Analýza vlivu reprezentace textu na kvalitu klasifikace pomocí metod umělé inteligence

Mikhail Larkin

Diplomová práce
2023

Univerzita Tomáše Bati ve Zlíně
Fakulta aplikované informatiky

Univerzita Tomáše Bati ve Zlíně
Fakulta aplikované informatiky
Ústav informatiky a umělé inteligence

Akademický rok: 2022/2023

ZADÁNÍ DIPLOMOVÉ PRÁCE

(projektu, uměleckého díla, uměleckého výkonu)

Jméno a příjmení:	**Mikhail Larkin**
Osobní číslo:	**A21233**
Studijní program:	**N0613A140022 Informační technologie**
Specializace:	**Softwarové inženýrství**
Forma studia:	**Prezenční**
Téma práce:	**Analýza vlivu reprezentace textu na kvalitu klasifikace pomocí metod umělé inteligence**
Téma práce anglicky:	**Analysis of Text Representation Influence on the Quality of Classification by Artificial Intelligence Methods**

Zásady pro vypracování

1. Zpracujte literární rešerši na dané téma.
2. Vytvořte datový korpus pro klasifikaci textu.
3. Experimentálně ověřte vhodnost aktuálně využívaných technik reprezentace textu.
4. Proveďte statistické vyhodnocení výsledků.
5. Specifikujte doporučení pro řešení praktických úloh z dané oblasti.

Forma zpracování diplomové práce: **tištěná/elektronická**

Seznam doporučené literatury:

1. AGGARWAL, Charu C. Data Mining: The Textbook. 2015th Edition. Imprint: Springer, 2015. ISBN 978-3319141411.
2. SONBOL, Riad, Ghaida REBDAWI a Nada GHNEIM. The Use of NLP-Based Text Representation Techniques to Support Requirement Engineering Tasks: A Systematic Mapping Review. In: IEEE Access. 2022, s. 62811-62830. ISSN 2169--3536. Dostupné z: doi:10.1109/ACCESS.2022.3182372.
3. ZELLE, John M. Python programming: an introduction to computer science. Third edition. Wilsonville: Franklin, Beedle & Associates, [2017], xv, 536 s. ISBN 978-1-59028-275-5.
4. OSMAN, Ahmed Hamza a Omar Mohammed BARUKUB. Graph-Based Text Representation and Matching: A Review of the State of the Art and Future Challenges. IEEE Access. 2020, 8, 87562-87583. ISSN 2169-3536. Dostupné z: doi:10.1109/ACCESS.2020.2993191.
5. BABIĆ, Karlo, Sanda MARTINČIĆ-IPŠIĆ a Ana MEŠTROVIĆ. Survey of Neural Text Representation Models. Information [online]. 2020, 11(11), 511. ISSN 2078-2489. Dostupné z: doi:10.3390/info11110511.

Vedoucí diplomové práce: **Ing. Adam Viktorin, Ph.D.**
Ústav informatiky a umělé inteligence

Datum zadání diplomové práce: **2. prosince 2022**
Termín odevzdání diplomové práce: **26. května 2023**

UNIVERZITA TOMÁŠE BATI VE ZLÍNĚ
Fakulta aplikované informatiky

L.S.

doc. Ing. Jiří Vojtěšek, Ph.D. v.r.
děkan

prof. Mgr. Roman Jašek, Ph.D., DBA v.r.
ředitel ústavu

Ve Zlíně dne 7. prosince 2022

Prohlašuji, že

- beru na vědomí, že odevzdáním diplomové práce souhlasím se zveřejněním své práce podle zákona č. 111/1998 Sb. o vysokých školách a o změně a doplnění dalších zákonů (zákon o vysokých školách), ve znění pozdějších právních předpisů, bez ohledu na výsledek obhajoby;
- beru na vědomí, že diplomová práce bude uložena v elektronické podobě v univerzitním informačním systému dostupná k prezenčnímu nahlédnutí, že jeden výtisk diplomové práce bude uložen v příruční knihovně Fakulty aplikované informatiky Univerzity Tomáše Bati ve Zlíně;
- byl/a jsem seznámen/a s tím, že na moji diplomovou práci se plně vztahuje zákon č. 121/2000 Sb. o právu autorském, o právech souvisejících s právem autorským a o změně některých zákonů (autorský zákon) ve znění pozdějších právních předpisů, zejm. § 35 odst. 3;
- beru na vědomí, že podle § 60 odst. 1 autorského zákona má UTB ve Zlíně právo na uzavření licenční smlouvy o užití školního díla v rozsahu § 12 odst. 4 autorského zákona;
- beru na vědomí, že podle § 60 odst. 2 a 3 autorského zákona mohu užít své dílo – diplomovou práci nebo poskytnout licenci k jejímu využití jen připouští-li tak licenční smlouva uzavřená mezi mnou a Univerzitou Tomáše Bati ve Zlíně s tím, že vyrovnání případného přiměřeného příspěvku na úhradu nákladů, které byly Univerzitou Tomáše Bati ve Zlíně na vytvoření díla vynaloženy (až do jejich skutečné výše) bude rovněž předmětem této licenční smlouvy;
- beru na vědomí, že pokud bylo k vypracování diplomové práce využito softwaru poskytnutého Univerzitou Tomáše Bati ve Zlíně nebo jinými subjekty pouze ke studijním a výzkumným účelům (tedy pouze k nekomerčnímu využití), nelze výsledky diplomové práce využít ke komerčním účelům;
- beru na vědomí, že pokud je výstupem diplomové práce jakýkoliv softwarový produkt, považují se za součást práce rovněž i zdrojové kódy, popř. soubory, ze kterých se projekt skládá. Neodevzdání této součásti může být důvodem k neobhájení práce.

Prohlašuji,

- že jsem na diplomové práci pracoval samostatně a použitou literaturu jsem citoval. V případě publikace výsledků budu uveden jako spoluautor.
- že odevzdaná verze diplomové práce a verze elektronická nahraná do IS/STAG jsou totožné.

Ve Zlíně, dne 18. 05. 2023 Mikhail Larkin, v. r.

ABSTRAKT

Tato práce se zabývá analýzou vlivu metod reprezentace textu na kvalitu kvalifikace pomocí metod umělé inteligence. Cílem práce je porovnat populární metody reprezentace textu a posoudit jejich vliv na kvalitu klasifikace textu pomocí technik strojového a hlubokého učení. Byl analyzován vliv metod reprezentace textu, jako jsou Bag of Words, TF-IDF, Word2Vec, GloVe, FastText a Doc2Vec. Jako klasifikátory byly zvoleny Random Forest, Support Vector Machine, Multilayer Perceptron a Convolutional Neural Network.

Klíčová slova: Klasifikace textu, NLP, BoW, TF-IDF, Word2Vec, GloVe, FastText, Doc2Vec, CNN, MLP, Random Forest, SVM

ABSTRACT

This thesis deals with the analysis of the influence of text representation methods on the quality of qualification using artificial intelligence methods. The aim of this paper is to compare popular text representation methods and assess their impact on text classification quality using machine and deep learning techniques. The influence of text representation methods such as Bag of Words, TF-IDF, Word2Vec, GloVe, FastText and Doc2Vec has been analyzed. Random Forest, Support Vector Machine, Multilayer Perceptron and Convolutional Neural Network were chosen as classifiers.

Keywords: Text classification, NLP, BoW, TF-IDF, Word2Vec, GloVe, FastText, Doc2Vec, CNN, MLP, Random Forest, SVM

Rád bych poděkoval panu ing. Adamu Viktorinovi, Ph.D. za jeho rady a pomoc ve všech fázích psaní této diplomové práce. Rád bych také poděkoval své rodině a přátelům za podporu a trpělivost.

Prohlašuji, že odevzdaná verze bakalářské/diplomové práce a verze elektronická nahraná do IS/STAG jsou totožné.

OBSAH

ÚVOD

V současné době je klasifikace textu jedním z hlavních nástrojů pro zpracování a organizaci textových dat. Klasifikace textů se používá ke zpracování informačních zpráv, k vyhledávání relevantních informací ve velkých objemech textů na internetu nebo k analýze nálad (třídění textů na pozitivní, negativní a neutrální), např. v sociálních médiích.

Účelem klasifikace textu je rozdělit dokumenty do pevně stanoveného počtu předem definovaných kategorií nebo tříd. Dokument může spadat do jedné, mnoha nebo žádné z těchto kategorií.

Protože hovoříme o lidských jazycích, hraje zvláštní roli způsob, jakým je text prezentován, protože lidský jazyk je nestrukturovaný, složitý a různorodý. Obsahuje běžná a žargonová slova, stejně jako velké množství zkratek a akronymů. Slova můžeme kombinovat se slovy z jiných jazyků, a tím obohacovat psaný i mluvený jazyk. Dalším problémem je kontext - stejná fráze může mít v různých souvislostech zcela různý význam.

První část tohoto článku je věnována podrobnému pojednání o klasických a moderních přístupech k reprezentaci textu při řešení problému klasifikace pomocí metod umělé inteligence. Pojednává o populárních klasifikátorech pro tuto úlohu, realizovaných pomocí umělých neuronových sítí i algoritmů strojového učení. Podrobně jsou rozebrány různé části předzpracování textových dat, metody vyhodnocování klasifikátorů a jsou analyzovány populární technologie, knihovny a rámce používané pro řešení úloh klasifikace i předzpracování dat. Oddělená pozornost je věnována metodám vyvažování dat a ladění klasifikačních modelů pomocí křížové validace.

Druhá část je věnována vytvoření datového korpusu, předzpracování získaných dat, analýze získaných dat a způsobům transformace dat pro efektivní fungování uvažovaných klasifikátorů. Byla provedena analýza vlivu různých metod reprezentace textových dat na kvalitu klasifikace pomocí klasifikátorů uvažovaných v tomto článku.

Cílem práce je porovnat známé metody reprezentace textových dat a analyzovat jejich vliv na kvalitu klasifikace pomocí metod umělé inteligence.

I. TEORETICKÁ ČÁST

1 PŘEHLED NLP

Zpracování přirozeného jazyka (NLP) je oblast umělé inteligence, která umožňuje strojům pochopit a extrahovat význam z lidských jazyků. Je to disciplína, která se zaměřuje na interakci mezi datovou vědou a lidským jazykem a zasahuje do bezpočtu průmyslových odvětví.

Lidský jazyk je úžasně složitý a rozmanitý. Vyjadřujeme se nekonečnými způsoby, jak verbálně, tak písemně. Existují stovky jazyků a dialektů. Každý jazyk má jedinečnou sadu pravidel gramatiky a syntaxe a slangu. Při psaní často děláme chyby nebo zkracujeme slova nebo vynecháváme interpunkční znaménka. Když mluvíme, máme regionální přízvuk, mumláme, koktáme a vypůjčujeme si slova z jiných jazyků. [1]

Zpracování přirozeného jazyka (NLP) je zásadní pro úplnou a efektivní analýzu dat textu a řeči. Dokáže si poradit s dialektovými rozdíly, slangem a gramatickými nepřesnostmi, které jsou běžné v každodenních rozhovorech.

Společnosti jej používají pro několik automatizovaných úkolů, jako jsou:

1. Zpracování, analýza a archivace rozsáhlých dokumentů.
2. Analýza zákaznických recenzí nebo záznamů call centra.
3. Použití chatbotů pro automatizovaný zákaznický servis.
4. Klasifikace a extrakce textu.

Integrace NLP s klientskými aplikacemi umožňuje efektivnější komunikaci s klienty. Chatbot například analyzuje a třídí požadavky zákazníků, automaticky odpovídá na běžné otázky a přesměrovává složité požadavky na helpdesk. Tato automatizace pomáhá snižovat náklady, šetří agentům plýtvání časem nadbytečnými požadavky a zvyšuje spokojenost zákazníků. [2]

Díky technologii velkých dat lze NLP použít ke zpracování velkých objemů textových dat prostřednictvím cloudu/distribuovaného počítání s nebývalou rychlostí. Moderní stroje mohou bez únavy analyzovat více jazykových dat než lidé: konzistentně a nestranně.

Vzhledem k obrovskému objemu nestrukturovaných dat, která se každý den generují, od lékařských záznamů po sociální média, bude automatizace zásadní pro efektivní kompletní analýzu textových a řečových dat.

NLP je důležité, protože pomáhá eliminovat nejednoznačnost v jazyce a přidává užitečnou numerickou strukturu k datům pro mnoho následných aplikací, jako je rozpoznávání řeči nebo analýza textu.

NLP změnilo způsob, jakým komunikujeme s počítači, a bude tomu tak i v budoucnu. Tyto technologie umělé inteligence budou hlavní silou při přechodu od činností založených na datech k činnostem založeným na inteligenci, protože v nadcházejících letech budou utvářet a zlepšovat komunikační technologie. [1]

1.1 Přehled metod předzpracování textu v úlohách NLP

Předzpracování textu bylo tradičně důležitým krokem pro úlohy zpracování přirozeného jazyka. Převádí text do lépe stravitelné podoby, aby algoritmy strojového učení mohly lépe fungovat.

Lowercase

Převod všech textových dat na malá písmena je jednou z nejjednodušších a nejúčinnějších forem předzpracování textu. Je použitelná pro většinu úloh textové analýzy a NLP a může pomoci v případech, kdy soubor dat není příliš velký, a výrazně zlepšuje konzistenci očekávaného výsledku.

Stemming

Stemming je proces hledání základu slova pro dané zdrojové slovo. Základ slova navíc nemusí být nutně totožný s morfologickým kořenem slova. Stemming používá hrubý heuristický proces, který odřezává konce slov v naději, že slova správně převede na «stem».

Stemming algoritmy jsou obvykle založeny na pravidlech: slovo prochází řadou podmíněných vět, které určují, jak jej zkrátit. Při extrakci kořenů se odstraní morfologické koncovky a případně předpony, například «ne-». [3]

Lemmatizace

Lemmatizace je velmi podobná stematizaci, rozdíl je však v tom, že lemmatizace ve skutečnosti převádí slova do jejich normální formy. Pro češtinu je to:

1. U podstatných jmen - pád jmenný, jednotné číslo.
2. U přídavných jmen - pád jmenný, jednotné číslo, mužský rod.
3. U sloves - sloveso v infinitivu (neurčitém tvaru) nedokonavého tvaru.

Například slovo „lepší“ by se zobrazilo jako „dobrý“. Pro mapování může používat slovník, jako je WordNet, nebo nějaký speciální přístup založený na pravidlech. [3]

Odstraňování stop slov

Stop slova jsou sada často používaných slov v jazyce. Příklady zastavovacích slov v češtině jsou "a", "aby", "ahoj", "brzo" atd. Myšlenka použití zastavovacích slov spočívá v tom, že odstraněním slov, která obsahují málo informací, se můžeme místo toho zaměřit na důležitá slova.

1.2 Technologie pro řešení problémů NLP

Jedním z nejpopulárnějších programovacích jazyků pro řešení strojového/hlubokého učení a problémů NLP je Python.

Python je univerzální programovací jazyk na vysoké úrovni. Má rostoucí ekosystém knihoven, frameworků a nástrojů. Tyto frameworky a knihovny jsou vybaveny předem napsanými kódy, které uživatelům pomáhají provádět mnoho funkcí a zároveň šetří značné množství času stráveného generováním kódu.

Některé běžné knihovny Pythonu používané pro strojové učení a NLP:

1. NLTK je knihovna, která podporuje úlohy a operace, jako je klasifikace, analýza, značkování, sémantické uvažování, tokenizace a stemming v Pythonu. Je to jeden z hlavních nástrojů pro zpracování přirozeného jazyka v Pythonu a slouží jako pevný základ pro vývojáře Pythonu pracující na projektech NLP a ML. [4]
2. Gensim je knihovna, která se zabývá detekcí sémantické podobnosti mezi dvěma dokumenty pomocí nástrojů tematického modelování a vektorového modelování prostoru. Dokáže sestavit velký text pomocí přírůstkových algoritmů a streamování dat. Schopnost Gensimu zvládnout kompilaci velkého textu je lepší než u jiných balíčků pro pouze paměť a dávkové zpracování. Jedinečné vlastnosti této knihovny jsou rychlost zpracování a neuvěřitelná optimalizace paměti, které NumPy dosahuje. Kromě pokročilých funkcí jsou nejmodernější možnosti modelování prostorových vektorů. [4]
3. Matplotlib - rychlé zpracování a tvorba vysoce kvalitní grafiky sloužící k vizualizaci a vykreslování dat. [5]
4. NumPy - vytvoření univerzální datové struktury užitečné pro analýzu a výměnu algoritmů; pokročilé matematické operace na velkých souborech dat. [5]

5. Pandas - zpracování dat, analýza dat, zploštění dat, restrukturalizace a segmentace datových sad. [5]
6. Scikit-Learn - analýza dat, dolování dat, statistické modelování. [5]
7. TensorFlow - vytváření a trénování neuronových sítí; Detekce vzoru; Numerické výpočty. [5]
8. PyTorch - aplikace pro umělou inteligenci, strojové učení a hluboké učení. [5]
9. PySpark je Python API pro Apache Spark. Apache Spark je analytický engine pro rozsáhlé aplikace pro distribuovanou datovou vědu a strojové učení. [6]

2 METODY REPREZENTACE TEXTOVÝCH DAT

Transformace dat je proces převodu nezpracovaných dat, kterými mohou být text, obrázky, grafy, časové řady atd., na numerické funkce (vektory), aby s nimi bylo možné provádět všechny algebraické operace.

Textová data se obvykle skládají z dokumentů, což mohou být slova, věty nebo dokonce odstavce volného textu. Vzhledem k tomu, že textová data jsou ze své podstaty zašuměná a nestrukturovaná, je pro techniky strojového učení obtížné pracovat přímo se surovými textovými daty.

2.1 Metoda Bag of Words

Toto je jeden z nejjednodušších modelů pro reprezentaci nestrukturovaného textu ve vektorovém prostoru. Vektorový prostorový model je matematický model pro reprezentaci nestrukturovaného textu jako numerické vektory, takže každý rozměr vektoru představuje konkrétní atribut. Model BoW představuje každý textový dokument jako číselný vektor, kde každý rozměr představuje konkrétní slovo z korpusu a hodnotou může být jeho frekvence v dokumentu, výskyt (označený 1 nebo 0) nebo dokonce vážené hodnoty. Název modelu je dán tím, že každý dokument je prezentován doslova jako „pytel“ vlastních slov, bez ohledu na slovosled, sekvence a gramatiku. Příklad reprezentace textu pomocí metody Bag of Words je na obrázku 1. [7]

Obr. 1. Reprezentace textu pomocí metody Bag of Words. [7]

Nevýhody použití modelu BoW:

1. V případě, že nové věty obsahují nová slova, velikost vektoru se zvětší
2. Vektory budou obsahovat mnoho nul, což povede k řídké matici
3. V textu neukládáme žádné informace o gramatice vět ani kontextu.

2.2 Metoda TF-IDF

Protože vektory prvků BoW jsou založeny na absolutní frekvenci výrazů, mohou se některé výrazy objevovat často ve všech dokumentech a mohou zastínit jiné výrazy v sadě prvků. Model TF-IDF se pokouší tento problém vyřešit pomocí škálovacího nebo normalizačního faktoru ve svých výpočtech. TF-IDF ve svých výpočtech používá kombinaci dvou metrik, a to termínovou frekvenci (TF) a inverzní frekvenci dokumentu (IDF).

Vzorec 1 ukazuje výpočet termínovou frekvenci (TF):

$$tf(t,d) = \frac{n_t}{\sum_k n_k} \tag{1}$$

kde n_t je počet výskytů slova t v dokumentu d a jmenovatel je celkový počet slov v dokumentu.

Vzorec 2 ukazuje výpočet termínovou frekvenci (TF):

$$idf(t,D) = \log \frac{|D|}{|\{d_i \in D \mid t \in d_i\}|} \tag{2}$$

kde $|D|$ je počet dokumentů ve sbírce a $|\{d_i \in D | \, t \in \, d_i\}|$ je počet dokumentů ve sbírce D, ve které je t (když $n_t \neq 0$)

Získané hodnocení je váha, ve které nejsou všechna slova stejně důležitá nebo zajímavá. Hodnocení má za následek zvýraznění slov, která se v daném dokumentu liší (obsahují užitečné informace). Vzácná slova mají ve srovnání s běžnými slovy vysoké hodnocení. [7]

2.3 Embedding metody

Tradiční přístupy k reprezentaci textových dat, jako jsou BoW modely a TF-IDF modely, nesbírají informace o významu nebo kontextu slova. To znamená, že potenciální vztahy, jako je kontextová blízkost, nejsou mezi sadami slov pevně dané. Například BoW nebo TF-IDF nemohou zachytit jednoduché vztahy, jako je definice, že slova „pes“ a „kočka“ odkazují na zvířata, o kterých se často mluví v kontextu domácích mazlíčků.

Taková kódování často poskytují dostatečný základ pro jednoduché úlohy NLP, jako jsou klasifikátory e-mailového spamu, ale postrádají dostatečnou složitost pro složitější úlohy, jako je překlad a rozpoznávání řeči. Tradiční přístupy k reprezentaci v NLP, jako je BoW nebo TF-IDF, nezachycují syntaktické (struktura) a sémantické (význam) vztahy mezi slovy, a proto reprezentují jazyk velmi naivním způsobem.

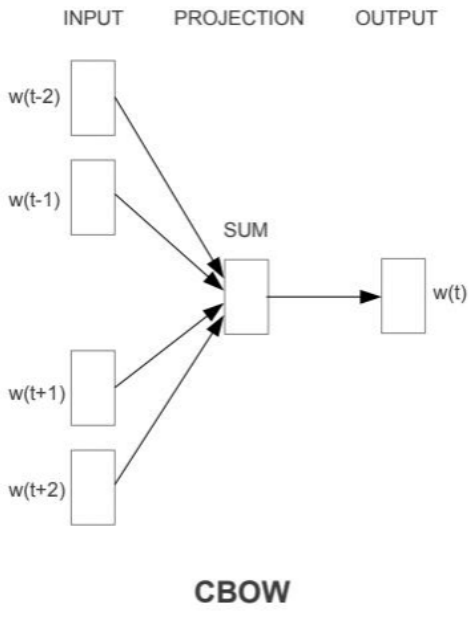
Naproti tomu Word Embedding metody představují slova jako vícerozměrná spojitá čísla s plovoucí desetinnou čárkou, kde jsou sémanticky podobná slova mapována na nejbližší body v geometrickém prostoru. Jinými slovy, slovní vektor je řada reálných čísel, kde každý bod představuje měření významu slova a kde sémanticky podobná slova mají podobné vektory. To znamená, že slova jako „kolo“ a „motor“ by měla mít slovní vektor podobný slovu „auto“ kvůli podobnosti jejich významů, zatímco slovo „jablko“ by mělo být dost vzdálené. Jednou z výhod reprezentace slov jako vektorů je, že se hodí pro matematické operátory. [8]

2.3.1 Model Word2Vec

Účinnost modelu Word2Vec je dána jeho schopností seskupovat vektory podobných slov. S dostatečně velkým souborem dat dokáže Word2Vec přesně odhadnout význam slov na základě jejich výskytu v textu. Tyto odhady poskytují asociace slov s jinými slovy v korpusu. Provedením algebraických operací nad výslednými vektory slov můžeme získat slova, která jsou si významově blízká. Vezmeme-li například vektor, který odpovídá slovu "král", odečteme od něj vektor "muž" a přidáme vektor "žena", získáme vektor, který odpovídá slovu "královna". Existují dvě základní architektury Word2Vec: CBOW a skip-gram.

Architektura CBOW

Tato architektura je velmi podobná neuronové síti s přímou vazbou. Tato architektura modelu se snaží předpovědět cílové slovo ze seznamu kontextových slov. Obrázek 2 ukazuje architekturu CBOW.



Obr. 2. Architektura CBOW [8]

Matice vah mezi vstupní a skrytou vrstvou se označuje W a má rozměr $V \times N$. Matice vah mezi skrytou a výstupní vrstvou se označuje W' a má rozměr $N \times V$. Je třeba poznamenat, že slova je třeba před trénováním vektorizovat, například pomocí TF-IDF nebo one-hot kódování.

Nechť vstupními daty jsou kontextová slova zakódovaná pomocí metody one-hot kódování. Počet kontextových slov C závisí na naší volbě (kolik kontextových slov chceme použít). Skrytou vrstvou se stává průměrná hodnota získaná z každého kontextového slova. Skrytá vrstva má rozměr N. Výstupní vrstva má stejnou dimenzi jako vstupní vrstva. Aktivační funkce skryté vrstvy je lineární a ve výstupní vrstvě je to zobecněná logistická aktivační funkce:

$$f(u)_i = \frac{e^{u_i}}{\sum_{k=1}^{K} e^{u}k} \tag{1}$$

Váhy se nejprve inicializují pomocí standardního normálního rozdělení. Poté se hodnoty ve skryté vrstvě vypočítají podle vzorce:

$$h = \frac{1}{C} W^T \sum_{c=1}^{C} x^{(c)} \tag{2}$$

$$u = W'^T h \tag{3}$$

Volba ztrátové funkce pro odhad výstupní vrstvy neuronové sítě je založena na získání rozsahu hodnot mezi 0 a 1. Vzhledem k tomu, že výstupem je logistická aktivační funkce, kterou lze v tomto problému interpretovat jako podmíněnou pravděpodobnost výskytu slova po určitém kontextu, můžeme ztrátovou funkci definovat takto:

$$L = -u_{j^*} + \log \sum_i e^{u_i} \tag{4}$$

Je tedy nutné minimalizovat tuto ztrátovou funkci a maximalizovat pravděpodobnost, že model předpoví cílové slovo vzhledem ke kontextu.

K trénování modelu se používá gradientní sestup. Aby bylo možné použít algoritmus gradientního sestupu, je nyní třeba zvolit rychlost učení η. Obecně η hyperparametr pro rychlost učení jakéhokoli modelu strojového učení nebo neuronové sítě. S volbou tohoto parametru mohou nastat potíže, a sice: pokud je zvolen velký parametr, může algoritmus přeskočit minimální bod funkce nebo se vůbec "zaseknout", skákat ze strany na stranu, ale nikdy nedosáhnout minima. V případě nižších hodnot bude rychlost učení velmi pomalá. Je však třeba poznamenat, že v tomto případě je velmi obtížné minout minimální bod. [9]

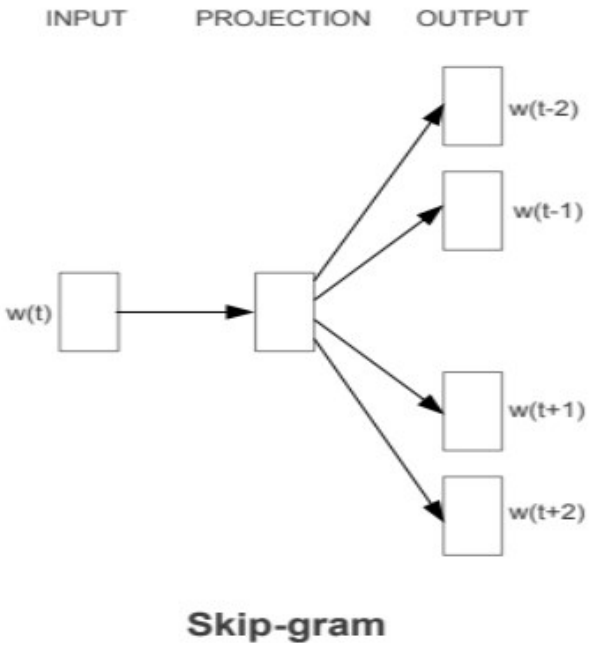
Váhy W a W' se aktualizují podle následujících vzorců [9]:

$$W_{new} = W_{old} - \eta \frac{\partial L}{\partial W} \tag{5}$$

$$W'_{new} = W'_{old} - \eta \frac{\partial L}{\partial W'} \quad (6)$$

Architektura Skip-gram

Skip-gram se používá k předpovědi kontextového slova pro dané cílové slovo. Jedná se o inverzní algoritmus k algoritmu CBOW. Zde je vstupní hodnotou cílové slovo a kontextová slova jsou výstupními hodnotami. Jak je znázorněno na obrázku 3, $w(t)$ je cílové slovo nebo vstup. Existuje jedna skrytá vrstva, která provádí skalární součin mezi váhovou maticí a vstupním vektorem $w(t)$. Skrytá vrstva nepoužívá aktivační funkci. Výsledek skalárního součinu na skryté vrstvě se pak přenáší do výstupní vrstvy. Výstupní vrstva vypočítá skalární součin mezi výstupním vektorem skryté vrstvy a váhovou maticí výstupní vrstvy. Poté použijeme aktivační funkci softmax k výpočtu pravděpodobnosti výskytu slov v kontextu $w(t)$ na daném místě v kontextu.



Obr. 3. Architektura skip-gram [10]

Pro podrobný popis algoritmu uveďme následující pojmy:

1. Nechť V je slovník jedinečných slov přítomných v našem souboru dat.
2. N je počet neuronů přítomných ve skryté vrstvě.
3. Velikost okna je maximální umístění kontextu, v němž mají být slova předpovídána. Velikost okna se označuje c.

4. Dimenze vstupního vektoru je V. Každé slovo je zakódováno pomocí jednorázového kódování.
5. Váhová matice pro skrytou vrstvu W má rozměr $|V| \times N$.
6. Výstupní vektor H skryté vrstvy má rozměr N.
7. Matice vah mezi skrytou a výstupní vrstvou W' má rozměr $N \times |V|$.
8. Skalární součin mezi W' a H nám dává výstupní vektor U, který má dimenzi |V|.

