FAKULTA INFORMAČNÍCH TECHNOLOGIÍ ČVUT V PRAZE

# Zadání diplomové práce

| | |
|---|---|
| **Název:** | Framework pro automatické zlepšování klasifikace síťového provozu |
| **Student:** | Bc. Jaroslav Pešek |
| **Vedoucí:** | Ing. Dominik Soukup |
| **Studijní program:** | Informatika |
| **Obor / specializace:** | Počítačová bezpečnost |
| **Katedra:** | Katedra informační bezpečnosti |
| **Platnost zadání:** | do konce letního semestru 2022/2023 |

## Pokyny pro vypracování

Cílem závěrečné práce je vytvoření softwarového prototypu frameworku pro automatické vyhodnocování a vylepšování výsledků algoritmů strojového učení pro klasifikaci síťového provozu a detekci bezpečnostních hrozeb prostřednictvím aktualizací datových sad. Seznamte se s problematikou klasifikace síťového provozu pomocí IP Flows, které lze využít pro šifrovaný i nešifrovaný provoz. Dále prozkoumejte problematiku vyhodnocování přesnosti klasifikátorů v laboratorních podmínkách a při reálném nasazení a automatického vylepšování datových sad (např. pomocí Active Learning). Vytvořte návrh prototypu frameworku, který bude schopen automaticky vyhodnocovat algoritmů strojového učení a vylepšovat jej prostřednictvím aktualizace datové sady. Zaměřte se na univerzálnost a rozšiřitelnost celého řešení. Implementovaný framework otestujte na vybraných datových sadách (datové sady budou poskytnuty vedoucím práce).

---

*Elektronicky schválil/a prof. Ing. Róbert Lórencz, CSc. dne 16. prosince 2021 v Praze.*

Diplomová práce

# FRAMEWORK PRO AUTOMATICKÉ ZLEPŠOVÁNÍ KLASIFIKACE SÍŤOVÉHO PROVOZU

**Bc. Jaroslav Pešek**

Fakulta informačních technologií
Katedra informační bezpečnosti
Vedoucí: Ing. Dominik Soukup
5. května 2022

# Obsah

# Seznam obrázků

# Seznam tabulek

*Děkuji svému vedoucímu práce Ing. Dominiku Soukupovi za odborné vedení a věnovaný čas. Stejný dík patří Laboratoři monitorování síťového provozu za zázemí, připomínky a náměty, jmenovitě pak především Ing. Tomáši Čejkovi, Ph.D. a Ing. Karlu Hynkovi. Dále děkuji sdružení CESNET, z.s.p.o. za poskytnutí technických prostředků, bez kterých by tato práce nevznikla.*

# Prohlášení

Prohlašuji, že jsem předloženou práci vypracoval samostatně a že jsem uvedl veškeré použité informační zdroje v souladu s Metodickým pokynem o dodržování etických principů při přípravě vysokoškolských závěrečných prací.

Beru na vědomí, že se na moji práci vztahují práva a povinnosti vyplývající ze zákona č. 121/2000 Sb., autorského zákona, ve znění pozdějších předpisů, zejména skutečnost, že České vysoké učení technické v Praze má právo na uzavření licenční smlouvy o užití této práce jako školního díla podle § 60 odst. 1 citovaného zákona.

V Praze dne 5. května 2022 .....

# Abstrakt

Tato diplomová práce se zabývá problematikou klasifikace především šifrovaného síťového provozu pomocí algoritmů strojového učení. Strojové učení je podoblast umělé inteligence, která silně závisí na dostatečně obsáhlých a obecných datových sadách. Prvním cílem je analýza metod, které v průběhu času takovou klasifikaci nejen vylepšují, ale zároveň s tím iterativně tvoří aktuální datovou sadu. Druhým cílem je vytvořit prototyp softwarového frameworku, který je toho schopen a zároveň je schopen klasifikaci vyhodnocovat. V analytické části je čtenář seznámen s metodou aktivního učení a analyzuje a diskutuje state-of-the-art a vhodnost metod pro oblast síťového provozu. V návrhové části definujeme požadavky a navrhujeme architekturu řešení. Poslední část práce je věnována experimentům. Výstupem práce je prototyp softwarového frameworku a vyhodnocení jednotlivých metod aktivního učení pro oblast síťového provozu.



**Klíčová slova** strojové učení, aktivní učení, klasifikace síťového provozu, vytváření datových sad



# Abstract

This diploma thesis deals with the problem of classification of primarily encrypted network traffic by applying machine learning algorithms. Machine learning is a subfield of artificial intelligence which relies heavily on sufficiently large and general datasets. The first goal is to analyze methods that not only improve such classification over time, but also iteratively build the updated dataset. The second goal is to create a prototype of a software framework capable of doing so, while also being able to evaluate the classification. In the analysis part, the reader is introduced to the active learning method and analyzes and discusses the state-of-the-art and relevance of the methods to the network traffic domain. The design part defines the requirements and designs the solution architecture. The final part of the thesis is focused on experiments. The output of the work is a prototype of the software framework and an evaluation of various active learning methods for the network traffic domain.



**Keywords** machine learning, active learning, network traffic classification, creation of datasets

# Úvod

Šifrovaný provoz na síťové infrastruktuře se již dávno stal naprostým standardem. To s sebou nese evidentní výhody pro uživatele, stejně tak ale tato skutečnost slouží škodlivým aktivitám. Do samotné komunikace nahlížet nelze, máme pouze směr, velikost a čas. Analogie k tomuto problému ve fyzickém světě je nasnadě; máme k dispozici obálku, kterou nelze otevřít. Známe odesílatele, adresáta, rozměry obálky, její hmotnost i čas odeslání. Je obsah obálky nějak škodlivý? Na první pohled to nelze říct. Naštěstí se nenacházíme ve fyzickém světě a měřitelné vlastnosti síťových paketů se ukazují být dobrou metrikou k rozpoznání (samozřejmě s jistou pravděpodobností), co zhruba je obsahem a především, zda je hoden toho, aby se rozsvítila bezpečnostní kontrolka.

Existuje značné množství prací a v praxi existuje řada řešení, jak se s touto problematikou vypořádat. Některé služby a protokoly jsou rozpoznatelné z šifrované komunikace plně deterministickým způsobem. Takový způsob je vázán na daný problém a velmi často je značně náročný.

Strojové učení je rozsáhlá skupina metod, jak zobecňovat pravidla a vzory z nějaké množiny dat. K tomu, aby nějaká metoda strojového učení dobře klasifikovala síťový provoz, je potřeba mimo výběr konkrétního modelu a trénovacího algoritmu právě kvalitní datová sada. Ukazuje se však, že síťový provoz je velmi dynamická doména, kde se často mění a posouvá vzhled dat, tedy to, že model předvídá dobře v danou chvíli nemusí zaručovat, že bude dobře předvídat i za nějaký čas. Datová sada zastarává. Průběžné aktualizace datových sad tuto vlastnost mohou zvrátit.

V první kapitole zanalyzujeme problém klasifikace síťového provozu a definujeme si síťové toky. Popíšeme, jaké výhody má klasifikace provozu z hlediska bezpečnosti. Popíšeme metriky, které lze využívat. Ve druhé části představíme oblast aktivního učení, jako podmnožinu strojového učení a přehled základních heuristik.

Ve druhé kapitole navrhneme softwarový prototyp frameworku pro automatické vyhodnocování a vylepšování výsledků algoritmů strojového učení pomocí aktualizací datových sad. K tomu využijeme analýzu a rešerši z první kapitoly.

Třetí kapitola obsahuje samotnou implementaci takového prototypu. Budou vynechány banální části, které je možné nalézt na přiloženém médiu v technické dokumentaci, a naopak vyzdviženy ty kruciální nebo zajímavé.

Poslední kapitola je věnována experimentálnímu vyhodnocení. Budou zvoleny problematiky blízké bezpečnosti v počítačových sítích a pokusíme se jejich prostřednictvím ukázat, že navržené řešení je funkční.

V příloze potom lze nalézt návod, jak sestavit nějaký program z navrženého frameworku.

# Kapitola 1
# Analýza a rešerše

S nástupem strojového učení do takřka všech myslitelných oborů silně vzrostla potřeba různorodých a kvalitních datových sad. Datovou sadou se obvykle rozumí nějaká statická množina $n$ vzorků s $k$ příznaky (popřípadě často budeme hovořit o vektoru příznaků z *feature vector*). V proměnlivé oblasti, jako je monitorování síťového provozu, však statičnost přináší tu nevýhodu, že data velmi rychle zastarají. Nejdříve bude představen pojem síťového toku, což je formát, který abstrahuje a agreguje informace z paketů náležících do jedné komunikace. Pak bude v krátkosti popsána problematika klasifikace v oblasti síťového provozu pomocí algoritmů strojového učení a metody měření jejich přesnosti. V závěru kapitoly bude představena metoda aktivního učení, která zajistí postupné zlepšování klasifikátorů a automatické vytváření aktuálních datových sad.

## 1.1 Síťový tok

Jeden z ústředních pojmů této práce je síťový tok. Ten je popisným prostředkem síťového spojení [1] a obsahuje informace agregované z paketů náležících do tohoto spojení. Existuje několik standardizovaných verzí, ale obsahují minimálně [1]:

**adresy** zdroje a cíle,

**porty** zdroje a cíle,

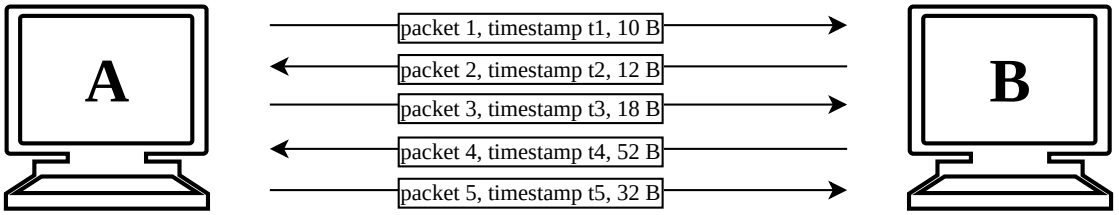
**časová razítka** začátku a konce spojení,

**počet paketů** ze zdroje do cíle a opačně,

**velikost** paketů.

Mimo to může obsahovat další informace. Vizualizace síťového toku je na obrázku 1.1. Výhodou síťových toků je nižší velikost oproti samotným paketům, ze kterých se skládají, z čehož vyplývá i vyšší rychlost zpracování, ale zároveň dostatečná popisná síla pro většinu užití, zvlášť pak v oblasti analýzy šifrovaného provozu.

Prvním standardem síťových toků byl `NetFlow` společnosti Cisco. Pak `IPFIX` definovaný organizací Internet Engineering Task Force (IETF). V práci akcentovaný framework NEMEA používá protokol `UniRec` [2], který je vizualizován na obrázku 1.2. `UniRec` je binární datový formát skládající se ze seznamu jednotlivých záznamů. Každý záznam se skládá z jednotlivých pojmenovaných polí, které jsou dány nějakým typem (jejich seznam je zanesen v tabulce 1.1). Jaká pole se v jednom záznamu objevují, je definováno pomocí šablony. Díky tomu je formát `UniRec` velmi




| IP zdroj | IP cíl | Začátek spojení | Konec spojení | Počet paketů zdroj -> cíl | Počet paketů cíl -> zdroj | Velikost paketů zdroj -> cíl | Velikost paketů cíl -> zdroj |
|---|---|---|---|---|---|---|---|
| 192.168.1.1 | 192.168.1.2 | t1 | t5 | 3 | 2 | 60 bajtů | 64 bajtů |

**Obrázek 1.1** Vizualizace síťového toku. Přístroj **A** je zdroj, protože komunikaci začal, a přístroj **B** je cíl. Lze vidět, že komunikace se skládala z 5 paketů a trvala celkem $(t5 - t1)$ jednotek času. Volitelně lze do toku přidat informace o jednotlivých paketech a podobně.

flexibilní. `UniRec` je podobný struktuře v jazyce `C`, takže oproti předchozím formátům umožňuje přímý přístup k jednotlivým položkám bez nutnosti syntaktické analýzy; je tedy zásadně efektivnější.

## 1.2 Klasifikace síťového provozu

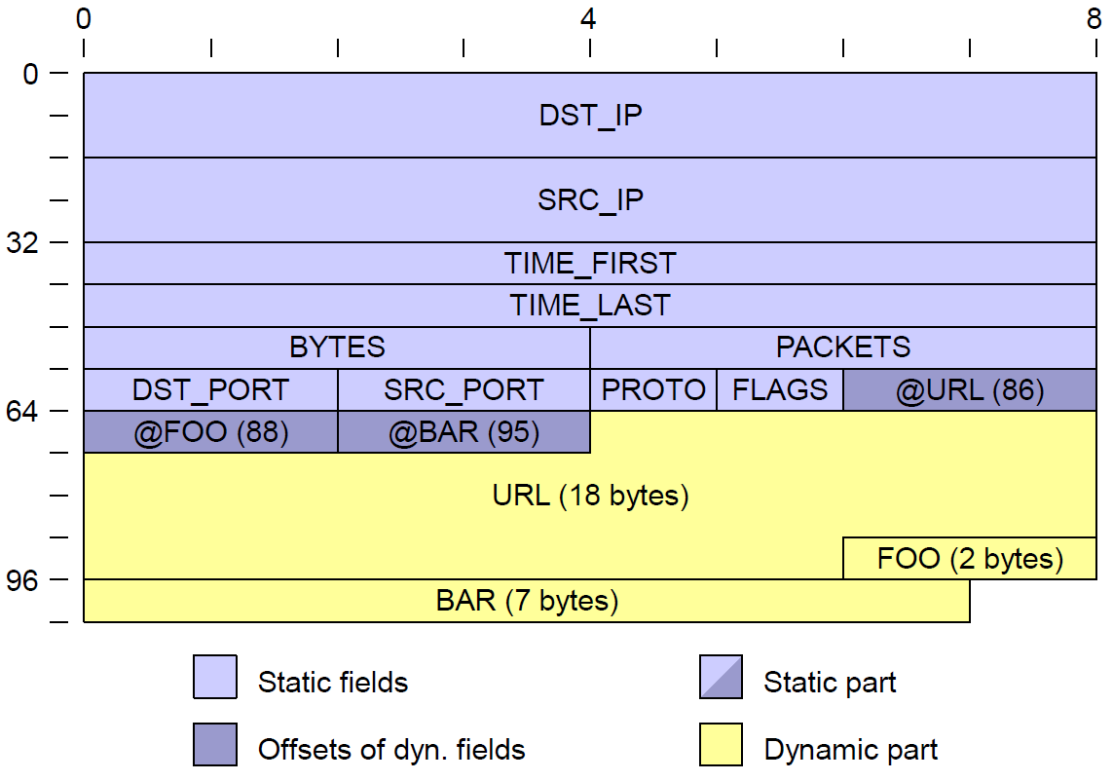
Klasifikace síťového provozu je jedna z hlavních problematik monitorování a analýzy síťového provozu. Dává větší vhled do provozu odehrávající se v nějaké síti a přináší řadu výhod od Quality of Service (QoS) po odhalování škodlivého softwaru, či jiného škodlivého provozu. Techniky klasifikace lze v zásadě kategorizovat do 3 skupin podle pozorovaných znaků, založené na [4]:

**portu** , využívajícího předpokladu standardního využití portů,

**obsahu** , kde se pozoruje samotný obsah paketů a

**síťových tocích** , což bude jediný způsob používaný v této práci.

Pravděpodobně nejjednodušší a nejnepřesnější je přístup první, který může být lehce oklamán a zneužit, neboť využívání portů je pouze standardizováno a navíc většina portů dokonce ani standardní službu přiřazenou nemá; a naopak, existují služby, které nevyužívají vždy stejný port. Druhý přístup může být již přesnější a méně závislý, ale může trpět omezeními vyplývajícími z šifrování komunikace. Zároveň tento přístup značně narušuje soukromí a bezpečnost uživatelů sítě. Přístup založený na síťových tocích je bezpečný a není závislý na šifrování ani na zvyklostech. Navíc povaha síťových toků přirozeně vybízí ke zpracování pomocí metod strojového učení.




**Obrázek 1.2** Příklad záznamu ve formátu `UniRec` reprezentující síťový tok. [3].

## 1.3 Problémy strojového učení v doméně síťového provozu

V laboratorních podmínkách můžeme problém klasifikace síťového provozu pohodlně redukovat na problém učení s učitelem (*supervised learning*), jelikož je obvykle v silách laboratoře opatřit jednotlivé toky disjunktivními třídami (dále je tento proces referován jako **anotace** a budeme o něm předpokládát, že je režijně náročný buď na výpočetní nebo na lidské zdroje) a následně vytrénovat prediktivní model nějakým algoritmem strojového učení a otestovat jej. Tento přístup přináší velmi dobré výsledky, například [5]. Nicméně ukazuje se, že ne vždy je takový výsledek dobře uplatnitelný v reálném provozu nebo v jiný časový úsek. Může to být způsobeno jednak rozdílem mezi uměle generovanou datovou sadou a jednak tím, že povaha dat se rychle mění (*data drift, concept drift*, [6]). Tento fenomén je demonstrován pomocí matic záměn 1.2 a 1.3, kdy první experiment je v souladu s publikovaným výsledkem v [5], nicméně při otestování takového modelu na reálně odchycených datech (více o této datové sadě v kapitole 4) takový model selhává. V uvedených tabulkách je uvedeno takzvané $F_1$ skóre, které bude s dalšími metodami vyhodnocení uvedeno v následující podkapitole (1.4).

## 1.4 Vyhodnocení prediktivního modelu

K hodnocení výkonu prediktivních modelů se používají rozličné metriky. Práce se omezí na problém diskrétní klasifikace *(multi-class)*. Vzorek se zařazuje obecně do $n$ tříd, kde $n \geqq 2$, tyto třídy s nepřekrývají, tedy vzorek patří právě do jedné třídy; zcela se zanedbává multiaspektová

**Tabulka 1.1** Přehled možných typů využitelných ve formátu `UniRec`

| Jméno | Velikost [bajty] | Popis |
|---|---|---|
| int8 | 1 | 8 bitový singed integer |
| int16 | 2 | 16 bitový singed integer |
| int32 | 4 | 32 bitový singed integer |
| int64 | 8 | 64 bitový singed integer |
| uint8 | 1 | 8 bitový unsigned integer |
| uint16 | 2 | 16 bitový unsigned integer |
| uint32 | 4 | 32 bitový unsigned integer |
| uint64 | 8 | 64 bitový unsigned integer |
| char | 1 | ASCI znak |
| float | 4 | Jednoduchá přesnost (IEEE 754) |
| double | 8 | Dvojitá přesnost (IEEE 754) |
| ipaddr | 16 | Typ pro IPv4/IPv6 adresu |
| macaddr | 6 | Typ pro MAC addresu |
| time | 8 | Typ pro přesné časové razítko |
| string | - | Pole proměnlivé délky pro textový řetězec |
| bytes | - | Pole bytů proměnlivé délky |

**Tabulka 1.2** Matice záměn pro replikovaný experiment z [5], použita stejná data a stejný model založený na algoritmu AdaBoost. Výsledek našeho experimentu je srovnatelný s tím publikovaným, s průměrným $F_1$ skóre 1.0.

| | | predikce | |
|---|---|---|---|
| | | HTTPS | DoH |
| skutečnost | HTTPS | 140860 | 10 |
| | DoH | 61 | 10170 |

**Tabulka 1.3** Matice záměn pro experiment, kdy je použit prediktivní model z experimentu z tabulky 1.2 podle [5], ale využit k predikci dat odchycených v síti CESNET. Průměrné $F_1$ skóre je zde 0.33

| | | predikce | |
|---|---|---|---|
| | | HTTPS | DoH |
| skutečnost | HTTPS | 85430 | 782 |
| | DoH | 110041 | 3747 |

kategorizace *(multi-labelled)*.

Základní nástroj pro vyhodnocování klasifikace o $n$ vzorcích je matice záměn $M$, což je matice počtu predikovaných tříd $\widehat{Y}_i$ ve sloupcích proti skutečným třídám $Y_i$ v řádcích [7, 8]. Příklady takových matic jsou například v předcházejícím textu v tabulkách 1.2 a 1.3. Z toho plyne, že na souřadnicích $(i, i)$ se nachází počet správně klasifikovaných tříd a na souřadnici $(i, j)$ se nachází počet vzorků klasifikovaných jako $j$tá třída, ale její skutečná třída je $i$tá. Pro tuto sekci si definujme $N$ jako celkový počet vzorků, $c$ jako nějakou konkrétní třídu a $n$ jako počet tříd.

Z matice záměn je možno ihned odvodit první metriku a tou je `accuracy (acc)`, neboli odhad pravděpodobnosti $P(\widehat{Y} = Y)$.

$$\mathrm{acc}(M) = \frac{\sum_{i=1}^{n} M_{i,i}}{n} \tag{1.1}$$

Tato metrika není relativní vůči třídě a nemůže být použita pro datové sady, které mají značně nevyvážený počet vzorků [7], tedy v případě, že pro nějakou třídu platí $P(Y = c) \ll P(Y \neq c)$.

Další metriky *precision* (značeno prec) a *recall* (značeno rec) jsou již relativní [8], tedy hovoříme například o precision nějaké třídy ($\mathrm{prec}(M, c)$). V případě precision odhadujeme pravděpodobnost $P(Y = c|\widehat{Y} = c)$, v případě recall $P(\widehat{Y} = c|Y = c)$. Zaveďme si ještě zkratky $N_c$, což je počet vzorků skutečně náležící třídě $c$ (součet sloupce $c$) a $\widehat{N_c}$, počet vzorků predikovaných jako

třída $c$ (součet řádku $c$).

$$\mathrm{prec}(M, c) = \frac{M_{c,c}}{\widehat{N_c}} \tag{1.2}$$

$$\mathrm{rec}(M, c) = \frac{M_{c,c}}{N_c} \tag{1.3}$$

Harmonický průměr prec a rec potom označujeme jako $F_1$ skóre. To je užitečné pro nevybalancované datové sady [7], kde lze očekávat malou pravděpodobnost $P(Y = c)$.

$$F_1(M, c) = 2\frac{\mathrm{prec}(M, c)\,\mathrm{rec}(M, c)}{\mathrm{prec}(M, c) + \mathrm{rec}(M, c)} \tag{1.4}$$

Často od dané metriky očekáváme jednu hodnotu v intervalu $[0, 1]$, která má pokrýt všechny třídy. K tomu lze použít průměrování vybrané metriky přes všechny třídy; například vážený průměr, kdy vyšší váhu bude mít málo pravděpodobná třída atd. Průměrování je použito například v knihovně `scikit-learn` [9] pro jazyk `Python`.

