FAKULTA INFORMAČNÍCH TECHNOLOGIÍ ČVUT V PRAZE

Zadání diplomové práce

Název:	Posouzení důležitosti zařízení podle chování na síti
Student:	Bc. Jan Rudolf
Vedoucí:	Ing. Michael Adam Polak
Studijní program:	Informatika
Obor / specializace:	Znalostní inženýrství
Katedra:	Katedra aplikované matematiky
Platnost zadání:	do konce letního semestru 2022/2023

Pokyny pro vypracování

Se vzestupem internetu věcí a obecně narůstajícím množstvím zařízení připojených k lokální síti je stále důležitější správně identifikovat jednotlivá zařízení a posoudit jejich důležitost pro správný chod společnosti, nebo i sítě jako takové.

Hlavním úkolem práce je prozkoumat možnosti identifikace kritických zařízení (uzlů) v lokální počítačové síti na základě poskytovaných služeb (např. DNS, LDAP, NFS) z dodaných záznamů síťové komunikace. Cílem diplomové práce pak je vybrat algoritmus strojového učení (případně algoritmy) vhodné pro klasifikaci poskytovaných služeb, odhad důležitosti zařízení na dané síti a případně posouzení škody způsobené výpadkem uzlu na základě skutečné síťové telemetrie.

1. Seznamte se s doménou síťové telemetrie a s problémy tohoto zdroje dat.
2. Prozkoumejte existující metody klasifikace služeb ze síťové telemetrie.
3. Prostudujte přístupy používané k obecné identifikaci (klasifikaci) zařízení na základě jeho chování v síti.
4. Navrhněte a implementujte algoritmy pro odhad důležitosti zařízení s využitím poznatků z bodů 2) a 3).
5. Proveďte učení a testování zvoleného přístupu nad omezeným vzorkem anonymizovaných dat, dodaných vedoucím práce.
6. Proveďte ověření přesnosti přístupu nad anonymizovanou síťovou telemetrií s ručně dodanou anotací.

Elektronicky schválil/a Ing. Magda Friedjungová, Ph.D. dne 29. června 2022 v Praze.

Diplomová práce

POSOUZENÍ DŮLEŽITOSTI ZAŘÍZENÍ PODLE CHOVÁNÍ NA SÍTI

Bc. Jan Rudolf

Fakulta informačních technologií
Katedra aplikované matematiky
Vedoucí: Ing. Michael Adam Polak
4. května 2023

Obsah

Seznam obrázků

Seznam tabulek

Seznam výpisů kódu

Děkuji vedoucímu práce Michaelovi Polakovi
za poskytnuté rady a trpělivost při vzniku této práce.

Dále děkuji rodině za neochvějnou podporu při studiu.

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 4. května 2023 .....

Abstrakt

Zvětšující se objem počítačových sítí přináší stále nové výzvy, a to nejen po hardwarových a softwarových stránkách. Udržet přehled v rozmanitých komplexních moderních sítích není snadné a automatická identifikace zařízení a rizik spojených s jejich výpadkem se stává kritickou pro správu sítě.

Tato práce se zabývá přístupy k automatické identifikaci zařízení na síti v kontextu služeb, které poskytuje na základě jejich chování. Dále poskytuje návrh přístupu k odhadu škod způsobených výpadkem zařízení.

Klíčová slova správa sítě, strojové učení, identifikace zařízení na síti,

Abstract

The increasing size of computer networks presents ongoing challenges, not only in terms of hardware and software. Keeping track of the diverse and complex modern networks is not easy, and automatically identifying devices and the risks associated with their failure is becoming critical for network management.

This research focuses on approaches to automatically identifying devices on a network in the context of the services they provide based on their behavior. It also proposes a method for estimating the damages caused by device outages.

Keywords network management, machine learning, network device identification

Úvod

V posledních letech jsme svědky strmého růstu objemu počítačových sítí a to jak po stránce komunikací tak i počtu zařízení. Odhaduje se, že v roce 2018 bylo v provozu 25 miliard po síti komunikujících zařízení a i do budoucna je třeba počítat s dalším nárůstem [1]. Zvětšující se sítě kladou stále větší nároky na hardware a software, ale ne jenom na ně. I správa sítě je čím dál více komplikovanější a v její doméně vzniká mnoho výzev. Jednou z nich je identifikace zařízení na síti. Automatizace této úlohy má veliký přínos k usnadnění monitoringu sítě, alokaci výpočetních zdrojů a odhadu důsledků výpadků zařízení. Často používané pravidlové systémy však mají problém držet krok se zvětšující se velikostí a komplexitou moderních sítí a proto je na místě vývoj nových, robustnějších přístupů [2].

V této práci se zabýváme návrhem přístupu k automatické identifikaci zařízení a služeb, které poskytuje na základě jeho chování. Pro identifikovaná zařízení navrhujeme algoritmus, který odhaduje potenciální rizika, která mohou vzniknou výpadkem zařízení na síti.

Naši úlohu plníme ve třech samostatných krocích:

V prvním kroku identifikujeme servery. Díky dostupnému anotovanému datasetu k tomu využíváme technik supervizovaného strojového učení. V druhém kroku se zaměřujeme na zařízení označené, jako servery a detekujeme služby, které poskytují. Ve třetím kroku modelujeme dopad hypotetického výpadku libovolných zařízení na chod zbytku sítě.

Tyto přístupy navrhujeme tak, aby obstáli speciálním výzvám, které přináší dataset zadavatele.

Kapitola čtenáře obeznamuje se základy použitých metod zejména strojového učení. Kapitola 1 popisuje předešlou práci, zabývající se přístupy k identifikaci zařízení. Kapitola 2 poskytuje detailní popis datasetu, pro který přístupy navrhujeme a popis výzev jež přináší. Kapitola 3 představuje námi navržené přístupy pro plnění jednotlivých kroků definované úlohy. Ty následně evaluujeme v Kapitole 4. Kapitola 5 diskutuje výsledky, jichž jsme dosáhli a pojednává o možnostech k jejich zlepšení.

Rešerše použitých metod

Metody strojového učení se těší stále se zvyšujícímu zájmu. Na rozdíl od standardních algoritmů, nevyžadují algoritmy strojového učení od vývojáře návrh exaktní algoritmus pro řešení problému. Místo toho se algoritmus snaží najít vhodné parametry modelu na základě množiny dat dané problematiky. Na vývojáře pak připadá zodpovědnost za předzpracování dat, volbu modelu strojového učení a zajištění vhodného učení/ladění parametrů modelu. Použití metod strojového učení je velmi výhodné, jestliže je návrh exaktního algoritmu náročný, nebo dokonce nemožný. Proto si našli své místo napříč obory jako jsou strojové vidění, počítačová bezpečnost, doporučování, detekce podvodu a mnoha dalších.

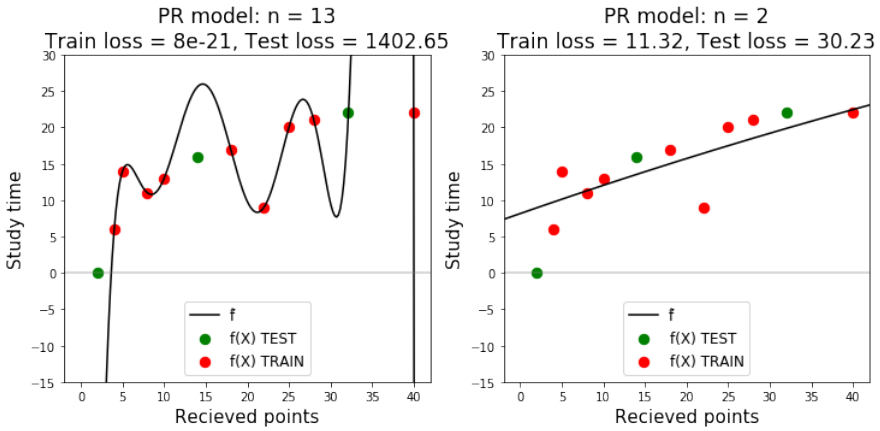
V této kapitole provádíme rešerši metod strojového učení použitých v dalších kapitolách. Jelikož se práce zabývá úlohou klasifikace, zaměřujeme se především na supervizované učení, které se touto problematikou zabývá.

Vztah modelu strojového učení formálně zapisujeme: $y = f(x)$, kde x je jeden vzorek dat, f model zobrazující vzorek na vysvětlovanou proměnnou y. Jestliže jsou k vzorkům k dispozici i vysvětlující proměnné, mluvíme o supervizovaném učení. Úlohy se v supervizovaném učení rozlišují podle typu vysvětlované proměnné: nabývá-li spojitých hodnot jedná se o regresi a pokud nabývá diskrétních hodnot, jedná se o klasifikaci. V případě že hodnoty vysvětlované proměnné známy nejsou, mluvíme o učení nesupervizovaném. Mezi úlohy nesupervizovaného učení řadíme například shlukovou analýzu, detekci anomalit a další [3].

Nedílnou součástí supervizovaného učení je učení – proces během kterého optimalizujeme parametry zvoleného modelu, tak aby pro danou množinu dat $X = (X_1, \ldots, X_n)$, výstupy naučeného modelu $\hat{Y} = (\hat{Y}_1, \ldots, \hat{Y}_n)$ co nejpřesněji odpovídaly známým vysvětlujícím proměnným $Y = (Y_1, \ldots, Y_n)$. To se běžně dělá pomocí minimalizace ztrátové funkce $L(Y, \hat{Y})$, která se zvyšuje s rozdílem mezi Y a $\hat{Y}$. Kvalita modelu se vyhodnocuje pomocí tzv. metrik, které podobně jako ztrátová funkce vyjadřují, jak moc se $\hat{Y}$ liší od Y. Na rozdíl od ztrátové funkce se metrika nevyužívá pro samotnou optimalizaci a bývá zpravidla intuitivní pro člověka [3].

Kvalitu modelu vyhodnocujeme na separátních datech, na nichž nebyl model učen (testovací množina dat). V opačném případě bychom nemohli s jistotou posoudit, zdali natrénovaný model porozuměl vztahům problematiky, jejíž závislosti se snažíme modelem zachytit, nebo jestli se model naučil vztahy platné pouze pro trénovací data. O rozdílu mezi trénovací a testovací metrikou běžně mluvíme jako o generalizační chybě. Je-li generalizační chyba nízká, znamená to, že model dobře generalizuje zkušenosti z trénovací množiny na testovací. Pokud je testovací množina dobrým, obecným vzorkem dat dané problematiky, pak je nasazení modelu vhodné pro celou problematiku. V opačném případě, kdy je generalizační chyba vysoká, mluvíme o přetrénování modelu. Obrázek 1 zobrazuje příklad tohoto jevu. Přetrénování modelu může nastat z mnoha důvodů. Komplexní modely jako neuronové sítě jsou obecně schopné zachytit složitější vztahy, nicméně bývají zároveň náchylnější k přetrénování. Takové vyžadují pečlivé ladění parametrů,

aby fungovali správně. Proto, pokud není problematika příliš komplikovaná, jsou preferovány jednodušší modely, které se snáze kontrolují. Navíc u prostých modelů jsme schopni vysvětlit jejich motivaci za predikcemi, což je se v mnoha aplikacích požadovaná vlastnost. Další možností, jak předejít přetrénování je technika zvaná regularizace - omezení zesložiťování modelu během trénování například pravidly, nebo přičtením penalizačního členu (vyjadřuje složitost modelu) ke ztrátové funkci. Evaluací modelů se zabývá druhá část této kapitoly [3].

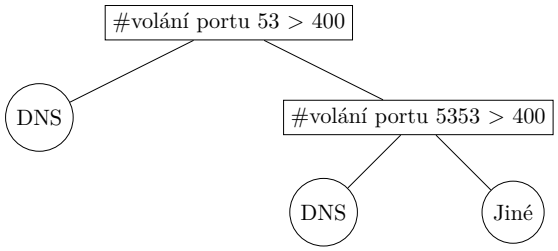


Obrázek 1 Grafy zobrazují dva modely polynomiální regrese, která modeluje f jako polynom stupně n. PR s vysokým stupněm výborně predikuje prvky trénovací množiny, ovšem jeho testovací metrika je mnohem vyšší a je tedy přetrénovaný. PR se stupněm dva dosahuje rozumných výsledků v obou metrikách [4].

0.1 Rozhodovací strom

Rozhodovací strom je jedním z nejjednodušších modelů. Bývá využíván pro klasifikaci i regresi tabulkových dat, tj. množina dat, kde jsou jednotlivé prvky reprezentované vektorem číselných příznaků. Rozhodovací strom je představován binárním stromem, který ve vnitřních uzlech obsahuje rozhodovací pravidla, založená na jednom z příznaků dat. Takový uzel má potomky pro obě možnosti vyhodnocení pravidla. Strom predikuje uzel tak, že začne v kořenu a vydá se cestou příslušící podmínkám vnitřních uzlů. List, v němž cesta skončí obsahuje finální predikci modelu. Příklad zkonstruovaného stromu je vidět na Obrázku 2 [5].

Zbývá otázka: jak rozhodovací strom natrénovat/zkonstruovat? Konstrukce optimálního stromu je NP-problém [6]. Proto běžně během učení rekurzivně hledáme pravidla hladovým algoritmem. Při vytváření pravidla pracujeme s podmnožinou dat trénovacího datasetu, jejíž prvky jsou vyfiltrovány pravidly předků. Pravidla jsou vždy ve formě $atr_i > t$, tedy je třeba vybrat atribut



Obrázek 2 Příklad rozhodovacího stromu určeného pro detekci služby, kterou zařízení poskytuje. Levý potomek je pro splnění podmínky, pravý pro nesplnění.

a mez. Brute force algoritmus vyzkouší všechny kombinace atributů a mezí[1] a vybere kombinaci maximalizující optimalizační kritérium. To udává, jak moc se sníží variabilita, rozdělíme-li množinu I na podmnožiny. Formuli kritéria definujeme jako:

$$IG(I, I_L, I_P) = imp(I) - \frac{\#I_L}{\#I} imp(I_L) - \frac{\#I_P}{\#I} imp(I_P)[5]$$

kde I je množina, I_L, I_P rozdělení množiny pravidlem a funkce imp udává variabilitu (znečištění) množiny, neboli jak moc různorodý je obsah množiny na vysvětlované proměnné. Čím nižší je, tím je méně různých vzorků množina obsahuje. Coby imp se běžně volí entropie:

$$H(I) = -\sum_{j=1} \left(\frac{\#I_j}{\#I} \right) \log_2 \left(\frac{\#I_j}{\#I} \right) [5],$$

případně gini variabilita, definovaná jako:

$$gini(I) = 1 - \sum_j \left(\frac{\#I_j}{\#I} \right)^2 [5]$$

Zde $\#I_j$ je počet prvků s klasifikací j náležící množině I. Vhodná volba imp funkce obecně závisí na konkrétním problému a datasetu [5, 6].

Pospali jsme, jak se konstruuje struktura stromu a určují pravidla v jeho uzlech. Dále se zabýváme tím, kdy větvení zastavit a zakončit větev listem, udávajícím predikci. Nabízí se hned několik podmínek, na jejichž základě je možné větvení zakončit, např: [7]

- Všechny prvky množiny I náleží jedné třídě
- Dosažení maximální hloubky stromu
- Počet prvků množiny I je menší než minimální mez
- Optimalizované kritérium vykazuje minimální zlepšení

Kdybychom ponechali první položku jako jedinou podmínku, pak by vznikl komplexní, hluboký a přeučený strom. Využitím dalších metod zabraňujeme přeučení [7].

Další možností jak strom regularizovat je prořezávání větví poté, co už je kompletně sestrojen. Ty snižují komplexitu stromu podle různých kritérií [7].

Dojde-li k naplnění podmínek pro zastavení vytváření pravidel pro danou množinu vytvoříme list, jehož predikce se rovná modus vysvětlované proměnné (průměr v případě regrese) [7].

0.2 XGBoost

Ensemble modely je třída modelů, která k predikci využívá vícero slabších modelů a jejich predikce slučuje agregační funkcí. Základní myšlenkou této třídy je, že kombinace různých modelů může vytvořit robustnější a spolehlivější model, než jeden komplexní model [8].

Jedním typem ensemble modelů je Gradient boosting, metoda, která sčítá predikce slabých modelů. Sčítáním predikcí modelů se však zcela mění úloha trénování slabých modelů, tedy namísto aby se učili predikovat vysvětlovanou proměnou na základě dat, se učí opravovat chybu součtu předchozích modelů všech datových bodů (x, y):[8]

$$\hat{y} = F(x) = \sum_t f_t(x), f_t \in \mathcal{F} \quad [8]$$

[1] Existují i optimalizace využívající předpočítávání histogramů, snižující počet kombinací

kde $\mathcal{F} = \{f(x) = w_{q(x)}\}$, $(q : x \to \{1, 2, \dots T\})$ je funkce příslušící stromu f s T listy, zobrazující datový bod na index listu stromu, do nějž ho pravidla zařadí a w_i je váha (predikce) i-tého listu stromu. Každé f zde označuje stromy o různých pravidlech, strukturách i váhách. Dále pro označení t-tého modelu budeme používat f_t a $\hat{y}^{(t)} = \sum_i^t f_i(x)$ predikce prvních t modelů[8].

XGBoost (eXtreme Gradient Boosting) staví na myšlence boostingu za použití stromů. Novotou přístupu je přidání regularizačního členu zabraňujícího přeučení. Od svého uvedení XGBoost nabyl veliké popularity díky skvělým výsledkům, snadného použití skrze knihovnu a snadnému ladění parametrů a stal se state-of-the-art modelem pro úlohy supervizovaného učení na tabulkových datech[8].

Formálně XGBoost minimalizuje funkci:

$$\mathcal{L} = \sum_i l(\hat{y}_i, y_i) + \sum_t \Omega(f_t), \quad \Omega(f) = \gamma T + \frac{1}{2}\lambda \|w\|^2 \quad [8]$$

kde l je konvexní ztrátová funkce měřící rozdíl mezi predikcí a cílovou hodnotou. Regularizační člen Ω penalizuje složitost stromu, γ a λ jsou hyperparametry. Zvyšováním těchto hyperparametrů se zvyšuje i Ω, algoritmus volí jednodušší stromy a zabraňuje se přeučení[8].

Jednotlivé slabé modely jsou učeny popořadě a jako základ se při klasifikaci často používá průměr vysvětlované proměnné $f^{(0)} = \bar{Y}$ nebo $f^{(0)} = 0.5$. Aby se následující stromy namísto odhadu vysvětlované proměnné učili predikovat chybu, je třeba upravit kritéria pro vytváření pravidel a zavést váhy listů. Pro nový (t-tý) strom minimalizujeme:

$$\mathcal{L}^{(t)} = \sum_i^n l\left(y_i, \hat{y}_i^{(t-1)} + f_t(x_i)\right) + \Omega(f_t) \quad [8]$$

Hladovým algoritmem volíme strom f_t, který minimalizuje optimalizační kritérium podle této rovnice. Funkci aproximujeme Taylorovým polynomem druhého stupně a následně jej optimalizujeme:

$$\mathcal{L}^{(t)} \simeq \sum_i^n \left[l\left(y_i, \hat{y}_i^{(t-1)}\right) + g_i f_t(x_i) + \frac{1}{2} h_i f_t^2(x_i) \right] + \Omega(f_t) \quad [8]$$

kde $g_i = \partial_{\hat{y}_i^{(t-1)}} l(y_i, \hat{y}_i^{(t-1)})$ a $h_i = \partial^2_{\hat{y}_i^{(t-1)}} l(y_i, \hat{y}_i^{(t-1)})$ substituujeme za první a druhou derivaci ztrátové funkce. Definujeme $I_j = \{i | q(x_i) = j\}$, množinu indexů všech vzorků dat příslušící listu s indexem j. Odstraněním (z pohledu optimalizace) konstantních členů a rozepsáním penalizačního členu Ω získáváme minimalizovanou funkci vyjadřující kvalitu nového stromu:

$$\begin{aligned}
\tilde{\mathcal{L}}^{(t)} &= \sum_i^n \left[g_i f_t(x_i) + \frac{1}{2} h_i f_t^2(x_i) \right] + \gamma T + \frac{1}{2}\lambda \sum_j^T w_j^2 \\
&= \sum_{j=1}^T \left[(\sum_{i \in I_j} g_i) w_j + \frac{1}{2} (\sum_{i \in I_j} h_i + \lambda) w_j^2 \right] + \gamma T
\end{aligned}$$

Pro strom dané struktury $q(x)$ spočítáme derivací optimální váhy:

$$w_j^* = -\frac{\sum_{i \in I_j} g_i}{\sum_{i \in I_j} h_i + \lambda} \quad [8]$$

a dosazením do původní funkce získáváme pro strom struktury $q(x)$ funkci vyjadřující kvalitu stromu[8]:

$$\tilde{\mathcal{L}}^{(t)} = \sum_{j=1}^T \left[\frac{(\sum_{i \in I_j} g_i)^2}{\sum_{i \in I_j} h_i + \lambda} \right] + \gamma T \quad [8]$$

Nový strom je stavěn stejně, jako v sekci 0.1, ovšem listy mají výstup daný w_j^* a pro vytváření pravidel využíváme obměnenou funkci:

$$\mathcal{L}_{split}(I, I_L, I_P) = \frac{1}{2}\left[\frac{(\sum_{i\in I_P} g_i)^2}{\sum_{i\in I_P} h_i + \lambda} + \frac{(\sum_{i\in I_L} g_i)^2}{\sum_{i\in I_L} h_i + \lambda} - \frac{(\sum_{i\in I} g_i)^2}{\sum_{i\in I} h_i + \lambda}\right] - \gamma \quad [8]$$

, kde v případě binární klasifikace, jíž se budeme dále zabývat, se používá ztrátová funkce binární cross-enropy:

$$l(y, \hat{y}) = -y \times ln(\hat{y}) - (1 - y)ln(1 - \hat{y}) \quad [8]$$

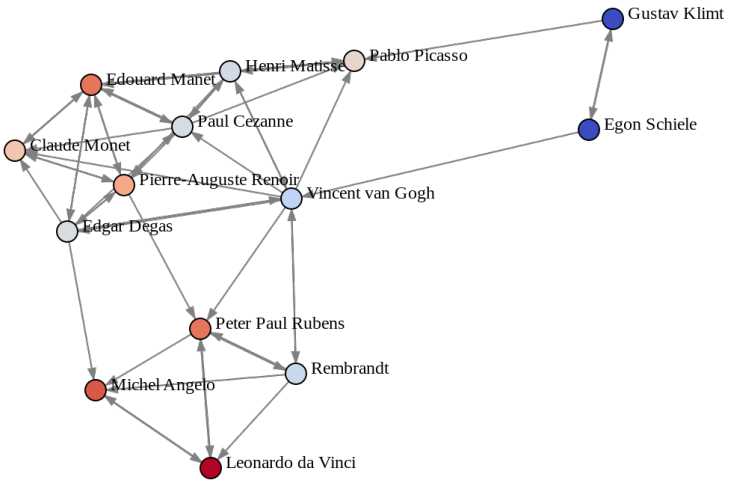
.

