FAKULTA INFORMAČNÍCH TECHNOLOGIÍ ČVUT V PRAZE

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

Název:	Posilováneho učení a neuronové sítě pro extrakce znalostí z textu
Student:	Pavel Hlubík
Vedoucí:	Juan Pablo Maldonado Lopez, Ph.D.
Studijní program:	Informatika
Studijní obor:	Znalostní inženýrství
Katedra:	Katedra teoretické informatiky
Platnost zadání:	Do konce letního semestru 2018/19

Pokyny pro vypracování

Hlavním cílem této práce je provést průzkum různých metod pro extrakci znalostí z textu. V rámci práce se budou zkoumat různé algoritmy a analyzovat jejich výkonnost pro různé skupiny dokumentů.

1) Seznamte se s technikami extrakce znalostí z textu (Tf-Idf, stemming, lemmatizace).
2) Prozkoumejte možnosti posilovaného učení pro klasifikaci dokumentů.
3) Prozkoumejte rekurentní neuronové sítě pro generování textů.
4) Seznamte se s metodami a nástroji pro hlubokého učení (Keras, pyTorch) a možnostmi jejich využití pro extrakci znalostí z textu.
5) Porovnejte jednotlivé nástroje z hlediska jejich přínosu pro extrakci znalostí z textu.

Seznam odborné literatury

Dodá vedoucí práce.

doc. Ing. Jan Janoušek, Ph.D.
vedoucí katedry

doc. RNDr. Ing. Marcel Jiřina, Ph.D.
děkan

V Praze dne 4. ledna 2018

FAKULTA
INFORMAČNÍCH
TECHNOLOGIÍ
ČVUT V PRAZE

Bakalářská práce

Posilované učení a neuronové sítě pro extrakci znalostí z textu

Pavel Hlubík

Katedra teoretické informatiky
Vedoucí práce: MSc. Juan Pablo Maldonado Lopez Ph.D.

6. května 2018

Poděkování

Na tomto místě bych chtěl poděkovat svému vedoucímu bakalářské práce, kterým byl Juan Pablo Maldonado Lopez, Ph.D., za trpělivost a ochotu při častých konzultacích a cenné rady, bez kterých bych tuto práci nedokázal dokončit.

Prohlášení

Prohlašuji, že jsem předloženou práci vypracoval(a) samostatně a že jsem uvedl(a) veškeré použité informační zdroje v souladu s Metodickým pokynem o etické 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. V souladu s ust. § 46 odst. 6 tohoto zákona tímto uděluji nevýhradní oprávnění (licenci) k užití této mojí práce, a to včetně všech počítačových programů, jež jsou její součástí či přílohou, a veškeré jejich dokumentace (dále souhrnně jen „Dílo“), a to všem osobám, které si přejí Dílo užít. Tyto osoby jsou oprávněny Dílo užít jakýmkoli způsobem, který nesnižuje hodnotu Díla, a za jakýmkoli účelem (včetně užití k výdělečným účelům). Toto oprávnění je časově, teritoriálně i množstevně neomezené. Každá osoba, která využije výše uvedenou licenci, se však zavazuje udělit ke každému dílu, které vznikne (byť jen zčásti) na základě Díla, úpravou Díla, spojením Díla s jiným dílem, zařazením Díla do díla souborného či zpracováním Díla (včetně překladu), licenci alespoň ve výše uvedeném rozsahu a zároveň zpřístupnit zdrojový kód takového díla alespoň srovnatelným způsobem a ve srovnatelném rozsahu, jako je zpřístupněn zdrojový kód Díla.

V Praze dne 6. května 2018 .....

Odkaz na tuto práci

Hlubík, Pavel. *Posilované učení a neuronové sítě pro extrakci znalostí z textu.* Bakalářská práce. Praha: České vysoké učení technické v Praze, Fakulta informačních technologií, 2018.

Abstrakt

Cílem práce je zhodnotit přínos konkrétních algoritmů posilovaného učení a neuronových sítí pro klasifikací textových dokumentů a sumarizaci textu. Je sekvenční čtení rychlejší a přesnější, než Bayesovská klasifikace? Je posilované učení vhodné pro sumarizaci textu? Literární rešerše se zabývá analýzou aktuální literatury k této problematice. V praktické části jsou pak podrobně zdokumentovány implementace a výsledky konkrétních algoritmů (Corss Entropy Methods, Reinforce). Všechny algoritmy jsou implementovány v jazyce Python. Na základě zjištěných údajů je možné usoudit, které algoritmy jsou pro řešení daných problémů použitelné a za jakých okolností a které nikoliv. V příloze práce lze nalézt zdrojové kódy všech hodnocených algoritmů.

Klíčová slova analýza algoritmů pro text mining, posilované učení, rekurentní neuronové sítě, klasifikace dokumentů, sumarizace textu

Abstract

The goal of this thesis is to evaluate contribution of algorithms of Reinforcement learning and neural networks for text classification and summarization. Is sequential reading faster but more precise than naive Bayesian classification? Is Reinforcement learning suitable for text summarization? Theoretical part deals with analysis of current literature regarding this problematic. Practical part consists of elaborately documented implementation and results of specific algorithms (Cross Entropy Method, Reinforce). All algorithms are implemented in Python language. Based on reported results it is possible to decide which algorithms are suitable for solving given problems and under what circumstances and which are not. In the attachment of the thesis contains source codes of all evaluated algorithms.

Keywords analysis of algorithms for text mining, reinforcement learning, recurrent neural networks, document classification, text summarization

Obsah

Seznam obrázků

Seznam tabulek

Úvod

Strojové učení a vytěžování znalostí z dat jsou dynamicky se rozvíjející odvětví, ve kterých se objevují stále nové metody. Velká část z dostupných informací se ale nevyskytuje ve strukturované podobě, ale v nestrukturovaných člověkem psaných textech. Pro vytěžování znalostí z takovýchto zdrojů je potřeba navrhnout speciální přístupy k řešení.

V této práci se zaměřím na dva konkrétní problémy, klasifikaci textových dokumentů do kategorií a sumarizaci textových dokumentů, a na možné přístupy k jejich řešení. Klasifikace je aktuální při detekci spamu, odhalování fake news, identifikaci nenávistných příspěvků na sociálních sítích a v mnoha dalších případech. Sumarizace je pak přínosná kdykoliv se potřebujeme zorientovat ve velké záplavě informací, například v množství uživatelských recenzí nebo novinových článků. V posledních letech se objevila myšlenka využít framework posilovaného učení (reinforcement learning), obvykle používaný pro umělou inteligenci schopnou učit se z vlastní zkušenosti, pro řešení těchto problémů. Výsledky výzkumu v této oblasti se ovšem liší a přínos nových metod pro problematiku není zřejmý.

Dalším nástrojem již delší dobu používaným pro text mining jsou neuronové sítě, respektive převážně jejich podtyp, rekurentní neuronové sítě. Ty lze využít jako jednu ze součástí posilovaného učení, není to však nutné, celý framework na nich není závislý, nebo i zcela samostatně.

Práce je určena pro čtenáře, který se chce zorientovat v nových metodách pro text mining a hledá jejich srovnání s klasickými postupy.

Tato práce si klade za cíl prozkoumat možnosti využití posilovaného učení a neuronových sítí pro klasifikaci a sumarizaci dokumentů, jejich přínos pro problematiku a porovnat výsledky dosažené tímto způsobem s výsledky klasických metod. Hlavní otázkou je, jestli agent čtoucí dokument po větách je schopen lépe identifikovat informaci obsaženou v dokumentu a díky tomu být stejně rychlý a přesnější než Bayesovská klasifikace?

Téma práce jsem si vybral, protože posilované učení se v poslední době ukazuje jako schopné řešit problémy úplně jiné podstaty, než pro které bylo

původně popsáno, a tedy jako velmi univerzální přístup ke strojovému učení. V této práci chci prozkoumat jedenu z takových možných nových aplikací.

Tato práce dále pokračuje v následující struktuře: Teoretická část v kapitolách 2 a 3, praktická část v kapitole 4. Kapitola 2 popisuje framework posilovaného učení a kapitola 3 popisuje možnosti předzpracování textu a analyzuje možnosti, jak použít posilované učení pro problémy klasifikace a sumarizace. Kapitola 4 pak popisuje použití, implementaci a výsledky algoritmů Reinforce a CEM a Q-learning pro tyto problémy.

Kapitola 1

Cíl práce

Cílem teoretické části práce (kapitoly 2 a 3) je analýza problematiky posilovaného učení, vysvětlení jeho principů a základních pojmů a přehled některých algoritmů využívaných Posilovaným učením. Dále pak seznámení se s možnostmi předzpracování textu, obecný úvod do klasifikace a sumarizace textů, analýza různých přístupů k této problematice. V poslední části rešerše je pak popsán možný způsob, jak problémy klasifikace a sumarizace formálně definovat jako problémy posilovaného učení.

Cílem praktické části je pak návrh a implementace řešení výše popsaných a definovaných problémů posilovaného učení za pomoci algoritmů Reinforce, Cross Entropy Method a Q-learning. Otázkou pro klasifikaci je, jestli je agent schopen se naučit sekvenčně číst dokument a pokud ano, je schopen rychleji dosáhnout stejně přesných výsledků, jako Bayes? U sumarizace se práce zabývá tím, jestli různá vektorová reprezentace stavů přináší rozdílné výsledky.

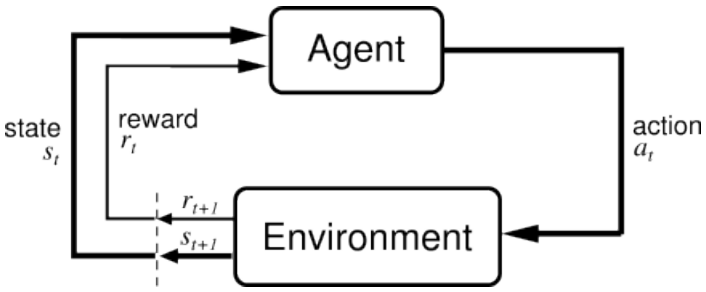
Kapitola 2

Posilované učení

Myšlenka, že se učíme interakcí s okolním prostředím je asi to první, co nás napadne, když budeme přemýšlet o podstatě učení. Opakovaná interakce s prostředím produkuje velké množství informací o příčinách událostí a následcích akcí, a tedy o tom, co máme dělat, abychom dosáhli nějakého určitého cíle. Pokud si dítě hraje, nemá žádného explicitního učitele, který by ho učil, přesto se postupem času zdokonalí a lépe ví, co musí udělat, aby dosáhlo toho či onoho cíle. Interakce s prostředím ale zůstává důležitým zdrojem informací i v pozdějším věku, ať už se učíme řídit auto nebo vést konverzaci, pořád sledujeme, jak nás naše akce přibližují k žádanému cíli a upravujeme svoje chování v závislosti na zkušenosti.