Průběh algoritmu:

1. Slova jsou převedena na vektor pomocí one-hot kódování. Dimenze těchto vektorů je $1 \times |v|$.
2. Slovo $w(t)$ je přeneseno do skryté vrstvy
3. Skrytá vrstva provede skalární součin mezi váhovou maticí W o rozměru $|v| \times N$ a vstupním vektorem $w(t)$. Získáme tak vektor H o rozměru $1 \times N$.
4. Skrytá vrstva nepoužívá aktivační funkci, takže vektor H bude předán přímo výstupní vrstvě.
5. Výstupní vrstva provede skalární součin mezi H a maticí W' a získáme vektor U.

Nyní pro zjištění pravděpodobnosti každého vektoru použijeme funkci softmax. Výsledkem každé iterace je výstupní vektor U, který se vztahuje k jednomu slovu reprezentovanému pomoci one-hot kódováním.

Pokud je předpovězené slovo pro danou kontextovou pozici chybné, použijeme zpětné šíření chyby, abychom změnili váhové matice W a W'. Tyto kroky provedeme pro každé slovo $w(t)$ přítomné ve slovníku.

Funkce pravděpodobnosti se vypočítá takto:

$$p(w_{c,j} = w_{O,c}|w_I) = \frac{\exp u_{c,j}}{\sum_{j=1}^{V} \exp u_j} \quad (7)$$

kde:

1. $w_{c,j}$ je j-té slovo předpovězené na c-té kontextové pozici.
2. $w_{O,c}$ je skutečné slovo přítomné na c-té kontextové pozici.
3. w_I je jediné vstupní slovo.
4. $u_{c,j}$ je j-tá hodnota ve vektoru U při předpovědi slova pro c-tou kontextovou pozici.

Naším cílem je maximalizovat pravděpodobnost předpovědi $w_{c,j}$ na pozici kontextu c. Můžeme reprezentovat ztrátovou funkci L podle vzorce:

$$L = -\log P(w_{c,1}, w_{c,2}, \dots, w_{c,C}|w_{c,O}) = -\sum_{c=1}^{C} u_{c,j^*} + \sum_{c=1}^{C} \log \sum_{j=1}^{V} \exp u_{c,j} \quad (8)$$

Tento model má následující výhody a nevýhody:

1. Jedná se o učení bez učitele, které může pracovat s jakýmkoli nezpracovaným textem.
2. Ve srovnání s jinými vektorovými reprezentacemi slov vyžaduje méně paměti - místo matic $V \times V$ jsou zapotřebí dvě váhové matice $N \times V$, přičemž N je v tomto případě obvykle kolem 300 a V může být v milionech.
3. Funkce softmax vyžaduje velké množství výpočetních prostředků.
4. Doba potřebná k tréninku tohoto algoritmu je poměrně dlouhá. [10]

Další vylepšení Word2Vec

V modelu Word2Vec jsou následující vylepšení, jako je Phrase Learning, Subsampling a Negative Sampling:

1. Phrase Learning znamená, že některá slova by měla být posuzována společně, například „New York“. Slovo „New“ může v některých případech znamenat „nový“, ale pokud jsou slova "New" a "York" pohromadě, znamená to pravděpodobně „New York“.
2. Negative Sampling je nutné ke snížení výpočetních nákladů na trénování. Výstupní vrstva má rozměr N, který se rovná velikosti slovníku. Pokud slovník obsahuje milion slov, je aktualizace vah pro každý neuron příliš nákladná. Proto lze aktualizaci vah provádět pouze pro kontextová slova o 5-6 slovech, která se neshodují s kontextem (při velkém množství dat se lze omezit na 2-3). Kromě toho lze negativní vzorkování nahradit hierarchickým softmaxem (anglicky hierarchical softmax), který rozvine síť do binárního stromu tak, že místo N vah aktualizuje $log(N)$ vah.
3. Subsampling zahrnuje odstranění příliš se opakujících slov. Předložky, spojky mohou být v každém kontextu, ale neodhalují jeho význam. Pro každé slovo se vypočítá pravděpodobnost, že by mělo být zohledněno při tréninku. Tato pravděpodobnost se vypočítá podle vzorce:

$$P(w_i) = \frac{0\,001}{f(w_i)} \sqrt{\frac{f(w_i)}{0.001}} \quad (9)$$

kde w_i slovo a $f(w_i)$ je frekvence výskytu tohoto slova v korpusu. Čím vyšší je tedy P, tím vyšší je pravděpodobnost, že slovo nemá žádnou informační hodnotu. [11]

2.3.2 Model GloVe

GloVe je učební algoritmus bez učitele pro získávání vektorových reprezentací slov. Hlavní nevýhodou skip-gramu a CBOW modelu Word2Vec je, že jsou lokální a nezohledňují globální shody, takže nemohou využít statistické informace přítomné v korpusu. V důsledku toho nevyužívají opakování slov v korpusu. Trénování GloVe se zároveň provádí na agregovaných statistikách globálních shod v korpusu a výsledné reprezentace vykazují zajímavé lineární substruktury vektorů prostoru slov. GloVe navíc na rozdíl od Word2Vec nepoužívá neuronové sítě. V GloVe je ztrátovou funkcí rozdíl mezi součinem počtu vložených slov a logaritmem pravděpodobnosti shody. Cílem je tuto ztrátovou funkci snížit. K nalezení minima této funkce se používá SGD. Výhodou tohoto přístupu je, že umožňuje rychlejší učení.

Algoritmy učení bez učitele využívají statistické informace obsažené ve výskytu slov v korpusu. Klíčová otázka zde zní: „Jak můžeme z těchto statistických údajů generovat význam a jak mohou výsledné vektory slov tento význam reprezentovat?“

Základní myšlenka modelu GloVe, spočívá v tom, že pro korpus obsahující V slov definujeme matici kookurence X, což je matice $V \times V$, kde i je řádek, j je sloupec X, X_{ij} označuje počet výskytů slova i společně se slovem j. Příklad matice kookurence pro angličtinu je uveden na obrázku 4.

	the	cat	sat	on	mat
the	0	1	0	1	1
cat	1	0	1	0	0
sat	0	1	0	1	0
on	1	0	1	0	0
mat	1	0	0	0	0

Obr. 4. Kookurence matice [12]

Abychom získali metriku, která měří sémantickou podobnost mezi slovy, je třeba vypočítat následující pravděpodobnost:

$$P_{ik}/P_{jk},\ kde\ P_{ik} = {}^{X_{ik}}/_{X_i} \tag{1}$$

kde P_{ik} označuje pravděpodobnost výskytu slov i a k společně, která se vypočítá vydělením počtu výskytů i a k společně (X_{ik}) celkovým počtem výskytů slova i v daném případě (X_i). Chceme být schopni vypočítat vektory, které v kombinaci s konkrétní funkcí F udržují tento vztah v prostoru vektorové reprezentace konstantní. Abychom mohli přejít od zavedené metriky ke slovním vektorům, je třeba vyřešit následující problémy:

1. Měla by být definována funkce $F(i, j, k) = P_{ik}/P_{jk}$, kterou lze mapovat na zadanou metriku.
2. Slovní vektory jsou vícerozměrné vektory, ale P_{ik}/P_{jk} je skalární vyjádření. Proto je nutné vyřešit problém neshody dimenzí tak, aby se vypočtená metrika namapovala na vícerozměrný vektor.
3. Jedná se o tři parametry (i, j a k). Výpočet ztrátové funkce se třemi prvky se však může stát těžkopádným a měl by být omezen na dva.

Pro řešení těchto problémů a definování funkce, která mapuje metriku na vícerozměrný vektor, zavádíme následující notace:

1. $w, u \in R^d$- Dvě samostatná vektorová slova
2. w^T- transponované w
3. X - kookurence matice
4. bw и bu - posuny w a u v tomto pořadí
5. V - velikost slovníku

Předpokládejme, že existuje funkce F, která přijímá vektory slov i, j a k a výstupem je vztah zájmu: $F(\mathrm{w_i}, \mathrm{w_j}, \mathrm{u_k}) = P_{ik}/P_{jk}$. Použití dvou různých vektorů slov w a u s různou inicializací snižuje pravděpodobnost přetrénování. Vektory slov jsou lineární struktury, a proto lze s nimi například provádět aritmetické operace např. $w\{král\} - w\{muž\} + w\{žena\} = w\{královna\}$. To umožňuje změnit původní funkci $F(\mathrm{w_i}, \mathrm{w_j}, \mathrm{u_k}) = P_{ik}/P_{jk}$ na $F(\mathrm{w_i} - \mathrm{w_j}, \mathrm{u_k}) = P_{ik}/P_{jk}$. Stojí za povšimnutí, že argumenty funkce F jsou vektory a pravá strana je skalár. Ačkoli na F lze pohlížet jako na komplexní funkci parametrizovanou například neuronovou sítí, mate to lineární strukturu, kterou se snažíme zachytit. Abychom

se tomuto problému vyhnuli, můžeme nejprve vzít skalární součin argumentů, čímž zabráníme nežádoucímu míchání vektorových měření F. Funkce má tedy tvar [13]:

$$F((\mathrm{w_i} - \mathrm{w_j})^{\mathrm{T}} \cdot \mathrm{u_k}) = P_{ik}/P_{jk}. \quad (2)$$

Dále, pokud předpokládáme, že F má homomorfismus mezi aditivní a multiplikativní grupou můžeme funkci F reprezentovat následující rovnicí:

$$F(\mathrm{w_i^T} * u_k - w_j^T * u_k) = F(w_i^T * u_k)/F(w_j^T * u_k) = P_{ik}/P_{jk} \quad (3)$$

Jinými slovy, tento konkrétní homomorfismus zajišťuje, že odčítání $F(A-B)$ lze také reprezentovat jako dělení $F(A)/F(B)$ a dává stejný výsledek. A proto:

$$F(w_i^T * u_k)/F(w_j^T * u_k) = P_{ik}/P_{jk} \quad (4)$$

a tedy $F(w_i^T * u_k) = \mathrm{c} * P_{ik}$, ale také dostaneme $F(w_j^T * u_k) = \mathrm{c} * P_{jk}$ pro libovolné j. Pokud se tedy podobnost mezi i a k zvýší o c, zvýší se o c i podobnost mezi j a k (pro libovolné j). To znamená, že všechny slovní vektory budou zvětšeny nebo zmenšeny o faktor c.

Jestli reprezentujeme F jako exponent tak, že $F = exp$, pak platí výše uvedená vlastnost homomorfismu. Pak můžeme určit,

$$exp(w_i^T * u_k) = P_{ik} = {}^{\mathrm{X_{ik}}}/_{\mathrm{X_i}} \quad (5)$$

$$w_i^T * u_k = \log \mathrm{X_{ik}} - \log \mathrm{X_i}. \quad (6)$$

Pokud si uvědomíme, že tato rovnice by měla výměnnou symetrii, kdyby na pravé straně nebyl $logX_i$. Tento člen je však nezávislý na k, takže jej lze zahrnout do posunu bw_i pro w_i. Nakonec přidáním dalšího posunu bu_k pro u_k se symetrie obnoví a rovnice bude vypadat takto:

$$w_i * u_k + bw_i + bu_k - log(\mathrm{X_{ik}}) = 0 \quad (7)$$

Redukcí rovnice na problém nejmenších čtverců a zavedením váhové funkce $f(X_{ij})$ získáme ztrátovou funkci, která k nalezení svého minima využívá stochastický gradientní sestup [12]:

$$J = \sum_{i,j}^{V} f(X_{ij})\,(w_i * u_j + bw_i + bu_j - log(\mathrm{X_{ij}}))^2 \quad (8)$$

Váhová funkce f musí splňovat následující vlastnosti:

1. $f(0) = 0$. Uvažujeme-li f jako spojitou funkci, která musí při $x \to 0$ konvergovat k nule dostatečně rychle, aby $\log_{x\to 0} f(x) log^2 x$ bylo konečné.

2. $f(x)$ musí být neklesající, aby vzácné zásahy nepřevažovaly.
3. Pro velké hodnoty x musí být $f(x)$ relativně malá, aby nebyly převáženy časté shody.

Jedním z příkladů takové funkce, jak je uvedeno v článku [13], je funkce:

$$f(x) = \begin{cases} \left(\frac{x}{x_{max}}\right)^{\alpha}, & jestli\ x < x_{max} \\ 1, & jinak \end{cases} \tag{9}$$

2.3.3 Model Doc2Vec

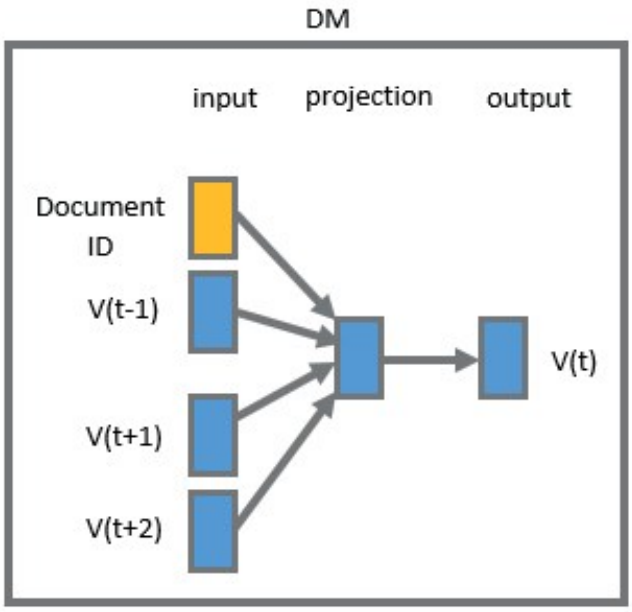
Model Doc2Vec je rozšíření modelu Word2vec, které kóduje celé dokumenty než jednotlivá slova. Vektory Doc2Vec představují téma nebo celkový význam dokumentu. V tomto případě dokument představuje větu, odstavec nebo článek atd.

V Doc2Vec je vstupem název dokumentu, například jméno souboru nebo ID souboru, a výstupem je posuvné okno slov z dokumentu.

Stejně jako v případě Word2vec existují dvě hlavní metody učení: vektorový model odstavců s distribuovanou pamětí (PV-DM) a vektorový model odstavců s distribuovanou sadou slov (PV-DBOW).

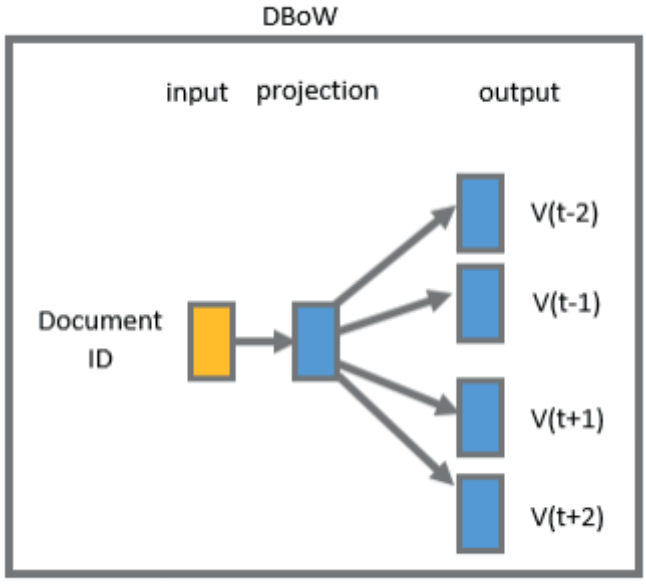
Jedná se o varianty modelů CBOW a skip-gramů používaných k trénování Word2Vec, které rozšiřují myšlenku kontextu na odstavce přidáním štítku nebo identifikátoru dokumentu.

Model distribuované paměti je velmi podobný modelu CBOW aplikace Word2vec. Tento model se snaží předpovědět cílové slovo vzhledem k okolním kontextovým slovům s přidáním identifikátoru odstavce. Architektura distribuované paměti (PV-DM) je znázorněna na obrázku 5.



Obr. 5. Architektura PV-DM [14]

Model distribuované množiny slov založený na modelu skip-gram Word2vec s jednou výjimkou: místo cílového slova bere jako vstup ID dokumentu a snaží se předpovědět náhodně vybraná slova z dokumentu. Architektura distribuované množiny slov (PV-DBOW) je znázorněna na obrázku 6.



Obr. 6. Architektura PV-DBOW [14]

Název vstupního souboru není důležitý. Výstupní slova jsou nezbytná a je třeba je sledovat, protože jsou převzata ze stejného dokumentu. Všechna tato slova jsou spojena s tímto názvem souboru.

Model Doc2Vec dokáže předpovědět slova dokumentu na základě jeho názvu souboru, model Doc2Vec ví, která slova jsou v dokumentu spojena.

Obsah dokumentů se obvykle týká jedné věci; například pozitivní zpětná vazba používá mnoho různých pozitivních slov a negativní zpětná vazba používá mnoho různých negativních slov.

Modely Doc2Vec lze použít následujícím způsobem: pro trénování je zapotřebí sada dokumentů. Pro každé slovo se vygeneruje vektor slov W a pro každý dokument vektor dokumentů D. Model také trénuje váhy pro skrytou vrstvu softmax. Ve výstupní fázi může být předložen nový dokument a všechny váhy jsou fixovány pro výpočet vektoru dokumentu.

Po tréninku můžeme mít k dispozici nový dokument, pro který potřebujeme najít jeho dokumentový vektor. Doc2Vec použije podobnosti slov získané během trénování ke konstrukci vektoru, který bude předpovídat slova v novém dokumentu.

Jakmile máme tento vektor nového dokumentu, můžeme tento vektor porovnat s vektory jiných dokumentů a zjistit, které vektory dokumentů z minulosti jsou si nejvíce podobné.

Tímto způsobem lze Doc2Vec použít k vyhledávání podobných dokumentů. To nám pomůže najít pozitivní a negativní recenze, protože dokument s pozitivními recenzemi bude mít zpravidla podobný vektor a dokument s negativními recenzemi bude mít podobný vektor. [15]

2.3.4 Model FastText

Word2vec považuje každé slovo v korpusu za minimální objekt a generuje pro každé slovo vektor. V tomto smyslu je Word2vec velmi podobný Glove - oba považují slova za nejmenší jednotku, kterou je třeba se naučit.

FastText, který je v podstatě rozšířením modelu Word2Vec, považuje každé slovo za složené z n-gramů znaků. Vektor slova se tedy skládá ze součtu n-gramů daného znaku. Klíčovou myšlenkou bylo využití vnitřní struktury slova ke zlepšení vektorové reprezentace získané metodou skip-gram.

Protože metoda skip-gram používá pro každé slovo samostatný vektor příznaků, ignoruje vnitřní strukturu slov. FastText se pokouší tento problém vyřešit tím, že s každým slovem zachází jako se souborem jeho n -gramových znaků. Vektor pro slovo se jednoduše považuje za součet všech vektorů jeho n-gramových znaků.

Modifikace metody skip-gram se používá takto:

1. Pro slovo vytvoříme n-gram znaků o délce 3 až 6, které se v něm vyskytují.
2. Vezmeme slovo a přidáme hranaté závorky pro označení začátku a konce slova.
3. Poté vygenerujeme n-gramy znaků o délce n. Například pro slovo lze vygenerovat n-gram délky 3 posunutím okna 3 znaků od začátku úhlové závorky ke koncové úhlové závorce. Zde posuneme okno vždy o jeden krok. Tímto způsobem získáme seznam n-gram pro dané slovo.

Příklad 3-gramu pro anglické slovo "eating" je uveden na obrázku 7.

Obr. 7. 3-gram slova „eating" [16]

Vzhledem k tomu, že může existovat obrovské množství jedinečných n-gramů, použijeme hashovaní, abychom omezili paměťové nároky. Namísto učení se vložení pro každý jedinečný n-gram se učíme celkový počet vložení B, kde B označuje velikost koše. unkce FNV-1a. Každý n-gram je zaheslován na celé číslo v rozsahu 1 až B. Ačkoli to může vést ke konfliktům, pomáhá to kontrolovat velikost slovníku. V článku [17] je použita varianta FNV-1a hašovací funkce Fowler-Noll-Vo pro hašování posloupností znaků na celočíselné hodnoty.

Předpokládejme, že máte k dispozici slovník n-gramů o velikosti G. Pro dané slovo w označme $G_i \subset \{1, \ldots, G\}$. Každému n-gramu g přiřaďme vektorovou reprezentaci z_g a v_j - vektorovou reprezentaci slova w_j.

Definujeme vyhodnocovací funkci dvojice (střed, kontext) pro FastText jako součet skalárních součinů vektorových reprezentací n-gram středového slova a kontextového slova, tj.

$$s(w_i, w_j) = \sum_{g \in G_w} z_g^T v_j \quad (1)$$

Po vytvoření n-gramů znaků podle vašeho výběru získáte korpus n-gramů o velikosti G.

Tento jednoduchý model umožňuje sdílet reprezentace pro slova, což mu umožňuje naučit se spolehlivé reprezentace pro vzácná slova. V důsledku toho simulace FastText vlastně řeší následující optimalizační problém [28]:

$$\min_{z,v} \sum_{i=1}^{m} \sum_{j \in C_i} \left(\log\left(1 + e^{-\sum_{g \in G_i} z_g^T v_j}\right) + \sum_{k \in N_{i,j}} \log\left(1 + e^{-\sum_{g \in G_i} z_g^T v_k}\right)\right) \quad (2)$$

Jednou z hlavních výhod modelu FastText ve srovnání s modely Glove a Word2Vec je tedy schopnost generovat slova, která se v tréninkovém korpusu nenacházejí. To lze provést přidáním znaku n-gramu ke všem n-gramovým reprezentacím. Předpokládejme například, že v testovací datové sadě je slovo "normálně", ale v trénovací sadě nemá žádnou reprezentaci. Trénovací soubor však má vektorovou reprezentaci všech jeho n-gramů. Takhle, můžeme tedy jednoduše zprůměrovat vektorovou reprezentaci všech jeho složek n-gramů a získat tak vektorovou reprezentaci tohoto slova, na rozdíl od modelů Word2Vec a Glove. [16]

3 PŘEHLED KLASIFIKAČNÍCH MODELŮ

Klasifikace je jednou z nejoblíbenějších úloh hlubokého a strojového učení. Úkolem klasifikace je předpovědět kategorii objektu a rozdělit objekty podle definovaných a předem určených znaků.

Úloha klasifikace se uplatňuje v mnoha oblastech:

1. V prodeji - klasifikace zákazníků a zboží umožňuje optimalizovat marketingové strategie, stimulovat prodej, snižovat náklady.
2. V telekomunikacích - klasifikace účastníků umožňuje určit úroveň loajality, vytvořit věrnostní programy.
3. V lékařství a zdravotnictví - diagnostika nemocí, klasifikace obyvatelstva podle rizikových skupin.
4. V bankovnictví - credit scoring.

Dalším efektem klasifikace podle předem definovaných parametrů je schopnost rozlišit cokoli, co nezapadá do standardních tříd. Například v medicíně může být vybraným fragmentem jakákoli odchylka od normy: ztluštění, prasknutí, novotvar, nadhodnocení nebo podhodnocení testů atd.

3.1 Support Vector Machine

SVM je algoritmus strojového učení s učitelem, který lze použít pro klasifikační i regresní úlohy. Používá se však především v klasifikačních úlohách, včetně klasifikace textu. V algoritmu SVM reprezentujeme každou datovou položku jako bod v n-rozměrném prostoru (kde n je počet funkcí, které máte k dispozici), přičemž hodnota každé funkce představuje hodnotu určité souřadnice.

Cílem je najít v n-rozměrném prostoru hyperplochu, která rozdělí datové body do jejich potenciálních tříd. Hyperplocha se musí nacházet v maximální vzdálenosti od datových bodů. Datové body s minimální vzdáleností od hyperplochy se nazývají referenční vektory. Vzhledem k jejich těsné blízkosti je jejich vliv na přesnou polohu hyperplochy větší než vliv ostatních datových bodů.

SVM má jádro, které převádí prostor vstupních dat na vícerozměrný prostor.

Funkce $\varphi(x_i)$ představuje funkci jádra, která transformuje vstupní prostor do vícerozměrného prostoru, takže ne každý datový bod je explicitně mapován, což lze také zapsat jako $k(x, x')$.

Která funkce je definovaná a užitečná pro konstrukci hyperploch, závisí na datech. Nejčastěji používané funkce jádra jsou lineární, polynomiální, radiální bázová funkce a sigmoida.

1. Lineární funkce:

$$k(x_i, x_j) = x_i * x_j \tag{1}$$

2. Polynomiální funkce:

$$k(x_i, x_j) = (1 + x_i * x_j)^d \tag{2}$$

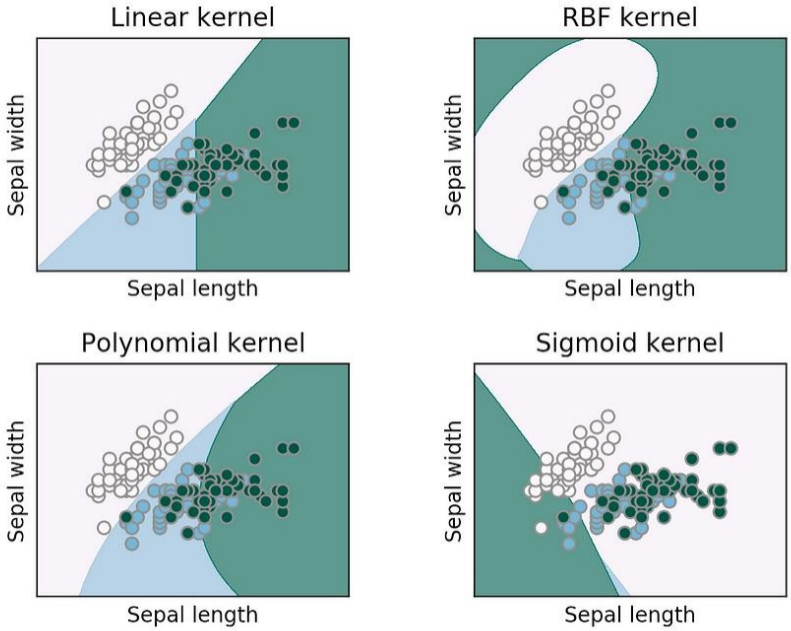
3. Radiální Bázová Funkce (RBF):

$$k(x_i, x_j) = \exp\left(-\gamma \|x_i - x_j\|^2\right), \ \gamma > 0 \tag{3}$$

4. Sigmoida:

$$k(x_i, x_j) = \tanh(kx_i * x_j + c), \ k > 0. \ c < 0 \tag{4}$$

Jádro vezme dva datové body x_n a x_m a vypočítá pro ně odhad vzdálenosti. Tento odhad bude vyšší pro bližší datové body a naopak nižší pro vzdálenější body. Použití tohoto odhadu pomáhá transformovat datové body do vícerozměrného mapování, což snižuje výpočetní náročnost a čas a je užitečné zejména pro obrovské množství dat. Tím se předejde nutnosti složitější transformace. Různá jádra navíc mohou podle datových bodů vytvářet různě tvarované hyperplochy. Je zřejmé, že možnosti lineárních hyperploch se rychle vyčerpají kvůli jejich omezené přizpůsobivosti různým datovým souborům. Na základě této skutečnosti byly vyvinuty různé jaderné funkce. Obrázek 8 ukazuje, jak může vypadat oddělení různých hyperploch při použití různých jaderných funkcí.



Obr. 8. Rozdělení dat pomocí různých funkcí jádra [18]

Následující vzorce představují optimalizační problém, který SVM řeší:

$$\min_{\omega,b,\delta} \frac{1}{2} w^T w + C \sum_{i=1}^{n} \delta_i \tag{5}$$

$$y_i(w^T \varphi(x_i) + b) \geq 1 - \delta_i \tag{6}$$

kde:

1. $\delta_i$označuje vzdálenost k pravému okraji s $\delta_i \geq 0. i = 1, \ldots, n$
2. C označuje parametr regularizace
3. $w^T w = \|w^2\|$ označuje normálový vektor
4. $\varphi(x_i)$ označuje transformovaný vektor vstupního prostoru.
5. b označuje parametr posunu
6. y_i označuje i-tou cílovou hodnotu

Cílem je správně klasifikovat co nejvíce datových bodů, maximalizovat rozpětí od referenčních vektorů k hyperploše a zároveň minimalizovat následující výraz:

$$w^T w \tag{7}$$

Jinými slovy lze cíl vysvětlit také jako nalezení optimálních hodnot w a b tak, aby většina vzorků byla předpovězena správně. V podstatě ne všechny datové body mohou být dokonale rozloženy, takže vzdálenost ke správnému poli je reprezentována δ_i.

Normálový vektor tvoří přímku procházející počátkem. Hyperplochy protínají tuto přímku ortogonálně ve vzdálenosti $\frac{b}{\|w\|_2}$ od počátku.

V ideálním případě bude hodnota $y_i(w^T\varphi(x_i) + b)$ větší než jednota, a bude tedy přesně předpovězena. Nyní, když máme k dispozici datové body se vzdáleností od jejich ideální polohy, musíme změnit ideální případ na $1 - \delta_i$.

Současně je třeba do minimalizačního vzorce zavést sankční člen C. Proměnná C funguje jako regulární parametr a řídí, jak silná je sankce s ohledem na to, kolik datových bodů bylo chybně přiřazeno, přičemž celková vzdálenost je $\sum_{i=1}^{n} \delta_i$.

