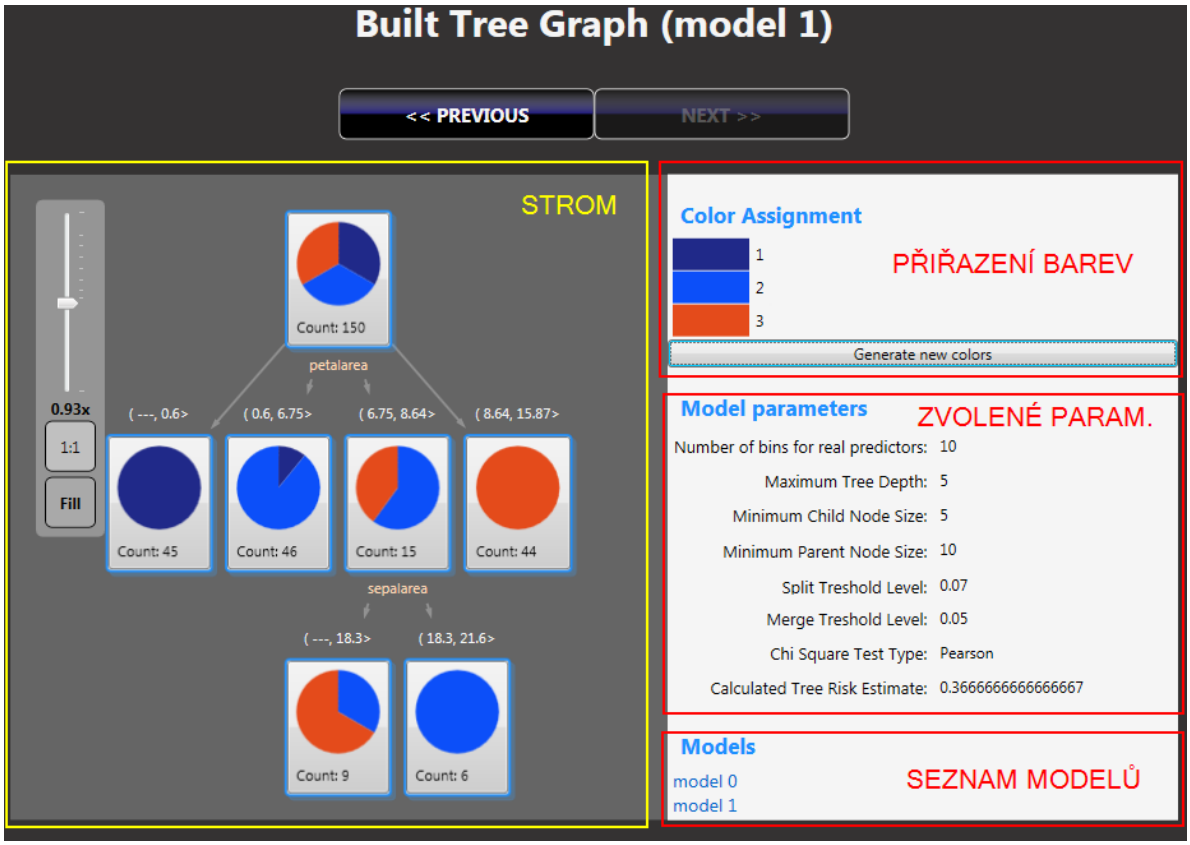
kontrastní, proto je uživateli umožněno tyto barvy „přegenerovat“ pomocí určeného tlačítka, dokud nenarazí na nějakou vhodnější kombinaci barev.

Volby parametrů pro zobrazený strom

Další částí je segment, kde jsou zobrazeny hodnoty parametrů, použitých k vytvoření zobrazeného stromu. Navíc se zde objevují některá data, která se vážou k celému stromu. Například odhad riskantnosti použití tohoto stromu ke klasifikaci.