Matthewsův korelační koeficient (značeno MCC) velmi dobře funguje pro nevybalancované datové sady [7]. Jeho výhoda je, že se nemusí průměrovat a dokáže velmi dobře vystihnout korelaci mezi $\widehat{Y}$ a $Y$ [9]. Jelikož se jedná o korelační koeficient, hodnoty mohou obecně být od $-1$ (antikorelace) do 1 (zcela přesný klasifikátor). Tuto hodnotu lze normalizovat na interval $[0, 1]$ [10], ale v této práci pro to není valný důvod.

$$\mathrm{MCC}(M) = \frac{n\sum_{i=1}^{n} M_{i,i} - \sum_{i=1}^{n} N_i\widehat{N_i}}{\sqrt{(n^2 - \sum_{i=1}^{n}\widehat{N_i^2})(n^2 - \sum_{i=1}^{n} N_i^2)}} \tag{1.5}$$

## 1.5 Aktivní učení

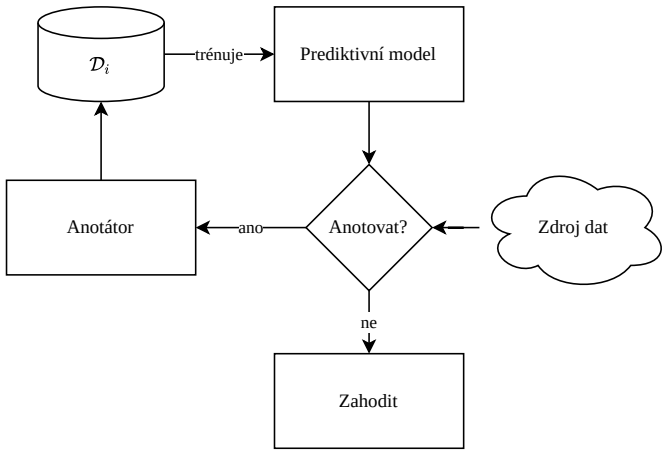
Standardní způsob (trénink → validace → test → nasazení) tvorby prediktivních modelů je vhodný pro statická data. Existují domény, jako je právě klasifikace síťového provozu, kde tento způsob naráží na několik výzev. Síťových toků je mnoho, jsou velmi lehce získatelné, ale jejich anotování je často drahé; je tedy potřeba vybrat toky, které mají pro model nejvyšší možnou hodnotu k dosažení rozumné míry generalizace jichž je zároveň co nejméně.

### 1.5.1 Základní smyčka

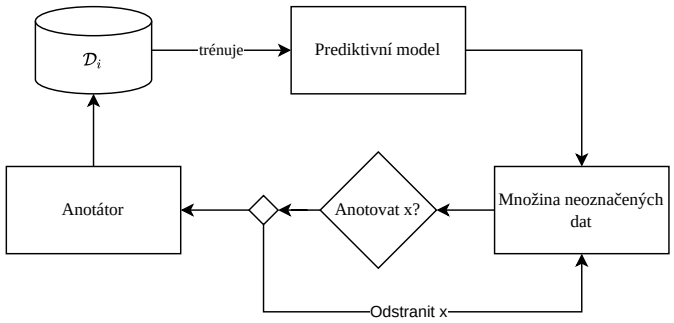
Existují dva základní přístupy k predikování neoznačených vzorků [11] lišící se způsobem vstupu dat a následným naložením s tím, co se rozhodneme neanotovat, a to buď proudové zpracování (obrázek 1.3, algoritmus 1), nebo hromadné zpracování (obrázek 1.4, algoritmus 2). V případě prvního přístupu získáváme stále nové toky ($\mathcal{U}_i$) v každé iteraci a zahazujeme je, jestliže se rozhodneme je neoznačovat. V případě druhého máme neznačená data od začátku k dispozici [11] jako množinu $\mathcal{U}$, tedy známe od začátku učení jejich distribuci, na rozdíl od proudového zpracovávání. Proudové zpracování se zdá být více flexibilním v případě nasazení v *online* provozu, kdy se očekává, že se model bude přizpůsobovat situaci. Hromadné zpracování se zdá být vhodnější v situaci, kde existuje velké množství neoznačených dat oproti označeným, tedy notací jako v algoritmu 2, $\mathcal{D}_0 \ll \mathcal{U}$, a máme v úmyslu vytvořit dostatečně vypovídající datovou sadu nebo dobrý model s co nejmenším počtem provedených anotací.

### 1.5.2 Vzorkovací strategie

V algoritmech 1 a 2 vystupuje vzorkovací strategie *(query strategy)* $\mathcal{Q}$, která určuje, které vzorky budou vybrány do trénovací sady $\mathcal{D}_i$. Tato strategie je fundamentální součást aktivního učení




**Obrázek 1.3** Proudové zpracování




**Obrázek 1.4** Hromadné zpracování

a existuje několik metod, které si v krátkosti představíme. Nejjednodušší strategie, která v experimentech zároveň slouží jako referenční bod, je **náhodné vzorkování**, kde výběr vzorků proběhne uniformně náhodně z množiny $\mathcal{U}$ nebo $\mathcal{U}_i$.

Požadavek na výběr vzorku k anotaci je co nejvyšší informační přínos, tedy s malým počtem vzorků chceme co nejvíce pokrýt prostor hypotéz [11] daného prediktivního modelu. Toho lze dosáhnout různými heuristikami, a to především využitím posteriorní pravděpodobnosti při klasifikaci vzorku nebo využitím míry podobnosti vzorku k aktuální trénovací sadě.

## Struktura vzorkovací strategie a používané funkce

Základní vzorkování často probíhá ve dvou krocích a to

1. Ohodnocením všech neoznačených vzorků (například v kontextu proudového aktivního učení množina $\mathcal{U}_i$) skórovací funkcí. Skórovací funkce využívaná v následujících heuristikách bude značena jako score($x$), což bude zobrazení mezi a vzorkem $x$ v rámci nějakého modelu a reálnými čísly.

2. Výběr vzorků na základě jejich skóre z prvního kroku:

   a. nastavením prahu (vyberou se ty vzorky se skórem menším než zadaný práh) [11]

   b. výběrem $n$ vzorků s nejmenším skóre (triviální případ je pak $n = 1$, kdy se vybere extrém)

   c. výběr málo navzájem podobných vzorků

**Algoritmus 1:** Proudové aktivní učení

$\mathcal{D}_0 \leftarrow$ iniciační množina trénovacích dat
$\mathcal{S} \leftarrow$ zdroj síťových toků
$\mathcal{Q} \leftarrow$ strategie výpočtu nejistoty
$i \leftarrow 0$

**Repeat**
  $\theta = \texttt{train}(\mathcal{D}_i)$
  $\mathcal{U} \leftarrow \texttt{ask}(S)$ //požádej zdroj o toky
  vyber $x^* \in \mathcal{U}$ nejvíce nejisté instance v modelu $\theta$ strategií $\mathcal{Q}$
  $y^* \leftarrow \texttt{anotate}(x^*)$
  $\mathcal{D}_{i+1} = \mathcal{D}_i \cup (x^*, y^*)$
  $i \leftarrow i + 1$

**Algoritmus 2:** Hromadné aktivní učení

$\mathcal{D}_0 \leftarrow$ iniciační množina trénovacích dat
$\mathcal{U} \leftarrow$ fond neanotovaných toků
$\mathcal{Q} \leftarrow$ strategie výpočtu nejistoty

**for** $i \in 0, 1, ...n$ **do**
  $\theta = \texttt{train}(\mathcal{D}_i)$
  vyber $x^* \in \mathcal{U}$ nejvíce nejisté instance v modelu $\theta$ strategií $\mathcal{Q}$
  $y^* \leftarrow \texttt{anotate}(x^*)$
  $\mathcal{D}_{i+1} \leftarrow \mathcal{D}_i \cup (x^*, y^*)$
  $\mathcal{U} \leftarrow \mathcal{U} \setminus x^*$
**end**

**d.** kombinací těchto přístupů (například maximálně $n$ málo podobných vzorků pod práh atd.)

Zaveďme nějakou funkci pro určování podobnosti, která se může využít například v bodě 2c. Označme množinu síťových toků (vzorků) jako $\mathcal{F}$. Podobnost *(similarity)* je nějaké zobrazení páru vzorků do reálných nezáporných čísel ($\text{sim} : \mathcal{F} \times \mathcal{F} \to \mathbb{R}^+$) a splňující vlastnosti pro $\forall f_1, f_2 \in \mathcal{F}$ definovány v 1.6, postupně non-negativitu, maximalitu a symetrii.

$$\text{sim}(f_1, f_2) \geq 0 \tag{1.6a}$$
$$\text{sim}(f_1, f_1) \geq \text{sim}(f_1, f_2) \tag{1.6b}$$
$$\text{sim}(f_1, f_2) = \text{sim}(f_2, f_1) \tag{1.6c}$$

Pro funkci sim se tedy dá použít nějaká vzdálenostní metrika $d$ upravená tak (rovnice 1.7, [12]), aby byly splněny vlastnosti výše, například euklidovská vzdálenost nebo manthattanská vzdálenost [13, 14].

$$\text{sim}(x, y) = \frac{1}{1 + d(x, y)} \tag{1.7}$$

## Vzorkování podle nejistoty *(Uncertainity Sampling)*

Tato strategie vzorkování preferuje takové vzorky, které mají nejslabší jistotu v predikci. Mějme nějaký model $\theta$, pak predikce $\widehat{y}$ pro vzorek $x$ je dán pravděpodobností $P(\widehat{y}|x)$, která je nejvyšší ze všech možných pravděpodobností predikce $x$.

Způsobů, jak takovou nejistotu měřit, je několik [6, 11] a to dle:

1. nejistoty (*least confident*, rovnice 1.8),
2. rozdílu (*margin*, rovnice 1.9); $\widehat{y}_i$ je $i$tá určená třída v sestupném pořadí,
3. entropie (rovnice 1.10), $n$ zde značí počet tříd.

$$\text{score}(x) = 1 - P_\theta(\widehat{y}|x) \tag{1.8}$$

$$\text{score}(x) = P_\theta(\widehat{y}_2|x) - P_\theta(\widehat{y}_1|x) \tag{1.9}$$

$$\text{score}(x) = -\sum_{i=1}^{n} P_\theta(\widehat{y}_i|x) \log P_\theta(\widehat{y}_i|x) \tag{1.10}$$

### Komise modelů *(Query-By-Committee)*

Systém se skládá ze dvou a více různých modelů (*ensanble*), které jsou trénovány na stejné množině $\mathcal{D}_i$. Tato strategie vytváří širší prostor hypotéz a různými přístupy dochází k měření neshody. Lze je využít dvěma základními způsoby a to buď:

1. zprůměrováním posteriorních pravděpodobností a dále s nimi pracovat jako v případě vzorkování podle nejistoty, nebo
2. využitím jednotlivých predikcí různých modelů, což rozebereme dále.

Základní strategie tohoto typu je založena na neshodě mezi modely *(query by disagreement)*, kdy vybíráme k anotaci právě ty vzorky, na kterých nepanuje jednohlasná shoda v komisi. Oproti výše uvedeným chybí možnost, jak smysluplně definovat funkci score($x$), jako v předchozím případě, a informace o přínosnosti jednotlivých vzorků je skryta.

Vylepšením takové strategie je zavedení entropie hlasů *(vote entropy)* a to buď měkké (*soft*, rovnice 1.11 ), nebo tvrdé (*hard*, rovnice 1.13).

Zavedeme $P_C(\widehat{y}_i|x) = \frac{1}{|C|}\sum_{j=1}^{|C|} P_{C_j}(\widehat{y}_i|x)$, kde $n$ je počet tříd, $C$ je množina komisí a $C_i$ je $i$tý model v komisi a vyjadřuje průměr pravděpodobností $\widehat{y}_i$ přes všechny modely v komisi.

$$\text{score}(x) = -\sum_{i=1}^{n} P_C(\widehat{y}_i|x) \log P_C(\widehat{y}_i|x) \tag{1.11}$$

Dále mějme funkci $\text{vote}_C(\widehat{y}_i, x)$ sčítající hlasy(rovnice 1.12).

$$\text{vote}_C(\widehat{y}_i, x) = \sum_{j=1}^{|C|} \begin{cases} 1, & \text{jestliže predikce modelu } C_j(x) = \widehat{y}_i \\ 0, & \text{jinak} \end{cases} \tag{1.12}$$

$$\text{score}(x) = -\sum_{i=0}^{n} \frac{\text{vote}_C(\widehat{y}_i, x)}{|C|} \log \frac{\text{vote}_C(\widehat{y}_i, x)}{|C|} \tag{1.13}$$

Poslední strategie založená na predikci jednotlivých modelů je strategie založená na Kullback-Leiblerově divergenční míře [11, 15] *(KL divergence, relativní entropie)*. Ta vyjadřuje míru rozdílnosti mezi dvěma pravděpodobnostními distribucemi – v tomto případě je to rozdílnost shody mezi jednotlivými členy komise od průměru celé komise.

$$\text{score}(x) = \frac{1}{|C|}\sum_{j=1}^{|C|}\sum_{i=1}^{n} P_{C_j}(\widehat{y}_i|x) \log \frac{P_{C_j}(\widehat{y}_i|x)}{P_C(\widehat{y}_i|x)} \tag{1.14}$$

## Informační hustota *(Information Density)*

Vylepšením předchozích heuristik je zahrnutí informační hodnoty neoznačeného vzorku v kontextu ostatních vzorků, neboť v dosavadních heuristikách jsme se na každý vzorek koukali izolovaně v tom smyslu, že jsme pouze sledovali posteriorní pravděpodobnost v rámci nějakého modelu, respektive množiny (komise) modelů. Předpokládáme [11], že eliminací daleko stojících vzorků *(outlies)* lze dosáhnout vyšší míry informovanosti o samotné struktuře neoznačených dat.

Ať $\mathcal{S}$ je libovolná funkce score definovaná výše, $\beta$ je hyperparametr regulující důležitost podobnosti, pak zahrnutí podobnosti do skóre lze učinit podle definice v rovnici 1.15 [11].

$$\text{score}(x) = \mathcal{S}(x) \left( \frac{1}{|\mathcal{U}_i|} \sum_{u \in \mathcal{U}_i} \text{sim}(x, u) \right)^{\beta} \tag{1.15}$$

## Konečný výběr podle skóre

Základní metody konečného výběru na základě hodnoty skóre byly popsány na začátku kapitoly. Diskutujme nyní bod 2c. Nějaká heuristika udělí skóre neoznačenému vzorku $x$ v rámci nějakého prediktivního modelu. Nejjednodušší způsob je vybrat $n$ vzorků s nejvyšším skóre, vylepšením je poté přidání nějakého prahu hodnoty skóre [11]. Potenciální problém s těmito způsoby je ten, že vzorky v tomto výběru mohou být velmi podobné, čímž ochuzujeme trénovací data pro další iteraci o potenciální informativnost, popřípadě je procedura anotování vykonána zbytečně.

Způsob, jak ještě vylepšit způsob vzorkování, je zavést váhu pro každý vzorek, kde se preferuje jednak hodnota $\text{score}(x)$ zavedená výše a jednak malá podobnost s již zařazenými vzorky v aktuální $\mathcal{D}_i$ [13]. Takový způsob je tedy velmi podobný zavedené heuristice informační hustoty. Říkejme tomu hodnocený výběr.

Finální pořadí vzorků se poté stanoví vyhodnocením formule podle rovnice 1.16 pro každý vzorek $x \in \mathcal{U}$. Parametr $\alpha$ slouží k vážení jednotlivých složek (tedy jednak skóre a jednak podobnosti). Dle [13] je vhodné nastavení $\alpha = \frac{|\mathcal{U}|}{|\mathcal{D}|+|\mathcal{U}|}$.

$$r(x) = \alpha(1 - \arg\max_{y \in \mathcal{D}_i} \text{sim}(x, y)) + (1 - \alpha)\,\text{score}(x) \tag{1.16}$$

Příklad základní kombinované strategie $\mathcal{Q}$, využívající tento krok spolu s omezením počtu je popsán v algoritmu 3.

## Zpětná vazba *(Reinforcement Active Learning (RAL))*

RAL (zpětnovazebné aktivní učení) [16, 17] je snaha o zajištění zpětné vazby.

Jedná se o strategii pro komisi modelů, která kombinuje strategii založenou na nejistotě a strategii náhodnou. Zpětná vazba zde funguje tak, že v případě, že se model při predikci, která byla podle nejistoty vybrána, splete, je systém zpětnou vazbou oceněn (neboť správně vybral vzorek, který je potřeba označit). V případě, že je označený vzorek k anotaci predikován správně, je vyslána negativní zpětná vazba. Zpětná vazba je zde také použita k vážení důležitosti jednotlivých členů komise. Schématické znázornění je zobrazeno na obrázku 1.5. Tato strategie s sebou přináší několik parametrů, které jsou v uvedených článcích blíže popsány a které obsahují i analýzu vlivu, jejž na funkcionalitu strategie mají; znovu je analyzovat v této práci by bylo mimo záběr této práce. Čtenáře tedy odkazujeme tam a zde předkládáme pouze výčet se stručným popisem s výchozí hodnotou:

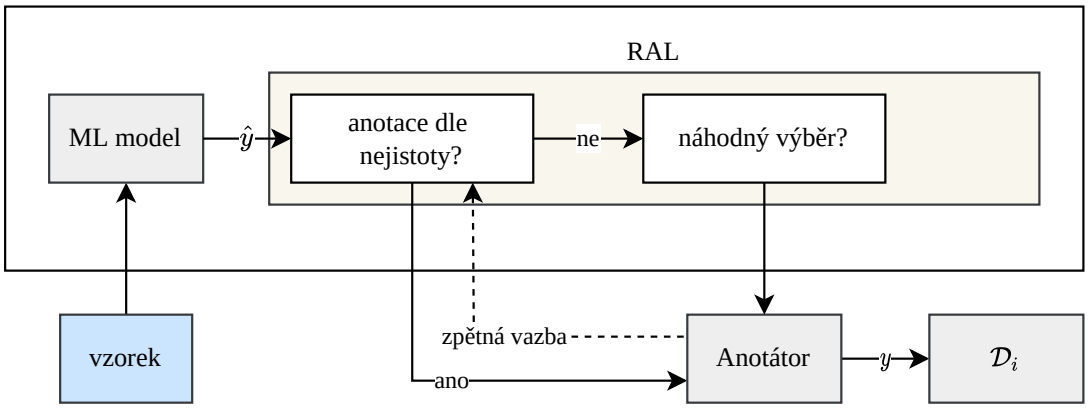
- počáteční práh nejistoty, ten je předmětem škálování v průběhu chodu (0.95),
- pravděpodobnost, že vzorek je vybrán náhodou (0.05),
- *learning rate*, tedy rychlost, jakou se mění práh nejistoty a důležitost členů v komisi (0.05),

**Algoritmus 3:** Obecná strategie $\mathcal{Q}$ pro výběr k anotaci založená na skórovací funkci a podobnosti

$\mathcal{D} \leftarrow$ aktuální kopie množiny trénovacích dat $\mathcal{D}_i$
$\mathcal{U} \leftarrow$ aktuální kopie množiny neanotovaných toků s před vypočteným score($x$) pro každý tok, lze vypustit ty vzorky, které mají dostatečně vysoké skóre
$\text{sim}(x) \leftarrow$ funkce definována jako $\arg\max_{y \in \mathcal{D}} \text{sim}(x, y)$
$n \leftarrow$ počet vzorků, kolik jich chceme anotovat
$\mathcal{R} \leftarrow \emptyset$ (výsledky)

**for** $i \in 1..n$ **do**
  $\mathcal{L} \leftarrow \emptyset$ (množina k ukládání mezivýsledků $(x, r(x))$)
  **for** $x \in \mathcal{U}$ **do**
    $\alpha = \frac{|\mathcal{U}|}{|\mathcal{D}|+|\mathcal{U}|}$
    $r(x) = \alpha(1 - \text{sim}(x)) + (1 - \alpha)\,\text{score}(x)$
    $\mathcal{L} \leftarrow \mathcal{L} \cup (x)$
  **end**
  $m \leftarrow \max(\mathcal{L})$ dle $r(x)$
  $\mathcal{U} = \mathcal{U} \setminus m$
  $\mathcal{D} = \mathcal{D} \cup m$
  $\mathcal{R} = \mathcal{R} \cup m$
**end**
vrať $\mathcal{R}$

- *budget*, množství vzorků, které je množné vybrat k anotaci; je definováno jako číslo z intervalu (0,1] a definuje maximální poměr anotovaného ku zahozenému (0.05),
- hodnota odměny a trestu (+1 a -1).




**Obrázek 1.5** Schéma strategie RAL.

### 1.5.3 Diskuze a state-of-the-art

Bylo zanalyzováno několik základních typů strategií, jak vzorkovat data určená k anotaci tak, abychom maximalizovali jejich informační přínos a tím minimalizovali počet vzorků, které je potřeba anotovat, čímž lze potenciálně šetřit zdroje, jako je například potřeba lidského experta nebo výpočetního výkonu. Analýza a popis strategií se omezil na jejich základní druhy (vycházející převážně z [11] a [13]) a neklade si za cíl podrobný a úplný přehled.

