Mendelova univerzita v Brně
Provozně ekonomická fakulta

Převod vybraných algoritmů data-mining z jazyka Java do binární (.exe) formy

Diplomová práce

Vedoucí práce:
Doc. Ing. Jan Žižka, CSc.

Bc. Jakub Šrom

Brno 2015

Tímto bych chtěl velice poděkovat Doc. Ing. Jan Žižka, CSc. za velkou ochotu, vstřícný postoj a cenné doporučení v různých situacích, které vedly k vypracování této práce.

Čestné prohlášení

Prohlašuji, že jsem tuto práci: **Převod vybraných algoritmů data-mining z jazyka Java do binární (.exe) formy**
vypracoval samostatně a veškeré použité prameny a informace jsou uvedeny v seznamu použité literatury. Souhlasím, aby moje práce byla zveřejněna v souladu s § 47b zákona č. 111/1998 Sb., o vysokých školách ve znění pozdějších předpisů, a v souladu s platnou *Směrnicí o zveřejňování vysokoškolských závěrečných prací.*

Jsem si vědom, že se na moji práci vztahuje zákon č. 121/2000 Sb., autorský zákon, a že Mendelova univerzita v Brně má právo na uzavření licenční smlouvy a užití této práce jako školního díla podle § 60 odst. 1 Autorského zákona.

Dále se zavazuji, že před sepsáním licenční smlouvy o využití díla jinou osobou (subjektem) si vyžádám písemné stanovisko univerzity o tom, že předmětná licenční smlouva není v rozporu s oprávněnými zájmy univerzity, a zavazuji se uhradit případný příspěvek na úhradu nákladů spojených se vznikem díla, a to až do jejich skutečné výše.

V Brně dne 18. května 2015 _____

Abstract

Šrom, J. Transfer of selected algorithms for data-mining from Java into binary executable (.exe) form. Thesis. Brno: Mendel University, 2015.

There are many successful systems for data-mining (eg. WEKA, RapidMiner, etc.), which currently hold many algorithms implemented in Java, which allows their use under different operating systems. The disadvantage of the interpreted source code is a slowdown in the calculation and limited memory usage. The thesis is focused on the transfer of several selected implementations of algorithms in Java binaries (.exe) through the conversion of source code in C ++ under MS Windows 7 64-bit. The aim is to speed up calculations and improve management of memory usage. Binary form must give identical results as the original form. In addition to the actual transfer, the thesis also includes comparing time and memory requirements of the original (using the Java Runtime Environment, JRE) interpreted implementation in Java (JRE 64-bit) and x64 resulting binary forms, for selected test data.

Keywords

Data mining, Java, C++, Weka, Machine learning, Bayes, Decision trees, IBL

Abstrakt

Šrom, J. Převod vybraných algoritmů data-mining z jazyka Java do binární (.exe) formy. Diplomová práce. Brno: Mendelova univerzita v Brně, 2015.

Existují úspěšné systémy pro data-mining (např. WEKA, RapidMiner, aj.) obsahující v současnosti desítky implementovaných algoritmů v jazyce Java, což umožňuje jejich použití pod různými operačními systémy. Nevýhodou interpretovaného zdrojového kódu je zpomalení výpočtu a limitované využití paměti. Diplomová práce je zaměřena na převod několika vybraných implementací algoritmů z Java do binární formy (.exe) prostřednictvím převodu zdrojového kódu do C++ pod operační systém MS Windows 7, 64bitová verze. Cílem je urychlení výpočtů a zlepšení správy využití paměti. Binární forma musí dávat identické výsledky jako forma originální. Kromě vlastního převodu zahrnuje práce také porovnání časových a paměťových nároků původní (pomocí Java Runtime Environment, JRE) interpretované implementace v jazyce Java (64bitové JRE) a výsledné binární 64bitové formy, a to pro zvolená testovací data.

Klíčová slova

Data mining, Java, C++, Weka, Strojové učení, Bayes, Rozhodovací stromy, IBL

Obsah

1 Úvod a cíl práce

1.1 Úvod

S rozvojem internetu a celkově oblasti informatiky se množství dat mnohonásobně zvětšuje každým dnem. Tento trend způsobuje řadu problémů. Prvním problémem je především velikost zabírané paměti. Tato překážka se dá vyřešit jednoduše zvětšením kapacit datových skladů. Samozřejmě to nejde dělat do nekonečna. Druhým problémem je, jak z tak velkého množství dat, získat nějakou informaci automatizovaným způsobem. Přece jen data nejsou informace. Tímto se právě zabývá disciplína dolování z dat (anglicky data mining). Metody dolování z dat jsou založeny především na metodách strojového učení (ML[1]), business intelligence (BI) a statistiky. Právě tyto oblasti odkrývají skryté informace v nepřeberném množství dat.

Dolování z dat se využívá především v řízení vztahu se zákazníky (Customer relationship management neboli CRM). Jednotlivé firmy mají ve svých databázích velké množství dat od zákazníků. Sbírají tyto data v různých anketách nebo při registraci ve věrnostním programu. Z těchto dat mohou získat důležité informace pomocí dolování a firma má možnost získat konkurenční výhodu.

Stávající řešení (systém Weka) tyto problémy řeší spíše obecně a hodí se pro vyzkoušení a pochopení práce jednotlivých algoritmů. Navíc systém Weka obsahuje většinu známých algoritmů a metod strojového učení. Je to takové komplexní řešení pro ML. Zajisté firmy toto řešení nevyužívají pro zlepšení konkurenční výhody. Ty se spíše zajímají o oblast business intelligence.

Tato práce se snaží odstranit některé nedostatky zaběhnutého řešení v oblasti dolování z dat a strojového učení. Především se snaží o zlepšení výpočetní složitosti a správy paměti. Práce se soustřeďuje na větší množství dolovaných dat.

Práci jsem si vybral záměrně s ohledem na řešenou problematiku. Dolování informací z velkých datových struktur je zajímavá oblast a myslím, že o ní v budoucnu ještě uslyšíme. Problémy, které limitují dolování z dat, jsou rovněž zajímavé a některé se tato práce snaží zvládnout.

1.2 Cíl práce

Primárním cílem práce je převést vybrané algoritmy data-mining z jazyka Java do binární formy. Tímto pak urychlit výpočty a zlepšit správu využití paměti.

Sekundárními cíli jsou výběr a popis vhodných algoritmů k převodu a také provedení experimentů mezi stávajícím systémem a aplikací vytvořenou pro potřeby práce. Experimenty budou provedeny na základě správnosti výsledků a poté porovnány z hlediska časové a prostorové složitosti u obou aplikací. Z těchto experimentů budou vyvozeny patřičné závěry a ty poté budou v závěru interpretovány.

[1] Machine learning – strojové učení.

Pro dosažení těchto cílů je třeba splnit celou řadu činností. Nejprve důkladně nastudovat stávající situaci (prostředí Weka), dále nastudovat podrobně vybrané algoritmy ve zdrojovém kódu i v obecné rovině a nakonec tyto algoritmy převést do jazyka C++ a ověřit správnost.

Hypotéza je taková, že po převodu do binární formy by měly jednotlivé algoritmy vykazovat podobné výsledky jako v případě Weky, časová a prostorová náročnost jednotlivých algoritmů by se měla zlepšit. Tato hypotéza vychází z předpokladů architektury jednotlivých programovacích jazyků.

2 Metodika

Výstupem práce je vytvořená aplikace, která implementuje vybrané algoritmy a především experimenty mezi touto aplikací a stávajícím řešením z univerzity v Austrálii (Weka). Je nutné nastudovat a popsat celkové prostředí stávajícího řešení a také zdrojové kódy vybraných algoritmů (teoretická část práce). Poté implementovat tyto algoritmy v jazyce C++ a vytvořit aplikaci. Dále následuje vytvoření grafického uživatelského rozhraní (GUI) a otestování celkové funkčnosti aplikace. Nakonec provést experimenty z hlediska správnosti výsledků, časové a prostorové náročnosti mezi zaběhnutým řešením a vytvořenou aplikací (praktická část).

Postup práce by se dal shrnout na základě těchto bodů:

- Vybrat a představit vhodné algoritmy pro převod.
- Seznámení se s prostředím Weka a zdrojovými kódy vybraných algoritmů.
- Převod jednotlivých zdrojových kódu z jazyka Java do jazyka C++ (binární forma).
- Vytvoření grafického uživatelského rozhraní (GUI) pomocí vhodného nástroje.
- Experimenty mezi vytvořenou aplikací a stávajícím řešením Weka.
- Zhodnocení a popis výsledků.

3 Vybrané algoritmy

V první řadě je důležité seznámit čtenáře se základy strojového učení a dolování z dat. Právě tyto témata jsou z velké části náplní práce.

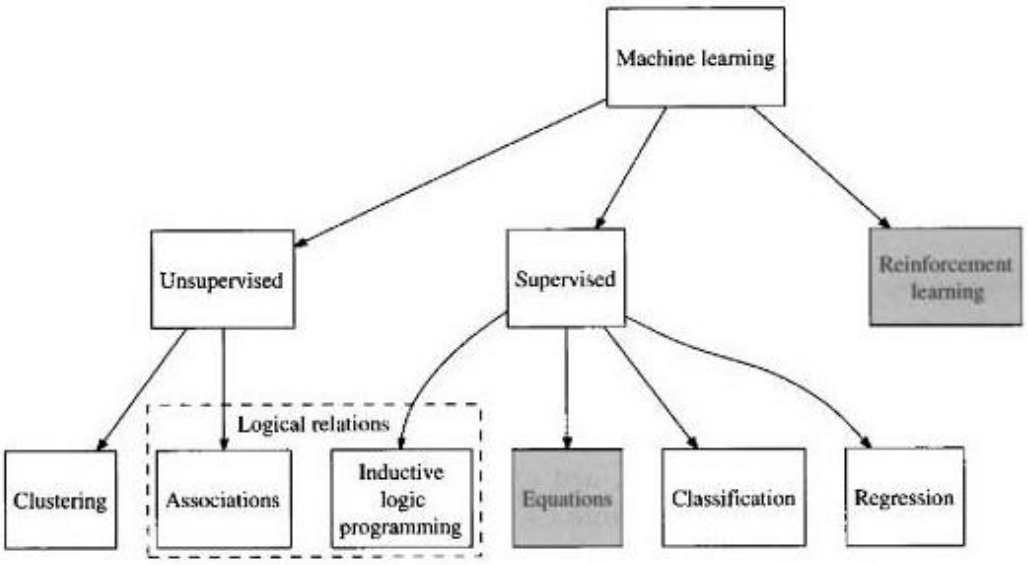
Dále je třeba představit, na jakých základech budou jednotlivé algoritmy vybrány. Po výběru vhodných algoritmů pro převod následuje představení těchto algoritmů a vysvětlení principů, na kterých jsou založeny.

3.1 Strojové učení

Strojové učení je součást umělé inteligence, která se zabývá algoritmy a technikami, které umožnují počítači, aby se učil. To znamená schopnost počítače při další změně systému být efektivnější.

Jako první zmínil pojem strojové učení Arthur Samuel a to již v roce 1959 při programu hraní šachu. Definoval ji jako: Oblast studie, která dává počítači schopnost se učit, aniž by byl explicitně naprogramován. Aby po každé šachové partii počítač zahrál další šachovou partii lépe. (NG, 2015)

Jak se strojové učení dělí a jaké jsou jeho metody je vidět na obrázku:



Obrázek 1 Strojové učení (Kononenko a Kukar, 2007, s. 6)

Rozdíl mezi učením bez učitele (unsupervised) a mezi učením s učitelem (supervised) je v tom, že s učitelem je určen výstup. Všechny algoritmy v této práci budou spadat do kategorie učení s učitelem a také do třídy klasifikačních algoritmů. Proto si klasifikaci blíže popíšeme.

3.1.1 Klasifikace

Metody strojového učení se často používají právě pro klasifikaci.

Klasifikace slouží především k vytvoření klasifikátorů. Tyto klasifikátory mají za úkol přiřadit třídu objektu, na který se dotazuje. Typické klasifikační úlohy jsou například lékařské diagnózy, předpovědi počasí aj.

Klasifikátor může být dodán přímo nebo může být naučen z dat. Tyto data obsahují příklady (instance) a popisují dříve zaznamenané problémy.

Do této skupiny algoritmu patří rozhodovací stromy, klasifikátory založené na bayesovu teorému, klasifikátory založené na algoritmu nejbližších sousedů a v poslední řadě také neuronové sítě. (Kononenko a Kukar, 2007, s. 5-14)

3.1.2 Ostatní metody strojového učení

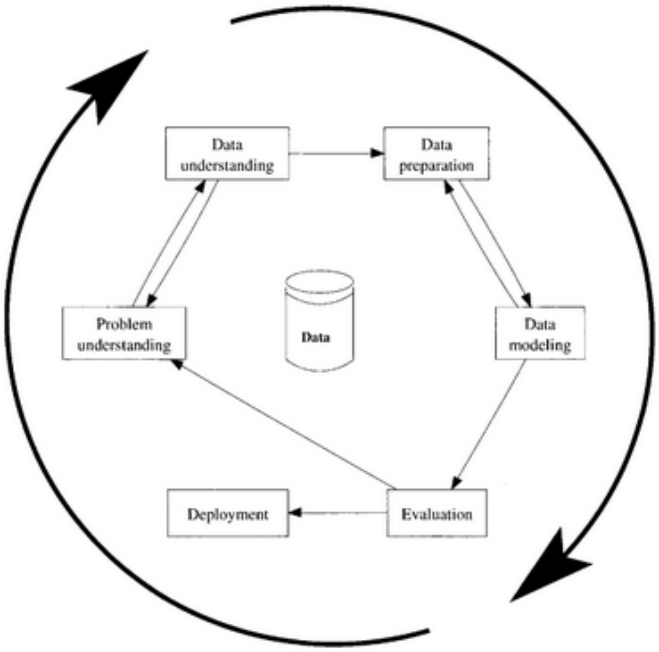
Další metody budou zmíněny pouze v krátkosti, jelikož nejsou náplní práce.

- **Regresní metody** – ty jsou podobné klasifikačním metodám. Patří do nich regresní stromy, lineární regrese aj.
- **Shlukování** – nejpoužívanější metoda učení bez učitele. V datech se algoritmus snaží najít shluky založené na podobnosti.
- **Pravidla**
- **Logické vztahy**
- **Zesílené učení** (Kononenko a Kukar, 2007, s. 5-14)

3.2 Dolování z dat

Dolování z dat (data mining) je součástí strojového učení. Využívá strojové učení z praktického hlediska. Strojové učení je spíše obecné a dá se použít ve více oblastech, kdežto dolování z dat se využívá především v databázích a informatice. Dolování z dat využívá metody strojového učení k dosažení požadovaných cílů a zabývá se především hledáním a popisováním skrytých struktur v datech. Je to také nástroj pro porozumění těmto datům a vytváření predikcí. (Shmueli, Patel a Bruce, 2007, s. 1-3)

Standard pro dolování z dat CRISP-DM definuje procesní model, který odráží životní cyklus projektu. Tento cyklus je možné vidět na obrázku 2. Obrázek ukazuje jednotlivé fáze projektu a také vztahy mezi nimi. Tyto fáze se opakují stále dokola, než je nalezen požadovaný cíl. (Kononenko a Kukar, 2007, s. 3)



Obrázek 2 Proces dolování dat (Kononenko a Kukar, 2007, s. 3)

Data jsou ve formě sady příkladů (např.: jaké kontaktní čočky lidé nosí). Výstup je ve formě predikce nových příkladů. Například, za určitých okolností predikuje, které kontaktní čočky bude zákazník potřebovat.

Obecný proces je velice jednoduchý. Jako první jsou zde tzv. raw data (to jsou data, která nejsou nijak upravená), tyto data se upraví a připraví tak, aby jim počítač porozuměl, a následně některou metodou počítač predikuje výstup. V této práci především metodou klasifikace.

Využití dolování z dat je obrovské. Data jsou všude kolem nás a je potřeba v nich hledat skryté struktury a z nich vyvodit patřičné závěry. V budoucnu se oblast dolování z dat a celkově umělé inteligence určitě bude rozvíjet a je důležité, aby se této oblasti věnovala značná pozornost.

3.3 Výběr algoritmů

Jak bylo dříve zmíněno, převáděny v této práci budou pouze algoritmy ze třídy klasifikačních algoritmů. To bylo vybráno proto, že jsou to algoritmy známé (často používané) a také relativně lehce implementovatelné.

Jako další je důležité zmínit celkovou použitelnost algoritmu pro rozsáhlé datové soubory. Rozsáhlými datovými soubory se v tomto případě myslí soubory o velikosti stovek až tisíců instancí a desítek atributů. To je často obtížně splnitelné. Většina algoritmů umí teoreticky pracovat s rozsáhlými soubory, ale limituje je

paměť a výkon výpočetních zdrojů. Později v experimentální fázi práce bude ukázáno, jak toto zvládá systém Weka a jak tento problém zvládá aplikace v C++[2].

Dále byly vybrány takové algoritmy, které mají dobrou dokumentaci. Ať již knižní nebo elektronickou. Všechny algoritmy v této práci lze snadno dohledat v různých zdrojích. Většina algoritmů má dobré vysvětlení i ve formě videa na youtube, což je určitě přínosné. V neposlední řadě jsou relativně jednoduše implementovány v systému Weka v jazyce Java.

Nakonec bylo vybráno pět algoritmů, kterými jsou: „Naive Bayes“, poté dva algoritmy založené na principu učení založené na instancích „IB1“, „IBk“ a nakonec dva rozhodovací stromy „ID3 Decision Tree“ a „Random Forest“. Navíc je zde i algoritmus „Random Tree“, který vzniká pokud je počet stromů v Random Forest roven jedné. Tyto algoritmy budou implementovány v rámci práce a budou přepsány do programovacího jazyka C++. Všechny tyto algoritmy jsou dobře známy, je pro ně dostatek informací na internetu i v knihách a jsou implementovány v různých systémech. Především jsou to také základní algoritmy jednotlivých tříd klasifikačních algoritmů.

Nyní jsou algoritmy vybrány a v dalších podkapitolách se s jejich principy a metodami seznámíme blíže. Nejprve bude popsán algoritmus z obecného hlediska a poté si ukážeme vždy jednoduchý příklad, který se skládá z klientů, kteří v minulosti žádali o úvěr a na základě různých charakteristik úvěr dostali nebo nedostali. Na příkladu bude jednodušší pochopit princip danému algoritmu.

3.4 Naive Bayes

Tento klasifikační algoritmus je založen na bayesovu teorému, který vychází z teorie podmíněné pravděpodobnosti.

Bayesův teorém pro výpočet podmíněné pravděpodobnosti, že platí hypotéza H za předpokladu E, vypadá následovně:

$$P(H|E) = \frac{P(E|H) \times P(H)}{P(E)}$$

Vzorec 1 Bayesův teorém

Apriorní pravděpodobnost hypotézy $P(H)$ odpovídá pravděpodobnosti zastoupení jednotlivých hypotéz (tříd), bez ohledu na další informace. $P(E)$ vyjadřuje pravděpodobnost evidence (atributu, který pozorujeme). Podmíněná (aposteriorní) pravděpodobnost $P(H|E)$ vyjadřuje, jak se změní pravděpodobnost hypotézy H, nastane-li jev E.