Seznam vytvořených modelů a jejich stromů

Poslední částí ve vizuální reprezentaci tohoto kroku je část obsahující seznam všech vytvořených modelů pro daná data.



Ilustrace 15 Krok zobrazující vygenerovaný strom

Samostatné okno pro detaily uzlu

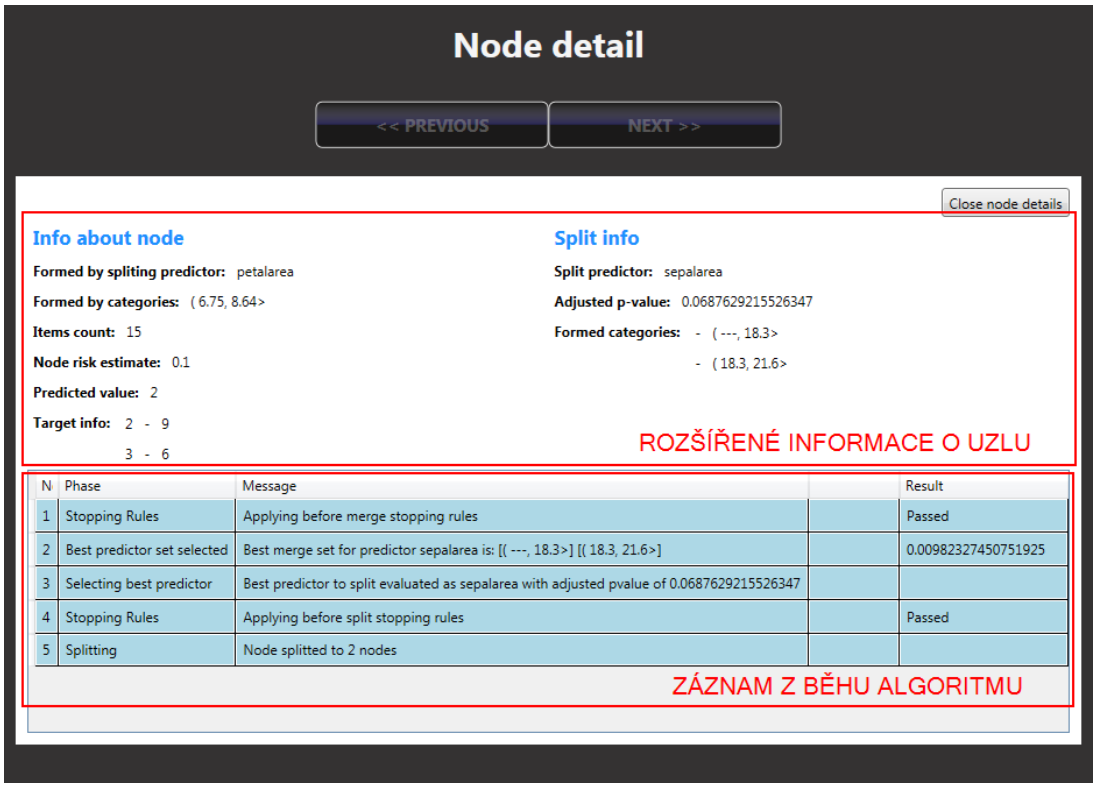
Pouhé zobrazení koláčového diagramu a počtu instancí v daném uzlu nemá dostačující informativní hodnotu pro uživatele, jehož cílem je pochopit algoritmus CHAID. Proto bylo

vytvořeno samostatné okno, zobrazené na Ilustrace 16, které uživateli poskytuje dva zásadní náhledy do daného uzlu.