Mnoho strategií a algoritmů se zaměřuje na hromadnou verzi aktivního učení. Ta má ale několik zásadních nevýhod oproti proudovému zpracování v diskutované doméně:

1. je nutný sběr dat a následné zpracování, oproti proudovému přístupu chybí výhoda plynulého přizpůsobování dynamické změně běžné v počítačových sítích (tento fenomén se nazývá *concept drift* [6]),
2. je zde zřejmá nutnost skladovat velké množství dat a s tím spojená vyšší výpočetní režie.

Naopak proudové zpracování má zásadní výhody, komplementární s výše uvedenými:

1. je přirozeně použitelné jako *online* metoda strojového učení a
2. de facto imituje inkrementální učení i s modely tento způsob neumožňujícími (například rozhodovací strom nelze učit inkrementálně, je třeba jej učit vždy na celé trénovací množině).

Z toho důvodu byl výběr strategií zaměřen především na proudové aktivní učení a bude na něj zaměřen i zbytek práce. Existují však zajímavé koncepty zamýšlené pro hromadné zpracování. Některé algoritmy [18, 19, 20, 21] jsou nevhodné kvůli tomu, že potřebují mít v každé iteraci k dispozici stejná neoznačená data $\mathcal{U}$ ke své zamýšlené funkčnosti, nicméně v proudovém zpracování se množina $\mathcal{U}_i$ mění. Problém s výkonem pak evidentně přináší algoritmy, u nichž je potřeba nějaká náročnější operace nad každým vzorkem z proudu, například ty strategie, které vybírají vzorky, jež nejvíce nějakým způsobem mění prediktivní model (rodina algoritmů *Maximizing Expected Model Change* [22] nebo *Expected Error Reduction* [11] nebo *Variance Reduction* [11]). Učení se aktivnímu učení (*Learning active learning*) [23, 24] je algoritmus, který formuluje výběr vhodné strategie jako regresní problém nad parametry klasifikátoru a příznaky neoznačených dat. Problém této metody je kombinací předchozích. Tyto algoritmy mohou být však velmi vhodné, pokud je potřeba lidský expert při *offline* provozu, a výpočetní náročnost není nutně problém. Takový scénář si lze představit například při tvorbě množiny $\mathcal{D}_0$.

Existují heuristiky rozvíjející ty základní vhodným rozdělením problému na menší pomocí vhodné metody, například [25], popřípadě přinášejí metody zpětné vazby [26, 27], a některé mají i aplikace v problematice monitorování síťového provozu [17]. Zpětnovazební metody například mění práh hodnoty výstupu funkce skóre pro přijetí nebo mění váhu jednotlivých modelů v systému využívajícím komisi. Velký úspěch přineslo využití SVM modelu v rámci aktivního učení [28] dokonce aplikované v klasifikaci P2P provozu [29].

Představené metody založené na podobnosti (informační hustota a pak vážení skóre podle podobnosti) nejsou v současném středním proudu příliš akcentovány, přitom může jít o zajímavou metodu uplatnitelnou právě v proudovém zpracování, protože může být relativně rychlá (složitost je zde sice polynomiální, ale výpočet podobnosti může být značně rychlejší než opakované přetrénovávání jako ve výše uvedených metodách). Jediné experimentální vyhodnocení je v článku zavádějící sestavování žebříčku [13] a poté v [6], ale bez uvedené metodologie.

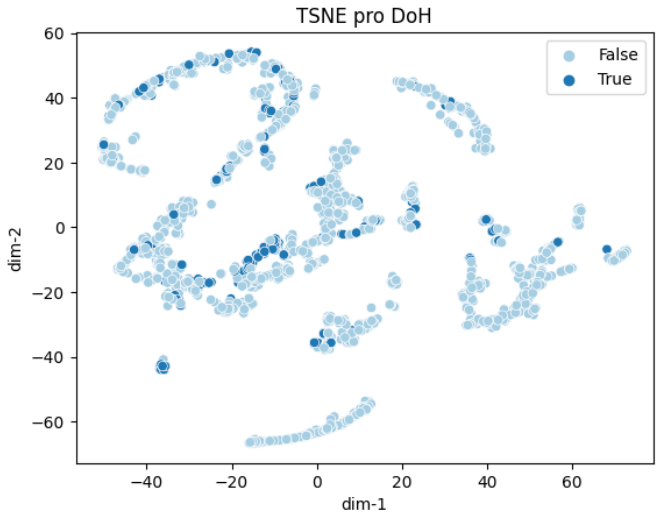
## 1.6 Učení s částečným dohledem

Učení s částečným dohledem *(Semi-Supervised Learning)* vychází ze stejné filosofie jako aktivní učení představené výše, tedy že existuje řádově méně označených dat než neoznačených. Tento přístup má několik zásadních předpokladů:

1. vzorky blízko sebe sdílejí třídu,
2. vzorky mají tendenci existovat v diskrétních klastrech a platí, že vzorky v klastru sdílejí třídu,
3. vzorky leží na množině mnohem menší dimenze, než je vstupní prostor.

Do jisté míry je tento přístup komplementární s metodami aktivního učení. Oba přístupy se snaží co nejvíce zužitkovat množiny neoznačených dat; aktivní učení vybírá neoznačená data k anotaci, kdežto přístup učení s částečným dohledem nevyužívá anotátor (poskytující základní pravdu, *ground truth*), ale označuje vzorky na základě struktury dat podle předpokladů výše.

Pro účely této práce je tato metoda poměrně nevýhodná v čisté podobě. Jednak předpoklady výše uvedené neplatí (viz obrázek 1.6, vizualizace pomocí metody TSNE) a jednak takto označené vzorky nelze využít k vylepšení dosavadní tréninkové sady, neboť by bylo potřeba tento vzorek stejně ověřit anotací, a i kdybychom to tak dělali, stejně musíme nějakým způsobem zajistit, aby se přidávaly pouze takové vzorky, které mají nějakou informační hodnotu. Po zahrnutí takových požadavků konvergujeme zpět k metodám aktivního učení. Popsána je kombinovatelnost [19, 30]. Vhodnost pro účely v diskutované doméně je pak tedy záležitostí experimentálního zhodnocení.




**Obrázek 1.6** Předpoklady pro učení s částečným dohledem v síťovém provozu neplatí. Pomocí metody TSNE [9] vizualizujeme data DoH (více o této sadě v kapitole 4) a demonstrujeme, že v diskutované doméně třídy ne vždy tvoří klastry a metody částečného dohledu nemohou fungovat uspokojivě. Zde zobrazeno celkem 2500 vzorků s prevalencí pozitivní třídy 14.4%.

## 1.7 Shrnutí

V této kapitole jsme se seznámili se síťovými toky jakožto s agregačním formátem nad jednotlivými pakety v rámci jednoho síťového spojení. Analýza pomocí toků může být zcela nezávislá na použitém šifrování, a tedy veškeré diskutované metody mohou být bezpečným způsobem výzkumu a monitorování sítí z hlediska soukromí jejich uživatelů. Představili jsme si formát `UniRec` z frameworku NEMEA, který bude ve zbytku práce využívaný.

Definovali jsme několik způsobů měření přesnosti klasifikačních prediktivních modelů, které jsou založeny na matici záměn. Ty lze použít v laboratorním i reálném provozu s tím, že je potřeba zvláště v reálném nasazení zvážit vhodnost jednotlivých metrik, zejména s ohledem na potenciální nerovnovážnou distribuci jednotlivých klasifikačních tříd. Problematika měření přesnosti a výkonnosti systému bude ještě rozvedena v kapitole o experimentálním vyhodnocení.

Popsali jsme základní principy takzvaného aktivního učení, umožňující splnit oba cíle práce, tedy:

1. vylepšování výsledků algoritmů strojového učení pro klasifikaci síťového provozu, případně detekci bezpečnostních hrozeb,

**2.** aktualizace trénovací datové sady.

Zároveň jsme zanalyzovali existující práce a zvážili jejich metody pro případné využití v našem frameworku. Z toho vyplynulo, že převážná část prací se zabývá pouze hromadným zpracováním dat a pouze minorita se věnuje proudovému, který je pro naší doménu mnohem důležitější. Dlužno však podotknout, že pokročilejší metody obvykle požadují statickou množinu neoznačených dat (tedy musí od začátku znát distribuci dat) a relativně vysoký výpočetní výkon; tyto požadavky však nelze dost dobře zaručit.

# Kapitola 2
# Návrh

V této kapitole navrhneme řešení pro zlepšování klasifikace síťového provozu a s tím spojenou tvorbu datových sad. Metody aktivního učení umožňují oboje; nějaký model predikuje jednotlivé klasifikační třídy a zároveň se rozšiřuje trénovací sada o ty nejlepší (to jest informačně nejpřínosnější vzorky). Navíc tento způsob zaručuje odolnost vůči zastarávání datových sad, pokud by systém byl nasazen v *online* režimu.

## 2.1 Požadavky a případy užití

Z úvodu přirozeně vyplývají následující požadavky a zároveň přidáváme další rozumné požadavky.

**1.** Nefunkční požadavky:

**a. *Online* a udržitelné nasazení**

Chceme-li v reálném čase klasifikovat síťový provoz a při tom vytvářet anebo aktualizovat trénovací datovou sadu, musíme toto provádět *online*, abychom omezili nutnost skladovat síťové toky, kterých je velmi mnoho. Z hlediska bezpečnosti je případný škodlivý provoz značně senzitivní na včasný odchyt. Zároveň požadujeme udržitelné nasazení, tj. systém musí být odolný vůči chybám (například proti chybnému vstupu, chybějícím informacím v nějakém toku atd.) a musí být schopný se v případě selhání obnovit do stavu, ve kterém byl před havárií. Zálohování stavu zároveň umožňuje udržitelnost ve smyslu přenositelnosti například při migraci mezi servery.

**b. Univerzálnost vůči použitelnému prediktivnímu modelu**

Každý klasifikační problém v rámci libovolné domény vyžaduje použití jiného modelu s různými parametry. Navrhovaný systém by tedy měl být zcela nezávislý na použitém modelu a tím se stát univerzální pro různé problémy. Tato zapouzdřenost zároveň přináší tu výhodu, že lze systém realizovat zcela bez ohledu na znalosti vnitřní implementace požadovaného modelu a naopak, vnitřní implementace modelu je zcela nezávislá na navrhovaném systému, který operuje nad ním.

**c. Univerzálnost vůči vstupu**

Bez ohledu na to, co je vstupem, by měl systém zpracovat vše, k čemu bude mít implementovaný parser – tj. část zajišťující vstup by měla být oddělena od zbytku systému a její funkce by měla být převést vstup do interního formátu reprezentující síťové toky. Pro systém by pak neměl být problém ani zpracovávat přímo pakety, pokud bude implementovaný

agregátor síťových toků; takový požadavek přímo není, ale mělo by to na tyto eventuality být připraveno.

**d. Nasaditelnost v prostředí frameworku NEMEA jako samostatného modulu**
Framework NEMEA je výkonný a efektivní systém na proudové zpracovávání síťového provozu [2, 3, 31]. Požadujeme, aby byl navrhovaný systém schopen v tomto prostředí fungovat jako nezávislý modul bez nutnosti jakéhokoliv zásahu do frameworku NEMEA.

**e. Jednoduchá rozšiřitelnost**
Problematika síťové klasifikace je značně dynamický obor s velmi rychlým rozvojem (například většina literatury, ze které vychází tato práce, je z poslední dekády). Vznikají nové formáty, nové vzorkovací strategie, nové datové struktury. Od navrhovaného systému očekáváme, že do jisté míry lze většinu součástí nahradit nebo případně jednoduše rozšířit bez detailní znalosti celého systému.

**f. Budoucí škálovatelnost** Do jisté míry vyplývá z prvního požadavku a předpokládá předchozí. Výstupní systém z této práce bude ve stádiu prototypu a nebudeme u něj klást příliš vysoké nároky na výkonnost a efektivnost, ale měl by být připraven na případné škálování ať už jeho paralelizací nebo distribucí některých nákladných částí na výpočetní servery (například trénink modelu, který může být značně náročný, nebo využívání nějakého file serveru či databázového serveru k ukládání dat).

**2.** Funkční požadavky:

**a. Predikování tříd síťových toků**
Evidentní požadavek vyplývající přímo z potřeby existence takového systému.

**b. Využití anotátoru jako garanta správnosti označení toku**
Ve strojovém učení je kruciální mít data označena třídou *co nejvíc správně*. K tomu chceme využít anotátor, což bude nějaká část systému poskytující předpokládané správné označení (*ground truth*). Anotátor je plně deterministický.

**c. Aktualizace datových sad**
Inkrementace od $\mathcal{D}_0$ k $\mathcal{D}_i$. Chceme, aby systém nejenom predikoval, ale zároveň vytvářel rozumné datové sady, které mohou být znovu využity například k různým analýzám. Měla by existovat i možnost nedestruktivní iterace, tj. předchozí sada není přepsána svou následovnicí.

**d. Vyhodnocování a sledování výkonu**
Je nutné sledovat výkon. Kromě prediktivního výkonu podle relevantních metrik by mělo být též možné sledovat doby výpočtů, počet vzorků prošlých systémem atd. Zároveň by měl existovat mód vyhodnocování, kdy evaluační procedura záměrně anotuje všechny vzorky a následně je porovná s predikcí. Tento mód by mohl být používán při laboratorních experimentech.

**e. Post zpracování datových sad**
Systém by měl umět po zpracování toků zároveň provádět operace nad získanými a současnými trénovacími daty v množině $\mathcal{D}_i$. Může se jednat například o *undersampling* nebo naopak *oversampling* či korekce třídy lidským anotátorem.

## 2.2 Framework

Softwarový framework leží na pomezí mezi knihovnou a klasickým softwarovým produktem; dal by se nazvat polotovarem. Zatímco knihovna je množina funkcí a struktur, framework je šablona, v rámci které se dají vystavovat softwarové produkty. Softwarový framework se skládá z měnitelné části (*hot spots*), což je prostor pro uživatele frameworku (programátora), který se může

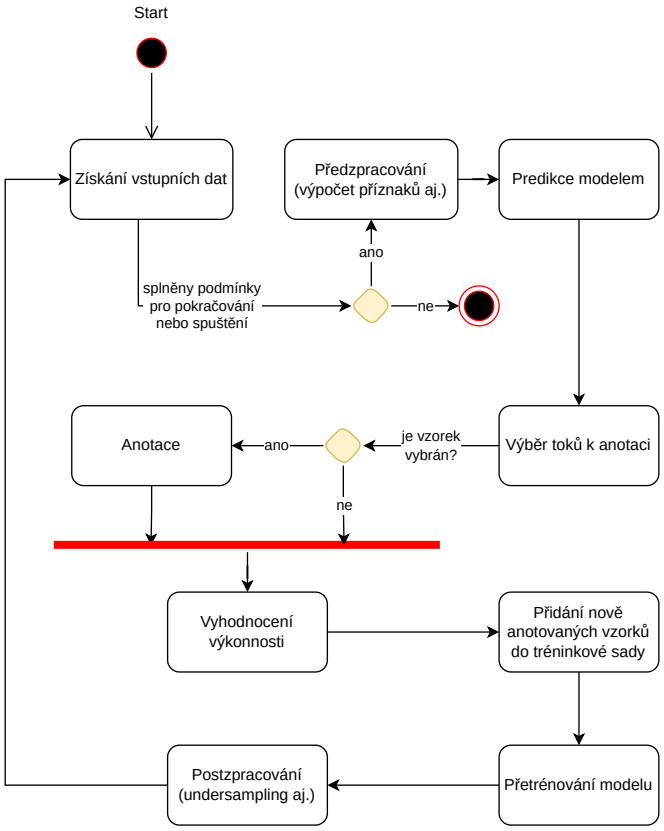
(a měl by se) měnit podle daného způsobu využití. Naproti tomu neměnné části (*frozen spots*) definují architekturu systému a vztahy mezi jednotlivými komponenty [32]. V řeči objektově orientovaného programování se framework skládá víceméně z abstraktních tříd a implementovaná je pouze část [33]. Programátor (uživatel frameworku) pak nijak nemusí implementovat komunikaci mezi třídami. Návrh řešení jako frameworku zajišťuje snadnou rozšiřitelnost a univerzálnost a splňuje tak část nefunkčních požadavků.

V rámci této práce budou nicméně všechny třídy nějak implementovány. Samotný framework bude sloužit k experimentálnímu vyhodnocení a demo nasazení. Zároveň implementace tříd poslouží jako příklad pro případné uživatele frameworku.

## 2.3 Architektura

Při návrhu jádra frameworku se vychází z metody aktivního učení představeného v předcházející kapitole. Na obrázku 2.1 je znázorněn diagram aktivit ukazující základní tok dat systémem. Zároveň díky požadované rozšiřitelnosti lze využít oba typy aktivního učení definované v první kapitole a návrh bude umožňovat implementaci různých modifikací, které byly diskutovány na konci téže kapitoly.

V této sekci budou představeny základní funkční části frameworku, jak je popsáno v diagramu 2.1. Jsou popsány základní funkce, které tyto celky musí splňovat.




**Obrázek 2.1** Diagram aktivit návrhu.

### 2.3.1 Zdroj dat

Pro dostatečnou univerzálnost je návrh frameworku nezávislý na zdroji dat. Zdroj dat de facto rozhoduje, jaká verze (hromadná versus proudová) aktivního učení bude využita.

- Vstupy:
  - Definice zdroje (soubor, databáze aj.)
- Výstupy:
  - Dále zpracovatelná množina síťových toků

### 2.3.2 Předzpracování

Ze zdroje dat putují toky ke klasifikaci. Občas je ale nutné data upravit, například znormalizovat číselné hodnoty, vypočítat příznaky (jako je třeba poměr přijatých a odeslaných bajtů apod.), popřípadě provést nějaký další *feature engineering*. K tomu slouží část předzpracování.

- Vstupy:
  - Množina toků ze zdroje
- Výstupy:
  - Předzpracovaná množina toků, která by měla mít stejný tvar jako data v trénovací sadě (tj. stejné příznaky), aby ji mohl prediktivní model zpracovat

### 2.3.3 Prediktivní model

Jádro frameworku, obsahující samotný prediktivní model. Jeho primární funkce je predikovat třídy předzpracovaných toků na základě trénovacích dat. Samotný model je jeden ze stavů, který je třeba zálohovat, aby byl splněn požadavek na udržitelnost chodu, neboť vnitřní parametry modelu jsou přesně tím, co se časem mění (a v některých modelech rovněž podléhá jisté míře náhodnosti) a nepodléhá zcela vstupním parametrům.

Prediktivní model má dva základní módy fungování, a to je predikce a trénink odpovídající stavům *Predikce modelem* a *Přetrénování modelu* v diagramu 2.1. Funkce (1) predikce samozřejmě predikuje třídy (s jistou pravděpodobností) na základě vstupního toku. Funkce (2) trénování prediktivní model vytvoří (například sestaví rozhodovací strom).

- Vstupy:
  1. Předzpracovaná data
  2. Anotovaná data
- Výstupy:
  1. Předzpracovaná množina toků, která by měla mít stejný tvar jako data v trénovací sadě (tj. stejné příznaky), aby ji mohl prediktivní model zpracovat
  2. Nový model na základě aktualizované datové sady

### 2.3.4 Vzorkovací strategie

Společně s modelem základ fungování celého systému. Vzorkovací strategie přijímá výstup prediktivní funkce z modelu, zároveň má přístup k označeným i neoznačeným datům a na takovém základě rozhoduje, které vzorky předat k anotaci a následně uložit do trénovací sady $\mathcal{D}_{i+1}$ a které zahodit nebo nechat do další iterace (v závislosti na užití).

- Vstupy:
  - Seznam toků s přiřazenými posteriorními pravděpodobnostmi z prediktivního modelu, celá sada $\mathcal{D}_i$ i $\mathcal{U}$
- Výstupy:
  - Množina toků určených k anotaci

### 2.3.5 Anotátor

Anotuje toky. Anotátor lze rozdělit dle přístupu na

**pasivní** například podle nějakého *blacklistu* IP adres, nebo

**aktivní** například přímým dotazem a zkoumáním odpovědi na zdroj nebo cíl toku.

Tato součást frameworku je nejvíce flexibilní. Všechny ostatní součásti mohou mít nějaké základní verze, ale v případě anotátoru je vždy na uživateli frameworku, jak má anotátor fungovat. Jeho výstup je považován za pravdivý.

- Vstupy:
  - Množina toků vybraných k anotaci z výstupu vzorkovací strategie
- Výstupy:
  - Anotované toky přidané do $\mathcal{D}_{i+1}$

### 2.3.6 Vyhodnocení

Vyhodnocením se sleduje prediktivní výkonnost celého systému. V závislosti na implementaci si lze představit dva hlavní scénáře v závislosti na nasazení, a to:

1. v případě ostrého provozu si na začátku iterace lze sadu $\mathcal{D}_i$ rozdělit na trénovací a testovací (v určitém poměru) a do vyhodnocení pomocí zvolených metrik posílat právě ty testovací,
2. nebo v případě experimentálního provozu v laboratorních podmínkách lze opět sadu rozdělit na testovací a trénovací, ale vyhodnocení provést pak i na celé $\mathcal{U}_i$, která se pro tyto účely celá anotuje (což s sebou potenciálně přináší značnou režii); v tomto režimu získáváme dvě metriky a lze je srovnávat (výkon na testovacích datech pak může být brán jako reference pro zkoumané vzorky).

### 2.3.7 Post zpracování dat

Po predikci, výběru a anotaci na konci funkční smyčky je občas nutné provést nějakou operaci nad sadou $\mathcal{D}_i$, například se můžeme pokusit o *oversampling*, nebo *undersampling* nebo aktualizovat třídy v sadě $\mathcal{D}_i$ na základně nějaké zpětné vazby. Tato část by měla přímo splňovat jeden z požadavků.

### 2.3.8 Správce tréninkové databáze

Tento modul obstarává správu datové sady. Technicky se jedná o adaptér buď k nějakému databázovému stroji, nebo případně k nějakému strukturovanému souboru. Musí umět základní úkony, jako načíst toky, uložit toky, změnit toky, vymazat toky. Žádná pokročilejší funkcionalita není požadována. Databáze by měla být dostupná ve většině částí frameworku; to je vhodné reflektovat správně zvoleným návrhovým vzorkem (což bude diskutováno v následující kapitole).

