FAKULTA INFORMAČNÍCH TECHNOLOGIÍ ČVUT V PRAZE

Zadání diplomové práce

Název:	Automatická optimalizace datových sad síťového provozu
Student:	Bc. Petr Skružný
Vedoucí:	Ing. Dominik Soukup
Studijní program:	Informatika
Obor / specializace:	Počítačová bezpečnost
Katedra:	Katedra informační bezpečnosti
Platnost zadání:	do konce letního semestru 2022/2023

Pokyny pro vypracování

Nastudujte problematiku monitorování síťového provozu pomocí síťových toků a jejich využití pro tvorbu datových sad. Dále se seznamte s využitím strojového učení nad datovými sadami pro analýzu síťového provozu.
Navrhněte algoritmus pro vyhodnocení datových sad ze síťových toků a optimalizaci datových sad (kritérium např. velikost a úplnost datové sady).
Navržený algoritmus implementujte jako SW prototyp a otestujte pomocí dat dodaných vedoucím práce.

Elektronicky schválil/a prof. Ing. Róbert Lórencz, CSc. dne 14. ledna 2022 v Praze.

FAKULTA
INFORMAČNÍCH
TECHNOLOGIÍ
ČVUT V PRAZE

Diplomová práce

Automatická optimalizace datových sad síťového provozu

Bc. Petr Skružný

Katedra informační bezpečnosti
Vedoucí práce: Ing. Dominik Soukup

4. května, 2022

Poděkování

děkuji vedoucímu práce za trpělivost a ochotu při vypracovávání této práce

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.

Praha dne 4. května, 2022 . . . . . .

Odkaz na tuto práci

Skružný, Petr. *Automatická optimalizace datových sad síťového provozu.* Diplomová práce. Praha: České vysoké učení technické v Praze, Fakulta informačních technologií, 2022.

Abstrakt

S rostoucím objemem šifrovaného provozu v síti narůstá potřeba korektní identifikace a monitorování tohoto provozu. Pro řešení toho problému se využívá algoritmů strojového učení, které je ale nejdříve nutné naučit na trénovací datovou sadu. Nejvhodnější je pak pro učení používat i záznamy z reálného provozu. Cílem této práce je analýza možných metod optimalizace datových sad síťového provozu, následný návrh možných algoritmů pro optimalizaci datových sad, jejich implementace a experimentální vyhodnocení těchto algoritmů. Výstupem práce pak je srovnání experimentálních výsledků navržených metod a jejich softwarový prototyp implementovaný v programovacím jazyce Python.

Klíčová slova optimalizace datových sad, autoenkódery, strojové učení, Python

Abstract

The need for precise identification and monitoring of encrypted network traffic grows accordingly to the growing volume of encrypted network traffic. The solution to this problem often lies in utilization of machine learning algorithms. Before such algorithms can be employed they have to be trained using prepared training datasets. It is best to use dataset which are in part comprised of real world network traffic. Goal of this theses is to analyze potencial method for network traffic dataset optimization and propose viable algorithms for optimizing these datasets. The proposed algorithms will be implemented and experimentaly evaluated. Results of this theses are comparison of experimental results of proposed algorithms and software prototype of such algorithms implemented in Python programming language.

Keywords dataset optimalization, autoencoders, machine learning, Python

Obsah

Seznam obrázků

Seznam tabulek

Kapitola 1

Úvod

V moderním světě dochází ke stále většímu sdílení informací a hlavním médiem, které pro sdílení používáme je právě Internet. S trochou nadsázky bychom mohli označit současnou dobu, věkem Internetu. Tato propojenost přináší obrovské výhody ať už z pohledu kooperace, výzkumu, obchodu, či jiných forem spolupráce, pro mnoho subjektů využívající Internet. Bohužel tato propojenost s sebou nese i řadu stinných stránek, ve formě různých podvodných webů, hackerských skupin, botnetů, či vládních organizací, které používají Internet jako médium pro šíření dezinformací. Dobrým příkladem může být aktuální informační válka vedená souběžně s probíhajícím konfliktem na území Ukrajiny.

Jako prevence těchto možných zneužití pak byly vytvořeny různé metody zabezpečení, které mají zabraňovat únikům citlivých dat organizací i jednotlivců, o které se mohou právě nekalé organizace pokoušet. Velkým skokem pro bezpečnost na Internetu pak byla implementace šifrování přenášených dat. Pro dnešního uživatele je takovou formou šifrování například použití protokolu HTTPS pro komunikaci s webovým serverem. Bohužel tato výhoda je zároveň poskytnuta i živlům, které mají nekalé úmysly. Z toho důvodu je nutná důkladná kontrola síťové komunikace, abychom dokázali odhalit tyto nebezpečné živly a buď jejich pokusy o útoky zastavili již v prvopočátku, nebo abychom alespoň případnou kompromitaci našich dat odhalili co nejdříve a dokázali jsme na vzniklou situaci korektně reagovat. V tomto případě pak šifrování komunikace představuje více problém než přínos. Samozřejmě existuje řada možností jak tento problém obejít, příkladem může být použití proxy ve firemní síti, která dešifruje veškerou komunikaci uvnitř sítě a po analýze data opět zašifruje a odešle dál. Tím ale často nechtěně vystavujeme další možný útočný vektor pro potenciálního útočníka, který by prolomením ochrany proxy serveru okamžitě získal přístup k veškeré komunikaci na síti. Navíc toto řešení nelze aplikovat na sítě sdílené více různými subjekty, jako jsou například sítě poskytovatele internetového připojení.

Zde pak přichází na scénu analýza šifrovaných dat na základě vlastností jejích posloupností paketů, či samotných paketů, případně skupin paketů. Takto analyzovaná data je pak třeba řádně označit, abychom měli k dispozici informace pro zaručení bezpečnosti dat organizace a jejích klientů. Pro označení dat jsou ve velké míře používány algoritmy strojové učení, které jsou schopné reagovat na menší změnit v označovaných datech a samy se jim přizpůsobovat a navíc jsou díky znalosti dat schopné odhalit případné anomálie. Obtížným problémem pro strojové učení je tvorba datových sad, které jsou použity pro naučení strojového učení. U těchto sad totiž požadujeme řadu vlastností, tak aby výsledný naučený algoritmus poskytoval přesné výsledky.

Souběžně s touto prací vzniká prototyp aktivního učení, který po prvotním naučení, využívá pro přeučení a zdokonalování zachycený provoz. Tento přístup však naráží na problém, kdy datová sada, která je použita pro učení postupem času nabývá obrovských rozměrů. Záměrem této práce je prozkoumat existující řešení pro redukci velikosti datových sad a navrhnout a otestovat několik algoritmů pro redukci velikosti datové sady a zhodnotit, zda algoritmy poskytují vzniklé datové požadované vlastnosti, jako je například přesnost, úplnost, vyváženost, apod...

Kapitola 2 si klade za cíl seznámit čtenáře s pojmy vyskytujícími se v tomto dokumentu a jejich aplikací. Kapitola 3 pak představí řadu metod pro optimalizaci problémů. Po ní následující kapitola 4 popisuje řadu existujících řešení a diskutuji v ní použitelnost těchto řešení pro zadaný problém. V kapitolách 5, 6 a 7 jsou pak postupně navržený, implementovány a otestovány metody optimalizace nad zadanou datovou sadou. Kapitola 8 pak diskutuje naměřené výsledky metod a jejich implikaci a použitelnost. V závěru je pak poskytnuto shrnutí průběhu a výsledků celé práce.

Kapitola 2

Rešerše

Problematika optimalizace datových sad a strojového učení je poměrně obsáhlé téma. Proto si v této kapitole kladu za cíl vysvětlit problematiku sběru síťových dat, co vlastně je datová sada, co to jsou klasifikátory a anotátory a jak jsou spjaté se strojovým učením. Následně se v sekci 2.5 zaměřím na vysvětlení co vlastně je strojové učení. Po vysvětlení pojmů strojového učení se zaměření dočasně přesune na Bayesův teorém v sekci 2.6, následovaným metrikou F skóre a matematické metody PCA a SVD pro analýzu dat.

Po vysvětlení těchto pojmů přijde na řadu algoritmus K-means a problematika hledání počtu shluků v sekci 3.3. Následuje obeznámením s myšlenkou genetických algoritmů a představení algoritmů simulovaného žíhání v 3.5, Particle Swarm Optimization v 3.6 a optimizace pomocí mravenčí kolonie v 3.7.

Po představení genetický algoritmů a jejich variant bude následovat krátká zastávka u tabu vyhledávání a následně přijde na řadu představení Rough set teorie v sekci 3.9 a jejích variant TSAR a RSAR.

Jako poslední pojem budou představeny autoenkodéry, jejich fungování a druhy.

2.1 Sběr dat

Než je možné vůbec jakákoliv data optimalizovat, je nutné je nejdříve někde získat. Sběr dat je možné provádět mnoha způsoby. Pro síťová data je možné využít sond na síťových prvcích, přes které probíhá tato komunikace, strojích přímo dedikovaných za tímto účelem, nebo třeba i na koncových zařízeních uživatelů. Zpravidla pak sbíráme jednotlivé pakety komunikace. Problémem v tomto ohledu je nezměrné množství dat, které takto získáme. Ukládat obrovské množství dat rozhodně není v ničím zájmu. Na tento problém naštěstí existuje několik možných řešení. Dvě řešení, které bych rád zmínil jsou rozdělení paketů do takzvaných dávek paketů (packet bursts)[1], založené na myšlence, že příchozí a odchozí komunikace funguje na bázi požadavku a odpovědi a, že dávky paketů v jednom směru v rámci jednoho požadavku nebo odpovědi

pak lze sloučit. Druhým řešením je pak sdružování paketů do toků (flows) [2], které jsou ve výsledku schopné poskytnout více informaci, jelikož sdružují spolu související pakety a lze díky nim získat celkový obraz dané komunikace. Sbíraná data jsou následně předávána dalším zařízením, které je dále zpracují, například systému NEMEA [3], který dokáže data analyzovat a v případě detekce anomálie, tyto anomálie hlásit. Systém NEMEA se řadí do kategorie nástrojů pro detekci útoku po síti (Network Intrusion Detection Systems). Nástroj vznikl díky potřebě detekovat i sofistikovanější sítové útoky, které často probíhají až na aplikační vrstvě komunikace a běžné systémy je tedy nedokáží detekovat.

Sběr dat a předzpracování proběhlo na síti CESNET a tato data tvoří datovou sadu, nad kterou má tato práce za cíl otestovat navržené metody optimalizace pro, pokud možno maximální redukci velikosti, při minimální ztrátě přesnosti.

2.2 Datová sada

Celou práci prolíná pojem datová sada. Dovolím si zde použít definice z článku [4]. Datovou sadou rozumíme strukturovanou množinu vstupních dat. Struktura a obsah těchto dat se liší na základě domény, které se sada týká. Pro úspěšné naučení modelu strojového učení je pak nutné aby data v sadě byla správně anotována.

Míru kvality datových sad, které vzniknou během této práce je pak třeba nějakým způsobem určit. Proto zavedu několik pojmů, které budu používat pro určení kvality.

Dobrá datová sada: Taková datová sada, která splňuje následující dvě podmínky: 1) každý řádek se sady je anotován, 2) sada obsahuje dostatek dat pro naučení algoritmu strojového učení, 3) sada dosahuje určitých metrik daných algoritmem (v této práci se zaměřuji především na přesnost a úplnost).

Úplná datová sada: Datová sada pokrývající všechny typy anotovaných příznaků. V článku [4] je míra úplností uvedena jako vektor pravděpodobností úplnosti C_k:

$$C_k = P(x|x = L_k), \forall k \ \in \{0, 1, \ldots, dim(L)\},$$

kde L je vektor správných anotací, $dim(L)$ je počet různých anotací a x je pozorovaná hodnota v datové sadě D.

2.3 Klasifikátory

Protože často pracujeme s velkým množstvím komplexních dat, tak není v lidských silách ani zájmu tato data ručně analyzovat a zařazovat. V těchto případech

vystupují do popředí metody strojového učení, schopné s určitou mírou přesnosti práci člověka zcela nahradit. Nazýváme je klasifikátory. Aby ale tuto činnost mohli vykonávat je nejdříve nutné tyto metody naučit, jako člověka nově nastupujícího na novou pracovní pozici, jak data zařadit. Pro přípravu testovacích dat, nad kterými se model klasifikátoru natrénuje, využíváme anotátory popsané v 2.4. Tato trénovací sada již obsahuje pro každý svůj záznam i třídu do jaké ho má klasifikátor zařadit. [5] Klasifikátor je pak po natrénování nad touto sadou schopen klasifikovat nově příchozí do předem specifikovaných kategorií, aniž by vyžadoval dohled člověka, či pomoc anotátoru. Příkladem relevantním pro tuto práci je právě klasifikace síťového provozu, kde klasifikátor, který je naučený trénovací sadou, automaticky přiřazuje třídy jednotlivým tokům síťového provozu a tím usnadňuje dohled nad danou sítí.

Klasifikátor lze popsat jako funkci, která přijímá hodnoty různých proměnných a přiřadí jim odpovídající třídu ohodnocení. Pro klasifikátor je důležitá kvalita trénovací sady. Bez dostatečně kvalitní trénovací sady může dojít k případu kdy klasifikátor neodhalí všechny vazby mezi parametry, případně se naučí špatné vazby, a produkuje pak chybné výsledky.

U trénovacích sad se obecně uvažuje, že čím je sada větší, tím je lepší. Tato myšlenka vychází z předpokladu, že pokud v sadě máme i nějaké špatně označené záznamy, které nám reprezentují šum, tak při dostatečné velikosti sady je klasifikátor schopen toto odhalit a záznam označit správně. To s sebou okamžitě přináší první problém a tím je volba velikosti datové sady. S příliš velkou trénovací sadou, bude doba učení, případného přeučení klasifikátoru velmi dlouhá a v případě mnoha tříd přichází v potaz i velikost, kterou data zaberou na disku. Výhodou by pak naopak měla být vysoká přesnost. Pokud však je trénovací sada příliš malá, může u klasifikátoru dojít k případu, kdy nebude schopen rozeznat šum od korektního záznamu a výsledky klasifikace budou velmi nepřesné. Cílem této práce je optimalizovat datové sady, tak aby model klasifikátoru nad ním trénovaný poskytoval výsledky s pokud možno co nejmenší ztrátou přesnosti a zároveň aby sada byla pokud možno co nejmenší. V rámci testování bude primárně používán klasifikátor typu AdaBoost.

AdaBoost klasifikátor (Adaptive Boosting klasifikátor), neboli klasifikátor adaptivního posilování, je přístup strojového učení založený na ideji vytváření vysoce přesného předpovídajícího pravidla na základě kombinací mnoha relativně slabých a nepřesných pravidel. První praktickou aplikací tohoto přístupu je AdaBoost algoritmus od Freund a Schapira [6], který zůstává jedním ze nejvíce používaných a studovaných algoritmů dodnes.

Vstupem algoritmu je m anotovaných příkladů $(x_1, y_1), \ldots, (x_m, y_m)$, kde x_i leží v nějaké doméně X a značky $y_i \in \{-1, +1\}$. Každá iterace algoritmu probíhá následovně podle obrázku 2.1:

Obrázek 2.1 Iterace AdaBoost

for $t = 1, \ldots, T$ **do**
Natrénuj slabý klasifikátor s rozdělením D_t.
Získej slabou klasifikaci $h_t : X \to \{-1, +1\}$.
Vyber h_t s nejnižší váženou chybou:

$$\epsilon_t = Pr_{i \sim D_t}[h_t(x_i) \neq y_i]$$

.
Zvol $\alpha_t = \frac{1}{2} \ln(\frac{1-\epsilon_t}{\epsilon_t})$.
for $i = 1, \ldots, m$ **do**

$$D_{t+1}(i) = \frac{D_t(i) \exp -\alpha_t y_i h_t(x_i)}{Z_t}$$

▷ kde Z_t je normalizační faktor, takový aby D_{t+1} byla distribuce.
end for
end for

Výstupem algoritmu pak je závěrečná klasifikace:

$$H(x) = sign\left(\sum_{t=1}^{T} \alpha_t h_t(x)\right) \quad .$$

Ekvivalentní tvrzení k tomuto je, že H je spočítáno jako vážený většinový hlas slabých klasifikátorů h_t, kde každé klasifikaci je přiřazena váha α_t.