Klasifikátor Naive Bayes vychází z výše zmíněného teorému, ale (naivně) vychází z předpokladu, že jednotlivé evidence E jsou podmíněně nezávislé. V tomto případě tedy vzorec pro výpočet pravděpodobnosti vypadá takto:

[2] Jazyk C++ bude představen později v rámci kapitoly Vlastní práce.

$$P(H|E) = P(E|H) \times P(H)$$

Vzorec 2 Naive Bayes

Jmenovatel je možné odstranit, jelikož nás zajímají pouze ty případy, pro které je hodnota aposteriorní pravděpodobnosti maximální.

V praxi se tato pravděpodobnost jednotlivých hodnot v instanci násobí a vzorec vypadá takto:

$$P(H|E) = P(H) \times \prod_{k=1}^{K} P(E|H)$$

Vzorec 3 Naive Bayes v praxi

Kde K je počet jednotlivých hodnot v instanci. Tyto hodnoty se dají jednoduše spočítat z tréninkových instancí a jejich hodnot. Pravděpodobnosti se spočítají jako podíl výskytů jednotlivých hodnot v instanci a celkový počet hodnot pro danou třídu. (Berka, 2009)

Následující příklad ilustruje průběh klasifikace pomocí klasifikátoru Naive Bayes.

Tabulka 1 Tréninková data (Berka, 2009)

klient	příjem	konto	pohlaví	nezaměstnaný	úvěr
k1	vysoký	vysoké	žena	ne	ano
k2	vysoký	vysoké	muž	ne	ano
k3	nízký	nízké	muž	ne	ne
k4	nízký	vysoké	žena	ano	ano
k5	nízký	vysoké	muž	ano	ano
k6	nízký	nízké	žena	ano	ne
k7	vysoký	nízké	muž	ne	ano
k8	vysoký	nízké	žena	ano	ano
k9	nízký	střední	muž	ano	ne
k10	vysoký	střední	žena	ne	ano
k11	nízký	střední	žena	ano	ne
k12	nízký	střední	muž	ne	ano

Apriorní pravděpodobnosti cílového atributu „úvěr“:

$$P(\text{úvěr(ano)}) = 8/12 = 0{,}667$$
$$P(\text{úvěr(ne)}) = 4/12 = 0{,}333$$

Podmíněné pravděpodobnosti spočítáme podobně, např.:

$$P(\text{konto(střední)}|\text{úvěr(ano)}) = 2/8 = 0{,}25$$
$$P(\text{konto(střední)}|\text{úvěr(ne)}) = 2/4 = 0{,}5$$

…

Takto spočítáme všechny pravděpodobnosti a tím klasifikátor natrénujeme. Může se stát, že atribut úvěr nebude zastoupen v podmíněné pravděpodobnosti, např.:

$$P(konto(vysoké)|úvěr(ne)) = 0/4 = 0$$

Tento jev se nazývá „Problém nulové frekvence“. Tento problém se řeší velmi jednoduše a to tak, že se ke všem četnostem přičte hodnota jedna.

Testováním se jednoduše násobí jednotlivé pravděpodobnosti hodnot v instanci pro různé třídy. Na základě nejvyšší pravděpodobnosti je zvolena příslušná třída. Testovací instance je vidět na obrázku 3 a postup testování pod obrázkem:

příjem	konto	pohlaví	nezaměstnaný
vysoký	vysoké	muž	ano

Obrázek 3 Testovací instance (Berka, 2009)

$P(příjem(vysoký)|úvěr(ano)) \times P(konto(vysoké)|úvěr(ano)) \times P(pohlaví(muž)|úvěr(ano)) \times P(nezaměstnaný(ano)|úvěr(ano)) \times P(úvěr(ano)) = 0{,}056$

$P(příjem(vysoký)|úvěr(ne)) \times P(konto(vysoké)|úvěr(ne)) \times P(pohlaví(muž)|úvěr(ne)) \times P(nezaměstnaný(ano)|úvěr(ne)) \times P(úvěr(ne)) = 0{,}032$

Na základě těchto pravděpodobností je vidět, že testovaný klient úvěr dostane, jelikož je dle vypočítaných pravděpodobností hodnota vyšší pro „P(úvěr(ano))“.

3.5 Učení založené na instancích

Nejpoužívanější funkcí učení založeného na instancích (IBL[3]) pro klasifikování nových případů je metoda nejbližších sousedů, která je založena na podobnosti.

Oba algoritmy, které budou v této práci popsány a implementovány, jsou založeny na metodě nejbližších sousedů. S touto metodou se tedy seznámíme blíže v další podkapitole.

3.5.1 k-NN

Princip nejbližších sousedů je založen na podobnosti jednotlivých instancí v souboru. Tyto instance jsou chápány jako body v n-rozměrném prostoru.

Princip algoritmu je popsán v následujícím obrázku:

[3] Instance Based learning

Algoritmus k-NN

Učení

1. Pro každý příklad $[\mathbf{x}_i, y_i]$
   zařaď $[\mathbf{x}_i, y_i]$ do báze příkladů

Klasifikace

1. Pro nový příklad $\mathbf{x}$
   1.1. Najdi $\mathbf{x}_1, \mathbf{x}_2, \ldots \mathbf{x}_K$ K nejbližších příkladů z báze příkladů
   1.2. Přiřaď

$$y = y_i \Leftrightarrow \sum_{k=1}^{K} \delta(y_j, y_k) = \max_i \sum_{k=1}^{K} \delta(y_i, y_k) \ ,$$

kde $\delta(y_i,y_k) = 1$ pro $y_i = y_k$, jinak $\delta(y_i,y_k) = 0$

Obrázek 4 Algoritmus k-NN (Berka, 2009)

„Ve fázi učení si systém zapamatuje všechny příklady [xk, yk] z tréninkové množiny. Ve fázi klasifikace se pro nový příklad x nalezne (za použití zvolené metriky) K nejbližších příkladů, které pak „hlasují“ o zařazení příkladu x do třídy.“ (Berka 2009)

Vše si ilustrujeme na příkladu pro lepší pochopení. V příkladu bude použita euklidovská vzdálenost[4] mezi objekty. Vzorec pro tuto vzdálenost je následující:

$$Euklidovská\ vzdálenost\ (X,Y) = \sqrt{\sum_{i=1}^{k}(x_i - y_i)^2}$$

Vzorec 4 Euklidovská vzdálenost

Ze vzorce vyplívá, že vzdálenost se vypočítá tak, že se od testované instance (x) odečte instance v souboru (y). Takový výpočet algoritmus provede pro každou hodnotu a následně vše sečte a odmocní. Výsledek je euklidovská vzdálenost mezi objekty.

Může zde nastat problém s nominálními atributy. Ve většině souborů jsou numerické i nominální atributy. U numerických atributů se standardizují[5] jednotlivé instance pomocí zvolené metriky. Po standardizaci je možné nominální atributy jednoduše porovnat, a pokud se rovnají s testovanou instancí, je vzdálenost jedna.

[4] Běžně používaná metrika pro měření vzdálenosti nejbližších sousedů. (Sadawi, 2014) Systém Weka také používá Euklidovskou vzdálenost, ale uživatel může zvolit i jiné možnosti počítaní vzdálenosti.

[5] Dle zvolené metriky se nahradí hodnoty v soubory hodnotami mezi 0-1. Algoritmus standardizace se nachází v příloze A.

Jestliže se nerovnají, je vzdálenost nula. Více o metodách standardizace je v kapitole věnované souborům aplikace a práci s nimi. (Sadawi, 2014)

Vše bude ilustrováno na následujícím příkladu, kde přijde do banky nový klient, který chce úvěr a má tyto charakteristiky:

Příjem = 15 000 Kč
Konto = 32 000 Kč

Následně se spočítá euklidovská vzdálenost pro všechny instance (Tabulka 2). Pokud k = 1 (znamená to, že algoritmus bere v potaz pouze jednoho souseda), tak algoritmus bere v potaz pouze nejmenší vzdálenost (nejbližšího souseda), jako v našem příkladu.

Tabulka 2 Tabulka nejbližších sousedů (Berka, 2009)

klient	příjem	konto	úvěr	vzdálenost příkladu
k1	3000	15000	ne	20808.65
k2	10000	15000	ne	17720.05
k3	17000	15000	ano	17117.24
k4	5000	30000	ne	10198.04
k5	15000	30000	**ano**	**2000.00**
k6	20000	50000	ano	18681.54
k7	2000	60000	ne	30870.70
k8	5000	90000	ano	58855.76
k9	10000	90000	ano	58215.12
k10	20000	90000	ano	58215.12
k11	10000	100000	ano	68183.58
k12	17000	100000	ano	68029.41

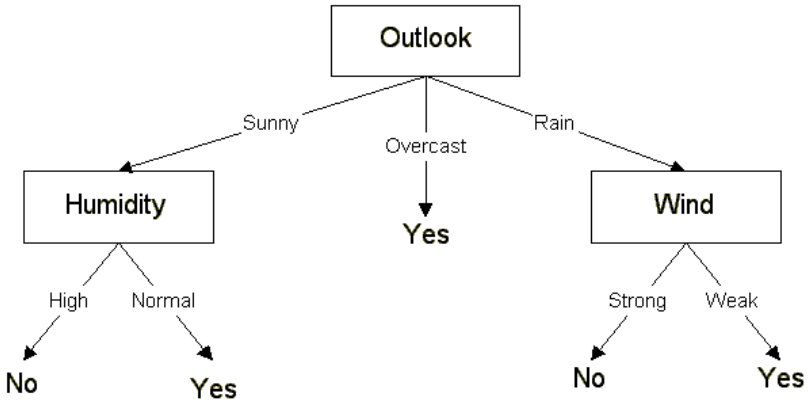
V příkladu vidíme, že nejbližší soused je klient k5, tím pádem nový klient úvěr dostane (úvěr = ano).

V případě většího k[6] (např.: k = 3) by se uvažovali tři nejbližší sousedi. Z těchto vzdálenosti by se poté vybrala třída s největší četností. V příkladu výše by tři nejbližší sousedi byli k5, k4 a k3. U k3 a k5 je třída ano, tím pádem i při třech nejbližších sousedech by žadatel úvěr dostal.

3.6 Rozhodovací stromy

Decision Trees (Rozhodovací stromy) je třída algoritmů, která na základě analýzy pravidel a vztahů vytváří grafy (především grafy stromové struktury, viz Obrázek 5). Cílem těchto algoritmů je rozdělit datový soubor tak, aby se snížila míra nejistoty (tzv. entropie).

[6] Konstantu k (počet sousedů) uživatel zadává jako parametr algoritmu



Obrázek 5 Příklad rozhodovacího stromu (Dankel, 1997)

„Při tvorbě rozhodovacího stromu se postupuje metodou „rozděl a panuj“ (divide and concquer). Tréninková data se postupně rozdělují na menší a menší podmnožiny (uzly stromu) tak, aby v těchto podmnožinách převládaly příklady jedné třídy. Na počátku tvoří celá tréninková data jednu množinu, na konci máme podmnožiny tvořené příklady téže třídy. Tento postup bývá často nazýván „top down induction of decision trees“ (TDIDT).“ (Berka, 2009)

TDIDT algoritmus

1. zvol jeden atribut jako kořen dílčího stromu,
2. rozděl data v tomto uzlu na podmnožiny podle hodnot zvoleného atributu a přidej uzel pro každou podmnožinu,
3. existuje-li uzel, pro který nepatří všechna data do téže třídy, pro tento uzel opakuj postup od bodu 1, jinak skonči.

Obrázek 6 Obecný TDIDT algoritmus (Berka, 2009)

Obecný postup TDIDT algoritmu je vidět na Obrázku 2. Dále si krátce vypíšeme výhody a nevýhody rozhodovacích stromů.

- **Výhody**
  - Jednoduché a efektivní.
  - Použitelné i pro velké databáze.
  - Extrakce jednoduchých pravidel.
- **Nevýhody**
  - Obtížnější zpracování spojitých dat.
  - Obtížnější zpracování při chybějících údajích. (tzv. missing values)
  - Přeučení. (tzv. over-fitting)
  - Vzájemná korelace mezi atributy se nebere v úvahu.

Tyto stromy se v oblasti strojového učení používají pro řešení klasifikačních úloh. Tato klasifikace má dva obecné kroky:

- Indukce rozhodovacího stromu podle tréninkových dat – to znamená vytvoření stromu na základě zvolené metriky.
- Podle vytvořeného rozhodovacího stromu se určí třída v testovacích datech. (Mrázová, 2012)

3.6.1 ID3 Decision Tree

Strom ID3 (Iterative Dichotomizer 3) je základním algoritmem rozhodovacích stromů. Indukce stromu se provádí na základě výpočtů míry neurčitosti (entropie) a informačního zisku (information gain). (Rizky, 2011)

U všech stromů je důležité vybrat vhodný atribut, dle kterého se bude strom dále větvit. Je třeba vybrat takový atribut, který od sebe nejlépe odliší příklady různých tříd. Toto se u ID3 provádí právě výpočtem míry entropie a následným výpočtem informačního zisku. Oba vzorce si představíme níže.

Vzorec pro výpočet entropie:

$$E(S) = -\sum_{t=1}^{T} (p_t \times \log_2 p_t)$$

Vzorec 5 Výpočet míry Entropie

Kde p_t je pravděpodobnost výskytu třídy t a T je počet tříd. Vzorec pro výpočet informačního zisku vypadá následovně:

$$Gain(S, A) = E(S) - \sum_{t=1}^{T} \frac{|S_v|}{S} \times E(S_v)$$

Vzorec 6 Výpočet informačního zisku

Vše, co je potřeba pro větvení stromu, je vysvětleno a nyní blíže k postupu, jakým tento algoritmus strom vytváří:

- Nejprve se vypočítá míra entropie tréninkových dat. Jinak řečeno míra entropie třídy, která se bude testovat - *E(S)*.
- Poté se vypočítá míra entropie pro každý atribut (sloupec dat) - $E(S_v)$.
- Tato míra entropie se vynásobí pravděpodobností jednotlivých hodnot v atributu - $\frac{|S_v|}{S}$.
- Tím se vypočítá informační zisk a následně se zvolí atribut s nejvyšším informačním ziskem jako nový uzel (pokud se algoritmus provádí poprvé, tak se tento uzel nazývá kořenový).
- Pokud v tomto uzlu nepatří všechny hodnoty do stejné třídy, tak opakuj postup rekurzivně, jinak vytvoř list. (Berka, 2009)

Vše si ilustrujeme na následujícím příkladu, kde tabulka tréninkových dat je stejná jako u klasifikátoru Naive Bayes.

Tabulka 3 Četnosti příjmů pro jednotlivé třídy (Berka, 2009)

		úvěr	
		ano	ne
příjem	[vysoký :	5.0	0.0]
	[nízký :	3.0	4.0]

Nejprve spočítáme entropii pro třídu:

$$E\,(\text{úvěr}) = (-\,p+ \times \log 2p+) + (-\,p- \times \log 2p-) = 1{,}571$$

Poté je třeba zjistit informační zisk, který spočítáme z tabulky 3 takto:

$$\text{Gain}(\text{úvěr, příjem}) = E\,(\text{úvěr}) - (5/12 + H(\text{příjem(vysoký)}) + 7/12 \times H(\text{příjem(nízký)})$$

přičemž

$$H(\text{příjem(vysoký)}) = (-\,p+ \times \log 2p+) + (-\,p- \times \log 2p-) = 0 + 0 = 0$$
$$H(\text{příjem(nízký)}) = (-\,p+ \times \log 2p+) + (-\,p- \times \log 2p-) = 0{,}9852$$

Tedy

$$\text{Gain}(\text{úvěr, příjem}) = 1{,}571 - (5/12 \times 0 + 7/12 \times 0.9852) = 0{,}996$$

Podobně se spočítají i další atributy a jejich informační zisk.

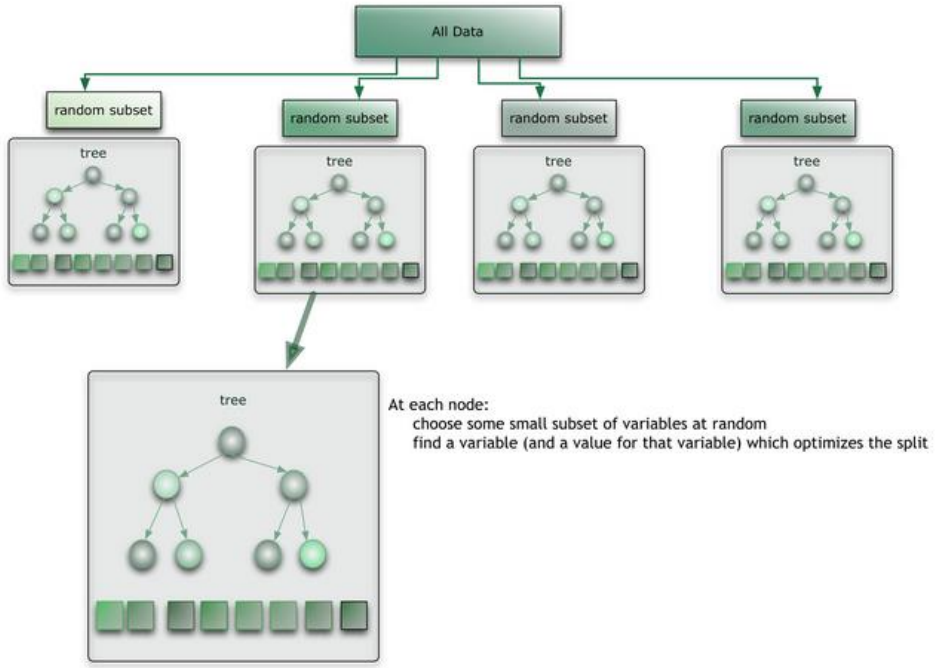
$$\text{Gain}(\text{úvěr, konto}) = 0{,}905$$
$$\text{Gain}(\text{úvěr, pohlaví}) = 0{,}593$$
$$\text{Gain}(\text{úvěr, nezaměstnaný}) = 0{,}746$$

Jak vidíme na příkladu, tak na větvení stromu použijeme atribut „příjem“, jelikož má největší hodnotu informačního zisku. Poté se celý proces opakuje, dokud není generování stromu zastaveno. (Berka, 2009)

3.6.2 Random Forest

Rozhodovací strom Random Forest (náhodný les), jak již název napovídá, je založen především na náhodě. Je podobný stromu ID3, s některými menšími odlišnostmi.

Random Forest vytváří více stromů, které tvoří les. Tyto stromy se vytváří z celkových dat, kde náhodně vybere 66 % (většinou se používá tato konstanta) z těchto dat a ty používá k vytvoření stromu. Ostatní data využívá k výpočtu odhadu chyby (tzv. Out of Bag error). Takových stromů vytvoří desítky, stovky a někdy i tisíce pro zlepšení klasifikace. Výsledek je vidět na obrázku níže.



Obrázek 7 Random Forest (Benyamin, 2012)

Každý strom se také dělí na základě výpočtu entropie a informačního zisku, jak tomu bylo u ID3. Existují i jiné metody výpočtu pro dělení stromu. Zde je pouze rozdíl v tom, kolik atributů vstupuje do výpočtu.