### 2.3.9 Logování a ukládání metrik

Logování událostí a metrik funguje globálně a mělo by být přístupné z celého systému. Tato součást musí fungovat zcela nezávisle na ostatních a neodesílá ostatním částem žádné zprávy, pouze přijímá a odesílá do standardního výstupu, souboru, databáze nebo nějakého metrikového serveru. Framework nepředepisuje žádný zvláštní požadavek.

## 2.4 Přehled existujících řešení

V této kapitole byly definovány požadavky a byl navržen framework, který tyto požadavky splňuje. Zbytek kapitoly je věnován krátkému přehledu existujících knihovnen a frameworků v oblasti aktivního učení a relevantním oblastem strojového učení; některé z nich poté budou využity v implementaci.

### 2.4.1 Komparace technologií strojového učení

Existuje značné [34] množství různých knihoven a frameworků implementujících modely strojového učení (popsané například v [12] nebo formálněji [14].

Nejpopulárnějším jazykem mezi uživateli je jednoznačně Python [34], který se těší všeobecné oblibě v mnoha oblastech[1]. Jeho výhoda je zřejmě značná syntaktická jednoduchost a rychlost prototypování (bohatá standardní knihovna s jednoduchým Application Programming Interface (API)). Také pro něj existuje velké množství nástrojů, jako je například `Jupyter Notebook` pro interaktivní používání, velké množství nativních vizualizačních nástrojů a podobně. Jeho nevýhoda může být nižší výkon, proto se standardně [34] kritické části přepisují do kompilovaného jazyka (C, C++, Rust).

V akademické sféře jsou populární dvě knihovny. Velmi kvalitní knihovna implementovaná v `C++` je `Shogun` [35], která implementuje mnoho modelů, je nejstarší a je stále v aktivním vývoji. Její nevýhodou je slabší dokumentace. Rozhodně nejpopulárnější [35] knihovnou současnosti je `Scikit-learn` [9], která je zároveň nejrychleji se rozvíjející s dobře hodnocenou kvalitou implementace. Její nevýhodou je omezená možnost paralelizace a tedy zrychlování základních úkonů, výhodou je velmi kvalitní dokumentace a snadnost použití. V průmyslu je využívaná velmi výkonná knihovna sponzorovaná Microsoftem `Vowpal Wabbit` s omezenější nabídkou modelů, ale s dobrou škálovatelností [35].

Pro využívání principů hlubokého učení a neuronových sítí existuje knihovna `TensorFlow` a nad ní existuje *high level* rozhraní `Keras`, zaměřený na vysokou modularitu [34]. Velmi používaný je pak i `PyTorch`.

### 2.4.2 Knihovny pro aktivní učení

Na rozdíl od samostatných modelů a algoritmů strojového učení, v oblasti aktivního učení neexistuje ucelený přehled a komparace. Cílem práce není poskytnout porovnání knihoven pro

[1] viz například výsledky každoročních anket serveru *StackOverflow*

automatické učení a ani se o to nebudeme snažit, pouze provedeme rešerši existujících řešení a prodiskutujeme, zda se hodí pro řešení výše uvedených požadavků.

Knihovna `Vowpal Wabbit` obsahuje podporu pro aktivní učení, ale s omezením na binární predikce. Jinak si knihovna klade za cíl rychlé *online* učení, ve kterých dosahuje dobrých výsledků, ale pro naše účely chybí způsob, jak vybírat data k anotaci a aktualizovat tak datovou sadu.

Knihovna `modAL` [36] je vysoce modulární a nadprůměrně dobře dokumentovaná knihovna s MIT licencí postavená nad knihovnou `Scikit-Learn`. Její architektura je navržena podobně jako jádro architektury našeho frameworku, tedy některé její části by nejspíše bylo možné převzít, popřípadě rovnou použít jako jádro. Obsahuje velkou část strategií, které byly popsány v první kapitole. Zásadní nevýhoda je, že se s velkou pravděpodobností jedná o *abadonware*, aktuálně více než rok neaktualizovaný kód a je zde příliš mnoho neimplementovaných částí a přes 60 neuzavřených *Issues*. Její hlavní přínos je právě implementace strategií (i když pokročilejší metody state-of-the-art chybí, včetně těch jednodušších, jako například skórování dle informační hustoty). Podobné výhody a nevýhody má i knihovna `ALiPy` [37].

Google má open source repozitář [38] s Apache licencí, kde experimentuje s různými metodami a implementuje i některé pokročilejší vzorkovací strategie. Existují projekty, které implementují metody aktivního učení v rámci hlubokého učení [39, 40]. Knihovna `libact` [41] přichází se svojí vlastní strategií `active learning by learning`, která si klade za cíl vybírat nejlepší strategii za běhu. Tato knihovna však explicitně nepodporuje proudové zpracování. Z hotových softwarových produktů lze vyzdvihnout například [42], který je však zaměřen na klasifikaci dokumentů z oblasti NLP (*Natural Language Processing*).

## 2.5 Shrnutí

V této kapitole byly formulovány požadavky, které máme od navrhovaného řešení. Byly představeny funkční části systému, které tyto požadavky splní. Tento návrh se stane základem pro následující kapitolu, kde takové řešení implementujeme do funkčního prototypu. Na konci kapitoly jsme provedli rešerši existujících knihoven a frameworků. Některé jsou vhodné jako součásti v samotné implementaci (například prediktivní modely). Z rešerše existujících implementací aktivního učení vyplývá, že neexistuje řešení, které vyhovuje formulovaným požadavkům, a je rozumné implementovat vlastní.

# Kapitola 3
# Implementace

V předchozí kapitole byl popsán návrh frameworku pro automatické zlepšování klasifikace síťového provozu a tvorbu datových sad. V této kapitole bude popsána konkrétní implementace a dokumentace některých důležitých součástí.

## 3.1 Výběr částí

Níže si představíme důležité části systému v jejich konkrétní implementaci.

- **Načítání dat**

  Problematika načítání dat je předepsána rozhraním `InputManager`. To předepisuje pouze jednu metodu `get()`, která dle nějaké definice načte další data. Jsou k dispozici implementace:

  - `TrapcapSocketInputManager` načítá toky ve formátu `UniRec` přímo ze socketu
  - `TrapcapFolderInputManager` načítá toky z jednoho adresáře ze souborů `.trapcap`, které obsahují toky ve formátu `UniRec`
  - `CSVFolderInputManager` načítá toky z jednoho adresáře ze souborů `CSV`
  - `NEMEAInputManager` implementuje TRAP Interface Specifier (IFC_SPEC) systému NEMEA

- **Prediktivní model**

  Samotný prediktivní model je zapouzdřen a reprezentován abstraktní třídou `MLModel`. Ta samotná implementuje metody:

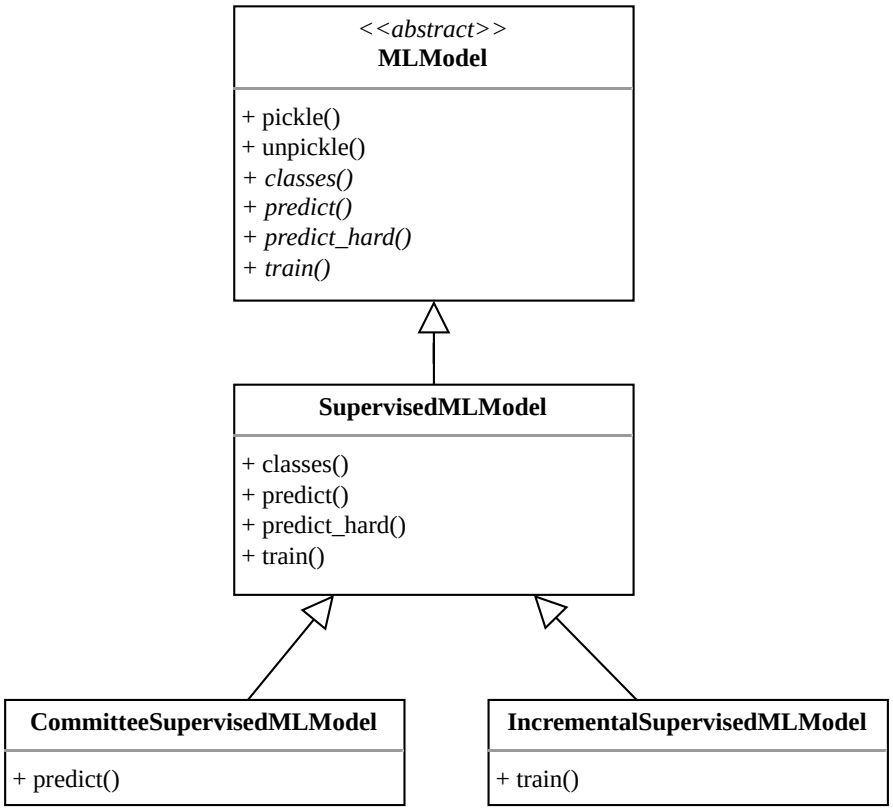
  - `pickle()`, která uloží model
  - `unpickle()`, která model načte ze souboru

  Tato komplementární dvojice metod zajišťuje požadavek na udržitelnost. Mimo to předepisuje následující funkce, které je nutno implementovat:

  - `classes()` vrací seznam tříd, se kterými se pracuje
  - `predict()` vrací list posteriorních pravděpodobností, popřípadě list listů při použití více klasifikátorů
  - `predict_hard()` vrací vlastní predikci třídy
  - `train()` trénuje klasifikátor nebo klasifikátory

Diagram tříd poskytující funkcionalitu prediktivního modelu je znázorněn na diagramu 3.1. Implementovány jsou celkem 3 třídy a všechny jsou vnitřně implementovány jako návrhový vzor adaptér pro API definované knihovnou `scikit-learn` [9, 43].

- `SupervisedMLModel` implementuje samostatný klasifikátor
- `InkrementalSupervisedMLModel` upravuje metodu `train()` tak, že místo metody `fit()` je používaná metoda `partial_fit()` [43]
- `CommitteeSupervisedMLModel` upravuje metodu `predict()` tak, že vrací seznam posteriorních pravděpodobností jednotlivých tříd pro každý klasifikátor; implementováno pomocí třídy `VotingClassifier` [44]



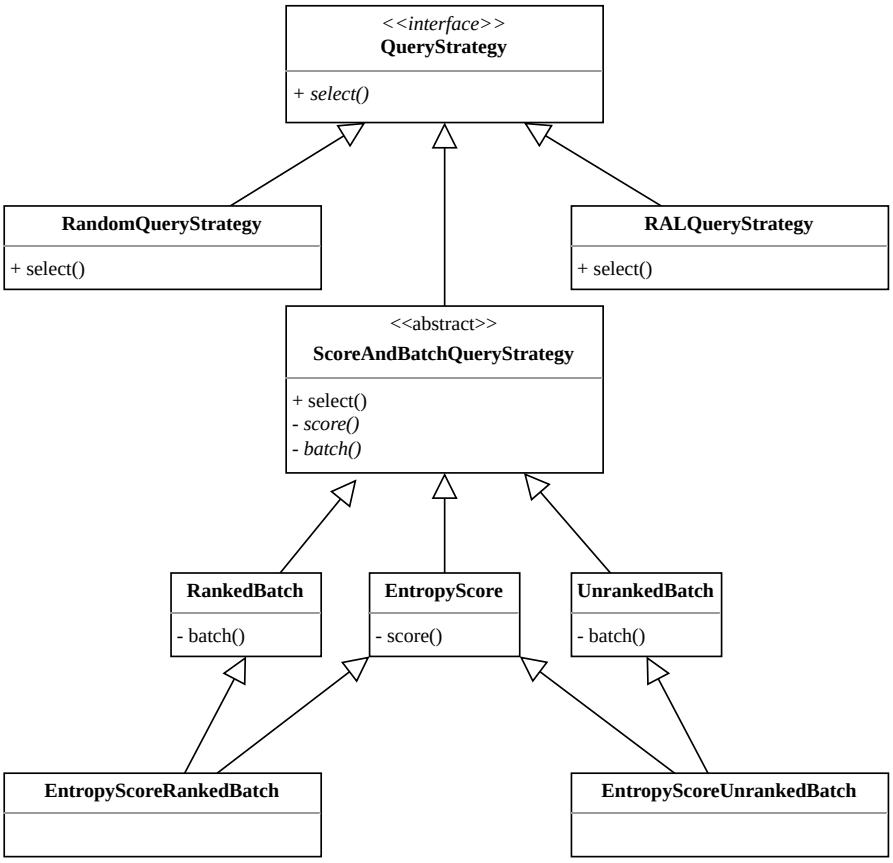

**Obrázek 3.1** Diagram dědičnosti jednotlivých implementovaných modelů.

- **Vzorkovací strategie**

  Základem je rozhraní `QueryStrategy`, předepisující metodu `select()`, která dle parametrů vybírá vzorky, které se pošlou k anotaci a uložení do $\mathcal{D}_{i+1}$. Existují 3 implementace tohoto rozhraní:

  - `RandomQueryStrategy` pro náhodné vzorování
  - `RALQueryStrategy` implementující strategii *Reinforcement Active Learning* z [16, 17]; jde o algoritmus vylepšující základní vzorkování o zpětnou vazbu v podobě *online* změn parametrů
  - `ScoreAndBatchQueryStrategy` implementuje základní strategie uvedené v první kapitole

Poslední jmenovaná třída je abstraktní. Implementuje sice funkci `select()`, ale pracuje v ní s privátními metodami `score()` a `batch()`. Ty kopírují definice z první kapitoly. Výsledné strategie se skládají; je potřeba, aby jazyk zvládal vícenásobnou dědičnost, což `Python` umí.




**Obrázek 3.2** Část tříd implementující vzorkovací strategii. Kromě třídy `EntropyScore` existují další skórovací třídy, ale pro přehlednost je uvedena pouze entropie. Úplný přehled je k dispozici v technické dokumentaci.

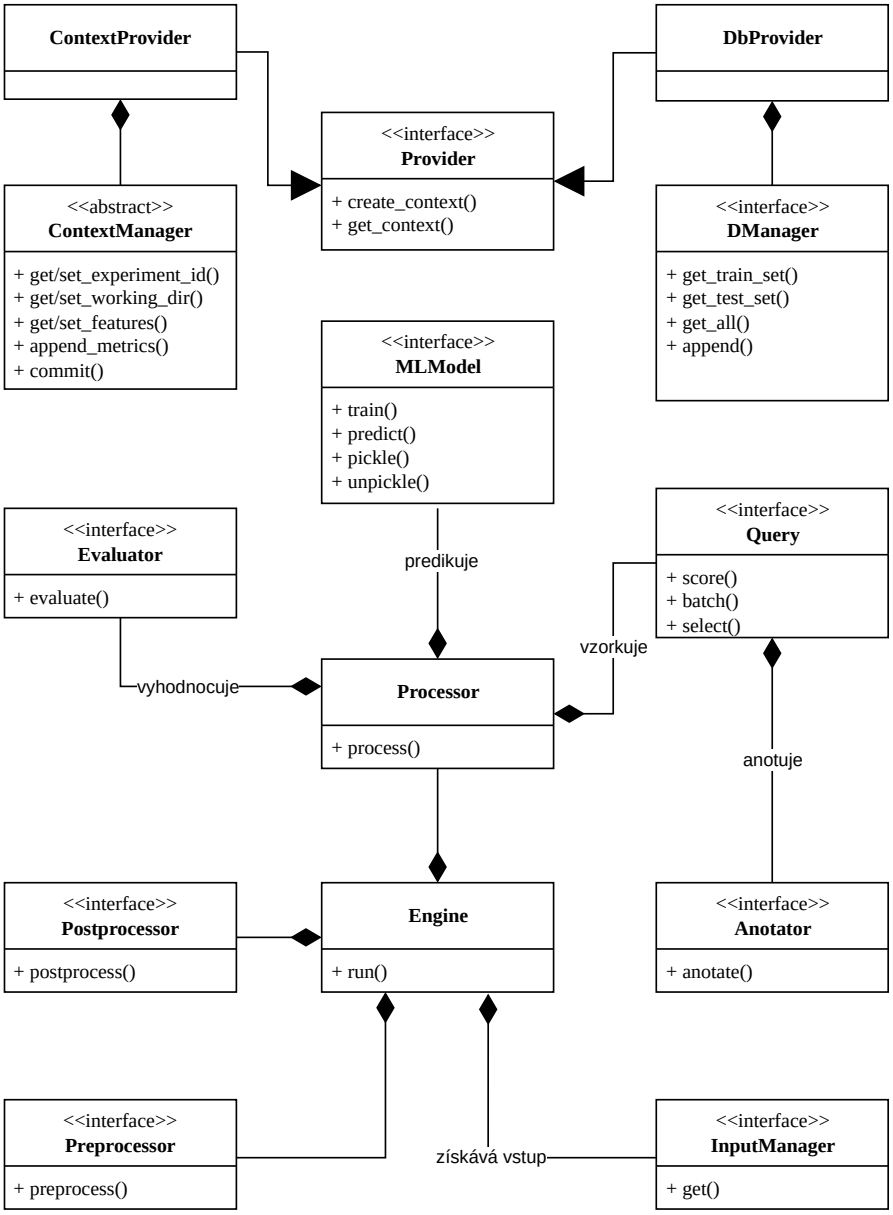
## 3.2 Kompletní pohled

Po diskuzi nejzajímavějších a nejdůležitějších součástí představíme a popíšeme framework jako celek. UML diagram tříd je zobrazen v obázku 3.3.

Centrem celého frameworku je třída `Engine`. Ta zajišťuje chod smyčky:

1. načíst data pomocí `InputManager`
2. předzpracování dat (`Preprocessor`)
3. zprocesování dat:
   a. predikce (`MLModel`)
   b. vzorkování a anotace (`QueryManager` a `Anotator`)
   c. vyhodnocení (`Evaluator`)
   d. postzpracování (`Postprocessor`)

Kontext celému běhu zajišťuje třída `ContextManager`, která drží technické věci, jako je `id` experimentu, pracovní složku a seznam příznaků, nad kterými se pracuje v predikcích a slouží pro ukládání metrik. Třída `DManager` pak poskytuje přístup do $\mathcal{D}_i$. Tyto třídy jsou globálně k dispozici odkudkoliv. Toho je dosaženo návrhovým vzorem singleton.




**Obrázek 3.3** Kompletní systém v UML diagramu. Jsou vynechány dědičnosti a je akcentováno propojení jednotlivých tříd.

## 3.3 Framework jako modul NEMEA

V této podkapitole představíme, jak framework použít, a demonstrujeme to na sestavení modulu pro systém NEMEA.

Podle dokumentace systému NEMEA, je modul samostatně spustitelný program nebo skript, který musí při spuštění přijímat argument `i` pro specifikaci rozhraní IFC_SPEC [45]. Vyvíjený framework má v kontextu systému NEMEA pouze 1 vstup a nemá výstup.

Toto chování je zajištěno třídou `NEMEAInputManager`. Ta v konstruktoru přijímá právě tuto specifikaci a umožňuje bezproblémové fungování v celém systému. Interně se používá knihovna `pytrap` [46] náležící do systému NEMEA.

Příklad minimálního sestavení tak, aby byl funkční v systému NEMEA, je ve výpisu kódu 1. Podrobný příklad sestavení je k nahlédnutí v příloze B.

```
from sklearn.ensemble import RandomForestClassifier

import alf

ifc = "u:input_socket"

ContextProvider = alf.context_manager.ContextProvider
DbProvider = alf.d_manager.DbProvider

ContextProvider.create_context("file")
ContextProvider.get_context().set_features(["f1", "f2", "f3"])
ContextProvider.get_context().set_experiment_id("example1")
ContextProvider.get_context().set_working_dir("/tmp/alf")

DbProvider.create_context(
    context_type="file",
    d_0_path="/tmp/alf/d_0.db")

anotator = alf.anotator.AnotatorDoH(blacklist_path="/tmp/alf/bl.txt")
model = alf.ml_model.SupervisedMLModel(RandomForestClassifier())
query_strategy = alf.query_strategy.RandomQueryStrategy(
        anotator_obj=anotator
)
input_manager = alf.input_manager.NEMEAInputManager(ifc)

engine = alf.engine.Engine(
    preprocessor=alf.preprocess.PreprocessorDoH(),
    postprocessor=alf.postprocess.PostprocessorIdentity(),
    ml_model_obj=model,
    query_strategy_obj=query_strategy,
    evaluator_obj=alf.evaluator.EvaluatorTestAnotatedAndAllPredicted(),
    input_manager_obj=input_manager
)

engine.run()
```

**Výpis kódu 1** Minimální příklad modulu sestaveného z navrhovaného frameworku. Konfigurace pro klasifikaci DoH.

## 3.4 Shrnutí

Představili jsme zajímavé a důležité části vyvíjeného frameworku a poskytli ucelený vhled do jeho fungování a propojení jednotlivých součástí. V této práci se implementaci věnujeme relativně povrchně a stěžejní je v tomto technická dokumentace, která je generovaná přímo z kódu z atributů `docstring`. Při vývoji byly využity standardní metody vývoje softwaru, takže případný další rozvoj by měl být velmi jednoduchý. K zdrojovému kódu v příloze je také sada jednotkových testů.

Všechny požadavky z předchozí kapitoly považujeme za splněné. Na konci této kapitoly jsme také poskytli příklad, jak používat samotný framework. Nejsou zde sice uvedené některé konkrétní implementace, ale zde opět čtenáře navádíme do zdrojových kódů a technické dokumentace.

