

Zpracování textu pomocí hlubokých neuronových sítí

Natural language processing

Bc. Vojtěch Prokop

Diplomová práce

Vedoucí práce: prof. Ing. Jan Platoš, Ph.D.

Ostrava, 2022

Zadání diplomové práce

Student: **Bc. Vojtěch Prokop**

Studijní program: N2647 Informační a komunikační technologie

Studijní obor: 2612T025 Informatika a výpočetní technika

Téma: Zpracování textu pomocí hlubokých neuronových sítí
Text Processing using Neural Networks

Jazyk vypracování: čeština

Zásady pro vypracování:

Cílem práce bude prostudovat použití neuronových sítí a hlubokých neuronových sítí pro zpracování textu, například analýza sentimentu, klasifikace, modelování topiku, extrakce znalostí, apod.

Práce bude obsahovat:

1. Přehled oblastí, kde lze neuronové sítě využít.
2. Detailní popis vybraných metod a aplikačních oblastí.
3. Implementaci jedné nebo více metod pro dané aplikační oblasti.
4. Ověření fungování implementovaných metod a porovnání výsledků s jinými metodami.

Seznam doporučené odborné literatury:

- [1] Torres-Moreno, Juan-Manuel, ed. Automatic text summarization. John Wiley & Sons, 2014.
- [2] Mani, Inderjeet, and Mark T. Maybury. Advances in automatic text summarization. MIT press, 1999.
- [3] Chris Manning and Hinrich Schütze, Foundations of Statistical Natural Language Processing, MIT Press. Cambridge, MA: May 1999
- [4] Dan Jurafsky: Speech and Language Processing, Pearson Education, 2014
- [5] Bing Liu: Sentiment Analysis: Mining Opinions, Sentiments, and Emotions, Cambridge University Press; 1 edition (June 4, 2015)
- [6] Charu C. Aggarwal: Machine Learning for Text, Springer, 2018.

Formální náležitosti a rozsah diplomové práce stanoví pokyny pro vypracování zveřejněné na webových stránkách fakulty.

Vedoucí diplomové práce: **prof. Ing. Jan Platoš, Ph.D.**

Datum zadání: 01.09.2021

Datum odevzdání: 30.04.2022

doc. Ing. Petr Gajdoš, Ph.D.
vedoucí katedry

prof. Ing. Jan Platoš, Ph.D.
děkan fakulty

Abstrakt

Diplomová práce se zabývá popisem jednotlivých bloků procesu zpracování přirozeného jazyka od přípravy textových dat, předzpracování, až po návrh modelů, které řeší klasifikační problém nad jazykovým korpusem.

V teoretické části jsou detailněji popsány modely od klasických přístupů strojového učení, až po hojně využívanou architekturu Transformer. Právě modely, které jsou založeny na této architektuře, jejich struktura a výkonost je hlavním doménou této diplomové práce.

V praktické části jsou provedeny experimenty nad různými přístupy a následně srovnány jejich výsledky. Jsou použity 3 přístupy, vektorizace textu a následné využití klasických modelů, využití architektur neuronových sítí až po architekturu Transformer a v poslední řadě využití derivátu BERT modelu ve spojení s hlubokou dopřednou sítí. Nad všemi těmito modely byla zkoumána kvalita přesnosti u problému autorství, kde k neznámému textu model odhadoval s určitou přesností možného autora.

Klíčová slova

zpracování přirozeného jazyka; neuronové sítě; určení autorství; Transformer architektura; BERT model; ELECTRA; DistilBERT

Abstract

The thesis deals with the description of the different blocks of the natural language processing process from the preparation of text data, pre-processing, to the design of models that solve the classification problem over the language corpus.

In the theoretical part, models ranging from classical machine learning approaches to the widely used Transformer architecture are described in detail. It is the models that are based on this architecture, their structure and performance that is the main domain of this thesis.

In the practical part, experiments are performed over the different approaches and then their results are compared. Three approaches are used, text vectorization and the subsequent use of classical models, the use of neural network architectures up to the Transformer architecture and lastly the use of a derivative of the BERT model in conjunction with a deep forward network. Over all of these models, the quality of accuracy was investigated for the authorship problem, where, given an unknown text, the model estimated a possible author with some confidence.