Jedná se nejprve o náhled ve smyslu rozšířených informací ohledně uzlu. Takovými informacemi jsou například:

- odhad riskantnosti klasifikace ve vybraném uzlu
- nejlepší atribut a jeho sada kategorií, podle kterého se bude uzel dále štěpit
- p-hodnota sady kategorií, podle které má dojít ke štěpení
- predikovaná kategorie
- v případě kontinuálního výstupního atributu informace o jeho střední hodnotě a rozptylu pro obsažené instance

Druhým náhledem, který poskytuje toto zobrazení, je vizuální reprezentace záznamů z běhu algoritmu, o kterém již byla řeč v kapitole o implementaci algoritmu CHAID. Podrobně se touto reprezentací budu zabývat v následující části diplomové práce.



Ilustrace 16 Samostatné okno s informacemi o uzlu

7.4.4 Vlastnosti aplikace za účelem výkladu při studiu algoritmu CHAID

V cílech praktické části bylo několikrát zmíněno, že aplikace je určena především pro studenty. Proto i prezentační aplikace obsahuje několik vlastností, které mají právě studentům pomoci při seznamování se s algoritmem CHAID.

Záznam z běhu algoritmu

O této konkrétní vlastnosti již byla v této práci řeč několikrát, proto se zde budu nadále věnovat jen její vizualizaci, zobrazené na Ilustrace 17.

No	Phase	Message		Result
1	Stopping Rules	Applying before merge stopping rules		Passed
2	Merge Step	Starting merge step for predictor 'sepalarea'		
3	Category set evaluation	Calculating pvalue for non merged category set of predictor sepalarea		0.146055973243006
4	Finding best pair	Search for best pair in category set started	Expand	
12	Finding best pair	Best pair to merge is [(18.3, 19.5>] and [(19.5, 20.77>]		1
13	Category set evaluation	Calculating pvalue of merge set [(---, 13.64>] [(15.75, 15.75>] [(15.75, 16.8>] [(16.8, 17.64>		0.0936660810093714
14	Best predictor set replacing	New best set for predictor sepalarea found		

Ilustrace 17 Záznam z běhu algoritmu

Jednotlivé úrovně záznamů byly barevně odlišeny pro uživatelovu lepší orientaci. Základní záznamy jsou modré, pokročilé záznamy jsou šedé a detailní záznamy jsou žluté. Součástí detailních záznamů jsou i výsledky testování jednotlivých párů z fáze hledání nejlepšího páru pro sloučení. Přesně tyto záznamy mají pak v seznamu největší zastoupení. Uživatele však tato informace nemusí zajímat pro všechny atributy, proto jsem se rozhodl ve výchozím stavu toto testování skrýt. Pokud má uživatel zájem vidět takto podrobné informace pro nějaký atribut, lze je zobrazit.

Každý záznam má:

- Fázi, do které je zařazen
- Zprávu, která ho a jeho parametry charakterizuje
- Výsledek, pokud se jedná o záznam o výpočtu.

Podpora více modelů pro stejná data

Druhou zásadní vlastností prezentační aplikace za účelem výukovosti je umožnění uživateli, aby si nad stejnými daty vytvořil několik modelů. Každý model může mít jiná nastavení parametrů. Cílem je, aby měl lehce dostupné vizuální porovnání toho, jak změna daných parametrů

ovlivní vzhled vygenerovaného stromu. Z tohoto důvodu je v kroku s vygenerovaným stromem seznam vytvořených modelů a především informační část o parametrech zvolených pro zobrazený rozhodovací strom.