Kapitola 4

# Experimentální vyhodnocení

V této kapitole poskytujeme vybraný přehled experimentů, které byly provedeny, aby ověřily funkčnost navrženého řešení a jeho implementace v různých prostředích, a představíme, jak lze framework používat v *online* provozu. Způsob vyhodnocení je tedy v podstatě dvojí – jednak *offline* vyhodnocení, kdy simulujeme síťový provoz z balíku připravených datových sad, a poté *online* vyhodnocení, kde je systém nasazen v provozu (viz dále) a sledujeme výsledky v delším časovém horizontu. K vyhodnocování se využívají metriky představené v první kapitole, především potom Matthewsův korelační koeficient (preferujeme) a $F_1$ skóre. Vyhodnocení proběhne na dvou problematikách. Sběr dat pro tuto práci a provádění *online* testů bylo realizováno na kolektoru Enta ve sdružení CESNET.

## 4.1 DNS Over HTTPS

DNS over HTTPS (DoH) je protokol pro zasílání Domain Name System (DNS) dotazů a jejich přijímání přes HTTPS. Každý pár dotaz-odpověď je mapován na výměnu HTTPS [47]. Cílem tohoto protokolu je zvýšení zabezpečení mezi klientem a serverem pomocí šifrování, které je zajištěno pomocí HTTPS protokolu. Příklad, jak DoH funguje, je ve výpisu 2.

Kromě zřejmých přínosů, které může mít DoH pro uživatele, jsou zde i jisté negativní aspekty. Byly popsané vektory útoků zneužívající DoH k bezpečné komunikaci v rámci botnetu, například `Godlua Backdoor` [48] nebo `Flubot` [49]. Kvůli nevyužívání konvenčního protokolu DNS jsou botnety velmi špatně identifikovatelné v ostatním provozu.

### 4.1.1 Předzpracování a anotátor

Jak bylo zmiňováno v předchozím textu, hlavní prerekvizitou k využití frameworku je mít definovaný anotátor pro každý klasifikační problém. Zde je to poměrně přímočaré, neboť ke klasifikaci DoH je použita pasivní metoda *blacklistu* [5]; tedy anotátor ověří, zda je zdrojová nebo cílová adresa toku na seznamu DoH serverů, a jestliže je, je označen pozitivně (tedy: je DoH), jinak negativně (není DoH).

Co se týká použitého vektoru příznaků, používají se jednak ty vyplývající přímo z `UniRec` (počet bajtů, paketů a podobně), a jednak i několik dopočítaných v rámci předzpracování. Přehled všech příznaků, které se využívají k predikcím, je v tabulce 4.1. Samotná implementace předzpracování i anotátoru je dodána v příloze na médiu. V tabulce se objevuje veličina symetrie, což je v tomto kontextu součet směrů paketů tak, že odchozí paket je mapován na $-1$ a příchozí na 1, tedy symetrie $= \sum_{p \in \text{pakety v toku}} \text{směr}(p)$. V rámci předzpracování také odfiltrujeme jednosměrné a krátké toky, ty nemohou z podstaty být DoH provoz.

```
$ curl -sH 'accept: application/dns+json' \
     'https://dns.google.com/resolve?name=www.fit.cvut.cz&type=A' | jq
{
  "Status": 0,
  "TC": false,
  "RD": true,
  "RA": true,
  "AD": true,
  "CD": false,
  "Question": [
    {
      "name": "www.fit.cvut.cz.",
      "type": 1
    }
  ],
  "Answer": [
    {
      "name": "www.fit.cvut.cz.",
      "type": 5,
      "TTL": 2612,
      "data": "wproxy.fit.cvut.cz."
    },
    {
      "name": "wproxy.fit.cvut.cz.",
      "type": 1,
      "TTL": 2612,
      "data": "147.32.232.212"
    }
  ]
}
```

**Výpis kódu 2** Příklad dotazu na DNS server společnosti Google přes protokol DoH. Odpověď je formátovaná jako JavaScript Object Notation (JSON). K dotazu je použit nástroj `curl`.

### 4.1.2 Počáteční trénovací datová sada

Nejdříve je potřeba počáteční datová sada $D_0$, ze které iteracemi vzniká aktualizovaná datová sada. Výběr takové datové sady vychází z [5], tedy využijeme datovou sadu s **vygenerovanými** toky. Tato datová sada v rámci našich experimentů simuluje situaci, kdy není možné z provozu vypreparovat reálná data, ale je nutné si nějaká vygenerovat. Již v první kapitole (tabulka 1.2 a 1.3) hovoříme o potenciální nevhodnosti takto generovaných dat jako zobecnění pro data odchycená v síti běžného provozu, navíc s časovým odstupem několika měsíců.

Byla vybrána podmnožina dat z [5] tak, aby pozitivní třída (pozitivní ve smyslu DoH) byla v rovnovážném poměru s negativní třídou. V experimentech vycházíme z velikosti $|\mathcal{D}_0^{\mathrm{DoH}}| = 1000$ s třídami $1:1$. Vizualizace této datové sady metodou TSNE je k dispozici na obrázku 4.1. Na pohled není vidět nějaká zásadní pravidelnost nebo rozdělení do klastrů, což vzhledem k tomu, že se jedná o standardní HTTPS protokol, není překvapivé.

**Tabulka 4.1** Seznam příznaků používané v experimentech s DoH. Kurzívou jsou vyznačeny ty, které jsou dopočítány v rámci předzpracování. Tučně pak navíc ty, které byly označeny jako nejvíce důležité pro klasifikaci (*feature importances*).

| Jméno příznaku | Popis |
|---|---|
| **bytes** | **Celková velikost odchozích paketů** |
| **bytes_rev** | **Celková velikost příchozích paketů** |
| packets | Celkový počet odchozích paketů |
| packets_rev | Celkový počet příchozích paketů |
| *packets_sum* | *Celkový počet paketů v toku* |
| ***bytes_ration*** | ***Poměr velikostí příchozích a odchozích paketů*** |
| ***num_pkts_ration*** | ***Poměr počtu příchozích a odchozích paketů*** |
| ***time*** | ***Celkový čas komunikace*** |
| ***av_pkt_size*** | ***Průměrná velikost odchozího paketu*** |
| ***av_pkt_size_rev*** | ***Průměrná velikost příchozího paketu*** |
| ***var_pkt_size*** | ***Rozptyl velikostí odchozích paketů*** |
| *var_pkt_size_rev* | *Rozptyl velikostí příchozích paketů* |
| *median_pkt_size* | *Medián velikostí odchozích paketů* |
| *median_pkt_size_rev* | *Medián velikostí příchozích paketů* |
| ***mindelay*** | ***Nejmenší časová prodleva mezi pakety*** |
| *avgdelay* | *Průměrná časová prodleva mezi pakety* |
| *maxdelay* | *Maximální časová prodleva mezi pakety* |
| *bursts* | *Třetina nejkratších prodlev toku* |
| *fizzles* | *Třetina nejdelších prodlev toku* |
| *time_leap_ration* | *Poměr bursts a fizzles* |
| *autocorr* | *Autokorelace prodlev* |
| *stSum* | *Symetrie první třetiny paketů* |
| *ndSum* | *Symetrie druhé třetiny paketů* |
| *rdSum* | *Symetrie třetí třetiny paketů* |

### 4.1.3 Offline experimenty

Zhodnotíme nejdříve chování systému v *offline* nastavení. Pro testovací data byl zvolen sběr na kolektoru Enta v síti CESNET v rozmezí pěti dnů. Nachystáme je tak, aby data obsahovala zhruba podobný poměr pozitivní a negativní třídy. Celkem se jedná o 90133 toků celkem ve 45 dávkách.

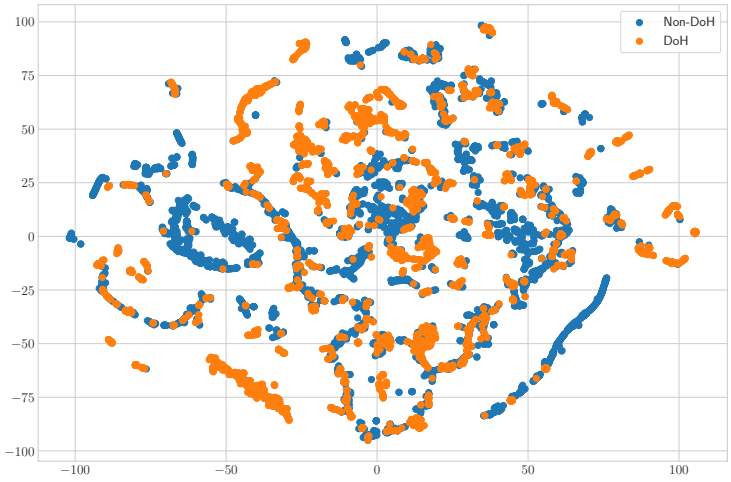
Celkem provedeme 3 sady experimentů nad touto datovou sadou:

**full** *ensemble* prediktivní model skládající se ze dvou náhodných lesů a tří stromů, všechny v základním nastavení hyperparametrů s tím, že růst stromů byl omezen, aby nedocházelo k *overfittingu*,

**reduced** stejný model, ale na redukovaném vektoru příznaků ve snaze předejít takzvanému *prokletí dimenzionality*; tyto příznaky jsou vyznačeny tučně v tabulce 4.1 a byly extrahovány z modelu z experimentu popsaném v tabulce 1.2,

**rf** náhodný les nad redukovaným vektorem příznaků s výchozí hyperparametrizací.

Provedli jsme celkem 6 typů experimentů pro 3 uvedené datové sady (viz tabulka 4.2). Tyto experimenty odpovídají popsaným základním strategiím z první kapitoly. Cílem těchto experimentů je ověřit, že strategie fungují, a porovnat je s naivním přístupem v podobě náhodného výběru.




**Obrázek 4.1** Vizualizace datové sady $\mathcal{D}_0^{\text{DoH}}$ metodou TSNE

**Tabulka 4.2** Tabulka provedených *offline* experimentů. Kromě náhodné strategie fixujeme velikost výběru na 20 vzorků v iteraci.

| název strategie | popis | sledované parametry |
|---|---|---|
| random | náhodné vzorkování | velikost výběru |
| uncert_unranked | nehodnocený výběr dle nejistoty | práh skóre funkce |
| uncert_ranked | hodnocený výběr dle nejistoty | práh skóre funkce |
| kl | výběr dle KL divergence (vyžaduje komisi) | práh skóre funkce |
| density_unranked | nehodnocený výběr dle informační hustoty | práh skóre funkce, $\beta$ |
| density_ranked | hodnocený výběr dle informační hodnoty | práh skóre funkce, $\beta$ |

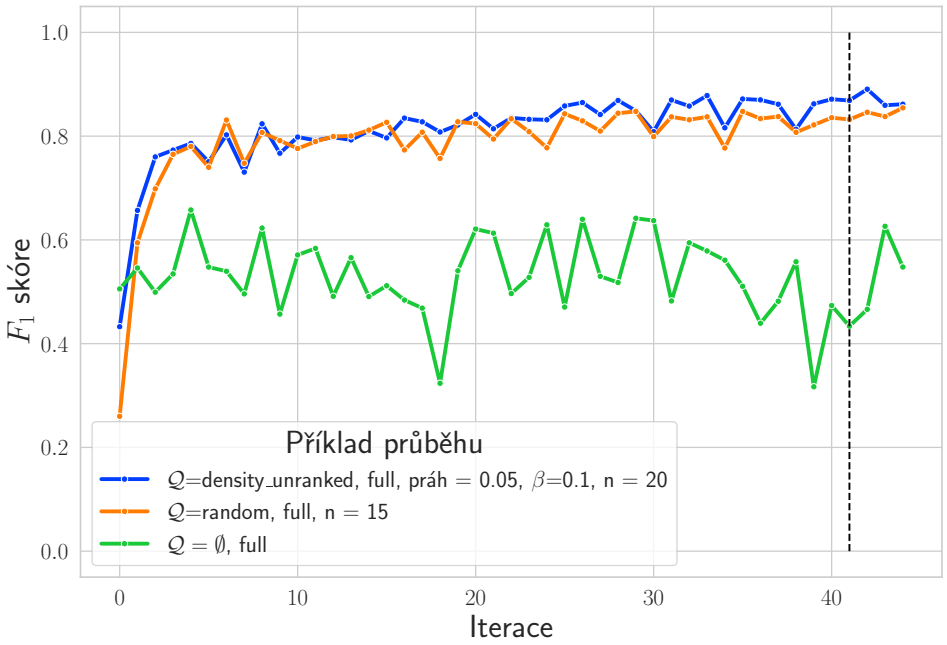
Experiment probíhá iterativně, ale je potřeba odvodit z celého průběhu výkonnost, abychom průběhy mohli jednoduše porovnávat. Empiricky bylo zjištěno, že nejlépe výkon vystihuje posledních 10% iterací. Příklady průběhů s vizualizovanou hranicí, odkud se vypočítává průměr, jsou na obrázku 4.2. Od každé parametrizace bylo provedeno 5 experimentů a za výsledek se pak považuje střední hodnota.

Náhled na konečné výsledky je na obrázku 4.3. Na této vizualizaci vidíme výsledky všech strategií v boxplotu a s výjimkou nejlepších a nejhorších výsledků vidíme, jak jsou jednotlivé strategie citlivé na parametry (čím širší box, tím je rozptyl výsledků větší a naopak, čím je užší, tím jsou výsledky podobnější pro všechny parametry).

První zjištění, které však není příliš překvapivé, je, že výkonnost velmi závisí na velikosti vzorku. Toto je demonstrováno na grafu 4.4.

Nejlepší výsledky pro každou strategii i s použitým prediktivním modelem a parametry lze vidět v tabulce 4.3. Výsledky jsou víceméně podobné mimo strategie náhodné a strategie založené na KL divergenci. Přesto je nutné konstatovat, že v tomto nastavení není náhodná strategie nejhorší a s ohledem na mimořádnou jednoduchost implementace by se jistě našly případy užití, kdy je tato strategie naprosto dostačující. To velmi koresponduje s výsledky v [6], kde dokonce náhodné vzorkování bylo nejlepší ze všech porovnávaných v problému klasifikace Virtual Private Network (VPN).

V *offline* experimentech jsme zjistili, že strategie založená na KL divergenci zcela propadá




**Obrázek 4.2** Příklad vybraných průběhů. Černá vertikální přímka určuje posledních 10 % iterací, ze kterých se odvozuje finální výsledek, který se poté porovnává.

i oproti náhodnému vzorkování. Náhodné vzorkování dopadlo velmi dobře. Konečný výběr vzorků nezávisí na způsobu, tedy výběr $n$ nejvyšších skóre je v tomto případě ekvivalentní s hodnoceným výběrem. Je to v souladu s klesající tendencí výkonu u zvyšování parametru $\beta$ u strategií density_* (obrázek 4.6) a také v souladu s vizualizací DoH vzorků metodou TSNE (pro připomenutí viz obrázek 4.1), tedy že vzorky v této problematice jsou si vzdálenostně velice blízko a snaha o promítnutí podobností do výběrů není příliš úspěšná. Poslední zjištění je, že práh skóre nemá na výkon modelu pozitivní vliv (obrázek 4.5.

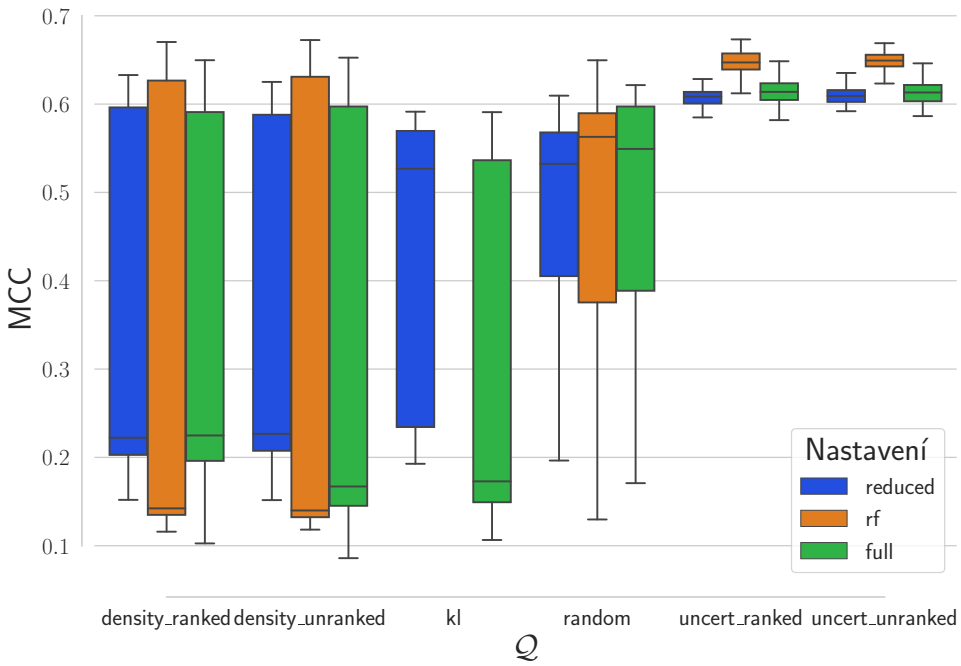
### 4.1.4 Online experimenty

Provádění experimentů v *offline* verzi má značné výhody; různá nastavení lze nad různými daty provádět opakovaně a sledovat tak vývoj chování systému. Méně je pak však vidět, jak se systém přizpůsobuje postupnému zastarávání datové sady.

Experimenty prováděné *online* mají v této práci dvojí význam, a to:

1. otestovat framework z hlediska spolehlivosti v delším časovém horizontu než řadově desítky iterací,
2. ověřit robustnost strategií v delším časovém horizontu a jejich schopnost se přizpůsobit zastarávání datové sady,
3. prezentace možností frameworku (nejen z hlediska kvality klasifikace, ale i z hlediska monitorování a nasazení v síťovém provozu, konkrétně na síti CESNET).

Pro účely *online* experimentu dodefinujeme dva požadavky, a to:




**Obrázek 4.3** Boxplot výsledků z *offline* experimentu nad DoH. Lze si všimnout, že nejméně citlivé na změny parametrů strategie jsou strategie uncert_*. Je ale nutné poznamenat, že ve strategiích density_* navíc sledujeme parametr $\beta$, nevyvozujeme z toho tedy zatím žádné závěry.

1. výsledky a logy je třeba průběžně uchovávat, a to v přístupné a dobře indexované podobě dle jednotlivých instancí a času,
2. výsledky je třeba vizualizovat v reálném čase.

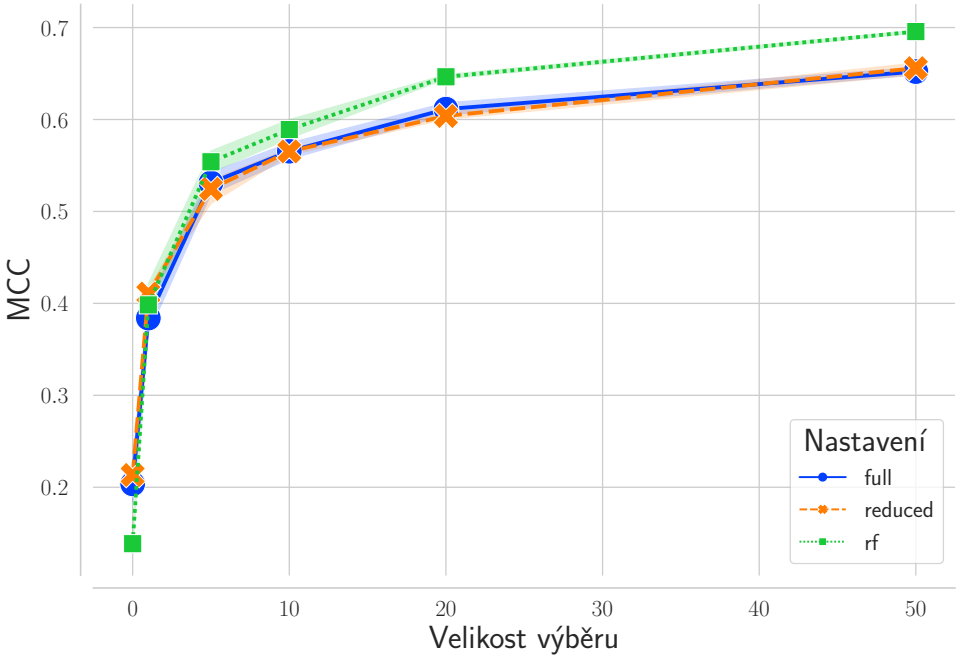
Pro ukládání výsledků volíme standardní relační Structured Query Language (SQL) open source databázi `MariaDB` [50]. Ta má tu výhodu, že kromě konvenčních polí zvládá ukládat a především parsovat i formát JavaScript Object Notation (JSON). To nese výhodu v tom, že formát logování a ukládání metrik je flexibilní – nejsme vázáni jedním konkrétním schématem a navíc formát JSON je přirozený datový formát pro datový typ `dict` v jazyce `Python`, který hojně užíváme. Pro naše účely zavedeme tabulku pro ukládání výsledků se schématem popsaným v tabulce 4.4.

Vizualizace takových výsledků je potom zajištěna vizualizačním nástrojem `Grafana` [51]. Ten se přímo připojí k výše uvedené databázi a lze nadefinovat, jaké metriky má zobrazovat a jakým způsobem. Příklad dotazu pro výběr $F_1$ skóre je ve výpisu 3. Kompletní implementace komunikace frameworku a databáze je popsána a k nahlédnutí v technické dokumentaci v příloze. Na obrázku 4.7 je diagram našeho frameworku pro *online* nasazení.

V *offline* verzi nás velikost datové sady $\mathcal{D}_i$ příliš nezajímala, jelikož množství vzorků nebylo nijak závratné a režijně bylo pohodlně zvládnutelné. V živém provozu s neznámou dobou běhu a neznámým celkovým počtem toků, které bude třeba predikovat, je nutné velikost datové sady omezit, aby nerostla takzvaně nad všechny meze. K tomu představujeme 3 jednoduché instrumenty v části zajišťující postzpracování [1]:

1. *undersampling* pomocí knihovny `imba`, konkrétně třídou `RandomUnderSampler`

[1] Byly zvoleny naivní metody postzpracování, pokročilejší metody již přesahují rámec této práce




**Obrázek 4.4** Lze si všimnout, že zvolená metrika se zlepšuje s vyšší velikostí výběru. Zde demonstrováno na strategii náhodného výběru.