Tento optimalizační problém lze nazvat duálním problémem protože se snažíme minimalizovat parametry a zároveň maximalizovat vzdálenost nejbližších bodů od hyperplochy. K řešení duálního problému se používá Lagrangeova funkce:

$$L(w, a, b) = \frac{1}{2}\|w\|^2 - \sum_{i=1}^{N} a_i\{y_i(w^T\varphi(x_i) + b) - 1\} \quad (8)$$

Dále můžeme použít následující dvě podmínky:

$$w = \sum_{i=1}^{N} a_i y_i \varphi(x_i) \quad (9)$$

$$\sum_{i=1}^{N} a_i y_i = 0 \quad (10)$$

Parametry w a b lze z Lagrangeova vzorce $L(w, b, a)$ vyloučit. To vede k následující Lagrangeově funkci, která se maximalizuje jako:

$$L = \sum_{n=1}^{N} a_n - \frac{1}{2}\sum_{n=1}^{N} a_n a_m y_n y_m k(x_n, x_m) \quad (11)$$

Tato funkce má následující omezení:

1. $a_n > 0$
2. $\sum_{n=1}^{N} a_n t_n = 0$

Ve svém nejjednodušším typu se algoritmus SVM používá k řešení binárních klasifikačních problémů rozdělením datových bodů na 1 nebo 0. Stejný princip se však používá i pro klasifikaci více tříd. Problém klasifikace více tříd se rozpadá na několik případů binární klasifikace, která se také nazývá jedna-vs-jedna (one-vs-one). V podstatě rozděluje datové body na třídu x a ostatní. Důsledně definovaná třída je odlišena od všech ostatních tříd.

Počet klasifikátorů potřebných pro klasifikaci více tříd one-vs-one lze získat podle následujícího vzorce $\frac{n*(n-1)}{2}$ (kde n je počet tříd):

Při použití pravidla jedna versus jedna tedy každý klasifikátor odděluje body dvou různých tříd a kombinací všech klasifikátorů jedna versus jedna vzniká klasifikátor více tříd. [18]

3.2 Random Forest

Náhodný les (anglicky Random Forest) je jedním z nejoblíbenějších a nejčastěji používaných algoritmů v datové vědě. Jedná se o algoritmus strojového učení s učitelem, který se hojně využívá v klasifikačních a regresních úlohách. Vytváří rozhodovací stromy na různých vzorcích a v případě klasifikace používá většinu hlasů a v případě regrese je průměr.

Jednou z nejdůležitějších vlastností algoritmu náhodného lesa je, že může zpracovávat soubor dat obsahující spojité proměnné, jako v případě regrese, a kategoriální proměnné, jako v případě klasifikace.

Algoritmus náhodného lesa používá techniku sborového učení. Sborové učení znamená kombinaci několika modelů. K predikci se tedy používá soubor modelů, nikoliv jediný model.

Ensemble používá dva typy metod:

1. Bagging - vytváří další trénovací podmnožinu ze vzorku trénovacích dat s náhradou a konečný výsledek je založen na většinovém hlasování
2. Boosting - kombinuje slabé modely do silných modelů vytvářením postupných modelů tak, aby výsledný model měl nejvyšší přesnost

Bagging, známý také jako Bootstrap Aggregation, je metoda souboru používaná náhodným lesem. Bagging vybírá náhodný vzorek/náhodnou podmnožinu z celého souboru dat. Následně je každý model vytvořen ze vzorků (Bootstrap Samples) poskytnutých původními daty, přičemž krok záměny je znám jako výběr řádků. Tento krok vzorkování řádků s nahrazováním se nazývá Bootstrap. Každý model je nyní trénován nezávisle, což přináší výsledky. Konečný výsledek je založen na většinovém hlasování po sloučení výsledků všech modelů. Tento krok, který zahrnuje kombinaci všech výsledků a vytvoření výstupu na základě většinového hlasování, se nazývá agregace.

Boosting je jednou z metod, která využívá koncepci učení souboru. Tento algoritmus kombinuje několik jednoduchých modelů a vytváří konečný výsledek. Provádí se tak, že se model sestaví pomocí konzistentně slabých modelů. [19]

3.2.1 Princip algoritmu Random Forest

1. V modelu náhodného lesa je pro konstrukci každého rozhodovacího stromu vybrána podmnožina datových bodů a podmnožina prvků. Ze souboru dat s k záznamy se vybere n náhodných záznamů a m atributů.
2. Pro každý vzorek jsou zkonstruovány samostatné rozhodovací stromy.
3. Každý rozhodovací strom vygeneruje výstup.
4. Konečný výstup je uvažován na základě většinového hlasování (pro klasifikaci) nebo průměrování (pro regresi).

Důležité výhody modelu Random Forest

1. Různorodost: při generování jednotlivého stromu se nezohledňují všechny atributy funkce, proto je každý strom jiný.
2. Odolnost vůči prokletí dimenze: protože každý strom nezohledňuje všechny atributy, prostor funkcí se zmenšuje.
3. Paralelnost: každý strom je vytvořen nezávisle na různých datech a atributech. To znamená, že k vytváření náhodných lesů můžeme plně využít CPU nebo GPU.
4. Stabilita: stabilita vzniká, protože výsledek je založen na většinovém hlasování nebo průměrování.

3.2.2 Sestavení Decision Tree

Rozhodovací stromy (anglicky Decision Trees) jsou základem mnoha klasických algoritmů strojového učení, jako jsou například náhodné lesy (Random Forest), sáčkové rozhodovací stromy (Bagging) a posílené rozhodovací stromy (Boosted Decision Trees).

Rozhodovací stromy se nyní široce používají v mnoha aplikacích prediktivního modelování, včetně klasifikace i regrese. Někdy se rozhodovací stromy nazývají také CART, což je zkratka pro klasifikační a regresní stromy.

Každý strom má kořenový uzel, přes který procházejí vstupy. Tento kořenový uzel je dále rozdělen na množiny rozhodovacích uzlů, v nichž jsou výsledky a pozorování založeny na určitých podmínkách. Proces rozdělení jednoho uzlu na více uzlů se nazývá rozdělení. Pokud se uzel nedělí na další uzly, nazývá se konečný uzel. Dělení rozhodovacího stromu se nazývá větev nebo podstrom.

Existuje ještě jeden pojem, který je přesným opakem rozdělení. Pokud existují nějaká rozhodovací pravidla, která lze eliminovat, ořezáváme je ze stromu. Tento proces se nazývá prořezávání a je užitečný pro minimalizaci složitosti algoritmu.

Existuje několik metod, které se používají k rozhodování o způsobu rozdělení dat. Hlavním účelem rozhodovacích stromů je provést nejlepší rozdělení mezi uzly, které optimálně rozdělí data do správných kategorií. K tomu je třeba použít správná rozhodovací pravidla. Právě pravidla přímo ovlivňují výkonnost algoritmu.

Je třeba vzít v úvahu některé předpoklady:

1. Všechna data považujeme za kořen, poté pomocí algoritmů provedeme rozdělení nebo rozdělení kořene na podstromy.
2. Hodnoty příznaků jsou považovány za kategoriální. Pokud jsou hodnoty spojité, jsou rozděleny před sestavením modelu.
3. Záznamy jsou přidělovány rekurzivně na základě hodnot atributů.
4. Seřazení atributů jako kořenového nebo vnitřního uzlu stromu se provádí pomocí statistického přístupu.

Index GINI

Pokud jsou všechny položky správně zařazeny do různých tříd (ideální scénář), je rozdělení považováno za čisté. GINI index se používá k odhadu pravděpodobnosti, že náhodně vybraný příklad bude určitým uzlem klasifikován chybně. Je znám jako „indexová“ míra, protože nám dává představu o tom, jak se model liší od čistého dělení.

Hodnocení GINI indexu je vždy mezi 0 a 1, kde 0 znamená, že všechny položky patří do určité třídy, a 1 znamená, že položky jsou náhodně rozděleny do různých tříd. GINI index 0,5 znamená, že prvky jsou rovnoměrně rozděleny do některých tříd. Matematický zápis míry GINI indexu je dán následujícím vzorcem:

$$Gini = 1 - \sum_{i=1}^{n} (p_i)^2 \quad (1)$$

kde p_i je pravděpodobnost, že určitá položka patří do určité třídy.

Kritérium informačního zisku

Kritérium informačního zisku ukazuje množství informací, které atribut získává. Říká nám, jak důležitý je daný atribut. Vzhledem k tomu, že při konstrukci rozhodovacího stromu jde o nalezení správného uzlu rozdělení, který poskytuje vysokou přesnost, jde při informačním

zisku o nalezení nejlepších uzlů, které vracejí nejvyšší informační zisk. Ten se počítá pomocí faktoru známého jako entropie. Entropie určuje míru dezorganizace systému. Čím větší je dezorganizace, tím větší je entropie. Pokud je vzorek zcela homogenní, je entropie nulová, ale pokud je vzorek částečně organizovaný, například z 50 %, je entropie rovna jedné.

To funguje jako základní faktor při určování kritéria informačního zisku. Entropie a kritérium informačního zisku se společně používají k sestavení rozhodovacího stromu a algoritmus je znám jako ID3.

Algoritmus, který se používá k výpočtu přírůstkového informačního kritéria, a tedy ke konstrukci rozhodovacího stromu, se skládá z následujících kroků:

1. Entropie výstupního atributu (před rozdělením) se vypočítá podle vzorce (zde p je pravděpodobnost úspěchu, q je pravděpodobnost selhání uzlu):

$$E(s) = \sum_{i=1}^{c} -p_i \log_2 p_i \tag{2}$$

2. Entropie všech vstupních atributů se vypočítá podle vzorce:

$$E(T, x) = \sum_{c \in X} P(c) E(c) \tag{3}$$

kde T - výstupní atribut, X - vstupní atribut, $P(c)$ je pravděpodobnost vzhledem k možnému datovému bodu přítomnému v X, $E(c)$ je entropie vzhledem k „pravdě“ vzhledem k možnému datovému bodu

3. Na základě dvou výše uvedených hodnot se kritérium informačního zisku nebo snížení entropie vypočítá odečtením entropie každého atributu od celkové entropie před rozdělením:

$$Gain(T, X) = Entropy(T) - Entropy(T, X) \tag{4}$$

4. Jako rozdělovací uzel je vybrán atribut s největším nárůstem informace.

Tyto kroky se opakují, dokud nejsou klasifikována všechna data. Listový uzel je tedy takový, který nemá žádnou entropii nebo jehož entropie je nulová. Na listovém uzlu se neprovádí žádné další rozdělení. Tímto procesem dělení musí projít pouze větev, která potřebuje další dělení, tj. když je entropie > 0 (když existuje index).

Metoda chí-kvadrát

Metoda chí-kvadrát funguje dobře, pokud jsou cílové proměnné kategoriální. Základní myšlenkou algoritmu je zjistit statistickou významnost rozdílů, které existují mezi dílčími uzly a nadřazeným uzlem. Matematická rovnice, která se používá k výpočtu chí-kvadrátu:

$$chi - square = \sqrt{\frac{(Actual - Expected)^2}{Expected}} \quad (5)$$

Je to součet čtverců standardizovaných rozdílů mezi pozorovanými a očekávanými četnostmi cílové proměnné.

Další velkou výhodou použití chí-kvadrátu je, že může provést více rozdělení v jednom uzlu, což vede k větší přesnosti. [20]

3.3 Convolutional Neural Network

Konvoluční neuronové sítě (anglicky CNN) byly původně vyvinuty v komunitě neuronových sítí pro zpracování obrazu, kde dosáhly průlomových výsledků při rozpoznávání objektu z předem definované kategorie (např. kočka, pes, kolo, auto atd.). Konvoluční neuronová síť obvykle zahrnuje dvě operace, které lze považovat za extraktory příznaků: konvoluci (anglicky convolution) a sdružování (anglicky pooling). Výstup této posloupnosti operací je pak obvykle připojen k plně propojené vrstvě, která je podobná tradiční vícevrstvé perseptronové neuronové síti.

Vstupní obraz můžeme reprezentovat jako matici, kde každý záznam představuje každý pixel a hodnota od 0 do 255 představuje intenzitu jasu. Předpokládejme, že se jedná o černobílý obrázek s jediným kanálem představujícím odstíny šedi. Pokud byste zpracovávali barevný obrázek a uvažovali barvy, měli byste 3 kanály podle barevného režimu RGB.

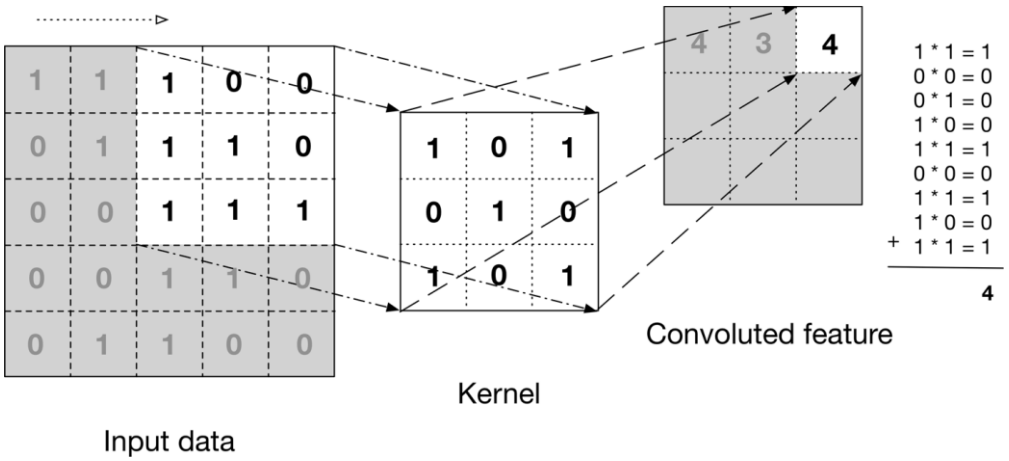
Jedním ze způsobů, jak pochopit operaci konvoluce, je představit si, že konvoluční filtr nebo jádro je umístěno v horní části vstupního obrazu tak, aby se jádro a levý horní roh obrazu shodovaly, a poté jsou hodnoty matice vstupního obrazu vynásobeny odpovídajícími hodnotami v konvolučním filtru. Všechny vynásobené hodnoty se pak sečtou a vznikne jediný skalár, který se umístí na první pozici výsledné matice.

Jádro se poté posune o x pixelů doprava, kde x označuje délku kroku a je parametrem struktury CNN. Proces násobení se poté opakuje, takže se vypočítá a vyplní další hodnota ve výsledné matici.

Tento proces se opakuje, přičemž se nejprve pokryje celý řádek a poté se sloupce posunou o stejnou délku kroku dolů, dokud nejsou pokryty všechny prvky vstupního obrazu.

Výsledkem tohoto procesu je matice s pokrytými všemi prvky, která se nazývá konvoluční funkce nebo mapa vstupních funkcí.

Vstupní obraz lze konvolvovat současně několika konvolučními jádry, přičemž pro každé jádro se vytvoří jeden výstup. Příklad operace konvoluce je uveden na obrázku 9.

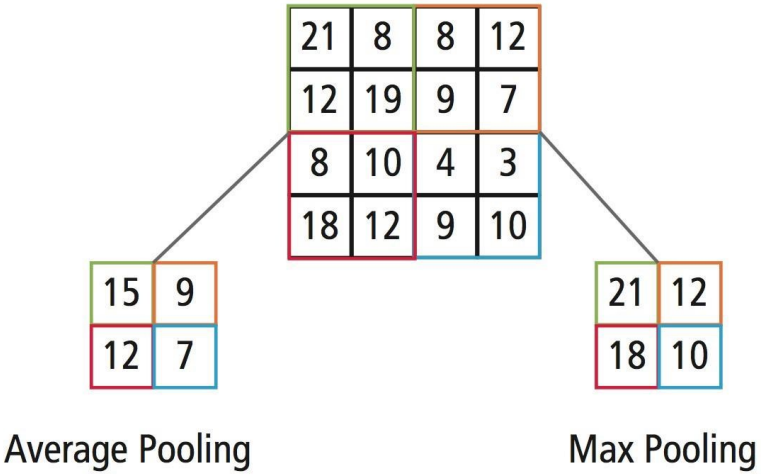


Obr. 9. Příklad operace konvoluce [21]

Další vrstva se nazývá agregační vrstva nebo vrstva se sníženým vzorkováním, která spočívá v aplikaci určité operace na políčka ve vstupní mapě objektů a extrakci určité reprezentativní hodnoty pro každé z analyzovaných políček.

Tento proces je do jisté míry podobný konvoluci, ale místo transformace lokálních políček pomocí naučené lineární transformace (tj. konvolučního filtru) se transformují pomocí operace s pevným kódem.

Dvě nejběžnější agregační operace jsou max-pooling a average-pooling. Max-pooling vybere maximální hodnotu v oblasti vstupní mapy každého kroku a average-pooling zprůměruje hodnoty v této oblasti. Výstupem v každém kroku je tedy jeden skalár, což vede k mnohem menší velikosti výstupu. Příklad operace pooling je uveden na obrázku 10.



Obr. 10. Příklad operace pooling [21]

Dva dříve popsané procesy, tj. konvoluci a pooling, si můžeme představit jako extraktory atributů, tyto atributy pak přenášíme, obvykle jako upravený vektor jednoho řádku, do poslední vrstvy, například vícevrstvého perseptronu pro trénování klasifikace.

V případě úloh NLP, tj. aplikovaných na text, nikoli na obrázky, máme jednorozměrné pole reprezentující text. Zde se architektura konvoluční neuronové sítě mění na jednorozměrné konvoluční a sjednocovací operace.

Jednou z nejtypičtějších úloh v NLP, kde se konvoluční neuronové sítě používají, je klasifikace vět, tj. klasifikace věty do množiny předem definovaných kategorií vzhledem k n-gramům, tj. jedná se o slovo nebo posloupnost slov a symbolů nebo posloupnost symbolů.

Je třeba definovat, co je to 1-D konvoluce nad textem. Nechť je dána posloupnost slov $w_{1:n} = w_1, \dots, w_n$, kde každé z nich je spojeno s vektorem vložení dimenze d. Jednorozměrná konvoluce o šířce k je výsledkem pohybu posuvného okna o velikosti k nad větou a použití stejného konvolučního filtru nebo jádra na každé okno v posloupnosti, tj. skalárního součinu mezi součinem vektorů vložení v daném okně a váhovým vektorem u, často následovaným nelineární aktivační funkcí g.

Vezmeme-li v úvahu okno slov $w_1, \dots, w_n$, pak je spojitý vektor i-tého okna definován jako:

$$x_i = [w_i, \dots, w_{i+k}] \in R^{k \times d} \quad (1)$$

Na každé okno se aplikuje konvoluční filtr, jehož výsledkem jsou skalární hodnoty r_i, vždy pro i-té okno:

$$r_i = g(x_i * u) \in R \quad (2)$$

Lze také použít více filtrů $u, \dots, u_l$, které pak lze reprezentovat jako vektor vynásobený maticí U a s přidáním členu posunu b

$$r_i = g(x_i * U + b) \in R \tag{3}$$

$$\text{kde } r_i \in R^l, x_i \in R^{k \times d}, U \in R^{k*d \times l}, b \in R^l$$

Konvoluční neuronové sítě mohou zpracovávat černobílé i barevné obrazy. V případě, že máme barevný obraz, bude každý pixel místo toho kombinací tří hodnot intenzity, jedné pro každou z červené, zelené a modré složky, a tato reprezentace bude uložena ve třech různých maticích poskytujících různé charakteristiky nebo vzhled obrazu. Běžně se na každý kanál aplikují různé sady filtrů a poté se tři výsledné vektory spojí do jediného vektoru.

Paradigma více kanálů můžeme použít také při zpracování textu. Například, pro větu nebo textové pole může být jedním kanálem posloupnost slov, dalším kanálem může být posloupnost příslušných částí řeči (POS-tag) a třetím kanálem může být forma slov. Příklad rozdělení věty do kanálů je zobrazen na obrázku 11.

Word:	The	plane	lands	in	Lisbon
PoS-tag:	DET	NOUN	VERB	PROP	NOUN
Shape:	Xxx	xxxx	xxxx	xx	Xxxxxx

Obr. 11. Příklad rozdělení věty do kanálů [21]

Aplikací konvoluce na slova získáme m vektorů w, aplikací na značky PoS získáme rovněž m vektorů a totéž platí pro formuláře, opět m vektorů. Tyto tři různé kanály pak lze kombinovat buď součtem:

$$p_i = words_{1:m} + pos_{1:m} + shape_{1:m} \tag{4}$$

nebo konkatenace:

$$p_i = [words_{1:m} : pos_{1:m} : shape_{1:m}] \tag{5}$$

Stejně jako při zpracování obrazu může mít každý kanál ještě různé konvoluce, které čtou zdrojový dokument pomocí různých velikostí jádra, například použitím různých kontextových oken pro slova, poziční značky nebo tvary.

Operace sloučení slouží ke spojení vektorů získaných z různých konvolučních oken do jednoho l-rozměrného vektoru. To se opět provede tak, že se vezme maximální nebo průměrná hodnota zjištěná ve výsledném vektoru z konvolučních oken. V ideálním případě by tento vektor měl odrážet nejdůležitější vlastnosti věty nebo dokumentu.

Tento vektor je pak předán dále v síti do vrstvy plných vazeb, kde se provede predikce. [21]

3.4 Multilayer Perceptron

Vícevrstvý perceptron (MLP) je výkonný člen rodiny umělých neuronových sítí, který lze použít k řešení složitých problémů, jež nelze vyřešit pomocí jediného perceptronu.

MLP jsou v podstatě souborem vzájemně propojených jednotlivých perceptronů, známých také jako neurony nebo uzly, které spolupracují při zpracování a analýze dat. Vícevrstvý perceptron se skládá z více vrstev vzájemně propojených neuronů.

MLP se používají v celé řadě aplikací. Mezi nejčastější aplikace MLP patří:

1. Rozpoznávání obrazu: MLP lze naučit rozpoznávat vzory v obrazech a klasifikovat je do různých kategorií. To je užitečné v aplikacích, jako je rozpoznávání obličejů, detekce objektů a segmentace obrazu.
2. Zpracování přirozeného jazyka (NLP): MLP lze použít k pochopení a vytváření lidského jazyka. To je užitečné v aplikacích, jako je převod textu na řeč, strojový překlad a analýza sentimentu.
3. Prediktivní modelování: Lze jej použít k vytváření předpovědí na základě minulých dat. To je užitečné v aplikacích, jako je předpovídání akciového trhu, předpověď počasí a detekce podvodů.
4. Lékařská diagnostika: Lze ji použít k diagnostice nemocí nebo k interpretaci lékařských snímků pomocí rozpoznávání vzorů v datech.
5. Doporučovací systémy: MLP lze použít k analýze preferencí a chování uživatelů a doporučit jim produkty nebo obsah.

3.4.1 Struktura MLP

Strukturu MLP lze rozdělit na tři hlavní části: vstupní vrstvu, skryté vrstvy a výstupní vrstvu.

Vstupní vrstva neuronové sítě se skládá z umělých vstupních neuronů a zavádí do systému vstupní data pro další zpracování následujícími vrstvami umělých neuronů. Vstupní vrstva je úplným začátkem pracovního postupu umělé neuronové sítě. Jednou z charakteristických vlastností vstupní vrstvy je, že umělé neurony ve vstupní vrstvě hrají jinou roli než neurony v ostatních vrstvách, protože vstupní vrstva se skládá z "pasivních" neuronů, které nepřijímají informace z předchozích vrstev, a protože jsou úplně první vrstvou sítě.

Skrytá vrstva v umělé neuronové síti je vrstva mezi vstupní a výstupní vrstvou, kde umělé neurony přijímají sadu vážených vstupů a vytvářejí výstupy pomocí aktivační funkce. Jedná se o typickou součást téměř každé neuronové sítě, ve které inženýři simulují činnosti probíhající v lidském mozku. Skryté vrstvy neuronové sítě jsou konfigurovány odlišně. V některých případech jsou vážené vstupy přiřazeny náhodně. V jiných případech jsou vyladěny a kalibrovány pomocí procesu zvaného zpětné šíření chyby. V obou případech umělý neuron ve skryté vrstvě pracuje stejně jako biologický neuron v mozku - přijímá pravděpodobnostní vstupy, zpracovává je a převádí na výstupy, které odpovídají axonu biologického neuronu. Mnoho analýz modelů strojového učení se zaměřuje na budování skrytých vrstev v neuronové síti. Existují různé způsoby ladění těchto skrytých vrstev, které vedou k různým výsledkům - například konvoluční neuronové sítě, které se zaměřují na zpracování obrazu, rekurentní neuronové sítě, které obsahují paměťový prvek, a jednoduché neuronové sítě typu feed-forward, které pracují přímo s trénovacími daty.

Výstupní vrstva v umělé neuronové síti je poslední vrstva neuronů, která vytváří daný výstup programu. Ačkoli se vytvářejí stejným způsobem jako ostatní umělé neurony v neuronové síti, neurony výstupní vrstvy mohou být konstruovány nebo pozorovány odlišně, vzhledem k tomu, že jsou posledními uzly sítě. Typická tradiční neuronová síť má tři typy vrstev: jednu nebo více vstupních vrstev, jednu nebo více skrytých vrstev a jednu nebo více výstupních vrstev. Jednoduché dopředné neuronové sítě se třemi odlišnými vrstvami poskytují snadno pochopitelné základní modely. Složitější, inovativní neuronové sítě mohou mít více než jednu vrstvu libovolného typu - a jak bylo zmíněno, každý typ vrstvy může být konstruován jinak.

Stojí za zmínku, že neurony ve vstupní vrstvě musí mít velikost trénovacích instancí a výstupní vrstva musí mít velikost výstupních štítků. Ve skryté vrstvě neuronové sítě však může být libovolný počet neuronů nebo vrstev v závislosti na potřebách, takže čím více neuronů ve skryté vrstvě, tím složitější řešení síť může mít.

Když data vstupují do sítě, procházejí nejprve vstupní vrstvou, ve vstupní vrstvě se neprovádějí žádné specifické operace, pouze předává vstupní data do další vrstvy, kterou je skrytá vrstva. Neurony ve skryté vrstvě provedou s daty nějakou operaci a poté je přenesou do další skryté vrstvy, pokud existuje. Nakonec se zpracovaná data přenesou do výstupní vrstvy, kde se vytvoří výstupní signál.

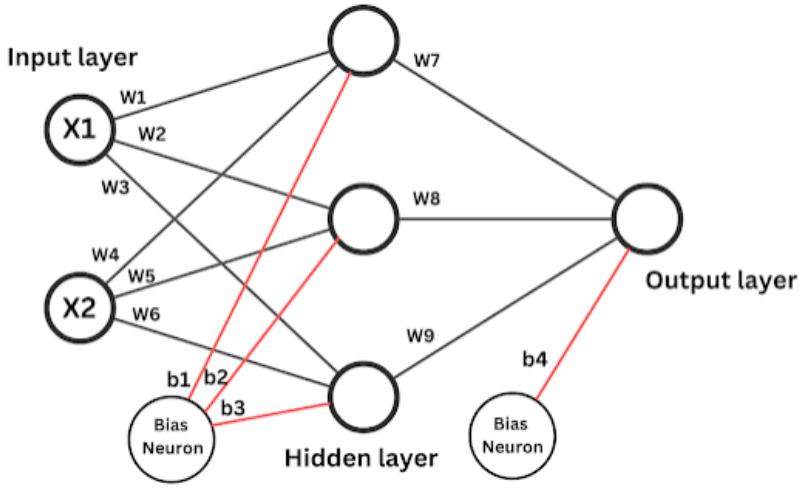
Každému z těchto neuronů přiřadíme určité váhy a posuny. Váhy jsou jednou z nejdůležitějších částí neuronové sítě. Pokud má například neuron vysokou váhu, znamená

to, že má silné spojení s dalším neuronem a jeho výstup bude mít větší vliv na konečný výstup sítě.

Naopak pokud má neuron nízkou váhu, znamená to, že má slabší spojení s dalším neuronem a jeho výstup bude mít menší vliv na konečný výstup sítě.

Na druhou stranu se posuny používají k určení úrovně aktivace neuronu. Posun si lze představit jako prahovou hodnotu, které musí neuron dosáhnout, než vytvoří výstup.

Pokud je vstup neuronu plus jeho offset větší než určitá hodnota, neuron výstup vytvoří, jinak nikoli. Díky tomu je síť pružnější a přizpůsobivější různým typům vstupů. Příklad architektury MLP je znázorněn na obrázku 12.



Obr. 12. Příklad architektury MLP [22]

Výpočet každého neuronu v síti lze provést následujícím způsobem:

$$z = WX + bias \tag{1}$$

kde W jsou váhy jednotlivých neuronů X jsou vstupní data.

To je známé jako vážený součet vstupů plus offset. Když tedy každý neuron obdrží vstup, provede vážený součet vstupů a přičte offset. Vážený součet vstupů není nic jiného než skalární součin vah a vstupů. Přesně to se děje uvnitř každého neuronu v neuronové síti.

Je tu však ještě něco, čemu je třeba věnovat pozornost. I když v každém případě zjistíme vážený součet vstupů plus posunutí, nejedná se o nic jiného než o čistě lineární model, například lineární regresi. Zde tedy potřebujeme do MLP přidat nelinearitu, a to je možné zavedením nelineární funkce, která je v kontextu neuronových sítí známá jako aktivační funkce.

Aktivační funkce si můžete představit jako digitální ekvivalent toho, jak neurony v mozku zpracovávají a reagují na příchozí podněty. Stejně jako se biologické neurony aktivují nebo neaktivují v závislosti na síle vstupních podnětů, které dostávají, aktivační funkce v MLP určují úroveň aktivace neuronu v závislosti na vstupních podnětech, které dostává.

Účelem aktivačních funkcí je modelovat způsob, jakým biologické neurony zpracovávají informace, a vnést do sítě nelinearitu, což jí umožňuje učit se složitým vztahům mezi vstupy a výstupy.

Aktivační funkce je matematická nelineární funkce, která určuje, zda je třeba neurony aktivovat, nebo ne. Tato aktivační funkce je zodpovědná za nelinearitu sítě, aby se mohla učit skutečně složité vzorce v datech.