Podobně jako stochastický gradientního sestupu (SGD) využívá learning rate pro regularizaci učení, tak XGBoost využívá η shrinkage faktor. Každý přírůstek f_t je přenásoben η, což omezuje jeho dopad na výslednou predikci zvyšuje robustnost modelu. Běžně se η pohybuje v intervalu $\langle 0, 1\rangle$ [8].

0.3 PageRank

S nástupem internetu vzešla potřeba efektivního vyhledávání na webu, což představovalo velikou technickou výzvu. V devadesátých letech bylo k dispozici hned několik vyhledávačů. Ty pro vyhledávání relevantních stránek a dokumentů využívali především porovnávání obsahu a klíčových slov s dotazem a počet hypertextových odkazů směřující na ně. Tyto přístupy jsou však náchylné vůči umělému nafukování důležitosti stránky za pomocí nekalých metod jejím autorem. V roce 1998 navrhli L. Page a S. Brin algoritmus PageRank, který vytvořili za účelem posouzení důležitosti jednotlivých stránek. Ten staví na myšlence, že jednotlivé stránky jsou důležité, jestliže na ně odkazují jiné důležité stránky. Algoritmus se stal klíčovým prvkem vyhledávače Google a podle autorů byl PageRank základem pro jeho úspěch[9, 10].

Ačkoliv byl PageRank navržen primárně pro posuzování internetových stránek, našel aplikaci napříč mnoha úlohami v oblasti teorie grafů. My představujeme možnost jeho využití pro posouzení důležitosti zařízení na síti v Sekci 3.1.2.



Obrázek 3 Příklad aplikace PageRanku na graf. Barvy vrcholů udávají přidělené stacionární rozdělení PageRankem [11].

PageRank pohlíží na strukturu webu jako na orientovaný graf $G = (V, E)$ kde stránky/dokumenty představují vrcholy V a hypertextové odkazy jsou hrany E. Intuicí algoritmu je, že stránka je

důležitá jako stránky na ni se odkazující, formálně:

$$r(P) = \sum_{Q \in N_{in}(P)} \frac{r(Q)}{deg_{out}(Q)}$$

sčítáme důležitost r všech prvků vstupního sousedství děleno výstupním stupněm souseda (tak prvek rozdělí svou důležitost mezi sousedy uniformně). Aby se rovnice neodkazovali na nedefinovanou r vrcholu zavádíme počáteční důležitost $r_0(V_i) = 1/\#V$. Grafy s topologií webových sítí běžně obsahují cykly a proto obecně neexistuje topologické uspořádání počítání vrcholů, které by zajistilo, že počítání r jednoho vrcholu neovlivní již vypočítanou důležitost jiného vrcholu. Problém se proto řeší iterativně. Definujeme vektor $\pi_k = (r_k(P_1), r_k(P_2), \ldots, r_k(P_n))^2$ jako rozdělení důležitosti a celý problém budeme zapisovat maticově [9, 10]:

$$\pi_k = \pi_{k-1}P, \quad P_{ij} = \begin{cases} 1/deg_{out}(P_i) & \text{existuje-li hrana z } P_i \text{ do } P_j \\ 0 & \text{jinak} \end{cases}$$

Na problém pak můžeme nahlížet jako na dobře prozkoumanou úlohu hledání stacionárního rozdělení homogenního markovského řetězce reprezentovaného maticí přechodu P. Z hlediska markovské teorie představuje tento řetězec náhodnou procházku a $(\pi_j)_i$ je pravděpodobnost, že náhodný chodec po j náhodných proklicích odkazů dojde do vrcholu i [10]. Příklad aplikace PageRanku je vidět na Obrázku 3.

Vystává zde však jeden problém: validní matice přechodu pro Markovský řetězec musí být stochastická (všechny její řádky se nasčítávají do 1 a všechny prvky jsou nezáporné), což pro případ kdy vrchol nemá žádnou výstupní hranu neplatí. Proto pro řádek představující vrchol bez výstupních hran zavádíme $P_i = \mathbf{e}/n$ [10].

Další problém může vyvstat obsahuje-li řetězec uzavřenou množinu stavů (což běžně nastává ve webových i internetových sítích). Stav P náleží uzavřené množině, jestliže se chodec z něj nemůže dostat do kteréhokoliv ze stavů za libovolné množství kroků. Následkem toho zbylé stavy budou mít nulovou pravděpodobnost ve stacionárním rozdělení, zatímco stavy z uzavřených množin všechnu pozřou. To se neshoduje s původní ideou, stránky bez vnějších odkazů, nebo s odkazy v uzavřené množině by získali všechnu důležitost [10].

Aby se předešlo problému uzavřených množin autoři zavedly tzv. metodu teleportace - z jakéhokoliv stavu náhodných chodec přeskočí s pravděpodobností $1 - \alpha$.

$$G = \alpha P + (1 - \alpha)F$$

kde $F = ee^T$. G je známá jako Google matice. Běžné hodnoty pro α jsou okolo 0.8 a 0.9.

O některých stránkách můžeme dopředu vědět, že jsou často využívané a můžeme proto chtít dopředu zvýšit jejich důležitost. Toho se u PageRanku dosahuje pomocí tzv. personalizace. Algoritmus ohýbáme tak, aby tíhnul k vybraným stavům, zavedením personalizačního vektoru v, kde v_i udává pravděpodobnosti, že teleportace skočí do stavu i. Matici teleportace pak definujeme jako $F = ev^T$ [10]. Příklad aplikace PageRanku s personalizací je ilustrován na Obrázku 4

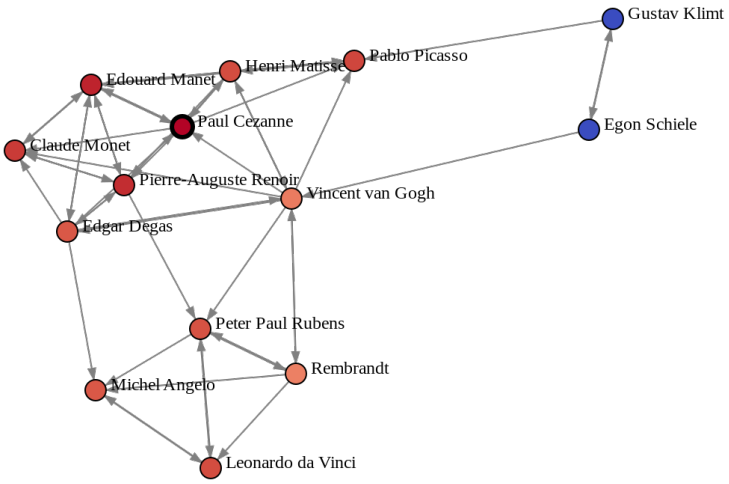
Výslednou rovnici:

$$\pi_{k+1} = \pi_k G$$

pak je možné řešit několika přístupy. Podle dostupných údajů, Google volí mocninnou metodu [10].

0.4 Ladění hyperparametrů modelu

Součástí mnoha modelů strojového učení jsou hyperparametry. Tak se nazývají parametry, které neladí algoritmus, namísto toho jsou nastaveny expertem než před samotným učením. Mezi ně například patří dříve zmíněné penalizační parametry λ a γ u XGBoost - Sekce 0.2.



Obrázek 4 Příklad aplikace PageRanku na graf. Barvy udávají přidělené stacionární rozdělení PageRankem. Vrchol se zvýrazněným okrajem má v personalizačním vektoru udanou podstatně vyšší pravděpodobnost, než ostatní [11].

Volba konfigurace hyperparametrů je stěžejní pro správné učení a regularizaci modelu. Bohužel, je velice náročné apriorně predikovat, jak se změna hyperparametrů projeví na kvalitě výsledného modelu, obzvlášť měníme-li více vzájemně se ovlivňujicích parametrů najednou. Proto se optimální konfigurace hyperparemetrů běžně hledá zkoušením konfigurací - tedy pro všechny prvky z množiny navržených konfigurací se natrénuje model a kvalita konfigurace je dána jeho metrikou.

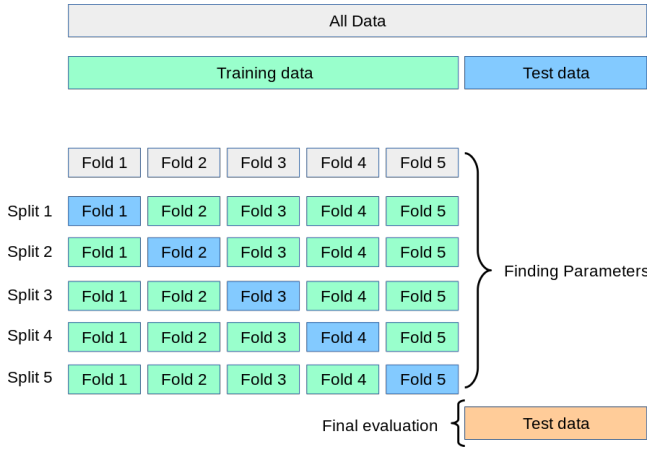
Konfigurace navhujeme z prostoru hyperparamtrů. Zde však narážíme na tzv. prokletí dimenze - nově laděný parametr představuje novou dimenzi a exponenciálně tak roste počet konfigurací, vyžadovaných pro stejnoměrné pokrytí prostoru. Aby se předešlo neúnosně dlouhému ladění parametrů, je důležité omezit počet prohledávaných konfigurací, vyžaduje-li proces učení veliké množství zdrojů.

Mezi běžné postupy pro navrhování konfigurací algoritmů řadíme:

- Grid search pro každý parametr zvolí několik hodnot k vyzkoušení. Poté vyhodnotí každou kombinaci těchto hodnot. Toto prohledávání je vhodné pro malé konečné prostory parametrů, nebo zevrubné zmapování prostoru s malou podmnožinou parametrů [12].
- Random search náhodně vybírá konfigurace z prostoru hyperparametrů. Přístup se používá při vysoké dimenzi prostoru pro zúžení volby vhodných hodnot [12].
- Bayesovské přístupy hledají nejlepší parametry aproximací funkce zobrazující set hyperparametrů na metriku. Vyzkoušené konfigurace se využívají pro zpřesnění aproximace funkce a na jejím základě je navrhován nový set parametrů, maximalizující očekávané zlepšení. Narozdíl od předchozích přístupů bayesovská optimalizace využívá znalostí nabytých z provedených evaluací a může tak podstatně zlevnit hledání hyperparametrů, protože vyžaduje méně iterací k nalezení dobré konfigurace než primitivní přístupy. Pro vysoké dimenze prostoru hyperparametrů se však kvalitou konfigurací blíží random search [13].
- Successive halving je metoda, která pracuje se zdroji (výpočetní čas, počet dat, a pod.) přidělovanými jednotlivým konfiguracím. Na počátku se zvolí veliké množství konfigurací. Poté mezi všechny konfigurace uniformě rozdělí zdroje. Po dokončení evaluace se vybere zlomek nejhorších vzorků a vyloučí se. Kroky přidělení zdrojů, evaluace a selekce provádí algoritmus iterativně, až zbude jediná, zvolená, konfigurace. Každá další iterace získává více zdrojů a evaluace konfigurací jsou tak přesnější a přesnější. Na tomto přístupu stojí i algoritmus

Hyperband, který odstraňuje některé slabiny Successive halving a je dnes často využívaný pro ladění hyperparametrů neuronových sítí [13].

Naskýtá se otázka, na jaké množině je vhodné porovnávat kvalitu konfigurací parametrů při jejich hledání? Evaluací na trénovacím datasetu není možné posoudit kvalitu generalizace modelu. Pokaždé když využijeme nějaká data pro evaluaci modelu a poté na jejím základě ho nějak poupravíme, "prosákne" jistá část informace o datasetu do modelu. Proto pokud využijeme pro evaluaci hyperparametrů testovací dataset, pak ovlivní jeho vzorky výsledný model, kompromituje se nezávislost modelu na testovací množině a nebude možné objektivní posouzení schopnosti generalizace modelu. Abychom tomu předešli, zavádíme třetí množinu zvanou validační, určenou právě pro evaluaci konfigurací hyperparametrů a různých experimentů.



Obrázek 5 Znázornění cross-validace s k = 5. [14]

Čím více úprav modelu provedeme na základě evaluací validačního datasetu, tím menší je věrohodnost naměřené generalizační chyby vůči trénovací metrice. Experti proto často namísto validační množiny vyžívají metodu Cross-validace, která tento problém potlačuje. Cross-validace se používá s rozdělením celého datasetu pouze na trénovací a testovací množinu. Metoda rozdělí trénovací množinu na k náhodných, disjunktních, přibližně stejně velikých podmnožin. Model je poté natrénován s pomocí $k-1$ podmnožin, zatímco jedna vynechaná podmnožina slouží k vyhodnocení modelu. To je provedeno pro každou podmnožinu a celková metrika konfigurace je vyjádřena jako průměr všech dílčích metrik. Detail na Obrázku 5. Čím vyšší k je zvoleno, tím nižší je rozptyl vyhodnocení konfigurace. Je možné dokonce zvolit k rovné velikosti trénovacího setu a získat tak takřka nevychýlené vyhodnocení (leave-one-out Cross-validation). Ovšem se zvyšujícím se k rostou i nároky na výpočetní výkon a to k-krát v porovnání s jednoduchým rozdělením na trénovací, validační a testovací množinu. Vysoká k nejsou únosná pro modely s výpočetně náročným trénováním, v praxi se proto obvykle užívá $k < 10$ [15].

Klademe na data požadavek, aby obecně reprezentovali všechny možné vzorky, které se mohou objevit. V nedokonalém světě však tento požadavek běžně zůstává nenaplněn, kvůli chybách v měřeních, změnám domény vzorků v čase a mnoha dalších. Budeme-li mít například data o návštěvnosti podniku, zaznamenávaná během čtyř dnů v jednom týdnu, může nás zajímat, jak dobře si model povede v ostatních dnech, pro něž data nemáme. Chceme tedy znát dovednost modelu generalizovat napříč dny. Abychom ji odhadli, změříme generalizaci modelu na dnech, jejichž data známe. Toho docílíme právě využitím Cross-validace, ovšem namísto náhodných podmnožin, rozdělíme data do množin podle dne původu. Takto využíváme Cross-validaci i v dalších případech, kdy je nám známá závislost mezi jednotlivými prvky a náhodné rozdělení na množiny pak není správně reprezentativní.

Kapitola 1

Rešerše metod pro klasifikaci zařízení a služeb

V posledních letech vídáme strmý nárůst zařízení a komunikace na počítačových sítích. Odhaduje se, že v roce 2018 bylo v provozu 25 miliard po síti komunikujících zařízení a do budoucna je třeba počítat s dalším nárůstem [16]. Zejména na sítě v podnicích, které obvykle bývají nejrozsáhlejší, jsou kladeny stále větší nároky jak po hardwarové a softwarové stránce tak i na jejich spravování. Vzestup IoT (Internet of Things) filozofie nejenže zvýšil počet zařízení a objem toků na síti, ale také kvůli ní narůstá variabilita zařízení, jež se na síti mohou vyskytovat a tím se výrazně ztěžuje identifikace zařízení. Právě identifikace zařízení je klíčová pro správu sítě a její automatizace je kritická pro monitoring sítě, identifikaci potenciálních rizik a správu výkonu velkých sítí [16, 17].

Zvětšující se objem zaznamenávaných dat a počet zařízení na síti, dal za vzestup uplatnění strojového učení napříč mnoha úlohami v oblasti počítačových sítí. Tato kapitola poskytuje čtenáři povědomí o využití strojového učení v networkingu. Zvláštní důraz klademe na metody obecně používané pro klasifikaci zařízení, jejichž poznatky lze přenést na naši úlohu klasifikace zařízení.

Mezi používané aplikace patří predikce síťové komunikace, jejíž poznatky lze využít pro apriorní alokaci výpočetních zdrojů, optimalizaci směrování a dalším účelům. Úloha dříve řešená především statistickou analýzu pro odhadování modelů časových řad (zejména ARIMA modely), je dnes běžně řešena s využitím pokročilejších modelů (neuronové sítě, rekurentní neuronové sítě), které dokáží naplno využít komplexnější data [16].

Dalším využitím strojového učení v počítačových sítích může být detekce abnormalit. Abnormální chování na síti mohou naznačovat výpadek prvku sítě, kybernetický útok na síť, nebo jakýkoliv jiný speciální stav, vyžadující pozornost správce sítě [16].

V této práci se budeme zaměřovat především na metody používané pro klasifikaci zařízení a detekce služeb, které poskytují. Za pomocí statistik naměřených na síti je třeba analyzovat chování zařízení. Klasifikace typu zařízení je jednou z běžných úloh v oblasti IoT, kde správce sítě zajímá, který typ IoT zařízení se nachází za danou IP adresou. Ačkoliv se tato práce klasifikací zařízení zda-li je či není server, má klasifikace typu zařízení k této úloze velmi blízko, neboť z pohledu strojového učení pracují se stejnými informacemi a tedy i předzpracování dat a extrakce atributů podléhají stejným krokům. Hlavním rozdílem bude použitý model a objektivní funkce užitá při jeho učení [17].

Typ zařízení je pro správce podstatnou informací pro monitoring a správu sítě. Získává díky ní přehled o zařízeních připojených síti a lze na jejich základě mimo jiné i provádět analýzu rizik, vzniklých výpadnkem stroje. Techniky, které se využívají rozděluje Studie [17] do tříd,

podle kterého kritéria prvky klasifikují: port, packet, chování a statistická analýza komunikací. Následující sekce nabízí jejich podrobnější popis.

1.1 Klasifikace podle portu

Jedním z nejpřímočařejších přístupů jak klasifikovat zařízení je klasifikace podle portu. Pro ni je třeba zaznamenat, na který portech jednotlivá zařízení pracují, popřípadě využít software, který oskenuje, na kterých portech zařízení přijímá komunikace. Na základě zaznamenaných portů můžeme zjistit, které služby dané zařízení poskytuje. Abychom mohli zařízení klasifikovat, je třeba zaznamenat profil portů, kterým daný typ zařízení disponuje. Pro přiřazení typu k zařízení pak stačí nalézt typ, který je profilem používaných portů nejpodobnější [17].

Výhodou této metody jsou jednoduchost a nízké výpočetní nároky v porovnání s ostatními přístupy. Ovšem posuzování zařízení pouze na základě portu není považováno za robustní. Číslo portu nemusí odpovídat službě, dle níž chceme zařízení posuzovat a to hlavně vlivem stále častějšího využívání dynamického přiřazení portu, síťového tunelování a užívání službám nepříslušících portů z nejrůznějších důvodů. Navíc ne všechny porty jsou registrované u autority IANA [18], takže není možné je jednoznačně přiřadit ke službě. Výsledné modely pro klasifikaci právě kvůli těmto problémům nedosahují tak vysoké přesnosti jako další metody [16, 17].

1.2 Klasifikace inspekcí packetu

Tato technika využívá pro klasifikaci analýzu packetu a to jak jeho hlavičky tak i obsahu. Podle nich je možné velmi přesně identifikovat známé aplikace a přístup díky nim získává velmi vysokou přesnost klasifikace. Stejně jako klasifikace pomocí portu i tato metoda trpí, je-li na síti použito tunelování. Navíc každý nový komunikační protokol, nebo evoluce starého vyžaduje úpravu v algoritmu [16, 17].