2. vytyčení horního prahu velikosti a náhodné vybrání $n$ vzorků z databáze, které hodláme překlopit do další iterace, cokoliv nad $n$ zahodit

3. stejný způsob, ale vybrání je nenáhodné; vyhazuje se od nejstarších, v řeči Unixu `tail`.

Popíšeme dvě sady experimentů ze dvou časových období, které jsme v tomto nastavení provedli. První experiment používá třetí způsob omezování velikosti databáze, druhý experiment první a druhý způsob. Schémata zapojení jsou na obrázcích 4.8 a 4.9.

První běžel ve 4 instancích (popsány v tabulce 4.5) od 3. března 2022 15:58 do 5. dubna 2022 23:59. Průběh se zaznamenávaným MCC skóre je na obrázku 4.10. Červeně podbarvené části grafu jsou časové úseky, kdy software nebyl v provozu. Tímto se demonstruje jeden z požadavků, který na tento software klademe, a to je ve druhé kapitole definovaná udržitelnost. Lze si všimnout, že po opětovném startu systém pokračuje tam, kde skončil bez většího poklesu. Například poslední, zhruba týden starý úsek neměl u některých nastavení na výkon zásadnější vliv.

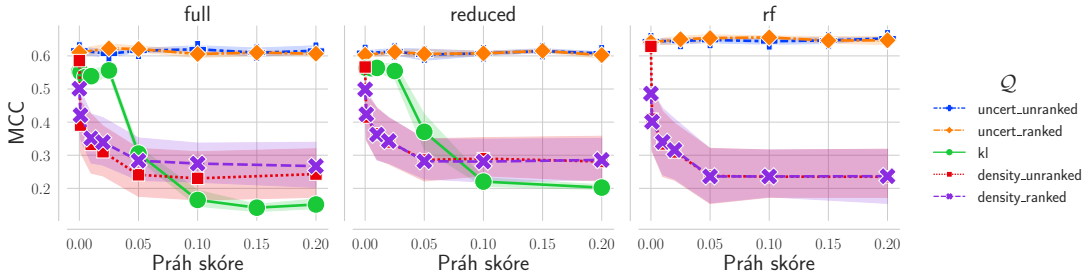
Pro porovnání výsledků jednotlivých experimentů budeme uvažovat skóre od 5. března. Rozsah hodnot, které jednotlivé parametrizace dosáhly, najdeme na obrázku 4.11. Přesný výčet hodnot je potom v tabulce.

Pro druhý experiment byla zvolena strategie RAL [16, 17] jako snaha o zajištění zpětné vazby. Oproti článkům [16, 17] přinášíme dvě změny: jednak hranice nejistoty se (pro binární klasifikaci) ustanovila tak, že musí ležet v intervalu $[0.6, 1]$ (empiricky jsme zjistili, že v pozorovaném problému s použitím takového modelu při poklesu nejistoty blízko 0.5 již prakticky nebylo možné hranici zvyšovat a RAL atrofoval k náhodné strategii) a jednak přetrénování modelu se odehrává v kontextu našeho frameworku po jednotlivých iteracích, které jsou dané proudem dávek, nikoliv po každém novém vyšetřeném vzorku.

Empiricky bylo již v *offline* experimentech zjištěno, že práh míry nejistoty nemá na výkonnost systému velký vliv. Zdálo by se tedy, že RAL nemůže mít velký vliv na funkčnost. Nicméně jeho

**Tabulka 4.3** Přehled nejlepších výsledků DoH *offline* experimentu. Seřazeno od nejlepšího po nejhorší v rámci strategie.

| Strategie | MCC | Práh skóre | $\beta$ | Nastavení |
|---|---|---|---|---|
| uncert_ranked | 0.673 | 0.050 | - | rf |
| density_unranked | 0.673 | 0.050 | 0.100 | rf |
| density_ranked | 0.670 | 0.020 | 0.100 | rf |
| uncert_unranked | 0.669 | 0.350 | - | rf |
| random | 0.650 | - | - | rf |
| kl | 0.591 | 0.005 | - | reduced |

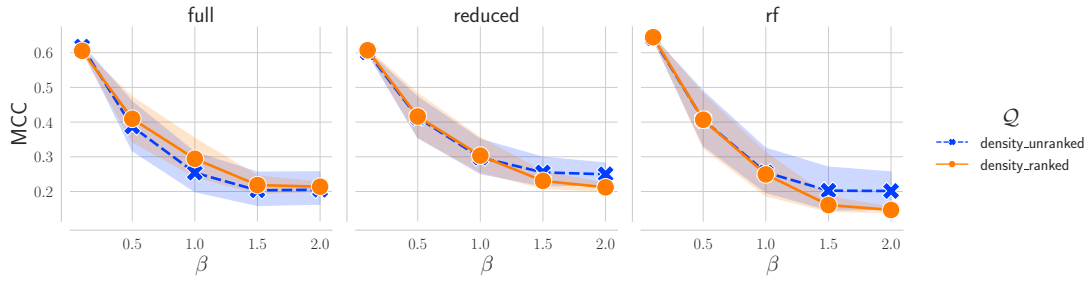



**Obrázek 4.5** Závislost strategií na prahu skóre.

výhoda leží jinde: díky tomu, že je zároveň regulována důležitost členů komise, může být tento model mnohem flexibilnější než ostatní komisionální strategie. V *offline* experimentech mimo jiné vyplynulo, že náhodný les převyšoval *ensemble* model složený z lesa a stromů. RAL by měl mít tu vlastnost, že špatně předvídající modely marginalizuje. Zároveň však zpětná vazba může tyto modely vrátit zpět do popředí. RAL tedy můžeme považovat za elegantní kombinaci základních strategií, které nějakým způsobem prokázaly v předchozích experimentech svou relevantnost. Naproti tomu strategie nějakým způsobem reflektující podobnost vzorků (hodnocený výběr a informační hustota) svou relevantnost neprokázaly, tedy takové prvky nemáme potřebu sem nijak implementovat.

Základem druhého experimentu je pouze jeden prediktivní model, a to *ensemble* model tří náhodných lesů bez žádné pokročilejší hyperparametrizace. V tabulce 4.7 je popis jednotlivých instancí.

Druhý experiment běžel v období od 26. dubna do 1. května. Průběhy se zaznamenaným MCC skóre jsou na obrázku 4.12. Pro porovnávání výsledků je vybráno období od 28. dubna. Instance `ral_small` byla spuštěna jako poslední s ohledem na průběhy zbylých dvou RAL. Ty totiž ve výchozí parametrizaci anotují zbytečně velké množství toků; `ral_small` bylo nastaveno tak, že budget je 250x menší než výchozí. Díky tomu anotoval podobné množství jako ostatní strategie, které měly nastavenou velikost výběru 10. Z toho důvodu nemá velký smysl srovnání zbylých strategií a RAL kromě `ral_small`. Ta instance navíc byla spuštěna až později, proto oproti ostatním non-RAL strategiím anotovalo pouze 50% všech toků. Přesto dokáže s přehledem dosahovat podobných výsledků jako `uncert*` strategie a překonávat KL a náhodnou strategii zřetelně. Navíc si lze dobře povšimnout, že opět nezáleží na zvoleném prahu skóre. Tyto výsledky jsou opět zanesené do boxplotu na obrázku 4.13 a tabulka s výsledky je v tabulce 4.8.




**Obrázek 4.6** Závislost na parametru $\beta$ u strategií založených na informační hustotě.

**Tabulka 4.4** Schéma tabulky v databázovém stroji MariaDB pro ukládání výsledků experimentů.

| Sloupec | Typ | Účel |
|---|---|---|
| id | int | ID záznamu |
| experiment_id | varchar | ID experimentu |
| timestamp | datetime | Časové razítko záznamu |
| log | json | JSON obsahující metriky nebo log |

## 4.2 Cryptominer

Rozvoj kryptoměn přináší mimo jiné potřebu je těžit, obvykle algoritmem *proof of work* (který využívá většina zásadních kryptoměn včetně Bitcoinu, Etherea nebo Monera). K těžení je používán specializovaný software, který je často propašován uživateli nelegálním způsobem a de facto se pak jedná o škodlivý software.

Sběr dat proběhl na síti CESNET pomocí pasivní metody *blacklistu*, kdy byly zjištěny IP adresy *mining poolů* 3 kryptoměn: Bitcoinu, Etherea a Monera. Blacklist se udržoval každodenním aktivním dotazem (`stratum request`, [52]).

### 4.2.1 Předzpracování a anotátor

Díky tomu, že se anotace prováděla již během sběru dat, nebylo potřeba vytvářet pokročilejší anotátor a tok vybraný k anotaci se pouze přidal do sady $\mathcal{D}_i$. V rámci předzpracování se z toků vybíraly příznaky uvedené v tabulce 4.9. Nasbíraná data byla rozdělena do jednotlivých iterací po 10 000 tocích.

### 4.2.2 Cryptominer a vytváření datových sad

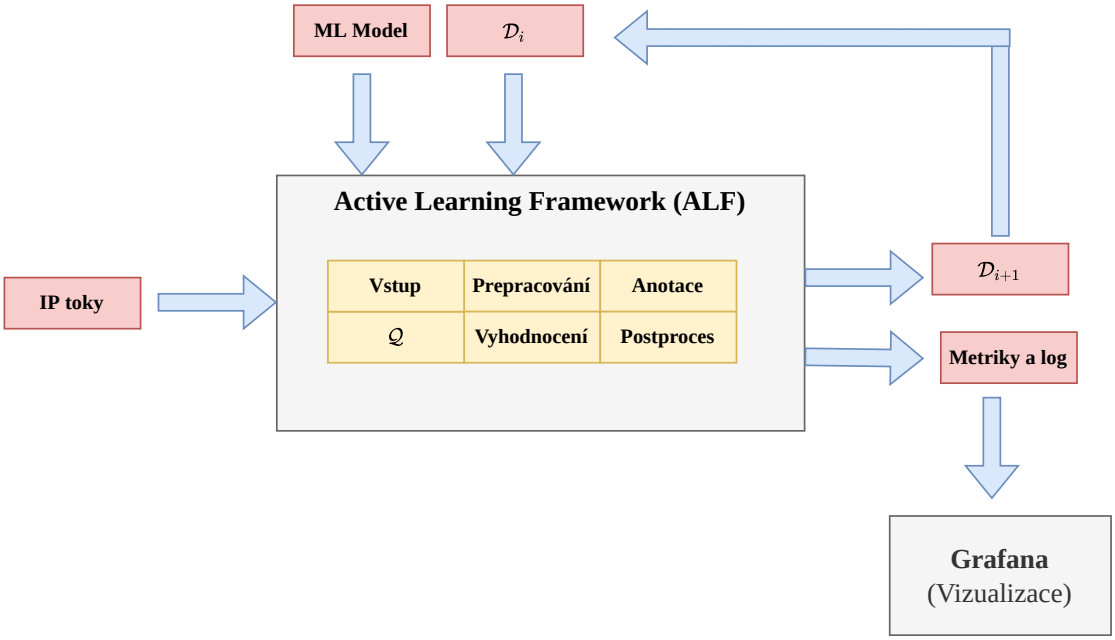
V těchto experimentech se zaměříme na případ užití k vytváření datových sad. Vybereme **zcela** náhodně extrémně malou datovou sadu $\mathcal{D}_\prime$ a postupně iterujeme. Provedeme 2 sady experimentů. V první sadě budeme přidávat v iteraci 10 vzorků do databáze, v druhé sadě pouze 1. Stejně jako v případě DoH, vyhodnocení jednoho experimentu proběhne na střední hodnotě posledních 10 % iterací (tedy 20 posledních iterací). Nomenklatura strategií zůstává jako v případě *offline* testů při DoH.

Diskutujme první experiment. Boxplot výsledků vystavěný stejně jako v případě DoH je na obrázku 4.14. Vliv parametrů strategií je pak na obrázcích 4.15a a 4.15b.

Diskutujme druhý experiment, kdy přidáváme právě jeden vzorek do sady. Výsledek prezentujeme opět prostřednictvím boxplotu na obrázku 4.16. Již bez ilustrace konstatujeme, že výkon-

```
SELECT
  timestamp AS "time",
  experiment_id AS "metric",
  CAST(JSON_EXTRACT(log, "$.test_all_predicted.f1") AS FLOAT) AS "f1"
FROM alf_experiment_record_v2
WHERE
  __timeFilter(timestamp)
ORDER BY timestamp
```

**Výpis kódu 3** Příklad získání $F_1$ skóre z databáze s rozdělením do jednotlivých experimentů (`experiment_id`), podle času.



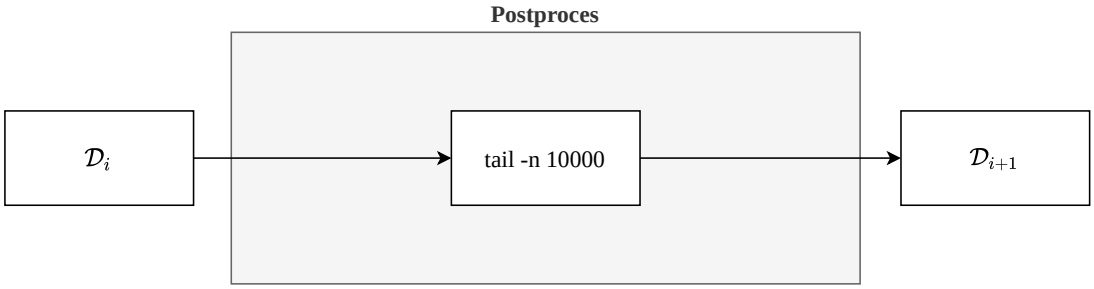

**Obrázek 4.7** Highlevel pohled na framework s vizualizační částí.

nost systému v závislosti na prahu nejistoty u `uncert` strategie se nemění. Průměrný výsledek náhodné strategie je MCC 0.808 se standardní chybou 0.005 a `uncert` má průměrný výsledek MCC 0.849 se standardní chybou 0.001. Konstatujeme, že náhodná strategie je v tomto případě již výrazně horší než `uncert`.

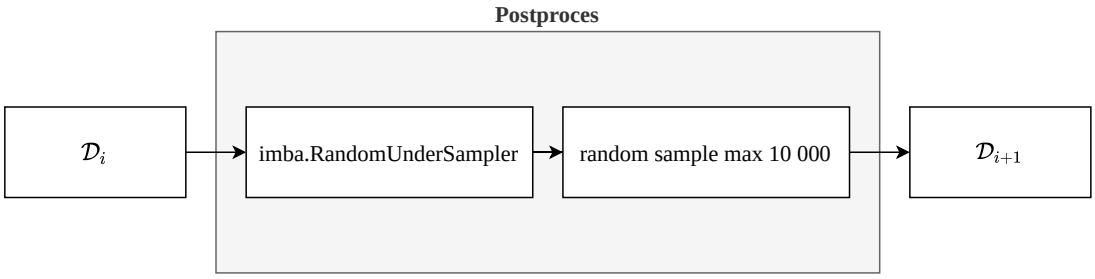
## 4.3 Shrnutí

Podařilo se experimentálně ověřit, že navržený a implementovaný framework funguje. Naivní přístup pomocí náhodného vzorkování funguje velice dobře (zcela v souladu s pozorováním v mnoha zmíněných článcích). Avšak ukazuje se, že výběry podle nejistoty predikce různých modelů fungují obvykle lépe. V mnoha případech se náhodná metoda velice blíží té založené na nejistotě, ale prakticky v žádném experimentu ji nepřekonává.

Strategie založené na podobnosti vzorků veskrze propadly. Vysvětlujeme si to tím, že zkoumané vzorky (respektive síťové toky obecně) jsou si v obecné rovině natolik podobné (a více to




**Obrázek 4.8** Postzpracování prvního online experimentu.



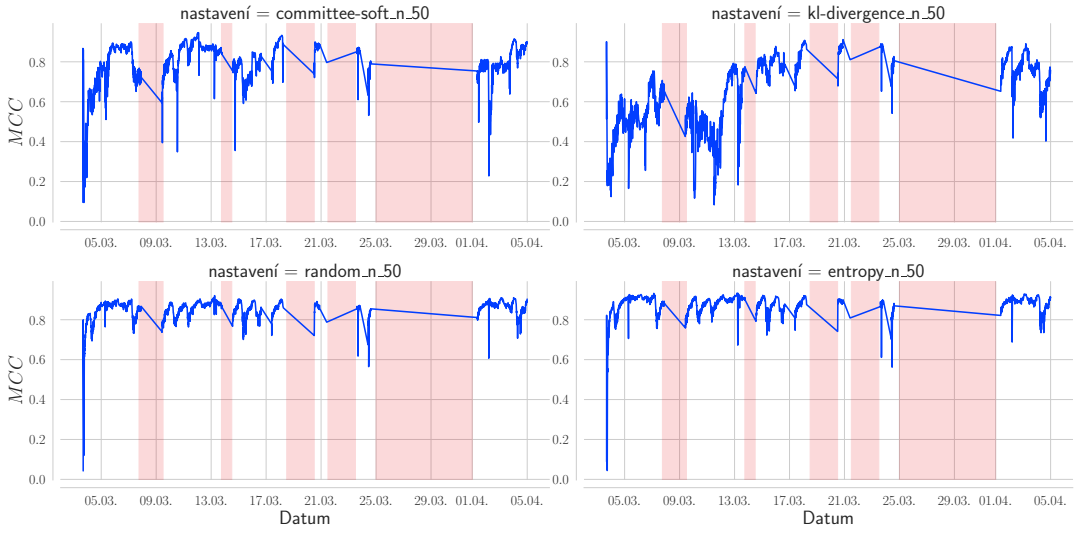

**Obrázek 4.9** Postzpracování druhého online experimentu.

vyniká v momentech, kdy se snažíme klasifikovat komunikaci nad stejným protokolem, jako například HTTPS v případě DoH), že tato metoda nejen nemá vliv, ale v mnoha ohledech dokonce negativně ovlivňuje výkon.

Strategie, která ve všech nastaveních propadla, je strategie založená na KL divergenci. Tato strategie je natolik špatná v diskutovaných doménách, že několikrát prošla revizí, ale konstatujeme, že implementace je správná a podpořena testováním s verifikovanými výsledky.

**Tabulka 4.5** Popis instancí v provozu od 3. března do 5. dubna

| experiment_id | popis |
|---|---|
| committee-soft_n_50 | 2x náhodný les a NB klasifikátor, soft vote entropie (rce 1.11) |
| kl-divergence_n_50 | stejná komise, ale použita KL divergence (rce 1.14) |
| random_n_50 | náhodné vzorkování, náhodný les |
| entropy_n_50 | kvantifikace nejistoty dle entropie (rce 1.10) |



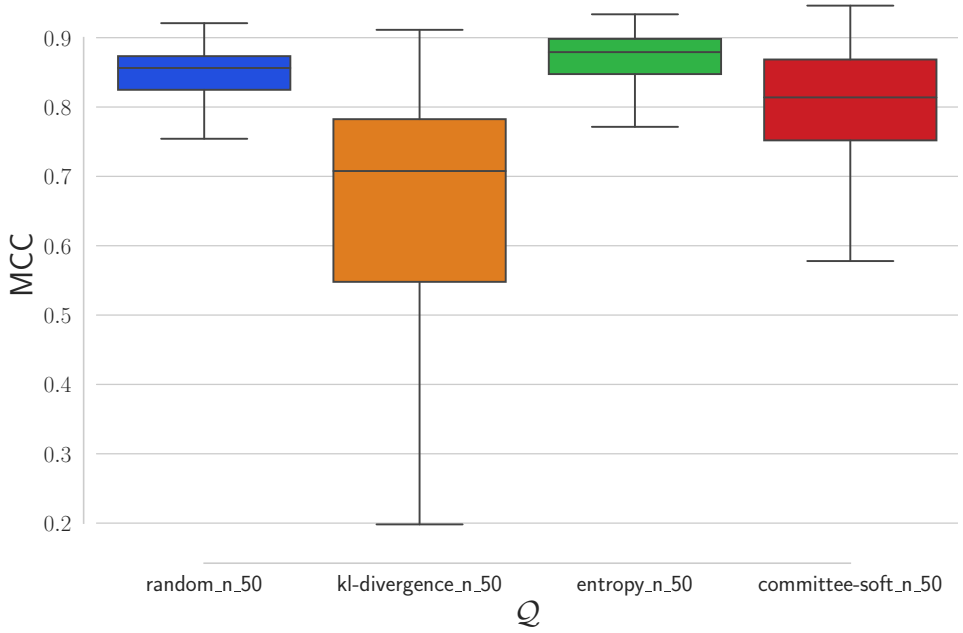

**Obrázek 4.10** Přehled průběhu první sady experimentů během jednoho měsíce. Červeně jsou podbarveny časové úseky, kdy systém nebyl v provozu. V případě kratších to byly výpadky způsobeny neodladěnými chybami, v případě delšího úseku probíhal update frameworku na další verzi.