Aktivační funkce je způsob normalizace vstupních dat. Jinými slovy, pokud máte na vstupu velké číslo, průchodem přes aktivační funkci získáte výstup v požadovaném rozsahu.

Existuje mnoho aktivačních funkcí, ale nejčastěji se používají tyto funkce:

1. Sigmoida: $f(x) = \frac{1}{1+e^{-x}}$
2. Softmax: $f_i(x) = \frac{e^{x_i}}{\sum_k e^{x_k}}$
3. Hyperbolický tangens: $f(x) = \frac{e^{2x}-1}{e^{2x}+1}$
4. ReLU: $f(x) = max(0.Z)$

V každém neuronu sítě se tedy kromě zjištění váženého součtu vstupních dat a přičtení offsetu tyto výsledky také procházejí aktivační funkcí, aby vznikl výstup každého neuronu, což je nelineární hodnota. Tato nelineární hodnota je poté předána další úrovni a proces se opakuje. [22]

3.4.2 Algoritmus učení neuronové sítě

Dopředné šíření (anglicky forward propagation) se týká výpočtu a ukládání meziproměnných (včetně výstupních dat) pro neuronovou síť v pořadí od vstupní vrstvy k výstupní vrstvě.

Uvažujeme jednoduchou neuronovou síť s jednou skrytou vrstvou. Pro zjednodušení předpokládejme, že vstupní data jsou dána jako $x \in R^d$ a že naše skrytá vrstva neobsahuje zkreslující člen. Definujme proměnnou:

$$z = W^1 x \tag{1}$$

kde $W^1 \in R^{h \times d}$ je váhový parametr skryté vrstvy. Po spuštění meziproměnné $z \in R^h$ přes aktivační funkci f získáme náš skrytý aktivační vektor délky h

$$h = \phi(z) \tag{2}$$

Výstup skryté vrstvy h je rovněž meziproměnnou. Za předpokladu, že parametry výstupní vrstvy mají pouze váhu $W^2 \in R^{q \times h}$, můžeme získat proměnnou výstupní vrstvy s vektorem délky q:

$$o = W^2 h \tag{3}$$

Za předpokladu, že ztrátová funkce je l a označení příkladu je y, můžeme vypočítat chybu pro jeden příklad dat:

$$L = l(o, y) \tag{4}$$

Definujte regularizační člen s:

$$s = \frac{\lambda}{2}(||W^1||_F^2 + ||W^2||_F^2) \tag{5}$$

kde Frobeniova norma matice je jednoduše norma l_2 aplikovaná po redukci matice na vektor. Regularizovaná modelová ztráta se vypočítá podle vzorce:

$$J = L + s \tag{6}$$

kterou považujeme za cílovou funkci. [23]

Algoritmus Backpropagation

Zpětné šíření (backpropagation) je podstatou trénování neuronových sítí. Jedná se o metodu jemného doladění vah neuronové sítě na základě chybovosti získané v předchozí epoše (tj. iteraci). Správné nastavení vah snižuje počet chyb a zvyšuje robustnost modelu tím, že zvyšuje jeho zobecnění.

Zpětné šíření v neuronové síti je zkrácená forma „zpětného šíření chyb“. Je to standardní metoda pro trénování umělých neuronových sítí. Tato metoda pomáhá vypočítat gradient ztrátové funkce vzhledem ke všem vahám v síti.

Zpětné šíření se vztahuje na metodu výpočtu gradientu parametrů neuronové sítě. Stručně řečeno, tato metoda prochází sítí v opačném pořadí, od výstupní vrstvy ke vstupní vrstvě, podle pravidla řetězového výpočtu. Algoritmus ukládá všechny meziproměnné (parciální derivace) potřebné při výpočtu gradientu některých parametrů.

Předpokládejme, že máme funkce $Y = f(X)$ a $Z = g(Y)$, kde vstup a výstup X, Y, Z jsou tenzory libovolného tvaru. Pomocí řetězového pravidla můžeme spočítat derivaci Z nad X takto

$$\frac{\partial Z}{\partial X} = prod\left(\frac{\partial Z}{\partial Y}, \frac{\partial Y}{\partial X}\right) \tag{7}$$

Zde používáme operátor prod, kterým po provedení potřebných operací argumenty vynásobíme. U vektorů je to jednoduché: jde o prosté násobení matice maticí. Pro tenzory vyšších rozměrů použijeme příslušný protějšek.

Účelem backpropagation je výpočet gradientů $\partial J/\partial W^1$ a $\partial J/\partial W^2$. K tomu použijeme řetězové pravidlo a postupně vypočítáme gradient každé meziproměnné a parametru. Pořadí výpočtu je oproti přímému šíření obrácené, protože musíme začít od výsledku výpočetního grafu a postupovat k parametrům. Prvním krokem je výpočet gradientů cílové funkce $J = L + s$ s ohledem na ztrátový člen L a regularizační člen s.

$$\frac{\partial J}{\partial L} = 1 \tag{8}$$

$$\frac{\partial J}{\partial s} = 1 \tag{9}$$

Poté vypočítáme gradient cílové funkce nad proměnnou o výstupní vrstvy pomocí řetězového pravidla:

$$\frac{\partial J}{\partial o} = prod\left(\frac{\partial J}{\partial L}, \frac{\partial L}{\partial o}\right) = \frac{\partial L}{\partial o} \in R^q$$

Poté vypočítáme gradienty regularizačního členu pro oba parametry:

$$\frac{\partial s}{\partial W^1} = \lambda W^1 \tag{10}$$

$$\frac{\partial s}{\partial W^2} = \lambda W^2 \tag{11}$$

Nyní můžeme vypočítat gradient $\partial J/\partial W^2 \in R^{q \times h}$ nejbližších parametrů modelu k výstupní vrstvě. Použitím řetězového pravidla získáme:

$$\frac{\partial J}{\partial W^2} = prod\left(\frac{\partial J}{\partial o}, \frac{\partial o}{\partial W^2}\right) + prod\left(\frac{\partial J}{\partial s}, \frac{\partial s}{\partial W^2}\right) = \frac{\partial J}{\partial o} h^T + \lambda W^2 \tag{12}$$

Abychom získali gradient vzhledem k W^1, musíme pokračovat ve zpětném šíření podél výstupní vrstvy do skryté vrstvy. Gradient vzhledem k výstupu skryté vrstvy $\partial J/\partial h \in R^q$ je definován výrazem:

$$\frac{\partial J}{\partial h} = prod\left(\frac{\partial J}{\partial o}, \frac{\partial o}{\partial h}\right) = {W^2}^T \frac{\partial J}{\partial o} \tag{13}$$

Protože aktivační funkce ϕ se aplikuje prvek po prvku, vyžaduje výpočet gradientu $\partial J/\partial z \in R^h$ mezilehlé proměnné z použití operátoru násobení prvek po prvku, který označíme jako $\odot$:

$$\frac{\partial J}{\partial z} = prod\left(\frac{\partial J}{\partial h}, \frac{\partial h}{\partial z}\right) = \frac{\partial J}{\partial h} \odot \phi'(z) \tag{14}$$

Nakonec můžeme získat gradient $\partial J / \partial W^1 \in R^{h \times d}$ parametrů modelu nejblíže vstupní vrstvě. Řetězovým pravidlem získáme:

$$\frac{\partial J}{\partial W^1} = prod\left(\frac{\partial J}{\partial z}, \frac{\partial z}{\partial W^1}\right) + prod\left(\frac{\partial J}{\partial s}, \frac{\partial s}{\partial W^1}\right) = \frac{\partial J}{\partial z} X^T + \lambda W^1 \tag{15}$$

Mezi nejvýznamnější výhody zpětného šíření patří:

1. reverzní propagace je rychlá, jednoduchá a snadno programovatelná
2. Nemá žádné parametry, které by bylo třeba konfigurovat, kromě počtu vstupů.
3. Je to flexibilní metoda, protože nevyžaduje žádné předchozí znalosti sítě.
4. Je to standardní metoda, která obvykle dobře funguje.
5. Není zde žádná zvláštní zmínka o vlastnostech zkoumané funkce. [23]

Trénování neuronových sítí

Při trénování neuronových sítí jsou dopředné a zpětné šíření na sobě závislé. Konkrétně při dopředné propagaci procházíme výpočetní graf ve směru závislostí a počítáme všechny proměnné na jeho cestě. Ty se pak použijí pro zpětnou propagaci, kdy se pořadí výpočtu na grafu obrátí. Tento proces se opakuje, dokud chyba učení není menší než určitá pevná hodnota nebo dokud se proces učení nezastaví po daném počtu iterací. [23]

Chyba je procentuální hodnota odrážející rozdíl mezi očekávanými a přijatými odpověďmi. Chyba se tvoří každou epochu a měla by klesat. Chybu lze vypočítat různými způsoby, ale nejčastěji se používají tyto metody: střední kvadratickou chybou (MSE), kořenovou MSE a také chybovou funkci využívající arctan. Je třeba poznamenat, že každá metoda počítá chyby jinak. U metody arctan bude chyba téměř vždy větší, protože pracuje na principu: čím větší rozdíl, tím větší chyba. Root MSE bude mít nejmenší chybu, proto se častěji používá MSE, kterou zachovává rovnováhu při výpočtu chyb:

1. MSE: $\frac{(i_1-a_1)^2+(i_2-a_2)^2+\cdots+(i_n-a_n)^2}{n}$
2. RootMSE: $\sqrt{\frac{(i_1-a_1)^2+(i_2-a_2)^2+\cdots+(i_n-a_n)^2}{n}}$
3. Arctan: $\frac{arctan^2(i_1-a_1)+arctan^2(i_1-a_1)+\cdots+arctan^2(i_1-a_1)}{n}$

Proto při trénování neuronových sítí po inicializaci parametrů modelu střídáme dopředné a zpětné šíření a aktualizujeme parametry modelu pomocí gradientů daných zpětným šířením.

Stojí za zmínku, že zpětné šíření znovu využívá uložené mezihodnoty z dopředného šíření, aby se zabránilo duplikaci výpočtů. Jedním z důsledků této skutečnosti je nutnost ukládat mezihodnoty, dokud není zpětné šíření dokončeno.

To je také jeden z důvodů, proč trénování vyžaduje podstatně více paměti než prostá predikce. Velikost těchto mezivrstev je navíc zhruba úměrná počtu vrstev sítě a velikosti paketu. Trénování hlubších sítí pomocí větších velikostí paketů tedy snadněji vede k chybám při nedostatku paměti. [23]

4 PŘÍSTUPY KE ZLEPŠENÍ KVALITY KLASIFIKAČNÍCH MODELŮ

Při vývoji klasifikačních modelů mohou vznikat různé problémy, které mají negativní dopad na kvalitu klasifikace. Například data mohou být nevyvážená nebo mohou obsahovat velmi velký počet atributů. Existují různé přístupy k řešení těchto problémů, například použití technik vyvažování dat nebo technik redukce dimenzionality, jako je PCA. Zvláštní roli při ladění modelu hraje křížová validace, která umožňuje zkvalitnit model během jeho vývoje.

4.1 Způsoby hodnocení modelu

Přesnost klasifikace (anglicky Accuracy) je celkový počet správných předpovědí dělený celkovým počtem předpovědí provedených pro danou sadu dat.

Přesnost je jednou z nejzřetelnějších metrik, je to míra všech správně identifikovaných případů:

$$Accuracy = \frac{TN+TP}{TN+FN+TP+FP} \quad (1)$$

kde TP je True Positive, TN je True Negative, FP je False Positive, FN je False Negative.

Jako odhadovací míra však není vhodná pro nevyvážené klasifikační úlohy. Hlavním důvodem je, že převažující počet příkladů z většinové třídy (nebo tříd) bude převyšovat počet příkladů z menšinové třídy, což znamená, že i nekvalifikované modely mohou dosáhnout přesnosti 90% nebo 99%, v závislosti na tom, jak závažná je v souboru dat nerovnováha tříd.

Alternativou k použití klasifikační přesnosti je použití metrik, jako je přesnost Precision a Recall.

Precision a Recall jsou metriky, které se používají při hodnocení většiny klasifikačních algoritmů. Někdy se používají samostatně, jindy jako základ odvozených metrik, jako je F-measure nebo R-Precision.

Precision je implikována jako míra správně identifikovaných pozitivních případů ze všech předpovězených pozitivních případů. Je tedy užitečná v případě, že cena falešně pozitivních případů je vysoká.

$$Precision = \frac{TP}{TP+FP} \quad (2)$$

Recall je míra správně identifikovaných pozitivních případů ze všech skutečných pozitivních případů. To je důležité v případě, že náklady na falešně negativní výsledky jsou vysoké.

$$Recall = \frac{TP}{TP+FN} \tag{3}$$

f-score je harmonický průměr přesnosti a úplnosti, který poskytuje lepší odhad chybně klasifikovaných případů než metrika přesnosti.

$$f - score = 2 * \frac{Precision*Recall}{Precision+Recall} \tag{4}$$

4.2 Oprava nevyváženosti v souboru dat

Datový soubor vykazuje nerovnováhu tříd, pokud obsahuje mnohem více příkladů jedné třídy než druhé. V důsledku toho mají klasifikátory tendenci ignorovat malé třídy a soustředit se na přesnou klasifikaci větších tříd.

4.2.1 Random over-sampling a under-sampling

Převzorkování zahrnuje vytvoření nové transformované verze trénovacího souboru dat, ve kterém mají vybrané příklady jiné rozložení tříd. Jedná se o jednoduchou a účinnou strategii pro nevyvážené klasifikační úlohy. Nejjednodušší strategií je náhodný výběr příkladů pro transformovaný soubor dat, který se nazývá náhodné převzorkování.

Existují dva hlavní přístupy k náhodnému převzorkování pro nevyváženou klasifikaci; jsou to převzorkování (over-sampling) a podvzorkování (under-sampling):

1. Random over-sampling - náhodný nadvýběr příkladů v menšinové třídě.
2. Random under-sampling: náhodné odstranění příkladů ve většině tříd.

Náhodné převzorkování zahrnuje náhodný výběr příkladů z menšinové třídy s nahrazením a jejich přidání do trénovacího souboru dat. Náhodný podvýběr zahrnuje náhodný výběr příkladů z většinové třídy a jejich odstranění z trénovacího souboru dat.

Jinými slovy, nadměrný výběr zahrnuje vytvoření nových datových bodů pro menšinovou třídu, zatímco nedostatečný výběr zahrnuje odstranění datových bodů z většinové třídy. Tím se poněkud sníží míra nerovnováhy v souboru dat.

Tyto metody se nazývají "naivní metody převzorkování", protože nepředpokládají nic o datech a nepoužívají žádnou heuristiku. Díky tomu jsou jednoduché na implementaci a rychlé na provedení, což je žádoucí pro velmi rozsáhlé a složité datové soubory.

Obě metody lze použít pro dvoutřídní (binární) klasifikační problémy i pro vícetřídní klasifikační problémy s jednou nebo více většinovými či menšinovými třídami. [24]

4.2.2 Algoritmus SMOTE

Algoritmus SMOTE (Synthetic Minority Oversampling Technique) generuje nové vzorky mezi stávajícími datovými body na základě jejich lokální hustoty a jejich hranic s jinou třídou. Provádí nejen převzorkování, ale může také následně použít čisticí metody k odstranění redundance na konci.

Kroky algoritmu SMOTE jsou následující. Pro každý minoritní vzorek:

1. Najděte k nejbližších sousedů minority
2. Náhodně vyberte j z těchto sousedů
3. Náhodně vygenerujte syntetické vzorky podél linií spojujících minoritní vzorek a jeho j vybraných sousedů (j závisí na požadovaném stupni převzorkování)

Nevýhody SMOTE:

1. Přílišné zobecnění. Postup SMOTE může být nebezpečný, protože slepě zobecňuje na menšinovou oblast bez ohledu na většinovou třídu. Tato strategie je problematická zejména v případě vysoce asymetrických rozdělení tříd, protože v takových případech je menšinová třída ve srovnání s většinovou třídou velmi řídká, což vede k větší pravděpodobnosti míchání tříd.
2. Nepružnost. Počet syntetických vzorků generovaných programem SMOTE je předem pevně stanoven, což neumožňuje žádnou flexibilitu v míře vyvážení.

Dalším potenciálním problémem je, že SMOTE může zavést příklady umělých menšinových tříd příliš hluboko do prostoru většinových tříd. Tuto nevýhodu lze odstranit hybridizací: kombinací SMOTE s algoritmy podvzorkování.

Jedním z nejznámějších z nich je Tomek Links. Tomek Links jsou dvojice instancí opačných tříd, které jsou si navzájem nejbližšími sousedy. Jinými slovy, jsou to dvojice opačných instancí, které jsou si velmi blízké.

Algoritmus Tomek Links vyhledává takové dvojice a odstraňuje z nich prvek většinové třídy. Smyslem je zpřesnit hranici mezi minoritní a majoritní třídou, čímž se minoritní oblast (oblasti) více odliší.

Algoritmus Tomek Links je tedy metodou podvzorkování, která funguje jako metoda čištění dat pro SMOTE za účelem korekce redundance. Existuje mnoho dalších metod dílčího vzorkování, které lze kombinovat s metodou SMOTE, aby plnily stejnou funkci.

Dalším příkladem jsou upravené nejbližší sousedy (anglicky Edited Nearest Neighbor). ENN odstraní každý příklad, jehož značka třídy se liší od třídy alespoň dvou jeho sousedů. ENN odstraní více příkladů než Tomek Links a může také odstranit příklady z obou tříd. [25]

Mezi další varianty SMOTE patří Borderline SMOTE, SVM SMOTE a KMeans SMOTE a varianty metod dílčích vzorků používaných s SMOTE jsou Condensed Nearest Neighbor (CNN), Repeated Edited Nearest Neighbor a Instance Hardness Threshold.

4.3 Principal Component Analysis

Analýza hlavních komponent (anglicky Principal Component Analysis) je technika redukce dimenze, která představuje proces redukce počtu predikčních proměnných v souboru dat. [26]

Účelem PCA je identifikovat vzory v souboru dat a poté redukovat proměnné na jejich nejdůležitější charakteristiky, aby bylo možné data zjednodušit, aniž by došlo ke ztrátě důležitých charakteristik. PCA se ptá, zda jsou všechny dimenze souboru dat radostné, a poté dává uživateli možnost vyloučit ty, které radostné nejsou.

Prokletí dimenzionality je soubor jevů, že s rostoucí dimenzionalitou má tendenci klesat ovladatelnost a efektivita dat. Na vysoké úrovni se prokletí dimenzionality týká skutečnosti, že s přidáváním dimenzí atributů do datového souboru se zvyšuje střední a minimální vzdálenost mezi body.

S rostoucí vzdáleností mezi známými a neznámými body se stává správná předpověď obtížnější. Kromě toho mohou mít rysy ve vašem souboru dat malou hodnotu nebo prediktivní sílu v kontextu cílové proměnné. Tyto funkce nezlepšují model, ale zvyšují šum v souboru dat a celkovou výpočetní zátěž modelu.

Kvůli prokletí dimenzionality je redukce dimenzionality často kritickou součástí analytických procesů. Zejména v datově náročných aplikacích, jako je počítačové vidění nebo zpracování signálů.

Při sběru dat není vždy zřejmé, které proměnné jsou důležité. Dokonce ani není zaručeno, že vybrané nebo poskytnuté proměnné jsou správné. V éře velkých dat se navíc samotný počet proměnných v souboru dat může vymknout kontrole a stát se dokonce matoucím a zavádějícím. To může ztížit ruční výběr smysluplných proměnných.

4.3.1 Princip fungování algoritmu PCA

PCA původně pochází z oboru lineární algebry. Jedná se o transformační techniku, která vytváří kombinace původních proměnných v souboru dat se záměrem, aby nové kombinace pokryly co největší část rozptylu v souboru dat a zároveň eliminovaly korelace. PCA vytváří nové proměnné transformací průměrně centrovaných pozorování v souboru dat do nové sady proměnných pomocí vlastních vektorů a vlastních hodnot vypočtených z kovarianční matice vašich původních proměnných.

Prvním krokem PCA je vystředění hodnot všech vstupních proměnných, čímž se průměr každé proměnné rovná nule. Centrování je důležitým krokem před zpracováním, protože zajišťuje, že výsledné komponenty zohledňují pouze rozptyl v souboru dat, a nikoli zachycují celkový průměr souboru dat jako důležitou proměnnou. Bez vycentrování průměru může první hlavní komponenta nalezená pomocí PCA odpovídat spíše průměru dat než směru maximálního rozptylu. Jakmile jsou data vycentrována, je třeba vypočítat kovarianční matici dat.

Kovariance se měří mezi dvěma proměnnými současně a popisuje, jak spolu hodnoty proměnných souvisejí. Velká hodnota kovariance naznačuje, že proměnné mají mezi sebou silný lineární vztah. Hodnoty kovariance blízké 0 naznačují slabý nebo neexistující lineární vztah. Kovariance se vždy měří ve dvou rozměrech. Pokud máte co do činění s více než dvěma proměnnými, nejefektivnějším způsobem, jak získat všechny možné hodnoty kovariance, je sestavit je do matice. V kovarianční matici představuje diagonála rozptyl pro každou proměnnou a hodnoty na diagonále jsou navzájem zrcadlovými obrazy, protože každá kombinace proměnných je v matici zahrnuta dvakrát. Jedná se o čtvercovou symetrickou matici.

Protože jsou čtvercové a symetrické, lze kovarianční matice redukovat na diagonální typ, což znamená, že na matici lze vypočítat vlastní rozklad. V tomto případě PCA najde vlastní

vektory a vlastní hodnoty pro soubor dat. Vlastní vektor lineární transformace je nenulový vektor, který se při aplikaci příslušné lineární transformace změní o násobek skalárního čísla. Vlastní hodnota je skalár spojený s vlastním vektorem.

V kontextu PCA jsou vlastní vektory kovarianční matice osami hlavních komponent v souboru dat. Vlastní vektory určují směry hlavních komponent vypočtených metodou PCA. Vlastní čísla spojená s vlastními vektory popisují velikost vlastního vektoru nebo to, jak daleko od sebe jsou body podél nové osy.

První vlastní vektor bude zahrnovat největší vzdálenost mezi body zjištěný v souboru dat a všechny následující vlastní vektory budou kolmé k tomu, který byl vypočten před ním. Díky tomu můžeme vědět, že jednotlivé hlavní komponenty nebudou vzájemně korelovat. Každý vlastní vektor nalezený pomocí PCA bude shromažďovat kombinaci rozptylů původních proměnných v souboru dat.

Vlastní čísla jsou důležitá, protože poskytují kritérium pořadí nově získaných proměnných. Hlavní komponenty (vlastní vektory) jsou seřazeny podle klesající vlastní hodnoty. Hlavní komponenty s největšími vlastními čísly jsou vybrány jako první, protože odpovídají za největší rozptyl v datech. Po určení hlavních komponent souboru dat je nutné transformovat pozorování původního souboru dat na vybrané hlavní komponenty. K transformaci zdrojových bodů vytvoříme projekční matici. Tato projekční matice je jednoduše vybrané vlastní vektory spojené do matice. Následně můžeme vynásobit matici našich původních pozorování a proměnných naší projekční maticí.

Výsledkem tohoto procesu je transformovaný soubor dat promítnutý do našeho nového datového prostoru, který se skládá z našich hlavních komponent. [26]

4.3.2 Omezení PCA

PCA předpokládá, že data lze aproximovat lineární strukturou a že data lze popsat pomocí menšího počtu rysů. Předpokládá se, že lineární transformace může a bude pokrývat nejdůležitější aspekty dat. Předpokládá se také, že vysoký rozptyl dat znamená vysoký poměr signálu k šumu.

Redukce dimenzionality vede ke ztrátě některých informací. Tím, že nejsou zachovány všechny vlastní vektory, dochází ke ztrátě některých informací. Pokud jsou však vlastní hodnoty nezahrnutých vlastních vektorů malé, neztrácí se příliš mnoho informací. [26]

4.4 Cross-validation

Křížová validace je postup převzorkování, který se používá k hodnocení modelů strojového učení na omezeném vzorku dat.

Tento postup má jediný parametr, který se nazývá k a který označuje počet skupin, do nichž má být daný vzorek dat rozdělen. Proto se tento postup často označuje jako k-násobná křížová validace. Je-li zvolena konkrétní hodnota k, lze ji použít místo k v odkazu na model, např. $k = 10$ se stává desetinásobnou křížovou validací.

Křížová validace se používá hlavně v aplikovaném strojovém učení k vyhodnocení dovedností modelu strojového učení na neznámých datech. To znamená, že se použije omezený vzorek k vyhodnocení toho, jak by měl model celkově fungovat, když se použije k předpovídání dat, která nebyla použita při trénování modelu.

Jedná se o oblíbenou metodu, protože je snadno pochopitelná a obvykle vede k méně zkresleným nebo méně optimistickým odhadům dovedností modelu než jiné metody, jako je prosté rozdělené učení/testování.

Obecný postup je následující:

1. Náhodně zamíchejte soubor dat.
2. Rozdělte soubor dat do k skupin.

Pro každou jedinečnou skupinu:

1. Je třeba vzít skupinu jako kontrolní nebo testovací soubor dat.
2. Zbývající skupiny jsou brány jako trénovací soubor dat.
3. Model se natrénuje na trénovací sadě a vyhodnotí se na testovací sadě.

Je důležité si uvědomit, že každé pozorování ve vzorku dat patří do jiné skupiny a v této skupině zůstává po celou dobu postupu. To znamená, že každý vzorek má možnost být použit v retenčním souboru 1krát a použit k trénování modelu $k - 1$ krát. Hodnotu k je třeba pečlivě zvolit pro váš vzorek dat.

Špatně zvolená hodnota k může vést k nesprávné reprezentaci dovedností modelu, například k odhadu s vysokým rozptylem nebo k vysoké systematické chybě.

Volba k je obvykle 5 nebo 10. ale neexistuje žádné formální pravidlo. S rostoucím k se zmenšuje rozdíl ve velikosti trénovacího souboru a podsouborů pro opakované vzorkování. Jak se tento rozdíl zmenšuje, zkreslení metodiky se zmenšuje.

Pokud je zvolena hodnota k, která nerozdělí vzorek dat rovnoměrně, pak jedna skupina bude obsahovat zbytek příkladů. Je vhodnější rozdělit vzorek dat do k skupin se stejným počtem vzorků, aby všechny vzorky odhadů dovedností modelu byly rovnocenné.

II. PRAKTICKÁ ČÁST

5 PŘÍPRAVA NA ANALÝZU METOD REPREZENTACE TEXTU

Pro analýzu vlivu metod reprezentace textu na kvalitu klasifikace byla vyvinuta aplikace v jazyce Python s následujícími komponentami:

1. vyhledávání textových dat prostřednictvím rozhraní API
2. předzpracování přijatých textových dat
3. analýza získaného korpusu dat
4. vyvažování dat a generování syntetických dat
5. reprezentace textů pomocí použitých metod
6. klasifikace textů
7. vizualizace výsledků

5.1 Vytvoření datového korpusu

Pro klasifikační úlohu byly vybrány texty českých písní. Cílem klasifikace je žánr dané písně, a to hip-hop, rock a pop. Data pro vytvoření datového korpusu byla převzata z webových stránek Musixmatch (https://www.musixmatch.com), které jsou jednou z největších platforem, jejichž prostřednictvím uživatelé vyhledávají a sdílejí texty písní.

Tato platforma poskytuje API pro vyhledávání písní podle autora, alba atd. Bezplatná verze API má však svá omezení, a sice:

1. Maximální limit je až 2 tisíce volání API za den.
2. K dispozici je pouze 30 % textů písní

Pro automatický sběr písní pomocí této platformy byl napsán speciální skript, který vyhledá hudebníka (nebo hudební skupinu) podle jména nebo názvu a poté stáhne a uloží celou diskografii daného interpreta do souboru csv.

Po předběžném vyčištění dat, jako je: odstranění duplicit, odstranění písní v cizích jazycích nebo s velkým množstvím cizích slov a odstranění písní s velmi malým počtem slov, byla velikost korpusu textových dat 3486 písní.

5.2 Předzpracování textových dat

Pro úlohy klasifikace textu je velmi důležité předzpracování dat. Předzpracování připravené sady písní se skládalo z následujících kroků:

1. Odstranění interpunkčních znamének
2. Převod slov na malá písmena
3. Odstranění číselných údajů
4. Odstranění stop slov - seznam stop slov pro češtinu byl převzat z repozitáře GitHub s názvem stopwords-cs (https://github.com/stopwords-iso/stopwords-cs)
5. Lemmatizace textu - tento krok předzpracování byl proveden pomocí knihovny třetí strany Spark NLP (https://sparknlp.org/2020/05/05/lemma_cs.html)

5.3 Analýza výsledného souboru textových dat

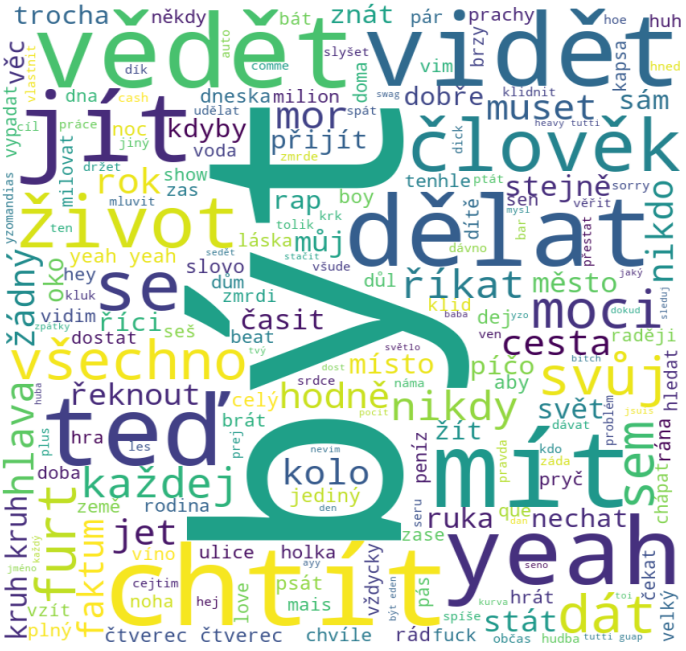
Aby bylo možné určit další kroky při práci s textovými daty, je třeba analyzovat stávající datový korpus.