Posilované učení (reinforcement learning) je způsob, framework, inspirovaný myšlenkou učení se z interakce s okolním prostředím, kterým se dokáže učit počítačový algoritmus. Tento přístup je silně orientovaný na dosažení cíle – co nejvyšší odměny, než jiné přístupy strojového učení. Hlavním cílem je, aby se agent naučil, co dělat v jaké situaci – mapovat akce na situace tak, aby byla dosažená odměna co nejvyšší. Jediný způsob, jak se agent může dozvědět, která akce je v dané situaci dobrá, je, že ji vyzkouší. Dalším důležitým faktorem je, že zvolené akce ovlivňují chování prostředí v budoucnu – akce, která ihned nevede k vysoké odměně, může zapříčinit zisk daleko vyššího skóre v budoucnu a naopak akce, která přináší vysokou okamžitou odměnu, může agenta zavést do slepé uličky. Postupné zkoušení možností, chybování a opožděný zisk odměny jsou určující vlastnosti posilovaného učení.

V situaci, kdy není možné získat či uchovat dostatek informací o chování prostředí potřebných pro klasické metody supervizovaného učení, může posilované učení dosáhnout velmi dobrých výsledků. Před agentem pohybujícím se v prostředí ale vyvstává zásadní problém. Na jednu stranu musí využívat dosavadní znalosti o zákonitostech prostředí pro zisk co nejvyšší odměny (exploitation), na druhou stranu ale musí zkoušet akce, které se mu v daný okamžik podle jeho dosavadních znalostí nejeví jako nejlepší, aby tak dokázal objevit nové zákonitosti a eventuálně získat vyšší odměnu (exploration).



Obrázek 2.1: Interakce agenta s prostředím [1].

Nalezení rovnováhy těchto dvou tendencí, exploration a exploitation je dalším ze základních problémů posilovaného učení, který se u supervizovaného učení obvykle nevyskytuje. [5, str. 3–6]

2.1 Elementy posilovaného učení

Mezi základní prvky ve frameworku posilovaného učení patří již zmíněné *prostředí*, *agent*, který s prostředím interaguje, učí se a činí rozhodnutí, *strategie* (policy), *funkce odměny* (reward function) a *hodnotová funkce* (value function)

Strategie definuje chování agenta v prostředí v čase. Zhruba řečeno se jedná o mapování z prostoru všech agentem vnímaných stavů do prostoru možných akcí.

Funkce odměny definuje cíl v problému posilovaného učení. Tato fukce mapuje agentem pozorovaný stav prostředí, nebo pár stav, akce na jediné číslo – odměnu.

Zatímco funkce odměny specifikuje, co je výhodné v jeden okamžik, *hodnotová funkce* značí, co je nejlepší z dlouhodobého hlediska. Přibližně se jedná o celkovou hodnotu odměn, kterou může agent očekávat, pokud vyjde z tohoto stavu[5, str. 7–10].

Interakci agenta s prostředím shrnuje obrázek 2.1. V čase $t \in \mathbb{N}$ obdrží agent reprezentaci stavu prostředí $s_t \in \mathcal{S}$, kdy $\mathcal{S}$ je množina všech možných stavů prostředí, a na jeho základě zvolí *akci* $a_t \in \mathcal{A}(s_t)$, kde $\mathcal{A}(s_t)$ je množina všech možných akcí ve stavu s_t. O jeden diskrétní časový krok později, částečně následkem zvolené akce, obdrží agent odměnu $r_{t+1} \in \mathbb{R}$ a bude se nacházet ve stavu s_{t+1}.

V každém kroku provádí agent mapování ze stavu na pravděpodobnosti zvolení každé možné akce. Toto je již zmíněná *strategie* agenta značená π_t, kdy $\pi_t(s, a) = P(a_t = a | s = s_t)$. Metody posilovaného učení specifikují, jak agent mění tuto svou strategii na základě zkušenosti.

Takto navržený framework je flexibilní a může být použit na různé problémy mnoha různými způsoby. Časové kroky nemusí být přesně dané, akce mohou být jakékoliv od regulace napětí v motoru robotické paže po čtení vět textového dokumentu[5, str. 51–56].

2.2 Celková odměna

Jak již bylo řečeno, hlavním cílem agenta je maximalizovat celkovou kumulativní odměnu (return), kterou získá. Pokud je sekvence odměn získaných po čase t $r_{t+1}, r_{t+2}, r_{t+3}...$, je otázkou, jaký přesný aspekt této sekvence chceme maximalizovat. Obecně jde o nějakou funkci očekávané celkové odměny R_t, v nejjednodušším případě pak:

$$R_t \coloneqq r_{t+1} + r_{t+2} + r_{t+3} + ... + r_T$$

kdy T značí finální časový krok. Tento přístup dává smysl tam, kde je interakci agenta s prostředím možno rozdělit do subsekvencí, které nazýváme *epizody*. Epizodou může být například jeden průchod robota bludištěm, nebo jedna partie dámy. Čas T pak logicky značí konec této epizody, ať už je pro agenta příznivý či nikoliv. V případě, že je interakce agenta s prostředím kontinuální, a prostředí je tudíž neepizodické, je výše uvedená rovnost problematická, jelikož $T = \infty$ a součet takovéto řady nemusí být známý.

Dalším potřebným konceptem je diskontování odměn, takzvaný *discounting*. V tomto případě je každá odměna r_{t+k} před sečtením vynásobena mocninou koeficientu γ v závislosti na tom, jak daleko v čase se od času t nachází. Výsledná rovnost pak tedy vypadá následovně:

$$R_t \coloneqq r_{t+1} + \gamma^1 r_{t+2} + \gamma^2 r_{t+3} + ... = \sum_{k=0}^{\infty} \gamma^k r_{t+k\ +1}$$

kdy $0 \leq \gamma \leq 1$ je diskontní faktor. Pokud $\gamma = 0$, koncentruje se agent pouze na nejbližší odměnu a pokud $\gamma < 1$, má výše zmíněná řada konečný součet [5, str. 57–60].

2.3 Hodnotové funkce

Skoro všechny algoritmy posilovaného učení jsou založeny na odhadech hodnotových funkcí. Jsou to funkce buď stavu prostředí, nebo páru stavu a akce, které se snaží odhadnout, jak „dobré“ je pro agenta nacházet se v daném stavu, respektive jak „dobré“ je v daném stavu zvolit danou akci. To samozřejmě závisí na tom, jakou strategií se agent řídí, proto jsou funkce definovány ve vztahu k politice.

Neformálně lze hodnotu stavu s při politice π, značenou jako $V^\pi(s)$, definovat jako očekávanou celkovou odměnu, pokud se agent nachází ve stavu s.

Formálně pak jako [5, str. 68–75]:

$$V^{\pi}(s) \coloneqq E_{\pi}\{R_t|s_t = s\} = E_{\pi}\{\sum_{k=0}^{\infty} \gamma^k r_{t+k+1}|s_t = s\}$$

Analogicky můžeme hodnotu akce a ve stavu s při politice π, značenou $Q^{\pi}(s, a)$, definovat jako očekávanou celkovou odměnu, pokud se agent nachází ve stavu s, zvolí akci a a následně se bude řídit strategií π. Formálně pak jako [5, str. 68-75]:

$$Q^{\pi}(s, a) \coloneqq E_{\pi}\{R_t|s_t = s, a_t = a\} = E_{\pi}\{\sum_{k=0}^{\infty} \gamma^k r_{t+k+1}|s_t = s, a_t = a\}$$

Tyto funkce je možné odhadnout ze zkušenosti. Bude-li se agent řídit strategií π a zaznamenávat průměrnou celkovou dosaženou odměnu, budou se hodnoty blížit skutečným hodnotám funkce $V^{\pi}(s)$ s tím, jak se bude počet průchodů jednotlivými stavy blížit nekonečnu. Analogicky budou při aplikaci stejného postupu pro páry stav – hodnota naměřené hodnoty konvergovat ke skutečné $Q^{\pi}(s, a)$. Metody odhadu těchto funkcí založené na této úvaze se nazývají metody *Monte Carlo*. [5, str. 68–75]

2.4 Metody odhadu hodnotových funkcí

Výše zmíněné formální definice funkcí nám bohužel neposkytují návod jak v praxi získat jednotlivé potřebné funkční hodnoty, proto je v praxi používáno široké spektrum metod, které mají za cíl jednotlivé funkční hodnoty *odhadnout*, *aprodimovat*.

2.4.1 Monte Carlo

Tyto metody jsou vhodné pouze pro epizodická prostředí – prostředí, kde je zkušenost rozdělena do dobře odlišitelných epizod, které vždy končí v konečném čase, bez ohledu na to, jaké akce agent v průběhu volí. Řeší dva základní problémy, problém *predikce*, tedy nalezení hodnot funkce V^{π} a problém *kontroly*, tedy nalezení optimální strategie π Aktualizace hodnot funkcí $V^{\pi}(s)$ a $Q^{\pi}(s, a)$ pak probíhá pouze mezi jednotlivými epizodami. Postup dobře ilustruje obrázek 2.2, takzvaný *backup diagram*. [5, str. 118–122]

Backup diagram pro metodu Monte Carlo. Plné kruhy ilustrují akce, prázdné kruhy pak stavy.

Důležitou vlastností takového přístupu je, že hodnoty takto odhadnuté funkce $V^{\pi}(s)$ jsou na sobě nezávislé, pro jejich výpočet je použita pouze celková odměna a ne jiné hodnoty $V^{\pi}(s)$. Je zřejmé, že výpočetní složitost odhadu hodnoty funkce pro jeden konkrétní stav nezávisí na celkovém počtu stavů. Díky tomu může tato metoda být dobrou volbou, pokud je cílem odhadnout

Obrázek 2.2: Monte Carlo backup diagram [2].

$V^\pi(s)$ pouze pro nějakou podmnožinu stavů a ne pro všechny stavy. [5, str. 118–122]

Při odhadování $V^\pi(s)$ jsme dosud implicitně předpokládali, že máme k dispozici *model* prostředí, tedy že agent ví přesně, jakou akci zvolit ve stavu, ve kterém se prostředí právě nachází, aby byl výsledkem jím požadovaný nový stav. Tuto informaci ale agent často nemá, proto může být užitečnější odhadovat hodnoty páru stav – akce, tedy $Q^\pi(s, a)$.

Jediným problémem takového přístupu zůstává, že v případě, kdy se řídíme deterministickou strategií, nemusí být některé akce v daném stavu nikdy vyzkoušeny, což znemožňuje efektivní progres. Musíme tedy volit *stochastickou* strategii [5, str. 112–118], u které má každá akce v každém stavu nenulovou pravděpodobnost, že bude vybrána.