Navíc jsou stále častěji v oblasti počítačových sítí využívané šifrované komunikace. Je-li packet šifrován, pak se tato metoda nedá využít [17, 19].

1.3 Klasifikace statistickou analýzou

Metoda pracuje ve dvou krocích: nejprve extrahuje vektor příznaků ze syrových dat, načež na těchto datech naučí některý model strojového učení pro klasifikaci. Mezi používanými atributy extrahovanými z packetů jsou časové řady, hlavička packetu, obsah dat a statistické údaje. Časové řady jsou konstruovány s využitím N posledních packetů a mohou obsahovat data jako je jejich velikost, časová prodleva mezi nimi, směr packetu a jiné. Statistické údaje pak mohou obsahovat různé statistické agregace zmíněných časových řad [1, 19].

Stejně jako klasifikace inspekcí packetu i tato může metoda produkovat kvalitní predikce. Ovšem na rozdíl od ní je možné klasifikaci statistickou analýzou využít i na šifrovaná data, vystačíme-li si s daty o časových řadách a statistickými údaji. Také se liší tím, že využívá vícero packetů a analyzuje se vztahy mezi nimi [1, 19].

1.4 Klasifikace podle chování

Poslední představenou třídou je klasifikace podle chování zařízení. Ta klasifikuje zařízení na základě toho, jak dané zařízení navenek interaguje s ostatními zařízeními. Hledá vzorce chování zařízení na základě informací obsažených v hlavičkách i payloadu paketu, které zařízení přijímá, popřípadě odesílá. Mezi používané mohou být například čas a interval komunikace, velikost payloadu, použitá čísla portů, počet komunikujících zařízení, použitý protokol a další [16, 17].

Díky využití i hlavičky namísto samotného obsahu je přístup robustní vůči šifrování komunikace, což je zvlášť důležitou výhodou v moderních sítích [16].

Pro realizaci se často využívají algoritmy strojového učení, které dokáží automaticky rozpoznávat a analyzovat vzory chování zařízení [16, 2].

Autoři Studie [2] se zabývají návrhem klasifikačního frameworku, který využívají vícevrstvých LSTM autoencoderů. Právě využití autoencoderů jim dovoluje extrakci klíčových atributů zařízením vykazované komunikace. Na těchto atributech poté provádí pravděpodobnostní matching zařízení s atributy příslušejícím daným třídám. Výhodou tohoto přístupu je, že přidání nového typu zařízení nevyžaduje nákladné přetrénování encoderu, ale stačí extrahovat atributy typické pro novou třídu a zahrnout ji do pravděpodobnostního matchingu. Tato metoda dosahuje v průměru 82% F1 skóre a 70% accuracy na typech IoT zařízení, které nebyli použity při učení [2].

Další práce se soustředí na detekci předem daných typů zařízení na základě charakteristik konkrétního typu. Studie [20] se zaměřuje na detekci peer-to-peer komunikace streamingových služeb za využití Support Vector Machines aplikovaných na velikost payloadu a počtu odeslaných packetů v malém časovém okně. Zvolené atributy mají podobné vlastnosti, nehledě na protokolu. Detekcí těchto zařízení může správce sítě efektivněji plánovat a řídit zdroje sítě, což vede ke snížení nákladů a zvýšení efektivity [20].

Představené přístupy nám poskytli vhled do problematiky klasifikace zařízení na síti. Nicméně se záasadně liší v dostupných datech, jejich omezeních (popsáno v Kapitole 2) a také v konkrétní úloze. Proto z těchto metod přímo nevycházíme a ani je nemůžeme srovnávat co do výkonu s našimi přístupy.

Kapitola 2

Dataset a reprezentace dat

V této kapitole se nachází detailní popis dat dodaných zadavatelem, jejich syrová forma a niance, které se v datech vyskytují. V druhé části se zaměříme na možné reprezentace dat, které je pak možné zpracovávat modely strojového učení.

Dataset dodaný zadavatelem práce obsahuje záznamy o komunikacích na sítích sedmi zákazníků. Od každého ze zákazníků byly získány záznamy komunikace v časovém období jednoho týdne. Všechny záznamy jsou unicast TCP/UDP sockety mezi prvky na interní síti. Jeden záznam obsahuje následující informace vyobrazené v Tabulce 2.1.

Atribut	Popis	Volitelný
Timestamp	Unixový čas komunikace	Ne
Company id	Hash - identifikátor zákazníka	Ne
Device id	Hash - identifikátor zařízení	Ne
Source ip	Hash - ip adresa původce komunikace	Ne
Source port	Port původce komunikace	Ne
Destination ip	Hash - ip adresa volaného	Ne
Destination port	Port volaného	Ne
URL	Hash - volaná url adresa	Ano
Hostname	Hash - název zařízení	Ano
SHA	Hash - jméno komunikaci navazujícího příkazu	Ne

Tabulka 2.1 Popis atributů záznamu komunikace

Z tabulky je vidět, že mnoho údajů je zahashováno. To bylo provedeno v zájmu anonymizace dat, protože data pochází z produkčních sítí zákazníků a uveřejnění jejich vnitřní struktury by představovalo bezpečností riziko.

Záznamy komunikací byli sbírány za pomocí softwaru nainstalovaného na koncových zařízeních ve formátu netflow. Komunikace jsou po zachycení odeslány na centralizovaný server k další analýze. Aby se předešlo nutnosti pracovat s ohromným objemem dat, je pro každý pár zařízení zaznamenáno pouze prvních 30 komunikací toku.

Častým problémem u praktických dat je jejich neúplnost. Naneštěstí jím trpí i dodaný dataset. To je způsobeno absencí softwaru pro sběr dat na některých zařízeních. V datech se sice objevují i tato zařízení, ovšem pouze jako příjemci komunikace, nikdy ji samy neotevřou. Právě proto se v práci klasifikace opírá zejména na datech o příchozích komunikacích, ačkoliv atribut jako je poměr příchozích/odchozích komunikací zařízení by měl výraznou informační hodnotu.

Krom záznamů komunikací nemáme k dispozici další data. Neboli zde není dataset, který by explicitně popisoval zařízení. Navíc příznak Device id, který se objevuje jako atribut dat

záznamů, není podle dodavatele dat pokládat za spolehlivé označený zařízení. Proto zařízení identifikujeme pomocí jeho ip adresy. Pokud zařízení během dne nekomunikuje, nebude se z našeho pohledu vyskytovat na síti. Nicméně je třeba mít na paměti, že zařízení komunikující za danou ip adresou se mohou v průběhu času měnit. To způsobuje, že starší zaznamenané komunikace přiřazené danému zařízení mu nemusí být přiřazeny právem. Abychom zredukovali dopady této komplikace a mohli data náležitě využít, pracujeme s omezeným časovým oknem. V práci pokud není zmíněno jinak využíváme interval dat jednoho dne.

Na základě popsaných dat chceme klasifikovat všechny prvky, které se na síti nachází. Za server považujeme zařízení, které poskytuje nějaké služby, nebo zdroje. Dalším úkolem je zjistit, které služby jsou pro daný server stěžejní a jak může ovlivnit výpadek zařízení ovlivnit dostupnost služeb na síti. V této práci budeme celý proces rozdělovat právě na tyto samostatné kroky.

Nejprve rozdělíme všechny zařízení na síti na servery a ostatní. Díky dostupnosti externího posouzení pro jednotlivé ip adresy - stroj je/není server - můžeme k úloze přistupovat jako k úloze binární klasifikace a využít zde modelů strojového učení. V ojedinělých případech mezi labely chybí některé z ip adres. Po prozkoumání těchto zařízení jsme zjistili, že se zpravidla jedná prvky, na něž směřovaly jenom jednotky komunikací. Usoudili jsme tedy, že se nejedná o servery a přistupujeme tak k nim při analýze dat i učení modelů.

Tabulka 2.2 poskytuje přehled o rozdělení zařízení jednotlivých záznamů zákaznických sítí. Za zmínku stojí nevyváženost datasetu, jak ve vysvětlované proměnné tak i v rozdělení počtu zařízení mezi zákazníky. Nevyváženost dat je častým problémem v oblasti strojového učení, který negativně ovlivňuje kvalitu predikcí minoritní třídy. To může vést k natrénování zkresleného modelu, neboť bude přikládat větší váhu majoritní třídě.

	Pondělí	Úterý	Středa	Čtvrtek	Pátek	Sobota	Neděle	Denní průměr	
								Servery	Ostatní
Z1	62	48	62	45	31	24	22	1.1	40.9
Z2	54	54	59	66	54	53	62	2.3	55.1
Z3	79	58	50	58	65	45	42	1.0	55.7
Z4	271	211	220	181	188	72	81	2.1	172.7
Z5	2310	2428	2999	2892	2647	1443	1287	89.0	2197.6
Z6	11676	13130	12005	11665	10775	4796	4533	640.6	9156.6
Z7	76051	89334	88816	89040	77096	28441	30605	468.1	68015.1

Tabulka 2.2 Tabulka popisuje, kolik zařízení bylo zaznamenáno na jednotlivých sítí Z1 – Z7 v průběhu týdne. Dva sloupce napravo udávají průměrné denní rozdělení klasifikačních tříd.

Vzhledem k tomu že známe pouze port, na němž komunikace proběhla, nikoliv službu jíž se týkala, není možné snadno určit, které služby dané zařízení poskytuje. Některé služby jsou charakteristické portem, na němž komunikují. Mezi porty které takto můžeme mapovat, řadíme "well known port numbers" v rozsahu 0 - 1023 a registrované čísla portů ve zbylém rozsahu, oboje definované normou RFC1700 [18]. Ovšem napříč sítěmi zákazníků se nachází různorodé služby, komunikující i na portech normou nedefinovaných. Abychom mohli obecně referovat i neznámé služby mlvuíme v práci o službách především v kontextu jejich portu.

2.1 Grafová reprezentace dat

Z dostupných dat můžeme zkonstruovat orientovaný multigraf $G = (V, E)$, kde vrcholy představují IP adresy a hrany proběhlé komunikace mezi nimi. Hrana navíc nese i atributy záznamu. V kontextu teorie grafu využijeme zaběhlé terminologie pro zkoumání vlastností komunikace na síti.

2.2 Vektorová reprezentace dat

Další možností jak reprezentovat problematiku je vektorem číselných atributů. Tyto atributy popisují vlastnosti zařízení a jeho sousedství. Sousedství budeme reprezentovat histogramem, který zaznamenává četnosti volání několika vybraných portů. Vektory zařízení společně představují tabulkový dataset, na nějž je možné využít dobře prozkoumané modely strojového učení.

Hlavní nevýhodou této reprezentace je ztráta velikého množství informací. Ta počíná u globálních vlastností sítě: počet zařízení, komunikací, srovnání s ostatními prvky. Také ztrácíme informace vyplývající ze širšího sousedství ne jenom přímých sousedů. Přijdeme i o přesný popis vstupních komunikací. Nakonec zcela ztracena je i informace o výstupních komunikacích, ovšem vzhledem k problémům s jejich zaznamenáváním se jen stěží dá hovořit o hodnotném atributu dat. Stejně tak se dá diskutovat o důležitosti širšího sousedství, jelikož se nedá předpokládat, že by hranami reprezentované komunikace na síti byly vzájemně závislé. Právě naopak se čeká, že většina komunikací probíhá ve vztahu klient-server a tedy volání dalších služeb v návaznosti na původní komunikaci ze strany volaného nedávají smysl.

Pro zachování části globální informace o grafu, zahrnujeme mezi atributy počet zařízení, komunikací a den v týdnu.

Pro konstrukci histogramu je třeba nejprve vybrat, které porty jsou pro komunikace nejdůležitější. Problémem je, že ne všechny sítě zákazníků využívají všechny služby stejně, tedy rozdělení portů komunikací se může výrazně lišit. Nejprve se snažíme vybrat tzv. **obecné porty** - porty, které aktivně využívá většina zákazníků. Předpokládáme, že takové porty nesou hodnotné informace, bez ohledu na konkrétní užitou doménu. Jedná se o porty služeb, jejichž přítomnost najdeme takřka na všech sítích: 53 - DNS, 80 - HTTP [18].

Nejprve posoudíme důležitost portů u každého zákazníka samostatně. Každému portu tedy chceme přiřadit skóre, které vyjadřuje jeho důležitost. Pro každého zákazníka sestavíme žebříček portů, dle četnosti jejich používání v průběhu celého týdne. Skóre portu u zákazníka odpovídá jeho postavení v žebříčku. Má-li vícero portů stejné postavení, je jim přidělené stejné číslo a následujícímu portu v pořadí je přiřazeno následující číslo. Tak metoda omezuje dopad náhodného šumu málo používaných portů na skóre. K tomu je i invariantní vůči objemu dat sítě, takže zohledňuje doménu každého zákazníka rovnoměrně.

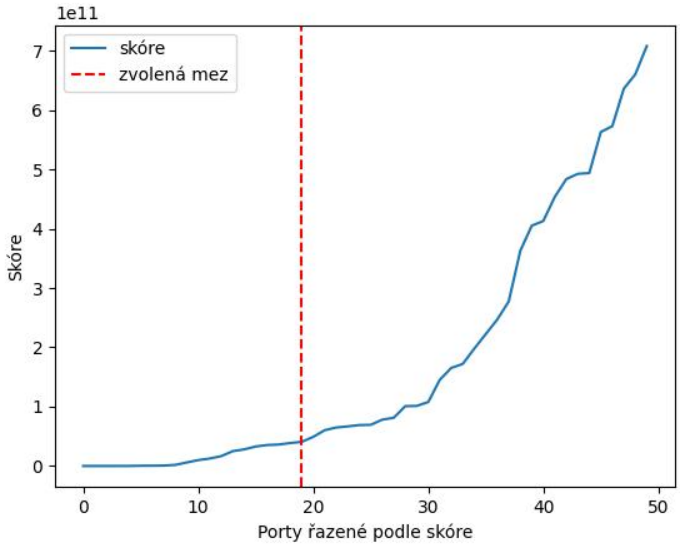
Celkové skóre portu je rovno produktu jeho skóre napříč zákazníky - nižší skóre představuje významnější port. Tato agregace je zvolená, protože výsledné skóre preferuje porty, které byly významné pro většinu zákazníků. Oproti tomu porty, které byly posouzeny jako významné u jednoho zákazníka zatímco u ostatních se nevyskytovali jsou násobením výrazně penalizovány. Po ohodnocení portů je třeba zvolit, kolik nejvýznamnějších bude využito v histogramu. Pomocí tzv. elbow metody jsme zvolili 19 příznaků. Elbow metoda se používá pro shlukování, kde volíme největší zlom na grafu hodnoty, pro určení vhodného dělení.

Konkrétní podoba vybraných příznaků a další vybrané atributy jsou zaznamenány v Tabulce 2.3.

Velmi podobně konstruujeme histogram i pro další porty, tzv. **specifické porty**. Z portů, které jsme nevybraly jako obecné, vybíráme porty, které jsou významné specificky pro danou doménu zákazníka. Důležitost portu pro konkrétního zákazníka posuzujeme zcela stejně jako v případě portů obecných. Zde vybíráme 20 nejčetněji používaných portů. Specifické porty mohou být užitečné pro modely, které dokáží pracovat s konkrétní částí invariantně vůči pořadí (například sdílené neuronové sítě se sdílenými vahami). V opačném případě může být jejich využití spíše na škodu, jelikož se pak jedná o negeneralizovatelnou znalost.

Ze zbylých, nevyužitých portů ještě konstruujeme tři příznaky. Prostor portů můžeme rozdělit na tři intervaly: rezervované čísla portů v intervalu 0 - 1023, registrované porty v intervalu 1024 - 49150 a dynamicky přiřazované porty 49151 - 65535. Obzvláště poslední příznak může hrát významnou roli při klasifikaci, jelikož informaci o službách, které je využívají není možné zachytit jedním portem.

Protože se snažíme zařízení klasifikovat především podle služeb které poskytuje, spíše než jen



Obrázek 2.1 Graf zobrazující skóre přiřazené portům

podle čísel portů provádíme spojování portů, které odpovídají tytéž službě do jednoho. Takže například porty služby HTTP: 80, 8080, 8008, 8009 jsou považovány za ekvivalentní. V praxi to vypadá tak, že všechny záznamy náležící službě, které přišli na kterýkoliv z portů přičítáme jednomu, pro službu nejvíce typickému portu. Toto mapování je použito pouze pro porty souvisejícími s obecnými službami, neboť nejsme obecně schopni posoudit, které porty náleží stejné službě.

Výsledný vektor obsahuje 47 atributů, jejichž přesná podoba je vyobrazena v Tabulce 2.3.

Globální porty	Specifické porty	Agregované a ostatní příznaky
Port 80	Specifický port #0	Ostatní rezervované porty
Port 389	Specifický port #1	Ostatní registrované porty
Port 53	Specifický port #2	Ostatní dynamické porty
Port 443	Specifický port #3	Počet všech komunikací
Port 135	Specifický port #4	Ostatní porty
Port 1433	Specifický port #5	Zařízení v síti
Port 1900	Specifický port #6	Komunikací v síti
Port 137	Specifický port #7	Den v týdnu (0 = Pondělí)
Port 49152	Specifický port #8	
Port 60000	Specifický port #9	
Port 123	Specifický port #10	
Port 5000	Specifický port #11	
Port 67	Specifický port #12	
Port 8082	Specifický port #13	
Port 5431	Specifický port #14	
Port 5004	Specifický port #15	
Port 3911	Specifický port #16	
Port 3389	Specifický port #17	
Port 3128	Specifický port #18	
	Specifický port #19	

Tabulka 2.3 Přehled příznaků vektorové reprezentace prvku

Kapitola 3
Navržená algoritmická řešení

V předešlých kapitolách jsme představili několik technik strojového učení, zadefinovali naši úlohu, ukázaly několik přístupů využitých k jejímu řešení a prozkoumaly data a jejich různé možnosti interpretace. V této kapitole navrhneme vlastní metody ke klasifikaci zařízení, zda je/není server na základě zjištění z předchozích kapitol. Dále se zabýváme otázkou, jak detekovat služby, které dané zařízení poskytuje. S využitím těchto informacích o zařízeních navrhujeme algoritmus, který modeluje hypotetický výpadek na síti a odhaduje jeho důsledek na další zařízení.

Naši úlohu plníme ve třech krocích následujícím postupem:

1. Každé zařízení klasifikujeme, zdali je, či není server
2. Zařízením klasifikovaným jako server přiřazujeme služby, které poskytuje
3. Na základě detekovaných tříd odhadujeme případný dopad výpadku zařízení

Níže v této kapitole představujeme námi navržené přístupy pro splnění dílčích kroků. Ty jsou navrženy tak, aby překonaly výzvy, které dataset představuje. Mezi zásadní patří:

- Chybějící záznamy o komunikacích ze zařízení bez nainstalovaného softwaru pro sběr dat
- Software pro sběr dat vzorkuje komunikaci a důsledkem toho je zaznamenáno pouze prvních 30 komunikací procesu
- Veliké rozdíly mezi sítěmi jednotlivých zákazníků, které je třeba generalizovat
- Významně nevybalancované třídy zařízení v datasetu

Následující sekce se zabývají jednotlivými, výše popsanými kroky. Sekce 3.1 představuje navržené klasifikační algoritmy. Sekce 3.2 popisuje algoritmus pro detekci konkrétních služeb, které dané servery poskytují. Sekce 3.3 představuje navržený algoritmus, který pro hypotetický výpadek množiny serverů modeluje jeho dopady na ostatní zařízení a síť jako celek.

3.1 Algoritmy pro klasifikaci zařízení

Tato sekce se zabývá algoritmy pro klasifikaci jednotlivých zařízení na síti. Prvotní úlohou je určit, zda-li dané zařízení je, či není server. Za server považujeme zařízení, které poskytuje nějakou službu a jeho potenciální výpadek může zásadně ovlivnit fungování zbytku sítě. O těchto zařízení mluvíme jako o serverech a využíváme externě anotovaných datových sad pro učení a následně i vyhodnocení klasifikačních metod. Přesný popis dat využívaných k řešení této úlohy a jejich používané interpretace jsou popsány v Sekci 2.

Následující podsekce popisují jednotlivé algoritmy. Sekce 3.1.1 popisuje heuristický algoritmus, který klasifikuje zařízení na základě jejich vstupního stupně. Tento algoritmus využíváme jako baseline pro naše metody. Sekce 3.1.2 popisuje využití PageRanku pro klasifikaci zařízení. Sekce 3.1.3 popisuje nasazení modelu XGBoost pro naši úlohu.

3.1.1 Heuristická klasifikace

První testovanou metodou je jednoduchý heuristický algoritmus, který dále bude sloužit jako baseline pro srovnávání s dalšími přístupy.

