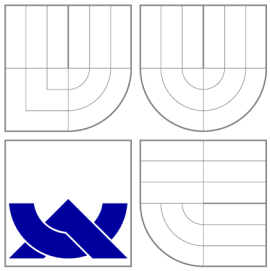
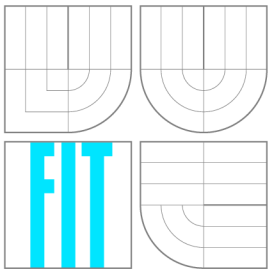


VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INFORMAČNÍCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INFORMATION SYSTEMS

KLASIFIKACE V PROUDU DAT POMOCÍ SOUBORU KLASIFIKÁTORŮ

CLASSIFICATION IN DATA STREAMS USING ENSEMBLE METHODS

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE
AUTHOR

Bc. MARTIN JAROSCH

VEDOUCÍ PRÁCE
SUPERVISOR

Ing. MARTIN HLOSTA

BRNO 2013

Abstrakt

Tato práce pojednává o problematice získávání znalostí z dat a je zaměřena na klasifikaci dat v prostředí datových proudů. Jsou zde popsány tři metody klasifikace dat v datových proudech, využívající soubory klasifikátorů. Metody jsou v praktické části implementovány a zařazeny do klasifikačního systému. Měřením a experimentováním jsou analyzovány a porovnány implementované metody, které byly následně integrovány do analytického systému MAS. Na závěr práce jsou zhodnoceny dosažené výsledky.

Abstract

This master's thesis deals with knowledge discovery and is focused on data stream classification. Three ensemble classification methods are described here. These methods are implemented in practical part of this thesis and are included in the classification system. Extensive measurements and experimentation were used for method analysis and comparison. Implemented methods were then integrated into Malware analysis system. At the conclusion are presented obtained results.

Klíčová slova

Dolování z dat, datové proudy, klasifikace, soubor klasifikátorů, analytický systém, MAS, CSHT, DoS, DWAA, změna konceptu

Keywords

Data mining, data streams, classification, ensemble, analytical system, MAS, CSHT, DoS, DWAA, concept drift

Citace

Martin Jarosch: Klasifikace v proudu dat pomocí
souboru klasifikátorů, diplomová práce, Brno, FIT VUT v Brně, 2013

Klasifikace v proudu dat pomocí souboru klasifikátorů

Prohlášení

Prohlašuji, že jsem tuto diplomovou práci vypracoval samostatně pod vedením Ing. Martina Hlosty. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

.....
Martin Jarosch
21. května 2013

Poděkování

Chtěl bych poděkovat vedoucímu mé diplomové práce Ing. Martinovi Hlostovi za poskytnutou pomoc, odborné připomínky a rady při řešení diplomové práce.

Obsah

1 Získávání znalostí z dat **4**
- 1.1 Dolování z dat . . . 4
- 1.2 Dolovací úlohy . . . 5
- 1.3 Zdroje dat . . . 6

2 Dolování v proudech dat **10**
- 2.1 Metody pro zpracování proudu dat . . . 11
- 2.2 Dotazování v proudech dat . . . 13
- 2.3 OLAP a datové kostky v proudech dat . . . 13
  - 2.3.1 Komprese časové osy: Tilted time frame . . . 14
  - 2.3.2 Kritické vrstvy . . . 14
- 2.4 Dolování frekventovaných vzorů . . . 15
- 2.5 Klasifikace v proudu dat . . . 16
  - 2.5.1 Problémy klasifikace v datových proudech . . . 16
  - 2.5.2 Klasifikace pomocí rozhodovacích stromů . . . 17
  - 2.5.3 Klasifikace pomocí souboru klasifikátorů . . . 18
- 2.6 Shluková analýza proudu dat . . . 19

3 Malware analysis system **21**
- 3.1 Vlastnosti systému . . . 21
- 3.2 Architektura systému . . . 22

4 Vybrané metody **23**
- 4.1 Algoritmus CSHT . . . 23
- 4.2 Algoritmus pro detekci DoS . . . 25
- 4.3 Algoritmus DWAA . . . 27

5 Návrh a implementace **29**
- 5.1 Základní klasifikátory . . . 29
  - 5.1.1 Naivní Bayes . . . 29
  - 5.1.2 Support Vector Machine . . . 30
  - 5.1.3 Rozhodovací strom C4.5 . . . 30
- 5.2 Systém pro klasifikaci v datových proudech . . . 30
- 5.3 Struktura systému . . . 31
- 5.4 Reprezentace dat . . . 31
- 5.5 Pomocné třídy . . . 32
- 5.6 Paralelizace systému . . . 32
  - 5.6.1 Task Parallel Library . . . 32

5.6.2 Rozdělení systému . . . 33
5.7 Jednotná rozhraní . . . 34
5.8 Řízení systému . . . 35
5.9 Zpracování vstupu . . . 35
5.10 Zpracování výstupu . . . 36
5.11 Implementace metod . . . 36
5.11.1 Basic . . . 36
5.11.2 CSHT . . . 37
5.11.3 DoS . . . 37
5.11.4 DWAA . . . 38
5.12 Napojení na MAS . . . 39

6 Měření a experimenty **41**
6.1 Testovací data . . . 41
6.2 Postup měření . . . 41
6.3 Naměřené hodnoty . . . 42
6.3.1 Metoda Basic . . . 42
6.3.2 Metoda CSHT . . . 43
6.3.3 Metoda DoS . . . 45
6.3.4 Metoda DWAA . . . 45
6.4 Pozvolná změna konceptu . . . 46
6.5 Vyhodnocení výsledků . . . 47
6.6 Konfigurace parametrů . . . 48
6.6.1 Velikost plovoucího okna . . . 48
6.6.2 Počet klasifikátorů . . . 48
6.7 Modifikace algoritmů . . . 49

7 Návrh na další postup **50**

8 Závěr **51**

A Obsah CD **54**

Úvod

V dnešní době jsou informace jednou z nejcennějších komodit. Není proto překvapením, že se snažíme získat co nejvíce informací z prostředí, které nás obklopuje. Naše společnost je protkaná sítí informačních systémů, databází a jiných prostředků pro shromažďování dat. Shromážděním dat však informace nezískáme a proto vznikl obor dolování z dat. Ten se zaměřuje na získávání znalostí z dat, obsahuje mnoho metod a postupů, jak extrahovat pro nás dříve neznámé znalosti z obrovského množství dat.

Vyvinuté postupy a metody jsou dnes hojně využívány a přináší svým uživatelům značné konkurenční výhody. Vývoj však postupuje stále dopředu, ukládání a dolování z dat již přestává stačit. Požadavky na okamžité a přesné výsledky stále rostou a vytváří tlak na vývoj nových metod. Jednou z relativně nových oblastí je dolování v prostředí datových proudů. Analýzou dat v datových proudech můžeme získat informace mnohem dříve než dovolují běžné dolovací techniky. Musíme se ale vyrovnat s novými problémy. Právě touto problematikou se zabývá má diplomová práce, která se soustředí na klasifikaci v proudu dat pomocí souboru klasifikátorů.

První kapitola představuje úvod do problematiky dolování z dat. Jsou v ní vysvětleny pojmy získávání znalostí z dat a dolování z dat. Popisuje základní dolovací úlohy, známé zdroje dat a datové sklady.

Po stručném úvodu se druhá kapitola zaměřuje na prostředí datových proudů, jejich specifické požadavky a problémy. Jsou zde uvedeny metodiky a postupy, jak pracovat s datovými proudy, jak provádět jejich zpracovávání, dolování frekventovaných vzorů, shlukování a klasifikaci. Techniky klasifikace jsou rozepsány podrobněji a je uveden obecný algoritmus pro klasifikační metody pracující se souborem klasifikátorů.

Kapitola třetí popisuje vlastnosti a architekturu analytického systému MAS, který je vyvíjen na naší fakultě. Do systému byly integrovány klasifikační metody popsané v následující kapitole. Metody byly vybrány tak, aby každá reprezentovala jiný přístup ke klasifikaci a adaptaci na změnu konceptu v datových proudech za pomoci souboru klasifikátorů. Čtvrtou kapitolou končí teoretická část práce a přecházím k části praktické.

Pátá kapitola uvádí návrh a implementaci klasifikačního systému, do kterého byly zasazeny implementované metody. Systém byl navržen pro efektivní zpracování datových proudů a využívá dostupných paralelních prostředků. Specifikace vybraných metod byly relativně strohé, proto zde uvádím implementované odchylky a částí algoritmů, které nebyly metodami popsány. Závěrem kapitoly je postup napojení na analytický systém MAS.

V šesté kapitole se zaměřuji na měření a porovnání implementovaných metod. Nechybí postup měření a popis použitých testovacích dat. Dosažené výsledky metod jsou následně kriticky zhodnoceny. Po jejich vyhodnocení následují dvě části, první věnující se nastavení parametrů metod a druhá obsahující experimenty s modifikací nejlepší metody.

Závěrem práce je uvedeno shrnutí dosavadní práce a návrh na další postup, jak lze práci nejlépe rozšířit.

Kapitola 1

Získávání znalostí z dat

První kapitola vysvětluje základní pojmy, které jsou v průběhu textu použity a uvádí stručný přehled dolovacích úloh.

1.1 Dolování z dat

Dolování z dat, anglicky *data mining* je v současnosti hojně používaným pojmem, který je spojován s celou řadou informačních technologií a zasahuje do mnoha oborů. Můžeme se s ním setkat při analýzách financí a trhů. Má nezastupitelné místo v bioinformatice, astroinformatice, podpoře managementu, při detekci podvodů a v bezpečnostních systémech. Dolování z dat se používá všude, kde je k dispozici velké množství dat, z něhož chceme získat informace. Vzhledem k množství a složitosti dat je ale náročné tyto informace získat.

Dolování z dat je dnes považováno za synonymum *získávání znalostí z dat*. Jde ale o dva různé pojmy, kde dolování z dat je pouhým krokem v procesu získávání znalostí z dat. Získávání znalostí z dat obsahuje sběr, prvotní předzpracování dat, samotné dolování, vyhodnocení a prezentaci výsledků. Cílem celého procesu je extrahovat potenciálně užitečné informace, které nelze získat jednoduchým příkazem, nelze je z dat vyčíst a měly by být pro nás dříve neznámé. Jednotlivé kroky procesu, jak byly popsány v [14], jsou proto následující:

1. Čištění dat - odstranění šumu a nekonzistentních dat.
2. Integrace dat - spojení dat z více zdrojů.
3. Selekce dat - výběr relevantních dat pro analýzu.
4. Transformace dat - úprava dat do vhodné formy.
5. Dolování z dat - aplikace inteligentních metod pro extrakci zajímavých vzorů dat.
6. Vyhodnocení vzorů - identifikace skutečně zajímavých vzorů reprezentujících získanou znalost na základě zvolené metriky.
7. Prezentace znalostí - vizualizační techniky pro prezentaci získaných informací.

Proces získání znalostí musí provést všechny kroky, aby mohl zpracovat vstupní data. Ty se vyznačují obrovským množstvím (v řádech miliard záznamů), dimensionalitou a složitostí. Data mohou reprezentovat prostor, čas, sekvence, multimédia, grafy, DNA a další složité struktury. Vhodným předzpracováním a použitím vysoce škálovatelných a sofistikovaných algoritmů lze získat souvislosti a informace, které by zůstaly jinak skryty.

1.2 Dolovací úlohy

Zde jsou vysvětleny základní dolovací úlohy používané při dolování dat. Mohou být použity jednotlivě, nebo mohou být v sofistikovaných metodách kombinovány pro znásobení jejich výhod a předností.

Dolování asociačních pravidel a frekventovaných množin

Získávání asociačních pravidel a frekventovaných množin umožňuje nalézt zajímavé asociace a korelace ve velkém množství dat. Mnoho firem a podniků vlastní obsáhlé transakční databáze, obsahující historii obchodních záznamů. Z nichž je pak možné získat zajímavé vazby mezi daty, které mohou ovlivnit obchodní rozhodování. Typickým příkladem je analýza nákupního košíku, umožňující nalézt položky nákupu, které se vyskytují často společně.

Nalezením společně se vyskytujících položek získáme *frekventované množiny*, ty musí splňovat podmínku minimální podpory, pravděpodobnost, že se položky vyskytují současně. *Asociační pravidla* pak získáme z frekventovaných množin, vyloučením množin nesplňující podmínku spolehlivosti, pravděpodobnost, že transakce obsahující jednu položku obsahuje i položku druhou.

Klasifikace a predikce

Klasifikace a *predikce* je formou analýzy dat, umožňující extrahovat skryté informace, které je možné použít pro tvorbu inteligentních rozhodnutí. Na základě těchto postupů jsme schopni odhadnout nejpravděpodobnější trendy dat. Obě úlohy jsou si navzájem podobné, liší se však v podstatném faktoru. Klasifikace dokáže rozdělit, klasifikovat data do jednotlivých tříd, určených diskrétním neseřazeným návěštím. Zatímco predikce dokáže odhadnout hodnoty spojitých funkcí nebo seřazených hodnot. S pomocí těchto metod jsme schopni odhadnout hodnoty diskrétních i spojitých atributů na základě předchozích dat.

Obě metody je nutné nejprve natrénovat na již označených datech, kdy při dostatečném množství trénovacích dat jsme schopní vytvořit modely odpovídající s velkou přesností současnému trendu dat. Na základě naučeného modelu se pak algoritmus rozhodne, jakou kategorii nebo hodnotu prvku přidělí.

Shluková analýza

Shluková analýza, neboli *shlukování*, slouží k rozdělení dat do skupin (*shluků*, *clusterů*), tak aby si byla data ve skupině co nejvíce podobná a zároveň byly jednotlivé skupiny dat navzájem co nejvíce rozdílné. Výhoda tohoto přístupu je zřejmá ve velkých databázích, kde je přidělování návěští jednotlivým záznamům náročné jak finančně, tak i časově.

Shlukování nepotřebuje předem určené třídy záznamů, právě naopak bývá využíváno k seskupení souvisejících záznamů do skupin, které mohou být následně označeny dalším algoritmem, nebo expertem v daném oboru. Shluková analýza se používá jak při kategorizaci záznamů, tak i při detekci odlehlých hodnot, které jsou vzdáleny od ostatních skupin záznamů a mohou představovat chyby, šum v datech nebo bezpečnostní hrozby a záznamy vymykající se běžnému chování.

1.3 Zdroje dat

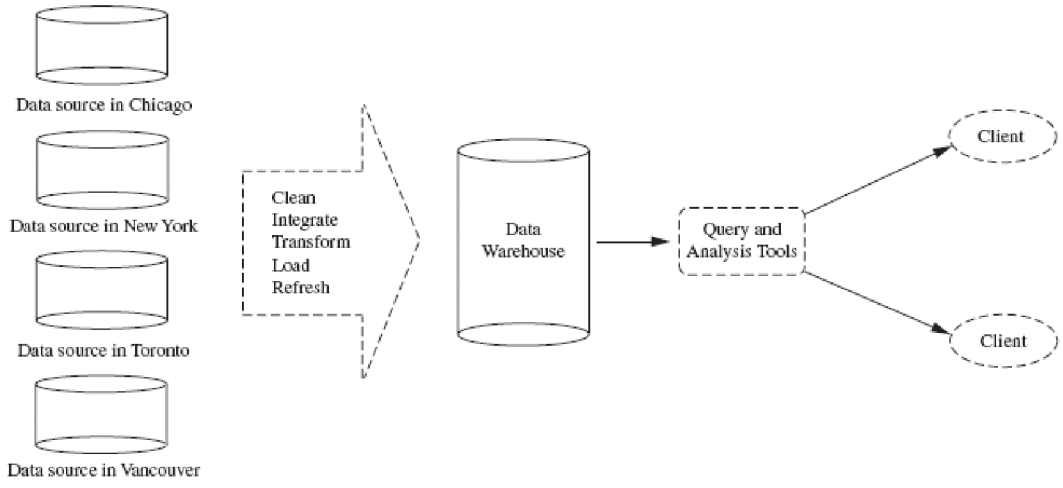
Dolování z dat může být principiálně provedeno na libovolném zdroji dat a to jak *perzistentních*, tak i *transientních*. V následující části jsou uvedeny tři nejpoužívanější druhy datových zdrojů, kterými jsou relační databáze, datové sklady a transakční databáze. Následovanými popisem specifičtějších druhů zdrojů dat, vyvinutých pro zpracování složitých dat. Tyto popisy se řídí výpisem zdrojů dat, uvedeném v [2].

Relační databáze

Relační databáze jsou dnes jedním z nejčastějších uložišť dat, proto jsou i nejčastějším zdrojem dat pro dolování. Databázový systém relační databáze se skládá ze systému řízení báze dat (SŘBD), který se stará o správu dat a z kolekce dat, databáze. Data v relační databázi jsou uložena do soustavy tabulek, obsahujících jednotlivé atributy. Databáze vetšinou obsahují velký počet záznamů, zpracovávaných databázovými dotazy, nejčastěji jazykem SQL. Dolování z dat pak rozšiřuje schopnosti SŘBD o možnosti analýzy a detekce zajímavých informací. V některých produktech pro správu relačních databázi jsou funkce dolování z dat již zahrnuty.

Datové sklady

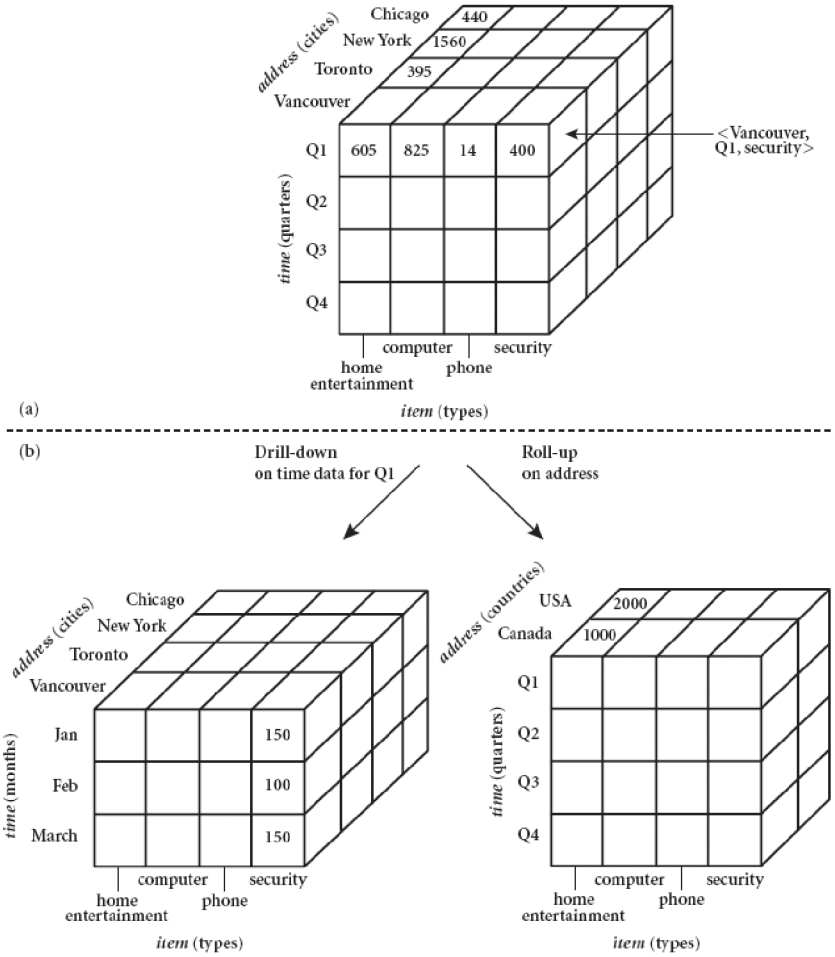
Datové sklady jsou dnes rozšířeným nástrojem pro archivaci a zpracování dat v podnikové sféře. Jsou stavěny ve velkých firmách za účelem sjednocení všech nashromážděných dat a jejich následném zpracování pro dolování dat. Smyslem dat v datovém skladu není reprezentovat současný stav, nepotřebují zpracovávat transakce ani každodenní operace relačních databází. Musí poskytovat rychlé čtení zpracovaných historických dat pro dolování a možnost nahrávání dalších dat.



Obrázek 1.1: Datový sklad, převzato z [2].