2.4 Anotátory

V předchozí sekci byla zmíněna potřeba přípravy trénovací sady pro trénování klasifikátorů. Tyto klasifikátory pak lze využít k automatizaci různých úkonů. Právě tato příprava trénovacích sad, obecněji označování dat nějakým příznakem, je problémem na který doposud nebylo nalezeno jednoznačné řešení ani metodika jak k němu přistupovat. Problémem je, že klasifikátory založené na principu učení s dohledem nejsou použitelné bez anotované trénovací sady. Problém lze obejít použitím klasifikátoru na bázi učení bez dohledu, případně učení se zpětnou vazbou, ale výsledky těchto klasifikátorů bývají zpravidla horší, případně trvá poměrně dlouhou dobu, než se výsledky dostanou na úroveň učení s dohledem nebo učení s částečným dohledem. Z toho vyvstává nutnost použití anotátorů, ať už lidských či strojových. [7]

U lidských anotátorů vyvstává hned několik otázek. Jaké množství dat je schopen přesně jeden člověk anotovat? Anotuje daný člověk data tak, jak bysme chtěli? Můžeme pro anotování využít více lidí? Začněme poslední otázkou. Pro anotování dat můžeme zcela určitě použít více lidí. Běžným předpokladem

u učení s dohledem je, že čím více anotovaných dat pro trénink máme k dispozici, tím přesnější výsledky by klasifikátor měl vykazovat. Tento přístup kdy dochází k přesunu závislosti z jednoho anotátora na více nazýváme *crowdsourcing* efekt [8]. To nás vede přímo na druhou otázku, pro kterou nemáme obecnou odpověď. Anotace v rámci nějaký skupiny anotátorů musí probíhat na základě předchozí domluvy podle stanovených pravidel, jinak není možné dosáhnout jednotné anotace. Postupně se pak dostáváme k první otázce. Obecně nelze tvrdit, že člověk je schopen anotovat velké množství dat přesně. Člověk vždy udělá nějakou chybu, ať už jde o špatnou identifikaci anotovaného záznamu, či prostě o překlep při anotaci. S rostoucím množstvím dat, které daný člověk musí zpracovat roste úměrně i pravděpodobnost chyby. Dalším extrémem, na který můžeme narazit jsou falešní anotátoři (spammeři)[9]. Jejich cílem je chybně anotovat záznamy, za účelem poškození výsledné sady, která pak může vést ke kompromitaci, či jen prostě ke špatnému fungování, klasifikátoru.

Kvůli výše zmíněným problémům se pak obracíme na strojové anotátory, jenž by je měly řešit. Okamžitě však narážíme na nový problém. Pokud neznáme přesné parametry podle kterých anotovat, jsme víceméně odkázáni na strojové učení a vniká nám nekonečná smyčka. Lidský anotátor bude tedy vždy potřeba, alespoň na přípravu trénovací sady pro anotátor založený na strojovém učení. Pokud bysme znali přesné parametry pro určení anotace, pak nepotřebujeme použít strojového učení či aproximačních algoritmů. Na základě známých parametrů jsem totiž schopni naprosto přesně záznamy anotovat.

Dostáváme se pak ke kompromisu, kdy spojujeme všechny přístupy dohromady a eliminujeme tím jejich nevýhody. Začínáme u důvěryhodných lidských anotátorů, kteří připraví trénovací sady pro specializované anotátory využívající strojové učení. Anotátoři jsou schopni na základě znalosti anotované sady upravit algoritmus strojového anotátoru, tak aby produkoval přesné výsledky. Tyto přesné výsledky pak mohou sloužit jako trénovací sada pro klasifikátory. Řešíme tak výše zmíněné problémy, strojové anotátory výrazně sníží chybovost anotace, lidští anotátoři zajistí přípravu úvodní trénovací sady pro strojový anotátor a specializací anotátorů získáme přesné výsledky, které můžeme agregovat do jedné sady pro klasifikátor. Problémem, který tento přístup neřeší je jednotnost stylu anotace, ale jak bylo vysvětleno dříve, zde jde o záležitost domluvy a stanovení pravidel.

2.5 Strojové učení

Strojové učení je souhrnné označení skupiny algoritmů, využívajících známých předchozích vstupů a jejich ohodnocení pro predikci ohodnocení příštích vstupů. Algoritmus se z této sady "naučí"jaké příznaky mají jednotlivé položky a při běhu s jinými daty je používá jako referenci. Zároveň také nově označené příznaky či metriky během provozu používá pro zlepšení sebe sama.

Obecně strojové učení dělíme do několika kategorií podle přístupu k učení algoritmu.

- Učení s dohledem

  Data pro budování modelu obsahují kromě vstupu, také požadovaný výstup, jinak řečeno se jedná o trénovací sadu. Algoritmus na základě těchto dat vybuduje svůj model a je dle něj schopen určovat ohodnocení nových vstupů, které nebyli v datové sadě. [10]

  Mezi metody učení s dohledem se řadí aktivní učení, klasifikace a regrese. [11]

- Učení s částečným dohledem

  Jakási střední cesta mezi učením s dohledem a učením bez dohledu, kdy část testovacích dat obsahuje očekávaný výstup a zbytek dat jej nemá. Tento přístup se jeví jako velmi silný, protože model na něm trénovaný často poskytuje vysokou přesnost. [12]

- Učení bez dohledu

  Model je vytvořen na základě dat obsahujících pouze vstupy, algoritmus prozkoumá strukturu dat a hledá podobnosti mezi jednotlivými vstupy. [13]

- Učení se zpětnou vazbou

  Metoda často využívaná v teorii her, informační teorii, či více agentních systémech. Zaměřuje se na to jaké akce by agent, potažmo model strojového učení, měl učinit pro maximalizace jakési souhrnné odměny. Ve strojovém učení je toto prostředí často reprezentováno jako Markovský rozhodovací proces. Často je implementován pomocí dynamického programování a v reálném světe bývá používán pro autonomní vozidla či umělou inteligenci hrající proti lidskému oponentovi. [14]

- Redukce dimenze

  Proces snižování počtu proměnných získáváním sady hlavních proměnných. Jednou z nejpopulárnějších metod je metoda PCA, která redukuje data více dimenzí (3 a více) do dimenze menší, zpravidla dimenze 2. [15]

2.6 Bayesův teorém

Bayesův teorém je fundamentálním stavebním blokem pro deduktivní statistiku a mnoho pokročilých modelů strojového učení. Jde o logický přístup k upravování pravděpodobnosti hypotéz na základě nových důkazů. Tento přístup umožňuje získávání odpovědí na otázky, na které doposud frekvenční statistické přístupy nedokáží odpovědět. [16]

Nejjednodušší formou Bayesova teorému je rovnice

$$P(A|B) = \frac{P(B|A)P(A)}{P(B)} \quad .$$

Tuto rovnici lze aplikovat na výpočet pravděpodobnosti hypotézy na základě pozorovaných dat:

$$P(hypotéza|data) = \frac{P(data|hypotéza)P(hypotéza)}{P(data)} \quad ,$$

kde $P(data|hypotéza)$ je pravděpodobnost pozorování těchto dat, za podmínky, že hypotéza je pravdivá. $P(hypotéza)$ je a priori pravděpodobnost hypotézy, také označováno jako stupeň důvěry v hypotézu, a $P(data)$ je pravděpodobnost pozorování dat, bez ohledu na to zda hypotéze platí či nikoliv.

2.7 F skóre

F skóre, případně F_1 skóre je populární způsob reprezentace kvality algoritmu strojového učení. Jeho popularita vychází, ze snadné interpretace hodnoty tohoto skóre, která se pohybuje v intervalu ($0 \leq F \leq 1.0$). Samotná hodnota je vypočítána jako harmonický průměr přesnosti P a recall R pomocí vzorce

$$F = \frac{2PR}{P+R} \quad .$$

Přesnost P reprezentuje podíl pravdivých výsledků a všech výsledků, které algoritmus označil jako pravdivé, včetně těch, které ve skutečnosti pravdivé nejsou.

Recall R pak reprezentuje podíl pravdivých výsledků a všech výsledků, které měli být označeny jako pravdivé. Recall bývá také označován jako citlivost. [17]

F_1 skóre vychází z obecné rovnice pro F skóre, která je následující

$$F_\beta = \frac{(\beta^2+1)PR}{\beta^2 P + R} \quad , (0 \leq \beta \leq +\infty).$$

Parametr β určuje poměr mezi P a R. Pokud $\beta = 1$, získáme F_1, tedy harmonický průměr mezi P a R. Jestliže by $\beta < 1$, pak se funkce F_β zaměřuje na přesnost a získali bysme $F_0 = P$. Alternativně pakliže $\beta > 1$, funkce se orientuje na recall.

Pro potřeby této práce používám vyváženou variantu F skóre, tedy $\beta = 1$ a z toho plyne

$$F_1 = \frac{2PR}{P+R} \quad .$$

Kapitola 3

Metody optimalizace

3.1 PCA

PCA, česky analýza hlavních komponent, je metoda strojového učení bez dohledu s hlavním využitím ve zmenšování dimenzí. PCA je počítáno, buď pomocí SVD navrhované matice, nebo konstrukcí kovarianční matice dat a dekompozicí této matice pomocí vlastních čísel. [18] Cíle PCA jsou:

1. získat nejdůležitější informace z datové sady,
2. zmenšit velikost sady zachováním pouze zmiňovaných důležitých informací,
3. zjednodušit popis sady,
4. analyzovat strukturu pozorovaných dat a proměnných.

Aby PCA dosáhlo těchto cílů musí vypočítat nové proměnné, zvané hlavní komponenty (odtud pochází název metody), které vzniknou lineární kombinací původních proměnných. Po první komponentě požadujeme aby měla co největší možnou variaci, neboli aby získala co největší množství informace z původní sady. Druhá komponenta je vypočítána za podmínek, že musí být ortogonální k první hlavní komponentě a opět musí mít co největší variaci. Další hlavní komponenty jsou pak počítány obdobně jako druhá komponenta. Hodnoty těchto nových proměnných nazýváme faktorové ohodnocení a lze je geometricky interpretovat jako projekce na hlavní komponenty.

3.2 SVD

Singular value decomposition je zobecněním dekompozice za pomoci vlastních čísel pro obdélníkové matice. Standardní dekompozice pomocí vlastních čísel je definována pouze pro čtvercové matice. Hlavní ideou SVD je rozložení matice

do tří menších matic, dvou ortogonálních a jedné diagonální matice. V běžné dekompozici pomocí vlastních čísel je matice rozložena do dvou jednoduchých matic. Výkonnostně je SVD považována za stejně silnou metodu jako PCA a multidimenzionální škálování (MDS). [19]

3.3 K-means

K-means je shlukovací algoritmus, řešící problém nalezení k bodů, zvaných středy, případně centroidy, v prostoru R, dimenze d, takových, aby střední hodnota druhé mocniny vzdálenosti každého bodu od nejbližšího středu byla minimální. [20]

Metrika pro určování středů a jejich vzdálenost od bodů je těžce závislá na konkrétním problému. Výše uvedená metrika je v literatuře uváděna pouze orientačně, protože ji není možné vždy použít.

Složitým problémem u shlukování pomocí K-means nebo jeho variant (K-medians,...) je správná volba parametru K, jelikož v drtivém množství případů nelze tuto hodnotu odhadnout pouhým náhledem dat. Nejstarší metodou pro určování hodnoty K je Elbow (loketní) metoda. Jedná se o takzvanou vizuální metodu, protože výběr výsledné hodnoty K je nejlépe určitelný pouhým okem.

Elbow metoda začíná s hodnotou $K = 1$ a v každém kroku hodnotu navyšuje o 1. Pro každou hodnotu K metoda vypočítá hodnotu W objektivní funkce, běžně se jedná o střední kvadratickou chybu (mean squared error function). Cílem je hodnotu této funkce minimalizovat. Vypočtením hodnot W pro řadu několika hodnot K jsme pak následně schopni vytvořit graf této metody. Vhodným kandidátem na výsledné K je pak co nejmenší číslo, při němž již hodnoty W_K a W_{K+1} nejsou příliš rozdílné. V grafu je toto viditelné jako "loket"křivky. [21]

Problémem Elbow metody je určení samotného "lokte", který ne vždy lze přesně identifikovat. V některých případech je dokonce možné, že takovýchto "loktů"má graf několik, nebo nemá vůbec žádné. V takových případech nelze hodnotu K určit přesně a metoda je tedy krajně nevhodná.

Rule of thumb je další o poznání jednodušší, a velmi nepřesnou, metodou je takzvaná metoda "palce". Hodnota K se v tomto případě určuje podle následujícího vzorce.

$$K \approx \sqrt{n/2}$$

Kde n je počet prvků datové sady.

Skoková statistika je alternativní metodou pro výběr počtu shluků popsanou článkem [22] od dvojice Sugar James. Tento přístup dělá pouze omezené předpoklady parametrů, založené na idejích z teorie hodnocení rozptylu.

Jedná se o přístup jednoduchý na pochopení i výpočet, která je vysoce efektivní na široké spektrum problémů. Skoková statistika využívá hodnoty W s rozšířením dle Gaussova distribučního modelu. Přesněji řečeno se jedná o výpočet vzdálenosti bodu od středu shlukům která je počítána pomocí rovnice

$$d(i, c_k) = (y_i - c_k)^T {\Gamma_k}^{-1} (y_i - c_k),$$

kde Γ_k se nachází v kovarianční matici shluku.

Samotný skok je definován jako $JS(K) = W_K^{-M/2} - W_{K-1}^{-M/2}$, za předpokladu $W_0^{-M/2} \equiv 0$. Maximální skok $JS(K)$ koresponduje se správným počtem shluků.

G-means je zajímavým přístupem k problému hledání K popsaném ve výtažku [23]. Ten využívá statistických testů pro zjištění rozdělení dat přiřazených ke středu nějakého shluku pro aktuální iteraci a počet shluků K. Pokud pro daný střed zjištěné rozdělení neodpovídá Gaussovskému rozdělení, pak je střed nahrazen dvěma novými a hodnota K je zvýšna o 2. Následně proběhne standardní K-means algoritmus pro těchto $K + 2$ středů, pro zpřesnění řešení, a algoritmus se opakuje dokud se počet středů v kroku $i + 1$ nerovná počtu středů v kroku i.

V samotném statistickém testu proti sobě stavíme nulovou hypotézu H_0: *Data kolem středu jsou z Gaussovského rozdělení* a alternativní hypotézu H_A: *Data kolem středu nejsou z Gaussovského rozdělení.* Jestliže nezamítáme H_0, pak nerozdělujeme daný střed. Pakliže zamítáme H_0 ve prospěch H_A, pak střed rozdělujeme na dva nové. Autoři výtažku doporučují použití Anderson-Darling statistického testu, který byl ukázán jako empiricky silný. [24]

3.4 Genetické algoritmy

Rodina algoritmů s původem inspirovaným evoluční teorií. Algoritmy kódují možné řešení do struktury podobné chromozomu, nad kterým provádějí rekombinační operace, s cílem zachovat důležité informace. Na genetické algoritmy je převážně nahlíženo jako na optimalizační funkce, byť možnosti jejich použití jsou daleko širší. [25]

Běžná implementace genetického algoritmu začíná výběrem, typicky náhodným, počáteční populace. Po výběru dojde k ohodnocení populace pomocí fitness funkce a každému členu je přiřazena tato hodnota, určující jak moc je daný jedinec vhodný. Jedinci s vyšší hodnotou fitness funkce mají větší pravděpodobnost k propagaci a rozmnožení do další iterace populace. Jedinci s horší hodnotou jsou typicky vynechání, kromě určitých případů, kdy jsou ponechání za účelem úniku z lokálního minima celkové hodnoty populace.

Rozmnožení jedince lze, díky chromozomové reprezentaci, implementovat jako n-bodovou kombinaci vlastností dvou jedinců s vysokou hodnotou fitness funkce. Nově vniklý jedince, označovaný potomek, je pak zařazen do nové

iterace populace. Po vytvoření této nové iterace je na každého jedince s nízkou pravděpodobností, typicky menší než 1%, aplikován mutační operátor.

Mutační operátor je běžně implementován buď jako vygenerování nového bitu na náhodné pozici chromozomu jedince, nebo přehození hodnoty tohoto náhodně vybraného bitu. Jedná se spíše o implementační detail a záleží pouze na autorovi algoritmu, jak moc si přeje snížit či zvýšit dopad mutace na populaci.

Po provedení rozmnožení a aplikaci mutačního operátoru je opět možné nově vzniklou populaci ohodnotit. Celou tuto posloupnost operací, tedy ohodnocení, rozmnožení a mutaci, značíme jako jednu generaci. Celý průběh genetického algoritmu končí ve chvílí, kdy algoritmus dosáhne hledaného globálního optima nebo počet generací překročí maximální možnou povolenou hodnotu.

3.5 Simulované žíhání

Kombinatorické optimalizační problémy, kde se snažíme nalézt nějakou konfiguraci parametrů, která minimalizuje cenu, či objektivní funkci, zpravidla využívají nějaký z heuristických postupů pro nalezení této hodnoty. Jedním takovým postupem je konstruktivní postup, který se snaží budovat optimální řešení krok po kroku, což pro velký počet proměnných může být problém. Dalším možným přístupem je iterativní strategie zlepšení, ve které dochází k rekombinaci existujícího neoptimálního řešení za cílem nalezení lepšího řešení. Rutenbar v článku [26] zmiňuje, že problém obou postupů je uváznutí v lokálním minimu, ze které z definice těchto postupů nedokáže algoritmus uniknout.

Jako nejjednodušší řešení tohoto problému se může jevit použití mnoha náhodných inicializačních řešení. To s sebou ovšem přináší velké výpočetní náklady a není tedy pro většinové použití rozumné. Simulované žíhání nabízí řešení tohoto problému použitím postupu iterativního zlepšování s jedním markantním rozdílem, zavedením funkce žíhání, která umožňuje kontrolované zhoršení řešení. Tím, že v algoritmu nyní existuje funkce umožňující přijetí horšího řešení, získáváme možnost jak uniknout z lokálního minima. Zároveň díky tomu, že toto zhoršení je kontrolované, nemůže nastat situace, kdy se blížíme ke globálnímu minimu a funkce by náhodně skočilo na nějaké daleko horší.