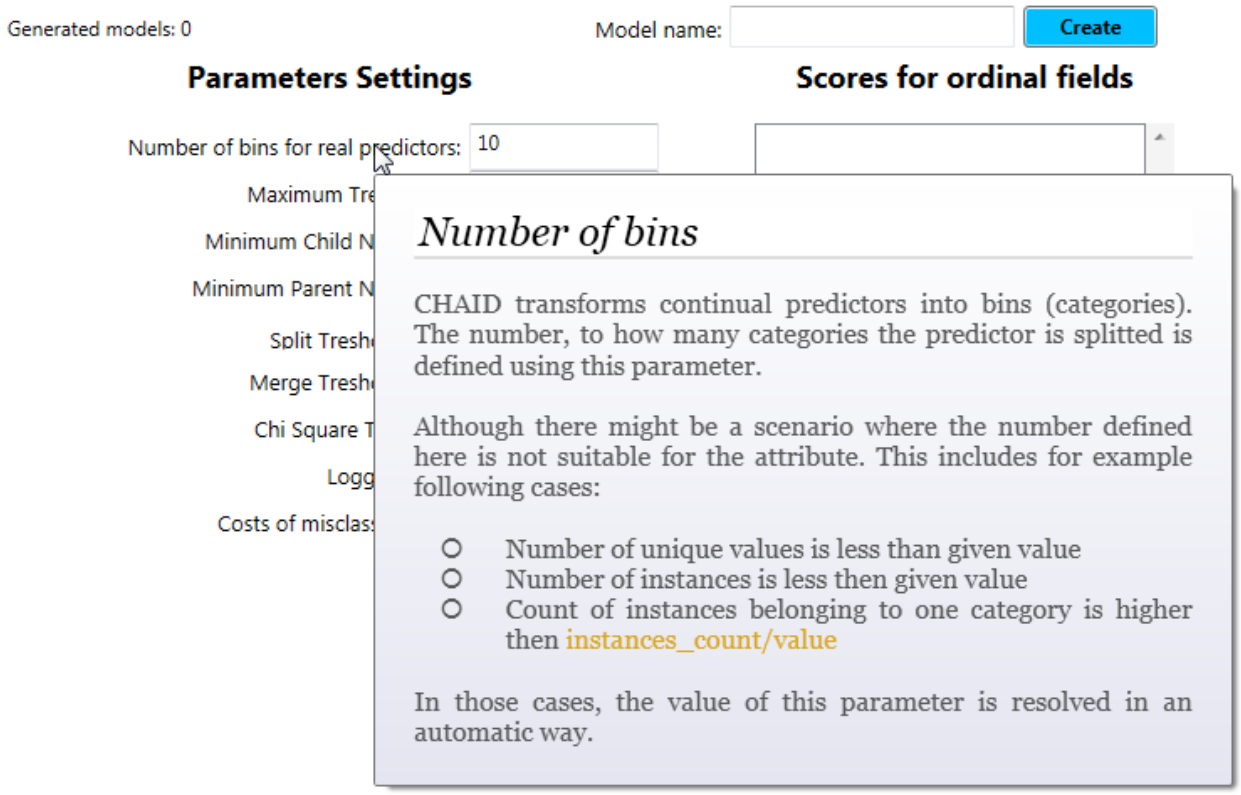
Tooltipy

Poslední vlastností, přidanou speciálně jako pomůcku studentům, jsou textově bohaté *tooltipy*. Tooltip je v běžných aplikacích malá a krátká nápověda, která se uživateli zobrazí, pokud najede kurzorem myši nad nějakou konkrétní oblast. Například na tlačítko, zaškrtávací pole, atd.

Díky WPF a jeho možnostem upravovat si téměř jakoukoli běžnou komponentu dle svých potřeb jsem změnil jednoduché a krátké textové políčko za informační pole, které za pomocí nástrojů pro formátování textu zobrazuje uživateli informace v graficky bohaté formě.

Tyto upravené tooltipy pak byly použity na kroku, ve kterém uživatel nastavuje parametry modelu, aby mohl lépe odhadovat, jak změna daného atributu ovlivní růst výsledného stromu.

Ukázka upraveného tooltipu je na Ilustrace 18.