**Tabulka 4.6** Přehled výsledků od 5. března prvního online experimentu s DoH.

| | MCC | | | Accuracy | | |
|---|---|---|---|---|---|---|
| experiment_id | stř. hodnota | max | std | stř. hodnota | max | std |
| committee-soft_n_50 | 0.803 | 0.946 | 0.080 | 0.906 | 0.977 | 0.0417 |
| entropy_n_50 | 0.870 | 0.934 | 0.040 | 0.938 | 0.971 | 0.0204 |
| kl-divergence_n_50 | 0.664 | 0.911 | 0.155 | 0.822 | 0.960 | 0.1005 |
| random_n_50 | 0.847 | 0.921 | 0.038 | 0.927 | 0.963 | 0.0195 |

**Tabulka 4.7** Přehled jednotlivých instancí pro druhý experiment.

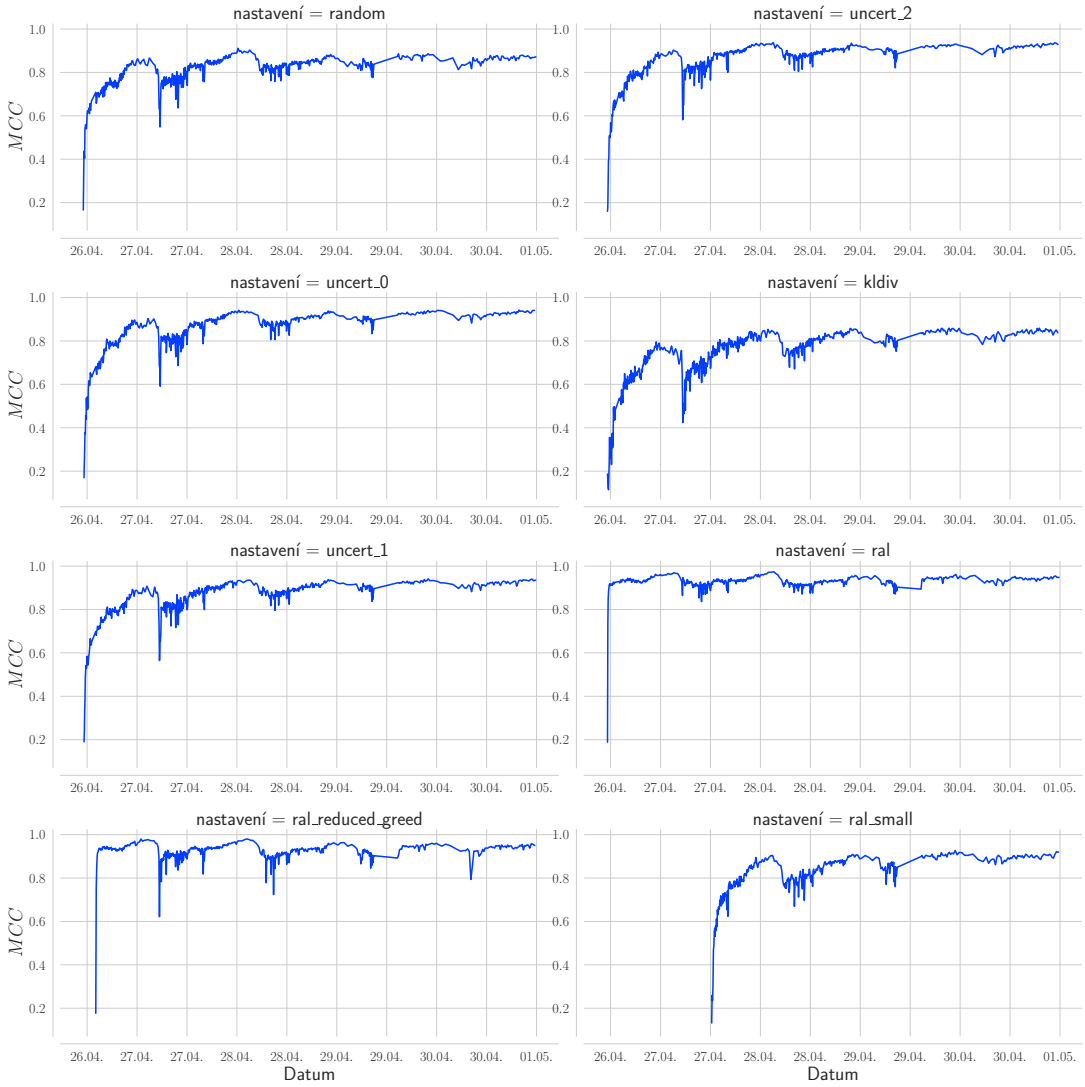
| id experimentu | $\mathcal{Q}$ |
|---|---|
| random | náhodný výběr |
| uncert_0 | práh nejistoty není |
| uncert_1 | práh nejistoty 0.9 |
| uncert_2 | práh nejistoty 0.8 |
| kldiv | KL divergence |
| ral | RAL výchozí |
| ral_reduced_greed | detto s redukcí náhody na 0.0001 |
| ral_small | detto s redukcí budgetu na 0.0002 |



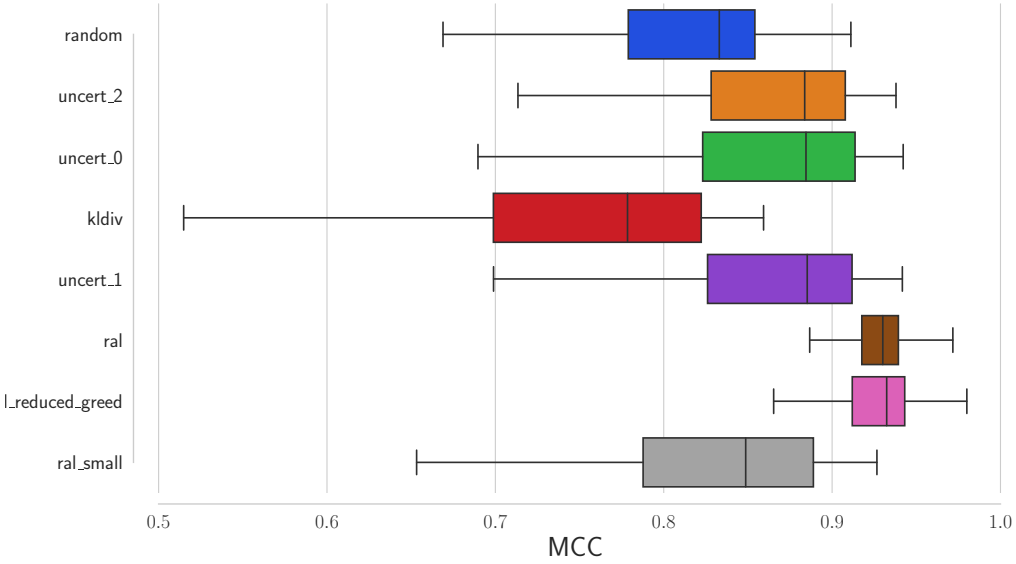

**Obrázek 4.11** Boxplot výsledků od 5. března prvního online experimentu s DoH.

**Tabulka 4.8** Výsledky druhého online experimentu s DoH. Hodnoty jsou agregovány z období od 28. dubna, tedy po 2 dnech spuštění experimentu. Je to kvůli tomu, abychom mohli poctivě porovnat i strategii `ral_small`, která byla spuštěna o den déle než ostatní.

| | MCC | | | Accuracy | | |
|---|---|---|---|---|---|---|
| experiment_id | mean | max | std | mean | max | std |
| kldiv | 0.826 | 0.859 | 0.024 | 0.918 | 0.942 | 0.013 |
| ral | 0.935 | 0.961 | 0.016 | 0.969 | 0.982 | 0.009 |
| ral_reduced_greed | 0.931 | 0.963 | 0.026 | 0.966 | 0.984 | 0.014 |
| ral_small | 0.884 | 0.927 | 0.031 | 0.944 | 0.966 | 0.016 |
| random | 0.855 | 0.886 | 0.022 | 0.930 | 0.947 | 0.012 |
| uncert_0 | 0.917 | 0.942 | 0.019 | 0.960 | 0.975 | 0.010 |
| uncert_1 | 0.916 | 0.942 | 0.018 | 0.960 | 0.973 | 0.009 |
| uncert_2 | 0.911 | 0.937 | 0.018 | 0.958 | 0.970 | 0.009 |



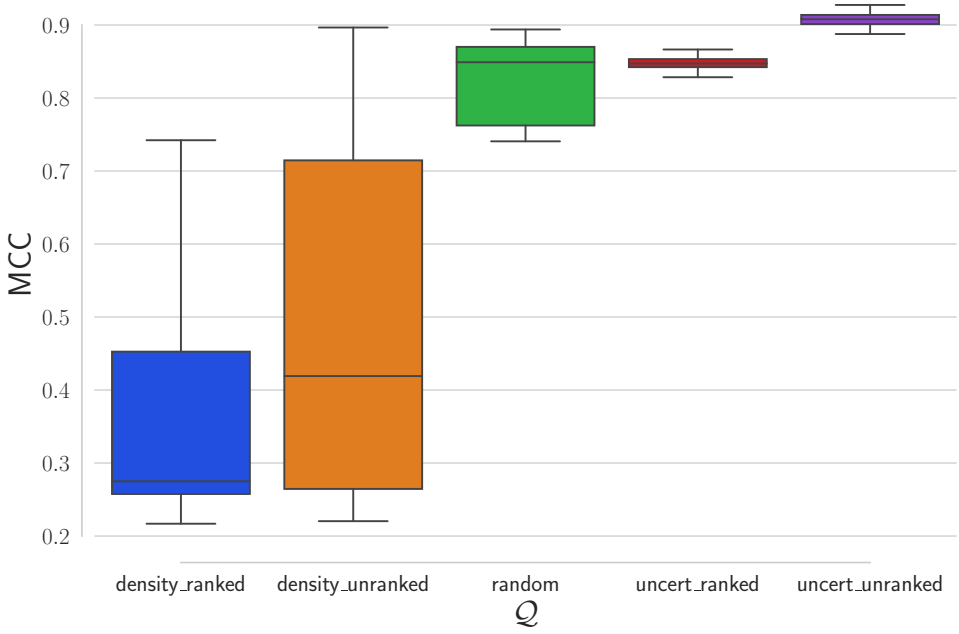

**Obrázek 4.12** Přehled průběhů druhé sady experimentů během jednoho týdne. V tomto již žádný výpadek není zaznamenán ani simulován.



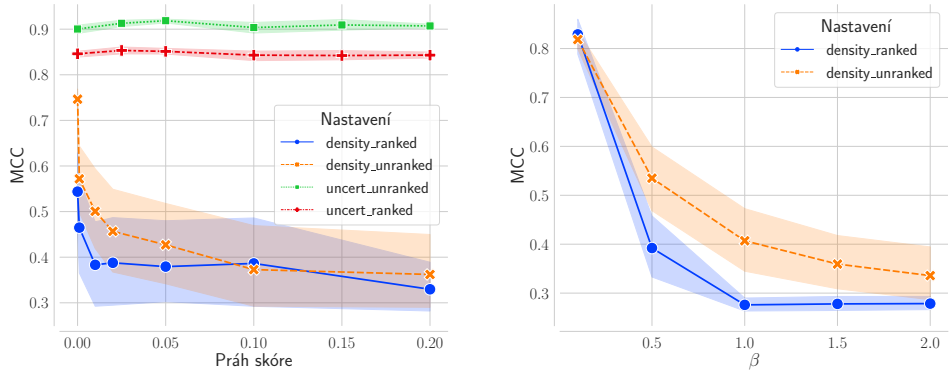

**Obrázek 4.13** Boxplot výsledků druhého experimentu.

**Tabulka 4.9** Seznam příznaků používaný v experimentech s cryptominery. Kurzívou jsou vyznačeny ty, které jsou dopočítány v rámci předzpracování.

| Jméno příznaku | Popis |
|---|---|
| bytes | Celková velikost odchozích paketů |
| bytes_rev | Celková velikost příchozích paketů |
| packets | Celkový počet odchozích paketů |
| packets_rev | Celkový počet příchozích paketů |
| *sent_percentage* | *Poměr odchozích paketů* |
| *recv_percentage* | *Poměr příchozích paketů* |
| *is_request_response* | *True jestliže poměry jsou stejné* |
| *avg_pkt_len* | *Průměrná délka paketů* |
| *avg_secs_between_pkts* | *Průměrný čas mezi pakety* |
| *overall_duration_in_sec* | *Celková délka spojení* |
| *psh_ratio* | *Poměr paketů s TCP flag PUSH* |



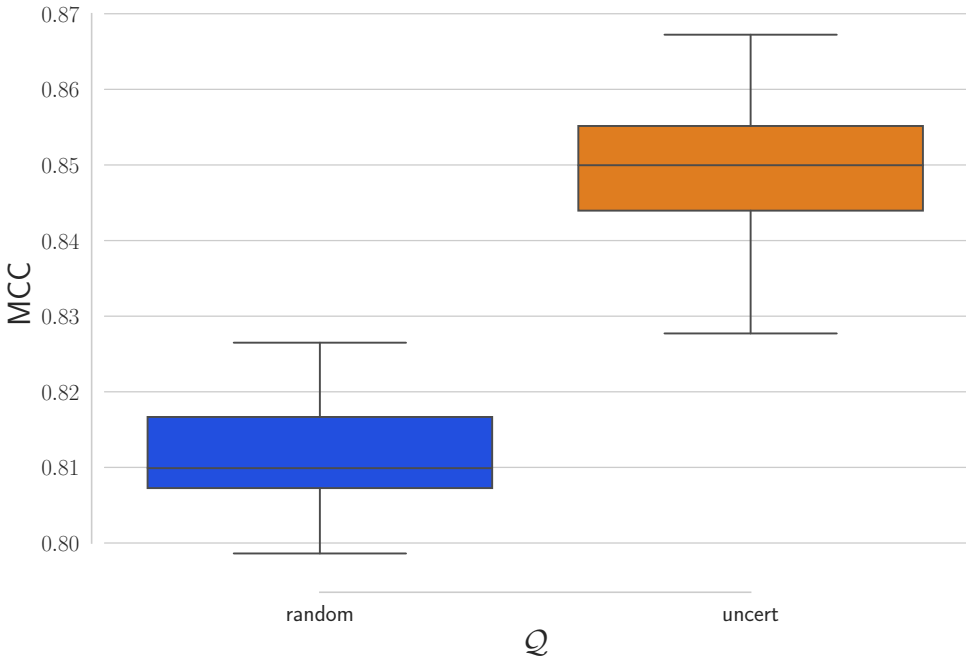

**Obrázek 4.14** Boxplot všech výsledků první sady experimentů s cryptominery. Rozdíl oproti DoH je, že strategie zahrnující podobnost zcela nepochybně zaostávají za `uncert_unranked` i `random`.




**(a)** Prahy skóre na `uncert` strategie nemají vliv, na `density` strategie mají negativní vliv.

**(b)** Stejně jako v případě DoH, parametr $\beta$ má na výkonnost systému negativní vliv.

**Obrázek 4.15** Vliv parametrů na výkonnost systému v první sadě experimentů s cryptominery. Výsledky jsou v souhlasu s pozorováním u *offline* experimentů s DoH.




**Obrázek 4.16** Porovnání dvou strategií využité v druhém experimentu s cryptominery. Lze vidět, že kde výkon náhodné strategie končí, tam `uncert` začíná. Podotýkáme, že `uncert` běžel v konfiguraci nehodnoceného výběru.

# Závěr

Diplomová práce se věnovala analýze možností toho, jak pomocí aktualizací datových sad zlepšovat klasifikaci síťového provozu s využitím algoritmů strojového učení. Dospěli jsme k tomu, že nejvhodnější metoda, jak toho dosáhnout, je takzvané aktivní učení. Ukazuje se, že tato metoda je vhodná nejen pro offline trénování modelů pomocí efektivního vybírání vzorků k anotaci (například i pomocí lidského experta), ale lze ji efektivně využít i online v reálném provozu. Napříč celou prací jsme se zabývali především proudovým zpracováváním, což je přirozený způsob, jakým pracovat nad síťovými toky; těch je mnoho a snaha ukládat je do zásoby a pak využívat k pokročilejším metodám je jednoznačně režijně nemoudrá. Stejně tak jsme se nezabývali metodami aktivního učení, které obnáší velký výpočetní výkon (například metody vyžadující minimálně lineárně rostoucí počet nutných tréninků s počtem vzorků v jedné dávce), neboť tím se poněkud vytrácí smysl používání takových metod.

Těžiště práce pak leželo v návrhu prototypu softwarového frameworku, který nejen využívá tyto metody, ale zároveň je dostatečně univerzální a rozšiřitelný. Motivací pro vznik takového nástroje je to, že takový nástroj s uspokojivými vlastnostmi prozatím neexistoval. Při návrhu se používaly osvědčené metody pro softwarový vývoj, jako je objektově orientované programování a používání návrhových vzorů. V průběhu celé práce jsme pak dokazovali, že takový návrh s vytyčenými požadavky je provozuschopný a funkční.

Poslední kapitola zabývající se experimenty měla vlastně dvojí funkci, a to:

1. ověření funkčnosti návrhu a implementace softwarového prototypu a
2. výzkum vzorkovacích strategií.

Díky důkladné analýze a návrhu se během implementace a testování neobjevily žádné zásadní problémy. Můžeme tedy konstatovat, že první bod byl naplněn a že všechny vytyčené požadavky, funkční i nefunkční, se podařilo splnit. Velkou zásluhu na tom má i systém NEMEA, který spoustu věcí usnadnil, především sběr dat a online nasazení.

Druhý bod vznikl víceméně jako vedlejší produkt práce. Již během analýzy jsme došli k závěru, že prezentované vzorkovací heuristiky a strategie nejsou zcela uspokojivě experimentálně ověřeny, a zvláště pak v oboru síťového monitoringu nad síťovými toky. I to je přínos naší práce. Prokázali jsme jednoznačnou převahu mnoha strategií nad náhodnou strategií. Obvykle (odkazujeme se na přehledový článek [6]) vychází randomizovaná strategie velmi podobně dobrá jako ostatní. Ve všech našich experimentech se ukázalo, že ačkoliv randomizovaná strategie funguje velmi dobře, je snadno překonána. To je velmi silný argument pro další výzkum v této oblasti. Další přínos této práce je předvedení, že práh skóre u strategií založených na nejistotě nehraje roli a v některých případech dokonce škodí. To je přímo v rozporu s doporučením ve velké přehledové knize Settlese [11], který u proudového zpracování doporučuje práh nastavovat. Poslední zásadní věc, jež z experimentů vyplývá je zavržení metod, které jsou založeny na vzájemných podobnostech toků. Tyto metody buď nepřinášely žádné výhody, nebo dokonce škodily. To je

nakonec veskrze pozitivní zjištění, protože tyto metody jsou výpočetně náročné a to, že málo náročná metoda převyšuje tyto náročné metody, nese pozitivní konotaci. K online experimentům jsme pak ještě lehce nad rozsah práce implementovali strategii, která nese prvky reinforcement learningu a vhodně kombinuje užitečné prvky úspěšných strategií.

Lze tedy prohlásit, že požadavky kladené na tuto práci byly splněny a že práce rámec požadavků místy přesahuje.

Pokračování této práce by se mohlo nést právě dalším výzkumem v oblasti vzorkovacích strategií, protože se ukazuje, že dobrá volba vzorkovací strategie je mnohem zásadnější, než plyne ze state-of-the-art článků a prací — minimálně v oblasti síťového monitoringu. Těžištěm a hlavním produktem této práce je softwarový prototyp. Ten umožňuje jednoduše implementovat další strategie a další modely a snadno umožňuje přidat další modifikace. Je schopen běžet a je implementován jako modul do systému NEMEA. Další možné pokračování práce je právě na úrovni softwarově-inženýrské. Je zde záměrně uvolněný prostor pro implementaci paralelismu. Stejně tak náš produkt přirozeně vybízí k fungování jako distribuovaný systém. V současné době se plánuje nasazení vyvinutého frameworku v síti CESNET.

# Příloha A
# Seznam zkratek

**`UniRec`** Unified Record

**API** Application Programming Interface

**CESNET** Czech Education and Scientific NETwork

**CSV** Comma-separated values

**DNS** Domain Name System

**DoH** DNS over HTTPS

**HTTPS** Hypertext Transfer Protocol Secure

**IETF** Internet Engineering Task Force

**IFC_SPEC** TRAP Interface Specifier

**IPFIX** Internet Protocol Flow Information Export

**JSON** JavaScript Object Notation

**NEMEA** Network Measurements Analysis

**P2P** Peer-to-peer

**QoS** Quality of Service

**RAL** Reinforcement Active Learning

**SQL** Structured Query Language

**SVM** Support Vector Machine

**TSNE** T-distributed Stochastic Neighbor Embedding

**UML** Unified Modeling Language

**VPN** Virtual Private Network

# Příloha B

# Příklad sestavení

Již v práci v kapitole věnující se implementaci jsme ukázali, jak vypadá sestavený software z našeho vyvinutého frameworku. V této části si jej podrobně rozebereme.