Analogií tohoto procesu je právě žíhání pevných látek. Abychom přinutili materiál dostat se do stavu s nízkou energií, tak ho musíme nejdříve ohřát a následně pomalu ochlazovat, aby dosáhl teplotní rovnováhy při každé teplotě. Cílem je dostat částice v mřížce materiálu (konfigurace) do stavu s minimální energií (cenou). Za tímto účelem byl vyvinut Metropolis algoritmus [27] ukázaný v obrázku 3.1.

Myšlenkou algoritmu, stejně jako v iterativním zlepšování, je návrh nějaké nové kombinace konfigurace a poté spočítat výslednou změnu energie ΔE. Pokud je energie snížena, $\Delta E < 0$, má nová konfiguraci nižší energii (cenu) a

Obrázek 3.1 Metropolis algoritmus

```
1: M= maximální počet tahů;
2: T= aktuální teplota;
3: for m = 1, ..., M do
4:     Vygeneruj náhodný tah (pohyb částice);
5:     Spočítej změnu energie ΔE;
6:     if ΔE < 0 then                          ▷ Přijímáme zlepšující tah
7:         Přijmi tah a aktualizuj konfiguraci;
8:     else                                    ▷ Možná přijmi zhoršující tah
9:         Přijmi tah s pravděpodobností P = e^{-ΔE/T};
10:        Aktualizuj konfiguraci pokud byl tah přijat;
11:    end if
12: end for
```

přijímáme jí jako nový výchozí bod pro další tah. Pokud ale dojde k nárůstu energie $\Delta E > 0$, můžeme tento tah s vyšší energií přijmout. Taková situace nastává i ve fyzickém světě, ale je řízena teplotou T. Při vysokých teplotách je pravděpodobnost posunu do stavu s vyšší energií velká a se snižující teplotou klesá. Metropolis algoritmus tento fakt simuluje pomocí Boltzmannova rozdělení, kde pravděpodobnost přijetí počítáme jako $P[přijetí] = e^{-\Delta E/T}$. Postupným snižováním teploty jsme pak schopni simulovat přechod materiálu do vyvýženého stavu jako při reálném žíhání.

3.6 Particle Swarm Optimization

Původní motivací, která vedla ke vzniku Partice Swarm Optimization algoritmu, byla snaha odhalit pravidla chování ptačích a rybích hejn při prudké změně směru, či hledání potravy. Postupem času se pozornost přenesla na pravidla sociální interakce a později toto vedlo k vzniku samotné metody Particle Swarm Optimization (PSO).[28]

Základním stavebním prvkem PSO je objekt, nazývaný částice, skládající se ze tří vektorů dimenze D, kde D je dimenze vyhledávacího prostoru. Jedná se o aktuální pozici, předchozí nejlepší pozici a rychlost. Aktuální pozici lze chápat jako souřadnice v daném vyhledávacím prostoru. Předchozí nejlepší pozice je pozice částice v minulých iteracích s nejlepší hodnotou fitness funkce. Rychlostní vektor pak lze chápat jako velikost kroku při každé iteraci, který ale zároveň určuje směr posunu částic.

Akce PSO probíhají následovně. Předem určený počet částic je umístěn do vyhledávacího prostoru pro zadaný problém. Každá částice během iterace spočte hodnotu fitness funkce pro svojí aktuální polohu. Pokud je hodnota funkce lepší než předchozí nejlepší, tak si částice tuto hodnotu spolu s pozicí uloží. Samotný výpočet fitness funkce pak bere v potaz aktuální pozici

a nejlepší pozice jednoho nebo více sousedů hejna. Směr a velikost posunu určuje rychlostní vektor, který je po každé iteraci aktualizován. Jedna iterace algoritmu končí až poté co se všechny částice pohnou. Zde je vidět analogie s hejnem ptáků, či hledajících potravu, protože celé hejno začne postupně konvergovat k optimu daného prostoru.

3.7 Ant Colony Optimization

Optimalizace mravenčí kolonií (ACO) získala, jako mnoho dalších metod, inspiraci v chování některých druhů mravenců při hledání potravy. Mravenci zanechávají na zemi feromonovou spotu označující vhodnou cestu, kterou by ostatní mravenci měli využít. ACO používá obdobné metody pro řešení optimalizačních problémů, jak popisují ve svém článku [29] Dorigo, Birattari a Stützle.

Populace umělých mravenců v ACO pracuje na budování řešení pro zadaný optimalizační problém a vyměňují si mezi sebou informace o kvalitě svých řešení. Tato výměna informací je podobná výměně informací mravenců v reálném světě pomocí feromonů, kde čím více mravenců používá danou trasu, tím více se zvyšuje koncentrace feromonů a zvyšuje se i počet mravenců, kteří tuto cestu používají.

Obrázek 3.2 ACO meta-heuristika

```
1: Nastav parametry;
2: Inicializuj feromonové stopy;
3: while nebyla splněna ukončovací podmínka do
4:     Vybuduj_řešení_mravenci
5:     Aplikuj_lokální_vyhledávání                ▷ volitelná funkce
6:     Aktualizuj_feromony
7: end while
```

Jednoduchou ilustrací fungování ACO je aplikace na problém cestujícího obchodníka. V problému cestujícího obchodníka máme zadanou sadu měst a známe vzdálenosti mezi všemi městy. Cílem je nalézt nejkratší trasu, která navštíví všechna města právě jednou. V ACO tento problém reprezentujeme pomocí grafu, kde každý vrchol je město a hrana představuje cestu mezi dvěma městy. S každou hranou navíc spojujeme proměnou nazývanou feromon, kterou mohou mravenci číst a modifikovat. Na začátku náhodně umístíme mravence na vrcholy grafu. V každé iteraci algoritmu pak každý mravenec provede přesun ze stávajícího vrcholu na nový podle stochastického mechanismu ovlivněného feromonem. Pokud je mravenec na vrcholu i, tak je následující vrchol j vybrán stochasticky z ještě nenavštívených vrcholů. Pokud nebyl vrchol j ještě navštíven, je pravděpodobnost jeho výběru úměrná síle feromonů spojeného s hranou $e(i, j)$. Na konci každé iterace jsou hodnoty feromonů

přepočítány na základě kvality řešení mravenců, aby budoucí řešení byla podobná těm dosud nejlepším vytvořeným. Výsledné řešení pak má podobu posloupnosti měst, v pořadí v jakém mají být navštívena.

Obecně lze ACO definovat pomocí meta-heuristiky pro kombinatorické optimalizační problémy. Průběh této meta-heuristiky lze vyjádřit následujícím pseudokódem 3.2.

- *Vybuduj_řešení_mravenci*: množina m mravenců vybuduje řešení z prvků konečné množiny možných komponent $C = \{c_{ij}, c_{ij} = (i, j)\}$. Budování řešení začíná z prázdného částečného řešení $s^p = \emptyset$. V každém kroku konstrukce je s^p rozšířeno přidáním přijatelného řešení z množiny $N(s^p) \subseteq C$, která je definována jako množina komponent, kterou lze přidat k současnému částečnému řešení s^p bez porušení omezovacích podmínek. Proces konstrukce řešení si lze představit jako procházku po hranách grafu.

  Volba komponent z $N(s^p)$ je řízena mechanismem ovlivňovaným feromony spojenými s každým prvkem $N(s^p)$. Pravidlo pro výběr tímto mechanismem se liší mezi různými variantami ACO.

- *Aplikuj_lokální_vyhledávání*: Procedura, kterou je možné vylepšit získaná řešení z předchozího kroku. Tato fáze je velmi úzce spjatá se specifikací řešeného problému a nelze ji snadno popsat. Většina state-of-the-art ACO algoritmů však tuto proceduru využívá.

- *Aktualizuj_feromony*: Cílem aktualizace feromonů je zvýšit hodnotu feromonu pro hrany asociované s dobrými nebo slibnými řešeními a snížit pro špatná řešení. Většinou je tato operace provedena ve dvou krocích:

  1. Snížení hodnot všech feromonů, tzv. vypařování feromonů.
  2. Zvýšení hodnoty feromonů spojených se vybranou množinou dobrých řešení.

3.7.1 Hlavní varianty ACO

Any System (AS) je první z variant algoritmu. Hlavní charakteristikou této varianty je, že při každé iteraci dojde k aktualizace hodnot feromonů všemi m mravenci, kteří zkonstruovali v dané iteraci. Hodnota feromonu spojeného s danou hranou je aktualizována pomocí následující rovnice:

$$\tau_{ij} \leftarrow (1 - \rho) \cdot \tau_{ij} + \sum_{k=1}^{m} \Delta\tau_{ij}^{k} \quad ,$$

kde ρ je míra vypařování feromonů, m je počet mravenců a $\Delta\tau_{ij}^{k}$ je množství feromonu na hraně (i, j) položeného mravencem k, která nabývá hodnot:

$$\Delta\tau_{ij}^{k} = \begin{cases} Q/L_k & \text{pokud mravenec } k \text{ použil hranu } (i, j), \\ 0 & \text{jinak,} \end{cases}$$

kde Q je konstanta a L_k je délka cesty konstruované mravencem k.

MAX-MIN Ant System (MMAS) je vylepšením varianty AS, kde pouze mravenec s nejlepším řešením aktualizuje hodnotu feromonů a hodnota feromonu je omezená shora i zdola. Přesněji řečeno,

$$\tau_{ij} \leftarrow \left[(1-\rho)\cdot\tau_{ij} + \Delta\tau_{ij}^{best}\right]_{\tau_{min}}^{\tau_{max}} \quad ,$$

kde τ_{min} a τ_{max} jsou dolní a horní mez hodnoty feromonu. Operátor $[x]_b^a$ je definován:

$$[x]_b^a = \begin{cases} a & \text{pokud x} > \text{a}, \\ b & \text{pokud x} < \text{b}, \\ x & \text{jinak}, \end{cases}$$

a $\Delta\tau_{ij}^{best}$ je:

$$\Delta\tau_{ij}^{best} = \begin{cases} 1/L_{best} & \text{pokud hrana } (i,j) \text{ patří do nejlepší cesty}, \\ 0 & \text{jinak}, \end{cases}$$

Ant Colony System (ACS) přidává proceduru lokální aktualizace feromonů navrch k aktualizaci feromonů na konci konstrukční procedury. Lokální aktualizaci feromonů provádí všichni mravenci po každém konstrukčním kroku. Tato aktualizace se týká pouze poslední použité hrany:

$$\tau_{ij} = (1-\varphi)\cdot\tau_{ij} + \varphi\cdot\tau_0 \quad ,$$

kde $\varphi \in (0,1]$ je koeficient rozpadu feromonu a τ_0 je původní hodnota feromonu.

Hlavním cílem lokální aktualizace je diverzifikace hledání, které mravenci budou provádět v dalších iteracích. Snížením hodnoty právě použité hrany jsou ostatní mravenci motivováni k použití jiných hran a tím pádem k tvorbě jiných řešení. Dochází tak ke zmenšení pravděpodobnosti, že mravenci vyprodukují identická řešení.

Aktualizaci feromonů na konci konstrukční procedury provádí pouze nejlepší mravenec, podobně jako v MMAS. Nejlepší mravenec může být vybrán buď podle toho zda byl nejlepší v iteraci, či celkově nejlepší doposud. Na rozdíl od MMAS je ale rovnice pro aktualizaci trochu jiná:

$$\tau_{ij} = \begin{cases} (1-\rho)\cdot\tau_{ij} + \rho\cdot\Delta\tau_{ij} & \text{pokud hrana } (i,j) \text{ patří do nejlepší cesty}, \\ \tau_{ij} & \text{jinak}, \end{cases}$$

jako v MMAS je $\Delta\tau_{ij} = 1/L_{best}$ a L_{best} může být buď L_{ib} nebo L_{bs}.

3.8 Tabu vyhledávání

Heuristická metoda pro vyhledávání nejlepšího řešení v daném stavovém prostoru. Hlavní vlastností tohoto typu vyhledávání je schopnost zapamatování navštívených informačních procesů nebo řešení, díky čemuž je metoda schopná prozkoumávat řešení i mimo lokální optima. Algoritmus s přizpůsobivou pamětí je schopen efektivně generovat sousední řešení z aktuálního a přijímá nejlepší řešení i kdyby nebylo lepší než stávající řešení. Paměť obsahuje seznam předchozích řešení a zabraňuje tak zacyklení a návštěvám již prozkoumaných řešení. Algoritmus je díky tomu schopen úniku z lokálního optima a dokáže prozkoumat celý stavový prostor, důsledkem čeho je nalezení globálního optima. [30]

3.9 Rough set teorie

Matematický nástroj pro analýzu nedokonalých dat, založený na předpokladu, že každý objekty zkoumaného univerza je spojen s nějakou informací, představená v článku Pawlakem [31]. Objekty charakterizované stejnou informací jsou z hlediska dostupných informací nerozlišitelné. Tato vazba nerozlišitelnosti je matematickým základem rough set teorie. Libovolná sada nerozlišitelných objektů je nazývána základní sadou a tvoří základní prvky znalosti o univerzu. Spojení libovolných základních sad je pak nazýváno přesnou sadou. V případě, že spojení neobsahuje pouze základní sady, říkáme, že spojení je nepřesné (rough - odtud název celé teorie).

Každá nepřesná sada obsahuje krajní případy, prvky které nelze přesně klasifikovat za pomocí stávajících znalostí. S každou nepřesnou sadou je pak asociován pár přesných sad označovaných jako dolní a horní aproximace nepřesné sady. Dolní aproximace obsahuje objekty, které zcela jistě patří do nepřesné sady. Naopak horní aproximace obsahuje objekty, které mohou patřit do sady.

Analýza touto teorií začíná tabulkou dat, značenou jako rozhodovací, kde sloupce označujeme atributy a řádky hodnoty atributu. Atributy rozhodovací tabulky jsou rozděleny na dvě disjunktní podmnožiny, zvané podmiňovací a rozhodovací atributy. Každý řádek tabulky vyvolává rozhodovací pravidlo, které určí výsledek na základě splnění nějakých podmínek. Pokud rozhodovací pravidlo rozhodne pouze na základě podmínek, pak takové pravidlo nazýváme jisté. V opačném případě je pravidlo nejisté. Obecně jsou rozhodovací pravidla úzce spjata s aproximacemi. Jistá pravidla popisují aproximaci dolní aproximace vzhledem k pravidlům. Nejistá pravidla pak aproximují takzvanou hraniční oblast, která vniká rozdílem dolní a horní aproximace.

S každým rozhodovacím pravidlem jsou spojeny dvě podmíněné pravděpodobnosti. Tyto pravděpodobnosti se nazývají jistota a *pokrytí*. Koeficient jistoty vyjadřuje podmíněnou pravděpodobnost, že objekt patří do rozhodovací třídy určené spjatým pravidlem, jestliže splňuje jeho podmínky. Koeficient po-

krytí udává podmíněnou pravděpodobnost důvodů pro dané rozhodnutí. Tyto dva koeficienty navíc splňují Bayesův teorém, čímž nabízí jiný podhled na jeho interpretaci a nabízí novou metodu jak vyhodnocovat závěry z dat.

3.10 TSAR

TSAR algoritmus, celým názvem tabu vyhledávání pro redukci atributů, je snoubením tabu vyhledávání a metod z teorie rough set pro redukci atributů v datových sadách. Motivací pro tuto kombinaci byly slibné výsledky tabu vyhledávání při řešení kombinatorických vyhledávacích problémů, zvláště pak v případech s dlouhou pamětí. Nejedná se zdaleka o jediný pokus o zkombinování různých přístup s rough set teorií. Zmínit mohu například tři algoritmy prezentované a zkoumané v práci dvojice Jensen a Shen [32]. Ti v ní představili metody GenRSAR, AntRSAR a SimRSAR. GenRSAR je algoritmus založený na bázi genomu a fitness funkce, která bere v potaz velikost získané podmnožiny a její vhodnost. AntRSAR je založen na přístupu mravenčí kolonie, kde počet mravenců je nastaven na počet atributů sady a každý mravenec začíná na jiném atributu. Mravenci pak budují možná řešení dokud nedosáhnou redukce sady. SimRSAR pak využívá mechanismu simulovaného žíhání pro výběr atributů. Metoda se pokouší vylepšit řešení přidáním či odebráním tří atributů ke stávajícímu řešení.

Algoritmus TSAR, částečně popsaný výše, byl navržen v článku [33] dvojicí Hedar a Wang. Část algoritmu založená na tabu vyhledávání má základ ve dvou konceptech. Vyhýbání se návratu do nedávno navštívených řešení a přijímáni tahů zhoršujících aktuální řešení pro únik z lokálního extrému. Menší část historie vyhledávání je uchovávána pro inteligentnější chování celého algoritmu. Přesněji se jedná o doposud nejlepší nalezené redukce a frekvence výběru jednotlivých atributů pro diverzifikační a intenzifikační schémata.
TSAR užívá tři diverzifikační a intenzifikační schémata, generátor rozdílných řešení, setřásávání nejlepšího řešení pro snížení kardinality a inspiraci elitními redukcemi.