Keywords

natural language processing; artificial neural networks; authorship identification; Transformer architecture; BERT model; ELECTRA; DistilBERT

Poděkování

Rád bych na tomto místě vyjádřil své poděkování vedoucímu práce prof. Ing. Jan Platoš, Ph.D. za ochotu při poskytování rad a vstřícný přístup během konzultací. Dále bych také chtěl poděkovat své rodině a blízkým, kteří měli trpělivost, byli psychickou podporou a umožnili dokončení této práce.

Obsah

Seznam použitých symbolů a zkratek	10
Seznam obrázků	12
Seznam tabulek	14
1 Úvod	15
2 Předzpracování textové datové sady	17
2.1 Stop слова	18
2.2 Stematizace	18
2.3 Lematizace	19
2.4 Tokenizace	19
2.5 Ostatní metody	19
3 Reprezentace textových dokumentů	21
3.1 One hot kódování	21
3.2 Bag of words	22
3.3 TF-IDF	22
3.4 Word2Vec	23
3.5 GloVe	25
3.6 Vektorová reprezentace vycházející z architektury Transformeru	26
4 Modely pro klasifikaci	32
4.1 Klasické modely	32
4.2 Modely založené na neuronových sítích	36
5 Transformer	40
5.1 Enkodér a dekodér bloky	41
5.2 Enkodér	41
5.3 Self-Attention	42

5.4	Výpočet v mechanismu Self-Attention	43
5.5	Optimální výpočet Self-Attention	44
5.6	Multihead-Attention	45
5.7	Dopředná síť	45
5.8	Dekodér	46
5.9	Lineární a softmax vrstva	46
6	Definice problému	47
6.1	Obecný popis řešení	47
6.2	Popis vlastního řešení	48
6.3	Datová sada	48
6.4	Explorační analýza nad vytvořenými soubory	51
6.5	Předzpracování textové sady	53
6.6	Kvalita modelů	53
6.7	Popis spouštění experimentů	54
7	Vývojové prostředí	56
7.1	Natural Language Toolkit	56
7.2	Keras a TensorFlow	56
7.3	Gensim	57
7.4	HuggingFace	57
7.5	gutenbergr	57
7.6	Výpočetní prostředí	57
8	Experiment 1 - vektorizace textu a klasické modely	58
8.1	Popis řešení	59
8.2	Výsledky	59
9	Experiment 2 - neuronové sítě	62
9.1	Popis řešení	63
9.2	Výsledky	64
10	Experiment 3 - BERT	66
10.1	Popis řešení	67
10.2	Výsledky	68
11	Finální zhodnocení	75
12	Závěr	82

Literatura	84
Přílohy	86
A Struktura projektu	87
B BERTPoolingLayer	90

Seznam použitých zkrátek a symbolů

- BERT – Bidirectional Encoder Representations of Transformers
- BoW – Bag of Words
- CBOW – Continuous Bag of Words
- CNN – Convolutional neural network
- CPU – Centrální procesorová jednotka (central processing unit)
- CUDA – Compute Unified Device Architecture
- DF – Document Frequency
- ELECTRA – Efficiently Learning an Encoder that Classifies Token Replacements Accurately
- GB – gigabajt (gigabyte)
- GPU – grafický procesor (graphics processing unit)
- GRU – Gated recurrent unit
- GloVe – Global Vectors for Word Representation
- HTML – Hypertext Markup Language
- IDF – Inverse document frequency
- LSTM – Long short-term memory
- MLM – Masked Language Model
- NER – rozpoznávání pojmenovaných entit (Named-entity recognition)
- NLP – zpracování přirozeného jazyka (natural language processing)
- NLTK – Natural Language Toolkit
- NSP – Next Sentence Prediction
- OOV – Out-of-vocabulary
- QA – Question answering
- RNN – Recurrent neural network
- RTD – Replaced token detection
- TF – Term Frequency
- TF-IDF – Term Frequency-Inverse document frequency
- URL – jednotný lokátor zdroje (uniform resource locator)