Data jsou pravidelně stahována z firemních databází, jsou čištěna, integrována do jednoho celku, transformována, sumarizována na optimální úroveň a jinak upravována. Tímto předzpracováním dat se dosáhne lepších výsledků při dolování z dat. Data jsou ve skladu organizována dle subjektů zájmu a jsou uzpůsobena pro modelování a analýzu dat pro strategické rozhodování. Datový sklad je obvykle modelován jako multidimenzionální datová

struktura, kde každá dimenze odpovídá jednomu nebo skupině atributů. Tato struktura se nazývá *multidimenzionální datová kostka*. Poskytováním multidimenzionálního pohledu, předzpracováním a sumarizací dat je datový sklad vhodný pro provádění *OLAP* (*on-line analytical processing*) operací. OLAP operace mohou prezentovat data na různých úrovních abstrakce a z různých úhlů pohledu. Typickými OLAP operacemi jsou *drill-down* a *roll-up*, zobrazující data na různých stupních sumarizace.



Obrázek 1.2: Multidimenzionální datová kostka a OLAP operace, převzato z [2].

Transakční databáze

Rozdíl mezi transakčními databázemi a ostatními druhy databází je v souboru obsahujícím záznamy jednotlivých obchodních transakcí. Obchodní transakce typicky eviduje identifikátor transakce a seznam položek tvořících transakci (zakoupené položky). Transakční databáze může obsahovat i další tabulky, udávající rozšiřující údaje o prodeji, např. datum, prodávajícího, pobočku. Tento druh databáze je ideální pro dolování asociačních pravidel a frekventovaných množin.

Objektově relační databáze

Objektově relační databáze jsou založeny na objektově relačním datovém modelu, rozšiřujícím relační model o nové datové typy pro manipulaci s komplexními objekty. Současně podporuje principy objektově orientovaného programování. Každá entita je považována za zapouzdřený objekt s proměnnými (atributy), zprávami pro komunikaci mezi objekty a s metodami implementujícími chování objektu po obdržení zprávy. Podobné objekty mohou být seskupeny do tříd a hierarchizovány využitím dědičnosti.

Temporální databáze

Databází ukládajících čas nebo časové souvislosti je více druhů. Temporální databáze ukládají relační data obsahující časové atributy (časová razítka). Databáze sekvencí ukládají záznamy v časovém pořadí a databáze časových řad ukládají hodnoty získané v pravidelných časových intervalech (např. pravidelná měření teploty). Dolováním v temporálních datech můžeme získat časové charakteristiky objektů a trendy závislé na čase.

Prostorové databáze

Prostorové databáze ukládají prostorová data ve formě rastrů (např. satelitní snímky), nebo ve formě vektorů. Vektory reprezentují prostorové objekty pomocí bodů, linií, polygonů, sítí atd. Prostorové databáze mají široké využití v kartografii, navigaci, designu a lékařství. Prostorová databáze spravující objekty měnící se v čase se nazývá prostorově-temporální databáze. Z prostorových a prostorově-temporálních databází jsme schopni vydolovat informace vztažené k objektům v prostoru, můžeme analyzovat vývoj počasí, demografické ukazatele vzhledem k lokalitě nebo vzájemné působení objektů v prostoru.

Textové databáze

Textové databáze obsahují slovní popis objektů. Ten může být vyjádřen slovy, větami, odstavci nebo i celými obsáhlými texty. Texty v databázi mohou být nestrukturované, částečně strukturované (XML, HTML), strukturované (katalog knihovny). Dolováním v textových datech můžeme extrahovat klíčová slova, klasifikovat texty dle jejich obsahu apod.

Multimediální databáze

Multimediální databáze pracují s obrázky, zvukem a videem. Vzhledem k jejich velikosti a požadavkům na plynulé získávání dat v reálném čase musí být spravovány specializovanými metodami. Dolováním můžeme analyzovat obsah a podobnost uložených dat.

Heterogenní databáze

Heterogenní databáze se skládá z množiny propojených autonomních databází, které se mohou výrazně lišit svým obsahem a druhem uložených dat. Složitost takového systému vyžaduje přesná transformační pravidla, pro převod dat do forem srozumitelných ostatním databázím. Heterogenní databáze může vzniknout účelně, nebo postupným technickým a historickým vývojem organizace, potom bývá označena jako zděděná *legacy* databáze. Techniky použité při dolování mohou být řešením, jak propojit jednotlivé databáze.

Proudy dat

Proudy dat jsou specifické vzhledem k ostatním zdrojům dat. Mohou být generovány libovolnými aplikacemi a hardwarem. Jsou potenciálně nekonečným zdrojem dat, které se dynamicky mění, plynou v určeném pořadí a často vyžadují reakce v reálném čase. Typickými příklady jsou měření a monitorování energetických sítí, počítačových sítí, počasí a provozu na webu. Jelikož je množství takto produkovaných dat obrovské, nelze jej celé uložit a musí být zpravidla zpracováno jediným průchodem. Dotazy na datové proudy jsou většinou spojité a vracejí přibližné odpovědi. Obsah datového proudu může být libovolný, proto se na něj dají aplikovat i různé dolovací úlohy. Všechny ale musí vycházet ze sumárních charakteristik a modelů dat.

Web

Web je složen z distribuovaných informačních služeb poskytujících přístup k síti vzájemně propojených objektů. Tím vzniká celosvětová heterogenní databáze s velkým potenciálem pro dolování. Mohou být analyzovány přístupové vzory uživatelů *Web usage mining*, *Weblog mining*, obsah webových stránek a sociální skupiny. Pro indexaci a vyhledávání na webu se využívá shlukování a klasifikace webových stránek.

Kapitola 2

Dolování v proudech dat

Nacházíme se v době ovládané informacemi, informace jsou to nejcennější co máme a tvoří základ našich rozhodnutí. Jelikož jsou informace v naší společnosti tak důležité, potřebujeme získávat nové informace co nejrychleji a mít staré vždy po ruce. Rozvoj celosvětových informačních sítí nám toto umožnil. Každým rokem lidstvo vyprodukuje a přenese nepředstavitelné množství dat, ze kterých informace čerpáme a toto číslo se neustále zvětšuje. Ve skutečnosti vyprodukujeme tak velké množství dat, že poměrnou část žádný člověk nikdy neuvidí. Máme data, jejichž přenos na jedno místo a následná analýza jsou příliš nákladné. Data můžou proudit příliš rychle a v množství, které nedokážeme uložit. Výsledky požadujeme co nejdřív, nejlépe okamžitě.

Právě dolování v proudech dat je řešením. S využitím známých metod pro dolování v databázích, jejich úpravou a vývojem nových metod jsme schopni analyzovat nepřetržitě přitékající a odtékající data, označovaná jako proud dat.

Datové proudy se vyznačují několika specifickými vlastnostmi, které dělají jejich dolování obtížnějším. Jsou potenciálně nekonečné, nevíme kdy a jestli vůbec někdy skončí. Data tudíž nelze uložit a analyzovat v celku jako při běžném dolování, musíme vycházet ze získaných modelů a sumárních charakteristik. Algoritmy musí být jednoprůchodové, náhodný přístup k datům je příliš drahý. Data se můžou rychle a dynamicky měnit a to obsahem i rychlostí přijímání dat. Proto bývá z pohledu uživatele i zpracování dat vyžadována odezva v reálném čase.

Data obsažená v datových proudech bývají většinou nízkoúrovňová nebo vysoce dimenzionální. To je zapříčiněno povahou zdrojů generujících datové proudy. Zdroji mohou být nejrůznější senzory, nasazené na sledování počasí, rozvodných sítí, internetového provozu apod. Data přenášená družicemi a sondami jsou neustále posílána na zem datovými proudy. Existují aplikace vyžadující neustálé monitorování a okamžité reakce systému jako jsou bezpečnostní systémy počítačových sítí nebo sledování bankovních transakcí. Ty se naučí, jak vypadá běžný provoz a při detekci abnormálního chování adekvátně reagují, spustí poplach, zablokují útočícího uživatele.

Dolování v proudech dat je vhodné použít i v případě, že máme všechny data uložena v databázi, ale jejich množství se pohybuje v terabytech nebo dokonce petabytech. Využití jednoprůchodových proudových dolovacích algoritmů zajistí mnohonásobně rychlejší zpracování než běžný přístup.

2.1 Metody pro zpracování proudu dat

Jak již bylo zmíněno, nelze procházet datovým proudem více než jednou a pokud jsou proudící data příliš rychlá, tak je nejsme schopni ani všechna zaznamenat. Množiny možných hodnot v proudech dat nejsou omezeny, nelze je tedy všechny exaktně popsat. Pro efektivní zpracování datových proudů musely být vynalezeny nové metody, datové struktury a algoritmy. Ty musí volit mezi množstvím paměti, která nebude nikdy schopna zaznamenat vše a mezi přesností získaných výsledků. Proto se při dolování v proudech dat neočekávají přesné výstupy, ale snažíme se dosáhnout *aproximace*. Aproximace, jež se co nejvíce blíží výsledkům, které by nám poskytl standardní víceprůchodový dolovací algoritmus nad perzistentními daty.

Pro zaznamenání obsahu datového proudu používáme *synopse*, neboli souhrny, přehledy dat. Jsou to datové struktury, které se snaží uložit velké množství dat na výrazně menším datovém prostoru. Snížením požadavků na přesnost a použitím synopsí můžeme vytvořit časově i prostorově efektivní algoritmy vracející výsledky s malou maximální chybou a velkou pravděpodobností. S pravděpodobností nejméně 1 - δ není relativní odchylka od skutečného výsledku větší než ε.

Metody pro zpracování proudu dat byly popsány v [2].

Náhodné vzorkování

Metoda se nezabývá celým datovým proudem, ale jen částí složenou z pravidelně vybíraných náhodných prvků. Běžně potřebuje pro získání nezkresleného vzorku vědět délku dat před samotným vzorkováním. Technikou *reservoir sampling* vybereme nezkreslený náhodný vzorek S prvků bez nahrazování. Vytvoříme rezervoár velikosti S obsahující vzorek a naplníme jej prvními S prvky datového proudu. Pro následující i-tý prvek vygenerujeme náhodné číslo r v rozmezí 0 až i a pokud $r < S$, tak nahradíme prvek na pozici r i-tým prvkem. Pravděpodobnost vložení i-tého prvku je S/i.

Plovoucí okna

Místo provádění výpočtů nad náhodnými vzorky, nebo všemi přijatými daty, můžeme vycházet jen z nedávných dat. Principem je vytvořit pracovní okno o velikosti w, které obsahuje i-tý prvek přijatý v době t_i do doby t_{i+w}. Okno potom obsahuje posledních w elementů, což snižuje nároky na paměť. Tato metoda je vhodná pro senzorové sítě a burzy, kde jsou podstatné poslední změny a události.

Histogramy

Histogram je souhrnná datová struktura, která může být použita pro aproximaci frekvenční distribuce prvků v datových proudech. Histogram rozděluje prvky do skupin (*buckets*) dle rozdělovacích pravidel. Do šířky podle rozsahu hodnot, nebo dle počtu elementů ve skupině. Tyto varianty jsou sice jednoduché, ale nemusí dobře popisovat pravděpodobnostní distribuční funkci rozložení dat. *V-Optimální histogramy* podobné shlukování již tímto problémem netrpí. Definují velikosti skupin, tak aby minimalizovali frekvenční odchylky v každé skupině.

Multirezoluční metody

Multirezoluční metody zobrazují data na několika úrovních rozlišení a patří mezi metody pro redukci dat. Umožňují nejen kompromis mezi přesností a velikostí zabrané paměti, ale nabízí i možnost zobrazit data na různých úrovních detailu. Multirezoluční je vyvážený binární strom, kde každá úroveň zobrazuje jiné rozlišení a čím dále je od kořene stromu, tím je detailnější.

Sofistikovanějším přístupem je použití shlukovacích metod tvořících hierarchickou strukturu stromů. Například *CF-tree* tvoří hierarchii *mikroshluků* s inkrementálně aktualizovanými souhrnnými statistikami dat, kde informace v mikroshlucích může být agregována do větších *makroshluků*.

Lze použít i techniku vlnek (*Wavelets*), určenou na zpracovávání signálů, v našem případě datového proudu. Rozbíjí vstupní signál na jednoduché ortogonální funkce. Nejjednodušší variantou jsou *Haarovy vlnky*, odpovídající rekurzívnímu průměrování a diferencování na několika úrovních rozlišení. Haarovy vlnky jsou zejména vhodné pro zpracování prostorových a multimediálních dat.

Skeče

Skeče se používají zároveň spolu s *frekvenčními momenty*. Frekvenční momenty poskytují užitečné informace o datech, jako je dotazování, databázovým aplikacím. Navíc indikují stupeň vychýlení (asymetričnost) dat, což je výhodou u paralelních databázových aplikací pro určení patřičného algoritmu pro dělení dat. Pokud je universum $U = \{1, 2, \ldots, v\}$ a datový proud je $A = \{a_1, a_2, \ldots, a_n\}$. Pak pro $k \geq 0$ je frekvenční moment F_k definován následovně:

$$F_k = \sum_{i=1}^{v} m_i^k$$

v je doména a m_i je frekvence hodnoty i v sekvenci. Výpočet F_0 reprezentuje počet rozdílných elementů v sekvenci. F_1 je délka sekvence (v tomto případě N) a F_2 je známo jako *Giniho koeficient homogenity*.

V případě, že je množství dostupné paměti menší než doména v, je nutné použít souhrnu. Právě u frekvenčních momentů se používají zmíněné skeče. Skeče tvoří sumarizaci zabírající malý prostor pro distribuční vektor (např. histogram) použitím náhodných lineárních projekcí na základní datové vektory. Poskytují záruku kvality aproximované odpovědi, například výsledek dotazu je 12 ± 1 s pravděpodobností $0,90$. Máme-li N prvků v univerzu U obsahujícím v hodnot, pak skeče aproximují F_0, F_1 a F_2 s prostorovou složitostí $O(\log v + \log N)$.

Náhodné algoritmy

Náhodné algoritmy, jak už název napovídá, využívají ve své logice náhodnost. Může se např. jednat o náhodné vzorkování nebo skeče. Bývají často použity na velkých, vysoce dimenzionálních datových proudech, kde náhodné algoritmy bývají jednodušší a efektivnější než algoritmy deterministické.

Pokud náhodný algoritmus vrací vždy správnou odpověď, ale liší se dobou zpracování, označujeme jej jako *Las Vegas algoritmus*. Naopak má-li algoritmus dobu běhu omezenou, ale nemusí vracet správný výsledek jedná se o *Monte Carlo algoritmus*.

Chceme-li omezit rozsah výstupů náhodných algoritmů, použijeme *Chebyshevovu nerovnost* pro libovolné kladné reálné číslo k.

$$P(|X - \mu| > k) \leq \frac{\sigma^2}{k^2}$$

Kde X je náhodná proměnná o průměru μ a se směrodatnou odchylkou σ (rozptylem σ^2).

Pro zvýšení spolehlivosti mnoha náhodných algoritmů lze použít více náhodných proměnných, jen pokud jsou tyto proměnné vzájemně nezávislé. To popisuje *Chernoffova mez* pro $X_1, X_2, \ldots, X_n$ reprezentující nezávislé *Poissonovy experimenty*, kde se pravděpodobnost úspěchu liší pokus od pokusu. Získáme-li X jako sumu X_1 až X_n, pak slabší Chernoffova mez vypovídá, že

$$Pr[X < (1 + \delta)\mu] < e^{-\mu\delta^2/4}$$

pro $\delta \in (0, 1]$. To ukazuje, že pravděpodobnost klesá exponenciálně, čím dál jsme od průměru. Špatné odhady jsou tedy mnohem více nepravděpodobné.

2.2 Dotazování v proudech dat

Jak bylo uvedeno v [2], v tradičních databázových systémech jsou data uložena v perzistentních databázích. Toto ale není v případě proudů dat možné, proto se pro správu místo SŘBD používá systém řízení proudů dat (SŘPD), anglicky *data stream management system* (DSMS). Úkolem SŘPD je vyrovnat se s již zmíněnými problémy zpracování jednoho nebo více datových proudů. Každý zpracovaný prvek je buď zahozen a ztracen, nebo archivován.

Architektura zpracovávající dotazy nad datovými proudy se skládá ze tří částí. Z koncového uživatele, dotazovacího procesoru a odkládacího prostoru (operační paměť a disk). Uživatel může zadávat dotazovacímu procesoru jednorázové nebo kontinuální dotazy. Jednorázové dotazy jsou vyhodnoceny v daném čase nad daty označených stejným časovým razítkem nebo *snapshotem*. Kontinuální (nepřetržité) dotazy jsou prováděny nad proudícími daty a vrací nepřetržité výsledky reflektující doposud zaznamenaná data. Dotazy se dále dělí na předdefinované, vytvořené před příchodem dat a ad hoc dotazy získané v průběhu přijímání datových proudů.

Dotazování nad proudy dat je náročné, bez znalosti velikosti vstupu je nemožné určit paměťové nároky většiny nejběžnějších dotazů, pokud nejsou omezeny doménou. Požadování exaktních výsledků dotazů bývá také nereálné, proto pracujeme s výsledky přibližnými. Vyhneme se neomezeným požadavkům na paměť, můžeme použít synopse a snížíme zátěž systému. Navíc ad hoc dotazy potřebují záznamy historie, kterou nemůžeme přesně zaznamenat.

2.3 OLAP a datové kostky v proudech dat

Vzhledem k obrovské velikosti a nízké úrovni dat v datových proudech je jejich celkové uložení v datových skladech nemožné. Proto je pro nalezení zajímavých informací nutné data *agregovat* (suma, průměr). To nám umožní nalézt kritické změny multidimenzionální analýzou v datech na vysoké úrovni abstrakce a zjistit detaily operací *drill down*.

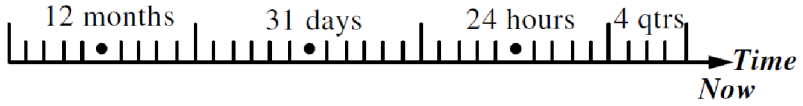
Hlavním rozdílem oproti relačním datovým skladům je nemožnost kompletně popsat detailní úroveň dat a vytvořit multidimenzionální datovou kostku v plné míře. Musíme komprimovat časovou dimenzi dat, ukládat jen data na kritických vrstvách a snažit se efektivně pracovat jen s částečně materializovanými datovými kostkami.

Tyto informace byly čerpány z [2].

2.3.1 Komprese časové osy: Tilted time frame

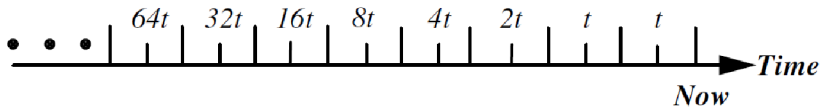
Uživatelé při analýze datových proudů většinou nejvíce zajímají detaily nedávných změn, zatímco u dlouhodobých změn si vystačí jen s hrubými hodnotami. Můžeme proto postupně se stářím zaznamenaných dat snižovat jejich úroveň granularity. Tento model snižování granularity se nazývá *tilted time frame*. Možností jak měnit rozlišení časové osy je více, tři nejznámější jsou uvedeny následovně.

1. Přirozeně nakloněný časový rámec je členěn dle běžně známých úrovní času. Podle aplikace to mohou být poslední 4 čtvrthodiny, následované 24 hodinami, 31 dny a 12 měsíci. Toto rozložení vytvoří za rok 71 jednotek času oproti 35040 jednotkám všech čtvrthodin za rok.



Obrázek 2.1: Přirozeně nakloněný časový rámec, převzato z [2].

2. Logaritmicky nakloněný časový rámec dle logaritmického měřítka. Např. aktuální čtvrthodina, minulá čtvrthodina, 2 minulé, 4 minulé atd. Tím za rok zaplníme 17 časových úseků.



Obrázek 2.2: Logaritmicky nakloněný časový rámec, převzato z [2].

3. Progresivně logaritmický nakloněný časový rámec ukládá snímky dat do jednotlivých rámců. Počet rámců může být od 0 do max_frame, $max\_capacity$ je maximální počet snímků a uběhlý čas od začátku proudu dat je T, pak

$$\log_2(T) - max\_capacity \leq max\_frame \leq \log(T)$$

Každý snímek je označen časovým razítkem t a podléhá pravidlům pro vložení. Snímek je vložen do rámce i pokud $(t \bmod 2^i) = 0$, ale $(t \bmod 2^{i+1}) \neq 0$ a zároveň $i \leq max\_frame$. Jinak je vložen do rámce max_frame. Pokud rámec dosáhne své maximální kapacity je nejstarší snímek v rámci nahrazen novým.