Výhodou TSAR je jeho poměrně nízká paměťová náročnost, díky reprezentaci řečení pomocí binárních řetězců, kdy přítomnost atributu v řešení je reprezentována hodnotou 1. V opačném případě je hodnota na dané pozici rovna 0.

3.10.1 Schémata TSAR

TSAR algoritmus používá celkem 4 strategie pro redukci atributů:

1. Vyhledávání sousedů

   Cílem této strategie je prozkoumat sousední řešení k aktuálnímu a vyhnutí se již prozkoumanému řešení. Tato řešení bývají často reprezentována jako binární vektory, kde hodnota jednotlivých složek udává

přítomnost atributu v možném řešení. Možnou metodou jak tato řešení generovat je několikabodový mutační operátor, kde hodnota na mutované pozici je náhodně generována.

Extrémním případem možného řešení je vektor, kde se na všech pozicích vyskytují pouze jedničky nebo nuly.

2. Diverzifikace

   Cílem diverzifikace je prozkoumání širšího spektra daného prostoru. Diverzní řešení lze vytvořit použitím atributů, které nebyly použity při generování možných sousedních řešení. Atributy jsou vybírány s pravděpodobností neúměrnou k pravděpodobnosti, že se objeví během generování možného sousedního řešení.

3. Setřásávání

   Strategie setřásávání se zaměřuje na optimalizaci stávajícího nejlepšího řešení. Procedura začíná postupným odstraňováním atributů po jednom, aniž by se snížila hodnota závislostní funkce. Jinak řečeno, úprava dosud nejlepšího řešení je možná pouze pokud je hodnota závislostní funkce zachována nebo zvýšena odstraněním atributu v tomto řešení.

4. Elitní redukce

   Záměrem elitní redukce je nalezení minimální sady atributů. Tato strategie velmi závisí na strategii setřásávání. Lze říci, že tato minimální sada je složená z průniku seznamu vektorů, vzniklých optimalizací nejlepšího nalezeného řešení.

TSAR začíná s počátečním řešením x^0 a pomocí omezení pro tabu vyhledávání vygeneruje zkušební řešení sousedící s aktuálním. Zkušební řešení jsou generována do doby než další řešení nepřinese žádné zlepšení skrze I_{div} po sobě jdoucích iteracích. Následně TSAR spustí vyhledávací proces z diverzního řešení x^{div} získaného pomocí diverzifikačního schématu. Pokud počet po sobě jdoucích iterací bez zlepšení překročí I_{shak}, TSAR vyvolá setřasávací schéma aby se pokusil vylepšit dosud nejlepší nalezené řešení x^{best}. Vyhledávání pak pokračuje z řešení x^{best}. To může být ukončeno pokud dojde k překročení maximálního počtu iterací I_{max} nebo počet po sobě jdoucích iterací bez zlepšení překročí hodnotu I_{imp}. Nakonec je aplikováno schéma elitní redukce jako poslední mechanismus diverzifikace a intenzifikace.

Autoři zmiňovaného článku tento algoritmus testovali nad 13 známými datovými sadami pro prokázání efektivity jejich práce. Z jejich experimentů vyplývá, že TSAR vykazuje dobré výsledky za snížené výpočetní ceny. Jako největší slabinou se pak jeví velké množství nastavitelných parametrů, které je třeba správně určit pro získání optimálních výsledků.

3.11 RSAR

Rough Set Atribute Reduction vychází obdobně jako TSAR, z 3.10, z teorie nepřesných sad, kde jsou užívány pro odstranění redundantních podmíněných atributů z datové sedy obsahující diskrétní hodnoty a přitom zachovat informační hodnotu [34].

Důležitým faktorem v odstraňování atributů je závislost mezi jednotlivými atributy. Pokud by změna v závislosti odstraněním atributu byla velká, ukazuje to na jeho významnost a tím pádem menší pravděpodobnost odstranění. Tento stupeň závislosti lze obecně vyjádřit rovnicí

$$k = \gamma_P(Q) = \frac{|POS_P(Q)|}{|U|} \quad , (0 \leq k \leq 1).$$

Množina $POS_P(Q)$ reprezentuje všechny objekty z množiny U, které lze namapovat na třídy množiny $U \setminus Q$ s pomocí znalostí z atributů P. Pro množinu P platí $P \subseteq A$, kde A je sloučením množin podmíněných a rozhodovacích atributů.

Samotná redukce atributů funguje na principu porovnávání rovností závislostí sad atributů. Cílem redukce je odebírání atributů, tak aby výsledná datová sada měla stejnou kvalitu klasifikace jako originální sada. Sada všech redukcí atributů je značena $R = \{X : X \subseteq C, \gamma_X(D) = \gamma_C(D)\}$, kde C je množina podmíněných atributů. Průnikem všech těchto množin z R se pojmenovává *core* a obsahuje prvky, které nelze ze sady odstranit aniž by došlo k porušení informační hodnoty. V RSAR je vyhledávaná taková redukce, aby její kardinalita byla minimální. Základní metodou jak tohoto dosáhnout je spočítat závislosti pro všechny možné podmnožiny C. Jakákoliv podmnožina X s $\gamma_X(D) = 1$ je redukcí a nejmenší taková podmnožina pak je minimální redukcí. Tato metoda je neefektivní pro velké datové sady.

3.12 Autoenkodéry

Autoenkodéry jsou druhem neuronové sítě, která je trénována za cílem kopírovat vstup na výstup. Tuto jednoduchou operaci stěžuje vnitřní konstrukce sítě, která zamezuje perfektnímu kopírování vstupu. To nutí neuronovou síť identifikovat a vybrat pouze složky vstupu, které jsou na sobě nezávislé. Složky, které jsou na sobě závislé, pak sloučí do složek nových, které reprezentují jejich závislost a hodnoty. Tím dochází k redukci dimensionality vstupních dat na výstupu.

Samotný autoenkóder se skládá z kódovací a dekódovací funkce. Kódovací funkce má za cíl reprezentovat vstupní data a jejich kombinace, čímž může docházet k redukci dimensionality. Dekódovací funkce následně tato data rekonstruuje na výstup.

Nejčastěji jsou autoenkodéry srovnávány s metodou PCA, která též slouží ke snížení dimenze dat. Díky vlastnosti neurálních sítí se učit nelineární vztahy

mezi daty, je možné vytvořit silnější nelineární zobecnění PCA. PCA se snaží odhalit nadrovinu nižší dimenze, než jaká popisuje původní data. Autoenkodéry se místo toho získat nelineární manifold, neboli spojitý neprotínající se plochu.

Identifikujeme několik variant autoenkodérů na základě způsobu modifikace kódovací funkce nebo vrstvy.[35]

- Nekompletní autoenkodéry

  Autoenkodéry mající nižší dimenzi kódu, než je dimenze vstupních dat. To nutí síť k zachycení nejvýznamnějších vlastností trénovacích dat. Celý proces učení lze snadno popsat jako minimalizace ztrátové funkce. Ztrátová funkce je funkce penalizujíce výstup sítě za rozdílnost od vstupu. Pokud je dekodér lineární a ztrátová funkce je střední hodnota druhé mocniny chyby, tak lze autoenkodér přirovnat k PCA.

  Důležité je u tohoto typu autoenkodéru zajistit, že kodér a dekodér mají dostatečnou malou kapacitu, aby nedošlo ke zkopírování vstupu na výstup. Tím by nedošlo k získání jakékoliv užitečné informace ze sítě.

- Regularizované autoenkodéry

  Řeší problém nekompletních autoenkodérů v případech, kdy je kapacita obou funkcí sítě dostatečná pro zkopírování vstupu. Místo omezování hloubky a dimenze kódu, využívají regularizované autoenkodéry ztrátovou funkci, která podporuje model k vlastnění dalších vlastností kromě schopnosti kopírovat vstup. Mezi tyto vlastnosti může například patřit řídkost reprezentace, podíl robustnosti oproti šumu či chybějícím vstupům.

- Rozptýlené autoenkodéry

  Rozptýlené autoenkodéry mají oproti klasickému upravenou trénovací podmínku na kódovací vrstvě. Ta zahrnuje penalizace za rozptýlenost, která se přičítá ke ztrátové funkci. Penalizace rozptylu může být v tomto případě chápána jako pojem přidaný k síti jejímž hlavním cílem je zkopírovat vstup na výstup bez dohledu a pokud možno, také provést nějaké akce pod dohledem.

  Většinou je tento typ autoenkodéru využíván pro naučení vlastností pro jinou úlohu, například pro klasifikaci. Pokud byl autoenkodér regularizován, pak musí reagovat na jedinečné statistické vlastnosti datové sady, na které byl trénován. Tímto způsobem lze získat model, který se navíc naučil užitečné vlastnosti během trénování.

- Odšumovací autoenkdéry

Na rozdíl od předchozí varianty, odšumovací varianta nepoužívá penalizační funkci, ale přímo mění způsob výpočtu ztrátové funkce. Standardní způsob výpočtu ztrátové funkce podporuje skládání vrstev, tak aby vznikla identitní funkce pokud je pro ní dostatečná kapacita. Odšumovací autoenkodér se místo toho snaží minimalizovat ztrátovou funkci mezi původním vstupem a výstupem modelu, který na vstup dostal poškozený původní vstup nějakým šumem. Tím pádem musí model odstranit poškození namísto prostého kopírování. To nutí obě vrstvy, aby se naučili strukturu data a potenciálně zjistili zajímavé vlastnosti datové sady.

- Stahovací autoenkodér

  Další strategie jako regularizovat autoenkodér, založený na penalizační funkci rozptýleného autoenkodéru. Rozdíl je zde ve formě penalizační funkce, která nutí model naučit se funkci, která se velmi málo mění, při změně vstupu. Tato penalizace je používána pouze na trénovací data, kde nutí autoenkodér, aby se naučil vlastnosti zachycující informace o rozdělení trénovací sady.

3.13 Shrnutí

V této kapitole jsem vysvětlil všechny důležité pojmy, které se mohou vyskytovat v příštích kapitolách, jako jsou strojové učení a jeho druhy, metrika hodnocení pomocí F skóre, metoda Particle Swarm Optimization, algoritmus K-means či metoda TSAR, založená na Rough set teorii.

Kapitola 4

Existující řešení

Tato kapitola si klade za cíl prozkoumat již existující akademická řešení problému optimalizace datových sad a redukce jejich dimenze.Problém optimalizace datových sad není nový, ale vzhledem k jeho relativnímu mládí, není ještě příliš prozkoumaný. Většina prací a materiálu zkoumaných pro účely rešerše se zabývá redukcí příznaku (sloupců) v datových sadách místo, redukce počtu záznamů (řádků). Bohužel tento přístup není v našem případě vhodný, jelikož dodané i používané datové sady jsou, co se týče počtu příznaků, optimalizované.

To ovšem neznamená, že použité algoritmy a postupy jsou nepoužitelné pro náš problém. Vhodnou úpravou lze docílit požadovaných výsledků. Na závěr kapitoly také připojím další osobní myšlenky k probíraným pracem a jejich využití.

4.1 Particle Swarm Optimization s Quick Reduct

V článku [36] se autoři zabývají redukcí příznaků nad datovou sadou pomocí metody Quick Reduct [37], založené na teorii Rough set pro snížení a identifikaci důležitých příznaků testované sady. Pro samotný proces redukce, pak využívají metody Particle swarm optimization v kombinaci s Quick Reduct. Důležitost příznaku určuje počet objektů v pozitivní oblasti příznaku. Pozitivní oblast lze chápat jako okolí nějakého příznaku. Vždy je vybírán příznak s nejvyšší hodnotou fitness funkce. Kombinace příznaku jsou počítány dokud hodnota fitness funkce nedosáhne 1. Následně tyto jedničky určují vybrané příznaky v nejlepším nalezeném globálním řešení.

Provedená měření ukazují, že použitá metoda kombinovaná s částečným učením s pomocí učitele (semi-supervised learning) produkuje optimální sadu příznaků pro zkoumanou datovou sadu.

4.2 Hyper-heuristika

Abdullah a spol. se ve svém článku [38] zaměřují spíše na způsob přístupu k optimalizaci než konkrétní metodě za pomoci hyper-heuristik. Jak je v článku vysvětleno, hyper-heuristika je učící algoritmus, který místo řešení problému, vybírá nebo konstruuje heuristiky pro řešení daného problému. Tím dosahují schopnosti řešit větší spektrum problémů, než heuristiky zaměřené přímo na specifickou problémovou doménu.

Jejich specifická heuristika buduje řešení iterativně s pomocí čtyř různých heuristik nižší úrovně. Heuristika použitá v iteraci je vybrána pomocí systému rulety s výběrem založeným na výkonu dané heuristiky v předchozích iteracích. Z naměřených výsledků je vidět, že tento přístup překonává na některých instancích problémů známé výkonné metody TSAR, RSAR a jejich varianty.

4.3 Particle swarm optimization s imunitou

Článek od skupiny Lin, Wu, Mao a Yu[39] se opět zaobírá aplikací Rough Set theory pro redukci počtu příznaků v datové sadě. Podobně jako Saraswathy a spol. [36] využívají techniky Particle Swarm Optimization, kde narozdíl od Saraswathy přidávají imunitu pro určité částice populace, čímž se snaží předejít konvergenci algoritmu do lokálního optima. Pravděpodobnost, že částice bude "vakcinována", neboli získá zmiňovanou imunitu je dána kombinací váhy atributu a hodnoty dvou proměnných, které jsou zvoleny dopředu.

Experimentální výsledky ukázali, že tento algoritmus je schopen efektivně odstranit redundantní atributy a atributy poskytující slabou informační hodnotu. Nikde ve článku bohužel není porovnání s dalšími používanými algoritmy, aby bylo možné posoudit zda je tento algoritmus opravdu efektivní.

4.4 Autoenkodéry

V článku [40] se autoři zabývají aplikací autoenkodérů pro snížení počtu příznaků nad známou dermatologickou datovou sadou. Toho dosahují nejdříve normalizací datové sady a následnou aplikací dvou sítí autoenkóderů s rozdílným počtem neuronů a prahovými hodnotami. Učící algoritmus zůstal pro obě sítě stejný v průběhu experimentu.

Provedené experimenty ukazují, že aplikace pouze jednoho autoenkóderu zlepší přesnost klasifikátoru ovšem ne příliš. Aplikace dvou autoenkóderů po sobě dále přesnost zvyšuje do takové míry, kdy je možné, i z grafu vytvořeného pomocí PCA, vidět jednotlivé shluky klasifikovaných příznaku s jasnými hranicemi mezi nimi. Proto se tento přístup jeví jako poměrně dobrý kandidát na redukci velikosti našich datových sad.

4.5 Density aware autoenkodéry

Zhou a spol. ve svém článku [41] popisují svůj návrh architektury shlukování grafů za pomoci autoenkodérů s nižší paměťovou složitostí než stávající metody. Vycházejí zde z problémů modelů vkládání sítí, které obecně nezohledňují lokální charakteristiky vrcholů a podgrafů, v nich zahrnutých. Jelikož řešení pomocí rozdělení na K disjunktních podgrafů by bylo příliš paměťově náročné, představují v tomto článku nový model DALSAE. Tento model obsahuje K lokálních autoenkodérů s individuálními parametry a strukturami, které souběžně trénují nad K podgrafy a optimalizují ztrátovou funkci. Na úrovni uzlů pak dochází k přidání jedinečného parametru hustoty, který ovládá aktivaci skrytých neuronů během učení a hustotu konečného vložení. Všechny tyto vlastnosti vedou ke zlepšení kvality predikce.

Dále ve článku prezentují nový algoritmus pro vzájemné vylepšování lokálních autoenkodérů na základě shlukování podle Studentova t-rozdělení. Díky tomu je celek schopný pomocí hyper-parametrů dynamicky upravovat jednotlivé autoenkodéry.

I přes svou zajímavost jsem nenašel aplikaci pro tento typ autoenkodéru v této práci. Celková struktura algoritmu je, dle mého názoru, příliš složitá a navíc pracuje s velkým množstvím parametrů.

4.6 Predikce podobnosti skupinou autoenkodérů

Hwang a spol. [42] se v jejich článku zaměřili na problematiku detekce a rekonstrukce podobností mezi datovými sadami a dokonce i v rámci datové sady. Celá idea vznikla pozorováním s jakou rychlostí se vytváří nové datové sady, díky úspěchu neurálních sítí a rozšířením nástrojů pro sdílení generování těchto sad. Výkon strojového učení je silně závislý na kvalitě trénovací sady a častou praktikou bývá aplikace datové sady na model, který byl předtím trénován s jinou, pravděpodobně podobnou, sadou. Pokud však sady nebyly podobné, tak nemohou ani výsledky takto naučeného modelu podávat správné výsledky.

Právě tento problém vedl autory k navržení modelu, který je schopný detekovat podobnosti mezi sadami pomocí množiny předem naučených autoenkodérů, kde každý z nich je specializovaný na rekonstrukci pouze části známých dat. Vstupem modelu jsou pak neznámá data, která se snaží autoenkodéry rekonstruovat pomocí znalosti trénovacích dat. Tato rekonstruovaná data jsou pak porovnána proti původnímu vstupu na podobnost. Hlavní myšlenkou je, čím větší je podobnost rekonstruovaných vzorků s neznámými vstupními daty, tím existuje větší šance, že neznámá datová sada je podobná trénovací sadě, vedoucí k přesnější rekonstrukci vstupu. Rozdíly mezi vstupem a rekonstruovaným výstupem je pak indikátorem podobnosti neznámé datové sady se sadou trénovací.