Tím je ohodnoceno, jak dobrých výsledků můžeme dosáhnout, pokud budeme sledovat strategii π. Jak ale dosáhnout zlepšení? Je to možné tak, že strategii založíme na hodnotách průběžně počítané funkce $Q^\pi(s, a)$. Strategie, která pro stav s volí akci a s nejvyšší hodnotou $Q^\pi(s, a)$ s pravděpodobností $1 - \epsilon$ a náhodnou akci s pravděpodobností ϵ se nazývá ϵ-hladová strategie (ϵ-greedy policy)[5, str. 112–118] a je obvyklou volbou.

2.4.2 Metody Temporální diference

Metody temporální diference (TD) oproti metodám Monte Carlo nečekají na finální výsledek – konec epizody, ale snaží se učit v průběhu samotné epizody (bootstraping). Řeší opět problém predikce a problém kontroly. Pro predikci je možné využít algoritmus TD(0), pro kontrolu pak algoritmy SARSA a q-learning. Existují i zobecněné verze těchto algoritmů TD(λ) a Sarsa(λ), které do sebe integrují principy jak TD, tak Monte Carlo[5, str. 133–138].

Esenciálním rozdílem algoritmů Sarsa a q-learning je, že Sarsa je takzvaně

on-policy, tedy že používá stejnou strategii jak pro odhad hodnoty dané strategie, tak pro kontrolu – volbu akcí v prostředí, a q-learning je takzvaně *off-policy*. V tomto druhém případě jsou tyto dvě strategie odděleny, strategie použitá pro generování akcí může být založena na volbě náhodných akcí po celou dobu, zatímco strategie, pro kterou odhadujeme $Q^\pi(s,a)$ může být deterministická, například hladová. [5, str. 126–128]

Sarsa je založena na pětici událostí $(s_t, a_t, r_{t+1}, s_{t+1}, a_{t+1})$, od které je také odvozeno jméno algoritmu. Aktualizace hodnoty funkce $Q^\pi(s,a)$ pak probíhá po každém přechodu ze stavu, který není konečný (nekončí v něm epizoda), do jiného stavu a je definována následovně:

$$Q^\pi(s_t, a_t) \longleftarrow Q^\pi(s,a) + \alpha[r_{t+1} + \gamma Q^\pi(s_{t+1}, a_{t+1}) - Q^\pi(s,a)]$$

kdy $0 < \alpha <= 1$ značí *learning rate*, tedy to, jakou důležitost přikládáme aktuální zkušenosti oproti již aproximované hodnotě. Pokud je stav s_{t+1} konečný, je hodnota $Q^\pi(s_{t+1}, a_{t+1})$ definována jako 0. [5, str. 145–148]

Q-learning je ve své nejjednodušší podobě definován takto:

$$Q^\pi(s_t, a_t) \longleftarrow Q^\pi(s,a) + \alpha[r_{t+1} + \gamma \max_a Q^\pi(s_{t+1}, a) - Q^\pi(s,a)]$$

Tímto způsobem získané hodnoty funkce Q přímo aproximují optimální hodnotovou funkci Q^* nezávisle na tom, jaká strategie je používána k volbě akcí. [5, str. 148–151] Rozdíl mezi oběma algoritmy je možné vidět i na jejich *backup* diagramech. Rozdíl mezi oběma algoritmy je možné také pozorovat na jejich backup diagramech (obrázek 2.3).

Obrázek 2.3: Backupdiagramy pro q-learning a sarsa [3].

2.4.3 Parametrizace

Všechny výše zmíněné metody pracují s reprezentací funkce v podobě matice, tabulky, ve které je možné jednoduše vyhledávat požadované funkční hodnoty. Představíme-li si, že každý řádek je stav a že každý sloupec je jedna z akcí, je jasné, že tato metoda je použitelná pouze pro diskrétní konečné stavové prostory s rozumným počtem stavů [6]. Je ovšem otázkou, jak často se při řešení praktických problémů setkáme s právě takovým prostorem.

Další možností je parametrizovat funkci $Q(s, a)$ vektorem parametrů θ. Problém učení se pak sestává z problému nalezení příslušného vektoru parametrů. Obecně lze pak vztah definovat takto [7]:

$$Q(s, a) \coloneqq \langle \Phi(s, a), \theta \rangle$$

Funkce $\Phi(s, a)$ značí nějakou vektorovou reprezentaci páru stav-akce. Její konkrétní forma je zcela předmětem návrhu řešení a může mít mnoho podob. $\langle ., . \rangle$ je pak jakákoliv funkce vhodná k řešení regresního problému. Její volba je opět zcela na nás, může se jednat o prostý vektorový součin, ale i o neuronovou síť. V případě, že je zvolena metoda vyžadující supervizované učení, jsou pro učení např. neuronové sítě využity hodnoty $Q(s, a)$ pozorované agentem při náhodné volbě akcí. [7]

V této práci budou dále Neuronové sítě, konkrétně Rekurentní neuronové sítě, využity k aproximaci funkce $Q(s, a)$. Základní stavební jednotkou neuronové sítě je *neuron*. Neuron přijímá vstup jako výstup neuronů z předchozí vrstvy, kterému přiřadí váhy, sečte ho a výsledek použije jako argument své vnitřní funkce. Vnitřní funkcí neuronu, které se také jinak říká *aktivační* funkce, bývá velmi často *sigmoida* definovaná vztahem:

$$\sigma(z) = \frac{1}{1 + e^{-z}}$$

kde z je suma vážených vstupů. Hodnota $\sigma(z)$ je pak výstupem neuronu. [8]

Výstup poslední vrstvy neuronů pak interpretujeme jako výstup neuronové sítě. Pro regresi je ve výstupní vrstvě obvykle jeden neuron, pro klasifikaci může být počet neuronů roven počtu tříd a pro jiné problémy záleží na programátorovi. Správnost výstupu lze pak hodnotit pomocí *ztrátové funkce* (loss function), kterou může být standardní normální odchylka výstupu jednotlivých neuronů od požadovaného výsledku.

Na ztrátové funkci je pak založen i jeden ze základních algoritmů pro učení neuronové sítě *stochastic gradient descent*. Zjednodušeně lze říci, že jeho cílem je minimalizovat hodnotu ztrátové funkce pro trénovací data. Ztrátová funkce je funkcí výstupu neuronové sítě, který je zase funkcí vstupních dat a vah jednotlivých neuronů. Teoreticky je tedy možné spočítat derivaci v bodě ztrátové funkce podle jednotlivých vah a následně v nějaké míře (learning rate) váhy upravit proti směru derivace, a tak snížit hodnotu ztrátové funkce. [8]

Jedním z typů neuronových sítí jsou *rekurentní* neuronové sítě, které se snaží zachytit *sekvenční* informaci, jaká je typická právě třeba pro texty. Rekurentní neuron má paměť, ve které je uložen jeho stav na základě předchozích vstupů. Jeho výstup je pak závislý nejen na vstupu, ale také na stavu a na vahách, které jsou stavu přisouzeny. Jak váhy vstupu, tak stavu jsou pak předmětem učení neuronové sítě. [9]

Kapitola 3

Klasifikace a sumarizace textu

3.1 Předzpracování textu

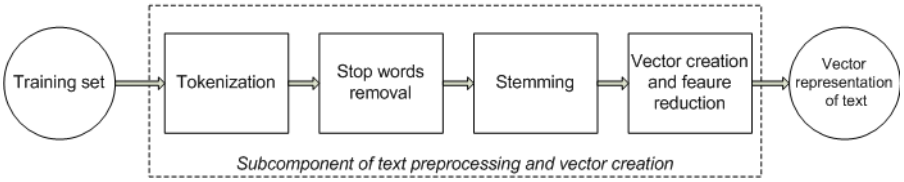
Informace v textu přirozeného jazyka jsou nestrukturované a obvykle v nevhodné formě pro strojové zpracování a extrakci netriviálních znalostí. V textu se mohou vyskytovat slova nepřinášející žádnou informaci, naopak slova přinášející stejnou informaci se mohou vyskytovat v různých tvarech atd.

Předzpracování textu tedy hraje důležitou roli při vytěžování znalostí z textu a je první činností při tomto procesu [10]. Základním krokem bývá rozdělení textu na jednotlivá slova – tokeny (tokenizace) a odstranění interpunkce, čísel a slov kratších než n (běžně $n = 2$). Obvykle pak následuje několik dalších kroků(1) – eliminace tzv. stop words, stematizace (stemming), lemmatizace, výpočet tf-idf. Schéma celého postupu ilustruje obrázek 3.1.

3.1.1 Stop words

Stop words jsou slova, která se v přirozeném jazyce vyskytují, ale nijak nepřispívají k významu textu. Jedná se především o zájmena, předložky, eventuálně členy (a, an, the) v anglických či jiných textech. Dalším důležitým přínosem odstranění těchto slov je redukce dimenzionality prostoru termů [11].

Nejjednodušší metodou odstranění stop words je odstranění všech slov, které se vyskytují v předem dané množině. Dále lze odstranit slova, která se vyskytují ve vysokém procentu dokumentů aktuálního korpusu, slova, která



Obrázek 3.1: Schéma obvyklého postupu při předzpracování textu. [4].

se vyskytují pouze v jednom dokumentu, takzvané sigletony, a slova s nízkým tf-idf (dále) [12].

3.1.2 Stematizace

Stematizace (stemming) je metoda identifikace kořene (stem) slova a nahrazení slova tímto kořenem. Všechna slova jako „connect“, „connected“, „connecting“, nebo „connections“ by tedy po aplikaci stematizace byla nahrazena řetězcem „connect“. Účelem je odstranit různé koncovky slov, tím redukovat počet termů – dimenzionalitu, paměťové nároky a čas potřebný k výpočtu. Přínos stematizace může být různý podle jazyka, v jakém je danný text napsán. Pro jazyky s komplexnější morfologií je přínos větší a provedení stematizace důležitější, než u jazyků s jednodušší morfologií [10].

Základní metodou je odstranění předpon a přípon slov. Nejjednodušší realizací je tzv. Truncate(n) stemmer, který ponechá prvních n písmen každého slova a odstraní zbytek a slova kratší než n ponechá v původním stavu. Se snižujícím se n se samozřejmě zvyšuje riziko přílišné ztráty informace. Dalším možností je S-stemmer [13], který podle předem danných pravidel sloučí singuláry a plurály podstatných jmen.

Pokročilejší metody založené na odstranění předpon a přípon jsou obvykle založené na jednom či více průchodu celým textem a postupné aplikaci množiny pravidel pro odstranění, nebo nahrazení přípony. V jednom z nejpoužívanějších algoritmů – Porters Stemmer [11] jsou pravidla tvořena tímto způsobem:

<podmínka><předpona> → **<nová předpona>**

Konkrétní pravidlo (m > 0) EED → EE má potom význam „Pokud slovo má alespoň jednu samohlásku a souhlásku a končí písmeny EED, změň konec slova na EE“. Při předzpracování anglického textu by se tedy například po aplikaci tohoto pravidla stalo ze slova „agreed“ slovo „agree“.