2.3.2 Kritické vrstvy

Kritické vrstvy jsou dalším způsobem jak snížit množství ukládaných dat. Principem je ukládat pouze vrstvy, které jsou pro nás kritické a ne celou datovou kostku. V mnoha aplikacích je výhodné dynamicky a inkrementálně zpracovávat a ukládat dvě kritické kostky (vrstvy). První je označována jako minimální vrstva zájmu a je to nejdetailnější vrstva, která datového analytika ještě zajímá. Tím se zbavíme zbytečně detailních a výpočetně i prostorově náročných vrstev. Druhá vrstva se nazývá vrstvou pozorovací. Je to vrstva,

Frame no.	Snapshots (by clock time)
0	69 67 65
1	66 62 58
2	68 60 52
3	56 40 24
4	48 16
5	64 32

Obrázek 2.3: Progresivně logaritmický nakloněný časový rámec, převzato z [2].

na které chce analytik nebo automatický systém sledovat data. V případě zaznamenání zajímavého jevu se může drill-down operací analytik přesunout na nižší vrstvu.

Materializace pouze dvou vrstev a dopočítávání mezivrstev za běhu však nemusí být efektivním řešením. Otázkou zůstává, zda mít předpočítané dopředu všechny, některé nebo žádné mezivrstvy. Efektivní metodou je metoda *popular path cubing*, která materializuje kostky od minimální vrstvy po vrstvu pozorovací. Vybírá kostky nacházející se podél populární cesty zanořování a ukládá tak nejpoužívanější pohledy na data. Metoda využívá hyperlinkové stromové struktury *H-tree* pro minimalizaci potřebných výpočetních a prostorových zdrojů.

2.4 Dolování frekventovaných vzorů

Jak bylo popsáno v [2], vzorem může být množina, sekvence nebo struktura. Vzor je pak považován za frekventovaný pokud jeho počet výskytů splňuje podmínku minimální podpory. Pro dolování frekventovaných vzorů existuje řada algoritmů, většina ale vyžaduje několikanásobný průchod daty. Navíc se v průběhu přijímaní proudu dat mohou frekventované množiny vyvíjet. Frekventované se mohou postupně stát nefrekventovanými a naopak. Zaznamenávat všechna data nebo frekventované množiny je však v prostředí datových proudů nemožné.

Pro překonání těchto problémů existují dva přístupy. Můžeme omezit sledovaný počet množin. Tento přístup má ale omezené využití a vypovídací schopnost, protože omezíme rozsah zkoumání na předem určené množiny.

Druhou možností je odvodit přibližné množiny odpovědí. Jak se v praxi ukázalo jsou přibližné odpovědi dostačující.

Algoritmus Lossy Counting

Tento algoritmus aproximuje frekvenci prvků v rámci uživatelem stanovené maximální odchylky a po úpravách dokáže najít i frekvenční množiny. Uživatel nejprve zadá hranici minimální podpory σ a velikost maximální možné chyby ε. Příchozí datový proud je rozdělen na části o šířce $w = \lceil 1/\varepsilon \rceil$. a do N je ukládán počet doposud přijatých prvků.

Algoritmus zaznamenává všechny prvky do seznamu frekvencí a pro každý udává přibližnou četnost výskytů f a maximální možnou chybu Δ záznamu. Při příjetí nové částí proudu jsou prvky vloženy do seznamu frekvencí. Pokud prvek v seznamu ještě neexistuje je vytvořen nový s počtem 1, existuje-li je inkrementován jeho počet. Vyskytuje-li se prvek v b-té částí, nastavíme $\Delta{=}b - 1$. Pokud počet zpracovaných prvků N odpovídá násobku šířky části w, provedeme přezkoumání frekvenčního seznamu. Smažeme všechny

prvky splňující $f + \triangle \leq b$, kde b je aktuální část. Algoritmus se tak snaží zachovat malou velikost frekvenčního seznamu, mazání prvků ale způsobuje podhodnocení některých výsledků.

Chyby algoritmu jsou proto způsobeny pouze podhodnocením smazaných prvků. Víme ale, že $b \leq N/w$, proto $b \leq \varepsilon N$. Skutečná frekvence prvku může být nejvíce $f + \triangle$, což znamená že prvek může být maximálně podhodnocen o εN. Výstupek jsou pak všechny prvky s frekvencí aspoň $(\sigma N - \varepsilon N)$, to zaručí přítomnost všech frekventovaných a několika nefrekventovaných množin. Celková paměťová náročnost algoritmu je $\frac{1}{\varepsilon}\log(\varepsilon N)$.

Nalezení frekventovaných množin je náročnější než nalezení frekventovaných prvků, protože počet možných množin roste exponenciálně oproti možným prvkům. Vyrovnáme se s tím, použitím upraveného algoritmu *Lossy Counting*. Oproti původnímu postupu se snažíme načíst co nejvíce částí β datového proudu do paměti a do frekvenčního seznamu ukládáme frekventované množiny namísto prvků. Existuje-li už množina v frekvenčním seznamu, přičteme k ní počet výskytů v načtených částech. Pokud množina vyhovuje $f + \triangle \leq b$, odstraníme jí ze seznamu a pokud má množina frekvenci $f \geq \beta$ a nevyskytuje se v seznamu, tak je do seznamu vložena s $\triangle = b - \beta$ jakožto maximální odchylkou.

2.5 Klasifikace v proudu dat

Klasifikace, jak již bylo v úvodu zmíněno, rozděluje vzorky dat do diskrétních tříd, pomocí atributu návěští. Klasifikační algoritmus má dvě fáze. V první fázi trénujeme klasifikační model. Získáme dostatečně velký blok již předem označených dat, na jehož základě se snažíme vytvořit model reprezentující s dostatečnou přesností rozdělení trénovacích dat do tříd. Ve fázi druhé aplikujeme model na neoznačená data a určíme do které třídy jednotlivé vzorky patří.

V tradičních databázích se provádí trénovací fáze několika průchody nad relativně statickými daty, proto je model většinou konstruován dávkovými algoritmy. Fáze druhá se podobá zpracování datovými proudy, příchozí vzorky jsou po přijetí okamžitě klasifikovány.

Jak jsme již poznali na dřívějších úlohách dolování v proudech dat, nemůžeme využít stejných metodik. Několikanásobný průchod daty je pro nás nerealizovatelný a musíme se vyrovnat s charakteristickými rysy datových proudů.

2.5.1 Problémy klasifikace v datových proudech

Zde jsou uvedeny základní problémy, se kterými se musí vyrovnat klasifikační algoritmy datových proudů.

Nekonečnost

Potenciální nekonečnost datových proudů je nejznámějším problémem a nejtypičtější vlastností dolování v datových proudech. Znemožňuje nám použít typickou klasifikaci dávkovým přístupem a je zdrojem většiny dalších problémů. Každý klasifikační algoritmus, pracující v prostředí datových proudů, se musí s tímto problémem vyrovnat. Musí být jednoprůchodový a umožňovat aktualizaci klasifikačního modelu nově příchozími označenými daty.

Concept drift

Concept drift neboli změna konceptu dat nastává z důvodu časové proměnlivosti obsahu datových proudů a jedná se o jeden z nejvíce zkoumaných problémů. Problémem je, že

mnoho skutečných aplikací závisí na skrytých souvislostech, ovládaných pro nás skrytými nebo neznámými změnami. Typickým příkladem může být nakupování zákazníků, které se může výrazně měnit působením médií, ročních období, nových zákonů atd. Změnu konceptu můžeme rozlišit na změnu náhlou nebo změnu pozvolnou.

Klasifikační algoritmus proto musí být dostatečně citlivý, aby zachytil i pozvolnou změnu konceptu, a musí rychle reagovat na náhlé změny, ale zároveň musí být robustní proti šumu.

Concept evolution

Concept evolution nastává v okamžiku, kdy se v datovém proudu vyskytne vzorek dříve neznámé třídy. Příkladem může být bezpečnostní systém, který zachytí úplně nový druh útoku. Algoritmus nereagující na tento druh změny pak chybně označuje všechny výskyty nové třídy, do doby něž dostane trénovací data se zmíněnou třídou. Řešením může být shlukování prvků nové třídy do doby než bude označena.

Zapomínání

Dalším problémem může být ztráta již získaných informací. Postupem času se modely dat aktualizují a ztrácejí reprezentace neaktuálních dat. Přijde-li třída, která se dlouho v proudu nevyskytovala, může být její klasifikace nepřesná nebo naprosto chybná. Proto jsou vyvíjeny techniky, jak zachovat dostatek historických dat a při tom pořád správně klasifikovat nová.

2.5.2 Klasifikace pomocí rozhodovacích stromů

Pro klasifikaci datových proudů byly vyvinuty speciální algoritmy na základě rozhodovacích stromů. Jejichž hlavní výhodou je inkrementální aktualizace.

Algoritmy byly uvedeny v [2].

Algoritmus Hoeffding tree

Jedná se o rozhodovací strom s inkrementálním učením, který produkuje skoro identické rozhodovací stromy jako tradiční dávkové algoritmy. Jako datovou strukturu používá *Hoeffdingovy stromy*, založené na myšlence, že i malý vzorek dat může postačovat k výběru optimálního dělícího atributu.

Myšlenka je založena na matematickém základu, prezentovaném *Hoeffdingovou mezí* (*aditivní Chernoffova mez*). Máme-li N nezávislých pozorování náhodné proměnné r v rozsahu R, kde r je míra informačního zisku a vypočteme průměr $\bar{r}$ vzorku, pak Hoeffdingova mez udává, že skutečný průměr r je nejméně $\bar{r} - \varepsilon$ s pravděpodobností $1 - \delta$, kde δ určí uživatel a

$$\varepsilon = \sqrt{\frac{R^2 \ln(1/\delta)}{2N}}$$

Algoritmus *Hoeffding tree* používá Hoeffdingovu mez, aby s vysokou pravděpodobností nalezl nejmenší počet N vzorků, potřebných pro výběr dělícího atributu. Hoeffdingova mez je nezávislá na pravděpodobnostní distribuci a měla by vybrat stejný dělící atribut jako při použití nekonečného počtu vzorků.

Vstupem algoritmu je sekvence trénovacích vzorků S, definovaná atributy A a parametr udávající přesnost δ. Vybereme vyhodnocovací funkci $G(A_i)$, kterou může být informační zisk, *Gini index*, *gain ratio* nebo jiná výběrová míra. V každém uzlu rozhodovacího stromu

maximalizuje hodnotu $G(A_i)$, pro zbývající atributy A_i. Cílem je najít nejmenší počet n-tic N, pro které bude Hoeffdingova mez splněna.

Výběr dělícího atributu uzlu se provádí takto. Máme atribut A_a, který dosáhl nejvyšší hodnoty G a atribut A_b, který je druhý největší. Je-li $G(A_a) - G(A_b) > \varepsilon$, pak je nejlepším dělícím atributem atribut A_a, s jistotou $1 - \delta$.

Jediné statistiky, které je potřeba uložit jsou počty n_{ijk} pro hodnotu v_j atributu A_i s návěštím třídy y_k. Paměťovou náročnost vypočteme z počtu atributů d, maximálního počtu hodnot atributu v, počtu tříd c a maximální hloubky stromu l. Paměťové požadavky se pak rovnají $O(ldvc)$, což jsou nízké nároky oproti jiným rozhodovacím stromům.

Hoeffdingův strom má vysokou přesnost na malý počet trénovacích vozků, vystačí si s jedním průchodem, je inkrementální a může klasifikovat data i při své konstrukci. S vyšším počtem trénovacích dat se i postupně zvyšuje jeho přesnost. Má ale i své slabiny. Příliš dlouho se zabývá atributy s podobnou rozdělovací schopností, není plně paměťově optimalizován a nedokáže se vyrovnat se změnou konceptu.

Very Fast Decision Tree

VFDT je modifikací Hoeffding tree algoritmu. Vylepšuje jeho nedostatky v oblasti rychlosti a nároků na paměť. Úpravy zahrnují zkrácení času ztraceného nad atributy s podobnou rozdělovací schopností, vypočítání funkce G až po přijetí několika trénovacích vzorků, ignorování špatných dělících atributů, úsporu paměti při nedostatku a vylepšenou inicializační metodu.

VFDT pracuje rychle s dobrými výsledky, ale pořád neumí zpracovat změnu konceptu.

Concept-adapting Very Fast Decision Tree

Označován zkratkou *CVFDT* je výsledkem pokračování vývoje VFDT. Je navržen tak, aby se dokázal vyrovnat se změnou konceptu. K tomu používá plovoucí okno. To přijímá nové vzorky a zbavuje se starých, čímž se přizpůsobuje novým konceptům. Tato technika je ale citlivá na velikost plovoucího okna. Je-li okno příliš velké, změna konceptu se v něm ztratí a naopak při malé velikosti nemusí obsahovat dostatek vzorků pro vytvoření přesného modelu.

CVFDT chytře aktualizuje model dle obsahu plovoucího okna. Upravuje statistiky v uzlech stromu inkrementováním hodnot spojenými s novými vzorky a dekrementováním hodnot spojenými se starými vzorky. Při změně konceptu dat se může stát, že některé uzly již nevyhovují Hoeffdingově mezi. Takové uzly se odstraní a namísto nich se vloží nový podstrom s nejlepším dělícím atributem v kořenu. To se ale neprovede okamžitě, nahrazení proběhne, až je nový podstrom natrénován na vyšší přesnost než má starý podstrom.

CVFDT dosahuje vyšší přesnosti a mnohem menší velikosti v časově proměnlivých datech než VFDT.

2.5.3 Klasifikace pomocí souboru klasifikátorů

Klasifikace pomocí souboru klasifikátorů se liší od předchozích metod, které používaly pro klasifikaci jen jeden model, snahou použít několik modelů zároveň. Tento přístup je podporován jak teoretickými, tak i praktickými důvody, které byly posány v [11]. Podstatou je získat co nejvíce expertních názorů a ty následně sloučit do jednoho výsledku, který zaručuje vyšší správnost. Je to ten samý princip, jako u stanovení složité lékařské diagnózy, kde se snažíme získat názory různých lékařů, abychom vyloučili možnost chyby. Pokud tedy

natrénujeme několik rozdílných klasifikátorů a při klasifikaci budeme převádět kombinaci jejích výstupů na jeden, můžeme, ale nemusíme překonat přesnost nejlepšího klasifikátoru. Určitě však dosáhneme výrazného snížení rizika produkce zvláště špatného výsledku.

Budeme-li mít n=25 vzájemně nezávislých klasifikátorů, kde každý klasifikátor bude mít míru chyby $\varepsilon = 0.35$, pak je pravděpodobnost, že soubor klasifikátorů udělá chybu neboli, že špatný výsledek r bude mít aspoň 13 klasifikátorů z 25 následující:

$$\sum_{i=13}^{25} P(r) = \sum_{i=13}^{25} \frac{n!}{r!(n-r)!} \varepsilon^i (1-\varepsilon)^{n-i} = 0.06$$

Podmínkou pro snížení společné chybovosti je, aby použité klasifikátory měly menší chybovost než náhodné hádání a byly navzájem rozdílné, dělaly odlišné chyby. Odlišnost klasifikátorů lze zajistit několika způsoby, můžeme natrénovat každý klasifikátor na jiné části trénovacích dat, k čemuž datové proudy poskytují optimální podmínky. Další možností je použít nestabilní klasifikátory nebo vzorkovat data.

V současné době je vyvíjeno značné množství algoritmů pro klasifikaci pomoci souboru klasifikátorů, ve všech se však více, či méně vyskytuje stejná struktura postupu zpracování. V prvé řadě je nutné zachytit proudící trénovací data do jednoho nebo více plovoucích oken. Ty reprezentují jednotlivé bloky dat, nad kterými budou klasifikátory trénovány. Po spuštění jsou natrénovány klasifikátory na jednotlivých oknech a jejich společná odpověď může být určena hlasováním většiny, váženým průměrem určeným dosavadní přesnosti klasifikátorů nebo jinou sofistikovanější metodou. Další částí je nahrazování starých klasifikátorů. Metodika nahrazování bývá různá, nejčastěji se však jedná o nahrazení nejméně přesného klasifikátoru novým přesnějším.

Díky výměnám klasifikátorů se algoritmy dobře přizpůsobují změnám konceptu dat, proto bývá tato část často předmětem zájmu a mohou být implementovány i metody pro detekci míry změny. Dalším častým rozšířením bývá detekování nových tříd a jejich shlukování, pro usnadnění označení vzorků.

Algoritmy postavené na principu souboru klasifikátorů jsou efektivnější než zmíněné rozhodovací stromy a řeší širší problematiku klasifikace v datových proudech. Hlavními výhodami jsou vysoká přesnost, dobré zpracování concept driftu, možnost použít za základní klasifikátory již známé dávkové algoritmy a možnost širokého rozšíření dle aktuální problematiky dat.

Závěrem práce jsou uvedeny tři metody klasifikace v datových proudech pomocí souboru klasifikátorů, které jsou zaměřeny na zpracování problému změny konceptu dat.

2.6 Shluková analýza proudu dat

Mnoho aplikací pracujících nad proudy dat využívá automatické shlukové analýzy. Zpracování dat lidmi je jednou z nejnákladnějších položek společností, proto je předzpracování příchozích dat do skupin vítanou pomocí. I shluková analýza se musí potýkat s prostředím datových proudů. To bylo důvodem vyvinutí několika metod. Vzhledem k omezené paměti jsou ukládány souhrnné statistiky minulých dat. Je používána taktika rozděl a panuj, příchozí data jsou rozdělena na bloky. Pro ně se spočítají statistiky a sloučí se do jedné. Shlukování musí být prováděno inkrementálně a využívá se principu nakloněného časového rámce. Dalšími metodami je využívat off-line procesů nad statistikami při dotazech, kde je to možné a počítat sumarizace mikroshluků a z nich následně makroshluky.

Informace o shlukové analýze v proudu dat, byly čerpány z [2].

Algoritmus STREAM

STREAM je jednoprůchodový shlukovací algoritmus vyvinutý na řešení *k-medians* problému. Problémem je rozdělit N datových bodů do k shluků, tak aby jejich kvadratická odchylka mezi body a středy shluků byla minimální. Řešením je vložit podobné body do stejného shluku a rozdílné body do jiných shluků.

Pro rychle zpracování datových proudů bere STREAM algoritmus data po blocích m bodů odpovídajících velikosti hlavní paměti. Algoritmus uchovává jen informace o k středech shluků, společně s jejich váhou určenou počtem přiřazených bodů, ostatní informace zahazuje. Až je získán dostatečný počet středů, jsou i ony shlukovány, aby vytvořili nový $O(k)$ shluk středů. Tento postup se opakuje na každé úrovni a je zachováno nejvíce m bodů.

Algoritmus má časovou složitost $O(kN)$ a prostorovou $O(N^{\varepsilon})$, pro $\varepsilon < 1$. Vytváří mnoho kvalitních k-median shluků, ale nerespektuje vývoj, ani časovou granularitu dat.

CluStream algoritmus

Jedná se o shlukovací algoritmus pro *evolving* (vyvíjející se) data, založený na uživatelem specifikovaných on-line shlukovacích dotazech. Rozděluje shlukovací proces na dvě části, *on-line* část a *off-line* část. On-line část počítá, ukládá a spravuje sumární statistiky datových proudů pomocí mikroshluků. Off-line část zpracovává makroshluky a odpovídá na uživatelské dotazy pomocí uložených sumárních statistik založených na nakloněném časovém rámci.

Mikroshluk pro množinu d-rozměrných bodů $X_1, \ldots, X_n$ s časovými razítky $T_1, \ldots, T_n$ je definován jako $(2d + 3)$-tice $(CF2^x, CF1^x, CF2^t, CF1^t, n)$, v níž $CF2^x$ obsahuje sumu druhých mocnin hodnot dat každé dimenze, $CF1^x$ obsahuje sumu hodnot v každé dimenzi. $CF2^t$ obsahuje sumu druhých mocnin časových razítek, $CF1^t$ obsahuje součet časových razítek a n značí počet bodů uložených v mikroshluku.

On-line zpracování je ještě rozděleno na dvě fáze, na sběr statistických kolekcí dat a na aktualizaci mikroshluků. Makroshluky navíc umožňují uživateli řídit tvorbu makroshluků, provádění analýzy vývoje shluků a umožňují nahlížet na shluky v různých časových horizontech.

Kapitola 3

Malware analysis system

Jedná se o analytický systém vyvíjený v rámci projektu Systém pro zvýšení bezpečnosti v prostředí Internetu analýzou šíření škodlivého kódu. Je navržen jako vysoce výkonný, robustní, dostupný, jednoduše rozšiřitelný, spolehlivý podnikový systém pro dolování dat. Podrobně byl systém popsán v [6].