Tento heuristický algoritmus posuzuje zda zařízení je či není server podle počtu vstupních komunikací. Na tento příznak můžeme nahlížet jako na vstupní stupeň zařízením reprezentovaného vrcholu v kontextu grafové reprezentace, kterou popisujeme v Sekci 2.1. Algoritmus zcela ignoruje ostatní atributy komunikací jako původce komunikace, port a další. Jednotlivé zařízení je tedy reprezentováno počtem vstupních komunikací a navrhovaný algoritmus porovnává tento atribut s naučenou mezní hodnotu. Zařízení je označeno jako server, právě tehdy když počet jeho příchozích komunikací překračuje zvolenou mez. Formálně zapisujeme:

$$y = \begin{cases} 1, & x \geq \text{thr} \\ 0, & x < \text{thr} \end{cases}$$

Takovýto algoritmus vyžaduje ke správnému fungování, aby data, která klasifikuje, byla dobře rozdělitelná mezní hodnotou na dvě množiny, kde jedna obsahuje servery a druhá ostatní zařízení. Vlastnost množiny, kterou můžeme rozdělit na pozitivní a negativní vzorky porovnáním s jednou hodnotou označujeme jako separovatelnost mezní hodnotou.

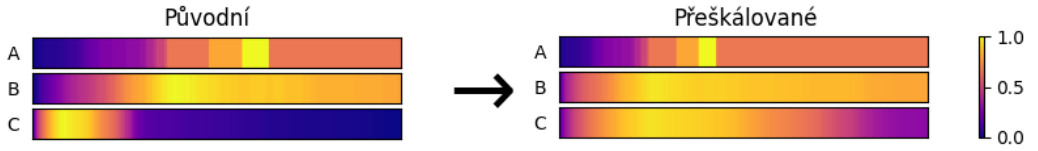
AUC křivka se běžně využívá pro vyhodnocení modelu, určeného k binární klasifikaci. Výstupem modelu bývá hodnota mezi 0 a 1 udávající pravděpodobnost příslušnosti k pozitivní třídě. AUC metrika vyjadřuje, jak dobře jsou tyto hodnoty seřazeny vůči vysvětlované proměnné a bez ohledu na konkrétní volbu klasifikační meze. Stejně tak s pomocí AUC metriky vyjadřujeme jak dobře je množina zařízení, reprezentovaná počtem příchozích komunikací, separovatelná mezní hodnotou (seřazená)[1].

Spočítáme-li takto AUC pro všechny dny každého zákazníka, zjistíme že nejnižší hodnota jaké nabývá je AUC = 0.993, což ukazuje na velmi dobrou separovatelnost zařízení mezní hodnotou. Zbývá najít její konkrétní hodnotu. Charakter sítí jednotlivých zákazníků se zásadně liší ve velikosti, vytížení samotných strojů, ale i například v topologii komunikace napříč sítí. Pro každou síť se tedy vhodná konkrétní hodnota meze může měnit. Zatímco na jedné síti může být zařízení s 50 příchozími komunikacemi nejvolanějším zařízením ze všech, na jiné síti nemusí být tento počet ani nadprůměrný. Z těchto důvodů nepřipadá v úvahu volit jako univerzální mez jakoukoli konstantní hodnotu.

První experiment, který jsme učinili bylo nastavení mezní hodnoty rovnou kvantilu p počtu komunikací zařízení. Tedy mezní hodnota je volena tak, aby dělila zařízení na množinu serverů obsahující $N/(1-p)$ nejvytíženějších (tím pádem pravděpodobně nejdůležitějších) zařízení a ostatní. To se může projevit jako nevýhoda přístupu, neboť tato proporcionalita dělení není nijak zaručena.

Další možností je volit mez na přeškálovaných datech. Přeškálování dat provádíme škálovací funkcí, která data upravuje na základě statistických vlastností celé množiny. Ideální škálovací metoda uvede různorodé sítě do jedné roviny, na níž bude možné zvolit mez univerzální napříč různými sítěmi, bez ohledu na velikost sítě a dalších nevhodných faktorů. Vhodné přeškálování je ilustrováno na Obrázku 3.1.

[1] Pro srozumitelnost můžeme jednotlivé vzorky přeškálovat, nicméně ROC křivka a tím pádem i AUC jsou vůči přeškálování invariantní



Obrázek 3.1 Ukázka přeškálování dat pro klasifikaci mezí. Hodnota udává F1 skóre pro danou mez. Vhodným přeškálováním je dosaženo přiblížení optimálních mezí napříč zákazníky. Takovou mez můžeme univerzálně aplikovat na nových, stejně přeškálovaných datech.

Používaným způsobem pro přeškálování dat je min-max normalizace, metoda která data z původního rozdělení zobrazí na interval $\langle 0, 1 \rangle$ pomocí vztahu:

$$x \leftarrow \frac{x - min(x)}{max(x) - min(x)}$$

Nevýhodou min-max normalizace může být citlivost na nejvyšší prvek.

Jednou z další možností, se kterou experimentujeme je standardizace. Ta škáluje data do standardního rozdělení $N(0, 1)$ vztahem:

$$x \leftarrow \frac{x - \bar{x}}{\sigma}$$

kde $\bar{x}$ je výběrový průměr a σ směrodatná odchylka.

3.1.2 PageRank

Algoritmus PageRank [10], určený pro posouzení důležitosti webových stránek, jsme nasadili i na naši úlohu. Síť webových stránek můžeme reprezentovat jako graf, kde stránka představuje vrchol a hypertextový odkaz na stránce představuje hranu mezi patřičnými dvěma vrcholy. Na takovém grafu PageRank přiřazuje každé stránce důležitost na základě důležitosti stránek, které na ni odkazují. Grafová reprezentace naší úlohy (Sekce 0.3) není nepodobná grafu webových stránek. A stejná úvaha o důležitosti vrcholů ukazujících na jiné vrcholy je na místě i zde.

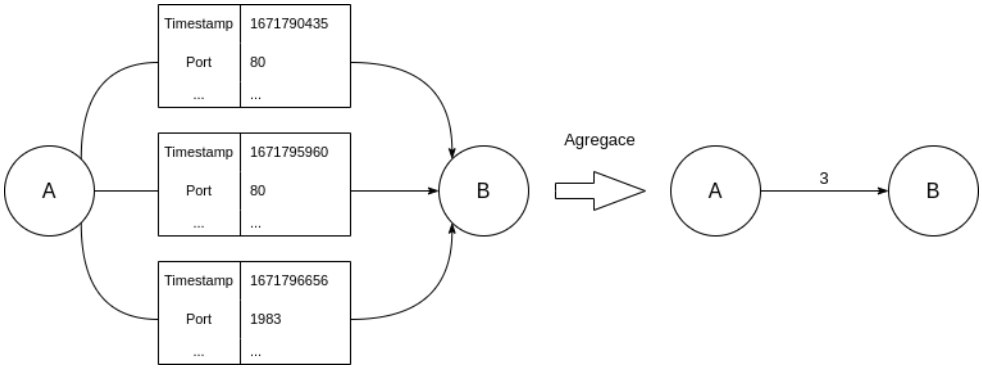
Algoritmus PageRank aplikujeme na grafovou reprezentaci úlohy, jak je popsaná v Sekci 2.1. Na rozdíl od běžného grafu reprezentujícího webovou strukturu, je náš graf multigrafem - mezi libovolnou dvojicí vrcholů může existovat vícero hran. Jak bylo zmíněno v Sekci 2.1, jednotlivé hrany multigrafu nesou informace o komunikaci jako je čas jejich odeslání, číslo portů a pod. Tyto informace při použití PageRanku zanedbáváme a využíváme samotného počtu volání. Agregujeme všechny hrany vedoucí z jednoho vrcholu do druhého do jediné hrany s váhou odpovídající jejich počtu. To je znázorněno na Obrázku 3.2. Tak vzniká nová reprezentace sítě obyčejným orientovaným grafem s váženými hranami.

Původní algoritmus popsaný v Sekci 0.3 nahrazujeme verzí PageRanku, která umí zužitkovat i váhy jednotlivých hran. Jedinou úpravou je záměna definice matice přechodu:

$$\pi_k = \pi_{k-1} P, \quad P_{ij} = \begin{cases} \text{Váha } P_i \to P_j / \sum_k \text{Váha } P_i \to P_k & \text{existuje-li hrana z } P_i \text{ do } P_j \\ 0 & \text{jinak} \end{cases}$$

tedy váhy jsou normalizovány, aby matice P stále byla stochastická, ovšem pravděpodobnost výběru hrany, kterou odejde náhodný chodec z vrcholu je úměrná její váze.

Je třeba mít na paměti, že PageRank není přímo určený pro klasifikaci, nýbrž je určený pro úlohu řazení vrcholů podle důležitosti. Výstupem algoritmu je vektor stacionárního rozdělení, který jednotlivým zařízením přiřazuje relativní důležitost v porovnání s ostatními. Abychom zařízení vektoru relativních důležitostí mohli klasifikovat, všechny jeho hodnoty standardizujeme a poté vybíráme klasifikační mez podobně jako v předchozí Sekci 3.1.1.



Obrázek 3.2 Agregace hran multigrafu pro použití PageRankem

Dále navrhujeme variantu s využitím personalizace PageRanku, rovněž představené v Sekci 0.3. K určení personalizačního vektoru využíváme jiný z klasifikátorů navržených v této kapitole. Personalizační vektor je roven normalizované predikci klasifikátoru. Tedy pravděpodobnost teleportace náhodného chodce do vrcholu je uniformně rozdělena mezi všechny zařízení klasifikované jako servery, zatímco ostatní mají nulovou pravděpodobnost. Výsledný, kombinovaný model pak může najít další zařízení, se kterým není tak často komunikováno, ale je kritické pro zachování funkcionality serveru zařízení. Příkladem takového zařízení může nadřazený server, z něhož si podřízené servery aktualizují databázi jen jednou za několik hodin.

3.1.3 XGBoost

Jako další metodu pro otestování jsme zvolili XGBoost, model představený v Sekci 0.2. Algoritmus se osvědčil jak v obecné klasifikaci tak i v mnoha předešlých přístupech v úloze klasifikace IoT zařízení. Model XGBoost jsme trénovali na vektorové reprezentaci dat představené v Sekci 2.2.

3.2 Detekce zařízením poskytovaných služeb

V této sekci se zabýváme druhým krokem naší úlohy a tím je detekce služeb, které poskytují jednotlivá zařízení, která jsme v prvním kroku označili za servery. Toto posouzení je klíčové pro monitoring a správu sítě, neboť díky němu získává správce mnohem lepší přehled o zařízeních i službách. Také lze na jejich základě určit riziko potenciálního výpadku zařízení a určit zařízení, která jím budou ovlivněny a jak.

Snažíme se detekovat na zařízeních konkrétní služby. Ovšem jedním z našich požadavků je, aby náš přístup byl obecně využitelný na jakékoliv síti. Obě tyto zásady lze zároveň splnit jenom velmi těžko, neboť na síti může existovat nepřeberné množství služeb, lišící se chováním. Například vykazuje-li zařízení velmi častou komunikaci na portu 53, je zřejmé, že poskytuje DNS službu. Pokud ale bude zařízení často komunikovat na portu, který není jednoznačně přiřazený autoritou IANA [18], pak nelze obecně přiřadit službu bez externího posouzení. To se v našich datech odehrává velmi často. Proto podobně jako v Sekci 2.2 si dovolujeme mluvit o službách a portech jako o dvou zaměnitelných věcech. Tedy pro zařízení namísto jmenovitých služeb detekujeme, na kterých portech významně komunikuje. Například výše zmíněný DNS server označujeme za zařízení, které poskytuje službu na portu 53.

Jedním z možných problémů při zaměňování čísel portů a služeb je již dříve diskutovaný fakt, že některé služby mohou využívat vícero různých čísel portů. I zde pro obecně známé služby jako je například HTTP považujeme čísla portů s nimi souvisejícími za identické (port 80, je ekvivalentní portům 8080, 8008, ...). Přesto je třeba mít na paměti, že na síti se může objevit jakákoliv služba využívající vícero portů a obdobné propojení těchto portů je mimo rámec práce.

V případě, že je ekvivalence portů v konkrétní síti zásadní, může správce toto spojení zadefinovat manuálně.

Možným zlepšením přístupu by bylo přidat sumu příchozích komunikací zařízení na dynamická čísla portů jako samostatný příznak, který reprezentuje službu (popřípadě vícero služeb) komunikující na dynamických portech. Tento příznak je pak možné škálovat stejně, jako kterékoliv jiné číslo portu. Průzkum dat ale ukázal, že některé služby aktivně využívají porty z dynamického rozsahu. Příkladem je port 60000, který je významně volán napříč záznamy vícero dostupných sítí.

Teoreticky by pro detekci poskytovaných služeb mohlo stačit vybrat porty, na nichž zařízení komunikuje. V praxi však zařízení přijímá veliké množství komunikací na různých portech a nenabízí na každém portu nějakou službu. To platí pro mnoho portů z dynamického rozsahu, ale i pro některé registrované porty. Například porty 137 (NetBios), 22 (SSH) nebo 389 (Active directory) jsou velmi často volané porty zákazníka Z7, nejen na serverech. Z toho plyne, že pokud zařízení nepřijímá výjimečně vyšší počet komunikací na portu než ostatní zařízení, dá se předpokládat, že na něm neposkytuje službu. Z toho důvodu se blíže zabýváme relevancí služby, spíš než samotnými počty volání.

Na relevanci služby v vůči zařízení můžeme nahlížet ze dvou pohledů. První je z pohledu zařízení. Tedy chceme určit, které služby jsou pro dané zařízení nejvýznamnější v porovnání s ostatními. Tato informace je užitečná v rámci monitoringu sítě, kde chceme získat představu o účelu zařízení.

Druhou možností je zabývat se zařízením poskytovanou službou s ohledem na celou síť. V tomto případě nás bude zajímat, jak moc ostatní prvky sítě využívají službu daného zařízení. Tuto informaci je možné využít k analýze rizik spojených s výpadkem daného zařízení.

V následujících přístupech bereme v potaz oba tyto pohledy na problematiku a hledáme kompromis mezi nimi. Relevanci služby i pro zařízení j udáváme jako $rel_{i,j}$.

Při posuzování významnosti služby na zařízení postupujeme pro každé použité číslo portu na síti samostatně. Porovnáváme, jak často byl daný port na zařízení volán v porovnání se zbytkem sítě. Problémem je, že pro každou službu u každého zákazníka může pojem "často volaný port" představovat zcela jiné číslo. To zobrazuje Obrázek 4.7 vlevo, kde je vidět srovnání histogramů volání portů 5004 a 80. Z toho důvodu prvně počty volání portu škálujeme. Díky tomu bude záznam vztahující se ke kombinaci portu a zařízení relativní vůči zbytku sítě, případně vůči počtům volání na ostatních portech daného zařízení.

V následujících podsekcích nabízíme vhled do možností přeškálování dat, vhodnou pro obecnou detekci libovolné služby.

3.2.1 Standardizace

První pokus o přeškálování dat do stejné roviny jsme učinili s užitím standardizace. Standardizace škáluje data z původního rozdělení do $N(0, 1)$ pomocí funkce:

$$rel_{i,j} = x \leftarrow \frac{x - \bar{x}}{\sigma}$$

kde $\bar{x}$ a σ jsou průměr a směrodatná odchylka volání služby i.

Histogram portů běžných služeb (klienti kontaktují server) mívá obvykle většinu zařízení blízkou nule a poté malou skupinu zařízení s výrazně vyšším počtem volání. Vlivem standardizace je málo volaná většina přeškálována do hodnot těsně podprůměrných - tedy velmi nízká záporná čísla, zatímco menšina často komunikujících zařízení je přeškálována do vyšších hodnot.

Využitím standardizace můžeme označit servery, které jsou outliery za pomocí meze. Zařízení, které danou mez překročí budeme považovat budeme považovat za relevantní k danému portu a přiřadíme jim ho.

Jedním z problémů, který vzniká vlivem standardizace je ztráta informace o absolutních počtech histogramu. Po aplikování standardizace se může stát, že sítí minimálně využívané číslo

portu (nízké desítky volání), nepředstavující žádnou nabízenou službu se projeví na nějakém zařízení jako důležitá služba, ačkoliv počet volání je v řádu jednotek. To se může stát pro dynamicky zvolený port z dynamického rozsahu 49151 - 65535, který bude mít na síti jednotky volání, nebo různé velmi vzácné služby navazované uživatelem (například port 7 - ECHO). Abychom se vyvarovali zařazení těchto portů mezi významné porty daného zařízení, klademe následující podmínku. Port musí být volán alespoň v 1% příchozích komunikací zařízení a musí jich proběhnout více než 30 komunikací. Procentuální počet je zahrnut kvůli velkým sítím, kde servery přijímají řádově více požadavků. Absolutní počet potlačuje chybnou detekci služby zařízení na malé síti, kde by procentuální počet mohl odpovídat i jednotkám volání.

V praxi, využíváme-li standardizaci jako škálování při klasifikací mezí, není potřeba data samotná standardizovat. Namísto toho můžeme na zvolenou mez aplikovat inverzi standardizace a jí poté klasifikovat původní příznaky. Tento přístup je ekvivalentní samotné standardizaci, nicméně je méně výpočetně náročný než standardizace vstupních stupně zařízení. Pro vybranou síť můžeme upravit mez následujícím způsobem:

$$thr = thr_{std} * \sigma + \bar{x}$$

3.2.2 L1 Normalizace

Posledním navrženým přeškálováním je L1 normalizace. Ta relevanci služby vůči zařízení definuje vztahem:

$$rel_{i,j} = \frac{x_{i,j}}{\sum_k x_{i,k}}$$

kde $x_{i,j}$ udává počet přijatých komunikací služby (na portu) i zařízením j. Data přeškálovaná L1 normalizací přímo udávají, jak velký poměr volání služby i bylo obslouženo zařízením j.

Pokud $rel_{i,j}$ dosahuje čísla blízkého jedné, je pravděpodobné, že zařízení je jediné, které danou službu poskytuje. Díky tomu můžeme detekovat zařízení, jejichž výpadek s sebou přinese i výpadek služby na celé síti.

Pokud $rel_{i,j}$ je rozdělena mezi více zařízení, pak se riziko odvíjí od konkrétního využívání služby. Například: jestliže jsou na síti dvě zařízení poskytující služby na portu 80 (HTTP), nelze z dostupných dat posoudit, jestli výpadek zařízení způsobí i výpadek dané služby. Na zařízeních totiž mohou být různé, ale i stejné služby.

Zásadní výhodou L1 normalizace je její snadná interpretabilita bez ohledu na síť. Stejně jako standardizace však trpí ztrátou informace o absolutních počtech a může tak zařízení významně přisuzovat službu, která na něm byla minimálně volána.

3.3 Odhad dopadu výpadku zařízení

V prvním kroku jsme ze všech zařízení vybrali servery, Sekce 3.1. Pro ně jsme v druhém kroku detekovali relevantní služby, Sekce 3.2. V této sekci se zabýváme posledním krokem a tím je návrh algoritmu pro posouzení škody, kterou potenciální výpadek daných zařízení způsobí. Navrhujeme algoritmus, který označuje zařízení, postižená potenciálním výpadkem.

Výpadkem zařízení se zabýváme z pohledu každé, jím poskytované služby samostatně. V první řadě se ptáme, kolik zařízení danou službu poskytuje. Tím zjišťujeme, zda-li existuje jiné zařízení, které může zařízení s výpadkem pro danou služby zastoupit. Jestliže taková zařízení existují, pak není zařízení pro službu považováno za kritické, ovšem v opačném případě přijde celá síť o možnost využívání dané služby.

Kvůli datům, které máme k dispozici, mluvíme v naší práci o všech službách, které jsou poskytovány na daném portu jako o jedné. V praxi se však může na jednom portu v síti objevit vícero různých služeb. Například se na síti může vyskytovat vícero serverů, které poskytují službu na portu 80. Ty mohou, ale nemusí poskytovat stejnou službu. Jedním z možných přístupů, jak toto posoudit, je detekovat load balancing v síti. Síť posuzujeme na základě časového okna a

protože server poskytující službu mohl kdykoliv v tomto okně začít, nebo přestat být aktivní, nestačí pro detekci load balancingu srovnávat celkový počet volání. Mohli bychom toho však dosáhnout analýzou distribuce komunikací koncových zařízení vůči zařízením poskytujícím službu na daném portu. Tato úloha je však na rozsáhlých počítačových sítích náročná a blíže se jí nezabýváme.

Výpadek služby může mít na síť širokou škálu účinků v závislosti na tom, o kterou konkrétní službu se jedná. Některé služby jsou známé jako kritické pro jakoukoliv síť. Pokud například síť přijde o službu DNS, pak budou koncoví uživatelé jen těžko přistupovat k ostatním zařízením. Obecně je však u služeb těžko odhadnutelné, jaký dopad bude mít jejich výpadek na síť. Například HTTP služba na portu 80 může být kritická pro síť, jestliže je důležitou součástí nějakého byznys procesu, ale stejně tak se může jednat o jinou službu, jejíž výpadek zbytek sítě neovlivní.

Při výpadku zařízení dále mluvíme o dvou typech následků na službu. Jestliže dojde k výpadku všech zařízení, která na portu x poskytuje službu, pak jej nelze zastoupit a dojde k *nutnému výpadku služby x*. Pokud ale dojde k výpadku pouze části zařízení poskytujících služby na portu x, nejsme obecně schopni říci, jestli je zbylé zařízení zastoupí. Proto mluvíme o *potenciálním výpadku služby x*. Můžeme zde definovat čísla portů, které vždy poskytují stejnou služby, tudíž se vždy mohou vzájemně zastoupit a potenciální výpadek je tak pro ně irelevantní. Například DHCP a DNS.