Jedná se o klasifikátor DoH využívající míru nejistoty ve verzi nehodnoceného výběru. Pro jednoduchost je vypuštěna parametrizace od případného uživatele a konstanty jsou *hardcoded*.

```
# knihovny potřebné pro logging
import logging
import sys

# jako prediktivní model využijeme náhodný strom z knihovny sklearn
from sklearn.ensemble import RandomForestClassifier

# import součástí frameworku
import alf.anotator
import alf.context_manager
import alf.d_manager
import alf.engine
import alf.evaluator
import alf.input_manager
import alf.ml_model
import alf.postprocess
import alf.preprocess
import alf.query_strategy
```

Framework hustě používá knihovnu `logging` k logování. Je vhodné si logování nakonfigurovat, například do standardního výstupu se zaznamenaným časem.

```
logging.basicConfig(
    stream=sys.stdout,
    format='[%(asctime)s]: %(message)s',
    level=logging.DEBUG
)
```

Následuje nastavení konstant a parametrů. Ty je obvyklé nechávat nastavovat uživatele z příkazové řádky nebo z konfiguračního souboru.

```
# seznam příznaků, typově se jedná o list[str];
# odpovídají jednotlivým sloupcům datových toků
DATASET_COLUMNS = ["f1", "f2", ..]

# vstupní rozhraní IFC_SPEC jak definuje
# framework NEMEA
IFC = "u:alf_socket"

# id, workdir; id by mělo být v rámci
# možností pro experiment unikátní
EXP_ID = "showcase"
WORKDIR = "/tmp/alf"

# konfigurace specifická pro anotátor
BLACKLIST = "conf/blacklist.txt"

# D databáze s definovanou maximální velikostí
D0 = "conf/doh_train_db_small.csv"
MAX_SIZE = 5000

# konfigurace specifická pro strategii
N = 10
THRESHOLD = 0.1
```

Po definici konstant vytvoříme kontexty. Jednak kontext pro $\mathcal{D}$ (trénovací databázi), jednak kontext pro ukládání metrik, modelů a databází.

```
# nastavíme kontext včetně používaného vektoru příznaků
ContextProvider.create_context("file") # ukládáme metriky do souboru
ContextProvider.get_context().set_features(DATASET_COLUMNS)
ContextProvider.get_context().set_experiment_id(EXP_ID)
ContextProvider.get_context().set_working_dir(WORKDIR)

# kontext pro trénovací databázi
DbProvider.create_context(context_type="file", d_0_path=D0)
```

Nyní zbývá definovat jednotlivé součásti.

```
anotator = alf.anotator.AnotatorDoH(blacklist_path=BLACKLIST)
model = alf.ml_model.SupervisedMLModel(RandomForestClassifier())
query_strategy = alf.query_strategy.UncertanityUnrankedBatch(
        anotator_obj=anotator, max_samples=N,
        score_threshold=THRESHOLD, dry_run=True)
input_manager = alf.input_manager.TrapcapSocketInputManager(
        definition=IFC)
postprocessor = alf.postprocess.PostprocessorUndersample(MAX_SIZE)
```

Součásti předáme modulu `Engine`.

```
engine = alf.engine.Engine(
    preprocessor=alf.preprocess.PreprocessorDoH(),
```

```
    postprocessor=postprocessor,
    ml_model_obj=model,
    query_strategy_obj=query_strategy,
    evaluator_obj=alf.evaluator.EvaluatorTestAnotatedAndAllPredicted(),
    input_manager_obj=input_manager
)
```

A celý systém musíme spustit metodou `run`.

```
engine.run()
```

# Příloha C

# Dokumentace a testování

V této části se budeme pohybovat ve složce `impl`. Kromě balíčků, které se instalují standardní cestou (viz `make init`, předpokládáme prerekvizitu NEMEA[1]). Vyvíjený produkt využívá dokumentaci generovanou ze zdrojových kódů. K tomu se využívá knihovna pro `Python`, která se jmenuje `Sphinx`. Kód (například třídy nebo funkce) jsou dokumentovány takzvaným `docstring`, což je rodina standardizovaných formátů pro dokumentování kódu. Naše práce využívá formát `Google styleguide`[2]. Dokumentace se na médiu nachází již vygenerovaná. Pro úplnost, generování probíhá následujícím způsobem zavoláním připraveného skriptu:

```
$> ./scripts/generate_docs.sh
```

To vygeneruje kostru dokumentace i s obsahem. Dokumentace se pak nachází ve formátu `HTML` v index souboru `docs/_build/html/index.html` V případě, že se v kódu provedou nějaké změny, již není potřeba generovat celou kostru, ale stačí zavolat příkaz `make html` ve složce `./docs`. Následuje popis jednotlivých pravidel v přiloženém `Makefile` ve složce `impl`:

- `make init` nainstaluje závislosti potřebné k fungování i k vývoji (například linter `flake8` nebo `pytest`)
- `make test` spustí jednotkové testy definované ve složce `tests`
- `make report` vygeneruje `HTML` soubor s výsledky testů a vloží je na hlavní stránku dokumentace
- `make lint` spustí statickou analýzu kódu pomocí programu `flake8`

K dalšímu testování a provozu je vhodný nástroj z projektu NEMEA `traffic_repeater`. Funguje jako opakovač. Na vstupu dostane vstupní a výstupní komunikační rozhraní a pouze přeposílá zprávy. Příklad použití může být následující:

```
$> /usr/bin/nemea/traffic_repeater -i "f:example.trapcap,u:alf_socket"
```

Stane se to, že síťové toky uložené v souboru `example.trapcap` se přepošlou na socket `alf_socket`, kde si je může odebrat náš framework.

Minimální příklad, jak ukázat provoz frameworku je potom takový, že v jednom terminálu spustíme instanci našeho produktu (zde sestavení pro klasifikaci DoH):

[1] Návod na instalaci zde: https://github.com/CESNET/Nemea-Framework

[2] https://github.com/google/styleguide/blob/gh-pages/pyguide.md#38-comments-and-docstrings

```
$> mkdir -p /tmp/alf
$> WORKDIR=/tmp/alf
$> python nemea_module_doh.py           \
    --i u:alf_socket                    \
    --id test_random                    \
    --workdir $WORKDIR                  \
    --model single                      \
    --query_strategy random             \
    --blacklist conf/blacklist.txt      \
    --dpath conf/doh_train_db_small.csv \
    --query_nmax 1                      \
    --max_db_size 10000
```

A v terminálu vedle se spustí zmiňovaný `traffic_repeater`. Výsledky se nacházejí v `$WORKDIR`, což je v tomto případě `/tmp/alf`. Tato složka by neměla obsahovat soubory vygenerované jinou instancí se stejným `id`, protože pak se je framework bude snažit využít, což nemusí být vždy zamýšlené chování a je s tím třeba počítat.

# Příloha D

# Obsah přiloženého média

- `readme.txt` ..... stručný popis obsahu média
- `impl`
  - `alf` ..... zdrojové kódy implementace
  - `conf` ..... konfigurační soubory
  - `docs` ..... dokumentace
  - `scripts` ..... pomocné skripty
  - `results` ..... výsledky experimentů
  - `experiments` ..... zpracování výsledků ve formě Jupyter Notebooků, obrázků, atd.
  - `Makefile`
  - `nemea_module_doh.py` ..... NEMEA modul pro DoH
  - `doh_experiment.py` ..... skript využívaný pro DoH experimenty
  - `miners_experiment.py` ..... skript využívaný pro cryptominer experimenty
  - `classes.png` ..... kompletní UML diagram včetně dědičností
  - `requirements*.txt` ..... soubory definující závislosti
  - `example.trapcap` ..... příklad trapcap souboru
- `text` ..... text práce
  - `src` ..... zdrojová forma práce ve formátu LuaLaTeX
  - `thesis.pdf` ..... text práce ve formátu PDF

# Bibliografie


1. CLAISE, B.; TRAMMELL, B.; AITKEN, P. *Specification of the IP Flow Information Export (IPFIX) Protocol for the Exchange of Flow Information* [online]. 2013-09 [cit. 2022-03-14]. RFC7011. RFC Editor. Dostupné z DOI: 10.17487/rfc7011.
2. ČEJKA, Tomas; BARTOŠ, Václav; SVEPES, Marek; ROSA, Zdeněk; KUBÁTOVÁ, Hana. NEMEA: A Framework for Network Traffic Analysis. In: *2016 12th International Conference on Network and Service Management (CNSM)* [online]. Montreal, QC, Canada: IEEE, 2016, s. 195–201 [cit. 2022-03-14]. Dostupné z DOI: 10.1109/CNSM.2016.7818417.
3. VÁCLAV BARTOŠ; TOMÁŠ ČEJKA; MARTIN ŽÁDNÍK. *Nemea: Framework for Stream-Wise Analysis of Network Traffic* [online]. 2013-09 [cit. 2022-04-13]. 9/2013. CESNET. Dostupné z: https://www.cesnet.cz/wp-content/uploads/2014/02/trapnemea.pdf.
4. WANG, Pan; CHEN, Xuejiao; YE, Feng; SUN, Zhixin. A Survey of Techniques for Mobile Service Encrypted Traffic Classification Using Deep Learning. *IEEE Access* [online]. 2019, roč. 7, s. 54024–54033 [cit. 2022-03-14]. ISSN 2169-3536. Dostupné z DOI: 10.1109/ACCESS.2019.2912896.
5. VEKSHIN, Dmitrii; HYNEK, Karel; CEJKA, Tomas. DoH Insight: Detecting DNS over HTTPS by Machine Learning. In: *Proceedings of the 15th International Conference on Availability, Reliability and Security* [online]. Virtual Event Ireland: ACM, 2020, s. 1–8 [cit. 2022-03-14]. ISBN 978-1-4503-8833-7. Dostupné z DOI: 10.1145/3407023.3409192.
6. SHAHRAKI, Amin; ABBASI, Mahmoud; TAHERKORDI, Amir; JURCUT, Anca Delia. Active Learning for Network Traffic Classification: A Technical Study. *IEEE Transactions on Cognitive Communications and Networking* [online]. 2022, roč. 8, č. 1, s. 422–439 [cit. 2022-03-14]. ISSN 2332-7731, ISSN 2372-2045. Dostupné z DOI: 10.1109/TCCN.2021.3119062.
7. VAŠATA, Daniel. *Přednáška NI-ADM: Evaluace Modelů* [online]. 2022 [cit. 2022-03-14]. Dostupné z: https://courses.fit.cvut.cz/NI-ADM/lectures/files/NI-ADM-02-en-slides.pdf.
8. SOKOLOVA, Marina; LAPALME, Guy. A Systematic Analysis of Performance Measures for Classification Tasks. *Information Processing & Management* [online]. 2009, vol. 45, no. 4, s. 427–437 [cit. 2022-03-14]. ISSN 03064573. Dostupné z DOI: 10.1016/j.ipm.2009.03.002.
9. PEDREGOSA, Fabian; VAROQUAUX, Gaël; GRAMFORT, Alexandre; MICHEL, Vincent; THIRION, Bertrand; GRISEL, Olivier; BLONDEL, Mathieu; MÜLLER, Andreas; NOTHMAN, Joel; LOUPPE, Gilles; PRETTENHOFER, Peter; WEISS, Ron; DUBOURG, Vincent; VANDERPLAS, Jake; PASSOS, Alexandre; COURNAPEAU, David; BRUCHER,

Matthieu; PERROT, Matthieu; DUCHESNAY, Édouard. *Scikit-Learn: Machine Learning in Python* [online]. 2018 [cit. 2022-03-14]. Dostupné z arXiv: 1201.0490 [cs].

10. CHICCO, Davide; STAROVOITOV, Valery; JURMAN, Giuseppe. The Benefits of the Matthews Correlation Coefficient (MCC) Over the Diagnostic Odds Ratio (DOR) in Binary Classification Assessment. *IEEE Access* [online]. 2021, roč. 9, s. 47112–47124 [cit. 2022-03-14]. ISSN 2169-3536. Dostupné z DOI: 10.1109/ACCESS.2021.3068614.
11. SETTLES, Burr. Active Learning. *Synthesis Lectures on Artificial Intelligence and Machine Learning* [online]. 2012, vol. 6, no. 1, s. 1–114 [cit. 2022-03-14]. ISSN 1939-4608, ISSN 1939-4616. Dostupné z DOI: 10.2200/S00429ED1V01Y201207AIM018.
12. SEGARAN, Toby. *Programming Collective Intelligence: Building Smart Web 2.0 Applications*. 1st ed. Beijing ; Sebastapol [CA]: O'Reilly, 2007. ISBN 978-0-596-52932-1.
13. CARDOSO, Thiago N.C.; SILVA, Rodrigo M.; CANUTO, Sérgio; MORO, Mirella M.; GONÇALVES, Marcos A. Ranked Batch-Mode Active Learning. *Information Sciences* [online]. 2017, vol. 379, s. 313–337 [cit. 2022-03-17]. ISSN 00200255. Dostupné z DOI: 10.1016/j.ins.2016.10.037.
14. KELLEHER, John D.; MAC NAMEE, Brian; D'ARCY, Aoife. *Fundamentals of Machine Learning for Predictive Data Analytics: Algorithms, Worked Examples, and Case Studies*. Second edition. Cambridge, Massachusetts: The MIT Press, 2020. ISBN 978-0-262-04469-1.
15. KULLBACK, S.; LEIBLER, R. A. On Information and Sufficiency. *The Annals of Mathematical Statistics* [online]. 1951, vol. 22, no. 1, s. 79–86 [cit. 2022-03-15]. ISSN 0003-4851. Dostupné z DOI: 10.1214/aoms/1177729694.
16. WASSERMANN, Sarah; CUVELIER, Thibaut; CASAS, Pedro. RAL - Improving Stream-Based Active Learning by Reinforcement Learning. In: *European Conference on Machine Learning and Principles and Practice of Knowledge Discovery in Databases (ECML-PKDD) Workshop on Interactive Adaptive Learning (IAL)* [online]. Würzburg, Germany, 2019 [cit. 2022-03-30]. Dostupné z: https://hal.archives-ouvertes.fr/hal-02265426.
17. WASSERMANN, Sarah; CUVELIER, Thibaut; MULINKA, Pavol; CASAS, Pedro. ADAM & RAL: Adaptive Memory Learning and Reinforcement Active Learning for Network Monitoring. In: *2019 15th International Conference on Network and Service Management (CNSM)* [online]. Halifax, NS, Canada: IEEE, 2019, s. 1–9 [cit. 2022-03-29]. ISBN 978-3-903176-24-9. Dostupné z DOI: 10.23919/CNSM46954.2019.9012675.
18. BOUNEFFOUF, Djallel; LAROCHE, Romain; URVOY, Tanguy; FERAUD, Raphael; ALLESIARDO, Robin. Contextual Bandit for Active Learning: Active Thompson Sampling. In: LOO, Chu Kiong; YAP, Keem Siah; WONG, Kok Wai; TEOH, Andrew; HUANG, Kaizhu (ed.). *Neural Information Processing* [online]. Cham: Springer International Publishing, 2014, sv. 8834, s. 405–412 [cit. 2022-03-29]. Lecture Notes in Computer Science. ISBN 978-3-319-12636-4 978-3-319-12637-1. Dostupné z DOI: 10.1007/978-3-319-12637-1_51.
19. GAO, Mingfei; ZHANG, Zizhao; YU, Guo; ARIK, Sercan O.; DAVIS, Larry S.; PFISTER, Tomas. *Consistency-Based Semi-supervised Active Learning: Towards Minimizing Labeling Cost* [online]. 2020 [cit. 2022-03-29]. Dostupné z arXiv: 1910.07153 [cs].
20. HUANG, Sheng-Jun; JIN, Rong; ZHOU, Zhi-Hua. Active Learning by Querying Informative and Representative Examples. *IEEE Transactions on Pattern Analysis and Machine Intelligence* [online]. 2014, roč. 36, č. 10, s. 1936–1949 [cit. 2022-03-29]. ISSN 0162-8828, ISSN 2160-9292. Dostupné z DOI: 10.1109/TPAMI.2014.2307881.
21. WEI-NING HSU; HSUAN-TIEN LIN. Active Learning by Learning. In: 2015. Dostupné také z: https://www.aaai.org/ocs/index.php/AAAI/AAAI15/paper/view/9636/9924.

22. XIONG, Hui; ELECTRICAL AND ELECTRONICS ENGINEERS, Institute of; SOCIETY, IEEE Computer (eds.). *2013 IEEE 13th International Conference on Data Mining (ICDM 2013): Dallas, Texas, USA, 7 - 10 December 2013 ; [Proceedings]*. Piscataway, NJ: IEEE, 2013. ISBN 978-0-7695-5108-1 978-1-4799-1328-2.

23. KONYUSHKOVA, Ksenia; SZNITMAN, Raphael; FUA, Pascal. *Learning Active Learning from Data* [online]. 2017 [cit. 2022-03-14]. Dostupné z arXiv: 1703.03365 [cs].

24. KONYUSHKOVA, Ksenia; SZNITMAN, Raphael; FUA, Pascal. *Discovering General-Purpose Active Learning Strategies* [online]. 2019 [cit. 2022-03-29]. Dostupné z arXiv: 1810.04114 [cs, stat].

25. ZHU, Xingquan; ZHANG, Peng; LIN, Xiaodong; SHI, Yong. Active Learning from Data Streams. In: *Seventh IEEE International Conference on Data Mining (ICDM 2007)* [online]. Omaha, NE, USA: IEEE, 2007, s. 757–762 [cit. 2022-03-29]. ISBN 978-0-7695-3018-5. Dostupné z DOI: 10.1109/ICDM.2007.101.

26. WOODWARD, Mark; FINN, Chelsea. *Active One-shot Learning* [online]. 2017 [cit. 2022-03-29]. Dostupné z arXiv: 1702.06559 [cs].

27. KRUEGER, David; LEIKE, Jan; EVANS, Owain; SALVATIER, John. *Active Reinforcement Learning: Observing Rewards at a Cost* [online]. 2020 [cit. 2022-03-29]. Dostupné z arXiv: 2011.06709 [cs, stat].

28. YIN, Lili; WANG, Huangang; FAN, Wenhui. Active Learning Based Support Vector Data Description Method for Robust Novelty Detection. *Knowledge-Based Systems* [online]. 2018, vol. 153, s. 40–52 [cit. 2022-03-29]. ISSN 09507051. Dostupné z DOI: 10.1016/j.knosys.2018.04.020.

29. LIU, San-Min; SUN, Zhi-Xin. Active Learning for P2P Traffic Identification. *Peer-to-Peer Networking and Applications* [online]. 2015, vol. 8, no. 5, s. 733–740 [cit. 2022-03-29]. ISSN 1936-6442, ISSN 1936-6450. Dostupné z DOI: 10.1007/s12083-014-0281-3.

30. YIN, Lili; WANG, Huangang; FAN, Wenhui; KOU, Li; LIN, Tingyu; XIAO, Yingying. Incorporate Active Learning to Semi-Supervised Industrial Fault Classification. *Journal of Process Control* [online]. 2019, vol. 78, s. 88–97 [cit. 2022-03-29]. ISSN 09591524. Dostupné z DOI: 10.1016/j.jprocont.2019.04.008.

31. *NEMEA System* [online]. 2022 [cit. 2022-03-31]. Dostupné z: https://github.com/CESNET/Nemea.

32. PREE, Wolfgang. Meta Patterns — A Means for Capturing the Essentials of Reusable Object-Oriented Design. In: TOKORO, Mario; PARESCHI, Remo (eds.). *Object-Oriented Programming* [online]. Berlin/Heidelberg: Springer-Verlag, 1994, vol. 821, s. 150–162 [cit. 2022-03-21]. Lecture Notes in Computer Science. ISBN 978-3-540-58202-1. Dostupné z DOI: 10.1007/BFb0052181.

33. BUSCHMANN, Frank (ed.). *Pattern-Oriented Software Architecture: A System of Patterns*. Chichester ; New York: Wiley, 1996. ISBN 978-0-471-95869-7.

34. GEVORKYAN, Migran; DEMIDOVA, Anastasia; DEMIDOVA, Tatiana; SOBOLEV, Anton. Review and Comparative Analysis of Machine Learning Libraries for Machine Learning. *Discrete and Continuous Models and Applied Computational Science*. 2019, roč. 27, s. 305–315. Dostupné z DOI: 10.22363/2658-4670-2019-27-4-305-315.

35. NGUYEN, Giang; DLUGOLINSKY, Stefan; BOBÁK, Martin; TRAN, Viet; LÓPEZ GARCÍA, Álvaro; HEREDIA, Ignacio; MALÍK, Peter; HLUCHÝ, Ladislav. Machine Learning and Deep Learning Frameworks and Libraries for Large-Scale Data Mining: A Survey. *Artificial Intelligence Review* [online]. 2019, vol. 52, no. 1, s. 77–124 [cit. 2022-03-30]. ISSN 1573-7462. Dostupné z DOI: 10.1007/s10462-018-09679-z.

36. TIVADAR DANKA; PETER HORVATH. *modAL: A Modular Active Learning Framework for Python*. [B.r.]. Dostupné také z: https://github.com/modAL-python/modAL.
37. NUAA-AL. *ALiPy: Active Learning in Python* [online]. 2022 [cit. 2022-03-30]. Dostupné z: https://github.com/NUAA-AL/ALiPy.
38. *Active Learning Playground* [online]. 2022 [cit. 2022-03-30]. Dostupné z: https://github.com/google/active-learning.
39. HUANG, Kuan-Hao. *DeepAL: Deep Active Learning in Python* [online]. 2022 [cit. 2022-03-30]. Dostupné z: https://github.com/ej0cl6/deep-active-learning.
40. MONARCH, Robert (Munro). *PyTorch Active Learning* [online]. 2022 [cit. 2022-03-30]. Dostupné z: https://github.com/rmunro/pytorch_active_learning.
41. *Libact: Pool-based Active Learning in Python* [online]. 2022 [cit. 2022-03-30]. Dostupné z: https://github.com/ntucllab/libact.
42. *AlpacaTag* [online]. 2022 [cit. 2022-03-30]. Dostupné z: https://github.com/INK-USC/AlpacaTag.
43. *API Reference* [online] [cit. 2022-04-14]. Dostupné z: https://scikit-learn/stable/modules/classes.html.
44. *Sklearn.Ensemble.VotingClassifier* [online] [cit. 2022-04-14]. Dostupné z: https://scikit-learn/stable/modules/generated/sklearn.ensemble.VotingClassifier.html.
45. *Nemea Framework* [online]. 2022 [cit. 2022-04-14]. Dostupné z: https://github.com/CESNET/Nemea-Framework/blob/80b7cde8d54bb185273a5e6e7de59f9009ca32b1/libtrap/README.ifcspec.md.
46. *Nemea-Framework/Pytrap at Master · CESNET/Nemea-Framework* [online] [cit. 2022-04-14]. Dostupné z: https://github.com/CESNET/Nemea-Framework.
47. HOFFMAN, Paul E.; MCMANUS, Patrick. *DNS Queries over HTTPS (DoH)* [online]. 2018-10 [cit. 2022-04-11]. Request for Comments, RFC 8484. Internet Engineering Task Force. Dostupné z DOI: 10.17487/RFC8484.
48. *An Analysis of Godlua Backdoor* [online]. 2019-07-01 [cit. 2022-04-27]. Dostupné z: https://blog.netlab.360.com/an-analysis-of-godlua-backdoor-en/.
49. *A Closer Look at Flubot's DoH Tunneling* [online]. 2022-01-14 [cit. 2022-04-27]. Dostupné z: https://blog.f-secure.com/flubot_doh_tunneling/.
50. *MariaDB Foundation* [online] [cit. 2022-04-28]. Dostupné z: https://mariadb.org/.
51. *Grafana: The Open Observability Platform* [online] [cit. 2022-04-28]. Dostupné z: https://grafana.com/.
52. *Stratum V1 Docs | Mining Protocol* [online] [cit. 2022-05-02]. Dostupné z: https://braiins.com/stratum-v1/docs.