Ilustrace 18 Upravený tooltip

8 Závěr

Nejrůznější aplikace, kde oblast souhrnně označovaná jako Big Data nachází uplatnění a její využívání velkými korporacemi jako je Google či Facebook, svědčí o novém trendu, který se může víc a víc promítat do každodenního života jedince. Ať už se jedná o cílenou reklamu na základě konverzací ze sociálních sítí, cílený marketing na základě historie nákupů v obchodech, či predikce výskytu epidemie chřipky z úvodu. Právě tyto široké možnosti, zvyšující se popularita a zvýšená poptávka pro analyticích z této oblasti je důvod toho, proč na univerzitách, jako je například TUL, vznikají předměty na tuto oblast orientované. To byl důvod vzniku této diplomové práce, jejíž vypracování je snahou podpořit výuku o oblastech analýzy dat.

Ve snaze pomoci studentům se studiem algoritmu CHAID jsem čerpal z vlastních zkušeností a snažil jsem se do programu zavést to, co bych si přál já v momentě, kdy jsem se seznamoval s náležející teorií. O tom, nakolik se toto osvědčí v praxi, rozhodne až první skutečné použití studenty.

Algoritmus CHAID se podařilo implementovat, a výsledky, ačkoli nebyly naprosto stejné s komerčním software, byly uspokojivé. Dokonce jsem měl prostor i pro optimalizaci algoritmu, a tak po optimalizaci a následné paralelizaci došlo k trojnásobnému zrychlení běhu algoritmu na testovací úloze.

Zavedená modularita a rozvržení systému, které jsem volil, může být jakousi předlohou nebo odrazovým můstkem pro to, jak podobné aplikace vytvářet. Oddělil jsem algoritmus jako takový od vrstvy, která se stará o jeho prezentaci uživateli. Díky tomu není problém, aby přišel někdo jiný a přidal další způsoby jak stejný, mnou implementovaný, algoritmus prezentovat.

Vytvořil jsem systém, který nemá ambice na to, aby přímo konkuroval profesionálním nástrojům, ale má za úkol být dobrou pomůckou pro studenta zabývajícího se vybraným algoritmem. Budoucnost této práce je rozhodně v možnosti ověřit v praxi její přínos pro studenty a adekvátně reagovat na zpětnou vazbu. Stejně tak lze díky modularitě přidat další způsob prezentace, a to například pomocí webové aplikace, což může být tématem jiné závěrečné práce.

9 Seznam použité literatury

[1] MAYER-SCHO¨NBERGER, Viktor a Kenneth CUKIER. *Big data: a revolution that will transform how we live, work, and think*. ISBN 9780544002692.

[2] HAND, David. 2001. *Principles of data mining*. Vyd. 1. Massachusetts: MIT Press, 546 s. ISBN 026208290x.

[3] PETR, Pavel. 2006. *Data Mining*. Vyd. 1. Pardubice: Univerzita Pardubice, 546 s. ISBN 80-719-4886-1.

[4] BERKA, Petr. 2003. *Dobývání znalostí z databází*. Vyd. 1. Praha: Academia, 366 s. ISBN 80-200-1062-9.

[5] AJAY, K.P. Soman; Shyam Diwakar; V. 2000. *Insight into Data Mining: Theory and Practice*. Vyd. 1. New Delhi: Prentice Hall of India, 366 s. ISBN 81-203-2897-3.

[6] HENDL, Jan. 2014. *Statistika v aplikacích*: *Theory and Practice*. Vyd. 1. Praha: Portál, 455 s. ISBN 978-802-6207-009.

[7] HENDL, Jan. 2014. *Přehled statistických metod zpracování dat*: *Theory and Practice*. 1. vyd. Praha: Portál, 455 s. ISBN 80-717-8820-1.

[8] PAVLÍK, Jiří. 2005. *Aplikovaná statistika: analýza a metaanalýza dat*. 1. vyd. Praha: Vysoká škola chemicko-technologická, 172 s. ISBN 80-708-0569-2.

[9] MACDONALD, Matthew. 2008. *Pro WPF in C# 2008: Windows presentation foundation with .NET 3.5*. 2nd ed. Berkeley, CA: Apress, 172 s. ISBN 978-159-0599-556.