Navrhujeme algoritmus, který pro výpadek posoudí, která zařízení ovlivní a přisoudí jim služby, které pro ně nebudou dostupné. Náš algoritmus se neomezuje na analýzu výpadku jediného zařízení, ale modeluje dopad výpadku jakékoliv množiny zařízení.

Náš přístup popisuje pseudokódem Algoritmus 1. Vstupem je množina zařízení, která prochází hypotetickým výpadkem. Dále je možné definovat kriticky důležité služby. Jeho výstupem jsou seznamy zařízení, které udávají, jaká zařízení přestanou fungovat, která potenciálně mohou přestat fungovat, ale nejsme to schopni blíže posoudit, a seznam služeb které přestanou být na síti dostupné.

Algoritmus nejprve určuje, u kterých služeb dojde k výpadku na celé síti. Říkáme, že k výpadku služby dojde, pakliže 95% komunikací dané služby směřovalo na nově nefunkční zařízení. Jestliže je určena jako kriticky důležitá, pak předpokládáme, že způsobí výpadek všech zařízení, která je na prvotní množině výpadku volala. Pokud se ukáže, že některé důležité zařízení nově postrádá kriticky důležitou služby, pak ho přidáváme do množiny, pro níž výpadek simulujeme a propagujeme výpadek dále.

Zařízení která volala služby, jejichž výpadek není považován za kritický označujeme za potenciálně nefunkční.

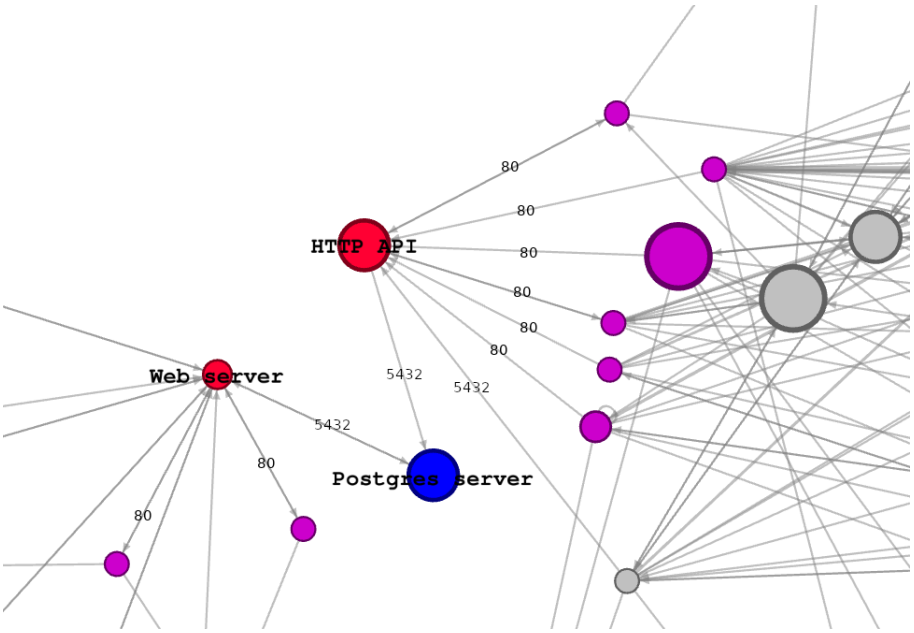
Služby pro které dojde k výpadku jen části zařízení, jež na stejných portech provozují služby považujeme za potenciálně nefunkční a stejně tak i zařízení, která je volala, neboť nejsme schopni posoudit, která konkrétní služba byla portem představována.

Na Obrázku 3.3 ilustrujeme fungování algoritmu 1.

Výstup tohoto algoritmu můžeme využít nejen k identifikaci rizik spojených s výpadkem uzlů. Počet ovlivněných zařízení, počet zařízení která přestanou fungovat, počet serverů které přestanou fungovat, to vše můžeme využít jako metriku pro důležitost zařízení. Pokud výpadek uzlu ovlivní veliký počet zařízení, nebo způsobí výpadek nějaké kritické služby pak jej označujeme za kritický uzel.

```
   input  : mnozina_vypadek, kriticke_sluzby
   output: jisty_vypadek_zar, potencialni_vypadek_zar, vypadek_sluzba
 1 Function posouzeni_dopadu_vypadku(mnozina_vypadek, kriticke_sluzby):
 2     jisty_vypadek_zar ← {}
 3     potencialni_vypadek_zar ← {}
 4     vypadek_sluzba ← []
 5     port_podil ← {}
 6     pribyl_jisty_vypadek_server ← False
 7     for server ∈ mnozina_vypadek do
 8         for port ∈ server.poskytovane_sluzby () do
 9             port_podil[port] += port.l1_norma()
10             ovlivnena_zar ← server.zar_volajici_sluzbu(port)
11             potencialni_vypadek_zar.insert(ovlivnena_zar)
12         end
13     end
14     for (port, l1_suma) ∈ port_podil do
15         if l1_suma > 0.95 then
16             vypadek_sluzba.append(port)
17         end
18     end
19     for port ∈ vypadek_sluzba do
20         ovlivnena_zar ← mnozina_vypadek.zar_volajici_sluzbu(port)
21         je_kriticka_sluzba ← (port ∈ kriticke_sluzby)
22         if je_kriticka_sluzba then
23             for zar ∈ ovlivnena_zar do
24                 jisty_vypadek_zar.insert(zar)
25                 if zar.je_server () & zar ∉ mnozina_vypadek then
26                     mnozina_vypadek.append(zar)
27                     pribyl_jisty_vypadek_server ← True
28                 end
29             end
30         end
31     end
32     if pribyl_jisty_vypadek_server then
33         return posouzeni_dopadu_vypadku(mnozina_vypadek, kriticke_sluzby)
34     end
35     return
         jisty_vypadek_zar, potencialni_vypadek_zar \ jisty_vypadek_zar, vypadek_sluzba
36 posouzeni_dopadu_vypadku(pocatecni_mnozina_vypadek, kriticke_sluzby)
```

Algoritmus 1: Navržený algoritmus pro odvození důsledků výpadku



Obrázek 3.3 Graf ilustrující aplikaci Algoritmu 1. Služba 5432 byla označena za kritickou pro fungování sítě a pro modré zařízení modelujeme výpadek. Červená zařízení byla přidána k modelovanému výpadku coby nefunkční, jelikož se jedná o servery, které ztratili přístup ke kriticky důležité službě. Fialová zařízení představují potenciální výpadek.

Kapitola 4
Výsledky experimentů

Tato kapitola dále navazuje na přístupy představené v Sekci 3.1. Ty evaluujeme na datech popsaných v Kapitole 2. Cílem této kapitoly je popsat hledání vhodných nastavení jednotlivých modelů, aby jejich fungování bylo na datech optimální. Dále se zaměřujeme na porovnání jednotlivých přístupů nejen v metrikách, ale srovnáváme i vhodnost jejich použití v různých podmínkách.

Sekce 4.1 se zabývá jednotlivými modely pro klasifikaci zařízení, jejich variantám, vlastnostem, dosahujícím výsledkům a vlastnostmi, kterými disponují. Sekce 4.2 vyhodnocuje následující krok naší úlohy. Tou je detekce služeb poskytovaných zařízením.

4.1 Výsledky klasifikace zařízení

Tato sekce popisuje učení, vyhodnocení a vlastnosti algoritmů, představených v Sekci 3.1.

4.1.1 Heuristická klasifikace

Baseline model popsaný v Sekci 3.1.1 je velmi jednoduchý heuristický algoritmus pro klasifikaci zařízení, zda je či není server. Pro posouzení zařízení využívá algoritmus jediný příznak a to počet příchozích komunikací. Jeho podstata stojí na myšlence, že servery přijímají podstatně více komunikací, než ostatní.

Algoritmus vybírá mez, která má nejlépe rozdělovat zařízení na servery a ostatní. V Sekci 3.1.1 jsme mluvili o problému různorodosti jednotlivých sítí, který ztěžuje hledání univerzální meze vhodné pro kteroukoliv síť. Tento problém jsme se snažili potlačit určením meze pomocí kvantilu. Dále jsme hovořili o možnostech přeškálování dat. S použitím těchto metod vytváříme nové příznaky se shodnými charakteristikami napříč dny i zákazníky, na nichž hledáme univerzální konstantní mez.

V této sekci se zaměřujeme na vyhodnocení kvality zmíněných přístupů a na jejich srovnávání.

4.1.1.1 Heuristická klasifikace - Kvantil

První přístup určování meze, který jsme podrobili experimentování je pomocí kvantilu. Obrázek 4.1, první řádek nalevo zobrazuje ROC křivku jednotlivých zákazníků. ROC křivka zákazníka je spočítána na zařízeních ze všech dnů (tedy jedna IP adresa se může vyskytnout až sedmkrát). “Průměr pro společné meze” počítá metriky všech zákazníků na všech mezích a pro každou mez je její hodnota rovna průměru metrik. Vidíme zde, že správnou volbou meze pomocí kvantilu

Zákazník	Kvantil, m. h. = 0.977		
	Precision	Recall	F1
Z1 (malý)	0.700	0.875	0.777
Z2 (malý)	1.000	0.875	0.933
Z3 (malý)	0.583	1.000	0.736
Z4 (malý)	0.483	1.000	0.652
Z5 (velký)	0.971	0.555	0.706
Z6 (velký)	0.933	0.307	0.462
Z7 (velký)	0.285	0.954	0.439
Průměr malých	0.691	0.937	0.775
Průměr velkých	0.730	0.605	0.536
Průměr všichni	0.708	0.795	0.672

Tabulka 4.1 Výsledky klasifikace heuristické klasifikaci pomocí kvantilu

je možné pro každou jednotlivou síť zákazníka zvolit mez, která bude data klasifikovat s velmi dobrými výsledky.

Obrázek 4.1, první řádek napravo zobrazuje, jakého F1 skóre dosáhne algoritmus, použijeme-li jako mez předepsaný kvantil zákaznických zařízení. Za povšimnutí stojí, že ačkoliv pro samostatné sítě dosahovala mez výtečných výsledků, není zde konkrétní hodnota kvantilu, která by tak dobře klasifikovala zařízení napříč všemi sítěmi.

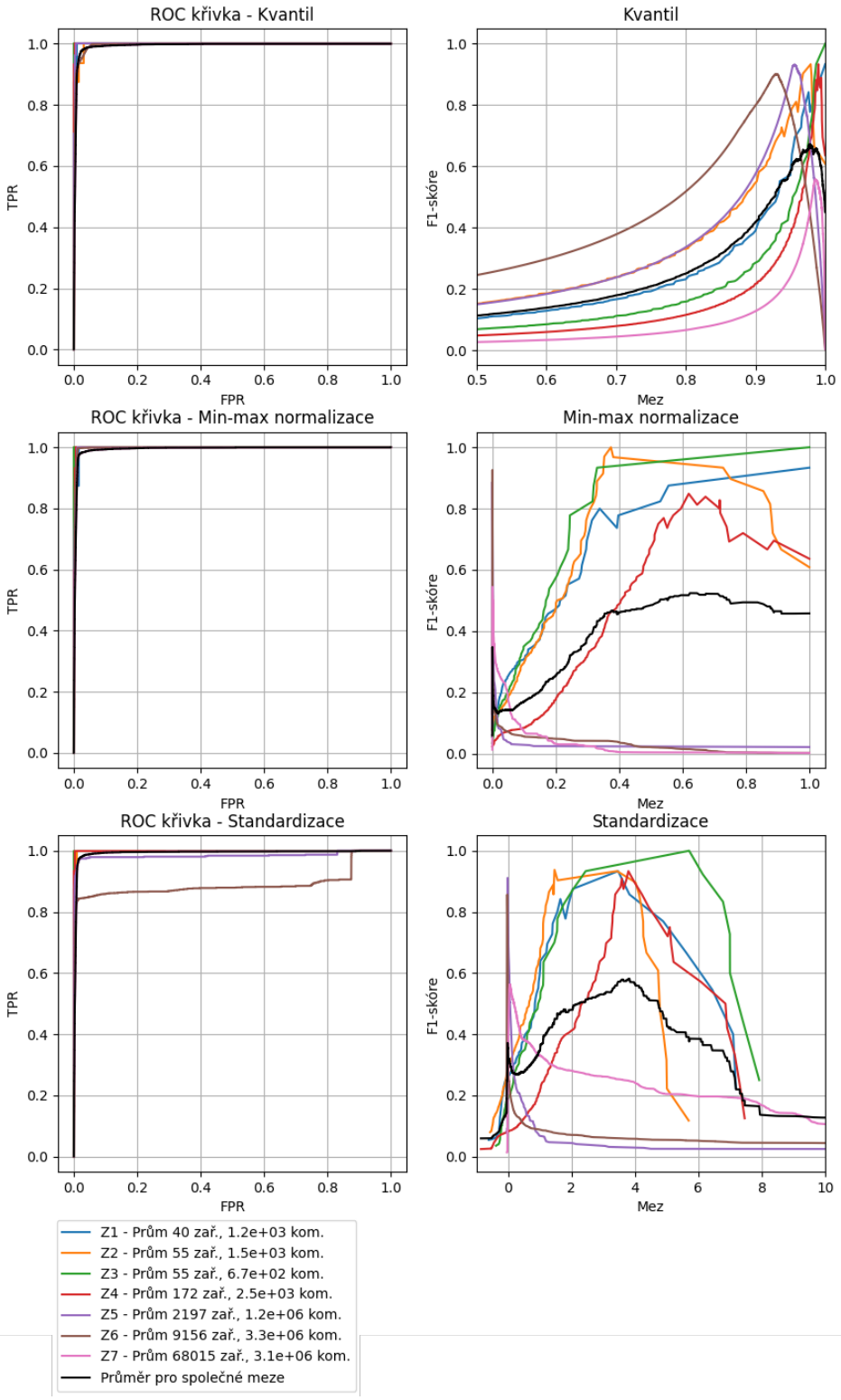
Jako mezní hodnotu volíme 0.977, jež měla nejvyšší průměrné F1-skóre napříč zákazníky. Vyhodnocení metrik pro danou konfiguraci je zobrazeno v Tabulce 4.1. Zde si všímáme, významného rozdílu metrik mezi malými a velkými sítěmi. Zařízení na velkých sítích jsou klasifikovány podstatně hůř, než na malých. Kdyby chyba byla systematický, tedy kdyby všechny velké sítě vyžadovali změnu mezní hodnoty jedním směrem, mohli bychom vyhodnotit zvláštní hodnotu pro obě skupiny sítí. Obrázek 4.1 i Precision a Recall v Tabulce 4.1 však ukazují, že tomu tak není.

Heuristický algoritmus celkově dobře klasifikuje zařízení. Jeho zásadními nevýhodami jsou předpoklad, že všechny sítě mají stejný poměr serverů a ostatních zařízení a neschopnost přizpůsobit se jiným situacím. Vidíme, že tento poměr v praxi nenastává. To v případě velkých sítí může mít za následek vysokých počet chyb prvního i druhého řádu, které pak musí administrátor manuálně řešit.

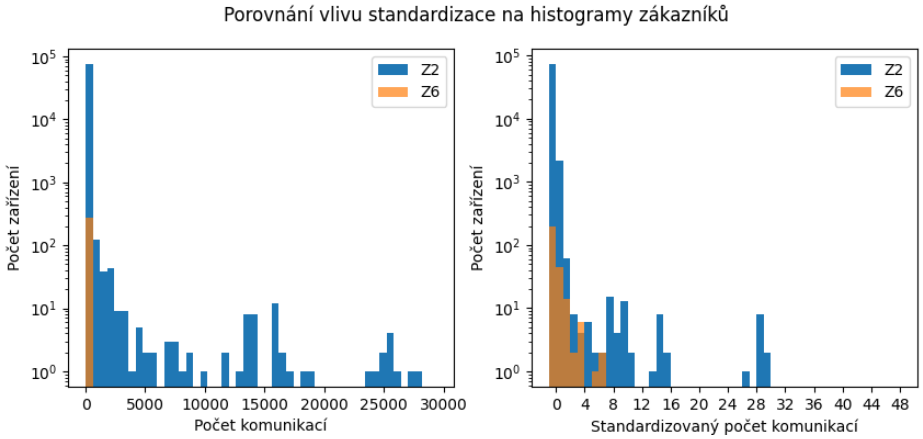
4.1.1.2 Heuristická klasifikace - Min-max normalizace

Jak bylo zmíněno, snažíme se problém rozdílných velikostí sítí řešit přeškálováním příznaku. Druhý a třetí řádek Obrázku 4.1 zobrazuje charakteristiky přístupů využívající přeškálování. Přeškálování určujeme pro každý den každého zákazníka samostatně, stejně jako jsme to dělali v případě kvantilu. Tedy pro jediného zákazníka jsou data každého dne přeškálována funkcí s jinými parametry, ale data ze všech dnů zákazníka jsou poté spojena dohromady a je k nim přistupováno jako k celku, pro který hledáme jedinou mez.

První navrženou škálovací funkcí byla min-max normalizace. Její charakteristiky jsou zachyceny na Obrázku 4.1, druhý řádek. Vidíme, že ROC křivka, stejně jako v předchozím případě, říká, že existuje pro každého samostatného zákazníka vhodná mez. Ovšem přeškálováním jsme nedosáhly kýženého výsledku. To je vidět v pravé části druhého řádku Obrázku 4.1, kde vidíme jen velmi malý překryv hodnot mezí, které dosahují lepších výsledků. Bližší zkoumání dat ukázalo, že min-max normalizace je zde příliš citlivá na konkrétní hodnotu nejvyššího vstupního stupně zařízení. Tato hodnota se může každý den značně měnit, ale neodráží vlastnost velikosti sítě jako celku. Následující přístup škálování se podle grafů jeví jako lepší i robustnější volba, proto min-max normalizaci dál blíže nezkoumáme.



Obrázek 4.1 Grafy zobrazující ROC křivky a závislost F1 skóre na hodnotě mezi navržených metod pro jednotlivé zákazníky



Obrázek 4.2 Grafy zobrazující vliv standardizace na sítě rozdílných velikostí

4.1.1.3 Heuristická klasifikace - Standardizace

Druhou možností škálování dat je standardizace. Její vliv můžeme vidět na posledním řádku Obrázku 4.1. Stejně jako v předchozích případech můžeme na ROC křivce vidět, že pro jednotlivé zákazníky existuje vhodná mez. Také je vidět pokles pro zákazníka Z6, důvody jsou blíže popsány níž.

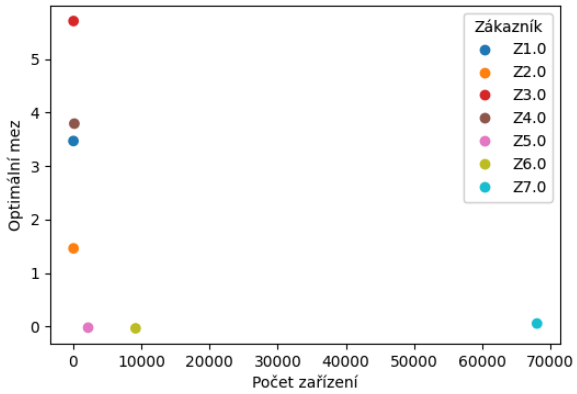
Na grafu vpravo vidíme, že křivky jsou vizuálně podobné min-max normalizaci, ovšem překryv lepších mezí je o něco větší.

Taky si na Obrázku 4.1 všímáme, že můžeme zákazníky rozdělit na dvě skupiny a to při použití kterékoliv ze zmíněných metod přeškálování. První skupina, zákazníci Z1, Z2, Z3, Z4 má vhodnou mez pohybující se okolo 4, zatímco druhá skupina zbývajících zákazníků Z5, Z6, Z7 má vhodnou mez okolo nuly. Zásadním rozdílem, který tyto zákazníky také rozděluje je velikost sítě. První skupina obsahuje menší sítě, druhá větší. Vliv standardizace na histogram různě velkých zákazníků můžeme vidět na Obrázku 4.2. Na malých sítích jsou si všechna zařízení počtem komunikací vcelku blízko a servery se neliší tak drasticky v počtu volání, neboť na síti je málo zařízení. V případě velkých sítí je tomu ale naopak. Na síti jsou služby, které používá každé, nebo většina zařízení, například DNS. S rostoucím počtem zařízení významně roste počet komunikací takových serverů, protože každý nový prvek volá hned několik služeb. To způsobuje strmý růst průměrného počtu volání i jeho standardní odchylky, na nichž stojí standardizace. Výsledkem je, že většina běžných, méně volaných zařízení je přeškálováno blízko nule, zatímco servery jsou přeškálovány do vysokých čísel (větší než 4).

Jedna z možností, jak tento problém řešit je použití odlišných mezí pro každou skupinu. Tím však vyvstává otázka, k jaké skupině bychom měli přiřadit, resp. jakou mezí bychom měli dělit novou neznámou síť? Na určení vhodného dělení nám schází dostatečné množství anotovaných zákaznických sítí ve vícero velikostních rozsazích, ale ani neexistuje záruka, že je lze opravdu rozdělit na dvě skupiny jak dostupná data naznačují.