U algoritmu Random Forest uživatel zadá dva parametry. První parametr je počet stromů, které se mají vytvořit. Druhý parametr je tzv. k-hodnota. Tato hodnota udává právě to, kolik atributů vstupuje do výpočtu entropie a informačního zisku. Dle velikosti k-hodnoty algoritmus náhodně vybere k-atributů. Poté už pokračuje stejně jako u předchozího rozhodovacího stromu (ID3). Následuje výpočet entropie a informačního zisku a na základě maximálního informačního zisku dělí strom, dokud není generování celého stromu kompletní. (Benyamin, 2012)

Trénování je, jako tomu u ID3, vytvoření lesu. Testování je procházení jednotlivých stromů a nalezení požadované instance a její třídy.

Náhodný přístup a také počet stromů vykazuje lepší výsledky jak samotný strom ID3 nebo Random Tree. Při větším počtu stromů se algoritmus lépe naučí a poté je klasifikace znatelně lepší. Otázkou je, kolik stromů zvolit, aby se algoritmus tzv. nepřeučil.

Pokud se počet stromů rovná jedné, vzniká tzv. Random Tree. Tento strom pracuje na stejném principu jako Random Forest, ale v lese se objevuje pouze jeden strom.

4 Dosavadní řešení

V této kapitole práce se seznámíme s dosavadním řešením v systému Weka. Je třeba se seznámit s celým prostředím a pozadím tohoto systému. Algoritmy byly popsány výše. Zde budou představeny pouze některé další sekundární algoritmy, se kterými Weka pracuje (filtry a algoritmy pro rozdělení dat). Také představit základní funkcionalitu systému Weka, prostředí a některé další možnosti, které tento systém nabízí.

Systém Weka obsahuje dohromady více jak šedesát algoritmů pro klasifikaci, clusterizaci a také asociaci dat. Nejprve si představíme, jak a pro jaký účel Weka vznikla.

4.1 Weka

Tento projekt vznikl na univerzitě Waikato na Novém Zélandě a reagoval na vynález a použití metod strojového učení (ML). Ty umožňují počítačovému programu automaticky analyzovat velké množství dat, a rozhodnout, jaké informace jsou nejvíce relevantní. Tyto krystalizované informace pak mohou být použity pro automatické předpovědi nebo pomáhat lidem rozhodovat se rychleji a přesněji. Cíle projektu:

- mít ML techniky všeobecně k dispozici,
- aplikovat je na praktické problémy, které má průmysl na Novém Zélandě,
- rozvíjet nové algoritmy strojového učení a ukázat je světu,
- přispívat k teoretickému základu této problematiky. (Machine Learning Group, 2008)

Software pro tento projekt se jmenuje „Weka“. Byl pojmenován podle ptáka, který žije pouze na ostrovech Nového Zélandu. Celý projekt i software je open source (tzn. transparentní zdrojové kódy).

Weka je sbírka algoritmů strojového učení pro dolování dat. Tyto algoritmy mohou být, buď aplikovány přímo na datovou sadu, nebo se mohou volat z vlastního kódu v jazyce Java. Weka obsahuje nástroje pro datové předzpracování, klasifikaci, regresi, shlukování, asociační pravidla, a vizualizaci. Je také velmi vhodný pro vývoj nového systému strojového učení. (Weka 3: Data Mining Software in Java, 2013)

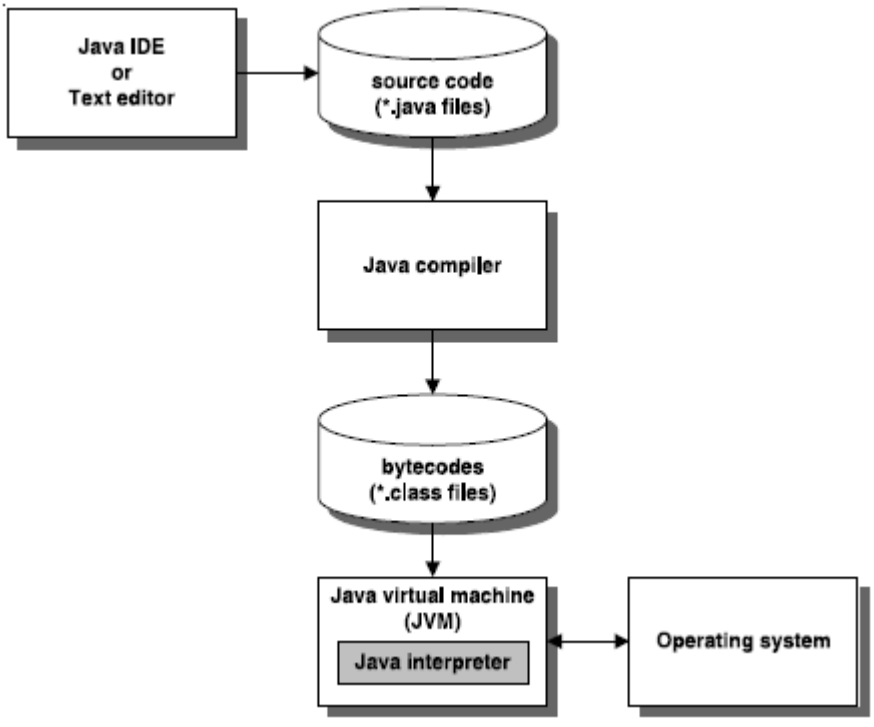
Výhod této aplikace je nespočet. Především jsou zde implementovány téměř všechny algoritmy strojového učení. Další výhoda spočívá v dokumentaci, která je značně rozsáhlá. Jedná se zde o spoustu knih a manuálů, ale také zodpovězených otázek na oficiálním fóru a transparentních zdrojových kódů. Další výhodou je zcela určitě propracovanost celé aplikace. Je zde možnost si vše zkoušet, vizualizovat, ukládat a mnoho dalších možností. Celkově je tenhle software nejlepším a velmi oblíbeným řešením strojového učení v kostce.

Jako první nevýhoda je práce převážně s arff soubory (soubory s touto příponou jsou vysvětleny později), jelikož většina databází tento formát souborů nepodporuje. Druhá nevýhoda je, že celý systém je napsán v jazyce Java. Jazyk Java je interpretovaný, tím pádem je potřeba interpret (Java virtual machine). Tento interpret interpretuje kód do tzv. mezikódu, místo toho, aby přeložil kód přímo do strojového kódu, jak to provádí například jazyk C++. Java bude vysvětlena podrobněji v následující kapitole. Tímto se celý proces počítání zpomaluje a zvyšuje také nároky na paměť. Tuto nevýhodu se snaží u některých základních algoritmů odstranit právě tato práce.

4.1.1 Java

Celý systém Weka je napsán v Jave. Ta byla poprvé představena v roce 1996 firmou Sun Microsystems. Dnes je to jeden z nejpoužívanějších objektově orientovaných programovacích jazyků.

Na obrázku níže je zobrazeno, jak funguje přeložení kódu v jazyce Jave. Nejprve uživatel vytvoří zdrojový kód, poté se použije Java kompilátor, který přeloží tento zdrojový kód do tzv. mezikódu (bytecode). Poté může být tento kód spuštěn na jakékoliv platformě[7] s nainstalovaným Java interpretem. Tento interpret je v podobě Java virtual machine (JVM). (Murach, 2011, s. 10-11)



Obrázek 8 Jak Java interpretuje zdrojový kód (Murach, 2011, s. 11)

[7] Platformou se myslí Window, Linux nebo Mac.

Nevýhoda C++ je taková, že pro každou platformu musí kompilovat kód jiný kompilátor, který je kompatibilní s danou platformou. Více o rozdílech mezi C++ a jazykem Java jsou vidět v následujícím obrázku:

Feature	Description
Syntax	Java syntax is similar to C++ and C# syntax.
Platforms	Compiled Java code can be run on any platform that has a Java interpreter. Similarly, compiled C# code (MSIL) can be run on any system that has the appropriate interpreter. Currently, only Windows has an interpreter for MSIL. C++ code must be compiled once for each type of system that it is going to be run on.
Speed	C++ and C# run faster than Java, but Java is getting faster with each new version.
Memory	Both Java and C# handle most memory operations automatically, while C++ programmers must write code that manages memory.

Obrázek 9 Java v porovnání C++ (Murach, 2011, s. 5)

Obrázek 9 shrnuje některé základní rozdíly mezi oběma programovacími jazyky.

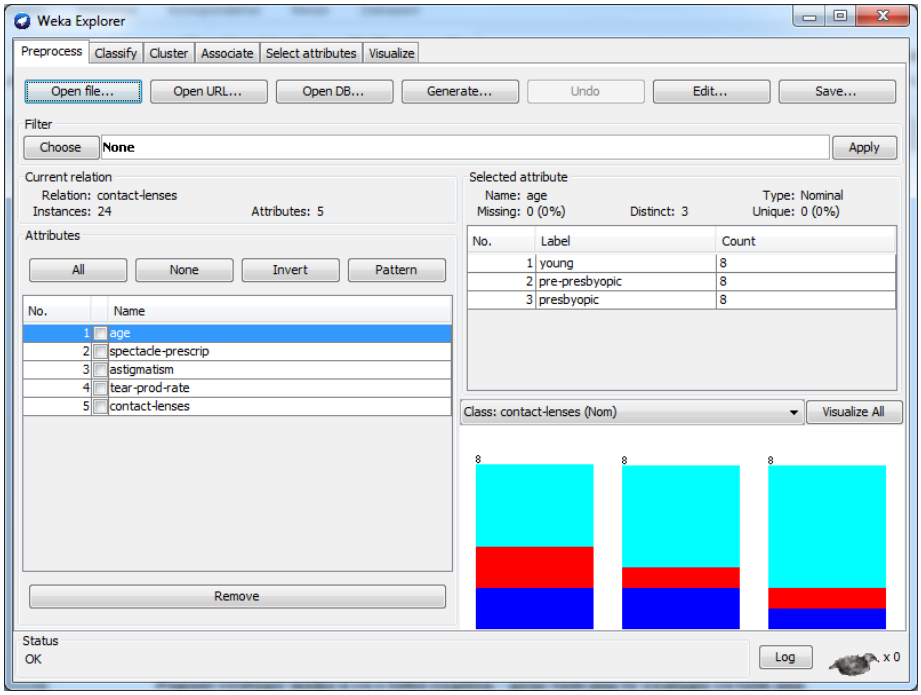
- **Syntax** – velmi podobná u obou jazyků.
- **Platformy** – vysvětleny v předchozím textu.
- **Rychlost** – C++ je rychlejší jak Java, ale Java je s každou novou verzí o něco rychlejší.
- **Paměť** – Java zvládá práci s pamětí automaticky, zatímco v C++ musí programátor napsat kód pro zvládání paměti. (Murach, 2011, s. 5)

4.2 Prostředí Weka

Prostředí programu Weka je velice jednoduché, přehledné a dobře propracované. Jako první ihned po instalaci se objeví dialog s informacemi o programu s možností výběru mezi různými nástroji (Obrázek 10). Pro úspěšné spuštění je potřeba mít nainstalovanou Javu a virtuální Java stroj v počítači (Weka má instalaci JVM zabudovanou přímo v instalátoru).

Obrázek 10 Úvodní obrazovka (Vlastní zpracování)

Pro účel práce postačí první nástroj, který se nazývá Explorer (Prohlížeč). Tato nabídka spustí základní program pro většinu nezkušených uživatelů. Po načtení nejlépe arff souboru Weka automaticky vypočítá základní charakteristiky a zobrazí je uživateli (Obrázek 11). Aplikace obsahuje různé záložky pro práci s daty a také velké množství dalších možností.

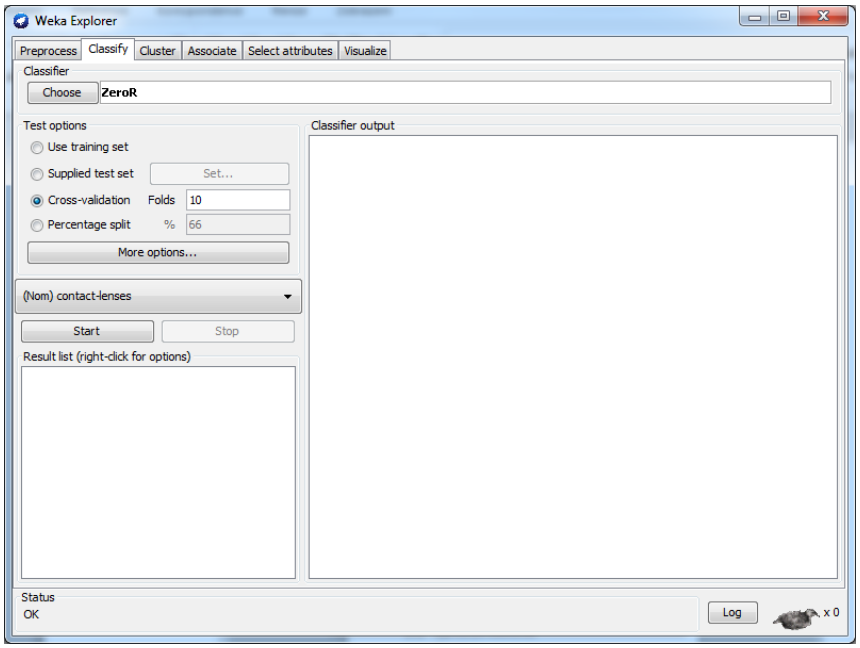


Obrázek 11 Weka Explorer (Vlastní zpracování)

Dále je možné aplikovat různé filtry na data, odstranit některé atributy a zobrazit různé grafy a další možnosti. (Witten a Frank, 2005, s. 407-409)

Pro potřeby této práce bude třeba pouze filtr „Discretize“ a „ReplaceMissingValues“. Tyto filtry budou představeny v samostatných kapitolách.

Pro tuto práci je stěžejní druhá záložka Classify (Klasifikace). Po přepnutí do této záložky je možný výběr algoritmů, jejich parametrů, dále možnost rozdělení dat a jiné další nástroje (Obrázek 12). Většina těchto nástrojů bude popsána v samostatných kapitolách.



Obrázek 12 Classify tab (Vlastní zpracování)

Takto jednoduše vypadá aplikace Weka. Nyní se podíváme na jednotlivé možnosti tohoto programu a také na jeho výstupy.

4.2.1 CSV versus Arff soubory

CSV (textové soubory s hodnoty odděleny čárkou) jsou základní soubory pro práci s databázemi. Většina databází a tabulek se dá exportovat do souboru ve formátu csv. První řádek vždy nese názvy atributů a další řádky obsahují jednotlivé hodnoty oddělené čárkou.

Jak již bylo dříve naznačeno, tak systém Weka pracuje nejlépe s arff soubory. Weka umí pracovat i s csv soubory, ale má to svá úskalí. Nejprve se tyto soubory musí zhodnotit, aby se zjistilo, jestli jsou atributy nominální nebo numerické. To zabere nějaký čas u rozsáhlých souborů. Také nemusí být kompatibilní testovací a tréninková data, z důvodu neznalosti typů a hodnot atributů, které nabývají. (Can I use CSV files?, 2005)

Soubory s příponou arff jsou speciálním formátem vyvinutým právě pro systém Weka. Na rozdíl od csv musí mít odděleně popsané jednotlivé atributy a také to jakých nabývají tyto atributy hodnot, jak je vidět na obrázku 13. Pro nominální atributy jsou ve složených závorkách vypsány hodnoty, kterých nabývají a pro numerické atributy následuje za názvem atributu klíčové slovo „numeric“. Soubory arff mohou mít navíc nějaké další hodnoty, jako např. popis dat, kde byly data sbírány, za jakým účelem a některé další informace spojené s daty. (Witten a Frank, 2005, s. 52-55)

```
% ARFF file for the weather data with some numeric features
%
@relation weather

@attribute outlook { sunny, overcast, rainy }
@attribute temperature numeric
@attribute humidity numeric
@attribute windy { true, false }
@attribute play? { yes, no }

@data
%
% 14 instances
%
sunny, 85, 85, false, no
sunny, 80, 90, true, no
overcast, 83, 86, false, yes
rainy, 70, 96, false, yes
rainy, 68, 80, false, yes
rainy, 65, 70, true, no
overcast, 64, 65, true, yes
sunny, 72, 95, false, no
sunny, 69, 70, false, yes
rainy, 75, 80, false, yes
sunny, 75, 70, true, yes
overcast, 72, 90, true, yes
overcast, 81, 75, false, yes
rainy, 71, 91, true, no
```

Obrázek 13 Ukázka arff souboru (Witten a Frank, 2005, s. 53)

Existují online převaděče mezi csv a arff soubory. Systém Weka má i vlastní algoritmy, které převedou csv na arff soubor. V praxi online převodníky nejsou úplně stoprocentní.

V aplikaci vytvořené pro tuto práci se pracuje pouze s csv soubory. Pro potřeby práce je to dostačující a navíc arff soubory jsou určeny přímo pro systém Weka, který je pro ně uzpůsoben.

4.2.2 Algoritmy pro rozdělení dat

V systému Weka existují tři možnosti jak rozdělit data na tréninková a testovací. Čtvrtá možnost přímo nabízí přidat soubor s testovacími daty. Tudíž se trénuje na celé množině dat a testovací data jsou ta, která uživatel nahrál. Tato možnost se nazývá „Supplied test set“ (volně přeloženo jako dodaný testovací soubor). Další tři možnosti si představíme podrobněji. Výběr mezi těmito algoritmy je ukázán na obrázku výše (obrázek 12).

a) Use training set

Tato metoda je ze všech nejjednodušší. Všechna data ze souboru jsou, jak testovací, tak tréninková. Soubor se nijak nedělí. Klasifikátor se natrénuje na všech

datech a na stejných datech také testuje. Metoda pro velké množství testovacích instancí může být někdy pomalejší. (Bouckaert et al., 2014)

b) Cross-validation

Hojně používaná metoda pro rozdělení dat. V této metodě uživatel zadává počet tzv. foldů (záhybů). V metodě Cross-validation se také octnou všechny instance v tréninkové i testovací množině. Nejčastěji se používá 10-fold Cross-validation, kterou si blíže představíme v následujícím odstavci.

Tato metoda rozdělí data do k=10 množin. V každé této množině použije k-1 instancí pro trénování a poslední instanci použije pro testování. Instance pro testování je vybrána náhodným výběrem. Tento postup se opakuje desetkrát (10-fold), tím pádem jsou všechny instance testovací i tréninková.