Statistické metody jsou pokročilejší metody stematizace, které k identifikaci a odstranění předpon a přípon využívají statistickou analýzu. Příkladem tohoto přístupu může být takzvaný N-Gram Stemmer. Tento algoritmus nejdříve rozdělí všechna po sobě jdoucí písmena slova do ntic a následně zkoumá statistický výskyt těchto n-tic [13]. Slovo INTRODUCTION by tak pro n=2 bylo rozděleno na tyto dvojice:

I, IN, NT, TR, RO, OD, DU, UC, CT, TI, IO, ON, NS, S

Nebo pro n=3 na tyto trojice:

I, *IN, INT, NTR, TRO, ROD, ODU, DUC, UCT, CTI, TIO, ION, ONS, NS*, S

Znak '*' je použit pouze pro dorovnání počtu znaků n-tice do n. Po rozdělení celého dokumentu na tyto n-tice je provedena analýza jejich výskytu.

Je jasné, že kořen slova se vyskytuje méně často, než přípony a předpony. Výhodou tohoto přístupu je, že je nezávislý na jazyku, velkou nevýhodou však zůstává vysoká paměťová náročnost, a proto je tento přístup v praxi těžko použitelný.

Většina klasických přístupů ke stematizaci má několik nevýhod. Je možné, že slova s naprosto odlišným významem, například anglické *policy* a *police* budou pravděpodobně změněna na stejný základ, zatímco *index* a *indices* by podle přístupu mohly, či nemusely být převedeny na stejný kořen.

3.1.3 Lematizace

Výše zmíněné nedostatky stematizace se snaží překonat lemmatizace (jako lemma je označován základní tvar slova). Stemming pracuje bez kontextu, lemmatizace se snaží postihnout kontext slova ve větě a slovní druh. Slovo *meeting* může být buďto základem samo o sobě, pokud se jedná o podstatné jméno, nebo je jeho základem *meet*, pokud se jedná o sloveso. Tento rozdíl není ve stemmingu rozeznatelný.

Dalším rozdílem těchto metod je, že stemming produkuje „slova“, která nejsou v běžném jazyce gramaticky korektní, zatímco lemmatizace nikoliv.

3.1.4 Tf-idf

Term Frequency – Inverse Document Frequency (zkráceně tedy tf-idf) je číslo, které vyjadřuje důležitost slova (termu) v daném dokumentu. Při vytěžování znalostí z textu je obvykle chápáno jako váha. Intuitivně lze říci, že čím se slovo v dokumentu vyskytuje častěji, tím je důležitější, a naopak, pokud se slovo vyskytuje ve velkém počtu dokumentů, jeho informační přínos klesá. Přesně toto uvažování se tf-idf snaží vystihnout.

Tf-idf je součinem dvou statistických údajů:

Term frequency

$tf_{i,j} \coloneqq \log(1 + f_{i,j})$

Kde $f_{i,j}$ je frekvence výskytu slova t_i v dokumentu d_j [14].

Inverse document frequency

$idf_i \coloneqq \log \frac{|D|}{|\{j : t_i \in d_j\}|}$

Kde $|D|$ je počet dokumentů v korpusu a $|\{j : t_i \in d_j\}|$ je počet dokumentů, které obsahují slovo i [14]. Tato složka reprezentuje právě „důležitost“ slova. Pro slova, která se vyskytují ve velkém počtu dokumentů, bude její hodnota nízká a pokud se slovo vyskytuje ve všech dokumentech jako například anglické členy *a* a *the*, bude pak dokonce nulová (konkrétně členy by už pravděpodobně byly odstraněny jako stop words, zde jen pro ilustraci). [10]

Vzorce pro výpočet obou složek nejsou ustálené a v literatuře lze nalézt lehce odlišné verze, které se liší převážně způsobem normalizace jednotlivých koeficientů. Pro Term Frequency lze použít takzvanou 0.5 normalizaci [10], kterou lze zobecnit pro jakoukoliv hodnotu [15]

Po výpočtu tf-idf vzniká matice, obvykle řídká, jejíž řádky reprezentují dokumenty a sloupce jednotlivé termy. Hodnoty jsou pak tf-idf skóre pro konkrétní term v konkrétním dokumentu. Pro další zpracování se pak obvykle využívá už jen tato matice, nebo kombinace obojího – matice a původní textový dokument. Celý řádek matice je pak takzvaná *vektorová reprezentace dokumentu*.

3.2 Klasifikace textových dokumentů a posilované učení

Běžné klasifikátory považují textový dokument za množinu slov, ve které se zcela ztrácí informace o pořadí vět a slov. To ovšem předpokládá, že informace je v dokumentu rozložena homogenně a že celý dokument je znám v čase klasifikace. Je ovšem zřejmé, že v lidmi psaném textu existují relevantní a irelevantní pasáže. Získávání dalšího textu navíc může být asociováno s vyššími náklady. Vyvstává tedy otázka, jestli je možné tyto nedostatky překonat? Snaží se to například model sekvenčního čtení [7]. V něm se *agent* učí číst dokument po řádcích, či větách a rozhodovat se, zda-li už má dostatek informací pro to, aby mohl klasifikovat dokument do kategorie.

3.2.1 V praxi často používané klasifikační metody

Pro ohodnocení výsledků jakéhokoliv nového klasifikátoru je třeba znát výsledky v praxi často používaných klasifikátorů:

K-Nejbližších sousedů (K-nearest neighbours, K-NN) je metoda, která v trénovací fázi pouze uloží trénovací množinu dat a ve vybavovací fázi najde k vzorků z trénovací množiny a klasifikuje testovaný vzorek majoritní třídou. Výsledky této metody mohou být silně ovlivněny dvěma faktory, volbou vhodné hodnoty k a způsobem výpočtu vzdálenosti jednotlivých vzorků. Často používané způsoby jsou například eukleidovská vzdálenost, manhattanská vzdálenost ($M(P,Q) = \sum_{i=1}^{n} |p_i - q_i|$) nebo cosinová podobnost [16]. Ta byla použita v tomto případě a je definována vzorcem [17]:

$$\cos(\theta) := \frac{A \cdot B}{||A|| \cdot ||B||} = \frac{\sum_{i=1}^{n} A_i B_i}{\sqrt{\sum_{i=1}^{n} A_i^2} \cdot \sqrt{\sum_{i=1}^{n} B_i^2}}$$

kde A a B jsou vektory a A_i a B_i jsou jejich složky.

Bayesovská klasifikace poskytuje zlatý standard, proti kterému lze porovnávat ostatní klasifikátory. Je založena na Bayesově větě a je statistickou metodou klasifikace a umožňuje vyjádřit jistotu, s jakou byla data správně oklasifikována. Bayesova věta, kterou formuloval Thomas Bayes, je popsána vzorcem [18]:

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

kdy $P(A|B)$ nazýváme *posteriorní pravděpodobnost* – pravděpodobnost hypotézy A potom, co vidíme B, $P(B|A)$ nazýváme *věrohodnost* – pravděpodobnost dat z B, pokud A je pravda, $P(A)$ nazýváme *apriorní pravděpodobnost* – pravděpodobnost hypotézy A před tím, než vidíme data a $P(B)$ nazýváme *normalizační konstanta* – pravděpodobnost dat B.

Ve vybavovací fázi pak hledáme maximálně pravděpodobný jev A na základě trénovacích dat B. Při klasifikaci je jevem A obvykle příslušnost ke kategorii a jevem B jakékoliv informace o daném vzorku.

Support vector machines je klasifikační metoda, která konstruuje množinu nadrovin takových, že mají co nejvyšší vzdálenost od bodů reprezentujících trénovací data jakékoliv třídy. [19]

TruncatedSVD provádí Latentní sémantickou analýzu – další analýzu textu založenou na distribuci jednotlivých termů. Zde je použita kvůli redukci dimenzionality. PCA není možné v tomto případě použít, jelikož nedokáže pracovat s řídkými maticemi, které jsou používány pro reprezentaci tf-idf hodnot. [20]

3.2.2 Definice klasifikace textu jako problému posilovaného učení

Zasadíme tedy problematiku do kontextu posilovaného učení, má agent při použití korpusu o n kategoriích k dispozici $n+1$ akcí – čti další větu a klasifikuj jako kategorii 1, ..., n. Volba další akce je plně závislá na *stavu*, ve kterém se agent aktuálně nachází, a je řízená *strategií* agenta π. Stav je funkce tf-idf vektoru právě čtené části textu a strategie mapuje akce na stavy v závislosti na jejich skóre – $Q(s, a)$ a volí tu akci s nejvyšší hodnotou. Jedná se tedy o hladovou strategii[7]:

$$a^* = \underset{a}{\operatorname{argmax}}\, Q(s, a)$$

kde a^* je nejlepší akce.

Zbývá vyřešit problém aproximace funkce $Q(s, a)$. Výše zmíněné algoritmy Sarsa a q-learning jsou pro řešení tohoto problému nevhodné, jelikož stavový prostor reprezentovaný pomocí tf-idf skóre je ze své podstaty spojitý a Sarsa, potažmo q-learning se potřebují setkat se všemi stavy.

Funkci Q je tedy nutné parametrizovat pomocí vektoru parametrů θ. Parametrizovaná funkce $Q_\theta(s, a)$ je pak vyjádřena vztahem:

$$Q_\theta(s, a) = \langle \theta, \Phi(s, a) \rangle$$

kde $\langle ., . \rangle$ značí vektorový součin a $\Phi(s, a)$ je vektorová reprezentace páru stav-akce. $\Phi(s)$ je tf-idf reprezentace právě čteného řádku/věty, nebo průměr doposud přečtených vět/řádků a $\Phi(s, a)$ je projekce $\Phi(s)$ do vícedimenzionálního prostoru v závislosti na a. Konkrétně $\Phi(s, a) = (0...\Phi(s)...0)$. Problém učení agenta je tedy redukován na nalezení optimálního vektoru parametrů θ^*, který vyústí v optimální strategii dosahující nejvyšší kumulativní odměny na trénovacích datech. Odměna je definována jako 1, pro akci klasifikace do správné kategorie a 0 jinak.

Je popsáno velké množství různých algoritmů pro nalezení optimálního vektoru parametrů. V praktické části práce bude popsána implementace algoritmů Cross Entropy Method [21], Reinforce [22] a jimi dosažené výsledky.

3.3 Sumarizace textu

S růstem internetu v posledních desetiletích dramaticky přibývá textových dat z nejrůznějších zdrojů, je proto třeba najít způsoby, jak se v této záplavě informací orientovat. Jednou z cest může být tvorba souhrnů textu neboli jeho sumarizace. Vytvořit výtahy z byť i malého zlomku textů, které Internet nabízí samozřejmě není v lidských silách, proto je třeba hledat způsoby, jak text sumarizovat automaticky.