Po předběžném zpracování dat, datový korpus má následující charakteristiky:

1. Průměrná délka jednoho textu je 43,99 slova:
    a. Pro žánr hip-hop je 71,72 slova.
    b. Pro žánr rock je 35,48 slova.
    c. Pro žánr pop je 35,0 slova.
2. Maximální délka textu:
    a. Pro žánr hip-hop je 202 slova.
    b. Pro žánr rock je 141 slovo.
    c. Pro žánr pop je 131 slovo.
3. Rozdělení dat podle tříd:
    a. Pro žánr hip-hop je 838 textů.
    b. Pro žánr rock je 1159 textů.
    c. Pro žánr pop je 1489 textů.

Mraky slov byly vytvořeny pro podrobnější analýzu původního datového korpusu. Vytváření slovního mraku pro každý žánr písní nám umožňuje vyhodnotit, která slova jsou pro každý žánr specifická a která jsou společná pro všechny zvažované žánry.

Sestavený mrak slov pro žánr hip-hop je znázorněn na obrázku 13.



Obr. 13. Mrak slov pro žánr hip-hop

Z výše uvedeného mraku slov je patrné, že žánr hip-hop charakterizují slova jako "být", „vědět“, „vidět“, „teď“, „život“, „člověk“, „chtít“ atd.

Mrak slov pro žánr rock je znázorněn na obrázku 14

Obr. 14. Mrak slov pro žánr rock

Z výše uvedeného mraku slov je patrné, že žánr rock charakterizují slova jako "mít", „vědět“, „sám“, „teď“, „dát“ atd. Na rozdíl od hiphopového žánru jsou však slova jako „vidět“ nebo „člověk“ mnohem vzácnější.

Na obrázku 15 je mrak slov pro žánr pop.

Obr. 15. Mrak slov pro žánr pop

V tomto žánru jsou nejčastější slova „láska“, "vědět", "jít", "chtít". Stojí za zmínku, že slovo „vědět“ se často vyskytuje v žánrech pop a rock a o něco méně často v žánru hip-hop.

Analýza datového korpusu ukázala, že v datech existuje nerovnováha. Ta se projevuje jak v rozdílném počtu příkladů pro jednotlivé žánry, tak ve vnitřních charakteristikách textů, jako je průměrná délka textu nebo maximální délka textu písně.

Sestavení mraku slov pro každou třídu ukázalo, že ačkoli se v ní pro každý žánr vyskytují slova nejčastěji, většina slov se v každém žánru používá a překrývá, což znamená, že je třeba vzít v úvahu kontext, v němž jsou slova použita.

5.4 Parametry použitých klasifikátorů

K porovnání vlivu různých metod reprezentace textu na kvalitu klasifikace byly použity následující klasifikátory:

1. SVM s jádrem RBF (Radial basis function).
2. Metoda Random Forest s 200 stromy.

3. Vícevrstvý perceptron (Multilayer perceptron) s tisícem neuronů na skryté vrstvě.
4. Konvoluční neuronová síť (Convolutional neural network) s následujícími funkcemi:
    a. Konvoluční vrstva s konvolučním jádrem o velikosti 5 a počtem výstupních filtrů v konvoluci rovným 128.
    b. Vrstva Max Pooling.
    c. Plně koherentní skrytá vrstva s 80 neurony.

Tyto parametry klasifikátorů byly získány pomocí křížové validace na souboru trénovacích dat.

5.5 Vytváření syntetických dat

Původní soubor dat má dva problémy. Prvním problémem je silná nevyváženost počtu příkladů jednotlivých tříd a druhým je malý vzorek textů. K vyřešení těchto problémů byla použita syntetická data.

Metoda vytváření syntetických dat spočívá v použití původního textového korpusu k vytvoření modelu Word2Vec. Tento model slouží k nalezení nejbližšího (ve vektorové reprezentaci) slova k danému slovu.

K vytvoření syntetického textu se používá následující algoritmus:

1. Pro každý text v datovém korpusu se náhodně vybrat token z tohoto textu.
2. Nalezení nejbližšího tokenu ve vektorové reprezentaci daného slova pomocí dříve natrénovaného modelu.
3. Nahradit vybraný token nalezeným blízkým tokenem.
4. Po rozdělení původního souboru dat na trénovací a testovací soubor dat a přidání syntetických dat k trénovacímu souboru dat, tak získáme soubory dat s následujícími charakteristikami:
    a. Velikost trénovacího datasetu byla 7862 (hip-hop - 2748, rock - 2736, pop - 2378).
    b. Velikost testovacího souboru dat byla 20 % původního souboru dat - 698 (hip-hop - 151, rock - 247, pop - 300).

Vyvážení dat pomocí generování syntetických dat neposkytlo «dokonalou rovnováhu», a proto byla pro trénovací soubor dat dodatečně použita metoda vyvážení dat Random Oversampling.

5.6 Datové struktury používané pro trénování modelu

Jedním z důležitých aspektů práce s daty je jejich transformace do formátu, se kterým může pracovat klasifikátor. Zatímco v případě metod Bag of Words a TF-IDF jsou textová data transformována do dvourozměrné matice, v případě metod Word Embedding jako jsou GloVe, FastText a Word2Vec, jsou vstupními daty pro klasifikaci trojrozměrné tenzory.

Mezitím klasifikační metody, jako jsou Random Forest a SVM z knihovny scikit-learn (https://scikit-learn.org), stejně jako vícevrstvý perceptron implementovaný pomocí knihovny Keras (https://keras.io), mohou zpracovávat vstupní data reprezentovaná jako dvourozměrná matice. Naopak konvoluční neuronová síť, která je rovněž implementována pomocí knihovny Keras, vyžaduje vstupní hodnoty ve formě trojrozměrného tenzoru.

K převodu 3D tenzoru na 2D matici byl použit následující algoritmus:

1. Všechny slovní vektory v textu jsou spojeny do 1 příznakového vektoru, takže délka tohoto vektoru je součinem délky textu a délky vektoru, kterým je reprezentován každý token.
2. Protože získaný příznakový vektor je několikrát větší než počet vzorků dat, použije se ke snížení dimenzionality dat algoritmus PCA.
3. Získaný soubor dat se použije pro klasifikaci dat.

5.7 Transformace textových dat a vytvoření Embedding matice

Při klasifikaci textů pomocí vektorové reprezentace slov je třeba provést následující operace:

1. Provést tokenizaci textového korpusu.
2. Normalizovat každý textový korpus na stejnou velikost.
3. Vytvořit Embedding matici.

Tokenizace umožňuje reprezentovat každý text v korpusu jako posloupnost minimálních textových jednotek, tj. tokenů. Každému jedinečnému tokenu je pak přiřazen index. Indexování se provádí na základě trénovacího souboru dat. V případě, že se v testovacím souboru dat vyskytne dosud neznámý token, bude jeho index nulový.

Aby vstupní data pro klasifikaci měla jednotný formát, je nutné provést normalizaci textů korpusu, jinými slovy uvést všechny texty do jednotné délky. Jako jednotná délka byla zvolena délka rovná 202, což je maximální délka textů žánru hip-hop.

K transformaci vstupních dat a mapování každého slova na jeho vektorovou reprezentaci pro konvoluční neuronovou síť se používá Embedding matice. Algoritmus pro vytvoření vkládací matice je následující:

1. Vytvoření matice o velikosti $(N + 1) \times K$, kde N je velikost slovníku, K je délka vektoru odvozená z Embedding modelu reprezentace textových dat (GloVe, FastText nebo Word2Vec).
2. Pro každý token získejte jeho vektorovou reprezentaci pomocí modelu Embedding a podle indexu přidejte tuto hodnotu do matice Embedding.
3. Tato matice je vytvořena z trénovacího souboru dat a používá se jako vstupní vrstva pro trénovací i testovací soubor dat.
4. Pokud se v testovací datové sadě vyskytne nový token, jehož vektorová reprezentace se nenachází v matici Embedding, bude tomuto tokenu jako vektorová reprezentace přiřazen nulový vektor o dimenzi K.

6 ANALÝZA METOD REPREZENTACE TEXTU

Různé metody reprezentace textu mají různý vliv na kvalitu klasifikace. Jako klasifikátory byly vybrány SVM, Random Forest, CNN a MLP. Uvažovány byly následující metody reprezentace textu:

1. Bag of Words
2. TF-IDF
3. Word2Vec
4. GloVe
5. FastText
6. Doc2Vec

Knihovna Gensim byla použita k vytváření vektorových reprezentací slov a dokumentů, jako Word2Vec, FastText a Doc2Vec . K implementaci metod BoW a TF-IDF byla použita knihovna scikit-learn. Pro vektorovou reprezentaci slov pomocí metody GloVe byla použita implementace GloVe, která uvedená v GitHub repozitáři nlp-paper-implementation (https://github.com/pengyan510/nlp-paper-implementation/tree/master).
Jako měřítko kvality klasifikace byla použita míra f-score s „macro“ agregací, protože tato agregace je vhodnější pro hodnocení nevyváženého souboru dat. [27]

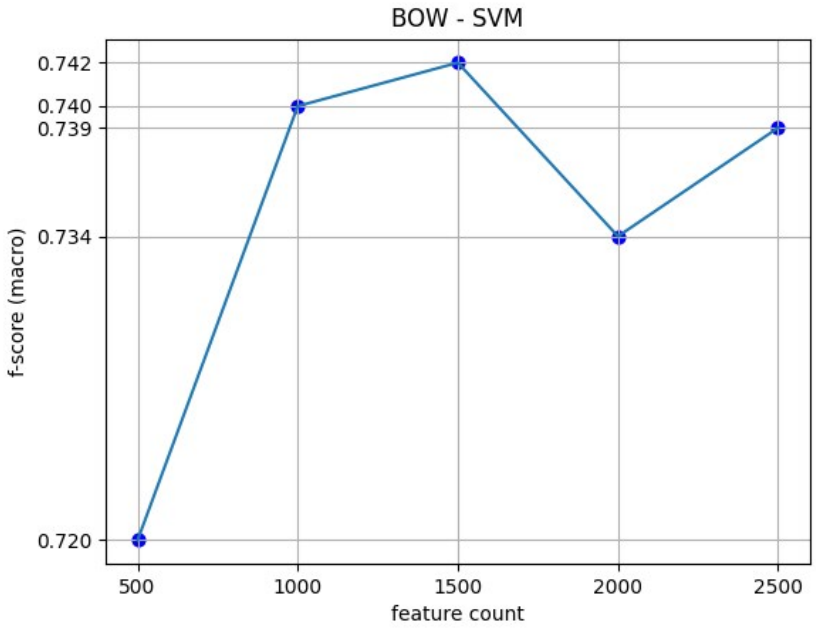
6.1 Analýza metody Bag of Words

Tato metoda reprezentace textu využívá celý slovník vyskytující se v korpusu tokenů, ale velikost slovníku je desetinásobkem velikosti datového korpusu pro trénování klasifikátoru, což negativně ovlivňuje rychlost trénování i kvalitu klasifikace.

Proto byla použita pouze nejfrekventovanější slova. Byly zvoleny následující počty nejfrekventovanějších slov: 500. 1000. 1500. 2000 a 2500.

Klasifikace textů pomocí SVM

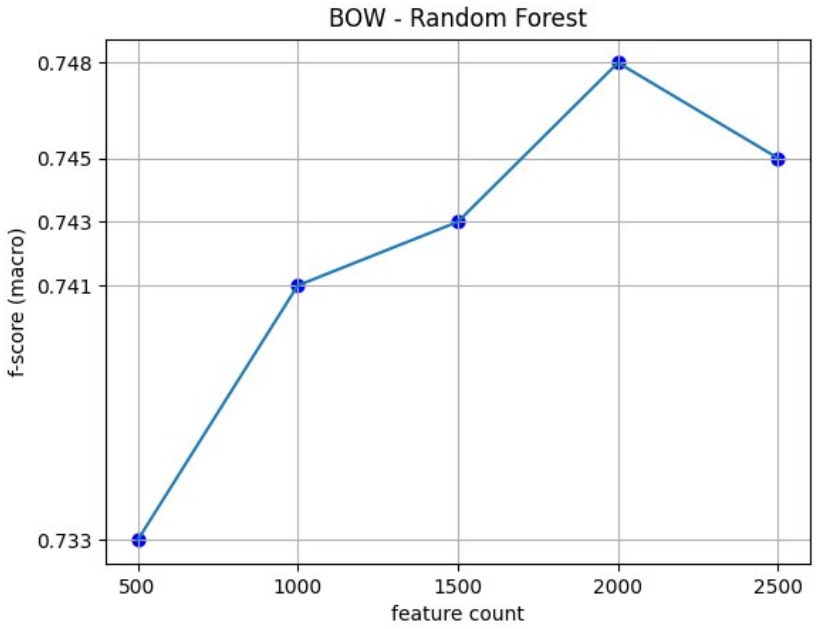
Analýza vlivu metody reprezentace textu Bag of Words na kvalitu klasifikace pomocí metody SVM ukázala, že konstantní zvyšování počtu atributů (často se vyskytujících tokenů v datovém korpusu) nevede ke zvýšení kvality klasifikace. f-score se zvyšuje s rostoucím počtem atributů a dosahuje maximální hodnoty 0,742 při počtu atributů rovném 1500. ale po označení 1500 atributů tato míra mírně klesá, ale. Minimální hodnota přesnosti f-score získaná klasifikací pomocí metody SVM je 0,720 při počtu atributů rovném 500. Graf změny f-score v závislosti na počtu atributů je znázorněn na obrázku 16.



Obr. 16. Graf změny kvality klasifikace SVM při použití metody BoW

Klasifikace textů pomocí Random Forest

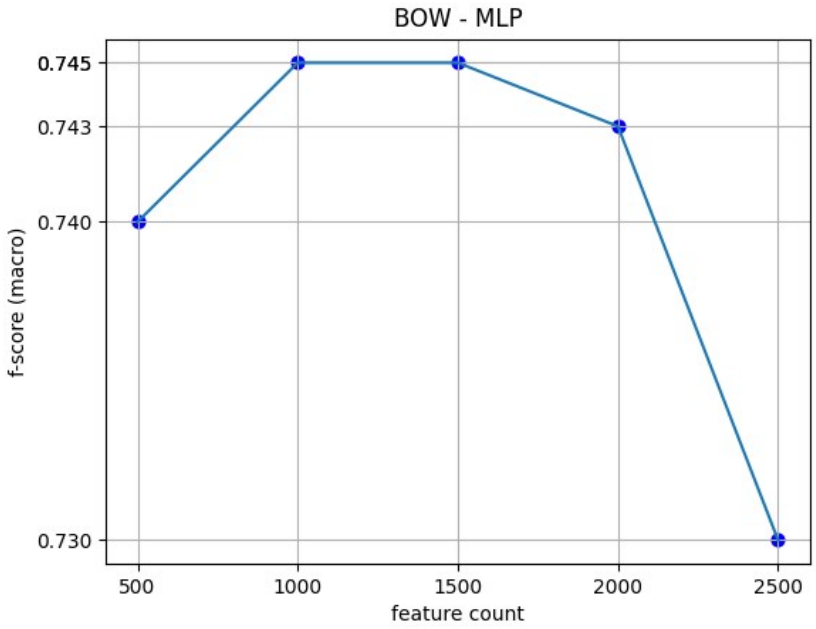
Použití metody reprezentace textu Bag of Words a klasifikace Random Forest ukázalo, že f-score dosahuje maximální hodnoty 0,748 při počtu atributů 2000. nicméně lze pozorovat, že stejně jako v případě klasifikace SVM se po 1500 atributech f-score téměř nemění k lepšímu a po 2000 atributech f-score klesá. Minimální f-score je 0,733 při 500 atributech. Graf změny f-score v závislosti na počtu atributů je znázorněn na obrázku 17.



Obr. 17. Graf změny kvality klasifikace Random Forest při použití metody BoW

Klasifikace textů pomocí MLP

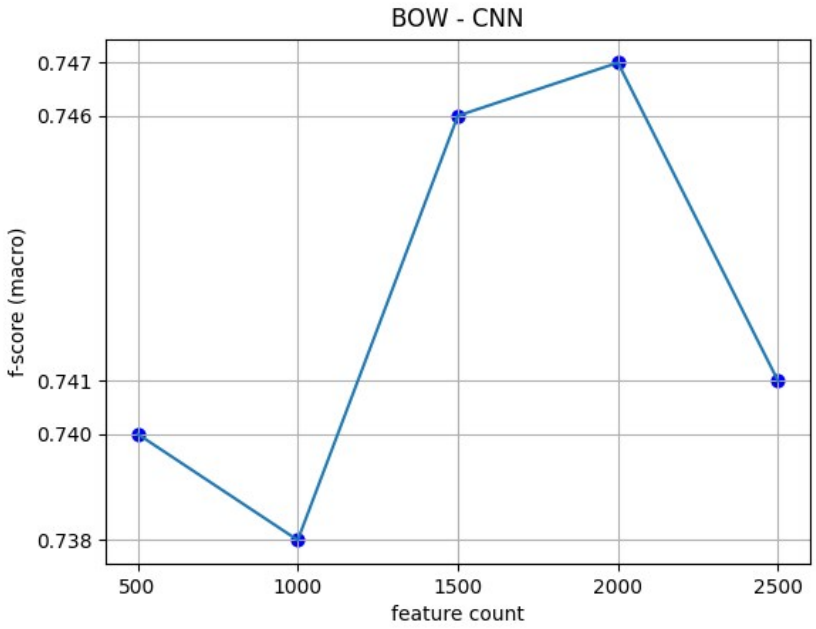
Analýza vlivu metody reprezentace textu BoW na kvalitu klasifikace pomocí neuronové sítě MLP s tisícem neuronů ve skryté vrstvě ukázala, že f-score roste s rostoucím počtem atributů a dosahuje maximální hodnoty 0,755 při počtu atributů rovném 1000. avšak lze si všimnout, že po 1000 atributech f-score klesá a dosahuje minimální hodnoty, která je 0,73, při počtu atributů rovném 2500. Graf změny f-score v závislosti na počtu atributů je znázorněn na obrázku 18.



Obr. 18. Graf změny kvality klasifikace MLP při použití metody BoW

Klasifikace textů pomocí CNN

Použití metody reprezentace textu BoW a klasifikace pomocí konvoluční neuronové sítě ukázalo, že f-score se zvyšuje s rostoucím počtem atributů a dosahuje maximální hodnoty 0,747 při počtu atributů rovném 2000. nicméně je vidět, že po 1500 atributech se přesnost f-score mírně zvyšuje a při 2500 atributech nabývá hodnota jedné z minimálních hodnot rovné 0,741 z předloženého vzorku atributů rovného 2500. Minimální hodnota přesnosti f-score získaná klasifikací pomocí konvoluční neuronové sítě je 0,738 při počtu atributů rovném 1000. Graf změny f-score v závislosti na počtu atributů je znázorněn na obrázku 4.



Obr. 19. Graf změny kvality klasifikace CNN při použití metody BoW

Shrnutí výsledků

Provedením experimentů s různými počty atributů a různými klasifikátory lze dospět k závěru, že průměrné f-score pro všechny klasifikátory bylo 0,74. Nejlepší kvality klasifikace bylo dosaženo při použití počtu atributů 1500-2000. Z uvažovaných klasifikátorů bylo maximální f-score dosaženo při použití klasifikátoru Random Forest s průměrným f-score 0,742. Ostatní klasifikátory se však v průměru mírně liší (SVM - 0,735, MLP - 0,74, CNN-0,742). Z toho lze vyvodit závěr, že tento způsob reprezentace textu lze použít pro libovolné klasifikační modely.

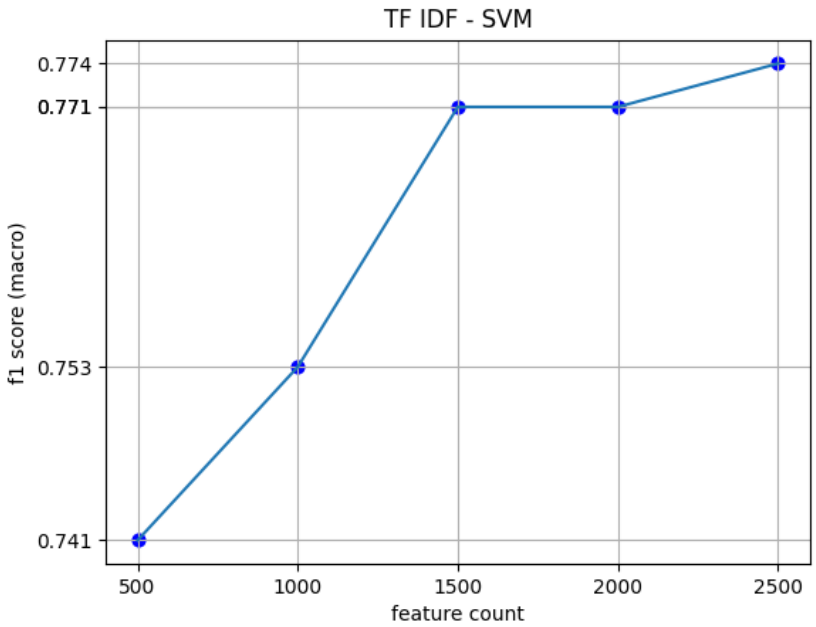
6.2 Analýza metody TF-IDF

Metoda TF-IDF, stejně jako BoW, využívá celý slovník slov vyskytujících se v korpusu. Při analýze touto metodou byl použit podobný počet nejfrekventovanějších slov, a to 500. 1000. 1500. 2000. 2500.

Klasifikace textu pomocí SVM

Analýza vlivu metody reprezentace textu TF-IDF a klasifikace metodou SVM ukázala, že při počtu atributů 1500 a více je hodnota f-score téměř stejná a je vyšší než 0,771. Maximální hodnota přesnosti je 0,774 při počtu atributů rovném 2500. Minimální hodnota f-score

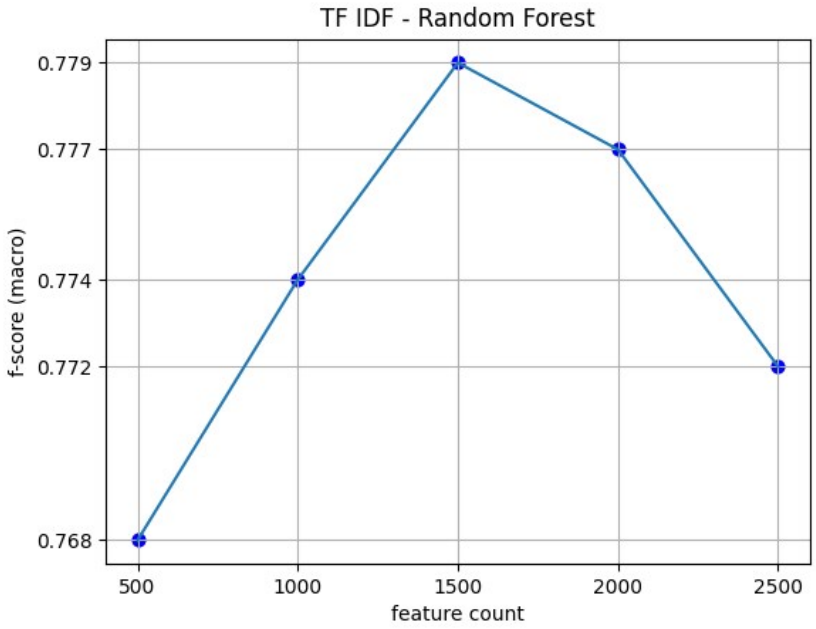
získaná při klasifikační metodě SVM je 0,741 a počet nejčastěji používaných slov je 500. Graf změny f-score v závislosti na počtu atributů je znázorněn na obrázku 20.



Obr. 20. Graf změny kvality klasifikace SVM při použití metody TF-IDF

Klasifikace textu pomocí Random Forest

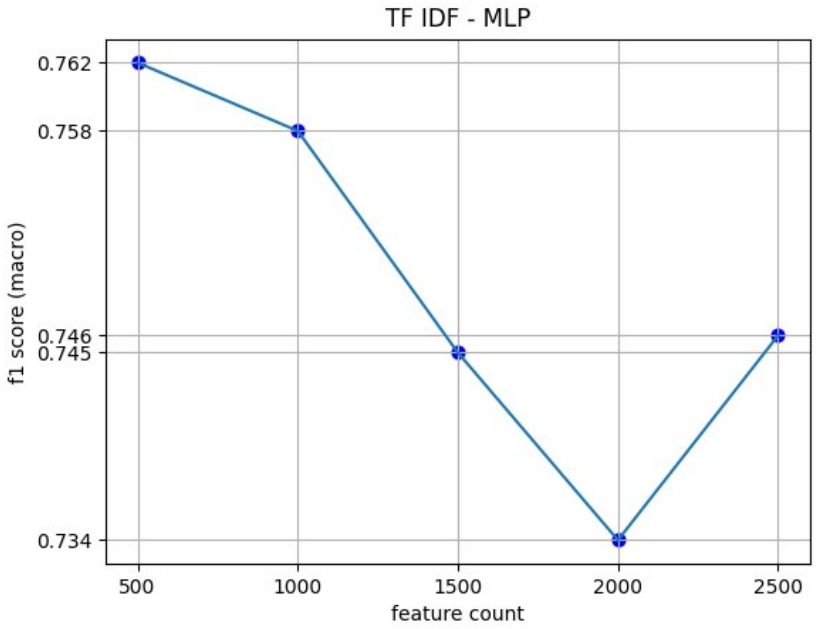
Metoda TF-IDF a klasifikace Random Forest ukázaly, že maximální hodnota metriky f-score je dosažena při počtu příznaků 1500 a činí 0,779. Následné zvyšování počtu rysů vede k poklesu f-score a činí 0,777 a 0,772 pro počet rysů 2000. resp. 2500. Minimální hodnota f-score je 0,768 při počtu znaků 500. Graf změny f-score v závislosti na počtu atributů je znázorněn na obrázku 21.



Obr. 21. Graf změny kvality klasifikace Random Forest při použití metody TF-IDF

Klasifikace textu pomocí MLP

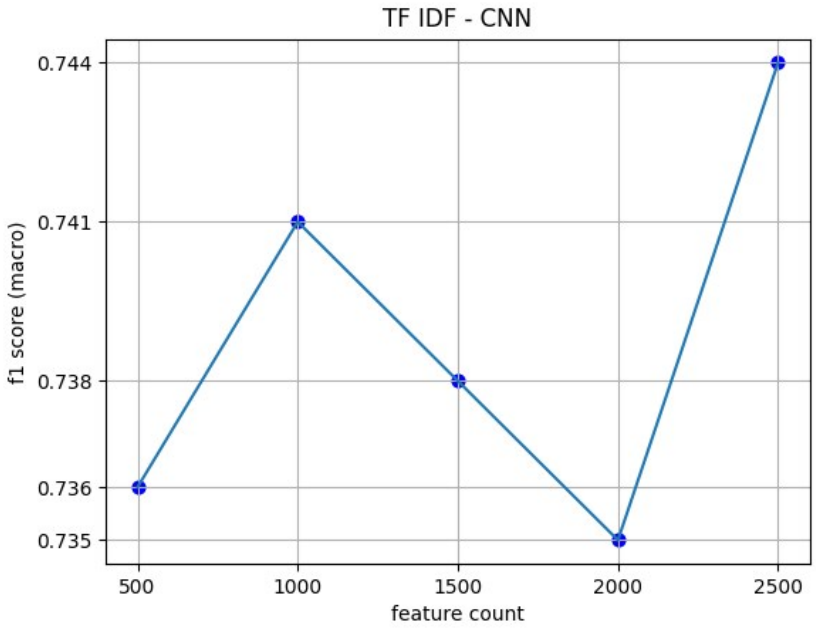
Na základě výsledků analýzy metody reprezentace textu TF-IDF na kvalitu klasifikace pomocí neuronové sítě MLP lze konstatovat, že f-score se s rostoucím počtem atributů nezvyšuje a dosahuje maximální hodnoty 0,762 při počtu atributů rovném 500. další zvyšování počtu atributů má negativní vliv na kvalitu klasifikace a dosahuje minimální hodnoty, která je 0,734 při počtu atributů rovném 2000. Graf změny f-score v závislosti na počtu atributů je znázorněn na obrázku 22.



Obr. 22. Graf změny kvality klasifikace MLP při použití metody TF-IDF

Klasifikace textu pomocí CNN

Podle výsledků analýzy vlivu metody reprezentace textu TF-IDF na kvalitu klasifikace pomocí konvoluční neuronové sítě lze konstatovat, že f-score dosahuje nejvyšší hodnoty 0,744, pokud je počet rysů roven 2500. Minimální hodnota f-score dosahuje při počtu znaků 2000 a je 0,736. Graf změny f-score v závislosti na počtu příznaků je znázorněn na obrázku 22.



Obr. 23. Graf změny kvality klasifikace CNN při použití metody TF-IDF

Shrnutí výsledků

Analýza metody TF-IDF s různým počtem rysů a s použitím různých klasifikátorů ukazuje, že průměrné f-score pro všechny čtyři uvažované klasifikátory je 0,756. Zároveň tato metoda reprezentace textu vykazuje lepší výsledky při použití klasifikátorů strojového učení, jako jsou Random Forest a SVM s průměrným f-score 0,774 a 0,762, oproti přesnosti získané při klasifikaci pomocí neuronových sítí MLP a CNN s průměrným f-score 0,749 a 0,7388. Nejlepším klasifikátorem při použití metody reprezentace textu TF-IDF byl tedy Random Forest a nejhorším CNN.