Vhodná mez by se od zmíněných 4 první skupiny k 0 druhé, větší skupiny mohla přibližovat spojitě se zvyšujícím se sítěmi, ovšem tuto tezi nemůžeme potvrdit ani vyvrátit neboť rozdíly v počtech zařízení i komunikací jsou mezi skupinami řádově odlišné a nelze je rozumně interpolovat. Zkoumali jsme počet zařízení na sítích, ale nepodařilo se nám najít přímý vliv na vhodnou mez; Obrázek 4.3.

Z těchto důvodů volíme pro obě skupiny sítí odlišné mezní hodnoty. Ty jsme volili jako optimální průměrné F1-skóre skupiny. Pro velké sítě volíme mez -0.02 a pro malé 3.467. Tabulka 4.2 zobrazuje vyhodnocené metriky pro obě skupiny. Mezní hodnota 3.467 malé skupiny predikuje velice dobře, ovšem pro větší dle předpokladu selhává. Bohužel ani mezní hodnota -0.02 určená



Obrázek 4.3 Graf zobrazující závislost optimální mezi (nejvyšší F1-skóre) jednotlivých zákazníků na počtu zařízení

Zákazník	Stand., m. h. = -0.02			Stand, m. h. = 3.467		
	Precision	Recall	F1	Precision	Recall	F1
Z1 (malý)	0.154	1.000	0.267	1.000	0.875	0.933
Z2 (malý)	0.165	1.000	0.283	1.000	0.812	0.897
Z3 (malý)	0.106	1.000	0.192	1.000	1.000	1.000
Z4 (malý)	0.043	1.000	0.082	0.789	1.000	0.882
Z5 (velký)	0.896	0.917	0.906	1.000	0.014	0.028
Z6 (velký)	0.938	0.641	0.762	0.958	0.031	0.060
Z7 (velký)	0.056	0.995	0.107	0.911	0.147	0.253
Průměr malých	0.117	1.000	0.206	0.947	0.922	0.928
Průměr velkých	0.630	0.851	0.592	0.956	0.064	0.114
Průměr všichni	0.337	0.936	0.371	0.951	0.554	0.579

Tabulka 4.2 Výsledky klasifikace heuristického algoritmu se standardizací

pro velké sítě nedosahuje příliš dobrých výsledků. Ačkoliv se na Obrázku 4.1 jevila jako vhodná mez pro větší sítě, bližší přezkoumání ukázalo, že ostré výkyvy, které se vyskytují na křivce všech sítí se vzájemně nepřekrývají u všech sítí. To má za následek, že mez volená pro danou skupinu klasifikuje skvěle sítě Z5 a Z6, ale velmi špatně síť Z7.

Heuristický algoritmus využívající standardizaci, stejně jako předchozí přístupy, nebyl schopný dobře přeškálovat záznamy tak, abychom mohli ke všem sítím přistupovat se stejnou mezní hodnotou. Navzdory tomu jsme určili mezní hodnoty pro každou skupinu zvlášť a upozornili na úskalí tohoto řešení. Pro skupinu menších sítí jsme zvolili mez, která pro ni dosahuje velmi dobrých výsledků. Ovšem u velkých sítí jsme narazili na stejný problém jako v případě kvantilu, tedy neexistenci vhodné společné meze, ačkoliv Obrázek 4.1 naznačoval opak. Proto preferujeme využití algoritmu využívající kvantil, neboť dosahuje robustnějších výsledků napříč všemi zákazníky a není zatížen problémem zařazování neznámého zákazníka do skupiny.

4.1.2 PageRank

Teoretické podklady algoritmu PageRank a jeho využití pro řazení důležitosti stránek jsme představili v Sekci 0.3. Na to jsme navázali v Sekci 3.1.2, kde popisujeme jak aplikujeme PageRank pro klasifikaci zařízení na síti reprezentované grafovou reprezentací dat, která je popsaná

v Sekci 2.1. V této sekci představujeme výsledky, jichž navrhovaná řešení dosahují.

Algoritmus PageRank, ačkoliv je původně navržený pro využití u webový stránek a dokumentů, našel uplatnění napříč mnoha úlohami, jejichž data lze popsat orientovaným grafem [9]. Tak lze popsat i naše úloha, kde zařízení představují vrcholy a komunikace jejich vrcholy, Sekce **??**. Jak bylo popsáno v Sekci 0.3, PageRank je v první řadě určený k řazení vrcholů grafu podle důležitosti. Jeho výstupem je vektor stacionárního rozdělení, jehož každá jedna složka udává důležitost příslušného vrcholu v grafu a tedy i pořadí důležitosti. Naší úlohou je však klasifikace jednotlivých zařízení. Jak bylo popsáno v Sekci 3.1.2, toho dosahujeme podobně jako v Sekci 4.1.1 určením mezní hodnoty, se kterou budeme PageRank skóre zařízení porovnávat. Je-li skóre vyšší než mez, pak jej klasifikujeme jako server, je-li nižší, přiřazujeme ho k ostatním.

Stejně jako při užití heuristického algoritmu zde vyvstává problém různorodosti jednotlivých sítí. Zejména velikost sítě zde hraje roli, jelikož složky PageRank vektoru jsou v součtu 1 a průměr složek je proto $1/N$ kde N je počet zařízení v síti. Běžně se stává, že všechny zařízení malé sítě o zhruba padesáti zařízení (například Z1) mají podstatně vyšší PageRank skóre než většina zařízení s nejvyšším PageRank skóre ze sítě o desetitisících zařízení[1] (například Z7). Budeme-li uvažovat běžně volenou hodnotu parametru $\alpha = 0.9$ (podrobně popsaný v Sekci 0.3), kde $1 - \alpha$ udává pravděpodobnost teleportace do náhodného vrcholu, pak PageRank skóre každého vrcholu sítě o padesáti zařízení musí být alespoň $\frac{0.1}{50} = 0.002$, na takovou hodnotu však v síti Z7 dosáhne méně než 0.1% zařízení. Stejně tak nám PageRank neposkytuje žádné záruky o rozptylu výstupního vektoru.

Z těchto důvodu vektor stacionární rozdělení standardizujeme. Nad přeškálovaným vektorem pak hledáme vhodnou univerzální mezní hodnotu stejně jako pro heuristický algoritmus v Sekci 4.1.1.

4.1.2.1 PageRank - základní varianta

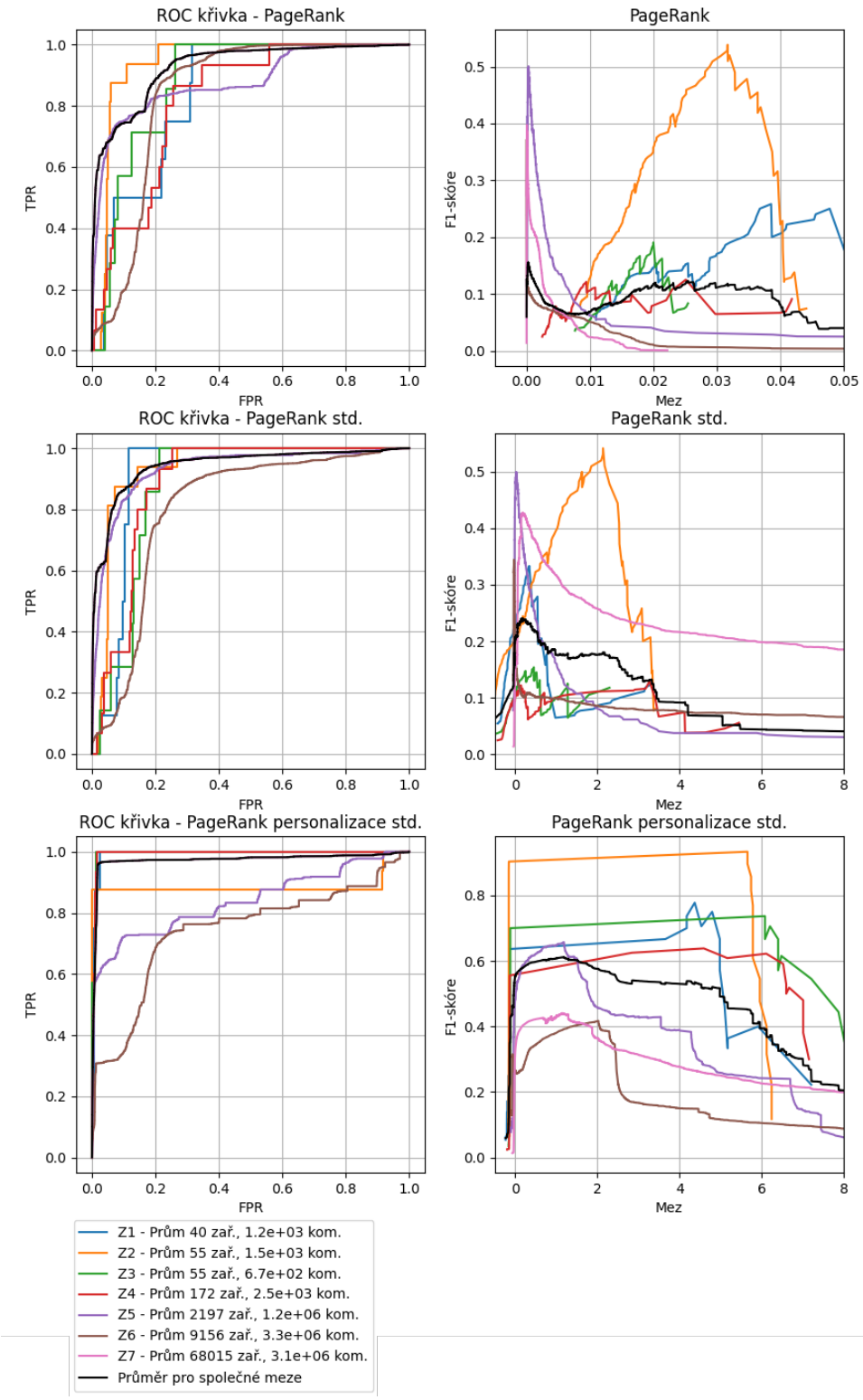
Jako první vyhodnocujeme základní variantu PageRanku pro klasifikaci, jak jsme ji dosud popisovali. Parametr α volíme 0.85, coby běžně používanou hodnotu 0.3. Grafy na Obrázku 4.4 na prvním a druhém řádku srovnávají klasifikaci s využitím standardizace (2. řádek) oproti aplikaci přímo na vektor stacionárního rozdělení (1. řádek). Z grafu si můžeme všimnout, že ačkoliv standardizace celkově zlepšila klasifikační možné dosažené F1-skóre a ROC křivka vykazuje lepší charakter, nedosahuje ani jedna z metod dobrých výsledků ve srovnání s Heuristickým algoritmem, Sekce 4.1.1, Obrázek 4.1.

Dále se zabýváme otázkou: Proč PageRank dosahuje tak slabých výsledků ve srovnání s výsledky heuristického algoritmu ze Sekce 4.1.1.1? Pro vysvětlení využíváme záznam úterní komunikace sítě Z5, jež obsahuje extrémní, ale velmi příkladný projev problému. Ilustrace vybrané sítě je zobrazena na Obrázku 4.5. Zařízení **A** je HTTP server a během dne přijal přes sta-tisíce komunikací (řádově víc, než kterékoliv jiné zařízení) z nichž se všechny odehráli na portech 80 a 8080. Díky nejvyššímu počtu přijatých komunikací od mnoha různých zařízení, získal po právu PageRank skóre 0.34. Zařízení **A** během dne samo navázalo jedinou komunikaci a to se zařízením **B** a důsledkem toho mu předává veškerou důležitost. Zařízení **B** za celý den přijalo pouhé čtyři komunikace, přesto však dosáhlo na PageRank skóre 0.29.

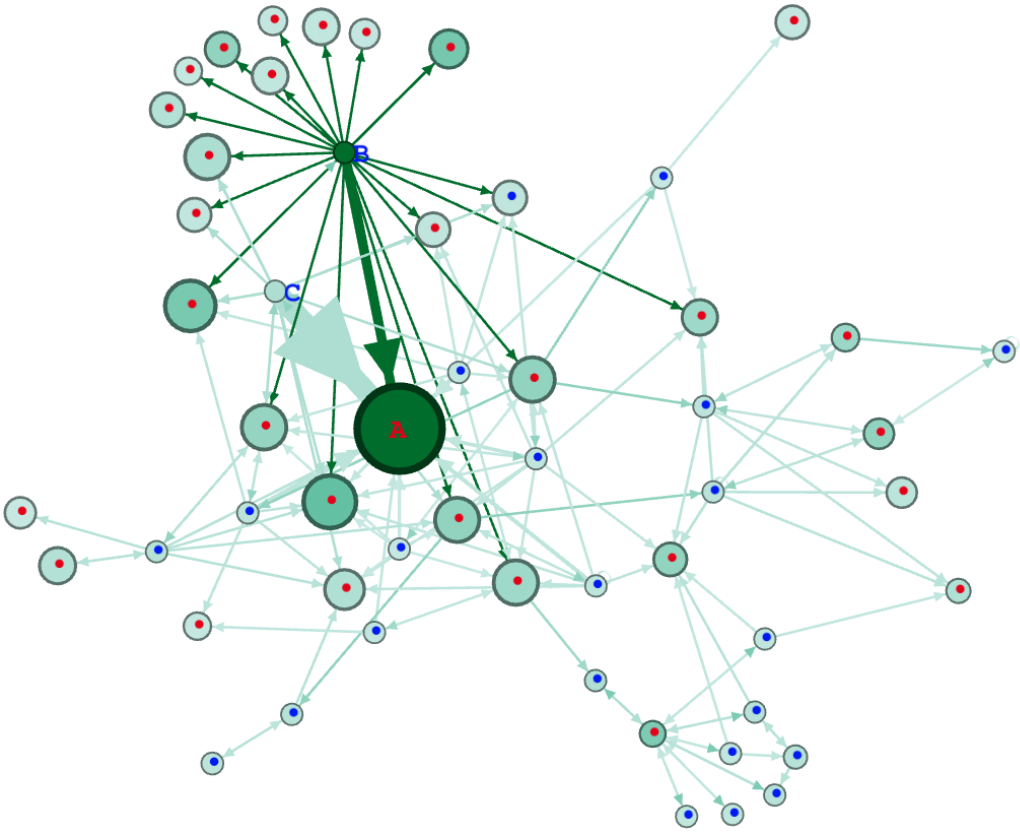
Vyvstává otázka, zda-li je toto vyhodnocení žádoucí? Může být pro jisté typy služeb. Některé služby si například mohou jen několikrát denně aktualizovat databázi z nějakého centrálního serveru. V našem případě se však pravděpodobně nejedná o tento případ, ale spíš o šum, jelikož zařízení **B** vykazuje chování koncového zařízení.

Manuální inspekce záznamů komunikace sítě v jiných dnech a ukázalo se, že podobné volání zařízení **B** už znovu nenastává. Nicméně, zařízení **A** v některé dny navazuje spojení hned s několika různými zařízeními a v jiných dnech nenavazuje žádné komunikace. Důsledkem toho jsou predikce značně ovlivněny jednotkami volání, které považujeme za šum.

[1] Platí pouze ve variantě bez personalizace



Obrázek 4.4 Grafy zobrazující ROC křivky a závislost F1 skóre na mezní hodnotě pro jednotlivé klasifikační metody založené na PageRanku



Obrázek 4.5 Graf aplikace PageRanku na úterní data sítě Z5. Z původního grafu je zobrazeno pouze 1% zařízení, které získali nejvyšší PageRank skóre. Velikost vrcholu je daná vstupním počtem hran. Zabarvení vrcholu představuje hodnotu PageRank skóre (tmavší = vyšší skóre). Barva značky představuje externě dodaný label zařízení, červené jsou servery, modré ostatní zařízení. Velikosti i barvy vrcholů zobrazují hodnoty nelineárně.

Zákazník	PageRank std., mez = 0.0255		
	Precision	Recall	F1
Z1 (malý)	0.140	1.000	0.246
Z2 (malý)	0.113	1.000	0.204
Z3 (malý)	0.060	1.000	0.114
Z4 (malý)	0.057	0.867	0.106
Z5 (velký)	0.439	0.579	0.499
Z6 (velký)	0.129	0.182	0.151
Z7 (velký)	0.092	0.677	0.161
Průměr malých	0.093	0.967	0.167
Průměr velkých	0.220	0.479	0.271
Průměr všichni	0.147	0.758	0.212

Tabulka 4.3 Výsledky klasifikace pomocí PageRanku

Zařízení **A** není touto charakteristikou výjimečné a podobné odchylky, ale s menšími dopady, nacházíme napříč všemi sítěmi. Právě podobné pochybné předávání důležitosti je důvodem pro časté přiřazování vysokého PageRank skóre málo vytěžovaným zařízením. Servery stále dostávají vyšší skóre než většina ostatních zařízení, ovšem nejvyšší příčky bývají protkané zařízeními podobným **B**.

Dalšího problému se zařízením **A** si všímáme, analyzujeme-li jej napříč dny. Zařízení v některé dny navázalo jednotky komunikací, zatímco v jiné nenavázalo žádnou. Jestliže **A** během dne naváže komunikaci, bude jeho důležitost $\times\ \alpha$ rozdělena mezi cílové zařízení, ovšem pokud nenaváže žádnou komunikaci, bude jeho důležitost rozprostřena po celé síti (Sekce 0.3). Kvůli tomu se mohou statistické vlastnosti výstupního vektoru stacionárního rozdělení výrazně měnit, což způsobí, že nebudeme schopni zvolit mezní hodnotu, která by dobře klasifikovala zařízení v různých dnech, natož pak v různých sítích. Jev kdy často volané zařízení nevykazuje žádnou odchozí komunikaci, je pravděpodobně způsoben absencí softwaru pro jejich záznam (ačkoliv je možné, že zařízení reálně nenavazuje žádnou komunikaci). Nicméně je tak častý napříč sítěmi, že je třeba ho brát v potaz.

Algoritmus PageRank, jsme volili pro jeho schopnost zachycení vlastnosti přenosu důležitosti mezi zařízeními. Té jsme chtěli využít pro nalezení zařízení, které nepřímají mnoho komunikací, ale přesto jsou kritické pro fungování často volaných zařízení. Naneštěstí v praxi se ukazuje, že server může velkou část výstupní komunikace věnovat nedůležitým zařízením. To i neúplnost grafu, zapříčiněná sběrem komunikace jen na podmnožině zařízení (Sekce 2), způsobují výše popsané, nepředvídatelné výkyvy v rozdělení PageRank skóre.

Tabulka 4.3 zobrazuje výsledky dosažené PageRank algoritmem. Mezní hodnotu 0.025 jsme zvolili coby hodnotu maximalizující průměrné F1 skóre všech sítí. Dosažené výsledky nejsou dobré a PageRank algoritmus je zde jednoznačně překonán Heuristickou klasifikací 4.1.1.1.

4.1.2.2 PageRank - personalizace

Druhou variantou navrženou v Sekci 3.1.2 je využití PageRanku s personalizací. PageRank, aby předešel přenesení veškeré důležitosti ve stacionárním rozdělení do uzavřených množin grafu zavádí již zmiňovanou teleportaci, způsobující, že náhodný chodec během každého kroku přeskočí s pravděpodobností $1 - \alpha$ do kteréhokoliv uzlu s uniformní pravděpodobností. Při využití personalizace však upravujeme pravděpodobnosti teleportace do jednotlivých uzlů neuniformně. Námi navržená metoda rovně rozděluje pravděpodobnost teleportace mezi všechny zařízení, které byli předem označeny jako servery jiným algoritmem pro klasifikaci, zatímco ostatní mají nulovou pravděpodobnost. Pro určení rozdělení pravděpodobnosti teleportace využíváme dosud nejúspěšnější Heuristický algoritmus využívající kvantil, který je popsaný v Sekci 3.1.1 a jeho

Zákazník	**PageRank pers., mez = 1.184**			HA - Kvantil, mez = 0.977		
	Precision	Recall	F1	Precision	Recall	F1
Z1 (malý)	0.538	0.875	0.667	0.700	0.875	0.778
Z2 (malý)	1.000	0.875	0.933	1.000	0.875	0.933
Z3 (malý)	0.583	1.000	0.737	0.583	1.000	0.737
Z4 (malý)	0.455	1.000	0.625	0.484	1.000	0.652
Z5 (velký)	0.785	0.563	0.656	0.972	0.555	0.707
Z6 (velký)	0.539	0.308	0.392	0.933	0.308	0.463
Z7 (velký)	0.415	0.463	0.438	0.285	0.954	0.439
Průměr malých	0.644	0.938	0.740	0.692	0.938	0.775
Průměr velkých	0.580	0.445	0.495	0.730	0.606	0.536
Průměr všichni	0.617	0.726	0.635	0.708	0.795	0.673

Tabulka 4.4 Výsledky klasifikace PageRanku s personalizací. Pro srovnání tabulka nabízí vhled i do výsledků Heuristického algoritmu využívající kvantil, který využívá.

vlastnosti jsou zhodnoceny v Sekci 4.1.1.1. Hodnotu parametru $\alpha = 0.85$ ponecháváme stejný jako v základní variantě.