3.1 Vlastnosti systému

Vzhledem k účelu systému, splňuje systém tyto vlastnosti:

1. Podporuje různorodé zdroje dat. Oproti běžným analytickým systémům podporuje MAS zpracování dat, nejen z relačních databází a datových skladů, ale i z prostředí datových proudů. Navíc dokáže pracovat s doplňkovými informacemi z ostatních zdrojů (např. webová služba).

2. Umožňuje tvořit pokročilé analytické procesy. Analytické procesy se v systému definují prostřednictvím jazyka platformy Microsoft .NET a jeho systémových knihoven. To přináší větší kontrolu nad definicí procesu než definování skrze grafické rozhraní. Procesy mohou být navíc spuštěny na požádání, v časových intervalech nebo událostí.

3. Má úložiště výsledků dolovacích úloh. Výsledky dolování mohou být ukládány ve zvoleném formátu pro budoucí porovnání a analýzu jejich vývoje.

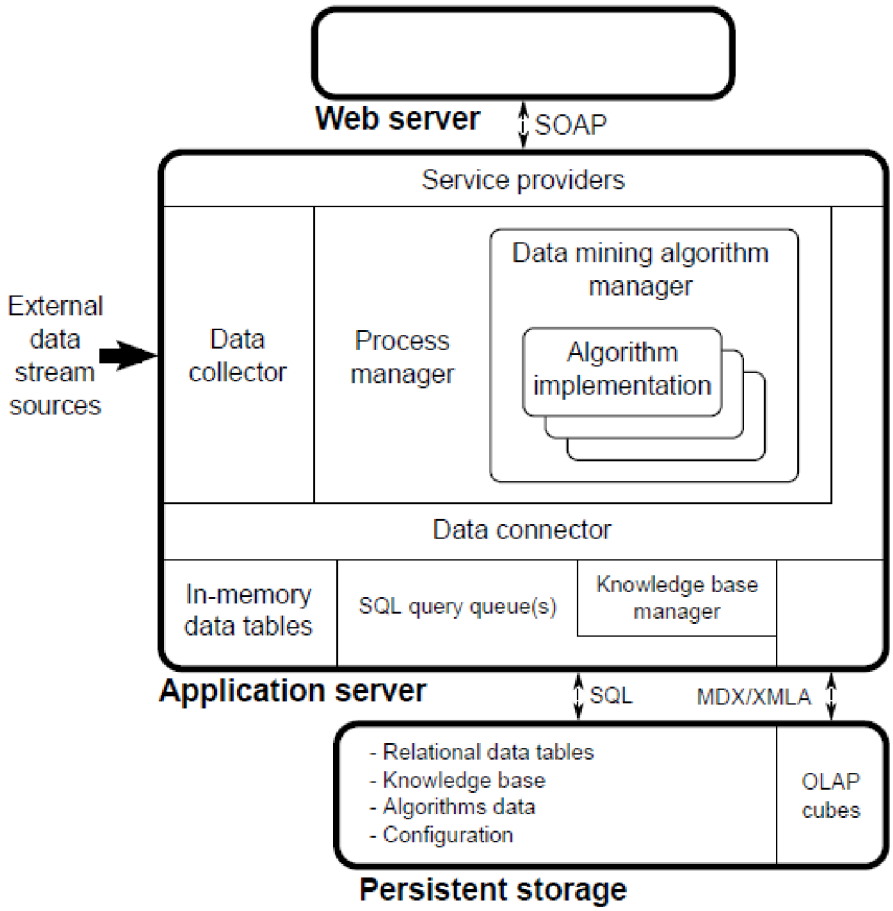
4. Pracuje jako služba. Systém je postaven na architektuře klient, server. Klient má možnost pracovat se serverem skrze webovou aplikaci, také k němu může přistupovat externí aplikací, pomocí vystavených služeb.

5. Rozšiřitelnost. Dolování dat je progresivním oborem, proto je nutné poskytovat jednoduše rozšiřitelné prostředí. Cílem projektu je vytvořit systém, jehož algoritmy a vizualizace bude možné jednoduše rozšiřovat.

6. Zaručuje vysokou dostupnost. Systém umožňuje dlouhodobý paralelní běh procesů i jejich spuštění v libovolném čase. Proto musí být všechny úpravy procesů možné i při jejich běhu.

3.2 Architektura systému

Jak již bylo zmíněno, systém je postaven na architektuře klient-server. Oddělení těchto dvou vrstev umožňuje umístit server na vysoce výkonný stroj, zatímco klient může pracovat z libovolného počítače. Systém je rozdělen na několik modulů a vrstev, což je zřejmé z následujícího obrázku 3.1.



Obrázek 3.1: Architektura systému MAS, převzato z [6].

Střední vrstva se skládá z několika komponent. *Data collector* přijímá a předzpracovává externí data. *Process manager* spouští a spravuje analytické procesy. Přístup k datům zajišťuje vrstva služeb *Data connector*. Ta poskytuje jednotný interface pro práci s daty v paměti, SQL databázi, OLAP kostkách i bází znalostí. Systém navíc spravuje přístup k datům na základě priorit, aby dokázal efektivně rozložit zátěž na zdroje dat. Poslední komponentou je *Service provider*, poskytující služby pro komunikaci s externími aplikacemi.

Celý systém je vyvíjen na platformě .NET 4.0 v jazyce C# a jednotlivé komponenty jsou realizovány jako vzájemně komunikující služby.

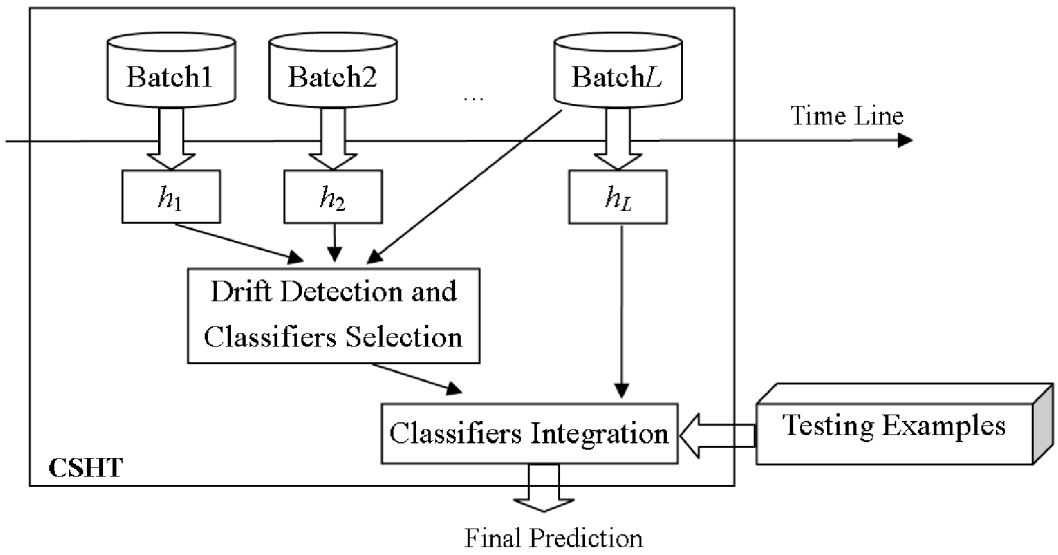
Kapitola 4

Vybrané metody

Byly vybrány tři metody pro klasifikaci datových proudů pomocí souboru klasifikátorů. Vzhledem k zaměření analytického systému MAS se zvolené metody zabývají problémem změny konceptu dat. Metody byly záměrně vybrány vzhledem ke své rozdílnosti a aktuálnosti.

4.1 Algoritmus CSHT

Algoritmus, jak byl popsán v [1], je postaven na principu souboru klasifikátorů a změnu konceptu detekuje pomocí testu hypotézy (*hypothesis test*). Algoritmus přijímá jednotlivé trénovací bloky dat a předpokládá, že nové vzorky pro klasifikaci odpovídají stejné distribuci, jako nově přijatý blok. Tento předpoklad trvá do doby, než je přijat blok další a je proto nutné po přijetí nového bloku aktualizovat celý systém.



Obrázek 4.1: Správa a aktualizace klasifikátorů, převzato z [1].

Aktualizace probíhá následovně. Máme soubor klasifikátorů $H = \{h_1, \ldots, h_l, \ldots h_{L-1}\}$, kde h_l s $l = 1, \ldots, L-1$ je základní klasifikátor natrénovaný na datovém bloku $D(l)$ a $L-1$ je počet klasifikátorů. V čase L obdržíme nový trénovací blok dat $D(l)$.

1. Natrénujeme nový klasifikátor h_L na novém bloku dat $D(l)$ a vložíme ho do souboru použitelných klasifikátorů S. Soubor klasifikátorů S je před každou aktualizací vyprázdněn.

2. Pro každý bývalý klasifikátor detekujeme změnu konceptu. Pokud ke změně nedošlo, vložíme klasifikátor do souboru S.

3. Váha každého klasifikátoru je určena jako

$$w_i = \log \frac{1 - error_{D(l)}(h_i)}{error_{D(l)}(h_i)}$$

   Kde $error_D(h)$ je chybovost klasifikátoru h na vzorku dat D o n prvcích.

$$error_D(h) \equiv \frac{1}{n} \sum_{x \in D} \delta(f(x), h(x))$$

$$\delta(f(x), h(x)) = \begin{cases} 1 & f(x) \neq h(x) \\ 0 & jinak \end{cases}$$

4. Klasifikace je pak provedena váženým hlasováním všech klasifikátorů v S, kde I značí funkci vracející 1, pokud klasifikátor h_i přiřadí prvek x třídě ω_j, a vracející 0 v případě opačném.

$$H(x) = argmax_{\omega_j} \sum_{h_i \in S} w_i I(h_i(x) == \omega_j)$$

Detekce změny konceptu dat je určena změnou v distribuci dat. Pokud došlo ke změně distribuce, došlo také ke změně konceptu dat. Test hypotézy byl používán k porovnání dvou algoritmů nad stejnými distribucemi dat, zde je ale použit pro porovnání distribucí dat pod stejnými klasifikátory. Detekce změny konceptu klasifikátoru h_i v souboru klasifikátorů $i = 1, \ldots, L-1$ se provede takto:

1. Navzorkujeme k skupin záznamů z posledního bloku dat $D(L)$. Přičemž každá skupina obsahuje $n \geq 30$ záznamů.

2. Vypočteme chybovost klasifikátoru $error_D(h)$ pro každou skupinu a uložíme je do sekvence $\{r_1, \ldots, r_k\}$.

3. Spočteme $\bar{r}$, s a statistiku $|t|$, pro $p = error_D(h)$.

$$\bar{r} = \frac{1}{k} \sum_{i=1}^{k} r_i; s^2 = \frac{1}{k-1} \sum_{i=1}^{k} (r_i - \bar{r})^2$$

$$t = \frac{\bar{r} - p}{s/\sqrt{k}}$$

4. Pokud je $|t| \geq t_{\alpha/2}(k-1)$, kde $t_{\alpha/2}(k-1)$ je kritický bod splňující

$$P\left\{t(k-1) > t_{\alpha/2}(k-1)\right\} = \alpha/2$$

   pak nejspíš došlo ke změně konceptu a klasifikátor není vložen do S.

Výpočty vyžadují tři parametry n, k a α. n je velikost vzorku získaného bootstrap vzorkováním a je požadována větší nebo rovna 30. k je počet vzorkovaných skupin a měl by být určen dle dostupných výpočetních zdrojů. α udává míru změny, při které je klasifikátor odmítnut. α se však vztahuje pouze k prostředí a ne k přesnosti klasifikátoru, proto může být jednoduše statisticky zvolena (např. na hodnotu 0.01).

4.2 Algoritmus pro detekci DoS

Tento algoritmus, uvedený v [13], byl nejprve navržen pro detekci datových proudů způsobujících DoS útoky, ale výměna vstupních dat za data jiné aplikační domény, funkčnost nezmění. Odlišností algoritmu oproti ostatním je, že trénuje více klasifikačních algoritmů na jednom bloku dat.

Pro každý blok dat natrénujeme několik klasifikátorů, přiřadíme jim podle přesnosti váhy a vybereme K nejlepších pro vytvoření souboru klasifikátorů.

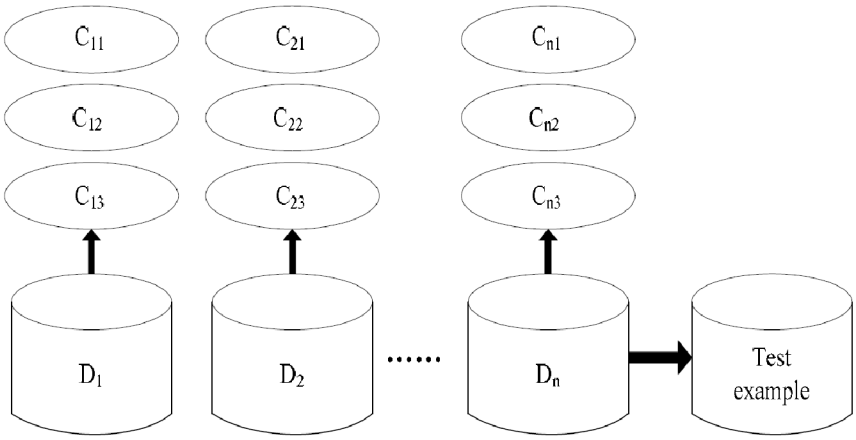
Značení je určeno takto:

C_{im}: m-tý základní klasifikátor bloku i
w_{im}: váha C_{im}
r_{im}: přesnost C_{im}
A_n: soubor nejlepších klasifikátorů bloku n
D_n: datový blok
$D' = D_{n-r+1} \cup D_{n-r} \ldots \cup D_n$: předchozí datové bloky
D_{n+1}: nově příchozí blok

Algoritmus:

Vstup: poslední bloky dat D_n, D', D_{n+1}
Výstup: soubor klasifikátorů An

1. Natrénuj základní klasifikátory $C_{n1}, \ldots, C_{nm}$ z D_n.
2. Vypočti pro všechny klasifikátory přesnost nad D_{n+1} a spočti jejich váhu.
3. Pro každý datový blok $D_i \in D'$: Natrénuj klasifikátory $C_{i1}, \ldots, C_{iv}$ nad D_i a spočti $w_{i1}, \ldots, w_{iv}$.
4. Vyber do souboru klasifikátorů A_n nejlepších K klasifikátorů z $C_{i1}, \ldots, C_{nv}$, i=n-r+1.
5. Vrať A_n.

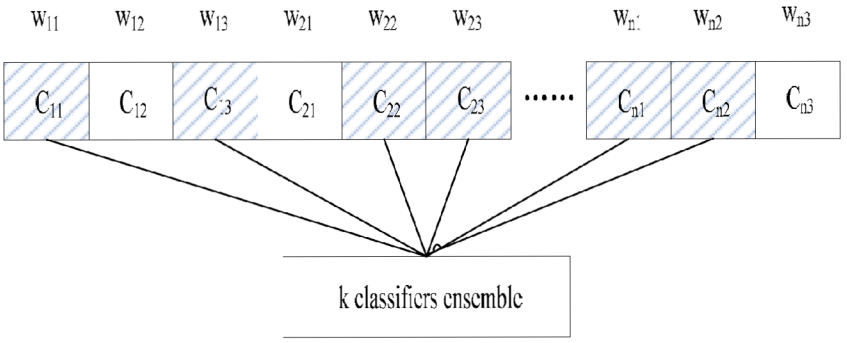


Obrázek 4.2: Trénování klasifikátorů, převzato z [13].

Přesnost klasifikátoru r_{im} se spočte jednoduchým poměrem počtu správně klasifikovaných vzorků oproti počtu všech vzorků. Váhu klasifikátoru pak spočteme jako přesnost klasifikátoru děleno celková přesnost (suma přesností všech klasifikátorů).

$$w_{im} = r_{im} / \sum_{i=1}^{r} \sum_{j=1}^{v} r_{ij}$$

Nejpřesnější klasifikátory pak budou přirozeně obsazovat prvních K pozic v souboru klasifikátorů.



Obrázek 4.3: Složení souboru klasifikátorů, převzato z [13].

4.3 Algoritmus DWAA

Poslední metoda je založena na odměňovací strategii a byla popsána v [12]. V mnoha metodách je použito odměňování, či trestání klasifikátorů dle jejich výsledků, ve většině ale přidávají pevné hodnoty. Algoritmus DWAA velikost odměny klasifikátoru zvyšuje v poměru k počtu správných odpovědí ostatních klasifikátorů. Pokud správně klasifikuje pouze jeden klasifikátor, je mu výrazně, ale ne příliš zvýšena váha, aby mohl snadněji ovlivnit celkové rozhodnutí. Opatrnost je na místě, protože se může jednat o šum.

Odměňovací strategie:

$(x_1, y_1) \ldots (x_n, y_n)$: reprezentuje testovací instance,
kde x_i obsahuje atributy a y_i představuje návěští třídy
S: velikost souboru klasifikátorů
$n(x)$: počet klasifikátorů se správnou odpovědí
$weight[i]$: váha i-tého klasifikátoru
$h_i(x)$: je odpověď i-tého klasifikátoru

	$n(x_i) \geq \frac{S}{2}$	$n(x_i) < \frac{S}{2}$
$h_j(x_i) = y_i$	$\frac{1}{n(x_i)}$	$1 - \frac{2[n(x_i - 1)]}{S}$
$h_j(x_i) \neq y_i$	0	0

Tabulka 4.1: Výpočet odměňovacích hodnot, převzato z [12].

Metodu je možné použít pouze s touto odměňovací strategií, kdy se klasifikátory trénují na předposledním bloku dat a vyhodnocují na nejnovějším. Vyhodnocení následně vynuluje váhy klasifikátorů a dle správných odpovědí přičte odpovídající odměny. Nejhorší klasifikátor je pak nahrazen novým.

Algoritmus však obsahuje ještě jednu modifikaci, která má za účel rozšířit váhovou mezeru mezi dobrými a špatnými klasifikátory.

Postup úpravy je následující:

Vstup: váhy klasifikátorů $weight[1 \ldots n]$
Výstup: modifikované váhy klasifikátorů

1. Vybereme nejvyšší $best$ a nejnižší $worst$ váhu klasifikátorů.
2. Vypočteme průměr $mean = (best - worst)/2$.
3. Lineárně transformujeme váhový vektor z rozsahu $(worst, best)$ na rozsah (-1,1) vzorcem:
$$weight'[1 \ldots n] = \frac{weight[1 \ldots n] - mean}{best - worst}$$
4. Aplikujeme vzorec pro úpravu:
$$weight''[1 \ldots n] = \frac{2 \arctan(factor \times weight'[1..n])}{\pi}$$

$$factor = \frac{(1 + \sqrt{1 - D^2})\pi}{D^2}$$

Kde D je standardní odchylka váhového vektoru klasifikátorů.

5. Lineárně transformujeme $weight''[1 \ldots n]$ zpátky na rozsah $(worst, best)$.

Úprava má za následek, že klasifikátory mají pořád stejné váhové pořadí, ale dobré jsou mnohem blíže k nejlepším a špatné k nejhorším klasifikátorům. Váhová mezera se proto rozšíří a zvýší se citlivost souboru klasifikátorů na změnu konceptu.

Kapitola 5

Návrh a implementace

Počínaje touto kapitolou se zaměřím na praktickou část diplomové práce. Uvedu zde, jak jsem postupoval při návrhu a implementaci systému pro klasifikaci v datových proudech. Popíšu implementaci jednotlivých metod. Zmíním také úpravy, které bylo potřeba provést pro měření, testování algoritmů klasifikačního systému a pro napojení na Malware analysis system.

Celá práce byla vytvořena v programovacím jazyce C# [4] a ve vývojovém prostředí Visual Studio 2012. Důležité je také zmínit, že algoritmy byly vytvořeny na základě .NET Frameworku 4. Ten představil nové programovací prostředky, pro tvorbu vícevláknových a asynchronních aplikací.

Implementace byla rozdělena na několik částí. Nejprve jsem získal a naprogramoval základní klasifikátory, které byly použity v souborech klasifikátorů. V další části jsem vytvořil systém pro klasifikaci v proudu dat a přidal potřebné třídy pro simulaci vstupu a zpracování výstupu. Následně jsem implementoval vybrané metody a nakonec vše upravil pro napojení na analytický systém.

5.1 Základní klasifikátory

Pro práci se souborem klasifikátorů bylo nejprve potřeba vybrat vhodné základní klasifikátory. Metody CSHT a DWAA vyžadují jeden klasifikační algoritmus a metoda DoS vyžaduje několik navzájem rozdílných algoritmů. Proto jsem vybral algoritmy Naivní Bayes, SVM a rozhodovací strom C4.5.