Automatickou sumarizaci textu lze definovat jako tvorbu stručného shrnutí původního textu při zachování klíčových informací a celkového vyznění. To je z hlediska automatizace netriviální úkol, jelikož počítače nedosahují lidských schopností práce s jazykem. [23]

Přístupy k sumarizaci lze rozdělit do dvou základních skupin, *extraktivní* metody a *abstraktivní* metody. *Extraktivní* přístupy se snaží detekovat důležité věty nebo sekvence v textu a z těch sestavit koherentní souhrn, zatímco *abstraktivní* sumarizace se snaží vytvořit zcela nový text na základě textu původního. Přestože souhrny vytvořené lidmi jsou spíše abstraktivního charakteru, výzkum v oblasti automatické sumarizace se soustředí spíše na metody extraktivní. Generování zcela nového koherentního textu, který navíc musí obsahovat kritické informace, je totiž velmi složitá úloha a čistě abstraktivní systém pro sumarizaci v dnešní době není znám. [23]

Extraktivní sumarizace se obvykle snaží vybrat podmnožinu vět z původního textu (jednoho, nebo více dokumentů) tak, aby v nich byly obsaženy všechny důležité informace. Tento proces lze rozdělit do tří relativně nezávislých úkolů. Na začátku je obvykle vytvořena reprezentace vět na základě nějakých jejich vlastností, jako například pozice věty, počet stop words, průměrné tf-idf skóre, cosinová vzdálenost tf-idf vectoru ke všem ostatním, délka

věty a další. V druhém kroku je každé větě na základě získaných indikátorů nějakým způsobem přiřazeno skóre. V tomto okamžiku jsou často využívány techniky strojového učení. Třetím krokem je pak výběr k nejdůležitějších vět. Nejjednodušším přístupem je hladová strategie podle skóre, na tento krok se ale lze také dívat jako na optimalizační problém, kdy je cílem maximalizovat soudržnost textu a minimalizovat nadbytečnost a opakování se informací. [23]

Dalším problémem je způsob hodnocení strojově vytvořených shrnutí. Spoléhat na zpětnou vazbu od lidí je samozřejmě pro strojové učení zcela nedostatečné, je proto třeba najít automatický způsob ohodnocení podobnosti strojového souhrnu se vzorovým. Nejpoužívanější metrikou pro automatické hodnocení je ROUGE (Recall-Oriented Understudy for Gisting Evaluation).

Metrika ROUGE má několik modifikací, mezi nejpoužívanejnější patří:

ROUGE-*n*: Tato metrika je založena na počtu společných *n-gramů* testovaného a referenčního souhrnu. **N-gram** je sekvence n po sobě jdoucích slov v textu. Skóre je pak vypočítáno jako poměr všech společných n-gramů v obou souhrnech ku počtu n-gramů v referenčním souhrnu. [23]

ROUGE-SU SU v názvu této metriky znamená *skip bi-gram and uni-gram*. ROGUE-SU počítá společné 1- a 2-gramy s tím, že mezi slova 2-gramů je možné vložit další slova, nemusí se tedy jednat o po sobě jdoucí slova. [23]

3.4 Sumarizace textu jako problém posilovaného učení

Posilované učení pro sumarizaci textu poprvé využili Seonngi Ryang a Takeshi Abekawa v roce 2012 [24], jedná se tedy o velmi nový a nekonvenční přístup k tomuto problému. Bylo popsáno několik odlišných způsobů, jak modelovat stavy, akce a odměny. Stavem je obvykle nějaká množina vět a akcí je obvykle přidání či odebrání věty do této množiny [25]. Výsledné shrnutí textu je pak vytvořeno jako podmnožina vět z původního textu, což má několik výhod. Můžeme si být jistí, že výsledný text bude po syntaktické stránce správný (za předpokladu, že takový byl i původní text), víme, v jakém pořadí věty uspořádat pro čtenáře a nemusíme řešit složitý problém generování smysluplného textu od začátku.

Stav prostředí je možné konkrétně modelovat jako množinu vět, ze kterých se skládá dosavadní, agentem vytvořený souhrn textu, který je potenciálně nekompletní, akcemi je pak přidání jedné ze zbylých vět do souhrnu. Agentovi nemusí být v daném okamžiku dostupné všechny zbylé věty, jelikož na výsledný souhrn je obvykle kladeno omezení jeho délky. Množinu akcí lze pak zapsat následovně:

$$\mathcal{A}_s = \{c | c \in D \setminus S, length(c \cup S) \leq LC\}$$

kde S je pracovní souhrn (množina vět). [25]

Pro definování odměny je důležitý externí feedback. Agentem vytvořený souhrn je možné hodnotit na základě fixních kritérií kvality, jako lepší přístup se však jeví využít referenční souhrny vytvořené člověkem. Oproti klasifikaci textu to také umožňuje udělovat agentovi odměnu za každou akci a nečekat až na konec epizody. Odměna se pak odvíjí od podobnosti stavu – pracovního souhrnu a očekávaného výsledku – člověkem psaného shrnutí textu H_D. Jako metriku podobnosti dvou textů je možné použít výše zmíněnou metriku `ROUGE`, která je velmi často využívaná pro různé úlohy v oblasti NLP, není to však bezpodmínečně nutné.

Jednou z definic odměny, která podle literatury vedla k dobrým výsledkům, je nárůst skóre částečného souhrnu po přidání věty, formálně pak [25]:

$$r_{t+1} = score(s_{t+1}; H_D) - score(s_t; H_D)$$

Důležitým rozdílem oproti klasifikaci textu je, že v tomto případě je pro agenta prostředí plně pozorovatelné. Pokud agent zná všechny věty z dokumentu, a to musí, jelikož z nich vybírá, má také informaci o tom, jak bude vypadat další stav po přidání věty do pracovního souhrnu ještě dříve, než akci provede. To, že agent může pro své rozhodování využívat ne dvojice (s_t, a_t), ale i trojice (s_t, a_t, s_{t+1}), může být signifikantní výhodou. Experimenty s oběma přístupy budou provedeny v praktické části práce.

Samotná vektorová reprezentace dvojic (s_t, a_t), potažmo trojic (s_t, a_t, s_{t+1}), je také otevřenou otázkou. Víme jen to, že s_t a s_{t+1} jsou množiny vět a a_t je jediná věta. V článku [25] je zmíněno velké množství vlastností různých vět, které je možné vzít v potaz:

- Základní vlastnosti jako počet slov, počet stop words, absolutní a relativní pozice věty, počet slov začínajících velkým písmenem a další.
- Lingvistické IR (information retrieval) vlastnosti jako absolutní/relativní frekvence termů, tf-idf vektor, cosinová podobnost různých tf-idf vektorů, skóre čitelnosti [26] věty a další.
- Vlastnosti specifické pro RL jako nová délka pracovního souhrnu po přidání věty, cosinová podobnost zvažované věty a zdrojových dokumentů, minimální/maximální cosinová podobnost věty a vět už přidaných do souhrnu a další.

Dalším, jednodušším přístupem se jeví přistoupit k tvorbě vektoru $\Phi(s_t, a_t)$, nebo v tomto případě také $\Phi(s_t, a_t, s_{t+1})$, podobně jako v případě sumarizace, tedy spojit za sebe dva, potažmo tři tf-idf vektory. Prvním vektorem by byl průměr tf-idf vět v s_t, druhým tf-idf vektor věty a_t, třetím potenciálně průměr tf-idf vektorů vět obsažených v s_{t+1}. Vzhledem k tomu, že tf-idf v sobě obsahuje informaci o „relevanci“ jednotlivých slov v rámci korpusu a dokumentu, mohl by tento přístup být dostatečný.

Lze usuzovat, že důležitou roli v tom, který přístup dosáhne lepších výsledků a v tom, jak dlouho bude trvat učení agenta, bude hrát to, jaký korpus bude využíván. Pro delší, částečně odborné texty, jako například články z wikipedie, může být informace o pozici věty v článku a odstavci důležitější, než u kratších textů, jako například novinové články nebo uživatelské recenze, ve kterých odstavce ani nemusí existovat.

Pro učení agenta je pak možné trénovat jakýkoliv regresor, který bude na základě vektorové reprezentace stavu odhadovat hodnotu $Q(s_t, a_t)$, nebo použít algoritmy jako CEM nebo Reinforce, stejně jako při klasifikaci dokumentů. Při použití regresoru je pak nejefektivnější výše zmíněný algoritmus q-learning, kdy regresor odhaduje takzvaný *q-target* $R_t = r_t + \max_{a \in \mathcal{A}_s} Q(s_{t+1}, a)$.

Kapitola 4

Realizace

Vše bylo realizováno v jazyku Python 3.5 a pomocí k němu dostupných knihoven. Všechny zdrojové kódy je možné najít v příloze.

4.1 Klasifikace dokumentů

Před vlastním učením klasifikátoru vždy proběhlo předzpracování textu, při kterém byly využity knihovny Natural Language Toolkit (NLTK) [27] a Scikit-learn [28]. Předzpracování probíhalo následovně:

- rozdělení textu na slova (tokeny),
- odstranění stop words,
- test shody tokenu s regulárním výrazem,
- stemming,
- výpočet tf-idf a převod na vektorovu reprezentaci dokumentu.

Knihovna NLTK má funkce pro tokenizaci textu. Nabízí také množinu stop words, jejich odstranění tedy probíhalo testem na přítomnost každého tokenu v množině. Každý token byl dále testován na shodu s regulárním výrazem '[a-zA-Z]{3,}', zajímají nás tedy jen slova složená ze třech a více písmen. Tento krok nás zbaví hlavně „tokenů“ v podobě interpunkčních znamének a číslic. Je možné namítat, že čísla z určitého úhlu pohledu výrazně přispívají k informaci obsažené v textu. Jejich příspěvek k informaci o kategorii textu je však značně omezený, proto je užitečnější jejich odstraněním snížit výslednou dimenzi vektorové reprezentace dokumentu.

Pro stemming byla opět využita knihovna NLTK a jí implementovaný `PorterStemmer`. Takto byl postupně zpracován text ze všech dokumentů trénovací množiny a poté bylo spočítáno jejich tf-idf skóre s pomocí třídy `TfidfVectorizer` z knihovny Scikit-learn.

4.1.1 Použití a výsledky obvyklých klasifikátorů

Pro experimenty s některými v praxi často používanými klasifikátory byla využita knihovna scikit-learn [28], její implementace příslušných klasifikátorů a korpus 20 Newsgroups, který obsahuje přibližně 18700 dokumentů rozdělených do 20 kategorií [29]. K ohodnocení úspěšnosti byla použita křížová validace s $N = 5$. Jako metrika úspěšnosti bylo použito F1-score. Tabulka 4.1 uvádí výsledky experimentů. Všechny klasifikátory mají na vstupu buď tf-idf matici, nebo tf-idf matici dále zpracovanou pomocí SVD. Parametry pro tf-idf a SVD uvádím spolu s parametry vlastního modelu. Údaje ve sloupci „Čas. náročnost“ jsou ve formátu trénovací fáze/testovací fáze.