6.3 Analýza metody Word2Vec

Metoda Word2Vec představuje každý token jako číselný vektor. Existují dva hlavní přístupy k získání vektorové reprezentace slov pomocí modelu Word2Vec: architektura CBOW a architektura skip-gram. Pro každou z těchto architektur existují parametry, které ovlivňují kvalitu vektorové reprezentace, a tím i kvalitu klasifikace. Hlavními parametry jsou délka vektoru, která odpovídá každému tokenu, a velikost okna.

Vzhledem k tomu, že klasifikátory jako Random Forest, SVM a MLP vyžadují, aby vstupní data byla ve formátu dvourozměrné matice, byl použit algoritmus pro převod trojrozměrného tenzoru na dvourozměrnou matici s počtem sloupců rovným 1500. Tento počet sloupců byl

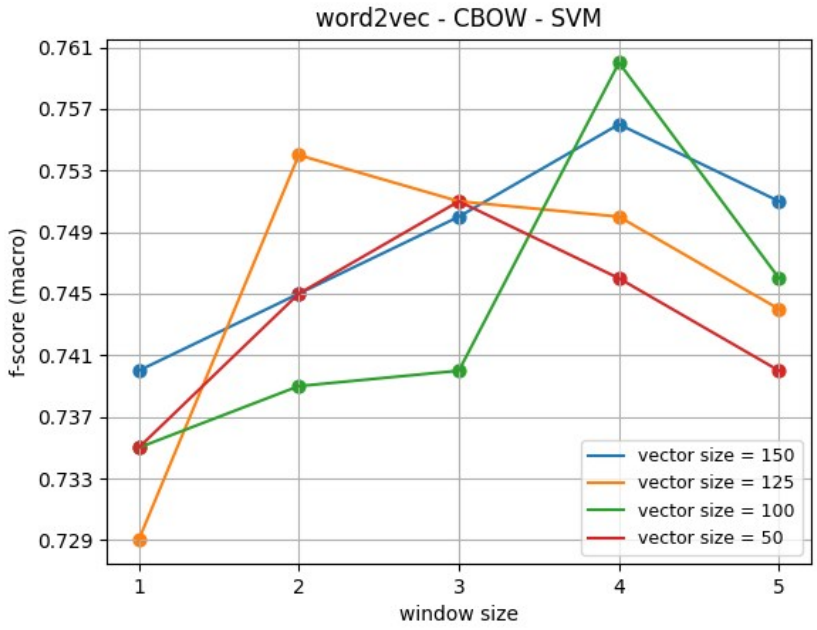
zvolen proto, že při analýze metod Bag of Words a TF-IDF počet znaků rovný 1500 vykazoval jeden z nejlepších ukazatelů kvality klasifikace.

6.3.1 Architektura CBOW

Při analýze metody Word2Vec architektury CBOW byly použity následující délky vektorů: 150. 125, 100. 50 a velikost okna se pohybovala od 1 do 5.

Klasifikace textu pomocí SVM

Analýza vlivu architektury CBOW Word2Vec na kvalitu klasifikace pomocí metody SVM ukázala, že maximální hodnota metriky f-score je dosažena při délce vektoru 100 a velikosti okna 4 a činí 0,76. Minimální hodnota přesnosti f-score je 0,729, přičemž délka vektoru je 125 a velikost okna je 1. Výsledky ukazují, že pro všechny délky vektorů se f-score snižuje při velikosti okna 5. Grafy změny f-score pro jednotlivé délky vektorů v závislosti na velikosti okna jsou uvedeny na obrázku 24.

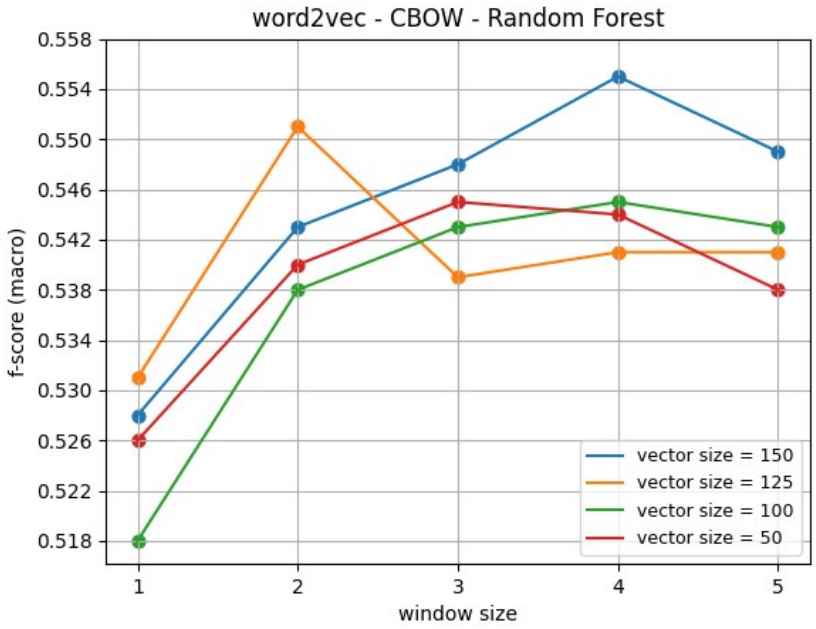


Obr. 24. Grafy změny kvality klasifikace SVM při použití metody Word2Vec (CBOW)

Klasifikace textu pomocí Random Forest

Po analýze vlivu metody Word2Vec architektury CBOW na kvalitu klasifikace pomocí metody Random Forest jsme dospěli k závěru, že maximální hodnota metriky f-score je dosažena při délce vektoru 150 a velikosti okna 4, a to 0,555. Minimální hodnota f-score je

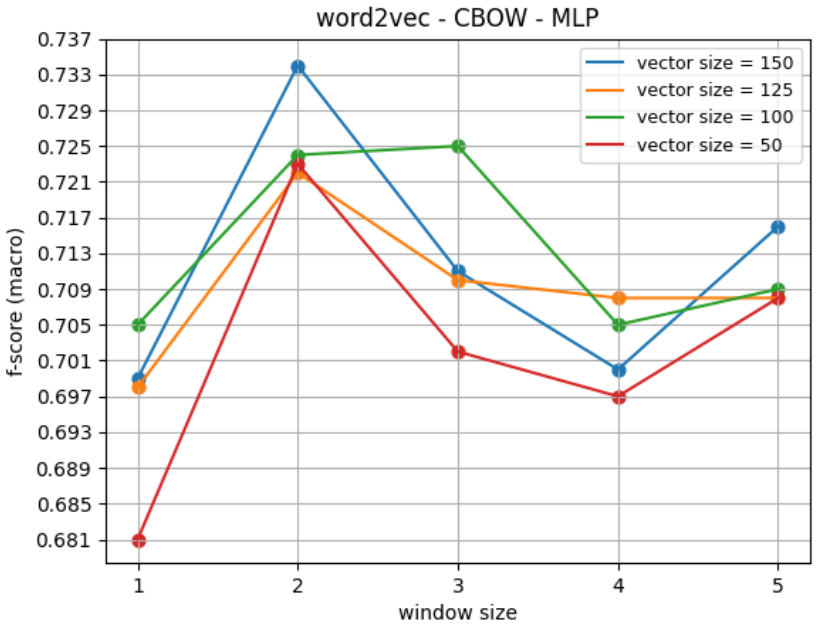
0,518 při délce vektoru rovné 100 a velikosti okna rovné 1. Výsledek analýzy ukazuje, že pro všechny délky vektorů se f-score snižuje při velikosti okna rovné 5. Grafy změn f-score pro jednotlivé délky vektorů v závislosti na velikosti okna jsou uvedeny na obrázku 25.



Obr. 25. Grafy změny kvality klasifikace Random Forest při použití metody Word2Vec (CBOW)

Klasifikace textu pomocí MLP

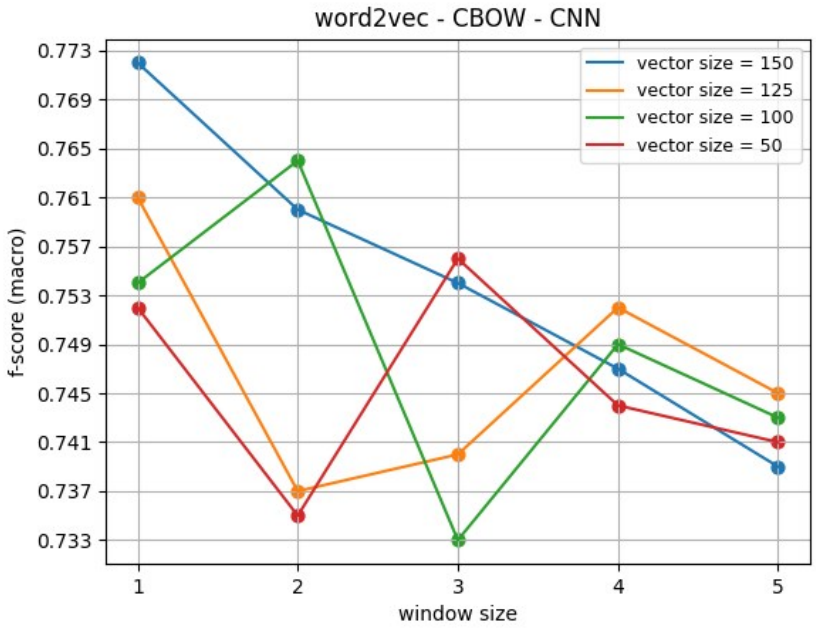
Analýza vlivu metody Word2Vec architektury CBOW na kvalitu klasifikace pomocí neuronové sítě MLP ukázala, že maximální hodnota metriky f-score je dosažena při délce vektoru 150 a velikosti okna 2 a činí 0,733. Minimální hodnota přesnosti f-score je 0,681 při délce vektoru rovné 100 a velikosti okna rovné 1. Ostatní hodnoty metriky f-score pro všechny délky vektorů a velikosti oken se pohybují v rozmezí od 0,697 do 0,725 a dosahují maximálních hodnot f-score při velikosti okna rovné 2. Grafy změny f-score pro každou délku vektoru v závislosti na velikosti okna jsou uvedeny na obrázku 26.



Obr. 26. Grafy změny kvality klasifikace MLP při použití metody Word2Vec (CBOW)

Klasifikace textu pomocí CNN

Výzkum vlivu metody Word2Vec architektury CBOW na kvalitu klasifikace pomocí konvoluční neuronové sítě ukázal, že maximální hodnota metriky f-score je dosažena pro délku vektoru rovnou 150 a velikost okna rovnou 1 a činila 0,772, přičemž další zvyšování velikosti okna při dané délce vektoru negativně ovlivňuje kvalitu klasifikace. Minimální hodnota přesnosti f-score je 0,733 při délce vektoru rovné 100 a velikosti okna rovné 3. Ostatní hodnoty metriky f-score pro všechny délky vektorů a velikosti oken se pohybují mezi 0,736 a 0,758. Pro všechny délky vektorů se f-score snižuje pro velikost okna rovnou 5. Grafy změny f-score pro každou délku vektoru v závislosti na velikosti okna jsou uvedeny na obrázku 27.



Obr. 27. Grafy změny kvality klasifikace CNN při použití metody Word2Vec (CBOW)

Shrnutí výsledků

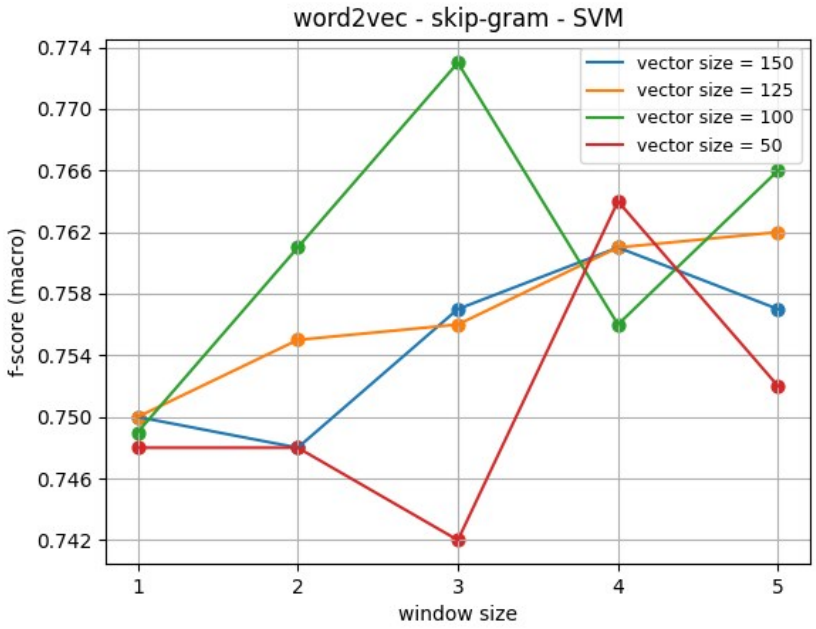
Provedením experimentů s různými kombinacemi délky vektoru a velikosti okna pro metodu Word2Vec architektury CBOW lze dojít k závěru, že průměrné f-score konvoluční neuronové sítě je ve srovnání s ostatními klasifikátory nejlepší a činí 0,749. Zatímco průměr pro SVM je 0,745 a pro MLP je 0,709. Nejhorších výsledků dosahuje algoritmus Random Forest s hodnotou 0,54. Nejčastěji je nejhorší kvality klasifikace dosaženo při délce vektoru rovné 50 nebo velikosti okna rovné 1.

6.3.2 Architektura skip-gram

Při analýze metody Word2Vec architektury skip-gram byly použity následující délky vektorů: 150. 125, 100. 50 a velikost okna se pohybovala od 1 do 5.

Klasifikace textu pomocí SVM

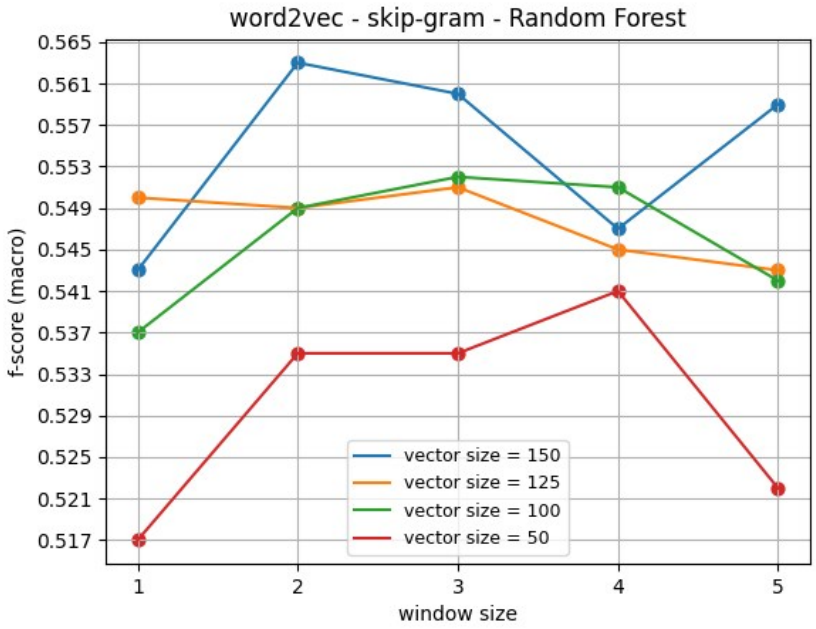
Studie vlivu metody Word2Vec architektury skip-gramů na kvalitu klasifikace pomocí metody SVM ukázala, že maximální hodnota metriky f-score je dosažena při délce vektoru 100 a velikosti okna 3 a činí 0,774. Minimální hodnota metriky f-score je 0,742 při délce vektoru 50 a velikosti okna 3. Ostatní hodnoty metriky f-score pro všechny délky vektorů a velikosti oken jsou mezi 0,748 a 0,766. Grafy změny f-score pro každou délku vektoru v závislosti na velikosti okna jsou znázorněny na obrázku 28.



Obr. 28. Grafy změny kvality klasifikace SVM při použití metody Word2Vec (skip-gram)

Klasifikace textu pomocí Random Forest

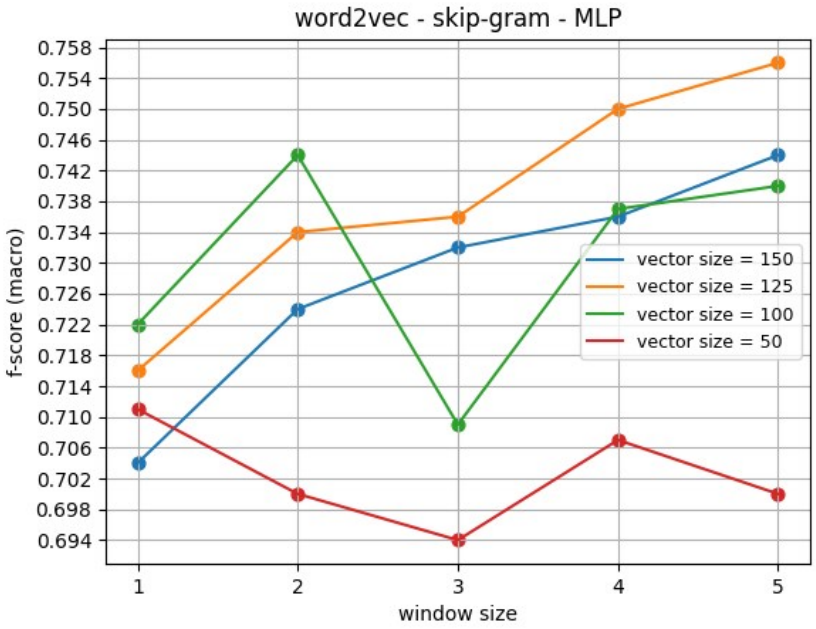
Analýza vlivu metody Word2Vec architektury skip-gramů na kvalitu klasifikace pomocí metody Random Forest ukázala, že maximální hodnota metriky f-score je dosažena při délce vektoru 150 a velikosti okna 2 a je rovna 0,563. Minimální hodnota přesnosti f-score je 0,517 při délce vektoru rovné 50 a velikosti okna rovné 1. Grafy změny f-score pro každou délku vektoru v závislosti na velikosti okna jsou uvedeny na obrázku 29.



Obr. 29. Grafy změny kvality klasifikace Random Forest při použití metody Word2Vec (skip-gram)

Klasifikace textu pomocí MLP

Výzkum vlivu metody Word2Vec architektury skip-gram na kvalitu klasifikace pomocí neuronové sítě MLP ukázal, že maximální hodnota metriky f-score je dosažena při délce vektoru 125 a velikosti okna 2 a činí 0,756. Minimální hodnota přesnosti f-score je 0,694 při délce vektoru rovné 50 a velikosti okna rovné 3. Pro všechny délky vektorů kromě délky vektoru 50 a 100 se f-score zvyšuje s rostoucí velikostí okna. Grafy změny f-score pro každou délku vektoru v závislosti na velikosti okna jsou uvedeny na obrázku 30.

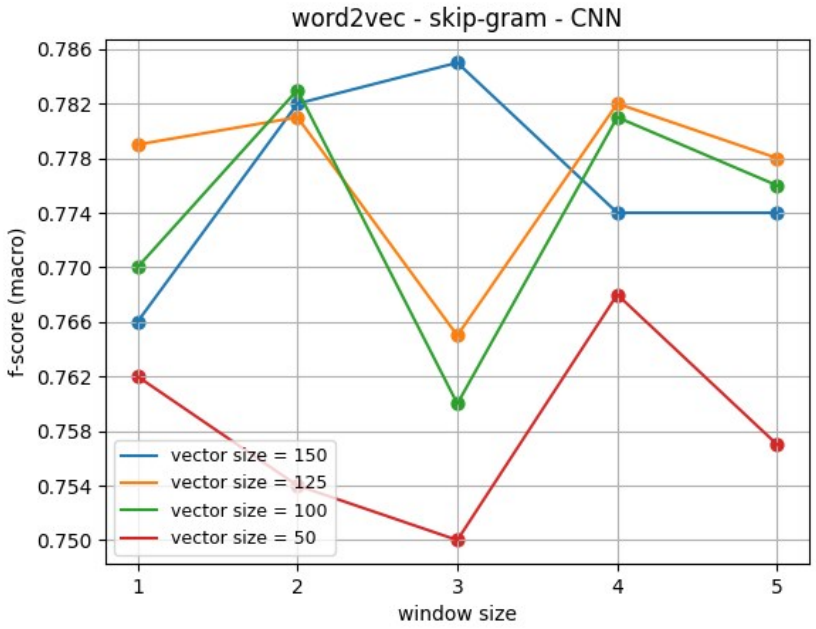


Obr. 30. Grafy změny kvality klasifikace MLP při použití metody Word2Vec (skip-gram)

Klasifikace textu pomocí CNN

Zkoumání vlivu metody Word2Vec architektury skip-gram na kvalitu klasifikace pomocí CNN ukázalo, že maximální hodnota metriky f-score je 0,785 při délce vektoru 150 a velikosti okna 3. Současně pro jiné délky vektorů se stejnou velikostí okna vykazuje f-score pro tyto délky vektorů horší hodnotu.

Téměř maximální hodnoty metriky f-score pro délky vektorů 125 a 100 je dosaženo, když je velikost okna rovna 2. Minimální hodnota metriky f-score je 0,75, když je délka vektoru 50 a velikost okna 3. Grafy změn f-score pro jednotlivé délky vektorů v závislosti na velikosti okna jsou zobrazeny na obrázku 16.



Obr. 31. Grafy změny kvality klasifikace CNN při použití metody Word2Vec (skip-gram)

Shrnutí výsledků

Na základě experimentů s různými kombinacemi délky vektoru a velikosti okna pro metodu Word2Vec architektury skip-gram lze konstatovat, že průměrné f-score konvoluční neuronové sítě je ve srovnání s ostatními klasifikátory lepší a činí 0,771.

Naproti tomu průměr pro SVM je 0,756 a pro MLP je 0,724. Algoritmus Random Forest dosahuje nejhorších výsledků s hodnotou 0,544. Nejčastěji je nejhorší kvality klasifikace dosaženo při délce vektoru 50 nebo velikosti okna 1.

6.4 Analýza metody GloVe

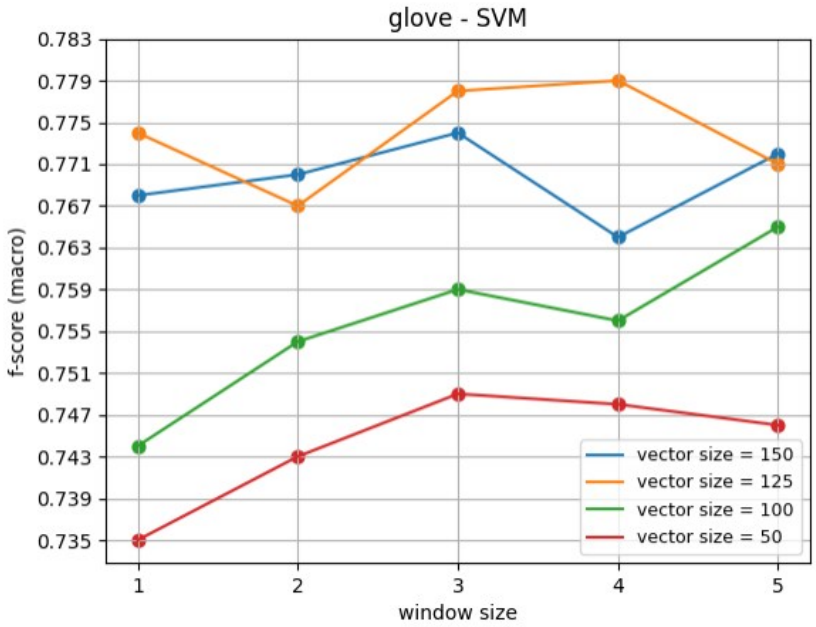
Metoda GloVe, stejně jako metoda Word2Vec, reprezentuje každý token jako číselný vektor pevné délky. Stejně jako metoda Word2Vec má metoda GloVe parametry, které ovlivňují kvalitu reprezentace tokenů jako číselného vektoru, a tedy i kvalitu klasifikace. Prvním z nich je délka vektoru, který odpovídá každému tokenu, a také velikost okna. Při analýze této metody byly zkoumány velikosti délky vektoru rovnající se 150. 125, 100. 50 a velikost okna, která se pohybovala od 1 do 5 bodů.

Při použití metody GloVe se vyskytuje podobný problém jako u metody Word2Vec, klasifikátory jako Random Forest, SVM a MLP vyžadují, aby vstupní data byla ve formátu dvourozměrné matice. Po transformaci trojrozměrného tenzoru na dvourozměrnou matici

byla získána matice s počtem sloupců rovným 1500. Tento počet sloupců byl zvolen proto, že při analýze metod Bag of Words a TF-IDF počet znaků rovný 1500 vykazoval jeden z nejlepších ukazatelů kvality klasifikace.

Klasifikace textu pomocí SVM

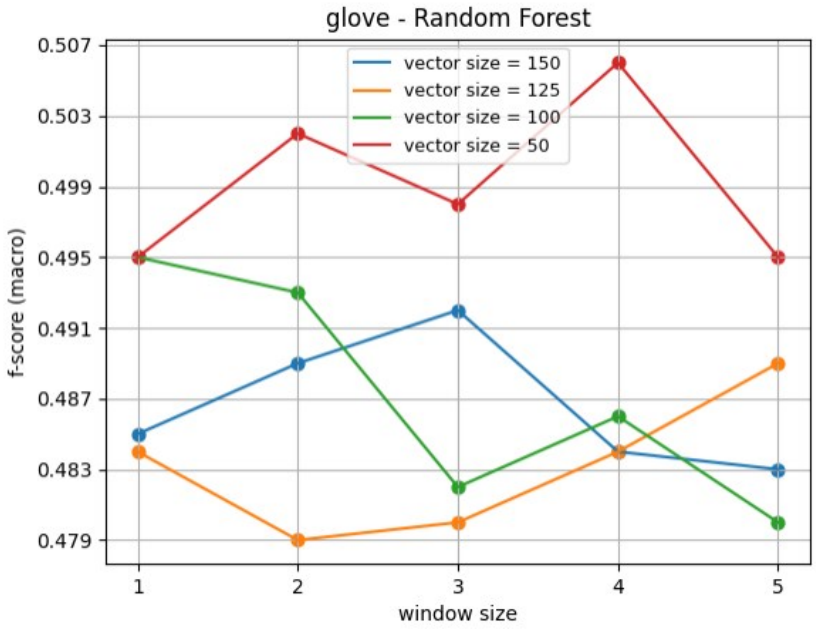
Zkoumání vlivu vektorové reprezentace slov metodou GloVe na kvalitu klasifikace metodou SVM ukázalo, že maximální hodnota metriky f-score je dosažena při délce vektoru 125 a velikosti okna 4 a činí 0,779. S rostoucí velikostí okna se hodnota metriky f-score zvyšuje, ale když je velikost okna větší než 3, kvalita klasifikace buď mírně roste jako v případě délky vektoru 125, nebo klesá jako v případě délky vektoru 50. 100 a 150. Minimální hodnota metriky f-score je 0,735 při délce vektoru rovné 50 a velikosti okna 1. Grafy změny f-score pro každou délku vektoru v závislosti na velikosti okna jsou uvedeny na obrázku 32.



Obr. 32. Grafy změny kvality klasifikace SVM při použití metody GloVe

Klasifikace textu pomocí Random Forest

Analýza vlivu metody vektorové reprezentace slov GloVe na kvalitu klasifikace s metodou Random Forest ukázala, že maximální hodnota metriky f-score je dosažena při délce vektoru 50 a velikosti okna 4 a činí 0,505. Minimální hodnota f-score je 0,479, délka vektoru je 125 a velikost okna je 2. Grafy změny f-score pro každou délku vektoru v závislosti na velikosti okna jsou uvedeny na obrázku 33.

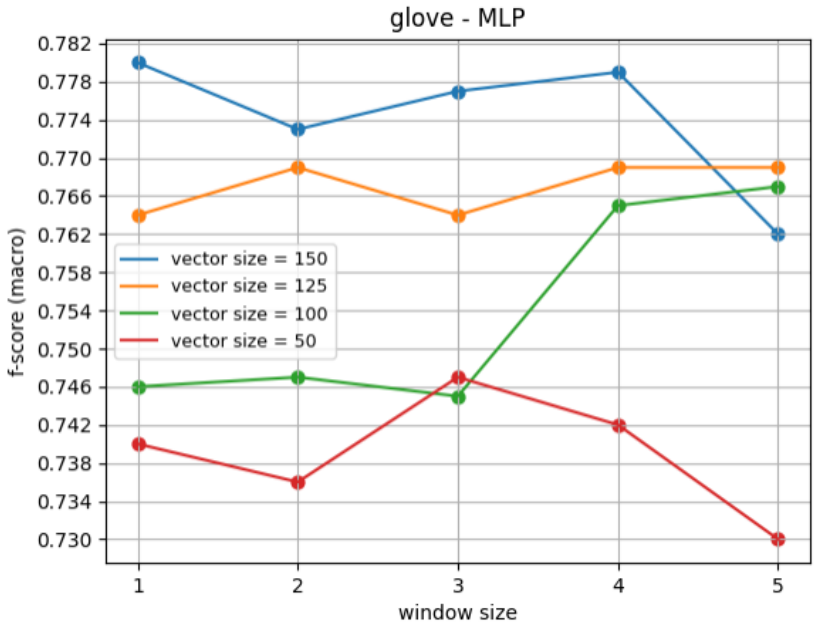


Obr. 33. Grafy změny kvality klasifikace Random Forest při použití metody GloVe

Klasifikace textu pomocí MLP

Výzkum vlivu metody vektorové reprezentace slov GloVe na kvalitu klasifikace pomocí neuronové sítě MLP ukázal, že maximální hodnota metriky f-score je dosažena při délce vektoru 50 a velikosti okna 4 a činí 0,779. Minimální hodnota f-score je 0,730. délka vektoru je 50 a velikost okna je 5.