Z časových důvodů jsem se zaměřil na návrh a implementaci klasifikačního systému a zvolených metod. Pro implementaci SVM a rozhodovacího stromu C4.5 jsem použil knihovnu Accord.NET[1].

5.1.1 Naivní Bayes

Naivní Bayes je jednoduchý statistický klasifikátor [10]. Rychle se učí, je nenáročný na výpočetní prostředky. Poskytuje kvalitní výsledky, které nejsou citlivé na šum v datech. Klasifikace pomocí souboru klasifikátorů upřednostňuje spíše jednodušší a rychlé klasifikační algoritmy před složitými, ale přesnými. Proto jsem zvolil Naivní Bayes jako referenční klasifikátor a použil ho ve všech metodách.

[1] Knihovnu Accord.NET lze nalézt na adrese `<http://code.google.com/p/accord/>`.

Algoritmus jsem implementoval tak, aby při trénování vyžadoval pouze blok trénovacích dat a informace o typech atributů v něm. Následně se vytvoří tabulka, do které se vypočtou všechny hodnoty potřebné pro klasifikaci. Klasifikace pak probíhá velice rychle, dosadí se chybějící hodnoty dle klasifikovaného vzorku a vrátí se třída, do které vzorek nejpravděpodobněji patří.

Klasifikátor je upraven, aby zpracovával diskrétní i spojité atributy a je v něm použita Laplaceova korekce.

5.1.2 Support Vector Machine

Metody CSHT a DWAA si vystačily s Naivním Bayesem, ale metoda DoS byla založena na použití několika rozdílných algoritmů. Proto jsem jako další klasifikátor zvolil algoritmus SVM [7]. Je diametrálně odlišný oproti algoritmu Naivní Bayes. Transformuje trénovací data do vyšších dimenzí, ve kterých hledá optimální rozdělení dat na třídy. Tato metoda přináší dle –odkaz na literaturu– velice přesné výsledky, nevýhodou je však dlouhá trénovací doba.

5.1.3 Rozhodovací strom C4.5

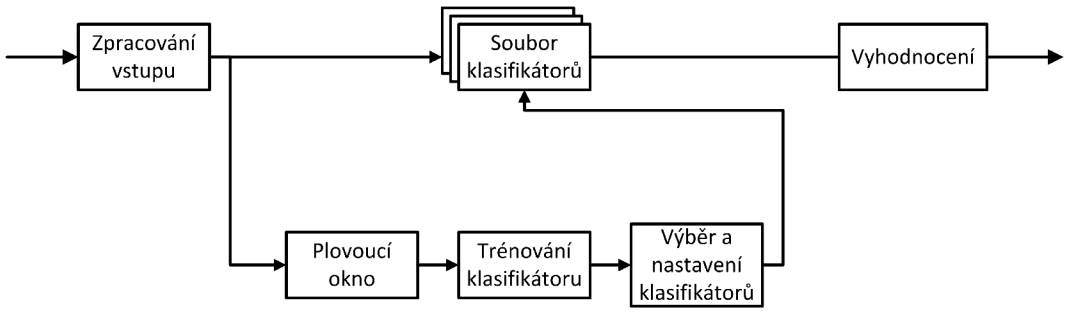
Jako třetí klasifikátor jsem zvolil algoritmus z rodiny rozhodovacích stromů. Algoritmus C4.5 dosahuje vysoké přesnosti, jak lze pozorovat v [3]. Není ale tak složitý jako SVM algoritmus a ani příliš jednoduchý jako Naivním Bayes. Hledá atributy, které nejlépe rozdělí trénovací data do jednotlivých tříd.

Po výběru posledního klasifikačního algoritmu pro metodu DoS, jsem se zaměřil na návrh a implementaci klasifikačního systému.

5.2 Systém pro klasifikaci v datových proudech

Návrh a implementace systému byla jednou ze stěžejních částí práce. Vzhledem k aplikačnímu prostředí datových proudů musel být systém navržen tak, aby mohl pracovat co nejrychleji a byl schopen zpracovat pokud možno všechna příchozí data. Příjem dat z datového proudu může trvat neurčitou dobu a může přenášet rozličné množství dat.

Systém jsem navrhl jako aplikaci, kterou data pouze protékají, proto se systém skládá z paralelních nebo navazujících částí. Na obrázku 5.1 je tento systém zobrazen.



Obrázek 5.1: Tok dat klasifikačním systémem.

Zpracování datového proudu probíhá následovně. Datový proud vstupuje do zpracování vstupu, v tom se rozdělí data na data trénovací a data pro klasifikaci. Zároveň se převede formát dat z datového proudu na datové typy používané systémem.

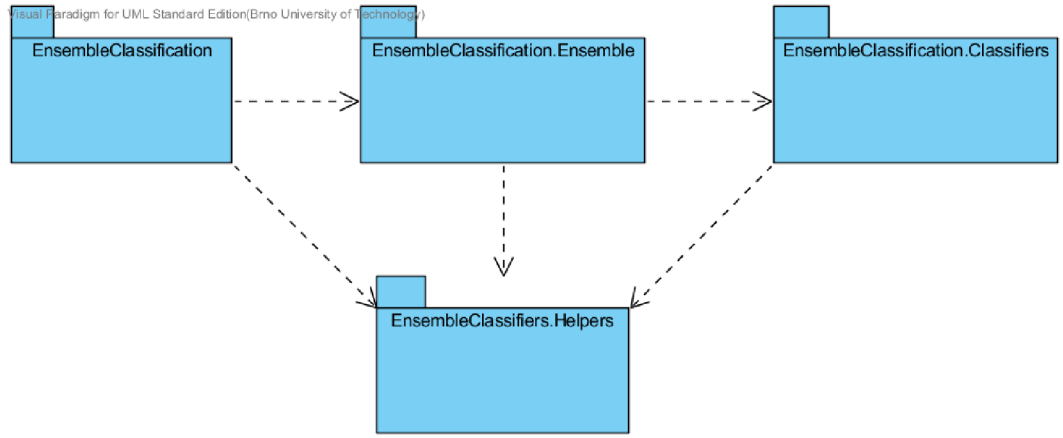
Data čekající na klasifikaci se pošlou klasifikovat do souboru klasifikátorů, kde je každý záznam klasifikován všemi aktivními klasifikátory. Dle zvolené metody je pak vytvořena společná klasifikace a jako výstup je odeslán celý záznam s vyplněným návěštím třídy, nebo pouze samotné návěští. Vyhodnocení následně převezme výsledek klasifikace.

Trénovací data jsou oproti klasifikačním sbírána do plovoucího okna. Po jeho naplnění je na shromážděném bloku dat natrénován nový klasifikátor a plovoucí okno se vyprázdní. Natrénovaným klasifikátorem se následně aktualizuje soubor klasifikátorů, dle pokynů zvolené metody.

Klasifikace a trénovaní klasifikátorů pracuje nezávisle na sobě, do chvíle, kdy začne aktualizace souboru klasifikátorů. Pro zvýšení propustnosti systému jsem navíc využil možností .NET Frameworku 4 tak, abych optimálně paralelizoval části systému. Této problematice se budu věnovat v samostatné části.

5.3 Struktura systému

Základ systému je rozdělen na čtyři balíčky, jak je znázorněno na obrázku 5.2. Hlavním balíčkem je balíček EnsembleClassification, který obsahuje řídící třídu *StreamControl*, ovládající celý systém. Vnořený balíček Ensemble obsahuje třídy s klasifikačními metodami a balíček Classifiers obsahuje třídy základních klasifikátorů. Pomocné třídy jsou uloženy v balíčku Helpers.



Obrázek 5.2: Diagram balíčků.

5.4 Reprezentace dat

Pro reprezentaci dat v systému jsem zvolil datové typy *DataTable* a jeho součást *DataRow* z jmenného prostoru System.Data. Třídy v tomto prostoru jsou určeny pro efektivní správu dat z různých datových zdrojů.

Jeden záznam je pak reprezentován řádkem *DataRow*, který sdílí strukturu jednotlivých atributů z *DataTable* a dá se s ním pracovat obdobně jako s polem objektů. Pro blok dat jsem použil datový typ *DataTable*, který se skládá z kolekce řádků *DataRow*. Klasifikátory proto klasifikují jednotlivé řádky *DataRow* a trénují se nad blokem dat *DataTable*, se kterým se dá dle potřeby pracovat jako s kolekcí, nebo jako s tabulkou v databázi. Použijeme-li ještě

LINQ (Language-Integrated Query) [8], který dodává dotazovací funkce do jazyka C#, pak se můžeme nad *DataTable* dotazovat podobně jako pomocí SQL.

5.5 Pomocné třídy

V klasifikačním systému jsem vytvořil dvě pomocné třídy. První byla třída *StreamSettings*, která obsahuje potřebné informace o datech přijímaných z datového proudu. Ukládá se do ní pozice návěští a typy přijímaných atributů, zda jsou diskrétní nebo spojité. Dále obsahuje pro zpřehlednění volitelné názvy jednotlivých atributů. Nejsou-li určeny, jsou systém vytvořeny názvy vlastní.

Druhou pomocnou třídou je třída *DataTableCreator*, která dle údajů v *StreamSettings* generuje *DataTable* s odpovídající strukturou.

5.6 Paralelizace systému

S nástupem vícejádrových procesorů a jejich rozšířením, i mezi běžné uživatele, se paralelizace algoritmů stala jednou z hlavních metod, jak zvýšit rychlost aplikací. Proto jsem se i já rozhodl využít možnosti paralelně zpracovávat části klasifikačního systému. Systém byl s touto myšlenkou koncipován již od začátku a jeho struktura skládající se z na sobě nezávislých, nebo z několika navazujících modulů, tomuto přístupu vyhovovala.

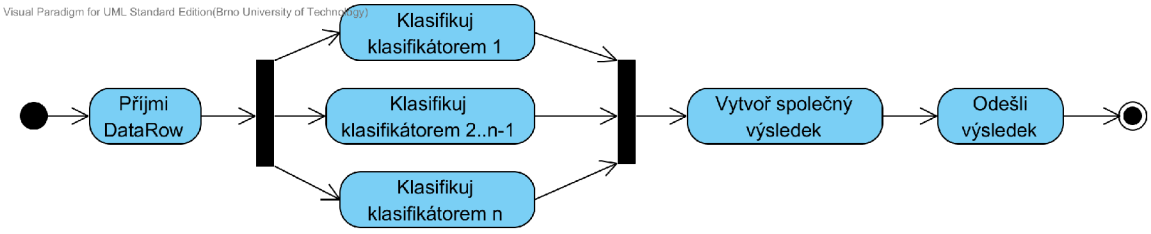
5.6.1 Task Parallel Library

Velkou výhodou bylo vydání .NET Frameworku 4, který mimo jiné představil nový programovací model pro tvorbu vícevláknových a asynchronních aplikací. Novinkou je Task Parallel Library (TPL) [9], která ke stávajícím paralelním prostředkům přidává koncept úloh (*Task*) a přináší nové paralelní datové struktury. TPL umožňuje pracovat s částmi kódu, jako s úlohami, které mohou představovat nová vlákna nebo asynchronní operace. Úlohou může být libovolná metoda, která se předá pomocí delegáta třídě *Task*. Delegát je typem reference metod, který se po přiřazení chová jako delegovaná metoda. Takto spuštěné metody pak pracují jako nezávislé operace, na které se čeká jen v případě, že požadujeme jejich výstup dříve než doběhnou.

Oproti pouhému ulehčení práce s tvorbou vláken se ale největší výhoda TPL skrývá v propracovaném systému správy úloh. TPL detekuje výpočetní prostředky poskytované algoritmu a adekvátně škáluje paralelní zpracování úloh. V případě jednojádrového systému je použito sekvenční zpracování a u vícejádrových nebo víceprocesorových strojů jsou využity algoritmy pro efektivní rozdělení zátěže. Úlohy navíc běží nad množinou předem vytvořených vláken (*Thread pool*), ve které se použitá vlákna znovu recyklují. Tím se vyhneme náročnému vytváření nových vláken při spouštění nových úloh. Použitím TPL nedochází ke zbytečnému nárůstu režie u starších strojů a algoritmy jsou prováděny co nejefektivněji, dle aktuální situace. Vytvoření aplikace se stejnými možnostmi, bez použití TPL, by bylo velice náročné.

5.6.2 Rozdělení systému

Paralelizaci systému jsem provedl rozdělením algoritmu do úloh. Hlavní dvě úlohy jsou klasifikace a aktualizace souboru klasifikátorů. Úloha klasifkace je zobrazena na obrázku 5.3. Vždy, když jsou přijata data pro klasifikaci, je vytvořena nová úloha. Ta obdrží na vstupu řádek, který má klasifikovat. Následně úloha vytvoří pro každý aktivní klasifikátor v souboru klasifikátorů novou úlohu a klasifikuje všemi klasifikátory přijatý řádek. Výsledky úloh se nakonec posbírají a dle váženého hlasování se vyhodnotí výstupní návěští třídy.



Obrázek 5.3: Diagram aktivity klasifikace.

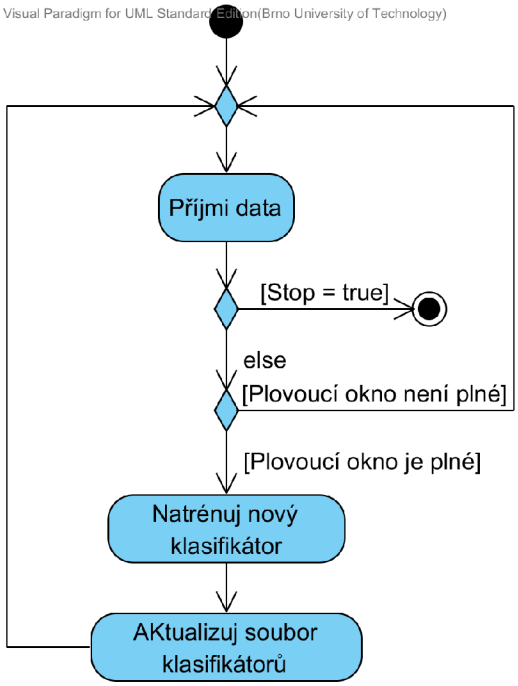
Hodnoty na výstupu takto zpracovávaných klasifikací nemusejí být ve stejném pořadí, ve kterém přišly. Proto je druhým vstupním parametrem klasifikační úlohy odkaz na datový typ *BlockingCollection*. Ten je jedním z nových datových typů v TPL a reprezentuje frontu bezpečnou pro vícevláknové operace, přidávání a odebírání prvků.

Úloha aktualizace souboru klasifikátorů se od klasifikace liší. Jedná se o jednu úlohu, která je spuštěna po inicializaci metody klasifikace a běží do doby, kdy je metoda ukončena. Plánovač zpracování úloh je o tomto postupu informován atributem TaskCreationOptions.LongRunning. Díky tomu plánovač pozná, že úloha bude zpracovávána dlouhou dobu a namísto použití množiny vláken pro ni vytvoří nové samostatné vlákno. Aktualizace je znázorněna na obrázku 5.4. Po spuštění úlohy se začnou zachytávat trénovací data do *BlockingCollection*, z té jsou následně vybírány do *DataTable*, reprezentující plovoucí okno. Úloha se při čekání na výběr prvků z *BlockingCollection* uspí a nedorazí-li další prvky, tak po čase nastane timeout, následovaný kontrolou, zda se nemá ukončit. Při pokračování vkládá prvky do plovoucího okna až do jeho naplnění.

Při naplnění plovoucího okna, se na něm natrénuje nový klasifikátor a dle zvolené metody se zařadí do souboru klasifikátorů. V průběhu aktualizace souboru se ve vhodných částech využívá dalších úloh.

Jediným slabým místem paralelizace je výměna klasifikátorů v souboru klasifikátorů. Jelikož se v tomto místě střetávají všechny klasifikační úlohy a úloha aktualizace, tak se jedná o velice vytíženou strukturu. Soubor klasifikátorů je reprezentován polem, které obsahuje odkazy na jednotlivé aktivní klasifikátory. Zvolil jsem proto přístup vyžadující co nejmenší synchronizaci. Každá klasifikační úloha si vytvoří vlastní kopii pole klasifikátorů, což zamezí změnám klasifikátorů v průběhu klasifikace. Výměna klasifikátoru pak probíhá přepsáním reference v poli na referenci novou. Toto lze použít, protože jazyk C# specifikuje změnu reference jako atomickou operaci. Nedochází tudíž nikdy k nekonzistentnímu stavu pole klasifikátorů.

Jedinou synchronizaci, kterou jsem v systému použil, jsem zařadil na základě testů. Při nichž jsem dospěl k názoru, že je lepší po nasbírání dostatečného počtu trénovacích dat, pro novou aktualizaci, přerušit klasifikaci dat. Do doby než je aktualizace dokončena. Dosavadní nezpracované klasifikace jsou pak dokončeny na starém souboru klasifikátorů a nová data nebudou chybně klasifikována starými modely, které již nemusí odpovídat skutečnosti.



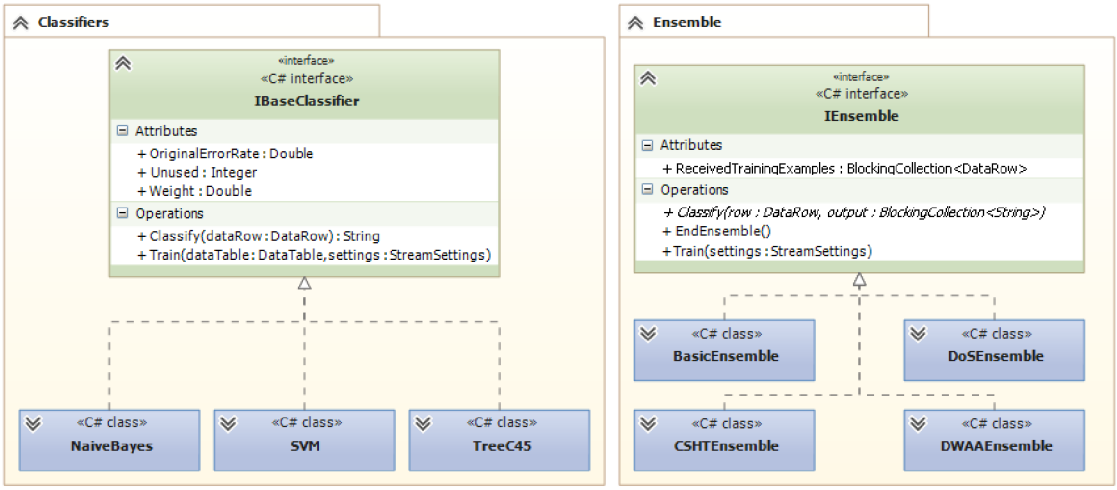
Obrázek 5.4: Diagram aktivity aktualizace souboru klasifikátorů.

Mohlo by se stát, že zbytečně ohodnotíme množství dat chybně. Cenou za vyšší přesnost je zdržení po dobu trénování a aktualizace systému. Synchronizace je provedena pomocí *ManualResetEvent*, který blokuje vlákna dokud neobdrží signál a po obdržení signálu je otevřen do doby než je ručně resetován.

5.7 Jednotná rozhraní

V systému jsou implementována dvě rozhraní. Jsou to rozhraní *IBaseClassifier* a *IEnsemble*. Jak je z obrázku 5.5 patrné, rozhraní *IBaseClassifier* definuje veřejně přístupné metody a vlastnosti (*Properties*) všech základních klasifikátorů. Atributy *OriginalErrorRate* a *Unused* jsou používány v metodě CSHT. První značí chybovost klasifikátoru na datovém bloku, na kterém byl klasifikátor natrénován a druhý zaznamenává po kolik aktualizačních cyklů nebyl klasifikátor použit. Vlastnost *Weight* reprezentuje rozhodovací váhu klasifikátoru. Klasifikátory dále obsahují metodu *Classify*, která klasifikuje vstupní řádek *DataRow* a vrací návěští třídy. Poslední metodou je metoda *Train*, přijímající blok trénovacích dat, na němž je natrénován klasifikační model klasifikátoru. Prvky tohoto rozhraní jsou využívány všemi metodami pro klasifikaci souborem klasifikátorů a umožňují jednotný přístup a práci s libovolným typem základního klasifikátoru.