Nalezení nejlepší množiny parametrů proběhlo pomocí tříd `Pipeline` a `GridSearchCV`. Instanci třídy `Pipeline` je možné zadat sekvenci tříd, které chceme využít pro předzpracování a klasifikaci. Například tedy dvojici `TfidfVectorizer`, `MultinomialNB` pro naivní bayesovskou klasifikaci, nebo trojici `TfidfVectorizer`, `TruncatedSVD`, `SVC` pro použití Support Vector Machines a redukci dimenzionality pomocí Singular Vector Decomposition. `GridSearchCV` pak pro `Pipeline` a množinu parametrů (python dictionary) vybere ty, se kterými dosahuje křížová validace nejlepších výsledků. Sama se postará o efektivní paralelizaci výpočtů.

Klasifikátor	Parametry	F1-score	Čas. náročnost
Naivní Multinomiální Bayes	alpha=0.025 tfidf.max_df=0.3	0.961	0.125 s. / 0.014 s.
Naivní gaussovský bayes + Truncated SVD	svd.n_components=80 tfidf.max_df=0.3	0.871	4.997 s. / 0.129 s.
K-NN	k=1	0.915	0.014 s. / 2.310 s.
Support vector machines	kernel="linear" tfidf.max_df=0.5	0.951	71.134 s. / 30.911 s..
Support vector machines + Truncated SVD	kernel="linear" tfidf.max_df=0.2 svd.n_components=900	0.941	192.560 s. / 51.496 s.

Tabulka 4.1: Výsledky experimentů s běžnými klasifikátory

Můžeme si všimnout, že ve všech případech má parametr max_df (max. document frequency) relativně nízké hodnoty. To dává smysl – slova vyskytující se ve všech dokumentech nám nic neřeknou, proto je lepší je vůbec neuvažovat.

4.1.2 Realizace a výsledky Cross Entropy Methods

Algoritmus CEM pro nalezení vektoru parametrů θ začíná vygenerováním populace náhodných vektorů. Jednotlivé složky vektorů jsou generovány ze standardního normálního rozdělení. Následně jsou všechny vektory ohodnoceny, v tomto případě byla použita celková odměna, kterou agent získal, když se řídil strategií π_{θ_i} parametrizovanou příslušným vektorem θ_i. Algoritmus pak pokračuje iterativně. Složky vektorů v další iteraci jsou generovány z normálního rozdělení, s parametry μ a σ. Pro i-tou složku vektoru je μ_i rovno průměru i-tých složek nejlepších vektorů z předchozí iterace a σ_i je rovna standardní normální odchylce i-tých složek těchto vektorů. Počet vektorů generovaných v jedné iteraci a to, jaký jejich zlomek je použit pro generování nových vektorů jsou parametry algoritmu, se kterými je možné experimentovat.

Při experimentech s tímto algoritmem se ovšem ukázalo, že takto generované vektory mají tendenci rychle (během přibližně 20 iterací) konvergovat, a strategie agenta se tudíž přestává měnit a potenciálně zlepšovat. V začátku ovšem algoritmus nabízel lineární F1-skóre v každé iteraci. Vyvstala tedy otázka, jak zabránit předčasné konvergenci, nebo ji alespoň oddálit.

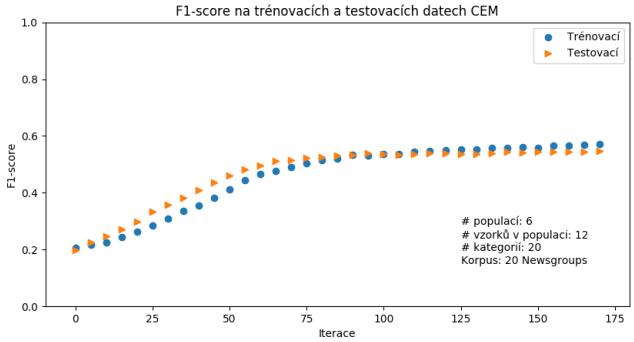
Jako řešením se ukázala být určitá „paralelizace“ CEM. Místo jedné populace vektorů θ bylo simultálně vylepšováno několik (3, 6) takových populací. Každou pátou iteraci byly pak parametry μ a σ vypočteny za pomoci nejlepších vektorů ne z jedné populace, ale za pomoci jednoho nebo více vektorů z každé populace, viz obrázek 1. Jako nejlepší zlomek byla brána třetina, resp. polovina, vektorů pro 3, resp. 6 paralelních populací. Na níže uvedených výsledcích a grafech je patrné, že toto řešení bylo částečně úspěšné, konvergenci se podařilo oddálit, ale ne jí úplně zabránit. Dále se jeví, že se zvyšujícím se počtem populací by se výsledky algoritmu dále zlepšovaly, tyto experimenty však vzhledem k výpočetní náročnosti nebylo možné provést na osobním notebooku se čtyřjádrovým procesorem, na kterém experimenty probíhaly.

K ohodnocení konkrétního vektoru θ by bylo velmi náročné použít celý korpus, bylo proto voleno 50 náhodných souborů z každé kategorie v trénovací množině. Trénovací a testovací data byla rozdělena v poměru 80:20 z každé kategorie. Výběr do skupin probíhal náhodně. Výsledky na testovacích datech byly měřeny s použitím všech souborů z testovací množiny.

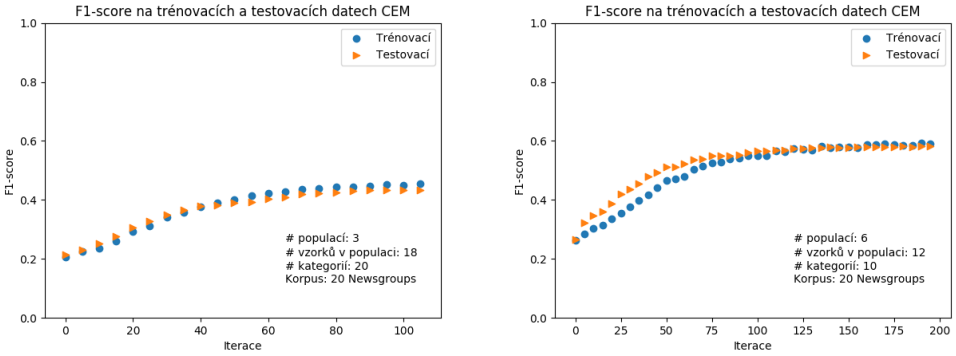
Celý algoritmus bylo dále vzhledem k výpočetní náročnosti nutné paralelizovat ve smylsu vláknování. Při výpočtech bylo spuštěno celkem 5 vláken – 1 hlavní a 4 pracovní. Hlavní vlákno přidávalo úkoly pro pracovní vlákna do globální fronty, ze které je pracovní vlákna postupně odebírala a řešila. Jako jeden úkol ve frontě bylo považováno použít konkrétní vektor θ pro klasifikaci souborů z jedné kategorie. Pokud byl využíván celý korpus, bylo tedy ohodnocení jednoho vektoru θ rozděleno do 20 úkolů. Všechny dále uváděné časy jsou časy z pohledu uživatele. Při sekvenčním běhu programu by bylo nutné očekávat časy přibližně 3x vyšší.

Grafy 4.1, 4.2 a 4.3 uvádí výsledky běhu programu pro různé konfigurace

a tabulka 4.2 další statistiky měřené na testovacích datech.



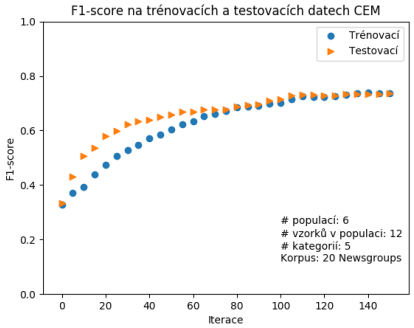
Obrázek 4.1: Průběh učení CEM.



Obrázek 4.2: Průběh učení CEM pro různé konfigurace.

Z výše zmíněných výsledků je možné pozorovat, že po dobu učení klasifikátoru nedocházelo k přeučení, výsledky na testovacích datech jsou stále srovnatelná s těmi na trénovacích datech, i když ta trénovací začínají v druhé polovině ta testovací lehce převyšovat. Vidíme, že rychlost růstu skóre je od určité iterace téměř nulová. Celkový počet populací má evidentně vliv na výsledné skóre klasifikátoru (obrázek 4.2). Pro velikosti populací 3 a 6 byl rozdíl přibližně 10%, což vede ke zmíněné úvaze, že se zvyšujícím se počtem populací by bylo možné výsledky dále zlepšit.

Při analýze toho, kolik vět agent přečetl (tabulka 4.2) se ukázalo, že agent ve většině případů nepokračoval ve čtení dokumentu a ihned jej klasifikoval (čtení první věty bylo bráno jako povinné). To se jeví být hlavním nedostatkem tohoto postupu, agent často nemohl mít dostatečné množství informací. I přesto však dosahoval až 8-krát lepších výsledků, než náhodný klasifikátor.



Obrázek 4.3: Průběh učení CEM.

Kategorií	Prům. poč. vět	Přečteno	Čas (s)
20	18.258	1.396	13.058
10	18.960	1.451	6.711
5	17.116	1.387	3.220

Tabulka 4.2: Statistiky testovacích dat CEM.

4.1.3 Realizace a výsledky algoritmu Reinforce

Algoritmus Reinforce se nesnaží vektorem θ aproximovat funkci $Q(s, a)$, ale přímo strategii π. Tato strategie musí být dále založena na funkci softmax. Pro všechny akce $a \in \mathcal{A}(s)$ je spočítána hodnota $\exp(\langle\phi(s, a), \theta\rangle)$. Na tyto hodnoty je dále aplikována funkce softmax a výsledky slouží jako pravděpodobnost volby akce a. Update vektoru θ je pak založen na gradientu strategie, který je díky použití funkce softmax lehce přístupný, a na míře učení (learning rate) α. Implementace algoritmu je založena na pseudokódu z [6, Lecture 7].

Při experimentech se tento algoritmus ukázal být jako úspěšnější než CEM, ikdyž i u něj přetrvával problém s tím, že se nedařilo naučit agenta číst další věty z dokumentu. Z tohoto důvodu bylo vyzkoušeno definovat odměnu za akci čtení další věty ne jako 0, ale jako $p^{\#\ přečtených\ vět}, 0 < p < 1$. V závisloti na hodnotě p však buďto agent začal číst všechny věty (pro vyšší hodnoty p), ale bez zvyšující se úspěšnosti klasifikace, nebo počet přečtených vět konvergoval k 1 (v tuto chvíli dosahoval algoritmus relativně vysoké úspěšnosti). Hranicí rozlišující toto chování se zdá být hodnota 0.5, pravděpodobně z tohoto důvodu, že součet geometrické řady je pak > 1.