Model SimEx, jak jej autoři nazvali, má 3 hlavní scénáře použití: 1) určení podobnosti mezi obecnými datovými sadami, 2) určení podobnosti mezi třídami v rámci několika heterogenních datových sad a 3) určení podobnosti mezi třídami v rámci jedné datové sady. Jádro modelu je založeno na množině autoenkodérů, v článku značenou $\mathcal{A}(X)$, sestávající se z autoenkodérů naučených na vzorcích $x_i \in X$, kde X je datová sada. Samotná množina X je tvořena disjunktními množinami $X = X_0 \cup X_1 \cup \ldots \cup X_N$. Bez újmy na obecnosti se může jednat o sjednocení datových sad nebo klasifikačních tříd v rámci jedné datové sady. Metoda pak využívá celou tuto množinu autoenkodérů $\mathcal{A} = \{\mathcal{A}(X_1), \mathcal{A}(X_1), \ldots, \mathcal{A}(X_N)\}$. Každá autoenkodér $\mathcal{A}(X_i)$ je specializovaný na rekonstrukci specifikované části datového prostoru.

Na příkladech bylo následně ukázána síla této metody, která kromě dobrých výsledků, poskytuje i dle autorů až desetinásobné zrychlení oproti podobným modelům.

Tento model se zdá být dobrým kandidátem pro implementaci prototypu optimalizace datových sad a bude dál zkoumán.

4.7 Diskuze existujících řešení

Všechna zkoumaná řešení nabízejí zajímavý náhled na moderní přístupy pro redukci dimensionality datových sad. Bohužel jediná práce nabízející přímé srovnání s dalšími běžně používanými metodami je práce 4.2. Postupy popisované v této práci jsou zaměřené na redukci dimenze eliminací sloupců, což není přímo cílem této práce, kde se snažím spíše zredukovat počet řádků, a tyto postupy nejsou vhodné na tento typ redukce. Metody PSO v sekcích 4.1 a 4.3 pak těžce spoléhají na fitness funkci pro vyhodnocení kvality aktuální populace. To představuje problém, jelikož kvalitu záznamu zkoumané datové nelze snadno určit. Ohodnocení každého záznamu je postavené na kombinaci velkého množství parametrů, kde každý parametr má specifickou váhu. Proto by bylo složité navrhnout dostatečně dobrou fitness funkci, která by navíc byla použitelnou pouze pro tuto sadu.

Sekce 4.5 sice představuje zajímavé kombinování autoenkodérů aplikací na podgrafy nějakého zkoumaného grafu. Bohužel naším zkoumaným subjektem je datová sada, kterou nelze, dle mých znalostí, reprezentovat jako graf. Navíc je celková struktura metody velmi složitá a tudíž si nemyslím, že by bylo vhodné ji implementovat pouze na jednu instanci problému jako je případ této práce.

Nejlépe vypadají ideje a výsledky experimentů v sekci 4.4 a 4.6.V 4.4 autoři ukazují zlepšení přesnosti klasifikátoru pomocí zřetězení dvou autoenkodérů. Pro mne velmi zajímavou možností je i pak výsledná možnost výsledky zobrazit v grafu metodou PCA, jelikož je pro mne snazší interpretovat tímto způsobem výsledky. To platí hlavně pro tento případ, kdy autoři jasně ukazují, že jejich experimenty vedly na vytvoření velmi jasně definovaných komponent,

což by mohlo pomoci i v optimalizaci datových sad, u kterých by tím mohlo dojít ke zpřesnění výběru kandidátů na členy optimalizované sady.

Sekce 4.6 pak ukazuje metodu SimEx pro detekci podobností mezi datovými sadami, potažmo třídami v rámci jedné datové sady. Experimentální měření a z nich plynoucí výsledky, které autoři provedli ukazují velkou sílu jejich modelu. Celkový přístup k problému s kombinací dobrých experimentálních výsledků, činí tuto metodu dobrým kandidátem na implementaci a testování. Z toho důvodu se pokusím v dalších kapitolách tuto metodu naimplementovat pro zkoumanou datovou sadu a pomocí metody SimEx dojít k její optimalizaci.

Tato kapitola seznámila čtenáře s již existujícími akademickými pracemi na téma optimalizace datových sad a redukce jejich dimenze. Byť se jedná o téma velmi zkoumané, leží většina pozornosti výzkumu na redukci počtu příznaku a nikoliv na detekci redundantních, či podobných záznamů v datové sadě. Naštěstí některé metody ukazují zajímavé postupy, které by bylo možné po nějaké formě úpravy použít, jak jsem popsal v sekci 4.7 výše.

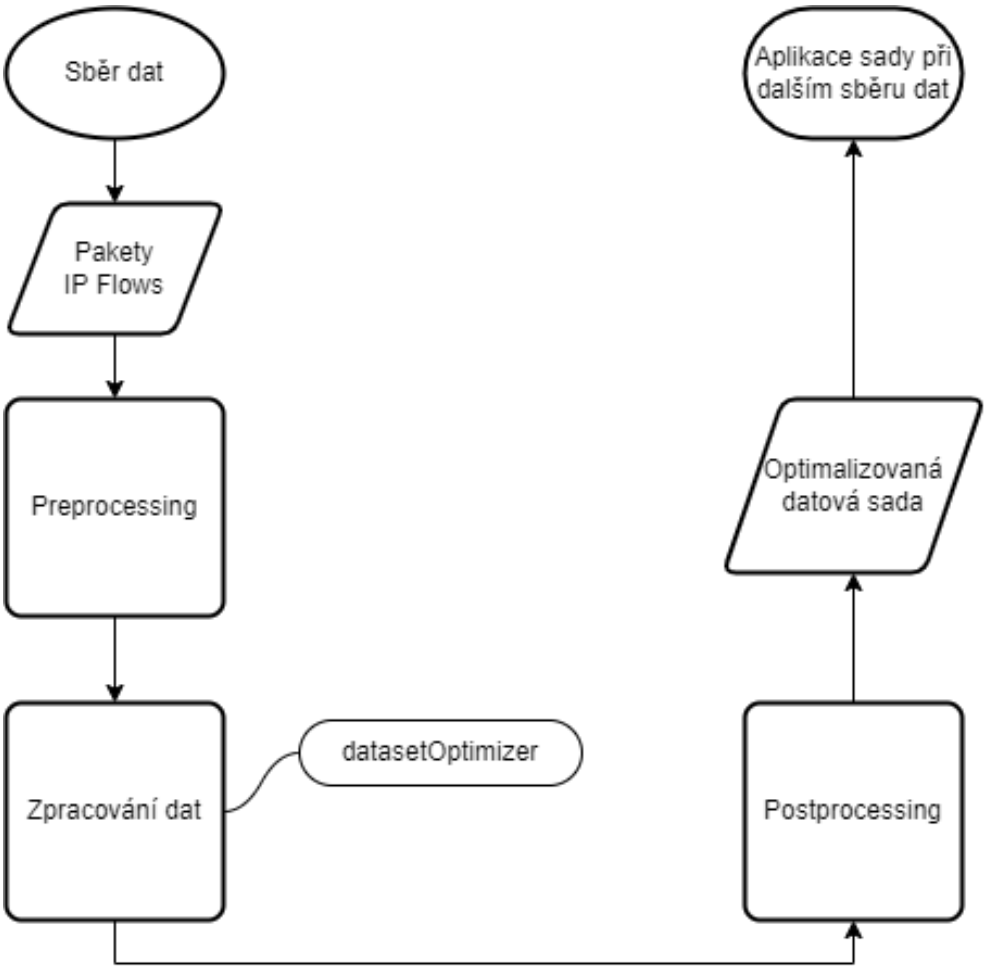
Kapitola 5

Návrh

Záměrem této kapitoly je představit návrh několik potenciálně použitelných metod pro snížení počtu záznamů v datové sadě s ohledem na minimalizaci ztráty hodnoty F_1 skóre dané sady. Hlavní cílem optimalizace pomocí navržených metod je redukce počtu řádků datové sady, s pokud co možno nejnižší ztrátou přesnosti. Dobrou vlastností pro metody by pak byla identifikace problémových řádků, které klasifikátory nepřesně určují a ty odstranit.

Jako programovací jazyk pro provedení experimentů a implementaci prototypu jsem zvolil jazyk Python. Hlavní motivací pro tuto volbu je vysoká popularita v komunitě zabývající se strojovým učením a běžné algoritmy jsou již implementovány formou knihoven a modulů, včetně dokumentace. Dalším důvodem je možnost integrace vytvořeného prototypu do vznikajícího frameworku Active Learning Framework (ALF). ALF je psán právě v jazyce Python a byl navržen na myšlence modulárnosti, proto bude možné vytvořený prototyp pro optimalizaci datových sad bez větších úprav integrovat do tohoto frameworku, kde může být dále testován a rozvíjen. Obrázek 5.1 zobrazuje, jak bude softwarový prototyp integrován do frameworku ALF.

Velká část Python knihoven a modulů je dále navržena pro práci s knihovnou Pandas, která poskytuje velmi dobré nástroje pro práci s datovými sadami jako je například DoH. Další klíčovou knihovnou, kterou budu používat je knihovna *scikit-learn*. Jedná se o knihovnu specializovanou právě na strojové učení.



Obrázek 5.1: Integrace prototypu do frameworku ALF

5.1 Náhodný výběr z celku

Jako první metodu navrhuji jednoduchý náhodný výběr určitého procenta záznamů z celé datové sady. V této metodě není žádná pokročilejší logika, pouze jednoduchý náhodný výběr pomocí funkce náhodného výběru nad datovým rámcem.

Jedná se velmi primitivní metodu a pravděpodobně poslouží pouze jako porovnání s ostatními metodami, aby se dalo nahlédnout jakých výsledků budou dosahovat složitější metody. Navíc by metoda mohla poodhalit nějakou z vlastností datové sady, o tom ale více až v kapitole 7, sekci 7.3.

5.2 Náhodný výběr z komponent

Tato metoda má za cíl rozšířit náhodný výběr z předchozí metody přidáním vyvažovacího pravidla. Tím získám datovou sadu, která bude vyvážená. Metoda vybere komponenty a v rámci nich bude proveden náhodný výběr obdobně jako v předchozí metodě 5.1.

Pro dosažení vyváženosti bude velikost výsledných komponent řízena dle původní velikosti menší z nich. V případě aplikace na jiné datové sady než testované, bude nutné určit minimální velikost komponent experimentálně, protože existuje možnost, že jedna z komponent bude mít pouze jeden záznam. Pokud by se pak nestanovila minimální velikost, může v krajním případě nastat situace, kdy bude jedna z komponent vynechána z výběru a výsledná sada by pak nedosahovala dobrých výsledků.

5.3 K means

Za pomoci algoritmu K means získám středy shluků datové sady. Následně vypočtu vzdálenosti jednotlivých bodů shluku od jejich přiřazeného středu a i hodnotu všech těchto vzdáleností. Všechny body jejichž vypočtená vzdálenost je menší než průměrná vzdálenost jsou následně odstraněny.

Pro K means lze použít několik přístupů pro výpočet vzdálenosti bodů od středu. Zaměřil jsem se pouze na klasickou střední Euklidovskou vzdálenost od středu.

Myšlenkou tohoto postupu je vytvořit jakési oblasti definované nejvzdálenějšími body od středu. Body blízko středu této oblasti nepřináší velkou informační hodnotu a tím pádem je odstraním. Nové záznamy v sadě s největší pravděpodobností budou spadat právě do prostoru, který byl algoritmem vyčištěn, nemělo by tedy tímto způsobem dojít ke zhoršení přesnosti klasifikace.

5.4 Autoenkodér

Autoenkodéry pracují na principu rekonstrukce vstupu s omezenou pamětí. Cílem kódovací vrstvy je tedy odhalit informace kritické pro dobrou rekonstrukci a vhodně je zakódovat. Aplikací na datovou sadu se pokusím zredukovat počet řádků sady. Následně otestuji přesnost jak zakódované sady, tak i sady dekódované. Na základě předchozí rešerše se pokusím naimplementovat vlastní jednoduchý autoenkodér pro optimalizaci datové sady a následně implementuji model SimEx ze sekce 4.6 pro detekci podobnosti tříd v rámci jedné datové sady a pokusím se ho upravit, aby díky němu algoritmus produkoval optimální datovou sadu.

5.4.1 Jednoduchý autoenkodér

Obecně používáme autoenkodéry pro komprimaci obrazových dat. Snížení počtu příznaků, neboli sloupců, datové sady pak můžeme vnímat jako jakýsi krajní případ pro použití autoenkodérů. Knihovna Keras implementující autoenkodéry pak počítá spíše s použitím na obrázky, bude tedy otázkou jakých výsledků bude takto implementovaný autoenkodér dosahovat na kombinaci numerických dat a znaků.

5.4.2 SimEx autoenkodéry

Autoenkodéry založené na principu SimEx popsané v sekci 4.6 lze pak vnímat jako nadstavbu nad klasickými autoenkodéry, které dosahují lepších výsledků specializací na jednotlivé třídy problému. Vzhledem k výsledkům předloženým v článku [42] předpokládám, že výsledky tohoto postupu budou dosahovat vyšší přesnosti než jednoduchý autoenkodér.

5.4.3 Shrnutí

V kapitole jsem představil několik možných návrhů jak optimalizovat datovou sadu DoH analyzovanou v sekci 7.1. Kromě samotného představení metod, jsem též vyjádřil nějaké osobní domněnky a očekávání a obavy pro jednotlivé metody.

Kapitola 6

Implementace

Tato kapitola se věnuje implementačním specifikům celé práce a detailům implementace jednotlivých metod navržených v kapitole 5.

V rámci popisu implementace jednotlivých metod bude i popis jednotlivých variant, které následně projdou testováním v kapitole 7.

Vzhledem k faktu, že výsledek práce má být zakomponován do nově vznikajícího frameworku Active Learning (ALF), je logicky vhodné aby navržené metody byly implementovány ve stejném programovacím jazyce. ALF je jako celek psaný v jazyce Python, proto pro implementaci metod použiji stejný jazyk.

Všechny optimalizační metody budou pak implementovány v rámci samostatného modulu **DatasetOptimizer**, aby jej bylo možné snadno modifikovat a integrovat v rámci budoucího vývoje. V rámci modulu se pak budu spoléhat na knihovny Pandas, pro operace nad datovou sadou pomocí třídy DataFrame, scikit-learn, implementující řadu metod strojového učení a pokytují funkce pro výpočet různých metrik, a Keras, která poskytuje robustní implementaci autoenkodérů.

Modul implementuje jednu třídu **Optimizer**, reprezentující sadu navržených optimalizačních metod. Při vytáření instance této třídy je povinný pouze jeden parametr a tím je Pandas DataFrame obsahující datovou sadu, která má být optimalizována.

6.1 Náhodný výběr z celku

Náhodný výběr je implementován jako metoda *optimizeRandom* s jedním parametrem α určujícím kolik procent řádků z datové sady má být vybráno pro novou datovou sadu.

Samotný náhodný výběr je zajištěn pomocí funkce *sample* z třídy DataFrame modulu Pandas. Celou metodu lze pak snadno naprogramovat pouze na jeden řádek, viz. obrázek 6.1.

Obrázek 6.1 Náhodný výběr (pseudokód)

```
1: function optimizeRandom(Alpha)
2:     return dataFrame.sample(frac = Alpha)
3: end function
```

6.2 Náhodný výběr z komponent

Tato metoda, pojmenovaná *optimizeRandomComponent* přijímá dva vstupy, α určující kolik procent řádků zvolit z každé komponenty a *Column_name*, který specifikuje název sloupce, ve kterém se nachází odhodnocení komponent podle něhož bude proveden výběr. Po spuštění metody dojde k výběru jednotlivých komponent a určení jejích velikostí. Následně jsou velikosti komponent porovnány a nejmenší z nich je určena pro výpočet velikosti výběru ze všech komponent. Vzhledem k testované sadě, se nabízí možnost implementovat výběr komponent specifikací dvou komponent, ale tento přístup není přenositelný na rozdílné datové sady. Proto jsem se nakonec rozhodl pro implementaci iterováním přes jednotlivé komponenty.

Obrázek 6.2 poskytuje náhled na implementaci v rámci třídy Optimizer.

Obrázek 6.2 Náhodný výběr z komponent

```
1: function optimizeRandomComponent(Alpha, Column_name)
2:     classifications = dataFrame[Column_name].unique()
3:     n_classifications = classifications.shape[0]
4:     collection =
5:     min_size = ∞
6:     for classification in classifications do
7:         collection[classification] = dataFrame[dataFrame[Column_name] ==
       classification]
8:         if collection[classification].shape[0] < min_size then
9:             min_size = collection[classification].shape[0]
10:        end if
11:    end for
12:    selection_size = ceil(min_size ∗ Alpha)
13:    df_result = DataFrame()
14:    for classification in classifications do
15:        df_result = df_result+collection[classification].sample(selection_size)
16:    end for
17:    return df_result
18: end function
```

Takto implementovaný náhodný výběr lze aplikovat i na jinou datovou sadu, než nad jakou je testován, díky identifikaci různých hodnot v zadaném sloupci.