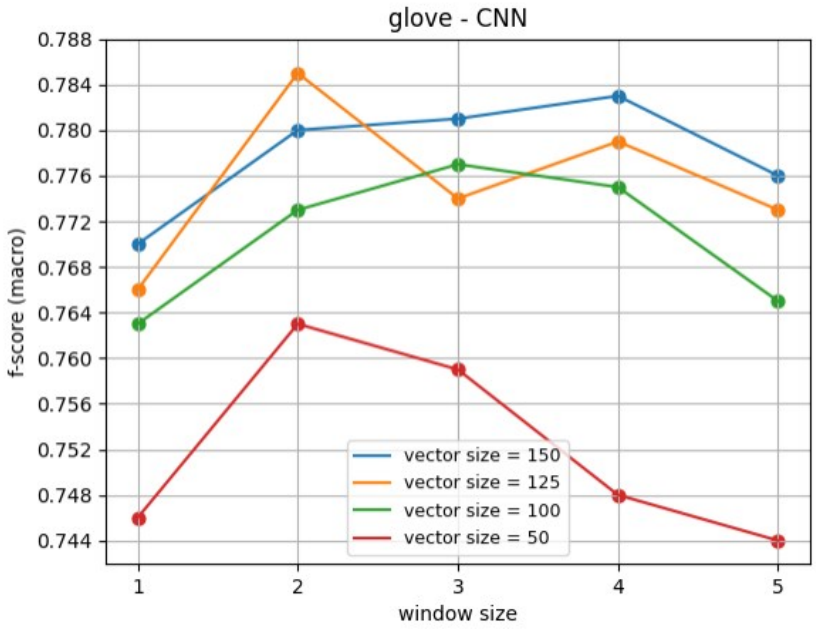
Vektor, jehož délka je 100. se nemění, pokud je velikost okna menší než 4. Při velikosti okna rovné 4 a 5 se kvalita klasifikace výrazně zvyšuje z 0,746 na 0,766. Vektor, jehož délka je 125, se výrazně nemění bez ohledu na velikost okna. f-score se pohybuje mezi 0,763 a 0,769. Grafy změny f-score pro každou délku vektoru v závislosti na velikosti okna jsou uvedeny na obrázku 34.



Obr. 34. Grafy změny kvality klasifikace MLP při použití metody GloVe

Klasifikace textu pomocí CNN

Analýza vlivu metody GloVe na kvalitu klasifikace pomocí CNN ukázala, že maximální hodnota metriky f-score je dosažena při délce vektoru 125 a velikosti okna 2 a činí 0,785. Minimální hodnota přesnosti f-score je 0,744 při délce vektoru rovné 50 a velikosti okna rovné 5. Pro všechny délky vektorů se f-score snižuje, když je velikost okna 5. Grafy změny f-score pro jednotlivé délky vektorů v závislosti na velikosti okna jsou uvedeny na obrázku 35.



Obr. 35. Grafy změny kvality klasifikace CNN při použití metody GloVe

Shrnutí výsledků

Po experimentování s různými kombinacemi délky vektoru a velikosti okna pro metodu GloVe můžeme dojít k závěru, že průměrné f-score konvoluční neuronové sítě je nejlepší ve srovnání s jinými klasifikátory a je 0,769.

Zatímco průměr pro SVM je 0,76 a pro MLP je to 0,758. Nejhorší výsledky má algoritmus Random Forest s hodnotou 0,488. Nejčastěji se nejhorší kvality klasifikace dosahuje s délkou vektoru 50 nebo velikostí okna 1.

6.5 Analýza metody FastText

Metoda FastText, stejně jako metody GloVe a Word2Vec, představuje tokeny jako číselný vektor. Existují dva hlavní přístupy k získání vektorové reprezentace slov pomocí modelu FastText: architektury CBOW a skip-gram. Pro každou z těchto architektur existují parametry, které ovlivňují kvalitu vektorové reprezentace a následně i kvalitu klasifikace. Hlavními parametry jsou délka vektoru, který odpovídá každému tokenu, a také velikost okna.

Při použití metody FastText nastává problém podobný problému metod Word2Vec a GloVe, klasifikátory jako Random Forest, SVM a MLP vyžadují, aby vstupní data byla ve formátu dvourozměrné matice. Po transformaci trojrozměrného tenzoru na dvourozměrnou matici

byla získána matice s počtem sloupců rovným 1500. Tento počet sloupců byl zvolen proto, že při analýze metod Bag of Words a TF-IDF počet znaků rovný 1500 vykazoval jeden z nejlepších ukazatelů kvality klasifikace.

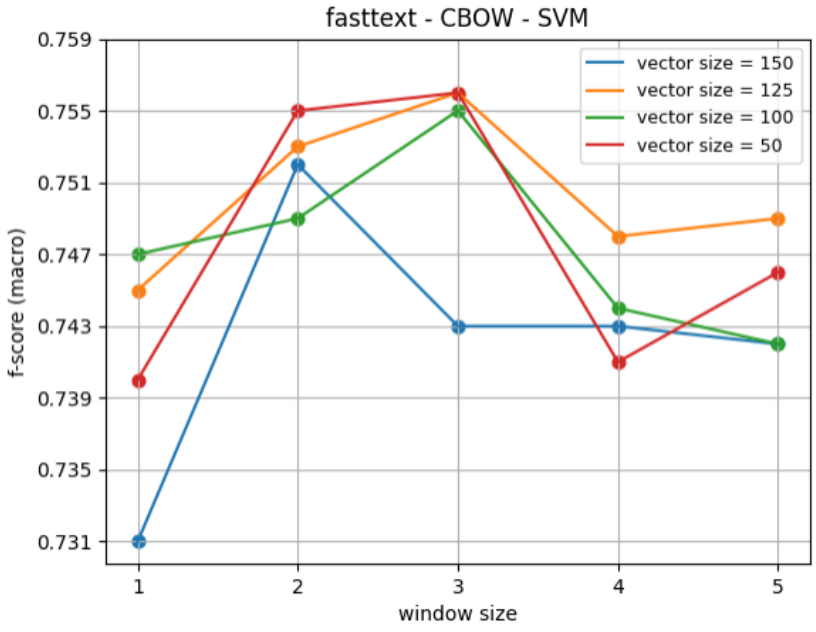
6.5.1 Architektura CBOW

Při analýze metody FastText architektury CBOW byly použity následující délky vektorů: 150. 125, 100. 50 a velikost okna se pohybovala od 1 do 5.

Klasifikace textu pomocí SVM

Zkoumání vlivu metody CBOW s architekturou FastText na kvalitu klasifikace SVM ukázalo, že maximální hodnota metriky f-score je dosažena při délce vektoru 50 a 125 a velikosti okna rovné 3 a činí 0,756. Délka vektoru 100 s velikostí okna 3 se rovněž velmi blížila maximální hodnotě metriky f-score a činila 0,755.

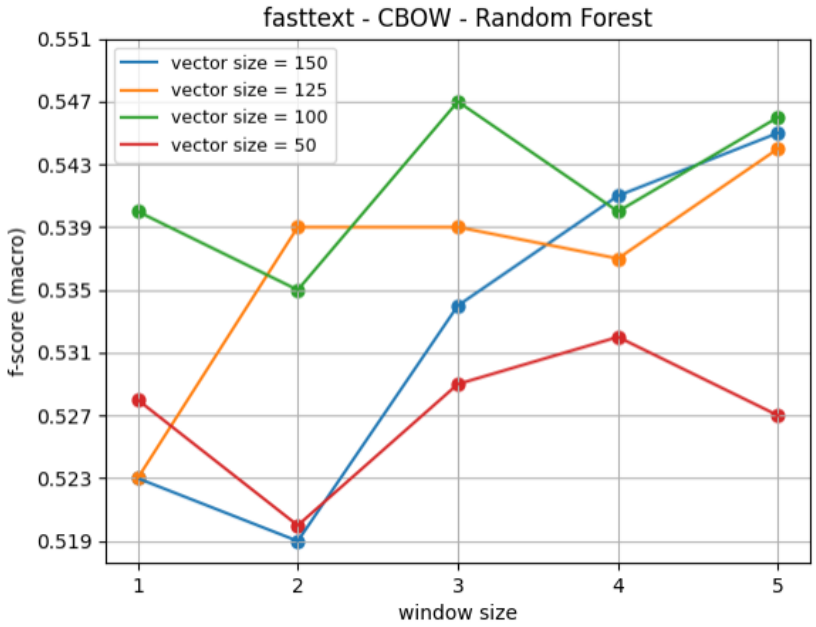
Minimální hodnota přesnosti f-score je 0,731 při délce vektoru 150 a velikosti okna 1. Grafy změny f-score pro každou délku vektoru v závislosti na velikosti okna jsou uvedeny na obrázku 36.



Obr. 36. Grafy změny kvality klasifikace SVM při použití metody FastText (CBOW)

Klasifikace textu pomocí Random Forest

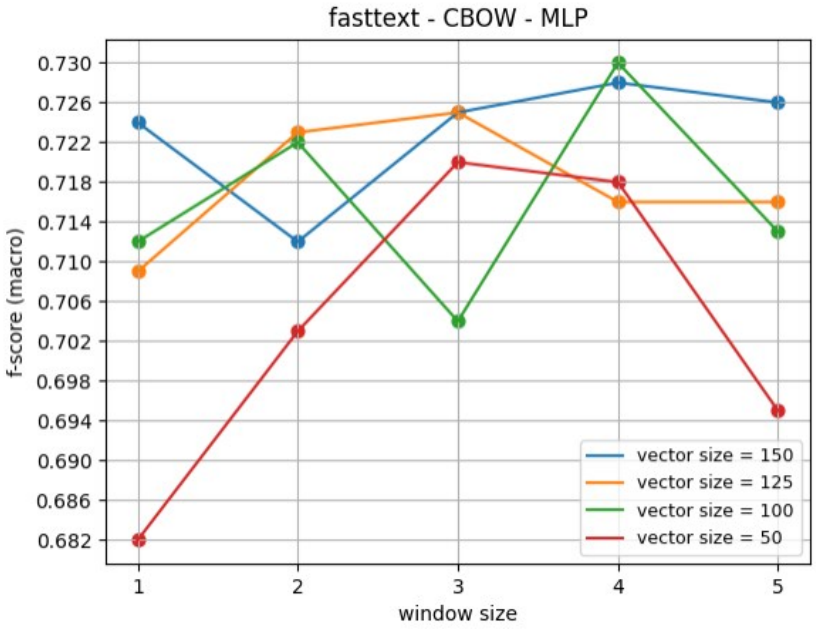
Analýza vlivu metody FastText architektury CBOW na kvalitu klasifikace pomocí metody Random Forest ukázala, že maximální hodnota metriky f-score je dosažena při délce vektoru 100 a velikosti okna 3 a je rovna 0,547. Minimální hodnota f-score je 0,519 při délce vektoru 150 a velikosti okna 2. Grafy změny f-score pro každou délku vektoru v závislosti na velikosti okna jsou uvedeny na obrázku 37.



Obr. 37. Grafy změny kvality klasifikace Random Forest při použití metody FastText (CBOW)

Klasifikace textu pomocí MLP

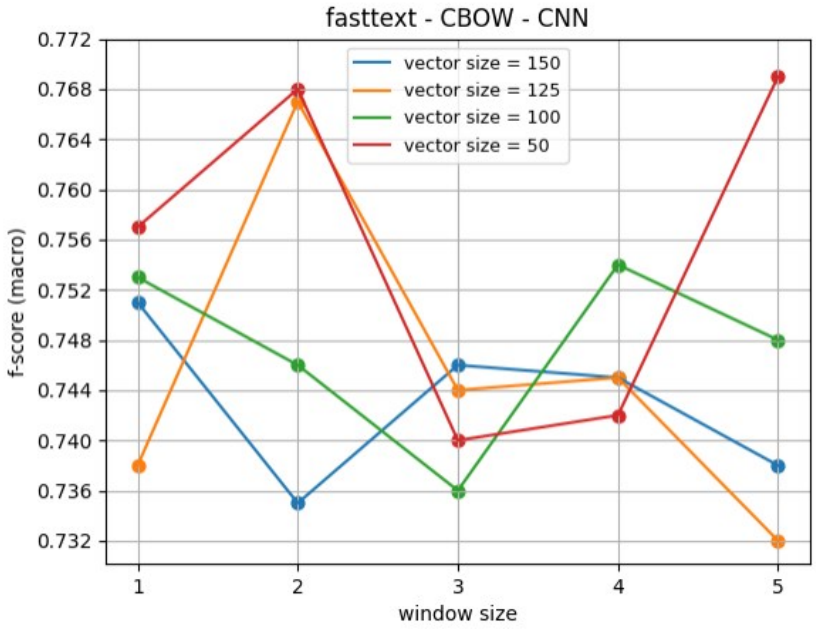
Výzkum vlivu metody FastText architektury CBOW na kvalitu klasifikace pomocí neuronové sítě MLP ukázal, že maximální hodnota metriky f-score je dosažena při délce vektoru 100 a velikosti okna 4 a je rovna 0,73. Minimální hodnota přesnosti f-score je 0,682 při délce vektoru 50 a velikosti okna 1. Pro všechny délky vektorů se f-score snižuje, když je velikost okna 5. Grafy změny f-score pro každou délku vektoru v závislosti na velikosti okna jsou uvedeny na obrázku 38.



Obr. 38. Grafy změny kvality klasifikace MLP při použití metody FastText (CBOW)

Klasifikace textu pomocí CNN

Analýza vlivu metody FastText architektury CBOW na kvalitu klasifikace konvoluční neuronové sítě ukázala, že maximální hodnoty f-score je dosaženo pro délku vektoru rovnou 50 a velikost okna rovnou 5, což je 0,769. Minimální hodnota přesnosti f-score je 0,732 při délce vektoru rovné 100 a velikosti okna rovné 5. Grafy změn f-score pro každou délku vektoru v závislosti na velikosti okna jsou uvedeny na obrázku 39.



Obr. 39. Grafy změny kvality klasifikace CNN při použití metody FastText (CBOW)

Shrnutí výsledků

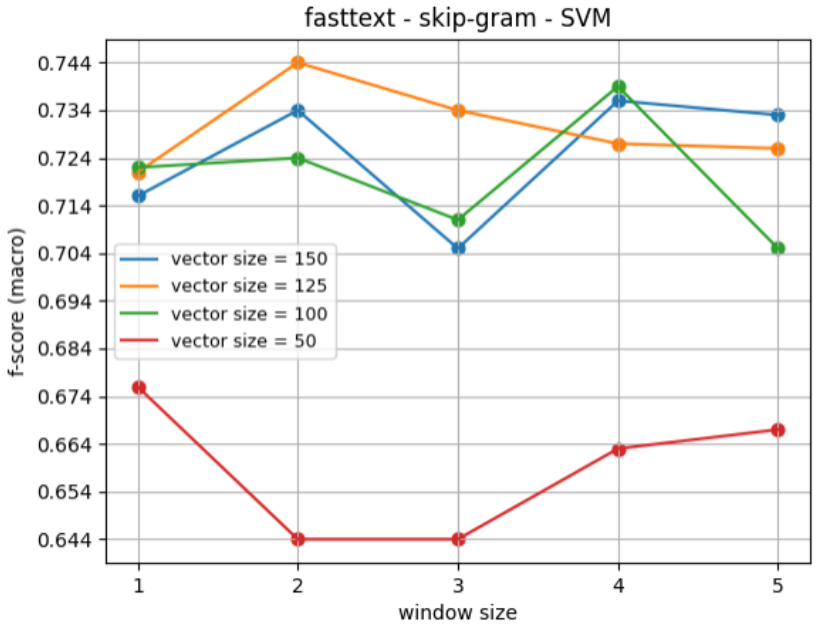
Experimenty provedené s různými kombinacemi délky vektoru a velikosti okna pro metodu FastText architektury CBOW ukazují, že průměrné f-score pro konvoluční neuronovou síť a metodu SVM jsou ve srovnání s ostatními klasifikátory nejlepší a činí 0,748, resp. 0,747. Zatímco průměrné f-score pro MLP je 0,715 a nejhoršího výsledku bylo dosaženo při použití algoritmu Random Forest, a to 0,534.

6.5.2 Architektura skip-gram

Při analýze metody FastText architektury skip-gram byly použity následující délky vektorů: 150. 125, 100. 50 a velikost okna se pohybuje od 1 do 5.

Klasifikace textu pomocí SVM

Analýza vlivu metody FastText architektury skip-gram na kvalitu klasifikace pomocí metody SVM ukázala, že maximální hodnota metriky f-score je dosažena při délce vektoru 125 a velikosti okna 2 a činí 0,744. Minimální hodnota metriky f-score je 0,644 při délce vektoru rovné 50 a velikosti okna 2 a 3. Ostatní hodnoty metriky f-score pro všechny délky vektorů (kromě délky vektoru 50) se pohybují mezi 0,704 a 0,734. Vektor délky 50 vykazuje nejhorší výsledek - f-score dosahuje maxima 0,674 při velikosti okna 1. Grafy změny f-score pro každou délku vektoru v závislosti na velikosti okna jsou uvedeny na obrázku 40.

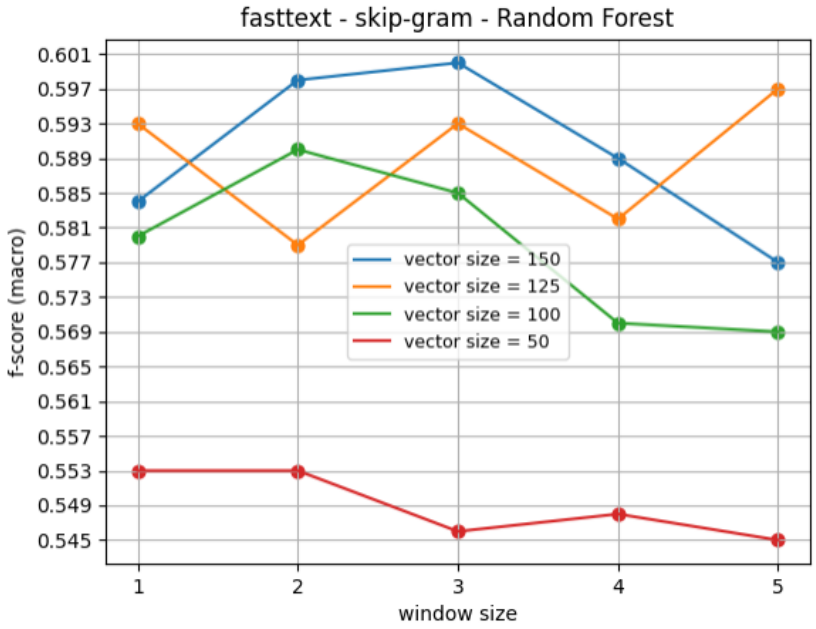


Obr. 40. Grafy změny kvality klasifikace SVM při použití metody FastText (skip-gram)

Klasifikace textu pomocí Random Forest

Zkoumání vlivu metody reprezentace textu FastText architektury skip-gram na kvalitu klasifikace pomocí metody Random Forest ukázalo, že maximální hodnota metriky f-score je dosažena při délce vektoru 150 a velikosti okna 2 a činí 0,563. Minimální hodnota přesnosti f-score je 0,517 při délce vektoru rovné 50 a velikosti okna rovné 1.

Ostatní hodnoty metriky f-score pro všechny délky vektorů (kromě délky vektoru 50) se pohybují v rozmezí od 0,57 do 0,597. Vektor délky 50 vykazuje nejhorší výsledek - f-score dosahuje maximální hodnoty 0,553 při velikosti okna 3. Grafy změny f-score pro jednotlivé délky vektorů v závislosti na velikosti okna jsou uvedeny na obrázku 41.

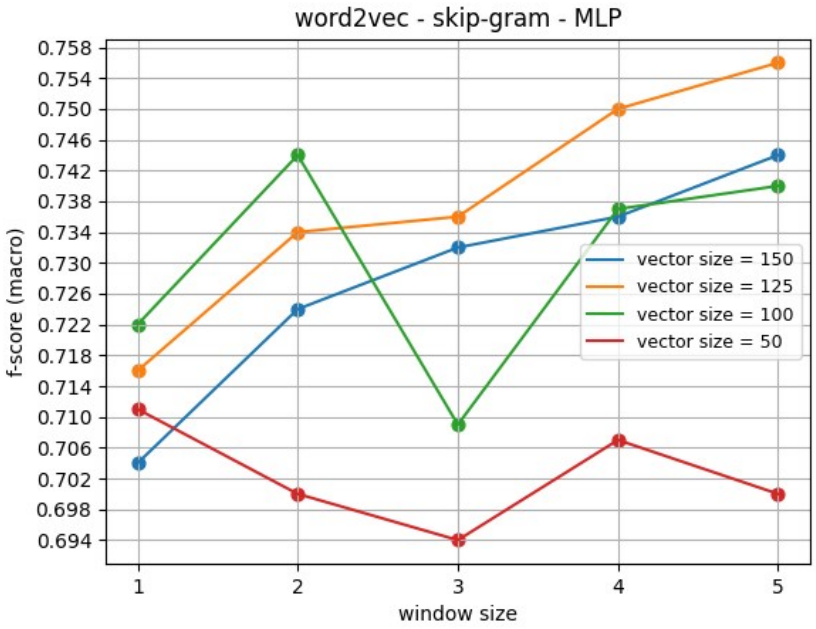


Obr. 41. Grafy změny kvality klasifikace Random Forest při použití metody FastText (skip-gram)

Klasifikace textu pomocí MLP

Analýza vlivu metody reprezentace textu FastText architektury skip-gram na kvalitu klasifikace pomocí MLP ukázala, že maximální hodnota metriky f-score je dosažena při délce vektoru 100 a velikosti okna 2 a činí 0,744. Minimální hodnota přesnosti f-score je 0,694 při délce vektoru rovné 50 a velikosti okna rovné 3.

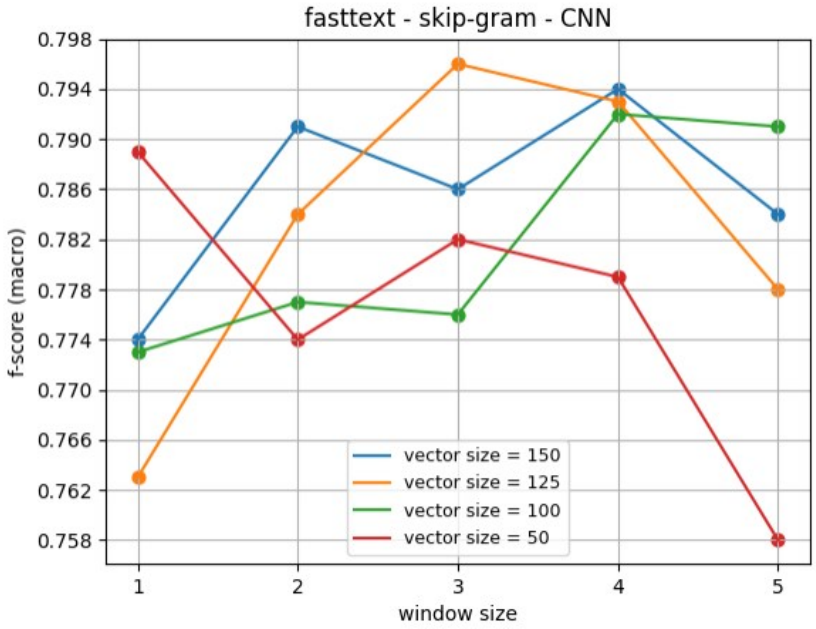
Ostatní hodnoty metriky f-score pro všechny délky vektorů (kromě délky vektoru 50) se pohybují v rozmezí od 0,71 do 0,744. Vektor délky 50 vykazuje nejhorší výsledek - f-score dosahuje maxima 0,706 při velikosti okna 4. Grafy změny f-score pro každou délku vektoru v závislosti na velikosti okna jsou uvedeny na obrázku 42.



Obr. 42. Grafy změny kvality klasifikace MLP při použití metody FastText (skip-gram)

Klasifikace textu pomocí CNN

Výzkum vlivu metody přeskočení architektury FastText na kvalitu klasifikace pomocí konvoluční neuronové sítě ukázal, že maximální hodnota metriky f-score je dosažena pro délku vektoru 125 a velikost okna 3, což je 0,796. Minimální hodnota přesnosti f-score je 0,758 při délce vektoru rovné 50 a velikosti okna rovné 5. Ostatní hodnoty metriky f-score pro všechny délky vektorů se pohybují mezi 0,762 a 0,794. Grafy změny f-score pro každou délku vektoru v závislosti na velikosti okna jsou uvedeny na obrázku 28.



Obr. 43. Grafy změny kvality klasifikace CNN při použití metody FastText (skip-gram)

Shrnutí výsledků

Provedením experimentů s různými kombinacemi délky vektoru a velikosti okna pro metodu FastText architektury skip-gram lze dojít k závěru, že průměrné f-score pro konvoluční neuronovou síť je lepší ve srovnání s ostatními klasifikátory a činí 0,781. Zatímco průměr pro SVM je 0,708 a pro MLP je 0,728. Nejhorších výsledků dosahuje algoritmus Random Forest s hodnotou 0,576. U všech klasifikátorů má nejhorší vliv na kvalitu klasifikace vektor délky 50 bez ohledu na velikost okna.

6.6 Analýza metody Doc2Vec

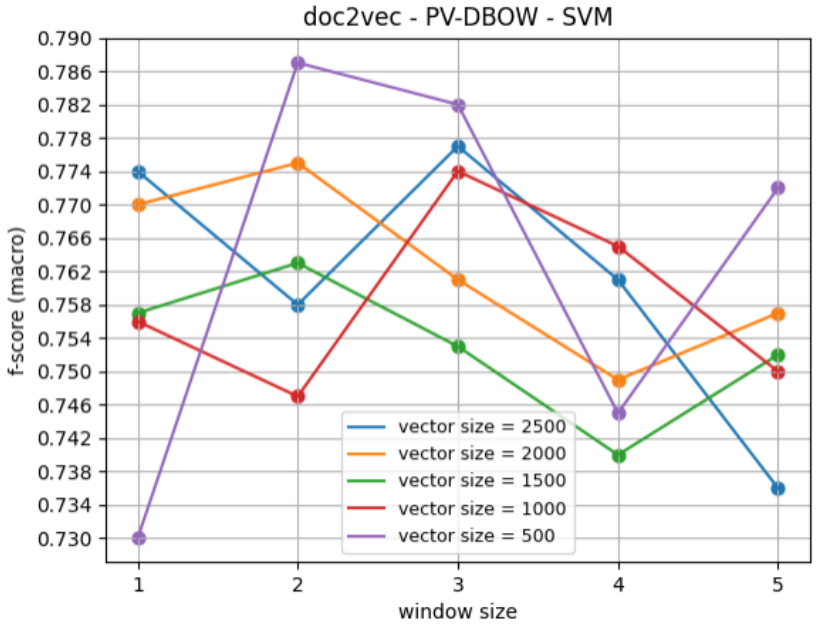
Metoda Doc2Vec reprezentuje dokument jako číselný vektor. Existují dva hlavní přístupy k vytvoření vektorové reprezentace dokumentu pomocí modelu Doc2Vec: architektura PV-DBOW a PV-DM. Pro každou z těchto architektur existují parametry, které ovlivňují kvalitu vektorové reprezentace, a tedy i kvalitu klasifikace. Hlavními parametry jsou délka vektoru, která odpovídá každému tokenu, a velikost okna.

6.6.1 Architektura PV-DBOW

Při analýze této metody byly uvažovány délky vektorů 2500. 2000. 1500. 1000 a 500. což odpovídá dimenzi při použití metod jako BoW a TF-IDF. Pro každou délku vektoru jsme zvažovali vliv změny velikosti okna, která se pohybuje v rozmezí 1 až 5.

Klasifikace textu pomocí SVM

Výzkum vlivu metody Doc2Vec architektury PV-DBOW na kvalitu klasifikace pomocí SVM ukázala, že maximální hodnoty metriky f-score 0,787 je dosaženo při délce vektoru 500 a velikosti okna 2. Minimální hodnota přesnosti f-score je 0,73, když je délka vektoru 500 a velikost okna 1. Ostatní hodnoty metriky f-score pro všechny délky vektorů se pohybují mezi 0,736 a 0,782. Grafy změny f-score pro každou délku vektoru v závislosti na velikosti okna jsou uvedeny na obrázku 44.