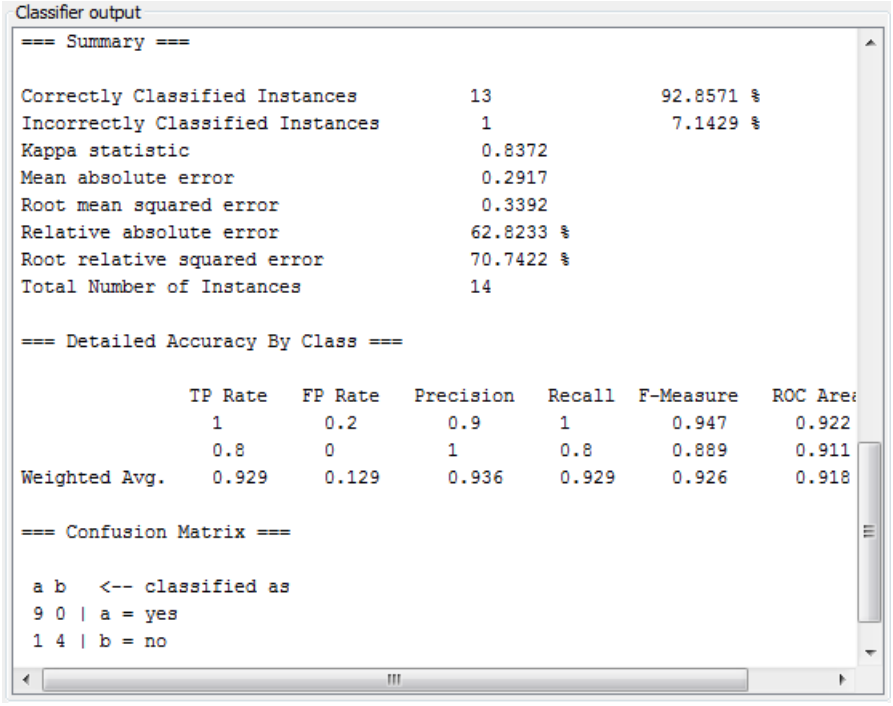
Nevýhoda této metody spočívá v tom, že se musí provést k-krát. Tím pádem zabere k-více času pro natrénování a testovaní instancí. (Schneider, 1998)

c) Percentage split

Metoda rozdělení souboru procentuálně. Nejčastěji se používá rozdělení 2:3, tedy 66 %. V praxi často používaná metoda, kde se náhodně vyberou data a rozdělí se. Toto rozdělení záleží na uživateli zadaném procentu. Procento, které uživatel zadá, říká, kolik procent z celkových dat se použije na data tréninková. Zbytek se použije na data testovací. Jednotlivé instance pro oba typy dat se vybírají náhodným výběrem. (Bouckaert et al., 2014)

4.2.3 Obrazovka s výstupy

Je důležité pochopit, jaký výstup systém Weka uživateli nabízí. Tento výstup se vypíše na obrazovku uživateli po kliknutí na tlačítko Start a vykonání zvoleného algoritmu. Výstup jako takový je vidět na obrázku 12. Pro každý algoritmus se výstup mírně liší. V této kapitole si představíme především výstupy, které jsou stejné pro každý algoritmus ze třídy klasifikačních algoritmů. Většina těchto výstupů bude implementována i v aplikaci práce.



Obrázek 14 Výstup Weka (Vlastní zpracování)

Jak je vidět na obrázku výše, tak nejprve je uživateli zobrazen obecný výstup „Summary“ (souhrn), poté jsou některé statistiky pro třídu, část „Detailed Accuracy By Class“ (Detailní přesnost podle třídy) a nakonec je vypsána „Confusion Matrix“ (Matice zmatení). Každou část výstupu si představíme podrobněji.

a) Summary

Z důvodu toho, že se v práci objevují pouze klasifikační úlohy a žádné jiné budou popisovány pouze některé charakteristiky. Všechny chyby („Mean absolute error“, „Root mean squad error“ a oba relativní errory) v oblasti souhrnu jsou při klasifikačních algoritmech jen málo významné. Používají se například v regresních úlohách. (Bouckaert et al., 2014)

- **Correctly Classified Instances** (Správně klasifikované instance) – Tato charakteristika udává, kolik instancí bylo správně klasifikováno příslušným klasifikátorem. Vedle číselného vyjádření je i procentuální.
- **Incorrectly Classified Instances** (Nesprávně klasifikované instance) – Zde jsou ostatní instance, které klasifikátor označil špatně.
- **Kappa statistics** – Tato charakteristika se správně označuje jako Cohenův kappa koeficient. Udává míru toho, jak je klasifikátor dobře natrénován vůči míře toho, jak by mohl být natrénován náhodou.
- **Total number of instances** – Celkový počet instancí, které byly testovány. (Markham, 2014)

b) Confusion Matrix

Tato matice, často nazývaná kontingenční tabulka, vyjadřuje výkon klasifikačního modelu. Na obrázku 15 je zobrazena matice z obrázku výše a přepsána pro lepší vysvětlení. Jedná se tedy o stejný příklad.

n= 14	**Predicted: NO**	**Predicted: YES**	
Actual: NO	TN = 4	FP = 1	5
Actual: YES	FN = 0	TP = 9	9
	4	10	

Obrázek 15 Confusion matrix (Vlastní zpracování)

Součet všech prvků musí dávat počet testovaných instancí. Diagonála ukazuje, kolik instancí bylo správně klasifikováno pro jednotlivé třídy (v našem případě yes a no). Ostatní části tabulky ukazují, jak byly klasifikovány další instance. (Markham, 2014)

c) Detailed Accuracy By Class

V této části výstupu jsou vidět jednotlivé charakteristiky pro třídu, kde každý řádek patří jedné třídě. Většina těchto charakteristika se počítá na základě výstupů z Confusion matrix a je považována za dobré, čím více se blíží jedné. Poslední řádek jsou vážené průměry těchto charakteristik.

- **True-positive rate** (Skutečně pozitivní hodnocení) – Často také nazýváno senzitivita. Charakteristika udává podíl toho, kolik instancí je hodnoceno správně vůči celkově hodnoceným instancím třídy v řádku matice. V našem příkladu pro yes: 9/(9+0) = 1
- **False-positive rate** (Falešně pozitivní hodnocení) – Opak skutečně pozitivních hodnocení. Udává podíl špatně klasifikovaných vůči všem klasifikovaným danou třídou v řádku. Pro případ yes: 4/5 = 0,2
- **Precision** (Přesnost) – Udává přesnost algoritmu. S jakou přesností určí danou třídu. Spočítá se jako podíl správně klasifikovaných instancí a všech instancí predikovaných danou třídou ve sloupci. Pro yes: 9/10 = 0,9
- **Recall** – Charakteristika se vypočítá dle vzorce: *TP/(TP+FN).* Udává podíl skutečně pozitivních hodnocení se součtem všech hodnocení dané třídy. V příkladu pro yes: 9/(9+0) = 1

- **F-measure** (F-metrika) – Metrika pro vyjádření přesnosti testování, která bere v potaz jak Precision tak Recall. Vypočítá se jako vážený průměr těchto dvou charakteristik.
- **ROC Area** (ROC oblast) – Tzv. ROC křivka shrnuje výkon klasifikátoru nad všemi možnými prahy. Generuje se pomocí TP rate (osa y) a FP rate (osa x), když se mění prahy pro vyjádření dané třídy. ROC area je oblast, která se nachází pod touto křivkou. Udává jaký je skutečný výkon klasifikace. Hodnoty mezi 0.5-1 jsou považovány za dobré. (Markham, 2014)

4.2.4 Filtry

Jak již bylo zmíněno v úvodu kapitoly, tak filtry jsou nedílnou součástí systému Weka. Ten obsahuje více jak třicet filtrů, které různě upravují data. Některé jsou s učitelem a jiné bez učitele. V přepisované aplikaci jsou použity dva filtry, především pro zjednodušení klasifikace. Nejprve bude popsáno, jak filtry fungují v aplikaci Weka a později v práci budou implementovány v jazyce C++.

a) Diskretizace

Filtr diskretizace (Discretize) je filtr pro nahrazení numerických (číselných) atributů oblastmi do kterých spadájí jednotlivé hodnoty v instanci. Některé algoritmy umějí pracovat pouze s nominálními daty a nejen k tomu slouží právě tento filtr. Pracuje na způsobu rozdělení numerických hodnot do tříd. Počet těchto tříd zadává uživatel jako parametr (obecně nastaveno na deset tříd). (Bouckaert et al., 2014, s. 17)

Filtr nejprve najde minimální a maximální hodnotu a dle vzorce uvedeného pod textem najde první třídu (FC).

$$FC = \frac{min\times\ (počet\ tříd - 1) + max}{počet\ tříd}$$

Vzorec 7 Výpočet prahu prví třídy

Dále pokračuje tak, že od minima odečte tuto první třídu a tím vypočítá rozdíl, který poté přičítá a tím nalezne všechny prahy.

Nakonec jednoduše nahradí všechny hodnoty atributů příslušnou třídou, podle toho, do které třídy hodnota patří.

b) Chybějící hodnoty

Tento filtr (ReplaceMissingValues), jak již název napovídá, nahradí chybějící hodnoty v datech. Tyto chybějící části dat jdou ve většině souborů označeny znakem „?“. Filtr pracuje jinak s numerickými daty a jinak s daty nominálními.

U nominálních dat je to jednoduché. Vypočítá četnosti jednotlivých hodnot a místo chybějící hodnoty, vloží hodnotu nejčetnější. U numerických atributů se vypočítá průměr ze všech hodnot a ten se vloží místo chybějící hodnoty. (Class ReplaceMissingValues, 2014)

Nevýhoda je, že při větším množství chybějících dat dochází ke zkreslení. Algoritmus problému chybějících hodnot bude později vysvětlen v rámci vlastní práce.

Tyto filtry Weka nepoužívá na každý zpracovávaný soubor. Filtry jsou zde vysvětleny, jelikož se používají v přepisované aplikaci. Každý algoritmus ve Wece pracuje s chybějícími hodnotami jinak. Například u algoritmu IBk se nahradí hodnoty pomocí tohoto algoritmu. U klasifikačního algoritmu Naive Bayes, se chybějící hodnoty jednoduše přeskakují.

Oba filtry jsou převáděny z důvodu ulehčení výpočtů jednotlivých algoritmů. Také pro sjednocení práce s atributy.

5 Vlastní práce

Dosavadní řešení bylo představeno. Systém Weka je opravdu propracovaný a dobře naprogramovaný. Nyní si ukážeme řešení, které bylo vytvořeno pro účely této práce. Aplikace je velice jednoduchá a její pracovní název je „Data Mining“. Je napsaná v jazyce C++ a pro grafickou stránku aplikace byl použit freamwork Qt a aplikace pro tento freamwork, která se nazývá QtCreator. Co je jazyk C++ a freamwork Qt bude vysvětleno v následující kapitole. Nakonec bude představena samotná aplikace a to, jak byly algoritmy převedeny.

5.1 C++ a freamwork Qt

5.1.1 C++

Jazyk C++ vznikl především přidáním nových konstrukcí a jen několika změnami z jazyka C. Aktuální verze C++ (verze ISO/IEC 14882-2011, většinou nazývána C++11) přináší oproti předchozí verzi značné zlepšení programovacího jazyka. (Stroustrup, 2013, s. 9)

„Jedná se o poměrně nízkoúrovňový a jednoduchý procedurální jazyk určený pro skutečné praktické programování a ne pro kochání se čistotou syntaxe jazyka. C je velmi efektivní a nevyžaduje náročnou runtime podporu jako třeba Java. Céčko se překládá do strojového kódu. Nejprve se spustí preprocesor, který před vlastním překladem upraví zdroják, následuje překlad do objektového kódu a závěrečnou fází je linkování. Jednotlivé konstrukce jazyka jdou velmi jednoduše přeložit do asembleru, kterému se navíc podobá volným přístupem do paměti (ale samozřejmě nikoli syntaxí).“ (NG, 2004)

Zmínka o Jave je zde důležitá, jelikož je to především náročná runtime podpora v podobě Java virtual machine, kvůli které tato práce vznikla.

- **Výhody**

Velká oblíbenost a s tím související dobrá dostupnost překladačů, knihoven a dalších nástrojů. Většina novodobých jazyku z C vychází. V C jsou napsány operační systémy, ovladače a servery. Z toho vyplývá, že jazyk C je multiplatformní[8]. (Němec, 2004)

V neposlední řadě je tento jazyk neinterpretovaný a také má velikou uživatelskou základnu a s tím je spjatá dobrá dokumentace ve formě různých manuálů, seriálů, videí a fóru StackOverflow[9].

[8] Ta znamená, že se dají spustit jak na platformě Windows, tak i Linux nebo Mac OS.

[9] Na tomto fóru je bezpočet otázek a odpovědí na většinu problémů programovaní v jazyce C++

- **Nevýhody**

Do nevýhod se často zařazuje přímý přístup do paměti, což může vést k chybám v kódu nebo přetečením zásobníku. Tyto chyby C programátorů vedou k častým útokům hackerů. (Němec, 2004)

Další nevýhodou je to, že pro každou platformu se musí kompilovat kód zvlášť. Jinak řečeno pro každou platformu se musí použít jiný kompilátor.

Některé nevýhody řeší právě jazyk C++, který z C vychází.

5.1.2 Qt freamwork

Qt freamwork (aplikační rámec) je podpůrná aplikace, která obsahuje nástavbu jazyka C++. Standardně používá jazyk QML, který je pro Qt určen a Qt s tímto jazykem lépe pracuje. Nicméně je možné psát program i v klasickém C++, jako je to v této práci. freamwork Qt byl použit především pro jednoduché vytváření grafického rozhraní pro uživatele.

Qt je složeno z různých modulů a widgetů. Widget je element, který dovoluje použít klasické desktopové prvky. Prvky jako jsou tlačítka (QPushButton), textová pole (QLineEdit) a další.

Qt bylo vytvořeno především pro vytváření multiplatformních aplikací, které vypadají stejně jako aplikace zabudované v systému Windows, Unix nebo Mac. (James, 2010)

Další možností v Qt jsou zdířky (slots). Tyto zdířky dovolují jednoduše zvládat akce s jednotlivými widgety. Zdířky zvládají například kliknutí na tlačítko, vybrání z listu možností aj.

- **Výhody**

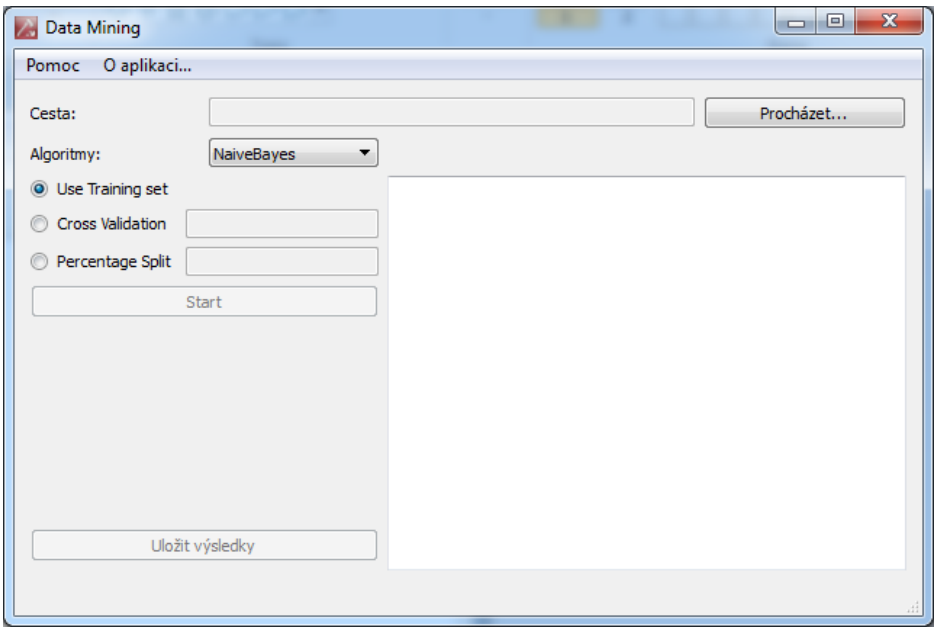
Výhody jsou především ve vytváření grafického rozhraní, které je jednoduché přehledné a uživatelsky přívětivé. Další výhodou je jednoduché zvládání uživatelských vstupů ve formě zdířek. Dále pak také rozsáhlá dokumentace na oficiálních stránkách a v oficiálním fóru. Jako poslední také podpora většiny světových jazyků pro aplikace.

- **Nevýhody**

Většina programátorů píše kód v C++ a není pro ně výhodné přepisovat kód do QML při potřebě uživatelského rozhraní. Navíc je potřeba se naučit práci s QML a některými datovými typy a jejich funkcemi, které v klasickém C++ fungují, ale v Qt podporu nemají.

5.2 Úvod do aplikace Data Mining

Aplikace je jednookná, jednoduchá a uživatelsky přívětivá. Grafické uživatelské rozhraní je podobné rozhraní ve Wece (Obrázek 16). Po spuštění je potřeba otevřít soubor, ze kterého se budou čerpat data. O souborech pro aplikaci bude vše řečeno později.



Obrázek 16 Data Mining aplikace (vlastní zpracování)

Z počátku je v aplikaci většina možností zašedlých. Je to proto, že nejprve je třeba vybrat soubor, se kterým se bude pracovat.

Samozřejmostí je také stručná nápověda, která je zobrazena v horní liště aplikace. V nápovědě je popsána funkcionalita jednotlivých polí a možností jejich výběru. Dále také popsání parametrů funkcí a celková nápověda pro ovládání aplikace.

5.2.1 Práce se soubory

Pro kliknutí na tlačítko „Procházet…“ vyskočí dialogové okno pro výběr souboru. V aplikaci Data Mining je možné nahrávat soubory pouze s příponou csv nebo txt. O csv souborech byla již řeč v dřívějších kapitolách.

Nahraný soubor musí mít názvy atributů v prvním řádku. Dále pak je třeba, aby byly hodnoty oddělené čárkami, a poslední atribut musí být třída. V aplikaci není možné vybrat jakýkoliv atribut jako třídu. Poslední omezení pro soubor je takové, že chybějící hodnoty musí mít znak ve formě otazníku („?“). Jedině tak algoritmus pozná chybějící hodnotu. Pro txt soubory platí stejná pravidla.

Po načtení souboru se objeví v levém dolním rohu (pod tlačítkem uložit výsledky) počet atributů (sloupců) a počet instancí (řádků).

a) Načtení souboru do vektoru

Po načtení souboru se cesta objeví vedle tlačítka „Procházet…“. Cestu není možné nijak upravovat ani měnit. Pokud chce uživatel vybrat jiný soubor, musí opět kliknout na tlačítko. Zdrojový kód načítání do souboru je možné vidět níže (Obrázek 17).

```
1  void MainWindow::fillVector(std::string fileName)
2  {
3  std::string line, field;
4  std::vector<std::string> row;
5  std::ifstream file;
6  QMessageBox msgBox;
7
8  file.open(fileName.c_str());
9
10 if (file.is_open()) {
11       while ( getline(file, line) ) { //get next line in file
12             row.clear();
13             std::stringstream strm (line);
14             //break line into comma delimitted fields
15             while (getline(strm, field, ',')) {
16                   //add each field to the 1D array
17                   row.push_back(field);
18             }
19             //add the 1D array to the 2D array
20             Data.push_back(row);
21       }
22 } else {
23       msgBox.setText("Nelze načíst soubor");
24       msgBox.setIcon(QMessageBox::Critical);
25       msgBox.exec();
26 }
27 file.close();
28
29 //shuffles vector for better validation
30 std::random_shuffle(Data.begin()+1, Data.end());
31
32 //replace missing values for data
33 Data = handleMissingValues(Data);
34 }
```

Obrázek 17 Načtení souboru do 2D vektoru (Vlastní zpracování)

Funkce „fillVector“ plní dvojrozměrný vektor jednotlivými hodnotami ze souboru. Jako parametr dostane pouze název souboru, který načte uživatel. Hodnoty musí být odděleny čárkami, toto omezení je vidět na řádku patnáct. Po načtení se soubor zavře a poté se řádky vektoru zamíchají, aby se při náhodném výběru zlepšila validnost.

Na řádku 29 se volá funkce „handleMissingValues“, kterou si vysvětlíme v další podkapitole.

b) Zpracování chybějících hodnot

O zpracování chybějících hodnot se stará funkce „handleMissingValues“ (Obrázek 18). Jako parametr dostane celková data a celková data také vrací, ale bez chybějících hodnot. Funkce pracuje stejně jako filtr ReplaceMissingValues v systému Weka. Pro numerické atributy nahradí chybějící hodnotu průměrem

a pro nominální atributy nahradí hodnotu takovou hodnotou, která je v atributu nejčetnější.