6.3 K Means

Metodu K Means jsem rozdělil do dvou funkcí. První funkce *optimizeKMeans* přijímá jeden parametr *cluster_count*, který udává počet shluků, které K Means vytváří z datové sady zadané při tvorbě třídy Optimizer. Po vytvoření shluků dojde k výpočtu Euklidovských vzdáleností jednotlivých bodů od svých středů a výpočtu průměru těchto vzdáleností. Návratovou hodnotou funkce je Pandas DataFrame se všemi prvky původní datové sady, jejichž spočtená vzdálenost je větší než průměrná vzdálenost od jejich středu.

Druhá varianta metody je založená na kombinaci PCA a K Means. Metoda *optimizeKMeansPCA* má dva vstupy, prvním je počet komponent PCA *pca_components* a druhým je počet komponent v K Means, *kmeans_components*. Metoda spočítá PCA skóre pro datovou sadu s počtem komponent určeným parametrem *pca_components*. Toto skóre je použito ve volání metody K Means s počtem shluků *kmeans_components*. Následně vznikne nový Data Frame složený z původní datové sady, PCA skóre každého záznamu, jeho přiřazené PCA komponenty a K Means přiřazení. Výběr výsledných řádků pak probíhá stejně jako v předchozí metodě a opět je vrácen nový DataFrame.

6.4 Autoenkodér

Pro implementaci autoenkodérů jsem zvolil populární knihovnu Keras ze sbírky knihoven Tensorflow. Tensorflow je sbírka knihoven specializovaná právě na neuronové sítě a má poměrně dobrou dokumentaci.

Kvůli vlastní nezkušenosti z autoenkodéry jsem se významně inspiroval při implementaci samostatného autoenkodéru z webových stránek Jasona Brownlee, který je specialistou na strojové učení a poskytuje řadu užitečných návodů. Konkrétně jsem se inspiroval návodem pro regresní[43] a klasifikační autoenkodéry [44]. Stejně jako v návodu používá tento autoenkodér tzv. husté vrstvy nasledované normalizačními vrstvami *BatchNormalization* a aktivačními vrstvami *LeakyReLu*.

Jednoduchý autoenkodér je implementován pomocí funkce se čtyřmi parametry *optimizeAE*. Parametr *Corr_limit* značí dolní hranici hodnoty korelačního koeficientu pro výběr řádku a parametr *Column_name* určuje název sloupce s klasifikací záznamu, *n_dimensions* značí počet sloupců a *n_epochs* určuje počet trénovacích iterací. Funkce jako první krok provede normalizaci a škálovaní datové sady. Následně provede kompilaci a učení autoenkodérů, ze které získá rekonstrukci původní sady. Zrekonstruovanou sadu porovná pomocí korelačních koeficientů s původní naškálovanou sadou a vybere z původní sady takové záznamy, které mají dostatečně vysoký korelační koeficient a jejichž klasifikace v rámci autoenkodérů odpovídá skutečné anotaci.

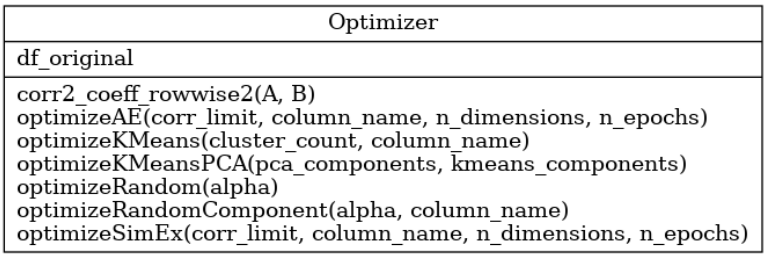
Takto získaná sada by měla produkovat datovou sadu, která obsahuje jen nejdůležitější řádky z původní sady, které by měly být zároveň pro klasifikátor

jednoznačné na klasifikaci a nemělo by docházet ke špatnému označení. Samotná funkce nijak neřeší duplicitu dat. Řeší ovšem vyváženost dat pomocí přizpůsobení větší kategorie menší odebráním nadbytečných záznamů.

Autoenkodér založený na metodě SimEx, jehož funkci jsem pojmenoval *optimizeSimEx*, popsané v 4.6, jsem se rozhodl implementovat jako zdvojení předchozího autoenkodéru, tedy i parametry jsou stejné, kde každý z nich je trénován pouze nad odpovídající klasifikací datové sady, tedy jeden na řádky anotovanými, že obsahují DoH provoz a druhý pak na řádky neanotované jako DoH provoz. Postup výběru pak je stejný jako v případě předchozím, jen s tím rozdílem, že autoenkodér nemůže vybrat řádek z původní sady pokud anotace řádku neodpovídá klasifikaci, kterou má daný autoenkodér ve své kompetenci. Tím by mělo dojít k předcházení situací, kdy by oba autoenkodéry vybrali stejný řádek do výsledné sady a vznikla by tak duplicita. Prevence špatného označení je pak řešena stejně jako v předchozí funkci. Vyváženost dat je pak implementovaná stejně jako v předchozí variantě.

6.5 Shrnutí

Záměrem této kapitoly bylo seznámení se specifiky implementace optimalizačních metod a jejich různých variant testovaných v kapitole 7. Nastíněna byla implementace jak metod náhodného výběru, tak i metod optimalizace pomocí K Means, K Means s PCA, a dvěma variantami autoenkodérů. Na obrázku 6.3 je zobrazena struktura třídy *Optimizer* modulu *datasetOptimizer*.



Obrázek 6.3: Diagram třídy Optimizer

Kapitola 7

Testování

Testování provádím nad datovou sadou reprezentující provoz DNS over HTTPS (DoH). Sada vznikla kombinací dvou podmnožin. První podmnožinou je zachycený reálný provoz na infrastruktuře CESNET. Druhou je uměle vygenerovaný provoz. Veškeré testování probíhá též na infrastruktuře CESNET, konkrétně na serveru netmon, laboratoře pro monitorování síťového provozu.

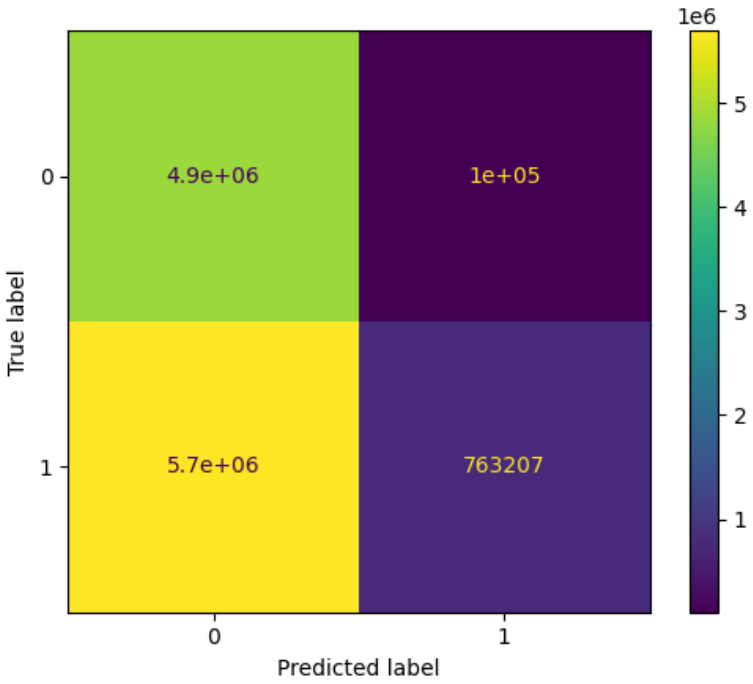
7.1 Analýza datové sady DoH

Před samotným testováním je důležité popsat samotnou testovanou sadu. Obě zmíněné podmnožiny sady byly již dříve anotovány a známe tedy jejich ohodnocení, které považuji za korektní, vzhledem k tomu, že sada s tímto ohodnocením je již nějakou dobu používána i v jiných experimentech a pracích než této.

Celá datová sada má celkem 76 149 600 řádků a 25 sloupců. Sloupce každého řádku obsahují anotaci řádku *is_doh*, počet bytů, počet paketů, součty a podíly paketů a bytů, celkový čas trvání flow, hodnoty zpoždění pro flow i autokorelaci. Nevyjmenovávám zde všechny sloupce, protože ne všechny jsou důležité při následných experimentech, resp. nejsou zadávány jako parametr nějakého z následujících algoritmů. Sloupec bude vysvětlen v případě, že jako parametr je použit. Jinak je pro nás nejvíce podstatný sloupec *is_doh*. Ten nabývá pouze hodnot $True$ a $False$.

V matici konfůze 7.1 lze nahlédnout přesnost přesnost klasifikátoru, při aplikaci na celou s trénováním provedeným na vygenerované podmnožině. Takto trénovaný model má poměrně nízkou přesnost s velkým množstvím *true negative* a *false positive* klasifikací, neboli klasifikátor predikuje, že záznam je DoH, ale ve skutečnosti tomu tak není, nebo naopak.

Aktuální hodnota F_1 skóre klasifikátoru nad touto datovou sadou se zvolenou konfigurací je 0.84. Dá se říci, že je to přesnost poměrně vysoká, byť oproti článku [45] je výrazně nižší, což přisuzuji nižším hodnotám parametrů klasifikátoru. Mým hlavním cílem je zredukovat velikost celé datové sady, tak



Obrázek 7.1: Matice konfůze pro původní datovou sadu

aby šla použít jako trénovací sada pro další iteraci. Po natrénování si bude klasifikátor sám přidávat sebou klasifikované záznamy, které do něj budou odeslané, Po nějaké určité době, až se velikost datové sady opět stane příliš velikou, dojde opět k použití vhodného algoritmu z dále testovaných pro opětovné zmenšení sady.

Prvním krokem před aplikací funkcí, které by měli datovou sadu optimalizovat je redukce sady pomocí deduplikace. Odstranění duplicitních řádků provedu funkcí *drop_duplicates(keep=first)*. Jedná se o implementačně jednoduchou operaci, která datovou sadu zmenšila o 2 328 450 řádků. Jedná se sice o redukci 3% z oproti celku, ale přesto jde zmenšení datové sady.

7.2 Metodika testování

Jak jsem výše zmínil, datová sada DoH je složená ze dvou podmnožin, jedna je zachycený reálný provoz, druhá podmnožina je strojově vygenerována. Pro prvotní testování jsem používal pouze menší podmnožinu celkové sady pro urychlení experimentů, čímž došlo k eliminaci idejí, které se myšlenkou zdáli rozumné, ale nevykazovali žádné výsledky.

Další experimenty jsem již nad celou datovou sadou provedl, pokud předběžné výsledky vykazovali zajímavé hodnoty. V další kapitole jsou k nahlédnutí

výsledky průběžných testů a v závěru kapitoly se nachází srovnání výsledků všech navržených metod s parametry vykazujícími nejlepší výsledky.

Jako metriku pro vyhodnocení kvality metody používám F_1 skóre popsané v sekci 2.7. Trénování a výpočet výsledného skóre zajišťuje klasifikátor AdaBoostClassifier z knihovny AdaBoost pro programovací jazyk Python. Hodnoty parametrů vychází z předchozích experimentů nad sadou DoH, dle článku [45], na rozdíl od článku jsou hodnoty parametrů poníženy, aby experimenty netrvali neúměrně dlouhou dobu.

Základním klasifikátorem v použitém AdaBoost klasfikátoru je rozhodovací strom s maxilmální hloubkou 6 a kritériem výběru je zvolena entropie. Jedná se lehkou variaci parametrů ze zmiňovaného článku, za účelem snížení využití paměti a zkrácení výpočetního času.

7.3 Náhodný výběr

Pro metodu výběru ze sekce 5.1 volím pouze jeden parametr α značící velikost výsledné datové sady vůči originální sadě.

Testování jsem začal s $\alpha = 80\%$ a postupně hodnotu snižoval až na $\alpha = 40\%$. Již při opakování prvního testu vyšla najevo nevýhoda výběru pomocí náhodné selekce, kdy při stejných parametrech získáváme rozdílné výsledky.

Náhodný výběr vykazuje poměrně zajímavé slabé výsledky, které šlo předpokládat už z nátury experimentu a divoké hodnoty F_1 skóre, pohybující se v rozmezí $0.15 - 0.84$, nejsou jedinou vadou. Bohužel tato metoda nezaručí pokrytí datové sady. Můžou proto nastat situaci, kdy náhodný výběr zvolí záznamy pouze z jedné třídy a tím pádem vznikne datová sada, která je schopna rozpoznat pouze tuto třídu a nezná žádnou další. Jak jsem již zmiňoval, tato metoda je zde pouze jako ukázka, ze které lze nahlédnout, že datová sada lze velmi výrazně zredukovat s malou ztrátou přesnosti.

Velikost datové sady po redukci touto metodou není třeba grafově zobrazovat neboť ji lze zcela jednoduše odvodit z hodnoty α.

7.4 Náhodný výběr z komponent

U náhodného výběru z komponent ze sekce 5.2 je v případě DoH opět jediným parametrem α obdobně jako v případě náhodného výběru v 7.3.

Testování probíhalo pro hodnoty $\alpha = \{0.2, 2, 40, 60\}\%$. Dosažené skóre však zůstalo konstantní pro všechny testované hodnoty. Hodnota dosaženého skóre je 0.98.

Výběr dosahuje výrazně lepších výsledků než předchozí varianta 7.3. Navíc díky implementaci vyvažování klasifikačních tříd došlo k zaručení úplnosti, kterou předchozí metoda neposkytovala. Díky tomu se předešlo krajnímu případu předchozí metody, kdy mohlo dojít k výběru záznamů pouze z jedné třídy. Avšak jde stále pouze o náhodný výběr, lze zde tedy najít záznamy, které

jsou velmi podobné a tím pádem se stávají redundantní a šlo by je odstranit. Tento fakt vychází z pozorování, že snižováním hodnoty α nedošlo ke zhoršení přesnosti. S tímto problémem by mohla pomoci další metoda založená na K means.

Velikost datové sady po redukci náhodným výběrem z komponent je 99.8% a ukazuje, že je možné dosáhnout velké míry redukce velikosti datové sady bez ztráty, či s minimální ztrátou přesnosti. Z toho lze zároveň usuzovat velkou míru redundance v rámci datové sady, kteroužto vlastnost by bylo vhodné prozkoumat, zda lze pomocí ní sadu nějakým způsobem redukovat.

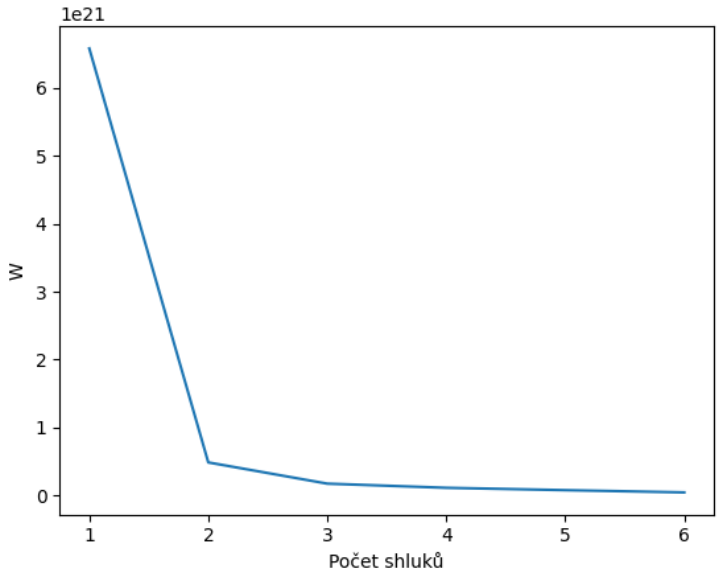
7.5 K means

Nastavení metody K means z 5.3 vyžadovalo již větší přípravu. Prvním a pravděpodobně nejdůležitějším parametrem, který bylo nutné zvolit je právě K. Ze znalosti datové sady jsem hodnotu odhadl na $K = 2$, protože klasifikace každého záznamu v datové sadě může nabýt pouze hodnot $True$ a $False$ v případě, že záznam je klasifikován jako DoH nebo není. Pro ověření odhadu jsem použil již o něco robustnější metodu Elbow, taktéž popsanou v 3.3. Graf 7.2 vykresluje hodnoty W pro zvyšující se K. Lze zde nahlédnout, že hodnota W přestává výrazně klesat od $K = 2$. Tím se potvrdil můj odhad, že $K = 2$. Přesnější a sofistikovanější metody nemá dle mého názor smysl aplikovat nad touto datovou sadou, protože není dostatečně složitá a navíc je očividné, že máme pouze dvě možné klasifikace záznamu. Respektive existuje varianta použití jiných komponent pro určení shluku než je pouze sloupec klasifikace. Tato varianta je blíže prozkoumána variantou K Means s PCA.

Provedené experimenty dosahují redukce počtu řádku z původních 11 421 978 záznamů na pouhých 221 822 záznamů v nové sadě. Funkce tedy dosáhla redukce 98.06% proti původní velikosti, při zachování pokrytí obou anotačních příznaků. Hodnota F_1 skóre této metody je 0.93. Pozorujeme tedy zlepšení oproti původní sadě.