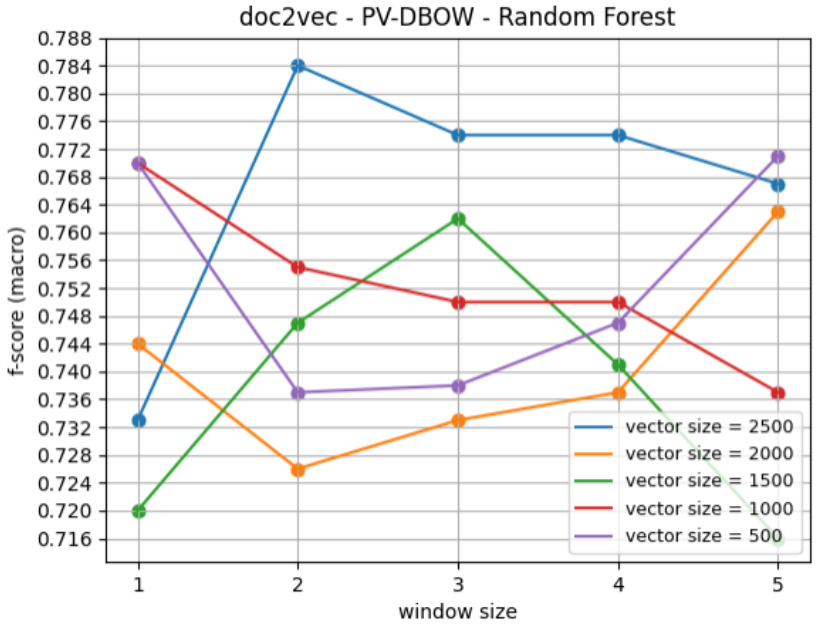


Obr. 44. Grafy změny kvality klasifikace SVM při použití metody Doc2Vec (PV-DBOW)

Klasifikace textu pomocí Random Forest

Analýza vlivu metody Doc2Vec architektury PV-DBOW na kvalitu klasifikace pomocí metody Random Forest ukázala, že maximální hodnoty metriky f-score rovné 0,784 je dosaženo při délce vektoru 2500 a velikosti okna 2. Minimální hodnota přesnosti f-score je 0,716, zatímco délka vektoru je 500 a velikost okna je 1. Ostatní hodnoty metriky f-score

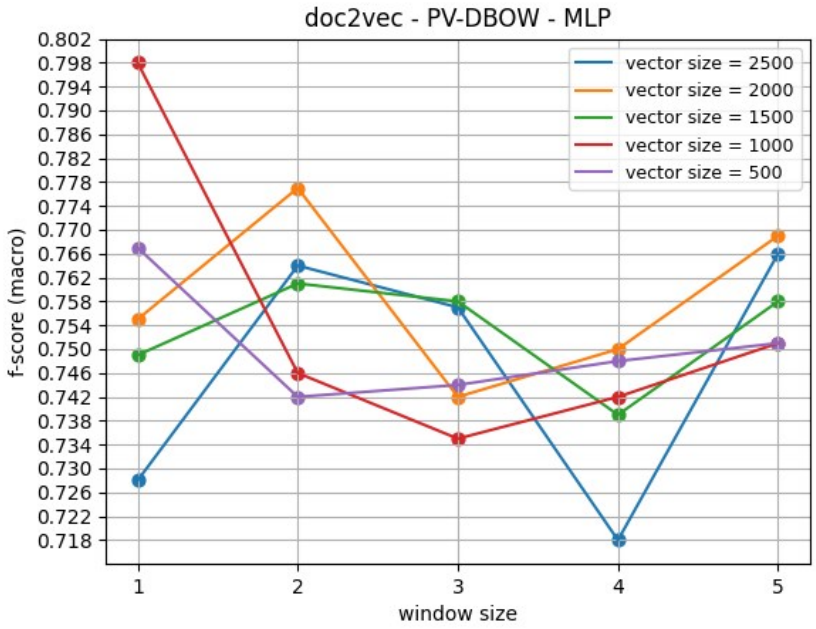
pro všechny délky vektorů se pohybují mezi 0,72 a 0,774. Grafy změny f-score pro každou délku vektoru v závislosti na velikosti okna jsou uvedeny na obrázku 45.



Obr. 45. Grafy změny kvality klasifikace Random Forest při použití metody Doc2Vec (PV-DBOW)

Klasifikace textu pomocí MLP

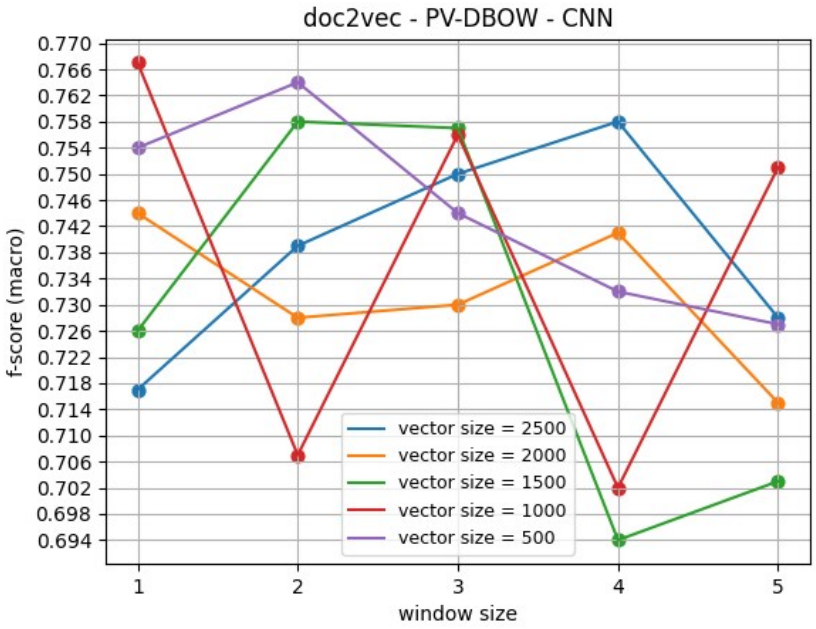
Analýza metody Doc2Vec architektury PV-DBOW na kvalitu klasifikace s neuronovou sítí MLP ukázala, že maximální hodnoty metriky f-score rovné 0,798 je dosaženo při délce vektoru 1000 a velikosti okna 1. Minimální hodnota přesnosti f-score je 0,718 při délce vektoru 2500 a velikosti okna 4. Ostatní hodnoty metriky f-score pro všechny délky vektorů se pohybují mezi 0,727 a 0,777. Grafy změny f-score pro každou délku vektoru v závislosti na velikosti okna jsou uvedeny na obrázku 46.



Obr. 46. Grafy změny kvality klasifikace MLP při použití metody Doc2Vec (PV-DBOW)

Klasifikace textu pomocí CNN

Zkoumání vlivu metody Doc2Vec architektury PV-DBOW na kvalitu klasifikace pomocí konvoluční neuronové sítě ukázalo, že maximální hodnoty metriky f-score rovné 0,766 je dosaženo při délce vektoru 1000 a velikosti okna rovné 1. Minimální hodnota přesnosti f-score je 0,694 při délce vektoru rovné 1500 a velikosti okna 4. Ostatní hodnoty metriky f-score pro všechny délky vektorů se pohybují mezi 0,702 a 0,764. Grafy změny f-score pro každou délku vektoru v závislosti na velikosti okna jsou uvedeny na obrázku 47.



Obr. 47. Grafy změny kvality klasifikace CNN při použití metody Doc2Vec (PV-DBOW)

Shrnutí výsledků

Analýza metody Doc2Vec architektury PV-DBOW s různými kombinacemi délky a velikosti vektoru ukázala, že průměrné f-score pro klasifikátor SVM je ve srovnání s ostatními klasifikátory nejlepší a činí 0,76.

Zatímco průměr pro metodu Random Forest je 0,75 a pro MLP je 0,753. Nejhorších výsledků bylo dosaženo při použití algoritmu konvoluční neuronové sítě, a to 0,736.

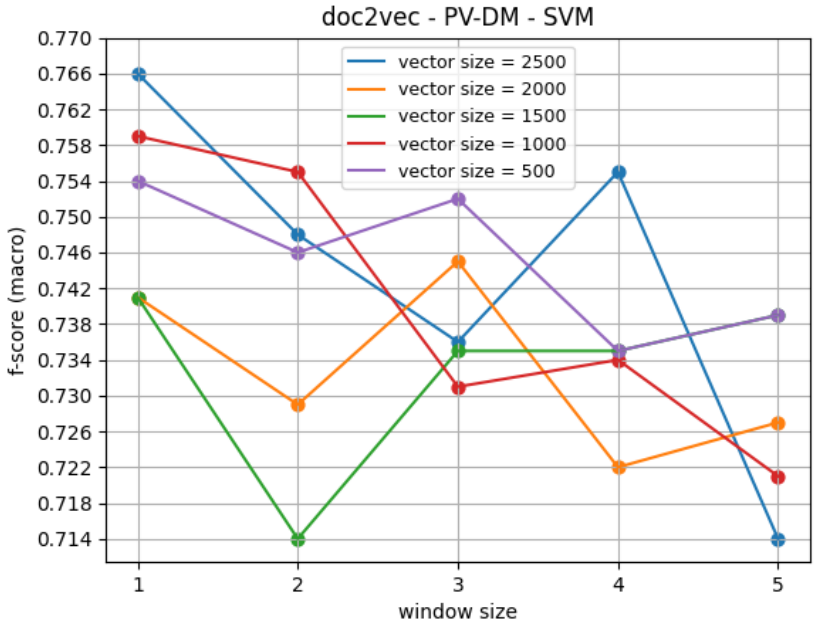
6.6.2 Architektura PV-DM

Při analýze této metody byly uvažovány délky vektorů 2500. 2000. 1500. 1000 a 500. což odpovídá dimenzi při použití metod jako BoW a TF-IDF. Pro každou délku vektoru byl zvažován vliv změny velikosti okna, která se pohybuje od 1 do 5.

Klasifikace textu pomocí SVM

Zkoumání vlivu metody Doc2Vec architektury PV-DM na kvalitu klasifikace pomocí metody SVM ukázalo, že maximální hodnoty metriky f-score rovné 0,766 je dosaženo při délce vektoru 2500 a velikosti okna rovné 1. Minimální hodnota přesnosti f-score je 0,714 při délce vektoru rovné 1500 2500 a velikosti okna rovné 2 a 5. Ostatní hodnoty metriky f-

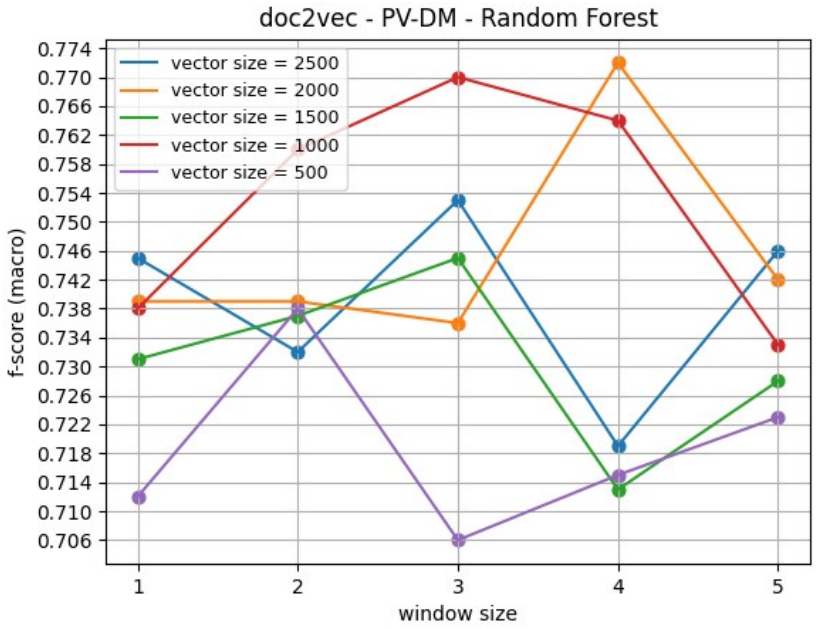
score pro všechny délky vektorů se pohybují mezi 0,722 a 0,759. Grafy změny f-score pro každou délku vektoru v závislosti na velikosti okna jsou uvedeny na obrázku 48.



Obr. 48. Grafy změny kvality klasifikace SVM při použití metody Doc2Vec (PV-DM)

Klasifikace textu pomocí Random Forest

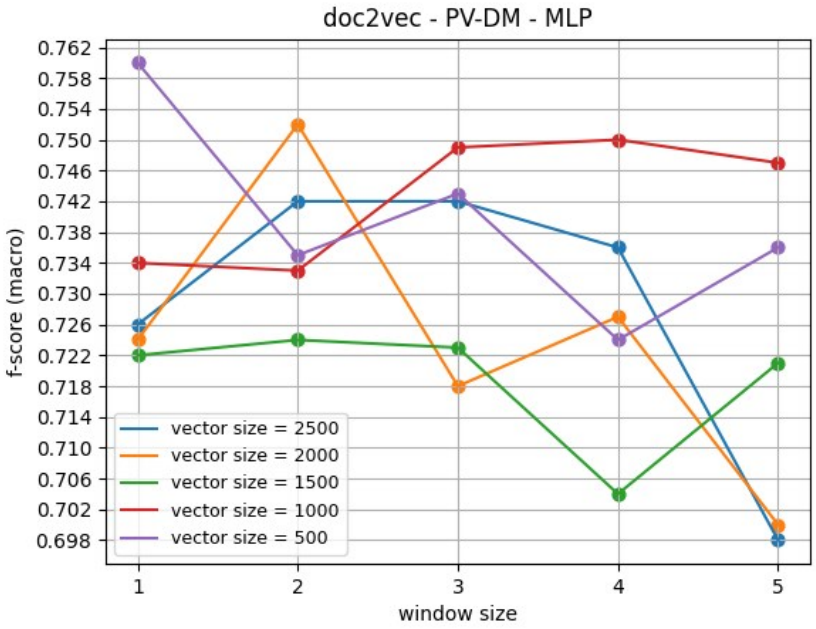
Analýza vlivu metody Doc2Vec architektury PV-DM na kvalitu klasifikace pomocí metody Random Forest ukázala, že maximální hodnoty metriky f-score rovné 0,772 je dosaženo při délce vektoru 2000 a velikosti okna 4. Minimální hodnota přesnosti f-score je 0,706, zatímco délka vektoru je 500 a velikost okna je 3. Ostatní hodnoty metriky f-score pro všechny délky vektorů se pohybují mezi 0,712 a 0,77. Grafy změny f-score pro jednotlivé délky vektorů v závislosti na velikosti okna jsou uvedeny na obrázku 49.



Obr. 49. Grafy změny kvality klasifikace Random Forest při použití metody Doc2Vec (PV-DM)

Klasifikace textu pomocí MLP

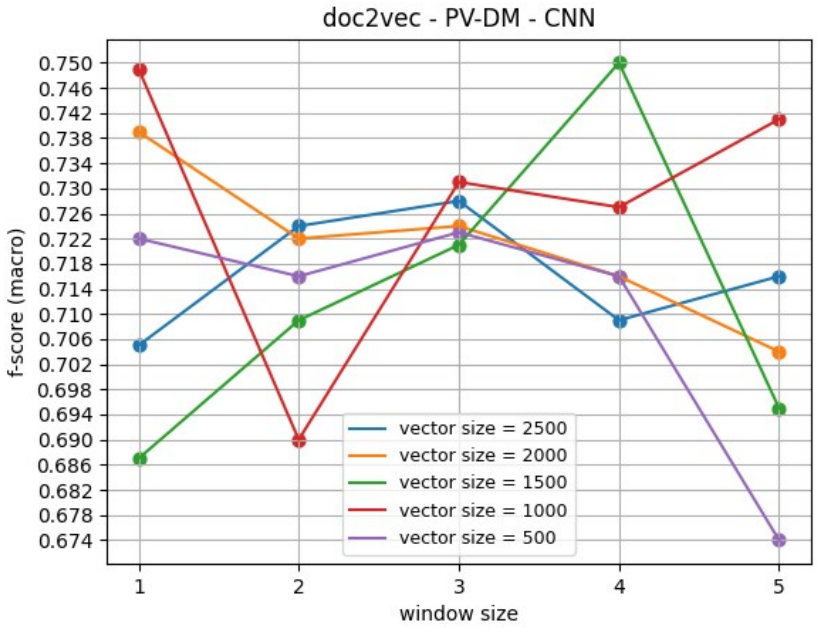
Výzkum vlivu metody Doc2Vec architektury PV-DM na kvalitu klasifikace pomocí neuronové sítě MLP ukázal, že maximální hodnoty metriky f-score rovné 0,76 je dosaženo při délce vektoru 500 a velikosti okna rovné 1. Minimální hodnoty metriky f-score jsou 0,698 a 0,70 při délce vektoru 2500 a 2000 a velikosti okna 5. Ostatní hodnoty metriky f-score pro všechny délky vektorů jsou mezi 0,708 a 0,752. Grafy změny f-score pro každou délku vektoru v závislosti na velikosti okna jsou uvedeny na obrázku 50.



Obr. 50. Grafy změny kvality klasifikace MLP při použití metody Doc2Vec (PV-DM)

Klasifikace textu pomocí CNN

Analýza vlivu metody Doc2Vec architektury PV-DM na kvalitu klasifikace pomocí CNN ukázala, že maximální hodnoty metriky f-score rovné 0,75 je dosaženo pro délku vektoru 1500 a 1000 a velikost okna rovnou 4 a 1. Minimální hodnota přesnosti f-score je 0,674, přičemž délka vektoru je 500 a velikost okna je 5. Ostatní hodnoty metriky f-score pro všechny délky vektorů se pohybují mezi 0,686 a 0,738. Současně při velikosti okna 3 vykazují všechny délky vektorů blízké hodnoty metriky f-score, které se pohybují mezi 0,72 a 0,731. Grafy změny f-score pro jednotlivé délky vektorů v závislosti na velikosti okna jsou uvedeny na obrázku 51.



Obr. 51. Grafy změny kvality klasifikace CNN při použití metody Doc2Vec (PV-DM)

Shrnutí výsledků

Provedením experimentů s různými kombinacemi délky vektoru a velikosti okna pro metodu Doc2Vec architektury PV-DM lze dojít k závěru, že průměrné f-score pro metody SVM a Random Forest je nejlepší ve srovnání s ostatními klasifikátory a činí 0,738 a 0,737. Zatímco průměr pro MLP je 0,73. Nejhorších výsledků dosahuje algoritmus konvoluční neuronové sítě s hodnotou 0,71.

6.7 Diskuze výsledků

Z analýzy vlivu reprezentace textu na kvalitu klasifikace pomocí metod umělé inteligence lze vyvodit následující závěry:

1. Lepší kvality klasifikace bylo dosaženo pomocí klasifikátoru CNN a Word Embedding metody FastText.
2. Při použití architektury skip-gram pro modely FastText a Word2Vec bylo dosaženo lepší kvality klasifikace než při použití architektury CBOW.
3. Při použití architektury PV-DBOW pro model Doc2Vec bylo dosaženo lepší kvality klasifikace než při použití architektury PW-DM.
4. Metoda Random Forest dosahuje dobrých výsledků při použití klasických metod reprezentace textu, jako jsou BoW a TF-IDF.

5. Metoda Random Forest má při použití Word Embedding metod slabé výsledky.
6. Velikost okna rovná 1 má horší výsledky na kvalitu klasifikace při použití Embedding metod.
7. Délka vektoru rovná 50 má horší výsledky při použití metod Embedding.
8. V průměru nejlepší kvality klasifikace s metodou Embedding je dosaženo při velikosti okna rovné 3 nebo 4
9. V průměru je nejlepší kvality klasifikace s metodou Embedding dosaženo při délce vektoru rovné 125 nebo 150.
10. Stálé zvyšování počtu rysů nevede ke zvýšení kvality klasifikace, takže počet rysů rovný 1500 a 2000 vykazuje lepší kvalitu klasifikace při použití metod BoW a TF-IDF.

Tabulka 1 ukazuje průměrné f-score pro každý klasifikátor v závislosti na použité metodě reprezentace textu. Nejlepší kvalita je zvýrazněna tučně.

Tabulka 1. Průměrná kvalita klasifikace v závislosti na způsobu reprezentace textu

	Klasifikátory			
Metody reprezentace	**CNN**	**MLP**	**SVM**	**Random Forest**
BoW	0,74	0,74	0,74	0,74
TF-IDF	0,74	0,75	0,76	**0,77**
Word2Vec (CBOW)	**0,75**	0,71	**0,75**	0,54
Word2Vec (skip-gram)	**0,77**	0,72	0,75	0,54
GloVe	**0,77**	0,76	0,76	0,49
FastText (CBOW)	**0,75**	0,72	**0,75**	0,53
FastText (CBOW)	**0,78**	0,73	0,71	0,57
Doc2Vec (PV-DBOW)	0,74	0,75	**0,76**	0,75
Doc2Vec (PV-M)	0,71	0,73	**0,74**	**0,74**

ZÁVĚR

Tato diplomová práce analyzuje vliv reprezentace textu na kvalitu klasifikátorů využívajících techniky umělé inteligence.

V první části práce byly považovány jak klasické metody reprezentace textu, jako jsou BoW a TF-IDF, tak novější Embedding metody, jako jsou Word2Vec, GloVe, FastText, Doc2Vec.

Dále byly zváženy metody umělé inteligence, které byly použity ke klasifikaci textů. Jednalo se jak o umělé neuronové sítě – MLP a CNN, tak o algoritmy strojového učení – SVM a Random Forest.

Zvláštní pozornost byla věnována přístupům, které umožňují zlepšit kvalitu modelu. Jedná se o vyvažování dat, redukci dimenzionality pomocí metody PCA, křížovou validaci.

Druhá část práce je věnována tvorbě datového korpusu, která zahrnuje dolování dat prostřednictvím API, čištění dat a předzpracování získaných textových dat.

Byl analyzován a porovnán vliv metod reprezentace textu na klasifikátory. Výsledkem analýzy bylo zjištění, že FastText je nejlepší metodou pro reprezentaci textových dat pomocí konvoluční neuronové sítě. Při použití metod strojového učení, jako jsou Random Forest a SVM, je nejlepší kvality klasifikace dosaženo při použití metody TF-IDF. Byl také zjištěn negativní vliv metod reprezentace textu na kvalitu klasifikace. Metoda Random Forest vykázala výrazný pokles f-score při použití Word Embedding metod, jako jsou Word2Vec, GloVe, FastText.

SEZNAM POUŽITÉ LITERATURY

[1] Why NLP is important and it'll be the future — our future. [online]. [cit. 2023-02-03]. Dostupné z: https://towardsdatascience.com/why-nlp-is-important-and-itll-be-the-future-our-future-59d7b1600dda

[2] What Is Natural Language Processing (NLP)? [online]. [cit. 2023-02-03]. Dostupné z: https://aws.amazon.com/what-is/nlp/

[3] Stemming vs Lemmatization in NLP: Must-Know Differences. [online]. [cit. 2023-02-03]. Dostupné z: https://www.analyticsvidhya.com/blog/2022/06/stemming-vs-lemmatization-in-nlp-must-know-differences/

[4] 8 best Python Natural Language Processing (NLP) libraries. 2018. [online]. [cit. 2023-02-03]. Dostupné z: https://sunscrapers.com/blog/8-best-python-natural-language-processing-nlp/#4-gensim

[5] Top 20 Python Libraries for Data Science for 2023. [online]. [cit. 2023-02-03]. Dostupné z: https://www.simplilearn.com/top-python-libraries-for-data-science-article

[6] PySpark Overview. [online]. [cit. 2023-02-05]. Dostupné z: https://spark.apache.org/docs/latest/api/python/

[7] Feature Transformation of Text Data — NLP. 2020. [online]. [cit. 2023-02-03]. Dostupné z: https://medium.com/swlh/feature-transform-of-text-data-nlp-c6ccedbeb3cc

[8] Word2Vec Explained. 2021. [online]. [cit. 2023-02-07]. Dostupné z: https://towardsdatascience.com/word2vec-explained-49c52b4ccb71

[9] The backpropagation algorithm for Word2Vec. 2018. [online]. [cit. 2023-02-04]. Dostupné z: http://www.claudiobellei.com/2018/01/06/backprop-word2vec/

[10] Skip-Gram: NLP context words prediction algorithm. 2019. [online]. Dostupné z: https://towardsdatascience.com/skip-gram-nlp-context-words-prediction-algorithm-5bbf34f84e0c. [cit. 2023-02-10].

[11] Как работает Word2Vec: нейросети для NLP. [online]. Dostupné z: https://python-school.ru/blog/what-is-word2vec. [cit. 2023-02-12].

[12] Intuitive Guide to Understanding GloVe Embeddings. 2019. [online]. [cit. 2023-02-15]. Dostupné z: https://towardsdatascience.com/light-on-math-ml-intuitive-guide-to-understanding-glove-embeddings-b13b4f19c010

[13] Jeffrey Pennington, Richard Socher, Christopher D. Manning. GloVe: Global Vectors for Word Representation. [online]. [cit. 2023-02-15]. Dostupné z: https://nlp.stanford.edu/pubs/glove.pdf

[14] An Intuitive Introduction to Document Vector (Doc2Vec). 2019. [online]. [cit. 2023-02-15]. Dostupné z: https://pub.towardsai.net/an-intuitive-introduction-of-document-vector-doc2vec-42c6205ca5a2

[15] A gentle introduction to Doc2Vec. 2017. [online]. [cit. 2023-02-15]. Dostupné z: https://medium.com/wisio/a-gentle-introduction-to-doc2vec-db3e8c0cce5e

[16] A Visual Guide to FastText Word Embeddings. [online]. [cit. 2023-02-15]. Dostupné z: https://amitness.com/2020/06/fasttext-embeddings

[17] Piotr Bojanowski, Edouard Grave, Armand Joulin, Tomas Mikolov. Enriching Word Vectors with Subword Information. 2017. [online]. [cit. 2023-02-17]. Dostupné z: https://arxiv.org/abs/1607.04606/

[18] Multiclass Classification with Support Vector Machines (SVM), Dual Problem and Kernel Functions. 2020. [online]. [cit. 2023-02-17]. Dostupné z: https://towardsdatascience.com/multiclass-classification-with-support-vector-machines-svm-kernel-trick-kernel-functions-f9d5377d6f02

[19] Understand Random Forest Algorithms With Examples. 2023. [online]. [cit. 2023-02-20]. Dostupné z: https://www.analyticsvidhya.com/blog/2021/06/understanding-random-forest/

[20] An Introduction to Decision Trees. [online]. [cit. 2023-02-20]. Dostupné z: https://blog.paperspace.com/decision-trees/

[21] Convolutional Neural Networks for Text Classification. 2018. [online]. [cit. 2023-02-20]. Dostupné z: https://www.davidsbatista.net/blog/2018/03/31/SentenceClassificationConvNets/

[22] Multi-Layer Perceptron Explained: A Beginner's Guide. 2023. [online]. [cit. 2023-02-20]. Dostupné z: https://www.pycodemates.com/2023/01/multi-layer-perceptron-a-complete-overview.html?m=1&utm_content=cmp-true

[23] Forward Propagation, Backward Propagation, and Computational Graphs. [online]. [cit. 2023-03-10]. Dostupné z: https://d2l.ai/chapter_multilayer-perceptrons/backprop.html

[24] Re-sampling Imbalanced Training Corpus for Sentiment Analysis. 2019. [online]. [cit. 2023-03-10]. Dostupné z: https://medium.com/analytics-vidhya/re-sampling-imbalanced-training-corpus-for-sentiment-analysis-c9dc97f9eae1

[25] SMOTE and Edited Nearest Neighbors Undersampling for Imbalanced Classification. 2021. [online]. [cit. 2023-03-10]. Dostupné z: https://fauzanvaldera.medium.com/smote-and-edited-nearest-neighbors-undersampling-for-imbalanced-datasets-a3f08d45f85c#:~:text=Edited%20Nearest%20Neighbor%20(ENN)%20is,together%20to%20handle%20imbalanced%20datasets.

[26] Tidying up with PCA: An Introduction to Principal Components Analysis. 2019. [online]. [cit. 2023-03-10]. Dostupné z: https://towardsdatascience.com/tidying-up-with-pca-an-introduction-to-principal-components-analysis-f876599af383/

[27] Micro, Macro & Weighted Averages of F1 Score, Clearly Explained. 2022. [online]. [cit. 2023-03-10]. Dostupné z: https://towardsdatascience.com/micro-macro-weighted-averages-of-f1-score-clearly-explained-b603420b292f

[28] GloVe and fastText Clearly Explained: Extracting Features from Text Data. 2022. [online]. [cit. 2023-02-15]. Dostupné z: https://levelup.gitconnected.com/glove-and-fasttext-clearly-explained-extracting-features-from-text-data-1d227ab017b2#3b6e

[29] Understanding FastText:An Embedding To Look Forward To. 2019. [online]. [cit. 2023-03-10]. Dostupné z: https://adityaroc.medium.com/understanding-fasttext-an-embedding-to-look-forward-to-3ee9aa08787

[30] Multiclass Classification with Support Vector Machines (SVM), Dual Problem and Kernel Functions. 2020. [online]. [cit. 2023-03-20]. Dostupné z: https://www.davidsbatista.net/blog/2018/03/31/SentenceClassificationConvNets/

[31] Tomas Mikolov, Kai Chen, Greg Corrad, Jeffrey Dean. Efficient Estimation of Word Representations in Vector Space. 2019. [online]. [cit. 2023-02-18]. Dostupné z: https://arxiv.org/abs/1301.3781

SEZNAM POUŽITÝCH SYMBOLŮ A ZKRATEK

NLP	Natural Language Processing
BoW	Bag of Words
TF-IDF	Term Frequency - Inverse Document Frequency
GloVe	Global Vectors
CNN	Convolutional neural network
MLP	Multilayer Perceptron
SVM	Support Vector Machine
SGD	Stochastic Gradient Descent
PCA	Principal Component Analysis
CBOW	Continuous Bag of Words
PV-DM	Paragraph Vector with Distributed Memory
PV-DBOW	Paragraph Vector with Distributed Bag of Words

SEZNAM OBRÁZKŮ

SEZNAM TABULEK

SEZNAM PŘÍLOH

PŘÍLOHA P I: OBSAH PŘILOŽENÉHO CD

1. CD obsahující, vytvořený datový korpus ve formátu csv, zdrojový kód, výsledky ve formátu json a obrázky s grafy.