Funkce nejprve zjistí, jestli atribut je numerický nebo nominální (řádek 9). Pokud je numerický, vypočítá průměr a nahradí příslušnou hodnotu. Pokud je nominální, spočítá nejčetnější hodnotu funkcí „returnOneClassFromMany“ a tuto hodnotu vloží místo hodnoty chybějící.

Proto po načtení souboru s větším počtem atributů a instancí může v aplikaci vzniknout menší odmlka.

```
1   std::vector < std::vector<std::string> > MainWin-
    dow::handleMissingValues(std::vector < std::vector<std::string>
    > Data) {

2   std::vector <std::string> column;
3   std::string maxS, str;
4   double pom, fill;

5   for (size_t i = 0; i < Data[0].size(); ++i) {
6    str = Data[1][i];
7    // converting string to vector of chars
8    std::vector<char> writable(str.begin(), str.end());

9    if (isdigit(writable[0])){ // if first char is digit
10   pom = sumColumn(Data, i);

11   for (size_t j = 0; j < Data.size(); ++j) {
12         if (Data[j][i] == "?") {
13         fill = pom / numberOfInstances();
14         std::ostringstream s;
15         s << fill;
16         std::string strn(s.str()); // converting double to string
17         s.flush();
18         // replacing missing values with mean
19         std::replace(Data[j].begin(), Data[j].end(), Data[j][i],
    strn);
20         }
21   }
22   } else {
23   for (size_t j = 1; j < Data.size(); ++j) {
24         if (Data[j][i] == "?") {
25         for (size_t k = 0; k < Data.size(); ++k) {
26         column.push_back(Data[k][i]);
27         }
28         maxS = returnOneClassFromMany(column);
29         for (size_t k = 0; k < Data.size(); ++k) {
30         // replacing missing values with instance with maximum
    count
31         std::replace(Data[k].begin(), Data[k].end(), Data[k][i],
    maxS);
32         }
33         column.clear();
34    }
35   }
```

```
36   }
37   }
38  return Data;
39  }
```

Obrázek 18 Chybějící hodnoty (Vlastní zpracování)

Zvládání chybějících hodnot je předmětem mnoha dohadů. Jak již bylo řečeno, Weka zvládá chybějící hodnoty s větší elegancí. Pro potřeby práce stačí, aby se chybějící hodnoty zvládaly způsobem stejným, jako to dělá filtr ve Wece.

Později v testovací fázi práce se tyto předpoklady budou ověřovat.

c) Diskretizace

Tato funkce převádí numerické atributy na nominální (Obrázek 19). Funkce je potřeba především pro stromy. Algoritmy založené na instancích přepisují numerické hodnoty vlastním způsobem. Algoritmus Naive Bayes pracuje s numerickými atributy pomocí statistických metod. Rozhodovací strom ID3 umí pracovat pouze s nominálními hodnotami.

Tím pádem byla zvolena metoda diskretizace pro zjednodušení u všech algoritmů, kromě algoritmů typu IBL. U těchto algoritmů se používá speciální metoda, která je vysvětlena později.

```
1. std::vector < std::vector<std::string> > MainWin-
   dow::discretize(std::vector < std::vector<std::string> > Data,
   int numberOfClasses) {
2.
3. std::vector <std::string> columnStr;
4. std::vector <double> column, pomVec, pomColumn;
5. std::string str, strData;
6. double min, max, pom, firstClass, difference, sumClass;
7.
8. for (size_t i = 0; i < Data[0].size(); ++i) {
9. str = Data[1][i];
10.    // converting string to vector of chars
11.    std::vector<char> writable(str.begin(), str.end());
12.
13.    if (isdigit(writable[0])){ //if first char is digit
14.        for (size_t j = 1; j < Data.size(); ++j) {
15.            strData = Data[j][i];
16.            //converting string to double
17.            pom = atof(strData.c_str());
18.            column.push_back(pom);
19.        }
20.    std::sort(column.begin(), column.end()); //sorting column
21.
22.    min = countMin(column);
23.    max = countMax(column);
24.
25.    // counting first class to divide instances
26.    firstClass = (min * (numberOfClasses - 1) +
   max)/numberOfClasses;
```

```
27.    // counting difference to divide other classes
28.    difference = firstClass - min;
29.    pomVec.push_back(firstClass);
30.    sumClass = firstClass;
31.    for (int l = 0; l < numberOfClasses-1; ++l) {
32.        // making all other classes
33.        sumClass = sumClass + difference;
34.        pomVec.push_back(sumClass);
35.    }
36.        // pushing classes to column
37.    for (size_t k = 0; k < pomVec.size(); ++k) {
38.        while (column[0] <= pomVec[k] && column[0] != co-
   lumn.back()) {
39.        pomColumn.push_back(pomVec[k]);
40.        column.erase(std::find(column.begin(), column.end(), co-
   lumn[0]));
41.        }
42.    }
43.    //input last member at the end of vector
44.    pomColumn.push_back(pomVec.back());
45.    for (size_t l = 0; l < pomColumn.size(); ++l) {
46.        std::ostringstream s;
47.        s << pomColumn[l];
48.        //converting double back to string
49.        std::string strn(s.str());
50.        columnStr.push_back(strn);
51.        s.flush();
52.    }
53.    for (size_t k = 0; k < columnStr.size(); ++k) {
54.        //replacing column with classes
55.        std::replace(Data[k+1].begin(), Data[k+1].end(), Da-
   ta[k+1][i], columnStr[k]);
56.    }
57.    pomColumn.clear();
58.    pomVec.clear();
59.    column.clear();
60.    columnStr.clear();
61.    }
62. }
63. return Data;
64. }
```

Obrázek 19 Diskretizace (Vlastní zpracování)

Funkce má dva parametry, první parametr jsou všechna data, která chce uživatel zdiskreditovat a druhý parametr je, kolik má být na výstupu tříd. Počet tříd se obecně volí jako deset. Uživatel aplikace Data Mining nemá možnost tento parametr měnit.

Podobně jako u předešlé funkce je třeba zjistit, který atribut je nominální. Pokud je atribut numerický, najde se maximum a minimum a poté se spočítá první třída podle vzorce (Vzorec 6). Poté se vypočítá rozdíl (řádek 28) mezi minimem a touto hodnotou a dopočítají se prahy zbývajících tříd (řádky 31-35). Nakonec se určí, do které ze tříd patří reálná hodnota v souboru a nahradí se příslušnou třídou.

Jako poslední práci se souborem si uvedeme standardizaci u algoritmů založených na instancích (IBL).

d) Standardizace

Jak již bylo dříve zmíněno, používá se tato metoda u algoritmu IB1 a IBk. Vzorec pro výpočet jednotlivých hodnot pro numerické atributy je:

$$X_i = \frac{X_i - min}{\max - min}$$

Vzorec 8 Standardizace

Metoda, jak je vidět ze vzorce, je založena na minimu a maximu. Pro každý atribut vypočítáme minimum a maximum a poté pro jednotlivé hodnoty počítáme výslednou hodnotu podle vzorce. Tato hodnota je vždy mezi 0-1. (Sadawi, 2014)

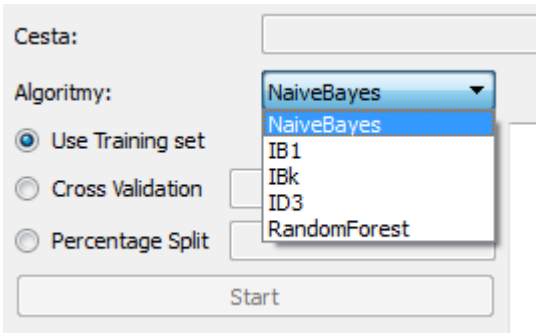
Zdrojový kód této metody je k nahlédnutí v příloze A. Zároveň je touto metodou uzavřena kapitola o souborech a práci s nimi.

5.3 Algoritmy aplikace

Všechny použité algoritmy již byly zmíněny v začátcích práce. Principy těchto algoritmů byly dodrženy i v aplikaci Data Mining. Zde si tyto algoritmy představíme pouze s hlediska ovládání, vstupů a výstupů.

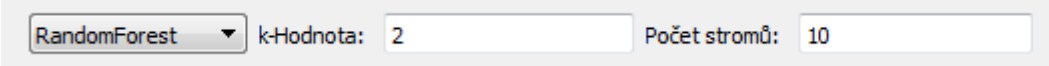
5.3.1 Výběr algoritmu

Výběr algoritmu je v aplikaci velice jednoduše vytvořen pomocí „QComboBoxu“ (Obrázek 20). V tomto widgetu si uživatel jednoduše vybere algoritmus, který chce použít. Funkce widgetu předává v parametru název algoritmu ve formě řetězce.



Obrázek 20 Výběr algoritmu (Vlastní zpracování)

Některé algoritmy mají parametry. Pokud má funkce parametry, ukáží se vedle výběru algoritmu (Obrázek 21). Parametry mají pouze algoritmy IBk a Random Forest.



Obrázek 21 Parametry algoritmu (Vlastní zpracování)

Parametry uživatel jednoduše zadá do příslušné kolonky. Omezení parametrů je zvládnuto vestavěnou funkcí pro „QLineEdit“ (Obrázek 22). Validátor je nastaven na maximální hodnotu rovnu stem, jelikož se nepředpokládá větší počet sousedů pro výpočet. Podobně bylo uvažováno i u ostatních parametrů jednotlivých algoritmů. Minimum je vždy jedna.

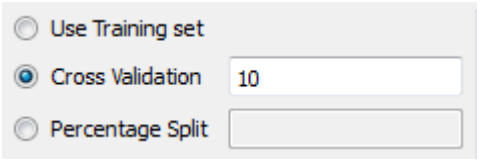
```
1. ui->KnnOrKvalue->setText("Počet Sousedů: ");
2. ui->KnnOrKvalueEditLine->setText("3");
3. //determining min and max value to be written in the line
4. ui->KnnOrKvalueEditLine->setValidator( new QIntValidator(1, 100,
   this) );
```

Obrázek 22 Validátor vstupu (Vlastní zpracování)

V dalším kroku si uživatel vybere, jakým způsobem chce rozdělit celková data na testovací a tréninková množinu.

5.3.2 Výběr algoritmu pro rozdělení

Pro výběr algoritmu pro rozdělení byl zvolen jednoduchý výběr z několika variant pomocí „QRadioButton“ (Obrázek 23).

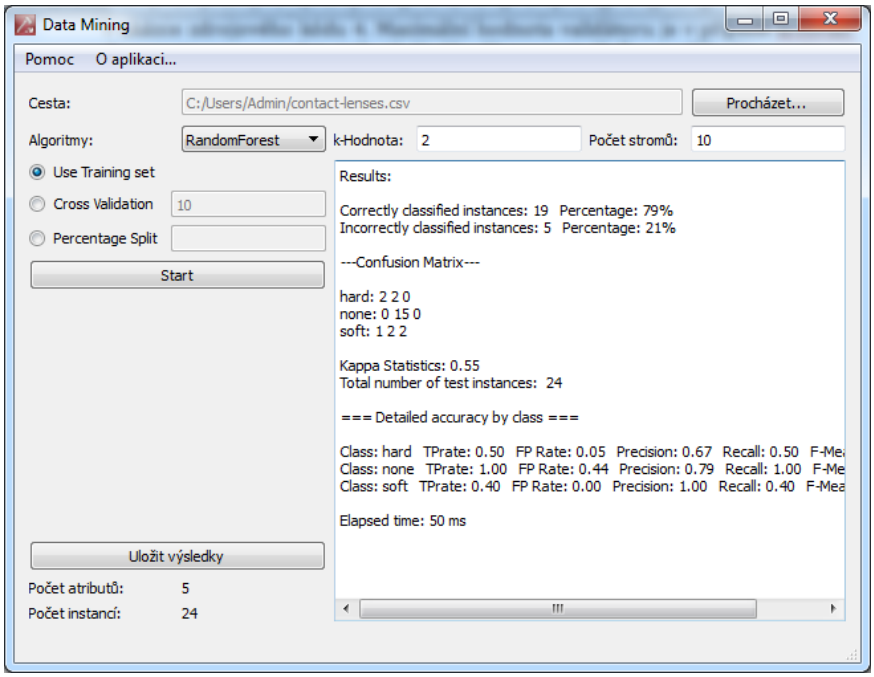


Obrázek 23 Algoritmy rozdělení (Vlastní zpracování)

I v tomto případě je možné zadat parametry a také zde je použit validátor jako v obrázku 22. Maximální hodnota validátoru je v případě metody Cross-validation nastavena na počet atributů. V případě procentuálního rozdělení je maximum sto. I v této části je minimum vždy jedna. Jednotlivé zdrojové kódy těchto funkcí jsou zobrazeny v příloze B.

5.3.3 Start a výsledky

Po načtení souboru, výběru algoritmu, vypsání požadovaných parametrů a výběru rozdělení datové sady, může uživatel kliknout na tlačítko Start a tím spustit trénování a testování vybraným algoritmem. Dle velikosti souboru a složitosti případného algoritmu mohou výpočty trvat i několik desítek minut. Dobu trvání můžeme vidět v posledním řádku výsledků (Elapsed time).



Obrázek 24 Start a výsledky aplikace (Vlastní zpracování)

Výstup je podobný jako v aplikaci Weka (Obrázek 24). Jsou zde základní charakteristiky i některé složitější. Základem jsou správně klasifikované instance a jejich procentuální vyjádření. Stejně tak i nesprávně klasifikované instance. Dále je zde Confusion Matrix i detailní přesnosti jednotlivých tříd (Detailed accuracy by class). Nakonec již zmiňovaný čas.

Uživatel si může tento výstup uložit a později se na něj podívat. Po kliknutí na tlačítko „Uložit výsledky“ si uživatel zvolí umístění souboru a soubor jednoduše uloží. Soubor se ukládá ve formátu txt.

5.4 Převod algoritmů

Výše bylo popsáno grafické uživatelské rozhraní a práce se soubory v aplikaci. Nyní bude probírán převod z programovacího jazyka Java do jazyka C++.

Převod mezi jednotlivými jazyky nikterak jednoduchý. Je zde spousta společných věcí (syntaxe, některé jednoduché funkce, datové typy, aj.), ale také spousta rozdílů (převážně ve složitějších datových typech, dědičnosti a formulaci tříd), které byli problémem při převádění jednotlivých algoritmů.

Existují i různé převaděče kódu, ale ty umí převést pouze syntaxi a vůbec nezvládají převést tak složité zdrojové kódy jako jsou v systému Weka.

Nedá se předpokládat, že si programátor zkopíruje kód v jazyce Java a přepíše pár funkcí, instrukcí a dostane výsledný kód v jazyce C++. Zvláště při tak rozsáhlém systému, jakým Weka je. Zdrojové kódy algoritmů Weka jsou sice transparentní, ale jsou nesmírně rozsáhle a složité. I přes skvělou dokumentaci bylo někdy opravdu těžké najít kýžený algoritmus a jeho funkce. Navíc u spousty tříd je abs-

trahováno od přímého kódu algoritmu a jsou zde vytvářeny obecné třídy pro vícero použití. Jednotlivé algoritmy nejsou závislé na jedné třídě, ale na spoustě tříd, které jsou závislé na dalších třídách. Taková provázanost jednotlivých tříd je také překážkou pro převod. Toto je dáno vyspělostí systému Weka a také dlouholetým vývojem zdrojových kódů.

Transparentní zdrojové kódy jsou velmi výhodné pro uživatele, kteří používají knihovnu Weka ve svých aplikacích v jazyce Java. Zde je velice jednoduché používat funkce knihovny Weka pouhým voláním. Uživatel této knihovny často ani nemusí vědět, jak jsou jednotlivé funkce napsány ve zdrojovém kódu, stačí vědět, co dělají, a jak se s nimi pracuje. Toto je vše přehledně popsáno v dokumentaci knihovny Weka. Převážně proto jsou kódy této knihovny transparentní.

Převod jednotlivých algoritmů probíhal na základě zdrojových kódu knihovny Weka a také na základě principů pro jednotlivé algoritmy popsaných na začátku práce. Zdrojové kódy aplikace byly použity především pro pochopení principů algoritmů a jejich jednotlivých funkcí. Také byly použity jako inspirace pro psaní zdrojových kódů aplikace práce. Spolu s obecnými principy vznikly zdrojové kódy této práce.

Jak již bylo řečeno, přesný přepis není možný. Právě proto se mohou lišit některé výsledky nebo prostorové a časové nároky algoritmů. Experimenty budou provedeny v následující kapitole.

6 Experimenty

Kapitola s názvem Experimenty je nejdůležitější a stěžejní kapitolou práce. Zde budou porovnány výstupy z Weky s výstupy aplikace Data Mining.

Celá aplikace Data Mining byla vytvořena s hypotézou, která je popsána v cíli práce. Tato hypotéza se bude zkoumat blíže v této části práce. U projektů zaměřených na dolování z dat je důležité se zaměřit především na prostorovou (paměťovou) a časovou náročnost aplikace.

Důvody pro tyto složitosti jsou jasné, Weka je vytvořena v Jave a aplikace Data Mining v C++. Tím pádem budou obě složitosti jistě rozdílné. Která aplikace má lepší výsledky, bude představeno právě v této kapitole.

Dále je nutné otestovat správnost výsledků. Výsledky se mohou mírně lišit, jelikož většina algoritmů je založena na náhodném rozdělení. Celkově by výsledky z aplikace Data Mining měly zhruba odpovídat výsledkům ze systému Weka.

Nejprve je nutné otestovat oba filtry a to, jestli na větších souborech vykazují velké rozdíly mezi výslednými testovanými instancemi. Důvody pro testování těchto filtrů jsou vcelku jasné. Jak již bylo řečeno, tak u aplikace Data Mining jsou tyto filtry použity ihned po nahrání souboru do paměti. Na druhou stranu Weka aplikace tyto filtry nepoužívá a u každého algoritmu zastupuje práci filtrů nějaká jiná metoda. Proto je důležité dokázat, zda aplikování filtrů na soubor v systému Weka vykazuje vzdáleně jiné hodnoty než bez aplikovaných filtrů.

Dále je třeba říci, že Weka 3. 6. 12 je vyvíjena 15 let[10] experty z oboru a aplikace Data Mining byla vytvořena za méně jak půl roku. To může mít velký vliv na celkové měření jednotlivých experimentů. Proto budou některé experimenty testovány i na starších verzích Weky (konkrétně 3. 6. 3), která je také postavena na 64bitové architektuře. Důvod pro tento předpoklad je jednoduchý. Weka se neustále vyvíjí, tím pádem je stále vylepšována a optimalizována. Zatímco aplikace Data Mining je v zárodku a vyvíjí se pouze pro potřeby práce.

6.1 Podmínky experimentů

Pro experimenty byla vybrána databáze census-income. Tato databáze obsahuje více jak 200 000 instancí. Instance obsahují chybějící hodnoty, numerické i nominální atributy. Data byla sbírána mezi lety 1994 a 1995 a obsahují průzkum americké populace. (Census-Income Database, 1997)

Tento soubor dat byl patřičně upraven pro potřeby experimentů a také zredukován. Je zbytečné testovat tak velký soubor dat. Počet atributů zůstal stejný a to 42. Instance byly vybrány náhodným výběrem a byly rozděleny do souborů o 500, 1 000, 2 000 a 5 000 instancí. Tyto soubory podlehnou všem dále zmíněným experimentům.