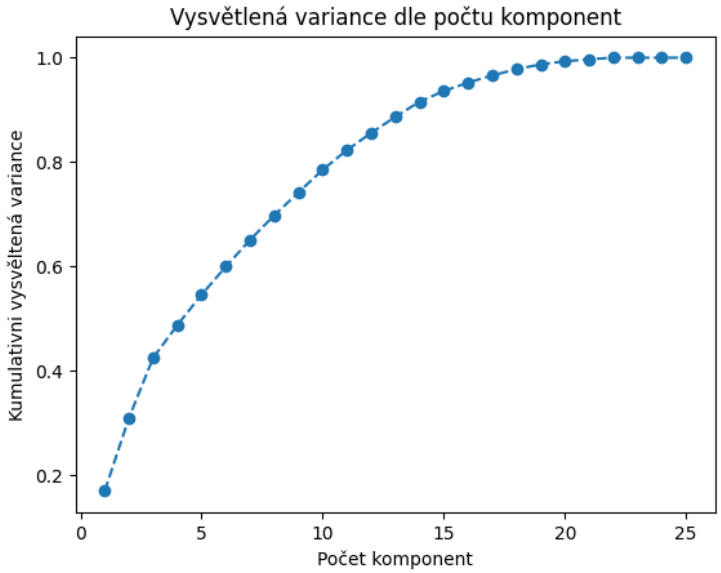
Z těchto naměřených výsledků si dovolím tvrdit, že je tento přístup poměrně silný, byť o něco slabší než náhodný výběr z komponent, který byl schopný datovou sadu zredukovat o 99.8%. Poskytnutá redukce je opravdu veliká a nedošlo ke ztrátě přesnosti, navíc musíme uvažovat fakt, že sada zůstane v tomto dobrém stavu pouze po určitou dobu než bude rozšířena o zachycené flows z reálného provozu. Stále však metoda vykazuje horší výsledky než náhodný výběr. Toto bych přisuzoval způsobu výběru řádků, které mají být odstraněny ze sady, i samotnému provedení experimentu. K Means při aplikaci na data vyšších dimenzí, v našem případě se jedná o opravdu velké matice, trpí na expanze Euklidovských vzdáleností a získané vzdálenosti nesprávně reprezentují skutečnost.

Před provedením dalšího experimentu jsem provedl zpracování datové sady pomocí PCA. Nejdříve jsem data naškáloval pomocí *StandardScaler* a po-

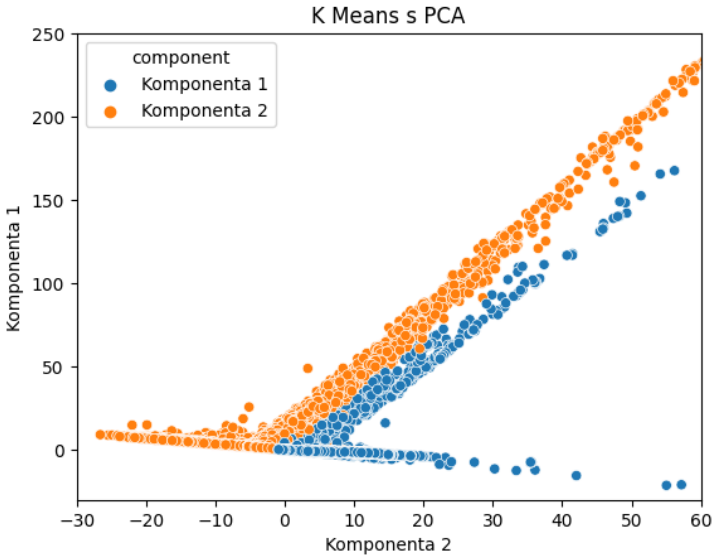


Obrázek 7.2: Hodnota W v závislosti na K pomocí Elbow metody

mocí PCA bez parametrů jsme získal poměr vysvětlených variancí jednotlivými komponentami. Graf 7.3 zobrazuje kumulativní vysvětlené variance v závislosti na počtu komponent. Pro experiment jsem zvolil varianci 80%, což odpovídá 10 komponentám.



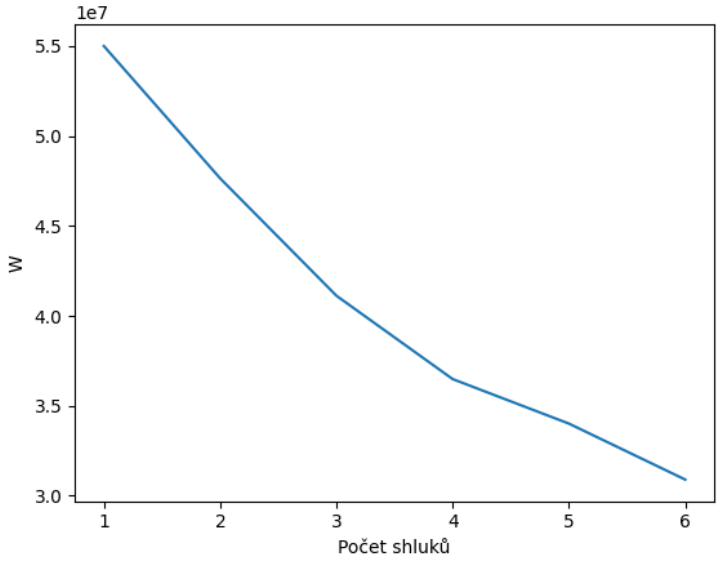
Obrázek 7.3: Vysvětlená variance dle počtu komponent



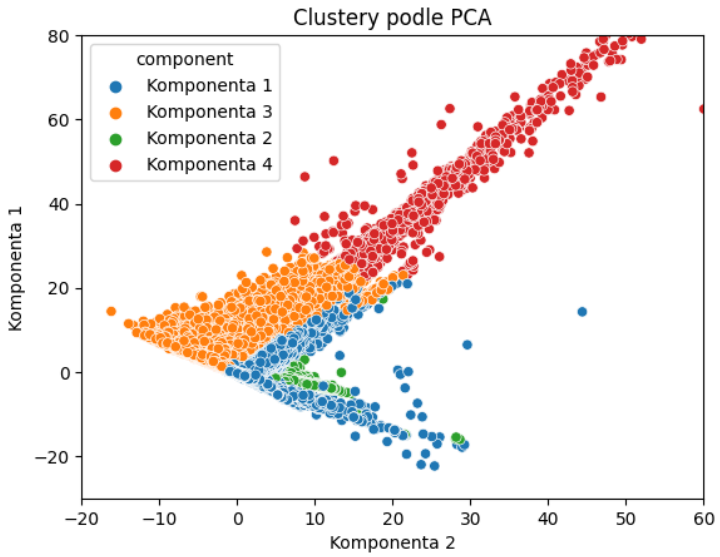
Obrázek 7.4: K Means s PCA pro 2 shluky

Následně jsem pomocí PCA s 10 komponentami získal ohodnocení prvků pomocí komponent. Další na řadě je volba počtu shluků pro K Means. Zkusil jsem nejdříve použít stejný počet shluků jako v předchozím experimentu, tedy dva. S těmito parametry jsem naměřil hodnotu F_1 0.98 a redukci 84.56%. V grafu 7.4 lze nahlédnout rozložení prvků podle prvních dvou nejdůležitějších komponent.

Protože zobrazené shluky se mi zdály hodně promíchané, rozhodl jsem se přepočítat počet shluků pro K Means pomocí Elbow metody za pomocí ohodnocení získaného z PCA. Hodnota Elbow metody je zobrazena v grafu 7.5. Nevýraznější zlom v grafu lze vidět pro čtyři komponenty, proto jsem tento počet komponent zvolil i pro tento experiment.



Obrázek 7.5: Hodnota funkce W pro počet shluků v K Means s PCA



Obrázek 7.6: K Means s PCA pro 4 shluky

K Means se čtyřmi komponenty je zobrazen v grafu 7.6. Graf je velmi podobný jako předchozí graf 7.4, ale jednotlivé komponenty jsou poměrně jasné definované a nedochází k příliš velkému vzájemnému mísení. Paradoxní je pak výsledná velikost sady. Sada získána touto metodu má sice vysokou přesnost 0.98, ale dochází pouze k 1.3% redukci oproti původní velikosti. Toto číslo lze pravděpodobně vylepšit použitím jiné metriky pro výběr prvků k zachování, než doposavad používané.

7.6 Autoenkodér

První z naprogramovaných autoenkodérů pracuje s vnitřní omezovací dimenzí o hodnotě 13, neboli s polovinou příznaků zaokrouhlenou nahoru, abych zaručil, že nedojde k pouhému kopírování vstupu na výstup. V průběhu testování jsem pak vyzkoušel několik druhů optimalizačních a ztrátových funkcí z knihovny Keras. První kombinací byl optimalizační funkci *Adadelta* se ztrátovou funkcí *Binary_Crossentropy*. Tato kombinace vykazovala po celou dobu učení velké ztráty na přesnosti, kdy ani po 50 iteracích nedošlo ke snížení míry ztrát. Dále jsem proto tuto kombinaci netestoval a použil druhou kombinaci funkcí, optimalizační funkcí *Adam* a ztrátovou funkcí *MSE*. Tato kombinace sice vykazovala menší ztrátovost než první, ale pořád se jednalo o ztráty v řádech jednotek, tudíž stále příliš velké. Poslední zkoušenou kombinací byla kombinace *SGD* a *CosineSimilarity*. U této kombinace se vyskytovala ztrátovost v řádech desetin, což je na poměry autenkodérů stále vysoká hodnota, ale podle mého názoru již dostatečně dobrá pro další testování.

Takto nakonfigurovaný autoenkodér, pak získal na vstupu celou datovou sadu a následným porovnáním zrekonstruovaného výstupu s původním vstupem pomocí korelačních koeficientů jednotlivých řádku jsem zvolil takové řádky, jejichž korelační koeficient byl vyšší než 0.98. Nápadem za tímto způsobem výběru bylo, že řádky, které projdou autoenkodérem a jsou nejméně změněné mají vysokou informační hodnotu, kterou již nelze více zkomprimovat a proto budou vhodné vybrat do výsledné datové sady.

Výsledky pro takto získanou datovou sadu však nejsou příliš optimistické. Sice lze výběrem prahové hodnoty korelačního koeficientu regulovat k jak moc velké redukci počtu řádku dojde, ale výsledné hodnoty F_1 skóre se pro různé velikosti výsledných sad pohybují v řádu $43 - 48\%$ přesnosti. Míra redukce, které se mi podařilo dosáhnout bez větší odchylky od přesnosti než stávající naměřené pak je 99.5% Takovou datovou sadu nelze považovat za optimální. Pokud bychom použili klasifikátor trénovaný nad touto sadou, tak nemůžeme očekávat přesné výsledky. Jedinými dobrými vlastnostmi pro takto získané sady jsou vyváženost, úplnost a absence duplicit.

SimEx autoenkodér jsem implementoval obdobně jako je popsáno v článku [42] rozebraném v sekci 4.6. Na rozdíl od implementace v článku jsem nepoužil model LeNet-5, ale použil jsem model z předchozího experimentu, pro-

vedeného výše, za cílem porovnání těchto dvou přístupů. Naprogramoval jsem tedy dva autoenkodéry, jeden specializovaný na záznamy, které jsou klasifikované jako DoH, a druhý specializovaný na jejich doplněk, neboli záznamy běžného HTTPS provozu. Ze znalostí z předchozích experimentů jsem použil stejnou kombinaci funkcí pro autoenkodér, *SGD* a *CosineSimilarity*.

Výběr řádků proběhl taktéž téměř identicky. Největším rozdílem bylo pouze rozdělení výběru, tak aby jeden autoenkodér nevybíral prvky, které dle anotace patří ke druhému, čímž by se mělo předejít duplicitám a případnému špatnému označení. Tento přístup evidentně spoléhá na to, že příchozí datová sada je korektně anotována. Na konci výběru jsem pak obě vybrané sady spojil dohromady.

Bohužel výsledky experimentů byly obdobné jako u jednoho autoenkodérů, tedy přesnost v rozmezí $43-48\%$. Hlavním rozdílem byla míra redukce, kdy se kombinaci autoenkodérů podařilo v nejlepším případě získat datovou sadu zredukovanou o 99.97% oproti originálu. Tato sada je pouze vyvážená a bohužel neobsahuje dostatek informací pro přesnou operaci klasifikátoru.

Navíc pro obě varianty autoenkodérů platí, že jsou velmi časově náročné. To plyne z faktu, že při každém použití je třeba autoenkodéry znovu natrénovat před použitím, protože každá datová sada je relativně proměnlivá a autoenkodér naučený na předchozí sadě nevykazuje tak dobré výsledky jako ten, který je naučen nově. Aby bylo dosaženo alespoň nějaké přesnosti je nutné nastavit při učení dostatečný počet trénovacích iterací, v obou variantách se jednalo o 100 iterací a průměrná doba jedné iterace pro obě varianty je 203 sekund.

Kapitola 8

Zhodnocení a diskuze výsledků

V předchozích několika kapitolách jsem představil různé metody strojového učení, nastínil jejich silné i slabé stránky, diskutoval existující řešení z akademické sféry a navrhl několik možných metod optimalizace. Tyto metody jsem v kapitole 6 popsal z pohledu jejich implementace, jakožto softwarového prototypu a následně jsem je v kapitole 7 otestoval. Tato kapitola se věnuje zhodnocení naměřených výsledků a diskuzi jejich implikací i následků.

Metoda náhodného výběru se ukázala jako extrémně jednoduchá na implementaci, přestože její výsledky nebyly příliš přesvědčivé. Zcela jistě lze tvrdit, že tuto metodu není vhodné používat. Metoda náhodného výběru z komponent naopak vykazovala výrazně lepší výsledky, navíc i s faktem, že výsledná datová sada byla vyvážená.

Otázkou však zůstává jakých výsledků by tato metoda dosahovala na sadách s více možnými hodnotami klasifikace, například na sadě TLS komunikace, kde jako klasifikace slouží SNI cílové destinace. Počet SNI, které se vyskytují v této sadě je totiž skutečně závratný a generování sady podle postupu popsaného v návrhu metody 5.2 by mohlo narazit na fakt, že některá SNI se vyskytují v sadě TLS jen jednou a vygenerovaná sada by obsahovala příliš málo záznamů na to aby se dala považovat za přesnou. Samozřejmě pak na druhou stranu je tato metoda poměrně rychlá, takže v případ, že nás až tolik netrápí vyváženost, či přesnost sady a chceme pouze redukovat velikost, může být tato metoda vhodná.

Velmi zajímavé metody vyprodukovala metoda K Means s odstraněním záznamů, které jsou ve vzdálenosti menší než je průměrná střední vzdálenost všech bodů od středu jejich shluku. Velkou výhodou této metody je její jednoduchost na implementaci pomocí modulů Pandas a scikit-learn. Při implementaci není třeba převodu mezi formáty a celá metoda lze poměrně snadno parametrizovat. Jako nevýhody tohoto přístupu bych označil potřebu správně určit počet shluků, což může být poměrně složitá záležitost v případě komplexnějších datových sad než je DoH. Další nevýhodou může, ale ne nutně musí být, je určování prvků, které se mají z datové sady vyřadit. Nemyslím

si, že každá datová sada bude mít jasně definované shluky s minimem protínání jako je tomu v případě sady DoH. Zároveň tento přístup nezaručuje vyváženost vzniklé sady, což nemusí být nutně na obtíž, ale může to způsobit bias u některých druhů klasifikátorů. Celkově bych však tuto metodu v rámci této práce označil jako velmi úspěšnou a domnívám se, že by ji bylo možné použít i pro další datové sady, které počtem klasifikací odpovídají DoH, tj. označují zda je daný flow zkoumaným typem komunikace či nikoliv, případně zda je komunikace jednoho či druhého typu.

Obdoba metody K Means kombinovaná s PCA přinesla velmi přesné výsledky, při použití dvou shluků pak dosáhla i velké míry redukce datové sady. Pro čtyři shluky, ale metoda dosáhla mizivé redukce velikosti sady. Domnívám se, že tato metoda může být využita pro optimalizaci, ale bude nutné najít správný počet shluků pro K Means nebo jinou metriku pro určení zachování či odstranění prvků, než byla použita v mnou provedených experimentech.

Co se týče časové složitosti, tak obě varianty K Means jsou výrazně pomalejší než náhodné výběry, ale zato jsou stále výrazně rychlejší než autoenkodéry.

Jednoduchý autoenkodér neposkytl příliš pozitivní výsledky. Lze diskutovat, že sice poskytuje velmi malou datovou sadu, která je úplná a vyvážená, ale to bohužel nevyrovná fakt, že výsledná přesnost sady je až o 40% menší než přesnost původní sady. Z toho pak vyvstává několik témat k diskuzi, jako toto zlepšit.

Hned prvním tématem je, zda je vůbec použití autenkodéru na tento problém vhodným řešením. Autoenkodéry jsou poměrně silné v ohledu na jaké množství problémů je lze použít. Bohužel dosahované výsledky ukazují, že se pravděpodobně nejedná o univerzální nástroj. Většina aplikací těchto neuronových sítí se zabývá kompresí obrázků či znaků a slov v textu. To si myslím, že je obor problémů, kde tento přístup skutečně vyniká a lze ho pro něj dobře implementovat. Problém optimalizace sad síťového provozu si naopak nemyslím, že je problém na který by bylo vhodné danou techniku používat ve světle provedených experimentů.