Hlavním praktickým rozdílem ve fungování algoritmu oproti základní verzi je původ důležitosti zařízení. V základní verzi všechna zařízení přispívají rovnoměrně důležitostí, která je přes hrany přerozdělena mezi zbytek sítě. Často volaná zařízení tak získávají velkou důležitost od koncových zařízení a předávají ji strojům a velký její díl předávají strojům, která samy volají.

S využitím personalizace se tento proces rozděluje na dva samostatné kroky. Nejprve Heuristický algoritmus vybere servery. Ty jsou často volané z koncových zařízení. PageRank s personalizací odpovídající predikcím HA zachovává díky teleportaci (a běžné topologie sítě do shluků okolo serveru) důležitost předem vybraných zařízení a zařazuje mezi ně další zařízení, na které se sami často odkazují.

Díky tomu, že v personalizaci přiřazujeme všem HA důležitým zařízením stejnou pravděpodobnost nemůže dojít k výraznému selhání z Obrázku 4.5, blíže popsané v Sekci 4.1.2.1, kde dominantní zařízení **A** nabývající PageRank skóre 0.34 sdílí veškerou svoji důležitost se zařízením **B**, na základě jediné komunikace. Ačkoliv i zde je pravděpodobné, že zařízení **B** bude PageRankem označeno jako důležité, neboť jeho důležitost bude alespoň $\alpha \times$ Počet HA důl. zař.$^{-1}$, neovlivní tento jev statistické vlastnosti vektoru stacionárního rozdělení natolik, aby výrazně změnil optimální mez ve srovnání s ostatními dny, v nichž zařízení **A** nenaváže žádnou komunikaci.

Tabulka 4.4 zobrazuje výsledky dosažené PageRank algoritmem s personalizací. Mezní hodnotu 1.184 jsme zvolili, neboť se jedná o hodnotu maximalizující průměrné F1 skóre všech sítí. Ty zde srovnáváme s výsledky Heuristického algoritmu, využitého pro nastavení hodnot personalizace. Podle F1 skóre ve všech případech sítí původní Heuristický algoritmus klasifikuje zařízení lépe než PageRank s personalizací. Ve většině sítí zaznamenáváme stejný Recall, ale stejný nebo menší Precision. To způsobuje právě vlastnost PageRanku, která označuje zařízení jako důležité, jestliže se na ně často odkazují HA zařízení. Tato vlastnost je buď celkově nevhodná, nebo nebyla brána v potaz při externím ohodnocení zařízení.

4.1.3 XGBoost

Dále se zabýváme naší poslední představenou metodou pro klasifikaci zařízení. Tou je XGBoost (eXtreme Gradient Boosting), state-of-the-art model strojového učení, určený pro aplikace na tabulková data. Jeho teoretické podklady a vlastnosti jsme představili v Sekci 0.2. Tento model nasazujeme na tabulkovou reprezentaci dat, jak byla popsána v Sekci 2.2. V této sekci popisujeme předzpracování dat, ladění hyperparametrů modelu a jeho učení. Dále vyhodnocujeme kvalitu modelu a jeho vlastnosti a srovnáváme je s dříve představenými přístupy pro klasifikaci.

Zákaznická síť	Z1	Z2	Z3	Z4	Z5	Z6	Z7
Podíl váhy	0.05	0.05	0.05	0.1	0.15	0.3	0.3

Tabulka 4.5 Váhy přiřazené jednotlivým sítím pro ladění XGBoost

Nejprve se zaměřujeme na problém různorodosti jednotlivých sítí. V předchozích případech jsme přistupovali ke každé síti jako k samostatnému vzorku a tak každá síť přispěla k výslednému modelu stejným dílem, bez ohledu na její velikost. Při trénování modelu XGBoost obsahuje trénovací množina konkrétní zařízení ze všech použitých sítí. To znamená, že dataset obsahuje řádově více zařízení pocházejících z velkých sítí, než těch z malých. Důsledkem toho se model naučí vlastnosti charakteristické pro největší sítě, zatímco vlastnosti malých, reprezentované jen malým vzorkem dat se ztratí v šumu.

Abychom zajistili schopnost modelu dobře klasifikovat zařízení jak malých tak i velkých sítí, udáváme všem prvkům datasetu váhu. Ta se přímo odvíjí od počtu prvků sítě, ze které pochází. Naším požadavkem je, aby každá síť měla svůj vliv na výsledný model. Kdybychom rozdělili váhy rovnoměrně, pak by každému zařízení byla přiřazena váha $(7n)^{-1}$, kde n je počet prvků. V takovém případě by však prvky z nejmenších sítí získali extrémně vysokou váhu a hrozilo by, že se na ně model přeučí. Z toho důvodu manuálně udáváme váhu pro každou síť tak, aby se její vzorky přiměřeně promítly do natrénovaného modelu. Přesné váhu sítě jsou udány v Tabulce 4.5. Váha individuálního prvku je pak rovna Váha sítě $\times\, n^{-1}$.

Použitý dataset není nevyvážený jenom co se zdrojových sítí týče. Potýkáme se i s nerovnoměrným zastoupení tříd vysvětlované proměnné, neboť zařízení s označením server, je řádově méně než ostatních. Zavádíme kombinaci dvou opatření, prvním je náhodný oversampling. Ten náhodně vybírá prvky z minoritní třídy (servery) a přidává jejich kopie do datasetu. Všechny prvky poté nezávisle na ostatní podrobujeme augmentaci dat. Augmentaci provádíme na všech atributech prvku (Tabulka 2.3) s výjimkou dne v týdny tak. Každý prvek násobíme náhodným číslem z rozdělení $N(1, 0.4)$ (prvky menší než nula zarovnáváme na nulu). Přenásobením atributu číslem blízkým jedné získáváme nové prvky, které mají poměrově podobné vlastnosti, jako původní prvek, ale zároveň nezpůsobí tendenci modelu se vůči nim přeučovat. Atribut Celkový počet komunikací poté znovu přepočítáváme, aby vše odpovídalo svému významu.

Vedle rozprostření opakujících se vzorků, které vznikli oversamplingem má tato augmentace dat další výhodnou vlastnost. Tou je i rozprostření atributů, které jsou shodné pro celou síť, např. počet zařízení na síti. To brání modelu učit se charakteristiky individuálních sítí a namísto toho hledá generalizované vlastnosti.

Ukázalo se, že aplikace náhodného oversamplingu, dokud data nejsou vyrovnaná navyšuje počet prvků datasetu tak, že opakované trénování modelu v rámci hledání hyperparametrů začne být příliš výpočetně náročné. Z toho důvodu užíváme náhodný oversampling jen tak, aby servery tvořily třetinu datasetu. Pro zahlazení zbylé nevyváženosti dat využíváme váhování stejně jako v předchozím disbalance velikosti sítí, zde však postupujeme tak, aby měly obě třídy stejnou váhu. Kombinovaný vzorec pro výpočet váhy prvku je:

$$w = \text{Váha sítě} \times n^{-1}c, \quad c = \begin{cases} \frac{\#\text{ serverů v síti}}{n} & \text{pro server} \\ 1 - \frac{\#\text{ serverů v síti}}{n} & \text{pro ostatní} \end{cases}$$

Jedním z hyperparametrů, který ladíme je standardizace (použít / nepoužít). V případě použití standardizace aplikujeme na každý jednodenní záznam sítě samostatně. Myšlenkou za využití standardizace je snaha přeškálovat data různých sítí do stejné roviny, jak to bylo popsáno v Sekci 3.1.1 a zvýšit tak schopnost modelu generalizovat znalosti na neznámé sítě. Abychom předešli ovlivnění statistických vlastností vzorků sítě, zaznamenáváme průměr a standardní odchylku potřebné pro výpočet standardizace datasetu před oversamplingem, ale aplikujeme ji jako poslední krok předzpracování dat. Důsledkem toho nejsou atributy výsledné datasetu normálně, v případě ladění modelu XGBoost to však nevadí.

Hyperparametr	Hodnoty			
Maximální hloubka stromu	**4**	6	8	10
Shrinkage η	0,2	0,3	**0,6**	1.0
Počet stromů	4	**7**	10	
Standardizace Histogramu	**Ano**	Ne		

Tabulka 4.6 Prostor hyperparametrů využitý při grid searchy pro XGBoost. Zvýrazněné hodnoty jsme vybrali pro finální model

Pro správné fungování modelu je třeba dobře vyladit jeho hyperparametry. To není výjimka ani pro XGBoost. Jak již bylo zmíněno jedním z hyperparametrů, který ladíme použití standardizace. Dalšími hyperparametry, kterými se zabýváme jsou striktně hyperparametry modelu, na něž je XGBoost, dle jeho autorů, citlivý [8]. Těmi jsou

- Maximální hloubka stromu. Hlubší stromy mohou modelovat komplexnější vztahy, ale jsou zároveň náchylnější vůči přeučení.
- Počet stromů. Podobně jako předchozí parametr obecně vyšší počet stromů může vést k lepším výsledkům, ale stejně tak hrozí vyšší riziko přeučení.
- Shrinkage (scvrknutí), popř. learning rate η je parametr udávající váhu jednotlivých stromů a omezuje tak přeučení.

K ladění hyperparametrů modelu využíváme Cross-validaci popsanou v Sekci 0.4. Abychom ověřili schopnost modelu generalizovat neviděné zákazníky, je dataset dělen na 7 dílů podle zákazníků a v každé iteraci vynechává data právě jednoho zákazníka, která jsou následně využita k validaci. Pro získání povědomí o důležitosti a vlivu jednotlivých hyperparametrů jsme provedli Grid search (Sekce 5) 4 zmíněných parametrů. Konkrétní hodnoty, které jsme testovali jsou popsané v Tabulce 4.6. Pro zbylé hodnoty hyperparametrů volíme výchozí hodnoty doporučené autory [8].

Ze všech laděných hyperparametrů měla největší vliv na model aplikace standardizace. Jak už naznačují poznatky z experimentu popsaného v Sekce 4.1.1.3, tak standardizace neškáluje data optimálně. To se projevilo i v případě standardizace příznaků XGBoostu. Ostatní parametry měly menší vliv a vhodnost konkrétních hodnot závisela na volbě ostatních parametrů a nejde tak jakýkoliv další hyperparametr vyzdvihnout.

Z nejlépe hodnocených setů hyperparametrů jsme na základě F1 skóre, AUC a vlastní preference více regulovaných modelů zvolili pro finální model kombinaci hyperparametrů, které je v Tabulce 4.6 zvýrazněna tučně.

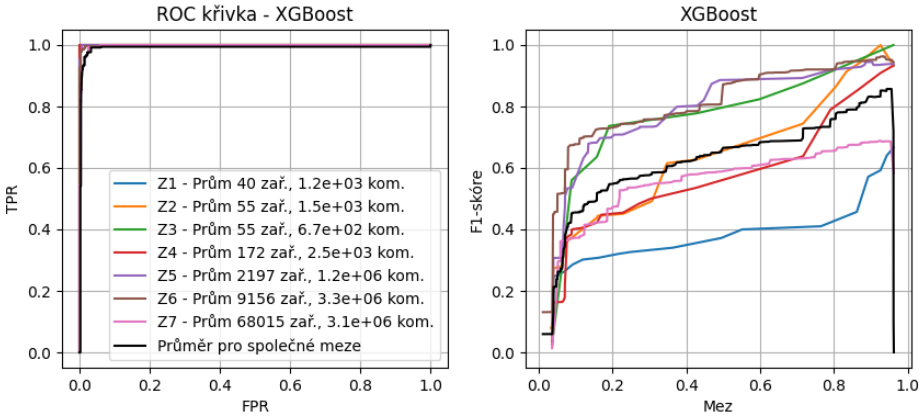
Podobně jako v předchozích Sekcích 4.1.1 a 4.1.2 této kapitoly dále zkoumáme vliv volby meze na metriky. Ty jsou vyobrazeny na Obrázku 4.6. Model výtečně predikuje prvky napříč sítěmi. Klasifikační mez volíme rovnou hodnotě 0.943, která maximalizuje F1 skóre.

Vyhodnocení metrik naučeného modelu jsou vyobrazeno v Tabulce 4.7. Zde vidíme, že model dosahuje na validační množině velmi dobrých výsledků a výrazně předčí předchozí představené metody klasifikace. Navíc si všímáme nízkého rozdílů mezi trénovací a validační metrikou, což naznačuje, že model je schopný dobře generalizovat naučené znalosti. Kvůli tomu považujeme jeho použití za vhodné pro jakékoliv sítě.

4.1.4 Analýza časové náročnosti klasifikačních algoritmů

V této sekci analyzujeme časové nároky klasifikačních modelů, představených v předchozích sekcí této kapitoly.

Časová měření, která jsme provedli jsou zaznamenány v Tabulce 4.8. Zde popisujeme naměřený čas předzpracování vyžadované datové reprezentace a samotné klasifikace algoritmu všech prvků



Obrázek 4.6 Graf zobrazující ROC křivky a závislost F1 skóre na mezní hodnotě pro jednotlivé pro XGBoost. Křivka každého modelu je vyhodnocena modelem, který byl naučen na ostatních datech.

	XGBoost, mez = 0.943					
Zákazník	Trénovací množina			Validační množina		
	Precision	Recall	F1	Precision	Recall	F1
Z1 (malý)	0.500	1.000	0.667	0.471	1.000	0.640
Z2 (malý)	1.000	1.000	1.000	1.000	0.875	0.933
Z3 (malý)	1.000	1.000	1.000	1.000	1.000	1.000
Z4 (malý)	0.933	0.933	0.933	0.933	0.933	0.933
Z5 (velký)	0.916	0.957	0.936	0.925	0.953	0.939
Z6 (velký)	0.964	0.957	0.960	0.965	0.940	0.952
Z7 (velký)	0.565	0.851	0.679	0.596	0.809	0.687
Průměr malých	0.858	0.983	0.900	0.851	0.952	0.877
Průměr velkých	0.815	0.922	0.858	0.829	0.901	0.859
Průměr všichni	0.840	0.957	0.882	0.841	0.930	0.869

Tabulka 4.7 Výsledky klasifikace XGBoost modelu. Pro evaluaci sítě coby trénovací množiny jsme použili model, který byl učen na všech krom jedné (malá síť odlišná od evaluované) sítě. Pro evaluaci validačních metrik sítě jsem použili model trénovaný na všech ostatních sítích.

Algoritmus	Síť	Prům. počet zař.	Celk. čas μ	Celk. čas σ	Před. čas μ	Před. čas σ	Model. čas μ	Model. čas σ
HA - kv.	Z1	42	2.6e-3	6.8e-4	5.1e-6	1.6e-6	2.6e-3	6.8e-4
	Z2	57	2.5e-3	4.6e-4	4.8e-6	7.1e-7	2.5e-3	4.6e-4
	Z3	56	2.7e-3	7.5e-4	6.0e-6	1.5e-6	2.7e-3	7.5e-4
	Z4	174	4.0e-3	2.1e-3	5.8e-6	1.7e-6	4.0e-3	2.1e-3
	Z5	2286	0.18	0.08	7.1e-6	5.9e-6	0.18	0.08
	Z6	9797	0.48	0.12	6.4e-6	4.5e-6	0.48	0.12
	Z7	68483	0.59	0.18	8.2e-6	2.0e-6	0.59	0.18
HA - std.	Z1	42	3.6e-3	7.0e-4	2.3e-6	6.7e-7	3.6e-3	7.0e-4
	Z2	57	3.3e-3	5.5e-4	2.1e-6	4.7e-7	3.3e-3	5.5e-4
	Z3	56	3.2e-3	7.7e-4	4.6e-6	6.2e-6	3.2e-3	7.8e-4
	Z4	174	3.7e-3	1.0e-3	1.8e-6	4.1e-7	3.6e-3	1.0e-3
	Z5	2286	0.17	0.07	1.9e-6	3.8e-7	0.17	0.07
	Z6	9797	0.48	0.11	1.6e-6	2.3e-7	0.48	0.11
	Z7	68483	0.61	0.21	1.9e-6	2.5e-7	0.61	0.21
PageRank	Z1	42	9.9e-3	2.1e-3	4.6e-3	1.4e-3	5.3e-3	1.2e-3
	Z2	57	7.6e-3	7.1e-4	4.0e-3	5.8e-4	3.6e-3	7.9e-4
	Z3	56	8.0e-3	2.1e-3	4.5e-3	1.5e-3	3.5e-3	1.2e-3
	Z4	174	0.02	4.9e-3	9.7e-3	3.9e-3	5.6e-3	1.5e-3
	Z5	2286	2.18	0.95	2.09	0.95	0.09	0.03
	Z6	9797	12.28	3.84	11.81	3.71	0.47	0.17
	Z7	68483	17.03	5.83	15.96	5.51	1.07	0.37
PR pers.	Z1	42	0.01	1.3e-3	3.9e-3	1.4e-3	6.7e-3	1.0e-3
	Z2	57	0.02	0.02	0.01	0.02	6.1e-3	1.1e-3
	Z3	56	0.01	1.5e-3	4.0e-3	4.8e-4	6.3e-3	1.6e-3
	Z4	174	0.02	3.6e-3	9.0e-3	3.6e-3	8.2e-3	1.3e-3
	Z5	2286	2.37	1.06	2.12	0.95	0.25	0.11
	Z6	9797	12.51	3.69	11.45	3.46	1.06	0.24
	Z7	68483	18.05	6.37	15.97	5.66	2.09	0.80
XGBoost	Z1	42	0.02	0.01	0.01	2.3e-3	0.01	0.01
	Z2	57	0.02	3.4e-3	0.01	1.3e-3	5.8e-3	3.6e-3
	Z3	56	0.02	9.5e-3	0.01	2.9e-3	8.7e-3	7.3e-3
	Z4	174	0.03	8.9e-3	0.02	3.5e-3	0.01	7.7e-3
	Z5	2286	2.91	1.25	2.90	1.25	0.01	0.02
	Z6	9797	8.53	1.83	8.52	1.84	9.0e-3	0.02
	Z7	68483	8.35	2.39	8.33	2.38	0.01	4.0e-3

Tabulka 4.8 Časové záznamy klasifikace představenými algoritmy. Veškeré uvedené časové údaje jsou v sekundách. Tabulka zaznamenává časové údaje zvlášť pro předzpracování dat, pro samotnou klasifikaci algoritmu a celkový čas. μ představuje průměr a σ směrodatnou odchylku napříč dny.

na síti. Je zde třeba mít na paměti, že optimalizace implementace jednotlivých algoritmů se liší a naměřené časy nemusí přímo odpovídat jejich složitosti.

Zaměřené experimenty byly měřeny na osobním počítači s procesorem Intel® Core™ i7-7700HQ CPU @ 2.80GHz × 8 a 16 GB operační paměti, za minimálního vedlejší využívání.

Heuristický algoritmus - Kvantil, Sekce 4.1.1.1 byl prvním představeným algoritmem. Jeho složitost je $\mathcal{O}(n \times log(n))$, jelikož pro výpočet kvantilu je třeba všechna zařízení seřadit podle počtu volání. Nad seřazeným polem přímočaře přiřazujeme každému zařízení kvantil a následně je srovnáváme s mezí. Tento přístup by šel optimalizovat tak, že spočítáme hodnotu požadovaného kvantilu a tu bychom dále využili pro klasifikaci původních příznaků zařízení. Naměřené celkové časy tohoto přístupu jsou nejnižšími zaznamenanými mezi všemi algoritmy.

Heuristický algoritmus - Standardizace, Sekce 4.1.1.3 jako předchozí přístup využívá stejně jako předchozí přístup klasifikační mez. Ovšem časová složitost se liší, neboť zde stačí pouze lineárně iterovat pole pro zaznamenání statistik vyžadovaných, které slouží jako parametry standardizace. Zbylý postup je stejný jako v případě předchozího algoritmu. Výsledná složitost je $\mathcal{O}(n)$. I tento přístup lze dále optimalizovat přepočítáním mezní hodnoty pro danou síť, namísto škálování samotných prvků. Navzdory nižší složitosti si varianty standardizace a kvantil časově odpovídají. Věříme, že je to způsobeno režií okolo agregování surových záznamů komunikací, která trvá podstatně déle, než samotná klasifikace.

PageRank, Sekce 4.1.2 využívá k výpočtu mocninnou metodu. Její složitost závisí na počtu zařízení, počtu (agregovaných) spojení a rychlosti konvergence. Pro takové metody je těžké odhadovat rychlost konvergence, nicméně naše grafy, reprezentující záznamy komunikace na síti, jsou velmi řídké. To zásadně zrychluje konvergenci a ačkoliv my jsme neměřili počet iterací, stačí podle autorů PageRanku sítě 10 - 100 iterací k dosažení konvergence [10]. V Tabulce 4.8 si všímáme, že výpočetní čas algoritmu je zhruba vteřina na největší síti. Předzpracování na druhou stranu trvá nejdéle ze všech metod. To je mimo jiné způsobeno i samotnou, implementací která nabízí vhled do vlastností algoritmu, jelikož reálně konstruuje graf. Implementace optimalizovaná pro výkon by měla používat nějakou formu řídké matice, jejíž naplnění by mělo být rychlejší.

PageRank s personalizací, Sekce 4.1.2.2 staví na algoritmech výše vyhodnocených algoritmech PageRank a Heuristická klasifikace kvantil. Prakticky je využívá v sérii a příslušné naměřené hodnoty tomu odpovídají.