Rozhraní *IEnsemble* je implementováno metodami souboru klasifikátorů. *BlockingCollection* s názvem *ReceivedTrainingExamples* je použita pro zachytávání trénovacích dat. Jelikož je zabezpečená vůči vláknům. Z fronty jsou data předávána do plovoucího okna *DataTable*, které se po naplnění použije pro trénování nového klasifikátoru. S tím souvisí metoda *Train*, která je spouštěna jako dlouho běžící úloha, na trénování nových klasifikátorů a aktualizaci souboru klasifikátorů. Metoda *Classify* se pak využívá v klasifikačních úlohách. Zbývající metoda *EndEnsemble* slouží k ukončení úlohy *Train*.



Obrázek 5.5: Diagram tříd použitých rozhraní.

5.8 Řízení systému

Správu a řízení přístupu k metodě souboru klasifikátorů má na starosti třída *StreamControl*. Úkolem třídy je inicializovat soubor klasifikátorů a spustit úlohu *Train*. Následně jsou přes třídu vkládána data pro trénování a klasifikaci. Třída obstarává vytváření úloh klasifikace a vkládání trénovacích dat do systému. Taktéž je zde ošetřeno blokování klasifikace při aktualizaci souboru klasifikátorů. Poslední úlohou třídy je ukončení celého klasifikačního systému.

5.9 Zpracování vstupu

Z popisu je zřejmé, že v systému doposud chybí zpracování příchozích dat. Tato situace nastala z několika důvodů. Není-li součástí systému zpracování vstupu, je pak možné vytvořit vlastní transformaci dat. Ta převádí datové typy používané hostitelským systémem na typy klasifikačního systému. Druhým důvodem je napojení na analytický systém MAS, který je stále ve vývoji.

Pro účely napojení na MAS byly potřebné transformace datových typů implementovány v řídící třídě. Napojení na MAS systém bude popsáno na konci kapitoly. Na testování systému však byla vytvořena zvláštní třída *DataFeeder*. Sloužící jako zdroj testovacích dat pro celý systém. Třídě se určí textový soubor, obsahující již označená data, vloží se znak oddělovače jednotlivých atributů a nastaví se, zda je atribut diskrétní nebo spojitý. Třída pak parsuje soubor po řádcích, které převádí na řádky *DataRow*. Pro zjednodušení jsou všechny diskrétní atributy převedeny na *String* a všechny spojité atributy na datový typ *Double*. Data jsou následně odeslána do řídící třídy pro klasifikaci a trénování.

Pro lepší simulaci reálného vstupu je navíc toto načítání dat realizováno jako dlouho trvající úloha. Aby se docílilo skutečné zátěže systému. Požadavky tak vznikají nezávisle na klasifikačním systému, jsou omezeny pouze přepínáním kontextu vláken a rychlostí čtení dat ze souboru.

5.10 Zpracování výstupu

Výstupem klasifikačního systému jsou ohodnocené vzorky dat, které jsou vkládány do *BlockingCollection* klasifikačními úlohami. Zpracování těchto výstupů je již na uživateli systému. Pro testování jsem však vytvořil třídu *DummyOutput*, která vlastní zmíněnou výstupní kolekci. Aby testování probíhalo efektivně, upravil jsem klasifikační metody. Jelikož se při testování klasifikují již ohodnocená data, přidal jsem porovnání výstupů metod se správným ohodnocením. Metody pak vrací jen výsledky, zda klasifikace byla úspěšná a třídy se shodují, nebo zda zvolila špatnou třídu. Tyto výsledky jsou pak ukládány do kolekce ve výstupní třídě. Ta je také implementována jako dlouho běžící úloha, která vybírá získaná data z fronty a ukládá do souborů průběžná měření přesnosti a rychlosti klasifikace. Zaznamenaná data jsem pak použil při vyhodnocování a experimentování s metodami klasifikace.

5.11 Implementace metod

V této části se budu věnovat popisu implementace jednotlivých metod klasifikace pomocí souboru klasifikátorů. Popisy metod, ze kterých jsem čerpal, jsou zaměřené na řešení problému změny konceptu a neuvádí přesný algoritmus. Proto zde uvedu postup implementace metod, změn a modifikací oproti definici.

5.11.1 Basic

Metoda Basic nebyla v kapitole 4 zmíněna. Jedná se o základní metodu, kterou jsem vytvořil pro prvotní testování systému. Metoda slouží jako referenční metoda. Neobsahuje žádné mechanismy pro adaptaci při změně konceptu a znázorňuje tak základní systém souboru klasifikátorů. Naměřené hodnoty se pak dají porovnat s výsledky jen samotného souboru klasifikátorů.

Metoda Basic pracuje následovně. Inicializuje *BlockingCollection* a *DataTable* sloužící jako plovoucí okno. Následně vstoupí do nekonečné smyčky, v ní začne vybírat data z *BlockingCollection* a předává je do plovoucího okna. Po naplnění plovoucího okna, natrénuje metoda nový klasifikátor Naivní Bayes na datech v plovoucím okně. Novým klasifikátorem nahradí nejstarší klasifikátor v souboru klasifikátorů. Soubor klasifikátorů je reprezentován polem, které velikostí odpovídá maximálnímu počtu používaných klasifikátorů a skládá se z typu *IEnsemble*. Metoda pro ukončení pravidelně kontroluje zda proměnná *stop* neindikuje, že se má klasifikační systém ukončit. Kontrola probíhá při vypršení limitu na výběr prvku z *BlockingCollection* a po ukončení aktualizace souboru.

Klasifikace probíhá u všech metod obdobně, proto ji zde zmíním pouze jednou. Metoda pro klasifikaci nejprve zjistí, zda existuje alespoň jeden klasifikátor v souboru klasifikátorů. Vytvoří pole úloh, ve kterých aktivní klasifikátory ohodnotí přijatý řádek pro klasifikaci. Výsledky jsou předány do slovníku složeného ze dvojic <návěští třídy, váha rozhodnutí>. V základní metodě mají všechny klasifikátory stejnou váhu, v ostatních se však již sčítají váhy hlasujících klasifikátorů. Ze slovníku se na závěr vybere návěští s nejvyšší vahou a určí se jako výsledek váženého hlasování.

5.11.2 CSHT

Metoda CSHT používá pro detekci změn konceptu test hypotézy. Není v ní uvedeno, jak obměňovat klasifikátory. Pouze se zmiňuje o množině použitých klasifikátorů, která neustále roste a uvádí jak detekovat klasifikátory se stejným konceptem. Ty jsou pak použity pro klasifikaci.

Proto jsem pro tuto metodu použil nejen pole s aktivními klasifikátory, ale i pole obsahující odložené klasifikátory. Toto pole může být několikrát větší než aktivní soubor klasifikátorů, jelikož reprezentuje jen historická data. Při výběru aktivních klasifikátorů se pak prochází i odložené klasifikátory, zda se jejich koncept neshoduje s aktuálním konceptem a nemáme-li už z minulosti natrénován vhodný klasifikátor. Nárůst počtu klasifikátorů nemá na rychlost metody výraznější vliv, pro klasifikaci je stále použit maximálně celý soubor klasifikátorů. Možný nárůst přesnosti při opakujících se konceptech je však značný.

Metoda CSHT při aktualizaci souboru klasifikátorů postupuje takto. Nasbírá dostatečné množství dat pro natrénování nového klasifikátoru. Natrénuje nový klasifikátor Naivní Bayes, vypočte pro něj míru chybovosti a váhu klasifikátoru na datech, která byla právě použita pro trénování. Při výpočtu se klasifikují všechna trénovací data novým klasifikátorem, za pomoci úloh. Následně se inkrementuje u všech odložených klasifikátorů počet kol, po která nebyly klasifikátory použity.

V dalším kroku proběhne detekce změny konceptu dat. Probíhá stejně, jak je uvedeno v sekci 4.1 Algoritmus CSHT. Detekce proběhne nad aktivními i nad odloženými klasifikátory a vrátí pole všech použitelných klasifikátorů, jejichž koncept se shoduje s konceptem posledního bloku trénovacích dat. Následuje výpočet vah použitelných klasifikátorů. Do souboru klasifikátorů se vloží nejnovější klasifikátor a zbylý počet se doplní z množiny použitelných klasifikátorů, počínaje těmi, které mají nejvyšší váhu.

Klasifikátory které se nevejdou do souboru jsou zahozeny, protože předpokládáme, že již máme minimálně n lepších klasifikátorů se stejným konceptem, kde n je velikost souboru klasifikátorů. Vyřazené klasifikátory ze souboru klasifikátorů jsou však uloženy do pole odložených klasifikátorů. Je-li plné, pak jsou nahrazeny klasifikátory, které nebyly použity nejdelší dobu.

5.11.3 DoS

Metoda detekce DoS používá více druhů klasifikačních algoritmů a také uchovává nepoužívané klasifikátory. Konkrétně udržuje klasifikátory natrénované z několika posledních bloků trénovacích dat.

Aktualizace souboru klasifikátorů začíná naplněním plovoucího okna, na němž jsou natrénovány tři klasifikátory. Jmenovitě to jsou Naivní Bayes, rozhodovací strom C4.5 a SVM. Tyto klasifikátory nejsou v tomto aktualizačním kole použity, protože se jejich přesnost a váha počítají až nad následujícím blokem trénovacích dat. Máme-li klasifikátory z minulého kola, pak se vloží do množiny uložených klasifikátorů a odstraní se klasifikátory natrénované nad nejstarším blokem dat. Pro uložené klasifikátory se spočte přesnost a váha dle definice v sekci 4.2 Algoritmus pro detekci DoS. Do souboru klasifikátorů jsou vybrány klasifikátory s největší přesností.

Největší modifikací oproti původní specifikaci algoritmu, byla úprava výpočtu finální klasifikace. Metoda původně pracovala jen se dvěma výstupy, normálním stavem a DoS útokem. Díky provedené změně vyhodnocení na vážené hlasování může metoda klasifikovat data do více tříd.

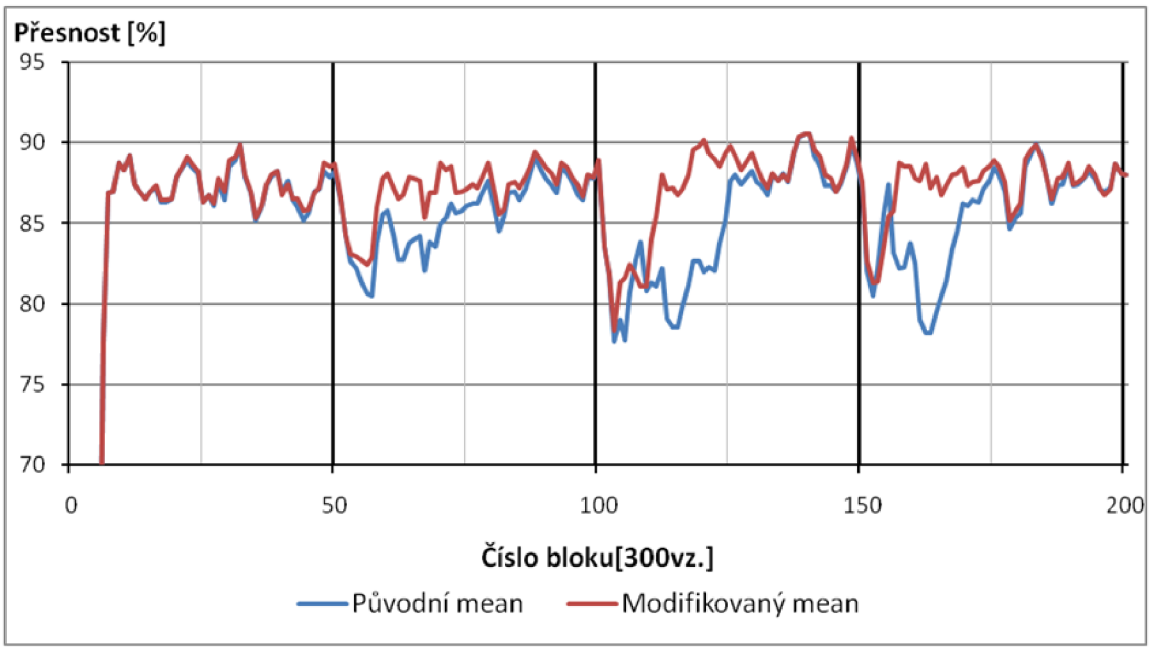
5.11.4 DWAA

Metoda DWAA je založena na odměňovací strategii a je podobná základní metodě Basic. Dle specifikace obsahuje pouze pole souboru klasifikátorů, nemá tudíž žádné další odložené klasifikátory.

Při aktualizaci souboru klasifikátorů se nad trénovacími daty natrénuje nový klasifikátor Naivní Bayes. Ten je vložen do souboru klasifikátorů až v příštím kole, jelikož se jeho váha počítá až na dalším bloku trénovacích dat. Vkládáme-li klasifikátor do plného souboru, pak jím nahradíme klasifikátor, který měl v minulém kole nejmenší váhu. Po vložení nového klasifikátoru se přepočítají váhy na nejnovějším bloku dat. Klasifikují se všechny vzorky trénovacích dat a při správné klasifikaci je klasifikátoru zvýšena váha o hodnotu uvedenou v sekci 4.3 Algoritmus DWAA.

Po spočtení vah klasifikátorů může být aktualizace ukončena, nebo se může provést ještě úprava vah. Ta zvětší váhovou mezeru mezi dobrými a špatnými klasifikátory, čímž zvýší citlivost na změnu konceptu dat. Postup úpravy je také uveden v sekci 4.3 Algoritmus DWAA. Zarazil mě ale výpočet hodnoty $mean = (best - worst)/2$, kde $best$ a $worst$ je nejvyšší, respektive nejnižší váha klasifikátoru. Tato hodnota určuje hranici mezi dobrými a špatnými klasifikátory a jsou od ní vzdalovány váhové hodnoty klasifikátorů.

Po úvaze jsem výpočet hodnoty upravil na $mean = (\sum weight(i))/count$, kde $weight(i)$ je váha klasifikátoru i a $count$ je počet aktivních klasifikátorů v souboru. Provedenou změnou se klasifikátory lépe rozdělí na dobré a špatné.



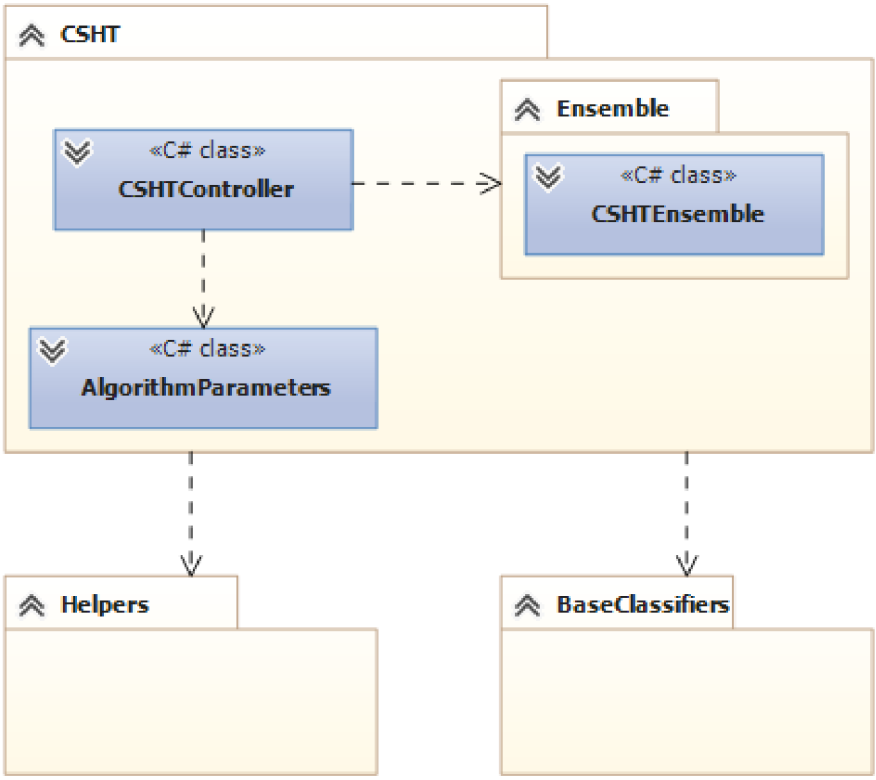
Obrázek 5.6: Porovnání přesnosti klasifikace po úpravě hodnoty $mean$.

Nevím zda se jednalo ve specifikaci algoritmu o chybu, ale naměřené výsledky na obrázku 5.6 jasně znázorňují zlepšení oproti původní variantě. Detaily způsobu měření jsou uvedeny v následující kapitole Experimenty.

5.12 Napojení na MAS

Malware analysis system pracuje s algoritmy implementovanými v jazyce C# jako s knihovnami, které nemají žádný odkaz na analytický systém. Tyto knihovny musí splňovat určité požadavky. Úvodní třída algoritmu musí dědit ze základní třídy, která je shodná pro všechny procesy analytického systému. Dále musí obsahovat třídu se vstupními parametry algoritmu, jejíž instance je uvedena atributem [AlgorithmParameters]. Základní třída analytického systému odhaluje abstraktní metodu *Run*, která reprezentuje vstupní bod procesu. Vytvořené knihovny mohou potom být použity nejen analytickým systémem, ale mohou být jednoduše spuštěny i u vývojáře algoritmu.

Pro nasazení mnou implementovaných metod do analytického systému, jsem musel provést několik úprav. V prvé řadě jsem změnil strukturu projektu. Třídy se základními klasifikátory a jejich rozhraní *IBaseClassifier* jsem přesunul do balíčku BaseClassifiers. Pomocné třídy zůstaly v balíčku Helpers, ale doplnil jsem k nim rozhraní *IEnsemble*, implementované klasifikačními metodami. Nakonec jsem pro každou metodu vytvořil samostatný balíček, který strukturou odpovídá diagramu na obrázku 5.7.



Obrázek 5.7: Struktura metody CSHT pro napojení na MAS.

Balíček každé metody obsahuje řídící třídu *Controller*, třídu s nastavením *AlgorithmParameters* a balíček Ensemble, ve kterém je implementace zvolené klasifikační metody. Třída *AlgorithmParameters* uchovává vstupní parametry algoritmu. Parametry jsou uvedeny atributy, určujícími zda jsou povinné, jakou mají defaultní hodnotu a jejich popisem. Tyto parametry jsou použity v řídící třídě *Controller*, která je vstupním bodem algoritmu. *Controller* splňuje všechny požadavky analytického systému. Konkrétně dědí ze základní

abstraktní třídy *ClassificationAlgorithmBase*, čímž navíc získá přistup k trénovacím datům typu *CaseDataSet*. *Controller* tvoří spojovací vrstvu mezi klasifikačním a analytickým systémem. Implementuje metodu *Run*, ta převede vstupní parametry do třídy *StreamSettings*, spustí klasifikační systém a trénovací úlohu. Trénovací úloha přijímá data z MAS, převádí je na typ *DataRow* a vkládá je do klasifikačního systému.

Controller dále obsahuje dvě metody *Predict*. První varianta obdrží řádek *CaseDataRow*, převede ho na *DataRow*, klasifikuje a vrátí návěští třídy. Druhá varianta pracuje stejně, ale vytváří klasifikační úlohy a výsledné klasifikované řádky ukládá do *BlockingCollection*, zadané na vstupu. Poslední metodou je metoda *Stop*, která ukončí klasifikační systém.

Po úpravách se projekt zkompiloval do knihovny .dll, která se vloží do oblasti spravované analytickým systémem. Ten automaticky za běhu detekuje nový algoritmus a přidá ho do systému. Pro spuštění v systému stačí zavolat metodu *CreateAlgorithm<název algoritmu>*, vyplnit vstupní parametry, napojit trénovací data a spustit algoritmus metodou *Run*. Klasifikace pak probíhá pomocí metod *Predict*.

Kapitola 6

Měření a experimenty

Implementace klasifikačních metod byla jen první částí práce. Druhou podstatnou částí bylo ověření funkčnosti a porovnání jednotlivých metod. Proto zde představím, jaká jsem použil testovací data a postupy pro měření výsledků klasifikačních metod. Dále uvedu naměřené výsledky a jejich zhodnocení. Na závěr popíšu své experimenty s modifikacemi metod.

6.1 Testovací data

Jako testovací data jsem použil SEA concepts dataset, který byl představen v –Odkaz na literaturu–. Dataset se skládá ze 60 000 vzorků a ze čtyř konceptů, každý o velikosti 15 000 vzorků. Vzorek se skládá ze tří spojitých numerických atributů a z návěští třídy. Spojité atributy se pohybují v rozmezí [0;10) a pouze první dva mají význam. Vzorky jsou rozděleny do dvou tříd dle koncepční funkce. Ta udává, pokud je $atribut1 + atribut2 > hranice$, pak je do návěští třídy vložena 0, jinak je vložena 1. Pro čtyři koncepty je *hranice* určena zvolenými hodnotami 8, 9, 7 a 9,5. Dataset navíc obsahuje 10% šumu.