[10] Weka se vyvíjí od roku 1999 a poslední stabilní verze 3. 6. 12 byla vydána v roce 2014.

Testovány budou celkem tři algoritmy: Naive Bayes, IBk a Random Forest. Zbývající algoritmy nejsou tak důležité, aby se zúčastnili experimentů. Tyto algoritmy byly implementovány jako základ pro další algoritmy, tudíž nemají v této práci takovou důležitost. Algoritmus ID3 pracuje pouze s nominálními daty a algoritmus IB1 je stejný jako algoritmus IBk, jen pracuje s jedním nejbližším sousedem. V praxi se optimální počet sousedů hledá prováděním experimentů, ale ve většině případů je tato hodnota vyšší jak jedna.

Nastavení algoritmů bude stejné pro všechny případy. Budou použity všechny algoritmy pro rozdělení a jednotlivě představeny. Pro IBk budou použiti tři sousedi a pro Random Forest bude použito 100 stromů a k-hodnota rovna třem.

Pro testování není použit žádný superpočítač. Bude použit obyčejný počítač (specifikace nehraje velkou roli, jelikož aplikace Data Mining i Weka budou testovány na stejném počítači). Proto také není možné mít v testované množině tisíce instancí a desítky atributů.

Všechny výše zmíněné algoritmy budou testovány na systému Weka 3. 6. 12 a operačním systému Windows 7 Professional s 64bitovou architekturou procesoru. Některé experimenty, jak již bylo řečeno, budou prováděny na systému Weka 3. 6. 3 (vydán v roce 2010).

6.2 Filtry

Filtry se budou testovat pouze v systému Weka. V tomto případě je třeba otestovat to, jestli se nějak zásadně liší hodnoty s aplikovanými filtry od těch, které tyto filtry nemají. Nejprve budou otestovány data bez filtrů, poté bude aplikován filtr ReplaceMissingValues a na tyto data bude aplikován filtr Discretize.

Pozorován bude čas a to, jak algoritmus otestoval jednotlivé instance. Výsledky jsou prezentovány v tabulce níže. Tam jsou vidět jednotlivé časy v sekundách a také počty správně a nesprávně klasifikovaných instancí.

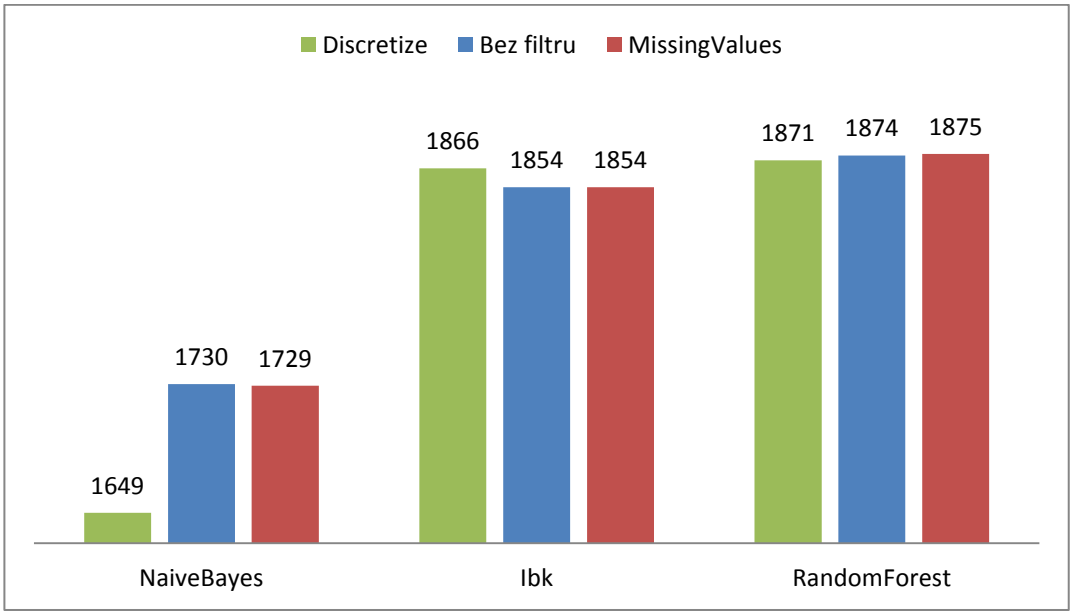
Tabulka 4 Testování filtrů (Vlastní zpracování)

Počet instancí	500			1 000			2 000		
bez filtru	čas (s)	správně	nesprávně	čas (s)	správně	nesprávně	čas (s)	správně	nesprávně
Naive Bayes	0,9	439	63	1,0	916	83	1,30	1730	271
IBk	0,8	453	49	1,8	936	63	5,25	1854	147
Random Forest	2,7	459	43	4,5	945	54	9,50	1874	127
MissingValues	čas (s)	správně	nesprávně	čas (s)	správně	nesprávně	čas (s)	správně	nesprávně
Naive Bayes	0,9	441	61	0,6	916	83	1,10	1729	272
IBk	0,8	453	49	1,5	938	61	3,80	1854	147
Random Forest	2,5	458	44	4,0	945	54	7,50	1875	126
Discretize	čas (s)	správně	nesprávně	čas (s)	správně	nesprávně	čas (s)	správně	nesprávně
Naive Bayes	1,2	455	47	0,6	871	128	0,80	1649	352
IBk	1,0	441	61	1,5	945	54	3,80	1866	135
Random Forest	1,6	459	43	3,5	944	55	6,50	1871	130

Při pohledu na čas je zřejmé, že se celkový čas postupně zmenšuje aplikováním filtrů. Nejvíce u algoritmu Random Forest. To je způsobeno tím, že při aplikování filtrů nemusí algoritmy používat ty metody, které jednotlivé filtry zastupují. Tímto se výpočet zjednodušuje a čas krátí.

Co se instancí týče, u IBk a Random Forest jsou téměř identicky klasifikovány instance při filtrech i bez něj. Je patrné mírné zlepšení při klasifikaci u obou algoritmů. U algoritmu Naive Bayes je klasifikátor méně účinný při aplikování filtrů. Především po aplikování filtru Discretize. Toto je způsobeno tím, že algoritmus Naive Bayes zvládá práci s numerickými atributy lépe, než jak pracuje s numerickými atributy filtr Discretize. Při dvou tisících atributech se zde jedná o téměř 200 instancí, což odpovídá 10 % instancí.

Pozornost by se měla ubírat především k oblasti, kde bylo testováno 2 000 instancí. Zde jsou rozdíly nejmarkantnější. Rozdíly jsou ukázány v grafu:



Obrázek 25 Filtry – 2 000 instancí (Vlastní zpracování)

Z obrázku je patrné, že při dvou tisících instancích nejsou rozdíly tak výrazné, jak by se mohlo zdát. Tím pádem není problém v používání těchto filtrů ihned na začátku spuštění každého algoritmu. Pro algoritmus Naive Bayes by se zde dalo uvažovat nad tím, vytvořit pro tento algoritmus práci s numerickými atributy bez filtru Discretize.

6.3 Experimenty algoritmů

Zde se především zaměříme na testování algoritmů z hlediska časového, prostorového a také z hlediska správnosti výsledků. Testovány budou opět pouze tři stěžejní algoritmy práce. Nastavení těchto algoritmů zůstává stejné.

Testování bude probíhat s aplikovanými filtry na všechny algoritmy. Nejprve budou testovány algoritmy na základě správnosti výsledků, poté časové a nakonec paměťové složitosti jednotlivých algoritmů.

6.3.1 Experimenty správnosti

Zde budou zobrazeny výsledky testování správnosti algoritmů. To znamená, jak se liší výstupy ze systému Weka s výstupy z aplikace Data Mining.

Tento experiment ukáže, jestli jsou algoritmy správně implementovány a zda vykazují alespoň podobné výsledky jako systém Weka.

Testovány budou postupně různé algoritmy pro rozdělení dat (Use training set, Percentage split a Cross-validation), pro ověření správnosti jednotlivých klasifikátorů právě po aplikování těchto metod rozdělení.

Všechny výsledky budou přehledně zobrazeny v tabulce a patřičně okomentovány. U důležitějších hodnot experimentů budou zobrazeny i grafy.

Tabulka 5 Správnost – Use training set (Vlastní zpracování)

Use training set							
		správně	nesprávně	správně	nesprávně	správně	nesprávně
	Počet instancí	500		1 000		2 000	
Weka 3. 6. 12	Naive Bayes	416	86	884	115	1549	452
	IBk	465	37	952	47	1916	85
	Random Forest	502	0	999	0	2000	1
Data Mining	Naive Bayes	400	102	820	179	1711	290
	IBk	487	15	972	27	1971	30
	Random Forest	466	36	951	48	1872	129

Jak je z tabulky 5 patrné, tak Weka vykazuje ve většině případů mírně lepší výsledky. Jedině při IBk jsou výsledky lepší v aplikaci Data Mining.

Celkově by se dalo říci, že při používání celých tréninkových dat pro testování, vykazuje aplikace Data Mining přijatelné výsledky.

Nyní se podíváme na metodu rozdělení dle procent (použito bude 66 %).

Tabulka 6 Správnost – Percentage split (Vlastní zpracování)

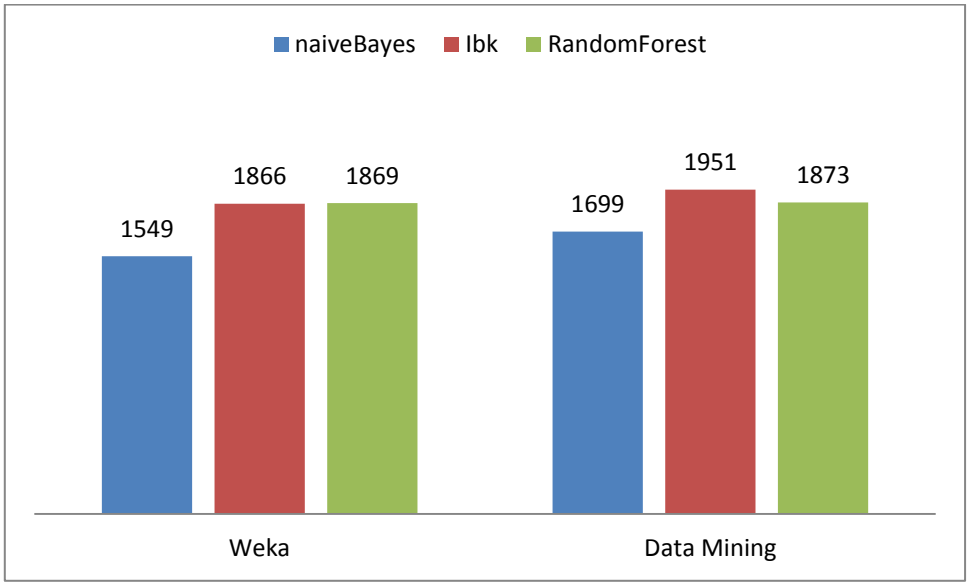
Percentage split							
		správně	nesprávně	správně	nesprávně	správně	nesprávně
	Počet instancí	50		1 000		2 000	
Weka 3. 6. 12	Naive Bayes	139	32	308	38	519	161
	IBk	144	27	323	17	634	46
	Random Forest	155	16	323	17	640	40
Data Mining	Naive Bayes	123	47	282	57	575	105
	IBk	164	6	327	12	662	18
	Random Forest	157	13	323	16	630	50

V tabulce 6 jsou velmi podobné hodnoty jako v předchozí tabulce. Správnost výsledků je i při procentuálním rozdělení tréninkových a testovacích dat podobná. Zde se výsledky podobají dokonce více než v tabulce předchozí.

Tabulka 7 Správnost – Cross-validation (Vlastní zpracování)

Cross-validation							
		správně	nesprávně	správně	nesprávně	správně	nesprávně
	Počet instancí	500		1 000		2 000	
Weka 3. 6. 12	Naive Bayes	408	94	871	128	1549	452
	IBk	455	47	945	54	1866	135
	Random Forest	460	42	946	53	1869	132
Data Mining	Naive Bayes	417	85	825	174	1699	302
	IBk	483	19	965	34	1951	50
	Random Forest	463	39	957	42	1873	128

Cross-validation je jedna z častěji používaných metod pro rozdělení dat. V tabulce 7 to vypadá tak, že s přibývajícími instancemi roste přesnost aplikace Data Mining. Celkově se dá říci, že metoda v aplikaci Data Mining má podobné nebo lepší výsledky než systém Weka. Proto si výsledky pro dva tisíce instancí zobrazíme pomocí grafu.



Obrázek 26 Správnost 2 000 instancí (Vlastní zpracování)

Jak je z obrázku 26 patrné, metoda vykazuje velmi podobné výsledky pro obě aplikace. Dále jsou výsledky procentuálně podobné (mírně lepší) dvěma předchozím metodám. Toto je dáno tím, že algoritmy jsou napsány stejně a mění se pouze testovací a tréninková data. Je jasné, že je vše založeno na náhodném rozdělení, pro další experimenty by se hodnoty mírně lišili, ale dá se říci, že hodnoty obou aplikací vykazují velmi podobné výsledky.

Tímto bylo dokázáno, že všechny algoritmy v aplikaci Data Mining jsou dobře implementovány, jelikož vykazují podobné výsledky jako systém Weka. Tím pádem je možné se posunout dále k důležitějším částem experimentů a tím jsou experimenty časového a prostorového charakteru.

6.3.2 Časové experimenty

Zde se ověří správnost hypotézy, která byla představena v cílech práce. Časy načítání souborů nebudou v této kapitole představeny. I soubory v řádech desítek MB se otvírají pár vteřin v obou aplikacích. Je to tedy zbytečné.

Časové experimenty budou opět prováděny pouze na třech stěžejních algoritmech práce. Algoritmy budou, stejně jako v předchozích případech, testovány pro všechny typy rozdělení a budou také stejně nastaveny. V této části přibudou experimenty i s 5 000 instancemi.

Při testovaní algoritmu Random Forest v aplikaci Data Mining bylo u většího počtu instancí počítáno s 10 stromy a poté výsledný čas vynásoben deseti. V tabulkách pro 5 000 instancí jsou časy pro Random Forest extrémně vysoké a trvalo by značnou dobu otestovat 100 stromů.

Nyní k jednotlivým výsledkům:

Tabulka 8 Čas – Use traning set (Vlastní zpracování)

Use training set					
		Čas (s)			
	Počet instancí	500	1 000	2 000	5 000
Weka 3. 6. 12	Naive Bayes	0,3	0,5	0,5	1,0
	IBk	1,0	1,5	4,0	23,0
	Random Forest	1,0	1,5	2,0	3,0
Data Mining	Naive Bayes	1,8	5,0	15,0	57,0
	IBk	5,0	24,0	117,0	656,0
	Random Forest	289,0	1370,0	6570,0	40020,0

Tabulka 8 ukazuje, jaký čas byl potřeba k provedení jednotlivých algoritmů na různých velikostech dat. Z těchto měření jasně vyplívá, že Weka je jednoznačně rychlejší při použití tréninkových dat jako testovacích. U aplikace Data Mining je jediný obstojný algoritmus Naive Bayes. Ostatní dva algoritmy vykazují mnohonásobně delší časy než systém Weka.

Tabulka 9 Čas – Percentage split (Vlastní zpracování)

Percentage split					
		Čas (s)			
	Počet instancí	500	1 000	2 000	5 000
Weka 3. 6. 12	Naive Bayes	0,2	0,3	0,5	0,5
	IBk	0,6	0,8	1,0	5,0
	Random Forest	0,8	1,0	3,0	6,0
Data Mining	Naive Bayes	0,5	1,0	2,0	10,0
	IBk	0,6	2,0	10,0	78,0
	Random Forest	137,0	300,0	1260,0	10390,0

Z tabulky 9 je patrné, že při procentuálním rozdělení dat vykazuje aplikace Data Mining mnohem lepší výsledky než při předchozím rozdělení dat. Stále je při větším souboru několikanásobně pomalejší než systém Weka. Zrychlení je způsobeno menším počtem testovaných instancí.

Zde stojí za zmínku i algoritmus IBk, který má zde obstojnější časy, než v případě použití tréninkových dat jako testovacích. Algoritmus Random Forest má sice lepší výsledky než v předchozím případě, ale celkově je čas stále příliš pomalý.

Tabulka 10 Čas – Cross-validation (Vlastní zpracování)

Cross-validation					
		Čas (s)			
	Počet instancí	500	1 000	2 000	5 000
Weka 3. 6. 12	Naive Bayes	0,3	0,5	1,0	2,0
	IBk	1,0	2,0	3,0	19,0
	Random Forest	2,0	3,0	6,0	17,0
Data Mining	Naive Bayes	1,0	4,0	9,0	31,0
	IBk	1,0	3,0	13,0	72,0
	Random Forest	1400,0	5560,0	20280,0	51860,0

Jako poslední zbývá představit tabulku 10. Při nižších datových sadách vykazuje aplikace Data Mining uspokojivé výsledky. Při větších sadách je opět Weka mnohonásobně rychlejší, i když rozdíly nejsou tak markantní, vyjma algoritmu Random Forest. Tento algoritmus je v aplikaci Data Mining extrémně pomalý při všech metodách rozdělení.

Celkově by se dalo říci, že algoritmy Naive Bayes i IBk obstojně soupeří se systémem Weka. Algoritmus Random Forest vykazuje vysoké časy a při větším počtu instancí je téměř nepoužitelný z časového hlediska. Při dalším vývoji by bylo vhodné se na tento algoritmus zaměřit a optimalizovat jej.

Dalším experimentem bude určitě zajímavé vyzkoušet časy, které jsou potřebné k naučení se z datové sady a časy, které jsou potřebné k testování dat v aplikaci Data Mining. Z výše zmíněných výsledků pro různé algoritmy rozdělení dat to vypadá, že celkově mnohem déle trvá testování nežli trénování. Při použití všech dat na testování jsou časy delší než při procentuálním rozdělení. Toto si ověříme následujícím experimentem, kde budeme sledovat zvlášť časy pro trénování a časy pro testování.

Tento experiment bude sledován pouze pro 2 000 instancí. Argumenty jednotlivých algoritmů budou stále stejné.