XGBoost, Sekce 4.1.3 je co do úspěšnosti klasifikace nejúspěšnějším algoritmem. Pro klasifikaci jednoho prvku je třeba projít všemi naučenými stromy až do listů, na jejichž základě vzniká výsledná klasifikace. Důsledkem toho je složitost algoritmu pro klasifikaci celé sítě $\mathcal{O}(n \times$ #stromů $\times$ max hloubka). Počet stromů i jejich hloubka jsou neměnné (konkrétní parametry jsme volili 7×4) a velmi nízké. Díky tomu a faktu, že nemusí zpracovávat surová data jako HA je samotná klasifikace nejrychlejší ze všech algoritmů a to i díky velmi efektivní implementaci knihovny XGBoost jeho autory [8]. Toto prvenství je třeba brát s rezervou, neboť oba přístupy tuto agregaci vyžadují, jen XGBoost ji provádí v rámci předzpracování. Na druhou stranu XGBoost disponuje podstatně delším předzpracováním dat, při němž jsou počítány histogramy volání, hledány specifické porty a počítány statistické vlastnosti sítě.

Ačkoliv celkové časy klasifikace se napříč algoritmy lišily, nevidíme zde žádný algoritmus, jehož vyžadovaná doba běhu by byla příliš dlouhá. Mluvíme-li o klasifikaci sítě na časovém okně jednoho dne, která neprobíhá často, jsou vyžadované časy téměř zanedbatelné a proto nám neslouží jako kritérium při posuzování modelu.

4.1.5 Shrnutí klasifikačních algoritmů

V této sekci shrnujeme poznatky, kterých jsme nabyli při experimentování s jednotlivými algoritmy ze Sekce 4.1.

Představili jsme si tři základní algoritmy a jejich další podvarianty.

Výsledky heuristického algoritmu s využitím standardizace nebyly příliš dobré v porovnání s ostatními metodami. Navíc jsme nebyli schopni najít univerzální mez, ale namísto toho jsme

zvolili dvě různé meze pro různé velikosti sítě. Jelikož máme data pouze ze 7 sítí není možné s jistotou říct, jak se optimální mez bude pohybovat pro nové, neviděné sítě. Proto tento přístup nedoporučujeme.

Heuristický algoritmus využívající kvantil na druhou stranu dosahuje poměrně dobrých výsledků napříč všemi sítěmi. Algoritmus nutně vybírá 2,3% nejvolanějších zařízení. Neexistuje však záruka, že tuto proporcionalitu budou vykazovat i jiné, neviděné sítě.

PageRank algoritmus nedosahoval dobrých výsledků ve srovnání s ostatními přístupy. Ať už důsledkem vlastní podstaty, absencí softwaru pro sběr dat nebo vzorkování, některé často volané zařízení vykazovaly minimální a podle všeho zašuměnou výstupní komunikaci, která však zásadně ovlivňovala ohodnocení celé sítě. Zde se ukázala vlastnost předávání důležitosti mez zařízeními jako zcela nevhodná.

PageRank s personalizací dosahoval podobných případně horších výsledků, nežli algoritmus udávající jeho personalizační vektor. Personalizační algoritmus tak zde měl hlavní váhu při určení predikcí. Přenos důležitosti, který měl PageRank nabídnout ani zde nevedl ke zlepšení klasifikace.

Konečně XGBoost byl posledním představeným algoritmem. Dosahuje velmi dobrých výsledků, které dominovaly ostatním přístupům. Jelikož jsme při cross-validaci zaznamenali minimální generalizační chybu, věříme, že nasazení XGBoostu bude vhodné i pro neviděné sítě. Protože i časová náročnost algoritmu byla nízká, považujeme XGBoost za model nejvhodnější pro aplikaci v praxi.

Další práce by se mohla zabývat detailnějším laděním hyperparametrů XGBoostu. Zajímavé by také mohlo být v případě Heuristické klasifikace využití škálování do log-standardního zobrazení, který, dle zběžného zkoumání, předčí běžnou standardizaci. Tuto vlastnost jsme našli v pozdních fázích experimentování a proto jsme se jí blíže nezabývali.

V kapitole zůstal nezmíněný námi provedený experiment, využívající pro klasifikaci neuronovou síť. Tu jsme aplikovali na tabulková data představená v Sekci 2.2, stejně jako XGBoost. Nicméně jsme nebyli schopni zajistit jakoukoliv schopnost generalizace modelu a proto se jím detailně nezabýváme.

4.2 Detekce zařízením poskytovaných služeb

V této sekci vyhodnocujeme přístupy k detekci poskytovaných služeb, které jsme představili v Sekci 3.2. Jelikož nemáme dostupná data, která by popisovala služby poskytované jednotlivými zařízeními, validujeme navržené metody manuálně. Vybíráme zajímavá zařízení a zkoumáme rozdílné vlastnosti představených škálovacích metod. Tyto vlastnosti zkoumáme na jediném dni – Pondělí.

V Sekci 3.2 jsme představili dvě škálovací metody. Jsou jimi Standardizace a L1 normalizace. Ty aplikujeme na součty příchozích komunikací pro jednotlivé porty zařízení. Jedním z problémů obou škálovacích metod je ztráta informace o absolutním počtu volání portu. Důsledkem toho se stává, že velmi malý počet komunikací služby, která se celkově na síti vyskytuje zřídka získá vysokou relevanci na zařízení. Z toho důvodu jsme zavedli podmínku, která říká, že pro detekci služby na zařízení musí být služba volána alespoň v 1% případů a alespoň 30 komunikacích. Tato kombinace zabraňuje chybné detekci služby.

Zprvu se zabýváme samotnou podmínkou a zkoumáme, jestli je toto filtrování vhodné. Tabulka 4.9 ukazuje statistické vlastnosti počtů služeb připadající zařízením. Tyto vlastnosti se jeví jako očekávané a dokonce by mohli být použity i coby finální detekce služeb.

Provedli jsme manuální inspekci filtrování. Servery malých sítí jsme zkoumali individuálně. Filtrace zde dosáhla optimálních výsledků dle posouzení autora. Ve velkých sítích jsme zkoumali servery s největšími i nejnižšími počty přiřazených služeb, ale i některé náhodně vybrané. Všímáme si zde nedostatku samostatného využití metody. V případě zařízení, které klasifikace chybně označila za server, nastane nutně přiřazení některé služby i takovému zařízení. To ukazuje Tabulka 4.10. Zde vidíme zařízení ze sítě Z7, které bylo pravděpodobně chybně klasifikováno. Port

Síť Název	Počet zařízení	Počet serverů	Statistické vlastnosti počtů služeb na server Průměr	S. o.	Kv 0.25	Kv 0.5	Kv 0.75	Kv 0.9	Min	Max
Z1 (malý)	33	1	1.00	-	1	1	1	1	1	1
Z2 (malý)	52	2	1.00	0.00	1	1	1	1	1	1
Z3 (malý)	76	1	1.00	-	1	1	1	1	1	1
Z4 (malý)	260	2	1.00	0.00	1	1	1	1	1	1
Z5 (velký)	3338	95	2.07	1.19	1	2	3	4	1	5
Z6 (velký)	13917	689	1.42	0.92	1	1	2	2	1	12
Z7 (velký)	50665	480	1.71	1.21	1	1	2	3	1	10

Tabulka 4.9 Statistické vlastnosti počtů služeb na server

Port	Počet volání	Standardizace	L1 normalizace	Odfiltrovaný
137	51	1.14	0.0005	Ne
138	17	2.62	0.001	Ano
21027	1	158.71	0.333	Ano
49151	4	0.00	0.00001	Ano

Tabulka 4.10 Příklad zařízení ze sítě Z7, které bylo chybně propuštěno filtrací. Tabulka obsahuje veškeré jeho přijaté komunikace.

137, NetBIOS často slouží k P2P komunikaci a podle toho usuzujeme, že zařízení neposkytuje jakoukoliv službu.

Přeškálování standardizací docílíme toho, že průměr příchozích volání každého jednotlivého čísla portu je roven nule a směrodatná odchylka rovna jedné. Na daném rozdělení pak můžeme detekovat outliery. Ty odpovídají zařízením, která na daném portu komunikovali výjimečně často a tudíž o nich předpokládáme, že hostují příslušnou službu.

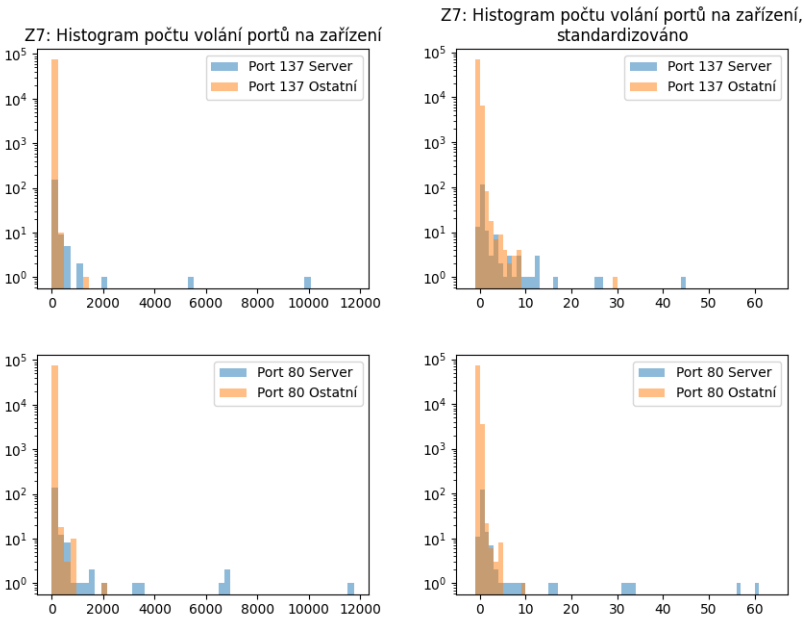
Představené filtrování bralo v potaz relevanci služby vůči zařízení. Standardizace se však zabývá voláním dané služby relativně vůči službě na zbytku sítě. Proto tyto přístupy kombinujeme a k filtraci využíváme i standardizaci. Její vliv na histogram zařízení k danému portu je vidět na Obrázku 4.7. Zde vidíme, že standardizace v případě portu 80 vyzdvihuje část serverů, zatímco ostatní jsou srovnatelné s ostatními zařízeními, což je správně, neboť očekáváme, že ne všechny servery budou poskytovat službu na portu 80.

Obdobně jako v případě samotné filtrace jsme manuálně analyzovali zařízení jednotlivých sítí. Obecně mají služby poskytované na malých sítí výrazně nižší přeškálované příznaky, než veliké sítě. Na základě této analýzy jsme odhalili, že nejnižší avšak vysoce relevantní standardizovaný příznak mají 2 proxy servery komunikující pouze na portu 1328 na síti Z2. Ty jsou zobrazeny v Tabulce 4.11. Na jejich základě, volíme konzervativní filtrovací mez rovnou 3,5. Ta by dle našeho názoru měla nechat prostor i pro větší množinu zátěžově balancovaných služeb nasazených na více zařízeních. Na Obrázku 4.7 si však můžeme povšimnou mnoha ostatních zařízení komunikujících na portu 137, jež získali vyšší standardizovaný atribut, než je zvolená mez. Těmto zařízením v případě chybné klasifikace jako serveru bude služba přiřazena.

Druhou zmíněnou možností škálování je L1 normalizaci. Na rozdíl od standardizace zde není

Id	Port	Počet volání	Standardizace	L1 normalizace	Odfiltrovaný
80de53..	3128	282	4.62	0.47	Ne
a12c09..	3128	320	5.27	0.53	Ne

Tabulka 4.11 Výpis serverů sítě Z2



Obrázek 4.7 Histogram volání portů a vliv standardizace. Třeba vzít v potaz, že některé servery se nevešli do mezí zobrazeného histogramu.

možné univerzálně detekovat outlier, neboť vysoká hodnota se zde přímo odvíjí od počtu zařízení, jež se liší podle počtu zařízení poskytující službu. Nicméně i tak může být výsledný příznak škálovací metody hodnotný při určení relevance zařízení vůči službě, nebo při monitoringu sítě, díky snadné vysvětlitelnosti přeškálovaného atributu.

Pro identifikaci služeb, které zařízení poskytuje, doporučujeme již výše zmíněný postup. Ten ze seznamu všech volaných čísel portů odstraní porty, jejichž počet volání byl:

1. menší než 1% počtu volání nejvolanějšího portu zařízení.
2. menší nebo roven 30.
3. standardizovaný počet volání portu nižší než 3,5.

a zbylé označí za služby relevantní danému zařízení.

Tento přístup se na základě statistických vlastností a manuální validace jeví jako robustní a spolehlivý.

Kapitola 5
Závěr

Hlavním úkolem naší práce byla identifikace kritických uzlů v lokální počítačové síti. K tomuto problému přistupujeme ve třech krocích:

1. klasifikujeme zařízení, zde je či není server za pomocí supervizovaného učení
2. identifikujeme služby příslušící serverům s užitím podmínek
3. posuzujeme dopady výpadku zařízení pomocí algoritmu, který sleduje jak výpadky služeb poskytovaných daným serverem ovlivní ostatní zařízení

Tyto jednotlivé kroky jsou popsány v Kapitole 3. V Kapitole 4 provádíme experimentální vyhodnocení přístupů a dáváme doporučení vůči jednotlivým přístupům.

Pro identifikaci serverů doporučujeme využití XGBoostu (Sekce 3.1.3), který dosáhl nejvyšších klasifikačních metrik ze všech metod. Také vykazuje nízkou generalizační chybu, což naznačuje, že jeho využití bude vhodné i pro nové sítě a má nízkou časovou náročnost.

Přístup který serverům přisuzuje porty, jsme vyhodnocovali manuálně v Sekci 4.2, neboť nemáme data, proti kterým bychom jej mohli objektivně ohodnotit. Na základě manuálního posouzení funguje tento přístup ve většině zkoumaných případů.

S využitím předchozích kroků Algoritmus navržený v Sekci 3.3, určuje které služby výpadek dané množiny ovlivní. Definujeme-li pro něj množinu kritických služeb, bez nichž se ostatní zařízení neobejdou, můžeme určit, i která další zařízení výpadek vyřadí a nadále modelovat další jejich dopady.

Ačkoliv se nabízí několik služeb, které jsou kritické pro všechny sítě (např. DNS, LDAP, NFS), necháváme jejich definici na správci konkrétní sítě. Díky tomu může algoritmus dobře fungovat na kterékoliv síti.

Za kritické uzly pak považujeme ty, jejichž výpadek způsobí výpadek celé kritické služby.

Budoucí práce by se mohla zaměřit na detekci load balancingu. V současné chvíli považujeme služby za shodné, komunikují-li na stejném portu, nicméně například na portu 80 můžeme mít vícero různých poskytovaných služeb napříč sítí. S využitím load balancingu bychom mohli lépe jednotlivé služby oddělit a na jejich základě i lépe modelovat výpadek.

Bibliografie

1. REZAEI, Shahbaz; LIU, Xin. Deep Learning for Encrypted Traffic Classification: An Overview. *IEEE Communications Magazine*. 2019, roč. 57, č. 5, s. 76–81. Dostupné z DOI: 10.1109/MCOM.2019.1800819.
2. JORGE ORTIZ Catherine Crawford, Franck Le. DeviceMien: network device behavior modeling for identifying unknown IoT devices. In: 2018. Dostupné z DOI: 10.1145/3302505.3310073.
3. LUXBURG, Ulrike von; SCHOELKOPF, Bernhard. *Statistical Learning Theory: Models, Concepts, and Results*. arXiv, 2008. Dostupné z DOI: 10.48550/ARXIV.0810.4752.
4. RUDOLF, Jan. *Detecting abnormalities in X-Ray images using Neural Networks*. 2020. Dostupné také z: https://hdl.handle.net/10467/88175.
5. MYLES, Anthony J.; FEUDALE, Robert N.; LIU, Yang; WOODY, Nathaniel A.; BROWN, Steven D. An introduction to decision tree modeling. *Journal of Chemometrics*. 2004, roč. 18, č. 6, s. 275–285. Dostupné z DOI: https://doi.org/10.1002/cem.873.
6. HYAFIL, Laurent; RIVEST, Ronald L. Constructing optimal binary decision trees is NP-complete. *Information Processing Letters*. 1976, roč. 5, č. 1, s. 15–17. ISSN 0020-0190. Dostupné z DOI: https://doi.org/10.1016/0020-0190(76)90095-8.
7. KOTSIANTIS, Sotiris. Decision trees: A recent overview. *Artificial Intelligence Review*. 2013, s. 1–23. ISBN 0269-2821. Dostupné z DOI: 10.1007/s10462-011-9272-4.
8. CHEN, Tianqi; GUESTRIN, Carlos. XGBoost: A Scalable Tree Boosting System. *CoRR*. 2016, roč. abs/1603.02754. Dostupné z arXiv: 1603.02754.
9. PAGE, Lawrence; BRIN, Sergey; MOTWANI, Rajeev; WINOGRAD, Terry. *The PageRank Citation Ranking: Bringing Order to the Web*. Stanford InfoLab, 1999-11. Technical Report, 1999-66. Stanford InfoLab. Dostupné také z: http://ilpubs.stanford.edu:8090/422/. Previous number = SIDL-WP-1999-0120.
10. LANGVILLE, Amy N.; MEYER, Carl D. A Survey of Eigenvector Methods for Web Information Retrieval. *SIAM Review*. 2005, roč. 47, č. 1, s. 135–161. Dostupné z DOI: 10.1137/S0036144503424786.
11. BONALD, Thomas; LARA, Nathan de; LUTZ, Quentin; CHARPENTIER, Bertrand. Scikit-network: Graph Analysis in Python. *Journal of Machine Learning Research*. 2020, roč. 21, č. 185, s. 1–6. Dostupné také z: http://jmlr.org/papers/v21/20-412.html.
12. YU, Tong; ZHU, Hong. Hyper-Parameter Optimization: A Review of Algorithms and Applications. *CoRR*. 2020, roč. abs/2003.05689. Dostupné z arXiv: 2003.05689.

13. LI, Lisha; JAMIESON, Kevin; DESALVO, Giulia; ROSTAMIZADEH, Afshin; TALWALKAR, Ameet. Hyperband: A Novel Bandit-Based Approach to Hyperparameter Optimization. 2016. Dostupné z DOI: 10.48550/ARXIV.1603.06560.

14. PEDREGOSA, F.; VAROQUAUX, G.; GRAMFORT, A.; MICHEL, V.; THIRION, B.; GRISEL, O.; BLONDEL, M.; PRETTENHOFER, P.; WEISS, R.; DUBOURG, V.; VANDERPLAS, J.; PASSOS, A.; COURNAPEAU, D.; BRUCHER, M.; PERROT, M.; DUCHESNAY, E. Scikit-learn: Machine Learning in Python. *Journal of Machine Learning Research*. 2011, roč. 12, s. 2825–2830.

15. REFAEILZADEH, Payam; TANG, Lei; LIU, Huan. Cross-validation. *Encyclopedia of database systems*. 2009, roč. 5, s. 532–538.

16. BOUTABA, R.; SALAHUDDIN, Mohammad; LIMAM, Noura; AYOUBI, Sara; SHAHRIAR, Nashid; ESTRADA-SOLANO, Felipe; CAICEDO, Mauricio. A Comprehensive Survey on Machine Learning for Networking: Evolution, Applications and Research Opportunities. *Journal of Internet Services and Applications*. 2018, roč. 9. Dostupné z DOI: 10.1186/s13174-018-0087-2.

17. SHRIYAL, Swati Shivkumar; AINAPURE, Bharati Sanjay. IoT Device Classification Techniques and Traffic Analysis - A Review. In: *2021 International Conference on Technological Advancements and Innovations (ICTAI)*. 2021, s. 244–250. Dostupné z DOI: 10.1109/ICTAI53825.2021.9673420.

18. REYNOLDS, Joyce K.; POSTEL, Dr. Jon. *Assigned Numbers* [RFC 1700]. RFC Editor, 1994. Request for Comments, č. 1700. Dostupné z DOI: 10.17487/RFC1700.

19. FINAMORE, Alessandro; MELLIA, Marco; MEO, Michela; ROSSI, Dario. KISS: Stochastic Packet Inspection Classifier for UDP Traffic. *IEEE/ACM Transactions on Networking*. 2010, roč. 18, č. 5, s. 1505–1515. Dostupné z DOI: 10.1109/TNET.2010.2044046.

20. BERMOLEN, Paola; MELLIA, Marco; MEO, Michela; ROSSI, Dario; VALENTI, Silvio. Abacus: Accurate behavioral classification of P2P-TV traffic. *Computer Networks*. 2011, roč. 55, č. 6, s. 1394–1411. ISSN 1389-1286. Dostupné z DOI: https://doi.org/10.1016/j.comnet.2010.12.004.

Obsah přiloženého média

- `README.txt` ..... stručný popis projektu
- `bbndia` ..... zdrojové kódy implementace
  - `alg` ..... implementace klasifikačních algoritmů
  - `notebooks` ..... jupyter notebooky s experimenty
  - `run` ..... spustitelné skripty
  - `utils` ..... implementace pomocných funkcionalit
- `Makefile`
- `text` ..... text práce
- `thesis.pdf` ..... text práce ve formátu PDF