6.2 Postup měření

Měření bylo prováděno na procesoru AMD Athlon 64 X2 Dual Core 6000+ 3,01GHz. Postup byl následující, testovací data byla načítána z textového souboru a byla vkládána do klasifikačního systému. Data byla nejprve vložena pro klasifikaci a následně pro trénování. Systém tedy klasifikoval neznámá data, která následně použil pro aktualizaci. Tímto byla simulována ideální situace, kdy systém zpracovává příchozí data z proudu dat, ta jsou potom klasifikována expertem, který je označí správnou třídou a vloží do systému pro aktualizaci.

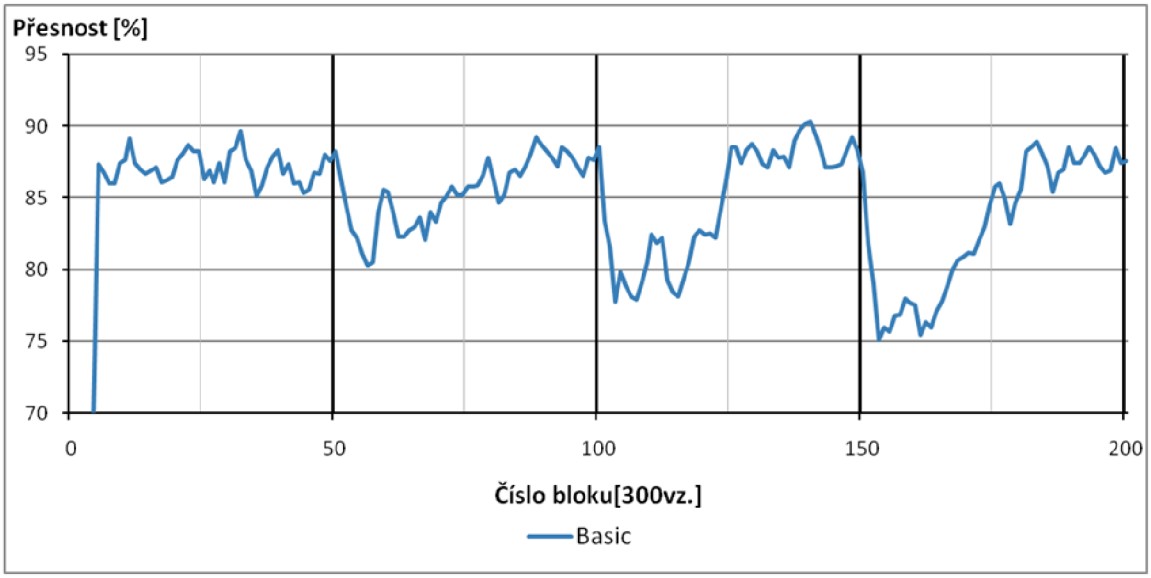
Pro zpřesnění měření byl do metod přidán navíc jeden synchronizační prvek. Blokující událost, která zamezuje přijetí dalšího vzorku dat do doby než je klasifikován předchozí. Tím jsem zamezil aktualizaci systému daty, která nebyla ještě klasifikována a znemožnil ovlivnění měření nevhodným přepínáním kontextu. Uvedené rychlosti metod jsou však naměřeny pro nesynchronizované metody, aby hodnoty odpovídaly reálnému běhu systému.

6.3 Naměřené hodnoty

Následující výsledky byly naměřeny s co nejpodobnějším nastavením, aby jsem je mohl objektivně porovnat. Plovoucí okno bylo nastaveno na velikost 500 vzorků a měření bylo prováděno na bloku 300 vzorků. V grafech je zaznačeno 200 hodnot přesnosti klasifikátorů, která se vypočetla jako poměr počtu správně klasifikovaných vzorků a celkového počtu vzorků v aktuálním měřeném bloku.

6.3.1 Metoda Basic

Metoda Basic reprezentuje klasifikační systém, který používá soubor klasifikátorů, ale není nijak upraven pro adaptaci dle změny konceptu. Hodnoty, které jsem naměřil u této metody, jsem proto použil jako referenční hodnoty souboru klasifikátorů. Všechny následující metody by měly dosahovat lepších výsledků, nebo v nejhorším případě alespoň výsledků totožných jako metoda Basic.



Obrázek 6.1: Přesnost metody Basic na trénovacích datech.

Metoda byla nastavena na použití 25 klasifikátorů a dosáhla průměrné přesnosti 84,37%, se směrodatnou odchylkou 4,12%. Rychlost metody se pohybuje kolem 1 163 klasifikovaných vzorků za sekundu.

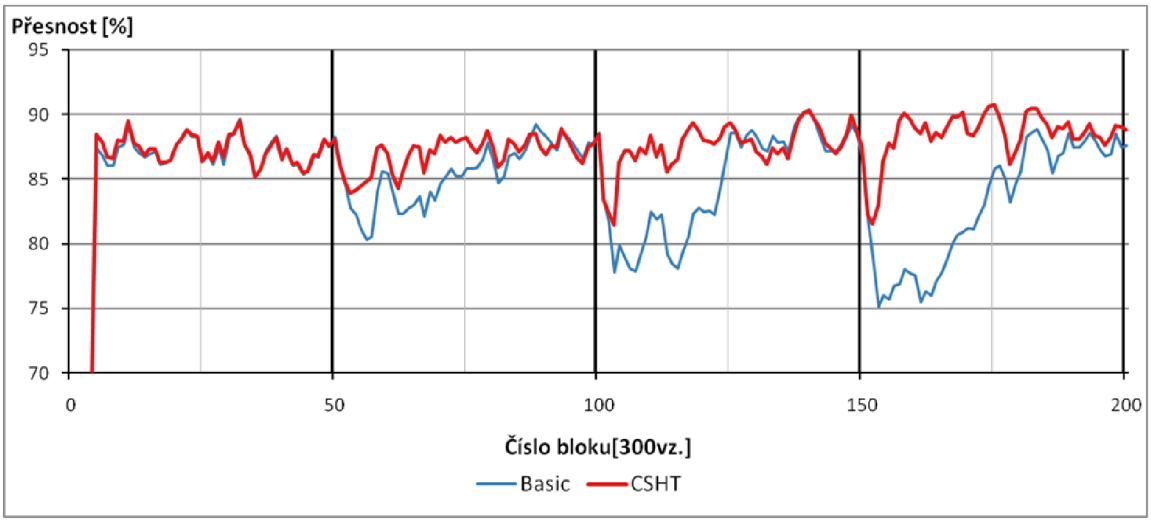
Na výsledném grafu, který se nachází na obrázku 6.1, znázorňujícím průběžnou přesnost metody, lze pozorovat tři propady. Ty nastávají v čase 50, 100 a 150 a představují tří místa změn konceptu. Jedna časová jednotka představuje blok 300 klasifikovaných vzorků. Z toho lze odvodit, že např. u druhé změny konceptu je zapotřebí nových 7 500 vzorků trénovacích dat, aby se metoda vyrovnala se změnou konceptu a vrátila se na původní hodnoty přesnosti. Cílem následujících metod je zkrátit tento potřebný čas na co nejmenší hodnotu, jelikož propady následující změny konceptu pro nás představují oblast s nejhoršími výsledky klasifikace.

V částech, kde je koncept stabilní, dosahuje soubor klasifikátorů přesnosti v rozmezí mezi 85-90%, což je dobrý výsledek, vezmeme-li v úvahu 10% šum v datech.

6.3.2 Metoda CSHT

Metoda CSHT vybírá použité klasifikátory dle testu hypotézy. Měla by proto používat vždy klasifikátory se stejným konceptem jako poslední blok trénovacích dat. Klasifikátory vyřazené metodou, jsou uloženy v množině odložených klasifikátorů. Metoda staví na již známých a prověřených matematických postupech.

Metoda byla nastavena na práci se souborem klasifikátorů o maximální velikosti 25 klasifikátorů, pro odložené klasifikátory bylo vyhrazeno 50 pozic. Vzorkování porovnávající koncepty dat bylo nastaveno na vzorkování 20 skupin o 50 vzorcích.



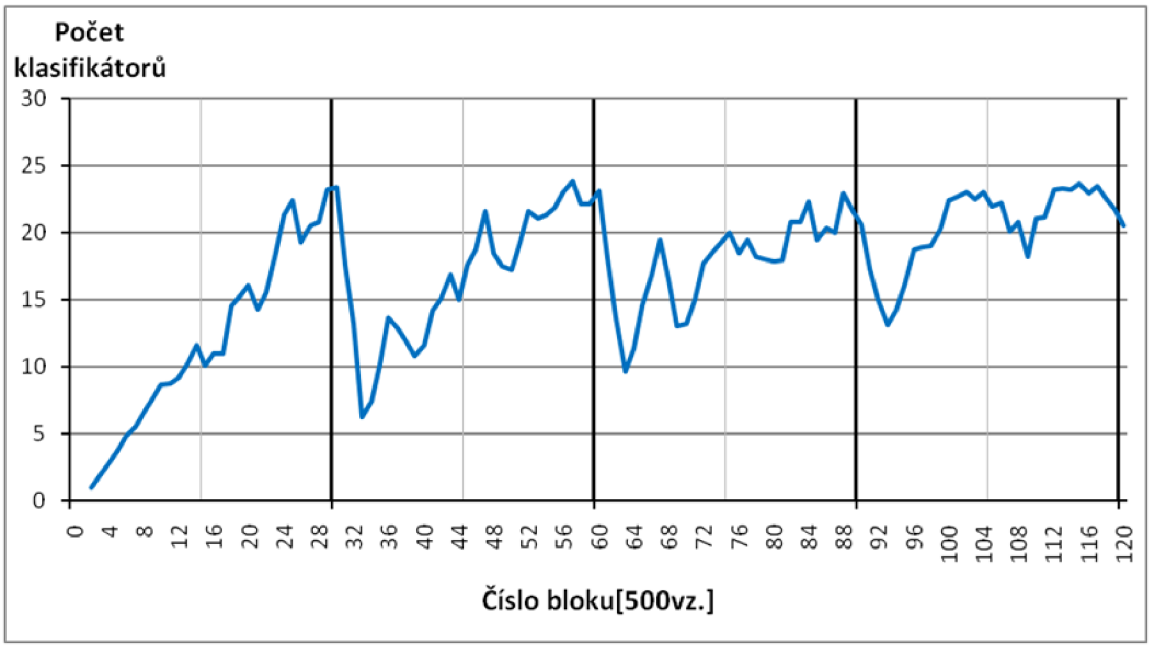
Obrázek 6.2: Přesnost metody CSHT na trénovacích datech.

Metoda dosáhla průměrné přesnosti 86,88%, se směrodatnou odchylkou 2,46% a zvládla klasifikovat 457 vzorků za sekundu. To je více než dvojnásobné zpomalení oproti základní metodě. Metoda však dosahuje mnohem lepších výsledků a pracuje s 50 klasifikátory navíc. Vzhledem k těmto okolnostem se dá metoda považovat za rychlou.

Z naměřených výsledku na obrázku 6.2 jde vidět, že se metoda přizpůsobila novým konceptům velice rychle. Při první změně konceptů nedošlo k tak velkému propadu a další dvě změny potřebovaly pouze pětinu původního času na adaptaci.

Dále uvádím graf na obrázku 6.3, který zobrazuje počet klasifikátorů zařazených do souboru klasifikátorů. Graf dokazuje, jak metoda postupně načítá nové klasifikátory v prvním konceptu, pak dochází ke změně a velká část použitých klasifikátorů je odložena. Následující změny konceptu již mají na počet použitých klasifikátorů menší vliv, jelikož metoda může použít odložené klasifikátory se stejným konceptem. Graf zobrazuje pouze 120 hodnot oproti předešlým 200 hodnotám, protože je zbytečné podrobněji měřit počet klasifikátorů, který je modifikován co 500 vzorků.

Z naměřených hodnot je patrné, že metoda efektivně vybírá používané klasifikátory. V nejhorším případě použije pouze nejnovější klasifikátor. Díky tomuto postupu není klasifikace nepříznivě ovlivněna klasifikátory, které představují starý koncept dat. Metoda je však pořád závislá na dodávaných trénovacích datech, ze kterých určuje aktuální koncept.

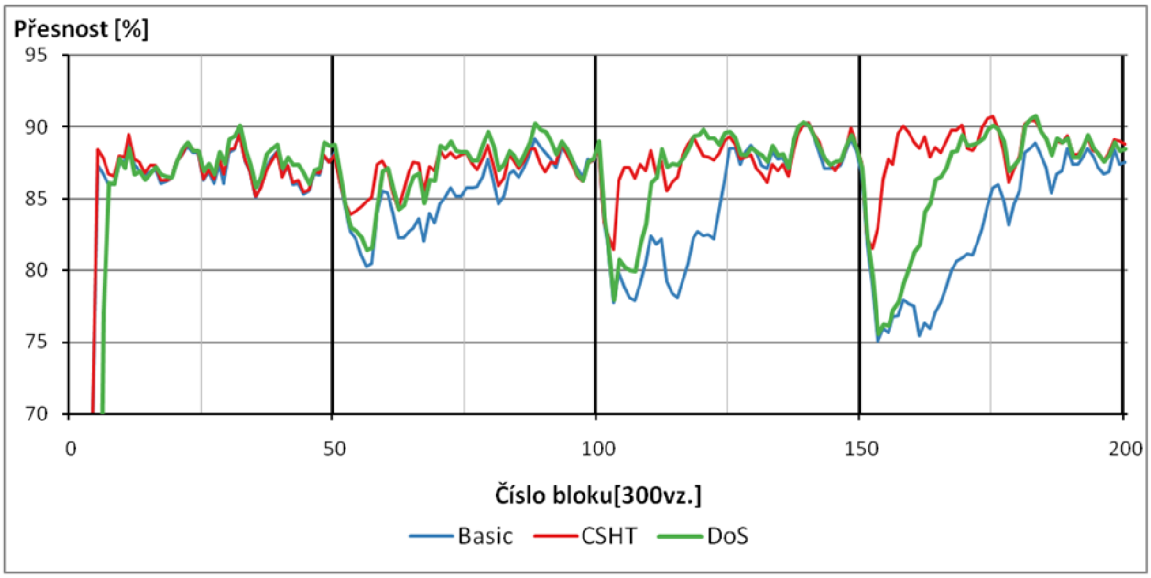


Obrázek 6.3: Průběh počtu používaných klasifikátorů metodou CSHT.

6.3.3 Metoda DoS

Metoda DoS používá při trénování více klasifikačních algoritmů, proto jsou nad každým blokem trénovacích dat natrénovány tři klasifikátory. Pro klasifikaci jsou použity nejlepší klasifikátory. Metoda si navíc pamatuje všechny klasifikátory z několika posledních trénovacích bloků, měla by těžit z větší rozdílnosti klasifikačních algoritmů.

Pro měření byla metoda nastavena na použití 25 aktivních a 51 odložených klasifikátorů. Hodnota odložených klasifikátorů byla zvolena co nejblíže počtu odložených klasifikátorů v metodě CSHT. Zároveň však musela zachovat všechny tři klasifikátory, které náleží do jednoho bloku dat.



Obrázek 6.4: Přesnost metody DoS na trénovacích datech.

Metoda dosáhla průměrné přesnosti 85,52%, se směrodatnou odchylkou 3,46%. Rychlost klasifikace se pohybovala kolem 243 vzorků za sekundu. Což je dramatické zpomalení oproti původní metodě. Příčinou je trénování třech klasifikátorů v každém kole aktualizace systému. Navíc klasifikační algoritmy SVM a rozhodovací strom C4.5 jsou výpočetně náročnější než Naivní Bayes a byly převzaty z knihovny Accord.NET, nemusí proto být plně optimalizovány.

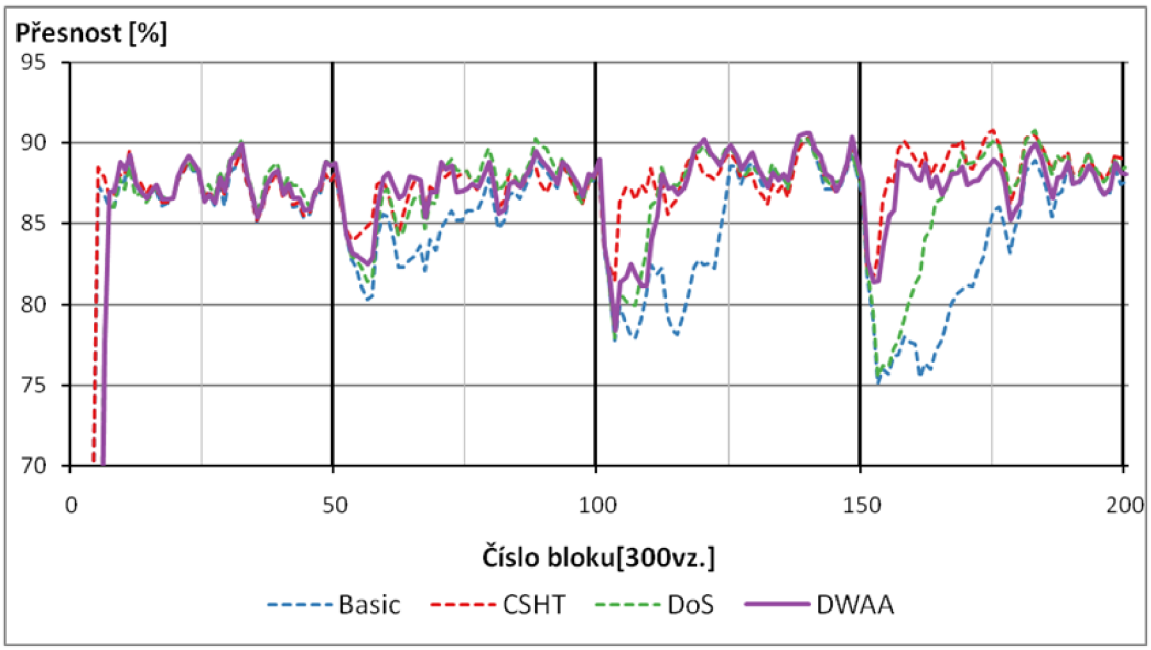
Na obrázku 6.4 jsou uvedeny výsledné hodnoty metody DoS. Metoda nepředčila výsledky metody CSHT, i když v některých oblastech stabilního konceptu dosáhla o několik procent lepší přesnosti. Co se týče adaptace metody na změny konceptu, tak se podařilo zkrátit původní čas zhruba na polovinu. Rychlejší adaptace se dá připsat na vrub trénování tří rozdílných klasifikátorů na novém bloku trénovacích dat. Metoda proto nahrazuje klasifikátory se starým konceptem 3 krát rychleji oproti základní metodě. Zůstává pak otázkou času, kdy se do souboru použitých klasifikátorů dostane dostatečný počet nových klasifikátorů, pro přehlasování klasifikátorů se starým konceptem.

6.3.4 Metoda DWAA

Metoda DWAA je poslední metodou. Na rozdíl od předchozích metod nemá metoda DWAA žádné odložené klasifikátory, což se může jevit jako značná nevýhoda. Metoda se vyrovnává

se změnou konceptu pomocí odměňovací strategie a úpravy vah klasifikátorů. Klasifikátory jsou rozděleny do dvou skupin a následně je rozšířena váhová mezera mezi těmito skupinami. Tímto docílíme větší citlivosti souboru klasifikátorů na změnu konceptu dat.

Metoda byla testována s 25 aktivními klasifikátory.



Obrázek 6.5: Přesnost metody DWAA na trénovacích datech.

Přesnost metody byla průměrně 85,8%, se směrodatnou odchylkou 2,76%. Rychlost byla 1 095 klasifikovaných vzorků za sekundu. Metoda se tak stala nejrychlejší z testovaných metod. Takové rychlosti určitě pomohlo odstranění 50 odložených klasifikátorů a jednoduchý princip metody, který nevyžaduje složité výpočty.