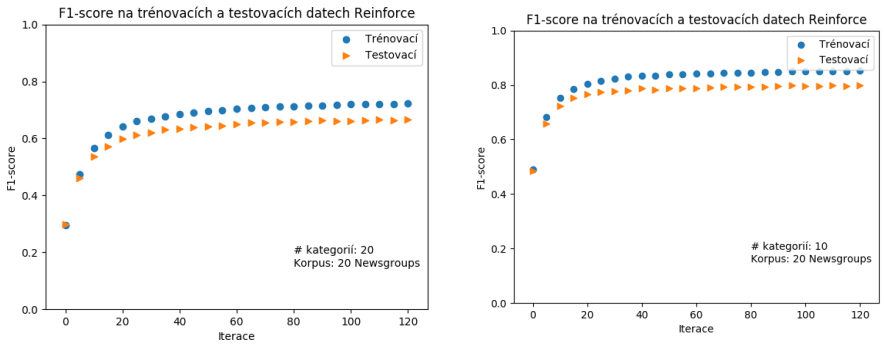
Obrázky 4.4 a 4.5 znázorňují výsledky v průběhu učení algoritmu Reinforce. Vidíme, že algoritmus signifikantně předčí CEM. Jeho výsledky sice nejsou srovnatelné s výsledky obvyklých klasifikátorů uvedenými výše, jsou ale jen o málo horší než ty publikované v [7].

4.1.4 Experimenty s korpusem Reuters 8

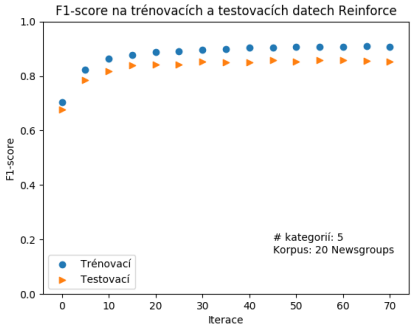
Další experimenty proběhly s algoritmem Reinforce, jelikož dosahoval nejlepších výsledků v předchozích experimentech. Jako korpus byl použit data set Reuters 8 [30], který obsahuje osm kategorií kratších dokumentů v podobě novinových zpráv. Data set Reuters 8 je zpřístupněný výše zmíněnou knihovnou NLTK. Jako benchmark byla zvolena třída `MultinomialNB` z knihovny Scikit-learn, taktéž proto, že tento klasifikátor dosahoval nejlepší úspěšnosti

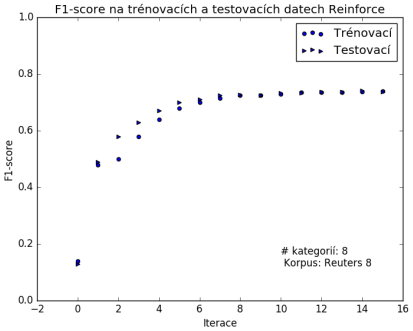
```
def cem(dim_th, n_pop, batch_size, elite_frac, n_iter):
  # Inicializace
  theta_mean = [np.zeros(dim_th) for j in range(n_pop)]
  theta_std = [np.ones(dim_th) for j in range(n_pop)]
  # Vlastni algoritmus
  for it in range(n_iter):
        # Generovani novych vzorku
    thetas = [np.random.normal(th_mean[j], th_std[j],
           (batch_size, dim_th)) for j in range(n_pop)]
    results = [[evaluate_theta(theta) for theta in thetas[j]]
                        for j in range(n_pop)]
    rewards = [[r[0] for r in results[j]] for j in range(n_pop)]
        # Ty nejlepsi vektory z~kazde populace
    n_elite = int(batch_size * elite_frac)
    elite_inds = [np.argsort(rewards[j])[batch_size -
                n_elite:batch_size] for j in range(n)]
    elite_th = [[thetas[j][i] for i in elite_inds[j]]
                                    for j in range(n_pop)]
    if it > 0 and it % 5 == 0:
    # Promichej elitni cleny populaci tak, aby v~nove populaci
    # Byl stejny pocet elitnich vektoru z~kazde puvodni populace
    th_mean=[np.mean(elite_th[j], axis=0) for j in range(n_pop)]
    th_std=[np.std(elite_th[j], axis=0) for j in range(n_pop)]
    rew_train = [eval_theta(theta_mean[j], test=True)[0]
                                  for j in range(n_pop)]
  return theta_mean[np.argmax(rew_train)]
```

Výpis kódu 1: Používaná implementace CEM



Obrázek 4.4: Průběh učení Reinforce.





Obrázek 4.5: Průběh učení Reinforce. Obrázek 4.6: Reinforce a Reuters 8.

v předchozích experimentech. V tomto případě dosáhl F1 skóre ve výši 0.886.

I v tomto případě se projevil nedostatek algoritmu pozorovaný u předchozích experimentů, tedy že počet vět přečtených agentem z každého dokumentu v průběhu učení konvergoval k 1. Průběh učení a dosažené F1 score je možné vidět na obrázku 4.6. I přes zmíněné nedostatky postupoval algoritmus velmi rychle a dosáhl relativně vysokého F1 score. V tomto případě byl pro výpočet F1 score použit vážený průměr, jelikož třídy byly v korpusu nerovnoměrně zastoupeny.

4.2 Sumarizace textu

V případě sumarizace textu neexistuje jednoznačný benchmark, vůči kterému porovnávat výsledky, jako byly K-NN, Naivní Bayes a další v případě klasifikace, bylo proto nutné implementovat vlastní jednoduchý algoritmus pro sumarizaci nazvaný „Top-n“. Problémem se ukázala být nedostupnost open-source dat pro sumarizaci, z tohoto důvodu probíhaly experimenty s korpusem [31] sestávajícím se z asi 4500 novinových článků a jejich referenčních shrnutí.

4.2.1 Top-n sumarizace

Jako nejjednodušší přístup pro sumarizaci se jeví přisoudit každému slovu v textu nějaké skóre, vybrat pak n slov s nejvyšším skóre a vybranou množinu slov považovat za shrnutí daného textu. Výsledek je pak možné pomocí metriky `ROUGE` porovnat s referencí. V celé práci byla využívána konkrétně metrika `ROUGE-1`. Takovýto přístup má samozřejmě několik nedostatků, dostaneme pouze „klíčová“ slova z textu, výsledkem tedy není ucelený a syntakticky správný text.

V tomto přístupu se nachází dvě proměnné, n a pak způsob výpočtu skóre. Pro určování skóre se jasně nabízí tf-idf, s hodnotou n je vhodné experimentovat. Data byla rozdělena na trénovací a testovací množinu v poměru 80:20,

tf-idf skóre pak bylo slovům z testovacích textů přisouzeno na základě statistiky spočtené z trénovacích dat. Tabulka 4.3 obsahuje výsledky experimentů s různými hodnotami n. Řádek `ROUGE-1` obsahuje průměrné skóre na testovacích datech. Je možné pozorovat, že s rostoucím n se skóre logicky zvyšuje, přibližně u hodnoty $n = 100$ ale skóre začíná stagnovat a dále se nezvyšuje.

`ROUGE-1` skóre uváděné jako dosažené výzkumníky v aktuálních článcích se pohybuje okolo hodnoty 0.2 [25], lze tedy usuzovat, že data set použitý v této práci je o něco jednodušší, než obvykle používané data sety, které jsou ale bohužel volně nepřístupné.

n	5	10	20	60	80	100	200	400
`ROUGE-1`	0.028	0.052	0.085	0.159	0.182	0.201	0.223	0.225

Tabulka 4.3: Výsledky Top-n sumarizace

4.2.2 Využití neuronových sítí a algoritmu q-learning

Jedním ze způsobů, jak využít posilované učení pro sumarizaci textu je pokusit se algoritmem q-learning aproximovat funkci $Q(s, a)$ výše zmíněným způsobem, tedy iterativním sbíráním vzorků prostředí a učením regresoru. Pro získání tf-idf vektoru vět je opět možné použít třídu `TfidfVectorizer` z knihovny Scikit-learn [28], pro vlastní regresi je pak v principu možné využít jakýkoliv regresor, vzhledem k dimenzionalitě prostoru a složitosti problému se neuronové sítě jeví jako vhodný nástroj.

Je možné použít neuronovou síť připravenou pro regresi v podobě třídy `MLPRegressor` z knihovny Scikit-learn, nebo sestavit vlastní neuronovou síť za pomoci jiných knihoven, jako například Keras [32]. Knihovna Keras poskytuje vysokoúrovňové API pro práci s neuronovými sítěmi, a proto je vhodná pro použití v tomto případě. Konkrétně byla neuronová síť sestavena ze dvou skrytých vrstev, první z nich dopředná se 100 neurony a aktivační funkcí $ReLU$ definovanou vztahem [33]:

$$R(z) = \max(0, z),$$

další rekurentní (konkrétně LSTM) s 10 neurony, a jedné výstupní s jedním neuronem, jehož výstup je považován za predikci hodnoty. V případě `MLPRegressor` se jednalo o dopřednou neuronovou síť s 200 neurony v jedné skryté vrstvě a aktivační funkcí $ReLU$.

Experimenty probíhaly s celkovým počtem vzorků mezi doučováním regresoru, vektorovou reprezentací vzorků (tedy tím, jestli uvažovat dvojice (s_t, a_t), nebo trojice (s_t, a_t, a_{t+1})) a s dvěma výše zmíněnými regresory. Tabulka 4.4 uvádí výsledky experimentů. Údaj LSTM ve sloupci „Regresor“ značí, že byla využita neuronová síť implementovaná pomocí knihovny Keras. Sloupec `ROUGE`

uvádí průměrné ROUGE skóre na testovacích datech. Dalším, fixním parametrem algoritmu byla maximální délka výsledného shrnutí textu, která byla nastavena 0.2násobek délky zdrojového textu.

V obou případech se postup učení shodoval s algoritmem q-learning, nejdříve je tedy regresor inicializován náhodnými hodnotami a poté byla pro každý vzorek prostředí zaznamenána hodnota R_t, takzvaný „q-target“, definovaný dle algoritmu q-learning takto:

$$R_t = r_t + max_a Q(s_{t+1}, a)$$

kdy $Q(s, a)$ je predikce regresoru, zpočátku zcela náhodná. Po dosažení požadovaného vzorků pak proběhlo doučení regresoru na tato nově pozorovaná data.

Z výsledků v tabulce 4.4 je možné pozorovat, že ve všech případech byl výsledek lepší než u Top-n sumarizace. Nedostatkem však po celou dobu učení zůstával jen malý nárůst ROUGE skóre od začátku učení s náhodnou konfigurací neuronové sítě.