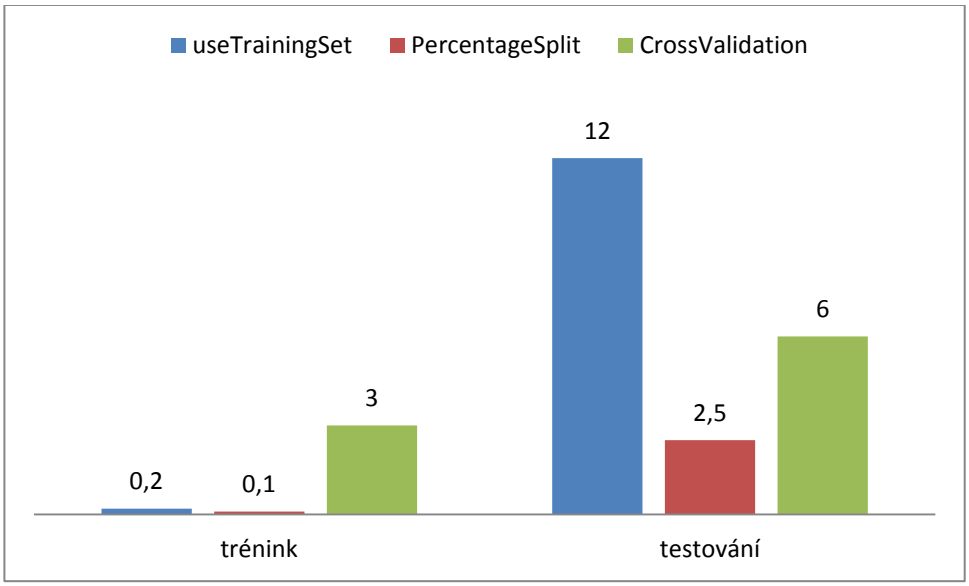
Tabulka 11 Čas tréninků a testování (Vlastní zpracování)

DataMining	Use training set		Percentage split		Cross-validation	
n = 2000	Čas (s)					
	Trénink	testování	trénink	testování	trénink	testování
Naive Bayes	0,2	12,0	0,1	2,5	3,0	6,0
IBk	0,3	108,0	0,3	10,0	4,0	9,0
Random Forest	3260,0	2430,0	1140,0	260,0	18840,0	1960,0

Tabulka 11 potvrzuje domněnku z předchozího textu. Celkové časy testování jsou opravdu delší než časy trénování. V případě algoritmu Random Forest je tato skutečnost naopak. Více o jednotlivých algoritmech v dalším textu.

Pro všechny algoritmy z tabulky 11 si vytvoříme graf pro lepší vizualizaci. Oba grafy znázorňují časy trénování a testování. Trénováním je myšleno vytvoření kla-

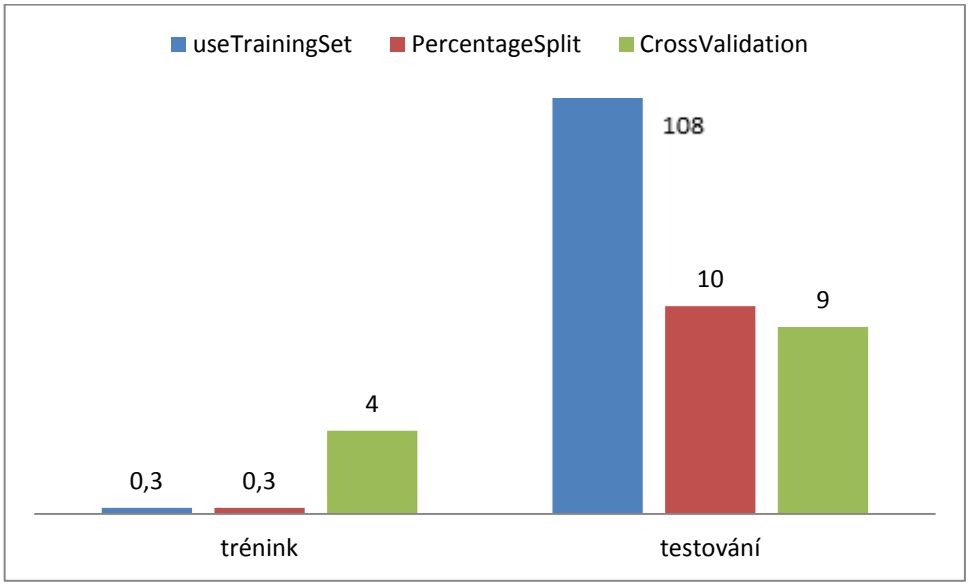
sifikátoru z tréninkových dat a testováním je myšleno otestování takto vytvořeného klasifikátoru na testovacích datech.



Obrázek 27 Naive Bayes (Vlastní zpracování)

Výše uvedený obrázek znázorňuje časy trénování a testování pro algoritmus Naive Bayes. V grafu je jasně vidět, že trénování zabere výrazně kratší dobu, než testování.

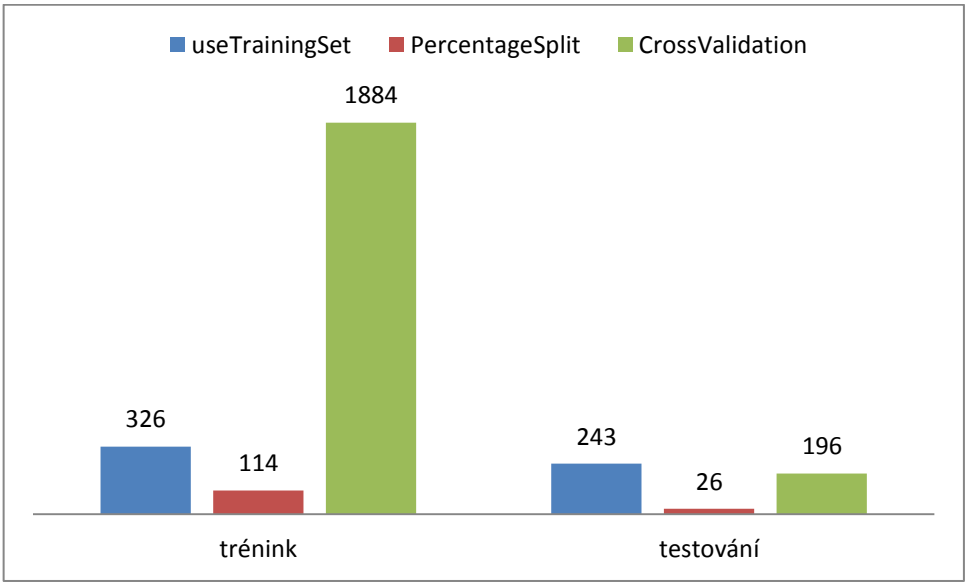
Toto je způsobeno nejspíše tím, že v implementaci algoritmu je trénování pouze počítání pravděpodobností jednotlivých instancí. V testování musí algoritmus projít všechny hodnoty jednotlivých instancí. Tyto hodnoty vynásobit pravděpodobnostmi, poté znormalizovat a nakonec vybrat třídu s největší pravděpodobností. Tím pádem čas pro testování bude logicky delší.



Obrázek 28 IBk (Vlastní zpracování)

Druhý obrázek (Obrázek 28) této části, má podobné výsledky jako graf předchozí. I v tomto případě je trénování mnohem rychlejší než testování.

Algoritmus IBk ve fázi trénování pouze upravuje data, aby byla lépe spočetná pro jednotlivé vzdálenosti. Je zde logické, že tato úprava zabere méně času než testování. V případě testování se musí nejprve spočítat všechny vzdálenosti a poté jednotlivé vzdálenosti porovnat. Pro algoritmus Use training set je třeba spočítat tyto vzdálenosti pro každou instanci a tím pádem je tento čas největší.



Obrázek 29 Random Forest (Vlastní zpracování)

Poslední obrázek (Obrázek 29) ukazuje časy pro Random Forest. Zde je to přesně naopak, než jak tomu bylo u algoritmu výše. Trénování je časově mnohem náročnější než testování. To je dáno tím, že vytváření většího množství stromů je časově náročnější, než procházení stromů a porovnávání jednotlivých prvků.

Při pohledu na tyto grafy se víceméně potvrzuje, že trénování je časově náročnější oproti testování. Toto je zcela jistě pozitivní, jelikož trénování většího počtu instancí vykazuje dobré časy. V budoucím vývoji aplikace by se tedy mělo zapracovat na testování jednotlivých instancí a snížit výpočetní složitost jednotlivých algoritmů.

Celkově při pohledu na grafy i tabulky je zřejmé že systém Weka 3. 6. 12 je, co se výpočetního času týče, lepším řešením. Toto je způsobeno celkovou vyspělostí systému Weka. Roky optimalizace všech algoritmů a optimalizace práce se soubory vedou k opravdu dobré časové náročnosti. Starší verze Weky (Weka 3. 6. 3) vykazuje o něco horší výsledky pro časovou složitost, než novější systém. Stále je i starší verze o mnoho rychlejší než aplikace Data Mining. U aplikace Data Mining byla optimalizace velice krátká a určitě by stálo za to, optimalizovat kód déle. Více o tomto tématu v kapitole Možnosti rozšíření.

6.3.3 Prostorové (paměťové) experimenty

Jako první bylo třeba vyzkoušet jak velké soubory aplikace Data Mining zvládne načíst do paměti. Také pak to jestli se nějak velikost těchto souborů liší od systému Weka.

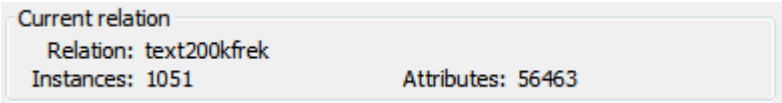
Pro tento experiment byl vybrán soubor, který obsahuje náhodný výběr z dvou milionů textových dokumentů. V tomto souboru jsou atributy reprezentovány jednotlivými slovy v dokumentu a instance četnosti jednotlivých slov. Celkový soubor obsahuje miliony instancí a desetitisíce atributů a jeho velikost je zhruba 22 GB. Pro potřeby práce byl soubor zredukován na několik MB.

Po několika experimentech bylo dosaženo maximální velikosti souborů, které mohou obě aplikace zvládnout.

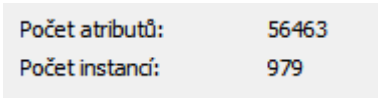
Tabulka 12 Maximální velikosti souborů (Vlastní zpracování)

	Weka 3. 6. 12	Data Mining	Weka 3. 6. 3
Velikost (b)	120 240 238	110 668 469	20 864 468

V tabulce 12 je zobrazen i systém Weka 3. 6. 3, který zvládne soubor pouze o velikosti 20 MB. Je zřejmé, že tato verze Weky nezvládá tak velké soubory jako verze novější. Je zde tedy vidět, jak velká je optimalizace paměti v systému Weka za čyřleté období vývoje.



Obrázek 30 Weka (Vlastní zpracování)



Obrázek 31 Data Mining (Vlastní zpracování)

Weka 3. 6. 12 (Obrázek 30) zvládá práci s pamětí o něco lépe než aplikace Data Mining (Obrázek 31). Celkově se zde jedná o zhruba 70 instancí a rozdílu ve velikosti souboru 10 MB, což při tak obrovském souboru není nijak velký rozdíl.

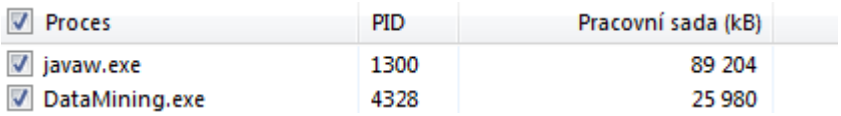
Velká nevýhoda v systému Weka je, že tento systém po načtení velkého souboru ponechá nějakou část tohoto souboru v paměti i po načtení dalšího souboru. Nejspíše pro další použití. Toto je problém, když chce uživatel načíst a odzkoušet více velkých souborů. Aplikace Data Mining při načtení nového souboru automaticky vyčistí paměť a nový soubor načte celý.

Dále se budou testovat jednotlivé algoritmy se soubory pouze se 2 000 a 5 000 instancemi. Soubory s 500 a 1 000 instancemi jsou zde vynechány, jelikož takto malé soubory se v paměti téměř neprojeví.

Také je potřeba říci, že měřit pamět přesně je velmi obtížné, tudíž v následujících experimentech budou hodnoty přibližné.

Měřit se bude pomocí zabudované aplikace ve Windows 7, která se nazývá Sledování prostředků. Do této aplikace se uživatel dostane pomocí Správce úloh nebo z Ovládacích panelů. V této aplikaci nás zajímá záložka Paměť. Aplikace měří různé charakteristiky paměti. Pro potřeby práce postačí sloupec s názvem procesu a poté sloupec Pracovní sada (kB). Tento sloupec ukazuje velikost fyzické paměti (RAM) v kB, kterou proces právě používá.

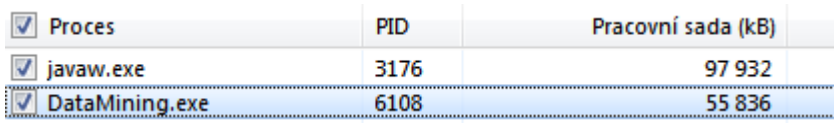
Nejprve je nutné změřit pamět ihned po zapnutí obou aplikací. Poté bude měřena změna po načtení souborů o velikosti 5 000 instancí a 42 atributů. Nebudou aplikovány žádné filtry.



Proces	PID	Pracovní sada (kB)
javaw.exe	1300	89 204
DataMining.exe	4328	25 980

Obrázek 32 RAM používaná procesem po spuštění (Vlastní zpracování)

Na obrázku 32 je zobrazeno, kolik paměti proces zabírá ihned po spuštění. U systému Weka po spuštění aplikace Explorer. Aplikace Data Mining vykazuje více jak třikrát méně používané paměti, což je určitě velká úspora. Toto je dáno tím, že pro systém Weka se musí spustit Java virtual machine, která zabírá spoustu paměti sama o sobě. Také proto proces nese název „javaw.exe“.



Proces	PID	Pracovní sada (kB)
javaw.exe	3176	97 932
DataMining.exe	6108	55 836

Obrázek 33 RAM používaná procesem po načtení 5000 instancí (Vlastní zpracování)

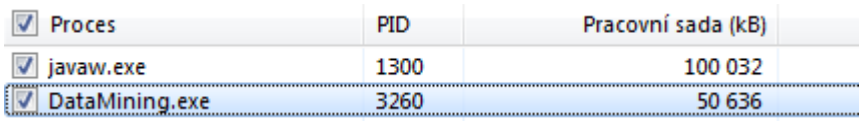
Na dalším obrázku (Obrázek 33) je, kolik paměti proces potřebuje po načtení 5 000 instancí. Souboru nijak velkého, ale je vidět, že pro každou aplikace zabere

soubor část paměti. Konkrétně v sytému Weka zabírá kolem 9 MB a v aplikaci Data Mining 30 MB.

Toto koresponduje s experimentem na začátku této kapitoly. Weka zvládá načíst větší soubory do paměti, tím pádem má lepší práci s pamětí, než aplikace Data Mining.

Na druhou stranu po načtení souboru o velikosti 5 000 instancí je celková konzumace paměti menší u aplikace Data Mining.

Velkou nevýhodou systému Weka je, jak již bylo dříve zmíněno, že soubory z paměti nevypouští, ale část z nich ponechává v paměti. Toto může být problém při testování větších souborů a následnému zahlcení paměti.



Proces	PID	Pracovní sada (kB)
javaw.exe	1300	100 032
DataMining.exe	3260	50 636

Obrázek 34 RAM používaná po dalším načtení (Vlastní zpracování)

Po dalším načtení souboru o velikosti 2 000 instancí se paměť systému Weka navýší, jak je patrno z obrázku 34. U aplikace Data Mining se naopak konzumace paměti sníží. Toto dokazuje nevýhodu práci s pamětí u systému Weka. Kdyby uživatel stále načítal další a další soubory, tak systém Weka vypíše chybovou zprávu a zastaví proces.

Nyní přejdeme k testování jednotlivých algoritmů a konzumaci paměti. Parametry testování budou stejné jako v celé kapitole Experimenty.

Po zapnutí algoritmu budou sledovány jednotlivé přírůstky konzumace paměti a také celková spotřeba paměti. Přírůstky budou vypočítané jako konzumace paměti při spuštěném algoritmu mínus konzumace před spuštěním algoritmu. Vše opět bude zobrazeno v tabulkách pro jednotlivé algoritmy rozdělení.

Tabulka 13 Paměť – Use training set (Vlastní zpracování)

Use training set					
		Přírůstky (MB)		Paměť (MB)	
	Počest instancí	2 000	5 000	2 000	5 000
Weka 3. 6. 12	Naive Bayes	5,6	7,0	111,5	178,5
	IBk	3,1	4,1	120,1	181,9
	Random Forest	23,0	37,0	163,4	242,3
Data Mining	Naive Bayes	19,0	50,2	70,3	107,4
	IBk	2,8	6,3	54,1	63,5
	Random Forest	18,0	35,3	69,3	92,5

Jako první si tradičně představíme metodu rozdělení Use training set, její výsledky ukazuje tabulka 13. Jak je z tabulky patrné tak celková konzumace paměti různými algoritmy není nikterak velká. Tomuto trendu se vymyká pouze algoritmus Naive Bayes v aplikaci DataMining.

Dá se říci, že u algoritmů v aplikaci Data Mining jsou přírůstky dvojnásobně větší při 5 000 instancích, než je tomu při 2 000 instancích. U systému Weka je tato konzumace pouze o něco větší při zvětšení datového souboru. Dalo by se tedy předpokládat, že při větším množství instancí spravuje Weka paměť lépe.

Celkově aplikace Data Mining zabírala méně paměti než systém Weka, jak je vidět v druhé části tabulky. Weka se pohybuje mezi 100-200 MB zaplněné paměti a aplikace Data Mining dosáhla maximální výše (107 MB) pouze při spuštění algoritmu Naive Bayes a testování 5 000 instancí. Toto je určité pozitivum v aplikaci Data Mining.

Tabulka 14 Paměť – Percentage split (Vlastní zpracování)

Percentage split					
		Přírůstky (MB)		Paměť (MB)	
	Počest instancí	2 000	5 000	2 000	5 000
Weka 3. 6. 12	Naive Bayes	2,4	3,5	109,3	176,4
	IBk	2,6	5,0	119,2	177,2
	Random Forest	21,0	35,0	158,7	219,2
Data Mining	Naive Bayes	7,6	16,1	58,9	73,3
	IBk	1,2	3,0	52,5	60,2
	Random Forest	14,0	28,3	65,3	85,5

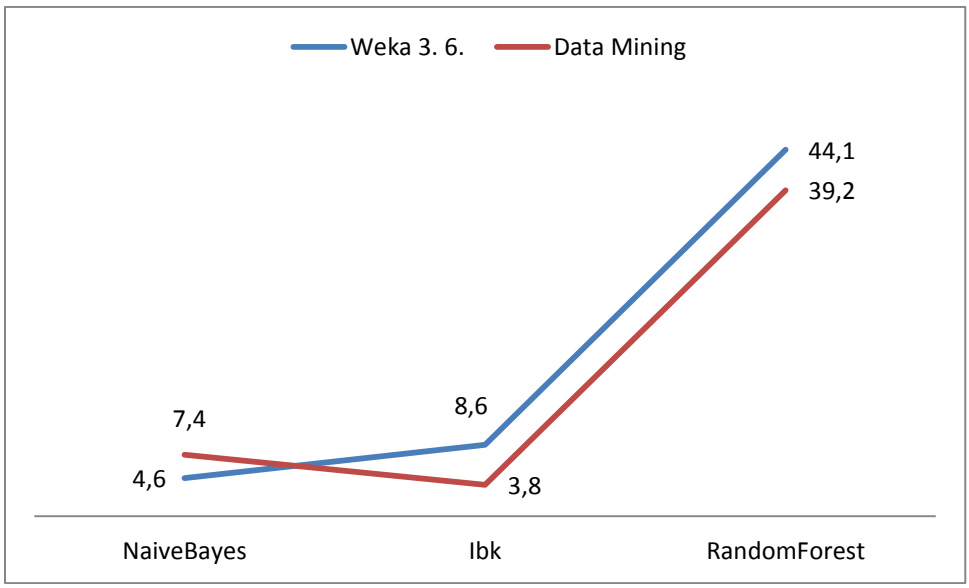
Konzumace paměti by měla být v tomto případě o něco menší, než tomu bylo při předchozím rozdělení, jelikož se testuje méně instancí. Tento předpoklad je potvrzen v tabulce 14, která ukazuje procentuální rozdělení dat a jeho paměťové vytížení. Zde platí podobná pravidla jako v předchozí tabulce.