Druhým tématem k diskuzi je výběr vrstev autoenkodérů. Knihovna Keras poskytuje veliké množství vrstev pro použití v autoenkodérech. To samo o sobě představuje problém, jaké vrstvy zvolit a jak je poskládat, tak abychom dosáhli optimálních výsledků. Vzhledem k objemu nastavitelných parametrů si dovolím tvrdit, že důkladná analýza použití autoenkodérů nad zadaným problémem by vydala na stejné, ne-li větší, množství práce jako práce stávající a mohla by být zajímavým tématem pro budoucí výzkum.

Posledním tématem k diskuzi je metodika výběru prvků pro zachování a odstranění. Metoda použitá v experimentech s jedním autoenkodérem s pomocí korelačních koeficientů ukázala, že se nejedná o příliš vhodnou metodu, jelikož výsledná přesnost vytvořené sady nedosahovala ani 50%. Otázkou však zůstává zda by výsledky byly stejné i v případě použití jiných kódovacích a dekódovacích vrstev a jiných dimenzích těchto vrstev. Opět se však jedná ex-

perimenty takového rozsahu, které nejsou časově únosné v rámci této práce.

Autoenkodér založený na přístupu SimEx složený z autoenkodérů z předchozí metody bohužel dosáhl stejných výsledku, vyjma míry redukce, která se ukázala jako velmi silná. Obě varianty autoenkodérů vykazovali velkou časovou náročnost, které násobně překročila ostatní metody. Vzhledem k tomu, že se jedná o kombinaci předchozího autoenkodéru, tak i témata k diskuzi o této variantě zůstávají stejná a zastávám názor, že pro zjištění zda jsou autoenkodéry vhodným řešením zadaného problému je nutné jejich problematiku velmi podrobně nastudovat.

Ze všech navržených řešení se mi jako nejvhodnější jeví řešení využívající varianty K Means. Obě metody vykazují poměrně dobrou přesnost a metoda bez PCA i velikou míru redukce, zároveň na rozdíl od náhodného výběru lze přesně určit vybírané prvky. V obou variantách je pak pravděpodobně možné zavést vyvažování sady. A jako největší výhodu bych označil nepřeberné množství kombinací, které lze z této metody vytvořit pro budoucí práce a výzkum.

Všechna navržená řešení jsem implementoval do softwarového prototypu, v rámci které je možné je volně upravovat. Pro implementaci do frameworku ALF je možné použít celý modul nebo pouze funkce náhodného výběru z komponent a K Means metody. Osobně preferuji obě varianty K Means, protože nabízí větší míru dohledu nad výběrem dat. Autoenkodérové metody bych nedoporučoval vzhledem k jejich slabým výsledkům. V tabulce 8.1 jsou jednotlivé metody porovnány na základě průměrných hodnot doby běhu, F_1 skóre a nejlepší dosažené míry redukce sady.

Metoda	Doba běhu (s)	F_1 skóre	Míra redukce (%)
Náhodný výběr	5.89	0.51	40.00
Náhodný výběr z komponent	5.51	0.98	99.80
K Means	159.30	0.93	98.06
K Means s PCA (2 shluky)	254.34	0.98	84.56
K Means s PCA (4 shluky)	343.08	0.98	1.30
Autenkodér	24880.00	0.48	99.50
SimEx autoenkodér	24964.00	0.44	99.97

Tabulka 8.1: Srovnávací tabulka metod

Závěr

V rámci práce jsem v kapitole 2 představil základní pojmy používané v této práci, na které následně navázala rešerše možných metod optimalizace v kapitole 3 s analýzou a diskuzí existujících řešení v kapitole 4. Po provedení těchto analýz jsem navrhl šest metod optimalizace, od nejjednodušších založených na náhodném výběru, přes aplikaci shlukovacího algoritmu K Means a jeho spojení s metodou matematické analýzy PCA, až po ty nejsložitější metody založené na autoenkodérech a jejich kombinaci.

Návrhu a motivacím za těmito metodami byla věnována celá kapitola 5. V kapitole 6 jsem rozebral implementační specifika metod v rámci zvoleného programovacího jazyka Python.

V kapitole 7 jsem provedl zběžnou analýzu sady DoH, nad kterou jsem prováděl všechny experimenty, a popsal jsem způsob testování včetně klíčových hledaných metrik, jako je přesnost klasifikátoru trénovaného nad vzniklou sadou, velikost sady, její úplnost, či zda obsahuje případně prvky způsobující nejasnou klasifikaci.

Kapitola 8 byla cílena na zhodnocení experimentů a diskuzi implikací, vzešlých z testování. Obě metody náhodného výběr se nad testovanou sadou ukázali jako poměrně dobré, zvláště pak metoda náhodného výběru z komponent, která produkovala velice malou vyváženou datovou sadu, která dosahovala vysoké přesnosti. Nevýhodou těchto přístupu je právě jejich náhodnost a tudíž i poměrná nestabilita výsledků, byť opakované experimenty vykazovali podobné výsledky.

Nejkurióznější výsledky vykazovaly metody založené na K Means. Varianta K Means se dvěma shluky dosáhla více než 98% redukce počtu řádků oproti původní sadě s vysokou přesností 93% s nízkým počtem špatných klasifikací. Největší nevýhodou metody je, že nedochází k vyvažování komponent. K Means v kombinaci s PCA vykazují velmi vysokou přesnost 98% a nízký počet špatných klasifikací, ale nezaručuje její vyváženost. Míra redukce pak dosáhla hodnot 84.56% pro dva shluky a 1.3% pro čtuři shluky. S ohledem na to, že hlavním cílem práce je sadu zmenšit, tak se varianta metody K Means

s PCA o dvou shlucích projevila jako dobrý kandidát.

Autoenkodéry, které na první pohled vypadaly jako možná varianta řešení problému se nakonec po řadě útrap ukázali jako nevhodné. Získané datové sady z obou variant vykazují nízkou přesnost, byť jsou sady vyvážené a úplné. Stále zde je však možnost, že zvolené vrstvení a metodika výběru prvků nebyli pouze vhodné vybrané, jak jsem naznačil v diskuzi k výsledkům, a poskytují možné téma budoucího výzkumu, neboť se jedná o poměrně rozsáhlou problematiku.

Výsledný implementovaná prototyp je možné použít, jak bylo ostatně zamýšleno již při zadávání práce, v rámci vznikajícího frameworku Active Learning pro optimalizaci datových sad. Konkrétně se jedná o integraci do modulu *postprocessing*, kde dochází k úpravám datové sady, aby byla použitelná jako trénovací sada pro používané klasifikátory.

V práci jsem tedy naplnil pokyny zadané pro tuto práci, nastudoval jsem problematiku monitorování síťového provozu a tvorbu datových sad, seznámil jsem se s využitím strojového učení pro analýzu těchto sad a následně jsem navrhl, implementoval a otestoval algoritmy pro optimalizaci těchto datových sad.

Literatura

[1] Luxemburk, J.; Hynek, K.; et al. Detection of https brute-force attacks with packet-level feature set. In *2021 IEEE 11th Annual Computing and Communication Workshop and Conference (CCWC)*, IEEE, 2021, pp. 0114–0122.

[2] Sperotto, A.; Schaffrath, G.; et al. An overview of IP flow-based intrusion detection. *IEEE communications surveys & tutorials*, volume 12, no. 3, 2010: pp. 343–356.

[3] Cejka, T.; Bartos, V.; et al. NEMEA: a framework for network traffic analysis. In *2016 12th International Conference on Network and Service Management (CNSM)*, IEEE, 2016, pp. 195–201.

[4] Soukup, D.; Tisovčík, P.; et al. Towards Evaluating Quality of Datasets for Network Traffic Domain. In *2021 17th International Conference on Network and Service Management (CNSM)*, IEEE, 2021, pp. 264–268.

[5] Pereira, F.; Mitchell, T.; et al. Machine learning classifiers and fMRI: a tutorial overview. *Neuroimage*, volume 45, no. 1, 2009: pp. S199–S209.

[6] Schapire, R. E. Explaining adaboost. In *Empirical inference*, Springer, 2013, pp. 37–52.

[7] Yan, Y.; Rosales, R.; et al. Learning from multiple annotators with varying expertise. *Machine learning*, volume 95, no. 3, 2014: pp. 291–327.

[8] Sindlinger, T. S. Crowdsourcing: Why the power of the crowd is driving the future of business. 2010.

[9] Paun, S.; Simpson, E. Aggregating and Learning from Multiple Annotators. In *Proceedings of the 16th Conference of the European Chapter of the Association for Computational Linguistics: Tutorial Abstracts*, 2021, pp. 6–9.

[10] Russell, S. J.; Norvig, P. *Artificial Intelligence*. Pearson Education, 2009.

[11] Kubat, M.; Kubat. *An introduction to machine learning*, volume 2. Springer, 2017.

[12] Ratsaby, J.; Venkatesh, S. S. Learning from a Mixture of Labeled and Unlabeled Examples with Parametric Side Information. In *Proceedings of the Eighth Annual Conference on Computational Learning Theory*, COLT '95, New York, NY, USA: Association for Computing Machinery, 1995, ISBN 0897917235, p. 412–417, doi:10.1145/225298.225348. Available from: `https://doi.org/10.1145/225298.225348`

[13] Buhmann, J.; Kuhnel, H. Unsupervised and supervised data clustering with competitive neural networks. In *[Proceedings 1992] IJCNN International Joint Conference on Neural Networks*, volume 4, 1992, pp. 796–801 vol.4, doi:10.1109/IJCNN.1992.227220.

[14] Wiering, M.; Otterlo, M. v. *Reinforcement learning: State-of-the-art*. Springer, 2012.

[15] Roweis, S. T.; Saul, L. K. Nonlinear dimensionality reduction by locally linear embedding. *Science*, volume 290, no. 5500, 2000: p. 2323–2326, doi:10.1126/science.290.5500.2323.

[16] Berrar, D. Bayes' theorem and naive Bayes classifier. *Encyclopedia of Bioinformatics and Computational Biology: ABC of Bioinformatics*, volume 403, 2018.

[17] Sasaki, Y. *The truth of the F-measure*, Oct 2007.

[18] Abdi, H.; Williams, L. J. Principal component analysis. *Wiley Interdisciplinary Reviews: Computational Statistics*, volume 2, no. 4, 2010: p. 433–459, doi:10.1002/wics.101.

[19] Abdi, H. 2007. Available from: `https://personal.utdallas.edu/~herve/Abdi-SVD2007-pretty.pdf`

[20] Kanungo, T.; Mount, D.; et al. An efficient k-means clustering algorithm: analysis and implementation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, volume 24, no. 7, 2002: pp. 881–892, doi:10.1109/TPAMI.2002.1017616.

[21] Kodinariya, T. M.; Makwana, P. R. Review on determining number of Cluster in K-Means Clustering. *International Journal*, volume 1, no. 6, 2013: pp. 90–95.

[22] Sugar, C. A.; James, G. M. Finding the number of clusters in a dataset: An information-theoretic approach. *Journal of the American Statistical Association*, volume 98, no. 463, 2003: pp. 750–763.

[23] Hamerly, G.; Elkan, C. Learning the k in k-means. In *Advances in Neural Information Processing Systems*, volume 16, edited by S. Thrun; L. Saul; B. Schölkopf, MIT Press, 2003. Available from: https://proceedings.neurips.cc/paper/2003/file/234833147b97bb6aed53a8f4f1c7a7d8-Paper.pdf

[24] Anderson, T. W.; Darling, D. A. Asymptotic Theory of Certain "Goodness of Fit"Criteria Based on Stochastic Processes. *The Annals of Mathematical Statistics*, volume 23, no. 2, 1952: pp. 193 – 212, doi: 10.1214/aoms/1177729437. Available from: https://doi.org/10.1214/aoms/1177729437

[25] Whitley, D. A genetic algorithm tutorial. *Statistics and computing*, volume 4, no. 2, 1994: pp. 65–85.

[26] Rutenbar, R. Simulated annealing algorithms: an overview. *IEEE Circuits and Devices Magazine*, volume 5, no. 1, 1989: pp. 19–26, doi:10.1109/101.17235.

[27] Kirkpatrick, S.; Gelatt Jr, C. D.; et al. Optimization by simulated annealing. *science*, volume 220, no. 4598, 1983: pp. 671–680.

[28] Poli, R.; Kennedy, J.; et al. Particle swarm optimization. *Swarm Intelligence*, volume 1, no. 1, 2007: p. 33–57, doi:10.1007/s11721-007-0002-0.

[29] Dorigo, M.; Birattari, M.; et al. Ant colony optimization. *IEEE Computational Intelligence Magazine*, volume 1, no. 4, 2006: pp. 28–39, doi: 10.1109/MCI.2006.329691.

[30] Dhahri, H.; Rahmany, I.; et al. Tabu search and machine-learning classification of benign and malignant proliferative breast lesions. *BioMed Research International*, volume 2020, 2020.

[31] Pawlak, Z. Rough set theory and its applications. *Journal of Telecommunications and information technology*, 2002: pp. 7–10.

[32] Jensen, R.; Shen, Q. Semantics-preserving dimensionality reduction: rough and fuzzy-rough-based approaches. *IEEE Transactions on Knowledge and Data Engineering*, volume 16, 2004: pp. 1457–1471.

[33] Hedar, A.-R.; Wang, J.; et al. Tabu search for attribute reduction in rough set theory. *Soft Computing*, volume 12, no. 9, 2007: p. 909–918, doi:10.1007/s00500-007-0260-1.

[34] Jensen, R.; Shen, Q. Fuzzy-rough sets for descriptive dimensionality reduction. In *2002 IEEE World Congress on Computational Intelligence. 2002 IEEE International Conference on Fuzzy Systems. FUZZ-IEEE'02.*

Proceedings (Cat. No.02CH37291), volume 1, 2002, pp. 29–34 vol.1, doi: 10.1109/FUZZ.2002.1004954.

[35] Goodfellow, I.; Bengio, Y.; et al. *Deep Learning*. MIT Press, 2016, `http://www.deeplearningbook.org`.

[36] Saraswathy, V. R.; Prabhu Ram, M.; et al. Application of Rough Set Based Reduction for Network data set. In *2018 International Conference on Intelligent Computing and Communication for Smart World (I2C2SW)*, 2018, pp. 175–177, doi:10.1109/I2C2SW45816.2018.8997516.

[37] Velayutham, C.; Thangavel, K. Unsupervised quick reduct algorithm using rough set theory. *Journal of electronic science and technology*, volume 9, no. 3, 2011: pp. 193–201.

[38] Abdullah, S.; Sabar, N. R.; et al. A constructive hyper-heuristics for rough set attribute reduction. In *2010 10th International Conference on Intelligent Systems Design and Applications*, 2010, pp. 1032–1035, doi: 10.1109/ISDA.2010.5687052.

[39] Lin, W.; Wu, Y.; et al. Attribute Reduction of Rough Set Based on Particle Swarm Optimization with Immunity. In *2008 Second International Conference on Genetic and Evolutionary Computing*, 2008, pp. 14–17, doi:10.1109/WGEC.2008.94.

[40] Caliskan, A.; Badem, H.; et al. The effect of autoencoders over reducing the dimensionality of a dermatology data set. In *2016 Medical Technologies National Congress (TIPTEKNO)*, 2016, pp. 1–4, doi: 10.1109/TIPTEKNO.2016.7863101.

[41] Zhou, Y.; Amimeur, A.; et al. Density-aware Local Siamese Autoencoder Network Embedding with Autoencoder Graph Clustering. In *2018 IEEE International Conference on Big Data (Big Data)*, 2018, pp. 1162–1167, doi:10.1109/BigData.2018.8621992.

[42] Hwang, I.; Lee, J.; et al. Simex: Express prediction of inter-dataset similarity by a fleet of autoencoders. *arXiv preprint arXiv:2001.04893*, 2020.

[43] Brownlee, J. Autoencoder feature extraction for classification. Dec 2020. Available from: `https://machinelearningmastery.com/autoencoder-for-classification/`

[44] Brownlee, J. Autoencoder feature extraction for classification. Dec 2020. Available from: `https://machinelearningmastery.com/autoencoder-for-classification/`

[45] Vekshin, D.; Hynek, K.; et al. Doh insight: Detecting dns over https by machine learning. In *Proceedings of the 15th International Conference on Availability, Reliability and Security*, 2020, pp. 1–8.

Příloha **A**

Obsah přiloženého USB

- `readme.txt` ..... stručný popis obsahu USB
- `thesis` ..... adresář zdrojových souborů textu
  - `img` ..... adresář s obrázky
  - `thesis.cls` ..... zdroj šablon pro .tex soubor
  - `thesis.bst` ..... zdroj formátování citací
  - `references.bib` ..... zdrojová forma citací
  - `DP-Skruzny-Petr-B212.tex` ..... zdrojový soubor práce
- `text` ..... adresář s textem práce
  - `DP-Skruzny-Petr-B212.pdf` ..... text práce ve formátu PDF
- `src` ..... adresář se zdrojovými kódy
  - `datasetOptimizer.py` ..... softwarový prototyp v jazyce Python