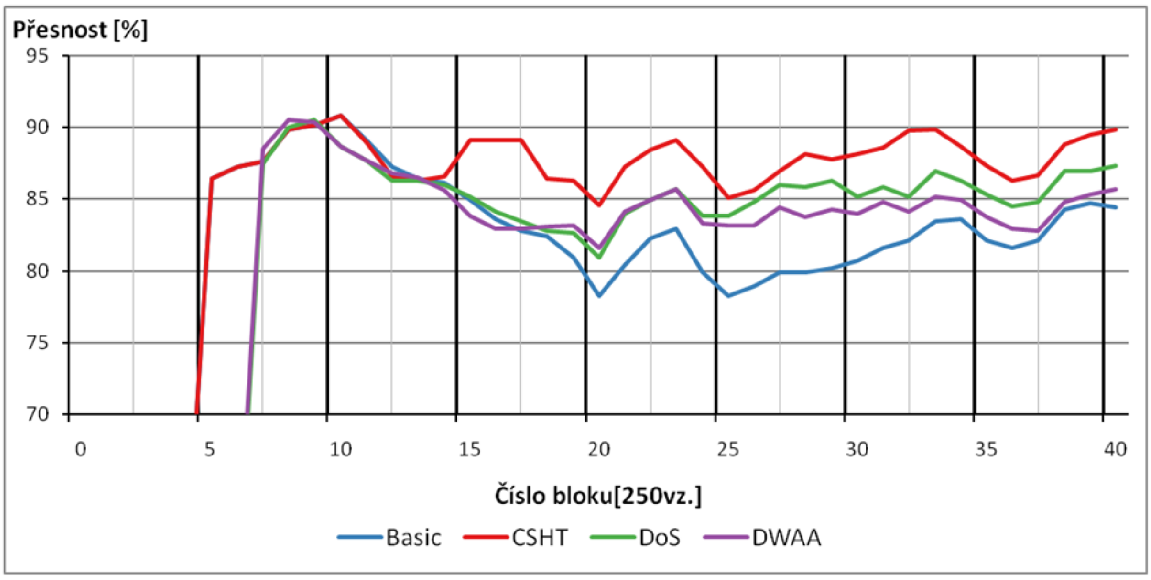
Metoda DWAA dosáhla překvapivě dobrých výsledků, jež jsou zobrazeny na obrázku 6.5. Metoda se při první a třetí změně konceptu velice blíží průběhu metody CSHT a v druhé změně se zase podobá metodě DoS. Tyto výsledky jsou překvapivé. Neočekával jsem, že metoda DWAA je schopna dosáhnout podobných výsledků jako metoda CSHT pouhým správným vyvážením klasifikátorů. Zajímavý je průběh vyrovnání se s třetí změnou konceptu, kde metoda CSHT může těžit z uložených klasifikátorů, ale metoda DWAA je jen o pár procent horší.

Co se týče stabilních částí konceptu, tak metoda DWAA podává obdobné výsledky jako další metody, i když je občas předčena metodou DoS.

6.4 Pozvolná změna konceptu

Výše popsané měření bylo provedeno na náhlých změnách konceptu, kdy se ohodnocení tříd změní v krátkém časovém úseku. V datových proudech se vyskytuje ještě jeden druh změny konceptu, tou je změna pozvolná. Při pozvolné změně dochází k postupnému vývoji. Oproti náhlé změně konceptu je zde nebezpečí, že klasifikátor bude považovat změnu za šum, nebo naopak může považovat šum za změnu. Je proto důležité, aby klasifikátor byl dostatečně citlivý pro detekci postupné změny konceptu, ale zároveň nebyl příliš ovlivněn šumem.

Pro testování této problematiky jsem použil dataset založený na problému pohybující se hyperroviny. Dataset se skládá z 10 000 prvků, které reprezentují body v prostoru, každý prvek má 10 dimenzí a návěští třídy. Atributy dimenzí obsahují spojité hodnoty v rozmezí [0;1] a návěští je buď 0 nebo 1. Návěští je určeno sumou násobků dimenze a váhy dimenze, jeli pak suma menší než hraniční hodnota, pak je třída 0, jinak je 1. Rozdělení prvků do tříd se postupem času mění, v mnou testovaném datasetu se postupně mění váhy 8 dimenzí. Dataset navíc obsahuje 5% šumu.



Obrázek 6.6: Přesnost metod při pozvolné změně konceptu.

Na obrázku 6.6 jsou znázorněny výsledky měření. Měření bylo prováděno na bloku 250 vzorků a trénování na bloku 500 vzorků. Počet klasifikátorů byl omezen na 10 aktivních klasifikátorů a 20 odložených klasifikátorů (pro DoS 21).

Na grafu lze pozorovat pozvolné zhoršování metody Basic, která je oproti nejlepší metodě CSHT horší zhruba o 5%. Při testování se ukázalo, že v tomto případě je lepší metoda DoS oproti DWAA, projevil se zde vliv rozmanitosti klasifikačních algoritmů. Metoda CSHT však i v tomto případě podává nejlepší výsledky.

6.5 Vyhodnocení výsledků

	Náhlá změna konceptu		Pozvolná změna konceptu		
Metoda	Přesnost[%]	Odchylka[%]	Přesnost[%]	Odchylka[%]	rychlost[vz./s]
Basic	84,37	4,12	83,46	4	1163
CSHT	86,88	2,46	87,94	2,51	457
DoS	85,52	3,46	85,68	2,78	243
DWAA	85,8	2,76	85	2,84	1095

Tabulka 6.1: Přehled naměřených hodnot

Z naměřených výsledků můžu vyzdvihnout tyto poznatky. Metoda CSHT dosáhla celkově nejlepších výsledků a je jí vhodné použít jak při náhlých, tak i při pozvolných změnách

konceptu. Metoda DWAA překvapila při náhlých změnách konceptu, kde se dokázala adaptovat lépe než metoda DoS, ale v některých případech i skoro stejně dobře jako metoda CSHT. Je ji proto vhodné použít v případech, kdy vyžadujeme adekvátní přesnost a zároveň vysokou rychlost klasifikace. Metodu DoS lze doporučit na stabilnější koncepty dat, kde se může projevit souhra většího množství klasifikačních algoritmů.

Dále jsem zjistil, že test hypotézy dokáže efektivně vybírat klasifikátory se shodným konceptem dat. Vhodně navržená odměňovací strategie podává kvalitní výsledky a pracuje velice rychle. Rozmanitost klasifikačních algoritmů, poskytuje výhody u částí se statickým konceptem dat, který v reálném prostředí zabírá většinu času.

Navíc toto testování posloužilo i k ověření funkčnosti naimplementovaných algoritmů.

6.6 Konfigurace parametrů

Pro metody používající soubory klasifikátorů je důležité vhodné nastavení. Jedná se zejména o velikost použitého plovoucího okna a o optimální počet klasifikátorů. Tyto hodnoty nelze správně určit bez znalosti aplikační domény, ve které budou metody použity, ale uvedu zde obecné principy řídící tuto problematiku.

6.6.1 Velikost plovoucího okna

Velikost plovoucího okna je důležitý parametr určující, jak často a jakým množstvím dat bude klasifikační systém aktualizován. Budeme-li mít plovoucí okno malé, bude jednodušší ho naplnit trénovacími daty a bude pro nás i jednodušší udržet klasifikátory aktuální. Nevýhodou malých plovoucích oken je však množství trénovacích dat, pokud bude množství příliš malé, mohou být klasifikátory trénovány na nedostatečném počtu vzorků a tím budou náchylné k šumu. Druhá varianta používající velká okna trénuje klasifikátory na kvalitní množině dat, avšak systém může zbytečně špatně klasifikovat data, u kterých došlo ke změně konceptu.

Pro ilustraci prezentuji na obrázku 6.7 naměřené hodnoty na třech různých velikostech plovoucího okna.

Jak jde vidět, metoda s oknem o velikosti 100 vzorků se přizpůsobuje změnám konceptu rychleji, dosahuje však horších výsledků ve stabilních částech. Toto chování je způsobeno vyšším poměrem šumu v malém množství trénovacích dat.

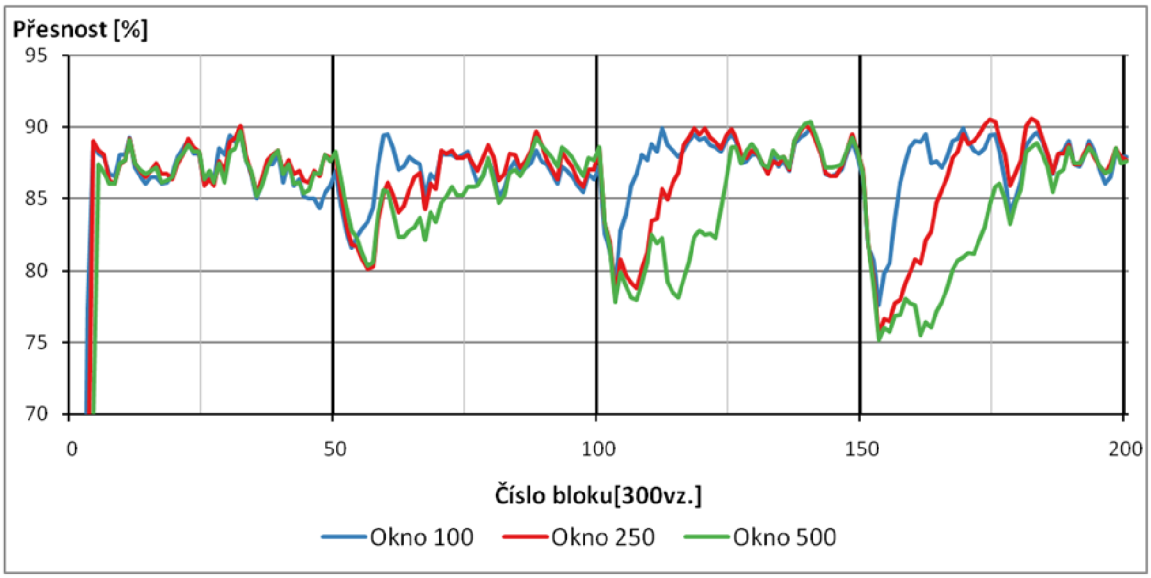
K odbornému rozhodnutí, jakou zvolit velikost plovoucího okna je potřeba znát jaká data budeme zpracovávat. Rozhodující je také, jak rychle a v jakém množstvím jsme schopní dodávat nová trénovací data a k jakým změnám konceptu může pravděpodobně dojít.

6.6.2 Počet klasifikátorů

Počet klasifikátorů je většinou vhodné nastavit na hodnotu odpovídající dostupným výpočetním prostředkům, ale určitě je nutné nenavyšovat počet klasifikátorů nad mez, kterou jsme schopni efektivně provozovat. Na druhou stranu však příliš velký počet klasifikátorů může některým metodám způsobit potíže. Z předešlých měření vyplývá, že metoda CSHT efektivně vybírá vhodné klasifikátory, ale např. metoda DoS může mít obtíže při klasifikaci, pokud jí bude dlouho trvat, než nové klasifikátory přehlasují velké množství zastaralých klasifikátorů.

Odložené klasifikátory jsou rozdílné. Reprezentují historické znalosti, které se mohou v budoucnu opakovat, zároveň nebývají překážkou aktuálně používaných klasifikátorů.

Je proto dobré jich mít vyšší počty, pokud to příliš nezpomalí zpracování aktualizace klasifikačního systému.



Obrázek 6.7: Přesnost klasifikátorů dle velikosti plovoucího okna.

6.7 Modifikace algoritmů

Po měření a experimentování s implementovanými algoritmy jsem získal představu, jak vytvořit efektivní metodu pro klasifikaci datových proudů za pomoci souboru klasifikátorů.

Novou metodu jsem navrhl, aby skloubila výhody předešlých algoritmů. Jako základ jsem použil metodu CSHT, tím jsem byl schopen vybírat klasifikátory se shodným konceptem jako mají aktuální data. Navíc již obsahovala systém jak zacházet s odloženými klasifikátory. Metodu CSHT jsem dále upravil, aby pracovala s více druhy klasifikačních algoritmů jako metoda DoS, také jsem při určování vah klasifikátorů použil odměňovací strategii metody DWAA.

Zmíněná metoda implementovala všechny výhody předešlých metod, navíc jsem experimentoval i s metodami, které se skládaly jen z kombinací výhod.

Naměřené výsledky však nebyly povzbudivé. Metoda CSHT byla efektivní a dosahovala výborných výsledků. Přidáním více klasifikačních algoritmů se metoda dle očekávání zpomalila, nedošlo však k prokazatelnému zlepšení přesnosti. Modifikací, aby metoda CSHT používala odměňovací strategii, nedošlo k negativním vlivům, avšak po výběru klasifikátorů metodou CSHT je plně dostačující jednodušší funkce pro výpočet vah klasifikátorů.

Experimenty se ukázalo, že prokazatelně vylepšit metodu CSHT bez navázání negativních vlivů pomocí zmíněných metod nelze. Lze jen doporučit použití stávajících metod na oblasti, ve kterých jsou silné.

Možnou efektivní modifikací může být rozšíření CSHT a DWAA o další klasifikační algoritmy, ty ale musí být rychlejší než SVM a C4.5 implementované v knihovně Accord. Dále by metoda DoS mohla těžit z odměňovací strategie metody DWAA. Na závěr je nutné podotknout, že tyto úpravy se mohu ukázat zbytečné, jelikož metoda CSHT dosáhla ve většině případů lepších výsledků než obě dvě zbývající metody.

Kapitola 7

Návrh na další postup

V současné době je implementován klasifikační systém, který obsahuje čtyři metody. Systém je implementován v samostatné verzi a ve verzi pro napojení na Malware analysis system.

Vhodným rozšířením klasifikačního systému, by mohlo být ošetření zbývající problematiky klasifikace v proudu dat. Implementované metody jsou vysoce závislé na trénovacích datech, proto by bylo vhodné implementovat mechanismus určující, která klasifikovaná data je nejvýhodnější expertně označit a použít jako trénovací data.

Dalším vhodným rozšířením je implementace metod zabývajících se problémem *concept evolution*, kdy se v datovém proudu mohou objevit dříve neznáme třídy. Pokud by systém nebyl na tuto variantu připraven, pak by špatně klasifikoval všechny výskyty, do doby než se nová třída objeví v trénovacích datech. Zmíněná dvě vylepšení by v mnohém zefektivnila trénování klasifikačního systému a snížila náklady na tvorbu trénovacích dat.

Poslední vhodnou oblastí, kde rozšířit implementované metody, je jejich práce s odloženými klasifikátory. Při hlubším studiu této problematiky, by se daly metody optimalizovat tak, aby si pamatovaly užitečné historické znalosti co nejdéle a spravovaly prostor odložených klasifikátorů efektivněji. Nasnadě je také vlastní implementace a optimalizace dalších základních klasifikátorů, které by byly dostatečně rychlé, aby výrazně nezpomalovaly klasifikační systém.

Na závěr bych ještě dodal, že množství kvalitních testovacích dat v oblasti datových proudů je značně omezené. Projekt, který by se zabýval získáním reálných a časem se vyvíjejících dat, pro testování dolování znalostí z datových proudů, by byl určitě přivítán všemi, kdo se touto problematikou zabývají.

Kapitola 8

Závěr

Cílem této diplomové práce bylo implementovat metody klasifikace v datových proudech pomocí souboru klasifikátorů, následně ověřit jejich funkčnost a navzájem je porovnat.

Svou práci jsem započal studiem problematiky získávání znalostí z dat a z oblasti datových proudů. Vědomosti, které jsem získal, jsou uvedeny v prvních kapitolách této práce. Jsou zde popsány základy dolování z dat a problematika prostředí datových proudů, se zaměřením na klasifikaci.

Následně jsem vybral tři slibné metody klasifikace v proudu dat, využívající soubor klasifikátorů. Metody CSHT, DWAA a DoS byly záměrně vybrány tak, aby každá představovala jiný koncept, jak efektivně klasifikovat data v datových proudech a jak se vyrovnat se změnou konceptu dat.

V praktické části jsem se zaměřil na implementaci vybraných metod a jejich porovnání. Pro implementaci metod jsem navrhl a vytvořil klasifikační systém, do kterého jsem následně metody zasadil. U všech metod byla ověřena jejich funkčnost. Systém s metodami byl integrován do analytického systému Malware analysis system, který je vyvíjen na naší fakultě.

Implementované metody jsem intenzivně testoval a měřil, abych je mohl objektivně porovnat. Metody byly testovány, jak na datech s náhlou změnou konceptu, tak i s pozvolnou. Výstupy těchto měření jsem analyzoval a došel k závěru, že nejlepší metodou je efektivní metoda CSHT. Relativně jednoduchá a rychlá metoda DWAA dosáhla překvapivých výsledků. Při náhlých změnách konceptu dat se vyrovnala metodě DoS a někdy dosahovala i přesnosti metody CSHT. Poslední metoda DoS měla nejlepší výsledky pouze ve stabilních částech konceptu, díky použití více klasifikačních algoritmů.

Na závěr jsem popsal své experimenty s implementovanými metodami a uvedl návrh nové klasifikační metody, která měla za cíl vylepšit metodu CSHT. Bohužel tohoto cíle se mi nepodařilo dosáhnout.

Výsledky práce jsem prezentoval na konferenci Student EEICT 2013 [5], kde jsem byl oceněn třetím místem v oblasti informačních systémů. Závěrem bych řekl, že diplomová práce splnila všechny body zadání a doufám, že pomůže bádání v oblasti získávání znalostí z dat.

Literatura

[1] Chen, H.; Ma, S.; Jiang, K.: Detecting and adapting to drifting concepts. In *Proceedings of the 9th International Conference on Fuzzy Systems and Knowledge Discovery (FSKD)*, 2012, ISBN 978-1-4673-0025-4, s. 775–779.

[2] Han, J.; Kamber, M.: *Data Mining concepts and techniques.* Morgan Kaufmann Publishers, druhé vydání, 2006, ISBN 1-55860-901-6.

[3] Hang, Y.; Fong, S.: An experimental comparison of decision trees in traditional data mining and data stream mining. In *Advanced Information Management and Service (IMS), 6th International Conference*, 2010, ISBN 978-1-4244-8599-4, s. 442–447.

[4] Hanák, J.: *C# 3.0: programování na platformě .NET 3.5.* Zoner Press, první vydání, 2009, ISBN 978-80-7413-046-5.

[5] Jarosch, M.: Classification in Data Streams Using Ensemble Methods. In *Proceedings of the 19th Conference STUDENT EEICT*, 2013, ISBN 978-80-214-4694-6, s. 210–212.

[6] Kupčík, J.; Hruška, T.: Towards Online Data Mining System for Enterprises. In *Proceedings of the 7th International Conference on Evaluation of Novel Approaches to Software Engineering (ENASE 2012)*, 2012, s. 187–192.

[7] Mathur, A.; Foody, G.: Multiclass and Binary SVM Classification: Implications for Training and Classification Users. *Geoscience and Remote Sensing Letters, IEEE*, ročník 5, č. 2, 2008: s. 241–245, ISSN 1545-598X.

[8] Microsoft: LINQ (Language-Integrated Query) [online]. `http://msdn.microsoft.com/en-us/library/vstudio/bb397926.aspx`, [cit. 2013-05-21].

[9] Microsoft: Task Parallel Library (TPL) [online]. `http://msdn.microsoft.com/en-us/library/dd460717.aspx`, [cit. 2013-05-21].

[10] Nguyen, H.; Cooper, E.; Kamei, K.: Online learning from imbalanced data streams. In *Soft Computing and Pattern Recognition (SoCPaR) International Conference*, 2011, ISBN 978-1-4577-1195-4, s. 347–352.

[11] Polikar, R.: Ensemble based systems in decision making. *Circuits and Systems Magazine, IEEE*, ročník 6, č. 3, 2006: s. 21–45, ISSN 1531-636X.

[12] Wu, D.; Wang, K.; He, T.; aj.: A Dynamic Weighted Ensemble to Cope with Concept Drifting Classification. In *Proceedings of the 9th International Conference for Young Computer Scientists (ICYCS)*, 2008, ISBN 978-0-7695-3398-8, s. 1854–1859.

[13] Yan, J.; Yun, X.; Zhang, P.; aj.: A New Weighted Ensemble Model for Detecting DoS Attack Streams. In *Proceedings of the 3th IEEE/WIC/ACM International Conference on Web Intelligence and Intelligent Agent Technology (WI-IAT)*, 2010, ISBN 978-1-4244-8482-9, s. 227–230.

[14] Zendulka, J.; kolektiv autorů: *Získávání znalostí z databází: Studijní opora*. 2009.

Příloha A

Obsah CD

Adresářová struktura přiloženého CD je následující:

- DP - Text diplomové práce.
- Ensemble classification - Složka obsahující varianty projektů se zdrojovými kódy.
- Data - Datasety použité pro testování.