Aplikace Data Mining se opět pohybuje v nižších číslech celkové konzumace paměti.

Tabulka 15 Paměť – Cross-validation (Vlastní zpracování)

Cross-validation					
		Přírůstky (MB)		Paměť (MB)	
	Počest instancí	2 000	5 000	2 000	5 000
Weka 3. 6. 12	NaiveBayes	3,4	4,6	112,3	178,2
	IBk	1,7	8,6	118,5	179,0
	Random Forest	39,0	44,1	167,8	245,3
Data Mining	NaiveBayes	2,4	7,4	53,7	64,6
	IBk	0,8	3,8	52,1	61,0
	Random Forest	22,7	39,2	74,0	96,4

Poslední tabulka již tradičně ukazuje metodu rozdělení Cross-validation. Zde jsou podobné výsledky jako při metodě rozdělení Use training set. To je logické, jelikož se testuje stejný počet instancí. Výpočet je o něco složitější, tím pádem jsou přírůstky i celková paměť o něco větší.



Obrázek 35 Přírustky 5 000 instancí (Vlastní zpracování)

V obrázku 35 jsou zobrazeny přírůstky paměti pro všechny algoritmy. Tyto přírůstky jsou ve dvou případech nižší v aplikaci Data Mining. Jedině v případě algoritmu Naive Bayes je přírůstek vyšší.

Opět se dá říci, že přírůstky jsou celkově u jednotlivých algoritmů v systému Weka menší a tímto tedy Weka vykazuje lepší správu paměti než aplikace Data Mining.

Nyní se ještě podíváme na velikost paměti potřebnou pro jednotlivé aplikace po všech testech.

Proces	PID	Pracovní sada (kB)
javaw.exe	4848	183 244
DataMining.exe	4732	55 904

Obrázek 36 RAM po experimentech (Vlastní zpracování)

Jak je možné z obrázku 36 vyčíst, tak konzumace paměti po vykonání všech testů je více jak třikrát vyšší u systému Weka. Tato problematika byla popsána již dříve. Vypadá to, jako by si Weka ukládala některé informace o algoritmech do paměti, aby poté měla k těmto informacím rychlejší přístup.

Celkový dojem z experimentů v oblasti paměti je jednoznačný. Systém Weka zabírá mnohem více z celkové paměti, ale zase si lépe umí poradit s rozjetými algoritmy. Přírůstky velikosti paměti systému Weka jsou menší, tím pádem je lépe zvládnutá práce s pamětí u jednotlivých algoritmů. Toto je zcela jistě dáno vyspělostí systému Weka. Aplikace Data Mining ale nemá nijak vysoké paměťové nároky a celková konzumace paměti se nedostala při větším počtu instancí do takových hodnot jako systém Weka.

6.3.4 Shrnutí experimentů

Bylo představeno vícero experimentů. Nejprve bylo vyzkoušeno aplikování filtrů na data. Podobností výsledků po aplikování filtrů bylo dokázáno, že se filtry bez větších problémů dají použít v aplikaci Data Mining.

Poté byla ověřena správnost výsledků a bylo vyhodnoceno, že se výsledky aplikace Data Mining téměř shodují se systémem Weka. Tím byla dokázána správná implementace jednotlivých algoritmů.

Po těchto dvou základních experimentech se mohlo přejít k časové a prostorové náročnosti aplikace. V časové náročnosti aplikace je jasně lepší systém Weka, kde jsou časy mnohonásobně kratší než u aplikace Data Mining. Aplikace si nevede špatně u menších souborů především pro algoritmy Naive Bayes a IBk. Na tuto oblast by se mělo zaměřit a optimalizovat ji v budoucím vývoji aplikace.

Prostorová náročnost aplikace na tom byla o něco lépe. Nejlépe na tom byla v celkové konzumaci paměti, kde výsledky, i při větším množství instancí, vykazovala aplikace Data Mining mnohem lepší než systém Weka. Pro jednotlivé přírůstky paměti algoritmů byl o něco lepší opět systém Weka. Při opakovaném načítání souboru a spouštění algoritmů se konzumace paměti v systému Weka značně zvětšovala. Tato nevýhoda je v aplikaci Data Mining eliminována.

Celkově experimenty vyšli lépe pro systém Weka. To je logické, vzhledem k dlouholeté tradici a vývoji systému. Aplikace Data Mining vykazuje velice přijatelné výsledky, vzhledem k tomu, že byla vyvíjena tak krátkou chvíli. Určitě je zde prostor pro zlepšení a optimalizaci aplikace.

7 Zhodnocení a diskuze

Po implementaci aplikace a provedení všech experimentů z pohledu správnosti výsledků, časové a prostorové náročnosti aplikace je možné přejít ke zhodnocení celé práce.

Po nastudování a převodu jednotlivých zdrojových kódů vznikla opravdu jednoduchá, uživatelsky přívětivá, aplikace Data Mining, kterou je opravdu snadné ovládat i pochopit. Navíc je v aplikaci zabudována nápověda, která je uživateli k dispozici.

Jazyk C++ byl dobrým řešením pro převod. Oproti jazyku C nebo Objective-C má více zabudovaných funkcí a jeho možnosti jsou opravdu velké. Také dokumentace k tomuto programovacímu jazyku je velmi dobrá. Rozdíly mezi C++ a Java jsou popsány v začátcích práce.

Freamwork Qt jazyka C++ je také velmi přehledný a jednoduchý. Aplikace Qt Creator udělá většinu grafiky za vás a také různé signály po kliknutí například na tlačítko zvládá velmi jednoduše a dokáži si představit i začátečníka, který s touto aplikací efektivně pracuje.

Aplikace Data Mining má velmi podobné výsledky jako systém Weka, což je určitě pozitivní. Celkové výsledky experimentů byly již zhodnoceny v kapitole Shrnutí experimentů.

7.1 Diskuze

Weka je opravdu složitá aplikace, která je velice dobře naprogramována a také optimalizována. Při studování a přepisu kódu nastaly některé problémy, které zde budou přiblíženy.

První zádrhel práce nastal při studování problematiky zdrojových kódu jednotlivých algoritmů. Tyto algoritmy jsou velmi složité a důkladné probádání zabralo hodně času. Z implementace algoritmů byla použita pouze malá část kódu, ale principy jednotlivých algoritmů zde byly důkladně nastudovány. Samozřejmě s pomocí některých dalších zdrojů.

Složitější byl převod jednotlivých algoritmů. Oba programovací jazyky mají sice podobnou syntaxi, ale jinak jsou velice rozdílné.

Jelikož je Weka dlouho zaběhlý a používaný systém (je vyvíjena od roku 1999), jen těžko mu může aplikace Data Mining konkurovat. Dlouholetý vývoj systému zaručuje opravdovou stabilitu a také dobrou uživatelskou základnu i dokumentaci.

Snad jen z hlediska paměti by se dala zvolit aplikace Data Mining. Hypotéza z cílů práce se z větší části nepotvrdila a celkové časy jednotlivých algoritmů se nepodařilo vylepšit. Toto je dáno dlouholetým vývojem systému Weka, zkušenými programátory a také experty na oblast dolování z dat.

Aplikace Data Mining má především velký potenciál do budoucna, kde po implementování některých rozšíření a vylepšení by mohla systému Weka konkurovat. Více o rozšíření a možnostech aplikace v budoucnu v následující kapitole.

7.2 Možnosti rozšíření aplikace

Aplikace Data Mining položila velice dobrý základ pro algoritmy strojového učení v binární formě. Prostor pro rozšíření aplikace je opravdu velký. V prvé řadě je nutné algoritmy optimalizovat. Jednotlivé algoritmy vykazují dobré časy pro menší datové sady. Pro větší datové sady jsou tyto algoritmy pomalejší než Weka.

Tento problém by se určitě měl v budoucnu odstranit. Zkušený programátor by jistě našel některé chyby a po opravě těchto chyb a celkové úpravě jednotlivých algoritmů by se časy zkrátily.

S pamětí je to podobné. I zde je určitě prostor pro zlepšení, přestože problémy s pamětí nejsou tak velké, jako problémy časového charakteru.

Dále by se dalo zapracovat na souborech, které se do aplikace nahrávají. Odstranit některá omezení souboru, nebo vytvořit možnost pro nahrání souborů s příponou arff, jako v systému Weka.

Jako další rozšíření by bylo vhodné odstranění filtrů z aplikace Data Mining a práci s nominálními a numerickými atributy tak, jak je zvládá systém Weka.

V neposlední řadě by se aplikace dala rozšířit o další algoritmy dolování z dat. Přidat například regresní úlohy, clusterizaci a také některé další algoritmy klasifikace. Do uživatelského rozhraní by se dala přidat vizualizace, grafy a některé další charakteristiky a možnosti.

8 Závěr

Cílem práce bylo vybrat a převést jednotlivé algoritmy z jazyka Java do binární (exe) formy. Dalším cílem bylo tento systém otestovat a porovnat se stávajícím řešením v systému Weka.

Nejprve bylo nutné nastudovat důkladně problematiku strojového učení a algoritmů dolování z dat. Dále vybrat vhodné algoritmy a nastudovat implementaci těchto algoritmů v softwaru Weka. Po nastudování problematiky bylo nutné tyto zdrojové kódy převést z jazyka Java do jazyka C++ (binární formy). Po převedení jednotlivých zdrojových kódů následovalo vytvoření uživatelského rozhraní.

Nakonec bylo nutné vše patřičně otestovat a provést experimenty. Tyto experimenty byly převážně tvořeny porovnáváním aplikace Data Mining a systému Weka. Experimenty byly provedeny na základě správnosti výsledků, časové a prostorové náročnosti obou aplikací. Byly provedeny i některé dílčí experimenty.

Jednotlivé cíle práce byly splněny vyčerpávajícím způsobem. Aplikace, která byla vytvořena pro potřeby práce, funguje, má implementované uživatelské rozhraní pro zadávání vstupů a výstupů, a také vykazuje velice podobné výstupy jako systém Weka.

Spolu s růstem dat každým dnem a potřebou z těchto dat získávat nějaké relevantní informace se metody dolování z dat postupně stávají stále více žádoucí. Tyto metody je třeba v budoucnu rozvíjet, především pro velké datové soubory. Zde právě naráží systém Weka na to, že je implementována v jazyce Java. Při stále větším růstu dat se systém zpomaluje a zabírá více paměti. Je jasné, že zde hrají roli i výpočetní prostředky počítače, na kterém je systém Weka spuštěn, ale přece jen je Java ve výpočetní a prostorové složitosti stále o něco horší než C++.

9 Literatura

BENYAMIN. A Gentle Introduction to Random Forests. *CitizenNet* [online]. 2012 [cit. 2015-04-30]. Dostupné z: https://citizennet.com/blog/2012/11/10/random-forests-ensembles-and-performance-metrics/

Can I use CSV files? 2005. Weka Wikispaces [online]. [cit. 2015-05-11]. Dostupné z: https://weka.wikispaces.com/Can+I+use+CSV+files%3F

Census-Income Database. UNITED STATES DEPARTMENT OF COMMERCE. The Bren School [online]. 1997 [cit. 2015-05-04]. Dostupné z: http://archive.ics.uci.edu/ml/machine-learning-databases/census-income-mld/

Class ReplaceMissingValues. Weka sourceforgre [online]. 2014 [cit. 2015-05-17]. Dostupné z: http://weka.sourceforge.net/doc.dev/weka/filters/unsupervised/attribute/ReplaceMissingValues.html

DAVIS, James. Quick Introduction to Qt Programming. University of California Santa Cruz: Introduction to Computer Graphics [online]. 2010 [cit. 2015-05-03]. Dostupné z: https://classes.soe.ucsc.edu/cmps160/Fall10/resources/qtexcerpt.pdf

Expertní systémy. BERKA, Petr. *Vysoká škola ekonomická* [online]. 2009 [cit. 2015-04-17]. Dostupné z: http://sorry.vse.cz/~berka/docs/izi456/kap_5.1.pdf

KONONENKO, Igor a KUKAR. *Machine Learning and Data Mining* [online]. 2007 [cit. 2015-04-18]. ISBN 978-1-904275-21-3. Dostupné z: http://goo.gl/1iNO2T

Machine Learning Group. Machine Learning Project at the University of Waikato in New Zealand [online]. 2008 [cit. 2015-01-31]. Dostupné z: http://www.cs.waikato.ac.nz/ml/index.html

MARKHAM, Kevin. Simple guide to confusion matrix terminology. Data School [online]. 2014 [cit. 2015-05-01]. Dostupné z: http://www.dataschool.io/simple-guide-to-confusion-matrix-terminology/

MURACH, Joel. Murach's Java Programming. 4. vyd. California: Murach, Mike & Associates, Inc., 2011. ISBN 978-1890774653.

NĚMEC, Jan. C/C++ (1) - Úvod. LinuxSoft.cz [online]. 2004 [cit. 2015-05-03]. Dostupné z: http://www.linuxsoft.cz/article.php?id_article=370

NG, Andrew. Machine Learning. *Stanford University* [online]. 2015 [cit. 2015-04-18]. Dostupné z: https://class.coursera.org/ml-005/lecture/2

R. BOUCKAERT, Remco et al. WEKA Manual. Weka [online]. 2014 [cit. 2015-05-01]. Dostupné z: http://www.cs.waikato.ac.nz/ml/weka/documentation.html

RIZKY, Soetam. ID3 Algorithm. *Youtube* [online]. 2011 [cit. 2015-04-17]. Dostupné z: https://www.youtube.com/watch?v=wL9aogTuZw8

Rozhodovací stromy. MRÁZOVÁ, Iveta. *Univerzita Karlova v Praze* [online]. 2012 [cit. 2015-04-17]. Dostupné z: http://ksvi.mff.cuni.cz/~mraz/datamining/lecture/Dobyvani_Znalosti_Prednaska_Rozhodovaci_stromy.pdf

Sadawi, Noureddin. How K-Nearest Neighbors (kNN) Classifier Works. *Youtube* [online]. 2014 [cit. 2015-04-17]. Dostupné z: https://www.youtube.com/watch?v=v6278Cjf_qA

Shmueli, G., Patel, N. R., Bruce, P. C.: Data Mining for Business Intelligence. Wiley, 2007. ISBN 978-0470526828.

SCHNEIDER. Cross Validation. *Carnegie Mellon School: Computer Science* [online]. 1998 [cit. 2015-05-01]. Dostupné z: http://www.cs.cmu.edu/~schneide/tut5/node42.html

STROUSTRUP, Bjarne. The C++ Programming Language (4th Edition). 4. vyd. Boston: Addison-Wesley Professiona, 2013. ISBN 978-0321563842.

The ID3 Algorithm. DANKEL, Douglas. *University of Florida* [online]. 1997 [cit. 2015-04-17]. Dostupné z: http://www.cise.ufl.edu/~ddd/cap6635/Fall-97/Short-papers/2.htm

Weka 3: Data Mining Software in Java. Computer Science Department, University of Waikato [online]. 2013 [cit. 2014-04-06]. Dostupné z: http://www.cs.waikato.ac.nz/ml/weka/index.html

Witten, I. H., Frank, E.: Data Mining: Practical Machine Learning Tools and Techniques. Elsevier, 2005. ISBN 978-0123748560.

10 Seznam obrázků

11 Seznam tabulek a vzorců

Přílohy

A Standardizace

```
1.  std::vector <std::string> IBk::standardizeNumeric(std::vector
    <std::string> vec){

2.  std::string str;
3.  double pom;
4.  double standard;

5.  std::vector <double> vektor;
6.  std::vector <std::string> vektorGood;

7.  //transfers string to double
8.  for (size_t k = 0; k < vec.size(); ++k) {
9.      str = vec[k];
10.     pom = atof(str.c_str());
11.     vektor.push_back(pom);
12. }

13. double max = countMax(vektor);
14. double min = countMin(vektor);

15. //transfers double back to string and adds value to vector
16. for (size_t k = 0; k < vektor.size(); ++k) {
17.     //counting value
18.     standard = (vektor[k] - min) / (max - min);
19.     std::ostringstream s;
20.     s << standard;
21.     std::string strn(s.str());
22.     vektorGood.push_back(strn);
23.     s.flush();
24.     }
25. return vektorGood;
26. }
```

Obrázek 37 Standardizace (Vlastní zpracování)

B Metody rozdělení souboru

- **Use training set**

```
1. trainTestPair SplittingMethod::useTrainingSet(std::vector <
   std::vector<std::string> > Data) {
2.
3. std::vector < std::vector<std::string> > trainData;
4. std::vector < std::vector<std::string> > testData;
5.
6. trainData = Data;
7. Data.erase(Data.begin());
8. testData = Data;
9.
10.  return std::pair<std::vector < std::vector<std::string> >,
   std::vector < std::vector<std::string> > > (trainData, testDa-
   ta);
11.  }
```

Obrázek 38 Use training set (Vlastní zpracování)

- **Percentage split**

```
1. trainTestPair SplittingMethod::percentageSplit(std::vector <
   std::vector<std::string> > Data, int percentageSplit) {
2.
3. std::vector <std::string> test;
4.
5. std::vector < std::vector<std::string> > trainData;
6. std::vector < std::vector<std::string> > testData;
7.
8. int numberOfSets = numberOfInstances - percentageSplit * num-
   berOfInstances/100;
9. int random = 0;
10.
11.  trainData = Data;
12.
13.  // spliting data to train set and test set randomly
14.  for (int j = 0; j < numberOfSets-1; j++) {
15.  random = rand() % (numberOfInstances-j) + 1;
16.  for (size_t k = 0; k < Data[j].size(); k++) {
17.          test.push_back(trainData[random][k]);
18.  }
19.  testData.push_back(test);
20.  test.clear();
21.  trainData.erase(std::find(trainData.begin(), trainData.end(),
   testData[j]));
22.  }
23.  return std::pair<std::vector < std::vector<std::string> >,
   std::vector < std::vector<std::string> > > (trainData, testDa-
   ta);
24.  }
```

Obrázek 39 Percentage split (Vlastní zpracování)

- **Cross-validaton**

```
1. trainTestPair SplittingMethod::crossValidation(std::vector <
   std::vector<std::string> > Data, int numberOfSets) {
2.
3. std::vector <std::string> test;
4.
5. std::vector < std::vector<std::string> > trainData;
6. std::vector < std::vector<std::string> > testData;
7.
8. int random = 0;
9.
10.  trainData = Data;
11.
12.  // spliting data to train set and test set randomly
13.  for (int j = 0; j < numberOfSets; j++) {
14.  random = rand() % (numberOfInstances-j) + 1;
15.  for (size_t k = 0; k < Data[j].size(); k++) {
16.  test.push_back(trainData[random][k]);
17.  }
18.  testData.push_back(test);
19.  test.clear();
20.  trainData.erase(std::find(trainData.begin(), trainData.end(),
   testData[j]));
21.  }
22.  return std::pair<std::vector < std::vector<std::string> >,
   std::vector < std::vector<std::string> > > (trainData, testDa-
   ta);
23.  }
```

Obrázek 40 Cross-validation (Vlastní zpracování)