Typ vzorku	Regresor	Vzorků v iteraci	ROUGE-1
(s_t, a_t)	MLPRegressor	500	0.276
		800	0.301
		1100	0.291
	LSTM	500	0.268
		800	0.241
		1100	0.329
(s_t, a_t, s_{t+1})	MLPRegressor	500	0.290
		800	0.317
		1100	0.328
	LSTM	500	0.273
		800	0.336
		1100	0.315

Tabulka 4.4: Výsledky sumarizace s použitím neuronových sítí

4.2.3 Algoritmus Reinforce pro sumarizaci

Aplikace algoritmu Reinforce pro sumarizaci probíhala velmi podobně, jako v případě klasifikace. Strategie agenta byla opět založena na funkci softmax – pro každý vzorek, resp. jeho vektorovou reprezentaci byl vypočten skalární součin $\Phi(s_t, a_t) \cdot \theta$, na vypočtené výsledky byla aplikována funkce softmax a výsledky považovány za pravděpodobnost volby jednotlivých akcí. Po proběhnutí určitého počtu epizod pak proběhl update vektoru θ na základě celkové odměny získané agentem v epizodě tak, jak jej popisuje algoritmus Reinforce [22].

V tabulce 4.5 jsou uvedeny výsledky algoritmu pro různý počet epizody mezi tím, než proběhl update vektoru θ. Epizodou je v tomto případě míněno vytvoření jednoho shrnutí textu. I v tomto případě vidíme, že výsledné ROUGE skóre není nízké, v průběhu učení se ale objevil stejný problém jako v případě využití ANN, tedy jen malý růst skóre v průběhu učení.

Typ vzorku	Epizod	ROUGE-1
(s_t, a_t)	100	0.276
	200	0.301
	300	0.291
(s_t, a_t, s_{t+1})	100	0.290
	200	0.317
	300	0.328

Tabulka 4.5: Výsledky sumarizace s použitím algoritmu Reinforce

Závěr

Cílem práce bylo prozkoumat přínos posilovaného učení a rekurentních neuronových sítí pro klasifikaci a sumarizaci textu a zjistit, jestli agent sekvenčně čtoucí dokument může být stejně přesný, ale rychlejší než Bayes. V případě sumarizace bylo cílem zjistit vhodnost použití posilovaného učení a jestli je pro rozhodování agenta přínosnější řídit se trojicí stav-akce-další stav, nebo obvykle užívanou dvojící stav-akce.

Bylo zjištěno, že je problematické naučit agenta sekvenčně číst dokument. Při experimentech vyplynulo, že se agent ve většině případů rozhodoval pouze na základě první věty, což mu nepřinášelo dostatek informací, a proto byl Bayes přesnější. Hypotéza, že takováto klasifikace bude rychlejší, se však nepotvrdila. I v případě čtení pouze jedné věty byl agent pomalejší než naivní Bayesovská klasifikace.

U sumarizace textu se rozdíl mezi dvojicí stav-akce (s_t, a_t) a trojicí stav-akce-násl. stav (s_t, a_t, s_{t+1}) nepotvrdil. Ukázalo se, že dvojice (s_t, a_t) ani trojice (s_t, a_t, s_{t+1}) reprezentované pouze svými tf-idf vektory agentovi pravděpodobně nepřináší dostatek informací a učení agenta proto bylo problematické.

V budoucnu by bylo možné zaměřit se hlavně na učení agenta v případě klasifikace a najít učící algoritmus a nastavení odměn takové, že agent naučí přečíst před rozhodnutím přiměřenou část dokumentu.

Literatura

[1] Beginner's guide to Reinforcement Learning & it's implementation in python. https://www.analyticsvidhya.com/blog/2017/01/introduction-to-reinforcement-learning-implementation/, navštíveno 19-03-2018.

[2] Monte-Carlo Prediction. https://dnddnjs.gitbooks.io/rl/content/mc_prediction.html, navštíveno 19-03-2018.

[3] Temporal-Difference Learning – Simulation | ML. https://jay.tech.blog/2016/12/28/temporal-difference-learning/, navštíveno 19-03-2018.

[4] Schema of the structure of the subcomponent of text preprocessing. https://www.researchgate.net/figure/Schema-of-the-structure-of-the-subcomponent-of-text-preprocessing-and-vector-creation_fig2_281934085, navštíveno 19-03-2018.

[5] Richard S. Sutton, A. G. B.: *Reinforcement Learning: An Introdouction*. The MIT Press, 1998, ISBN 0-262-19398-1.

[6] Teaching. http://www0.cs.ucl.ac.uk/staff/d.silver/web/Teaching.html, přednášky Davida Silvera, navštíveno: 07-03-2018.

[7] Dulac-Arnold, G.; Denoyer, L.; Gallinari, P.: Text Classification: A Sequential Reading Approach. *CoRR*, ročník abs/1107.1322, 2011, 1107.1322. Dostupné z: http://arxiv.org/abs/1107.1322

[8] Neural Networks and Deep Learning. http://neuralnetworksanddeeplearning.com/index.html, navštíveno 21-04-2018.

[9] Recurrent Neural Networks Tutorial. http://www.wildml.com/2015/09/recurrent-neural-networks-tutorial-part-1-introduction-to-rnns/, navštíveno 21-04-2018.

[10] Vijayarani, D. S.: Preprocessing Techniques for Text Mining - An Overview. *International Journal of Computer Science & Communication Networks*, ročník 5, č. 1, 2009: s. 7–16, ISSN 2249-5789.

[11] Porter, M.: An Algorithm for Suffix Stripping. *Program*, ročník 14, 1980: s. 130–137.

[12] Jivani, M. A. G.: A Comparative Study of Stemming Algorithms. *Int. J. Comp. Tech. Appl*, ročník 2, č. 6, 2010: s. 1930–1938, ISSN 2229-6093.

[13] Sharma, D.: Stemming Algorithms, A Comparative Study and their Analysis. *International Journal of Applied Information Systems (IJAIS)*, ročník 4, č. 3, 2012, ISSN 2249-0868.

[14] TFIDF Statistics | SAX-VSM. https://jmotif.github.io/sax-vsm_site/morea/algorithm/TFIDF.html, navštíveno: 18-02-2018.

[15] Maximum tf normalization. https://nlp.stanford.edu/IR-book/html/htmledition/maximum-tf-normalization-1.html, navštíveno: 18-02-2018.

[16] 1.6. Nearest Neighbours - scikit-learn 0.19.1 documentation. http://scikit-learn.org/stable/modules/neighbors.html, navštíveno: 07-03-2018.

[17] Machine Learning::Cosine similarity for Vector Space Models (Part III). http://blog.christianperone.com/2013/09/machine-learning-cosine-similarity-for-vector-space-models-part-iii/, navštíveno: 07-03-2018.

[18] 1.9. Naive Bayes - scikit-learn 0.19.1 documentation. http://scikit-learn.org/stable/modules/naive_bayes.html, navštíveno: 07-03-2018.

[19] 1.4. Support Vector Machines - scikit-learn 0.19.1 documentation. http://scikit-learn.org/stable/modules/svm.html#svm-kernels, navštíveno: 05-03-2018.

[20] sklearn.decomposition.TruncatedSVD - scikit-learn 0.19.1 documentation. http://scikit-learn.org/stable/modules/generated/sklearn.decomposition.TruncatedSVD.html, navštíveno: 05-03-2018.

[21] De Boer, P.-T.; Kroese, D. P.; Mannor, S.; aj.: A Tutorial on the Cross-Entropy Method. *Annals of Operations Research*, ročník 134, 2005: s. 19–67.

[22] Williams, R. J.: Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine Learning*, ročník 8, č. 3, May 1992: s. 229–256, ISSN 1573-0565, doi:10.1007/BF00992696. Dostupné z: https://doi.org/10.1007/BF00992696

[23] Allahyari, M.; Pouriyeh, S. A.; Assefi, M.; aj.: Text Summarization Techniques: A Brief Survey. *CoRR*, ročník abs/1707.02268, 2017, 1707.02268. Dostupné z: http://arxiv.org/abs/1707.02268

[24] Ryang, S.; Abekawa, T.: Framework of Automatic Text Summarization Using Reinforcement Learning. In *Proceedings of the 2012 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning*, Association for Computational Linguistics, 2012, s. 256–265. Dostupné z: http://dl.acm.org/citation.cfm?id=2390948.2390980

[25] Henß, S.; Mieskes, M.; Gurevych, I.: A Reinforcement Learning Approach for Adaptive Single- and Multi-Document Summarization. In *GSCL*, 2015.

[26] Free Flesh-Kiclaid Readability Test Tool. https://www.webpagefx.com/tools/read-able/flesch-kincaid.html, navštíveno 21-04-2018.

[27] Natural Language Toolkit - NLTK 3.2.5 Documentation. https://www.nltk.org/, navštíveno: 07-03-2018.

[28] Pedregosa, F.; Varoquaux, G.; Gramfort, A.; aj.: Scikit-learn: Machine Learning in Python. *Journal of Machine Learning Research*, ročník 12, 2011: s. 2825–2830.

[29] Home Page for 20 Newsgroups Data Set. http://qwone.com/~jason/20Newsgroups/, navštíveno: 04-03-2018.

[30] Reuters-21578 Text Categorization Collection. http://kdd.ics.uci.edu/databases/reuters21578/reuters21578.html, navštíveno 21-04-2018.

[31] NEWS SUMMARY | Kaggle. https://www.kaggle.com/sunnysai12345/news-summary, navštíveno 21-04-2018.

[32] Chollet, F.; aj.: Keras. https://keras.io, 2015.

[33] Activation Functions: Neural Networks - Towards Data Science. https://towardsdatascience.com/activation-functions-neural-networks-1cbd9f8d91d6, navštíveno 21-04-2018.

Příloha **A**

Seznam použitých zkratek

API Application interface

CEM Cross Entropy Method

ReLU Rectified Linear Unit

NLTK Natural Language Toolkit

Sarsa State, action, reward, state, action

SVM Support Vector Machines

SVD Singular Value Decomposition

Tf-idf Term frequency – inverse document frequency

Příloha **B**

Obsah přiloženého CD

- `readme.txt` ..... stručný popis obsahu CD
- `src` ..... adresář se zdrojovým kódem
  - `impl` ..... zdrojové kódy implementace (spustitelné)
    - `Classification` ..... kód použitý pro klasifikaci textu
    - `Summarization` ..... kód použitý pro sumarizaci textu
  - `thesis` ..... zdrojová forma práce ve formátu LaTeX
  - `images` ..... obrázky obsažené v práci
- `text` ..... text práce
  - `thesis.pdf` ..... text práce ve formátu PDF
  - `thesis.ps` ..... text práce ve formátu PS