- VŠB-TUO
- Vysoká škola báňská - Technická univerzita Ostrava
- csv
- comma-separated values
- epub
- electronic publication
- json
- JavaScript Object Notation
- pdf
- Portable Document Format
- seq2seq
- Sequence-to-sequence
- txt
- textová zpráva (text message)

Seznam obrázků

3.1	CBOW architektura [8]	24
3.2	Skip gram architektura [8]	25
3.3	Způsob předtrénování BERT modelu a jeho využití (<i>[CLS] token zkráceně C</i>) [10] . .	27
3.4	Vstupní vektor do BERT modelu [10]	28
3.5	Aplikace BERT modelu u rozličných problémů zpracování přirozeného jazyka [10] . .	30
3.6	ELECTRA [14]	31
4.1	Porovnání lineární a logistické regrese [15]	34
4.2	Vizualizace BoW vektorizace s modelem rozhodovacího stromu	34
4.3	Konvoluční síť u zpracování mluveného slova [20]	38
4.4	Vizualiacie mechanismu maximálního poolingu [21]	39
5.1	Transformer architektura [22]	40
5.2	Dekodér a enkodér blok transformer architektury [23]	41
5.3	Detailnější vizualizace enkodér bloku [23]	41
5.4	Self-Attention mechanismus [23]	42
5.5	Detailní vizualizace kalkulace vektoru v Self-Attention mechanismu [23]	43
5.6	Maticový výpočet self-attention mechanismu [23]	44
5.7	Multi-head attention [23]	45
5.8	Spolupráce enkodéru a dekodéru [23]	46
6.1	Gutenberg v jazyce R	48
6.2	Automatizovaná tvorba datové sady	50
6.3	.json formát jednoho staženého díla	51
6.4	.csv soubor s vytvořenou datovou sadou	51
6.5	Statistický výstup	51
6.6	Vizualizace počtu záznamů	52
6.7	Detailnější pohled na vybrané kvantily vzhledem k velikosti sekvence	53
6.8	Automatizované spouštění experimentů	54

8.1	Architektura klasických modelů	58
8.2	Grafová vizualizace klasických modelů s různými typy vektorizace	60
8.3	Různý počet autorů ve spojení s klasickými modely	61
9.1	Diagram popisující druhý experimentální blok	62
9.2	Modelování velikosti mnohadimenzionální prostoru pro reprezentaci slov	65
10.1	Architektura BERT modelu	66
10.2	Detailní průběh záznamů při klasifikaci	67
10.3	Kvalita obecného jazykového DistilBERT modelu	68
10.4	Experimentální ověření učící konstanty	69
10.5	Strategie získání číselných vektorů	71
10.6	Výsledek přesnosti nad 5 autory, 3 větami a různými typy strategií	72
10.7	Průběh derivátů	72
11.1	Výsledky vybraných modelů u více autorů	76
11.2	Výsledky vybraných modelů u experimentu s variabilním větným oknem	77
11.3	Srovnání chybových matic nejlepších modelů	78
11.4	Srovnání všech experimentů nad 5 a 15 autory	79

Seznam tabulek

8.1 Časové nároky na vektorizaci v kombinaci s klasickými modely	61
9.1 Výsledky experimentu s větným oknem	64
10.1 Experimentální výsledky nad testovací množinou vzhledem k učící konstantě	70
10.2 Počet trénovacích, testovacích a validačních záznamů v experimentu s větným oknem	73
10.3 Výsledky experimentu s větným oknem	73
10.4 Informace o množinách k různému počtu autorů	74
10.5 Výsledky experimentu s více autory	74
11.1 Výběr nejlepších modelů	75
11.2 Výsledky nejlepších modelů u experimentu s větším počtem autorů	75
11.3 Výsledky nejlepších modelů nad variabilním větným oknem	77

Kapitola 1

Úvod

Zpracování přirozeného jazyka (*anglicky Natural language processing, neboli zkráceně NLP*) je soubor několika vědních oborů, společně se zabývajících tvorbou počítačových modelů, které jsou způsobilé porozumět textovým datům a mluvenému slovu. Jinými slovy jde o schopnost porozumět mluvenému projevu a také sdělení v přirozeném jazyce vyprodukovat.

Mezi typické úlohy zpracování přirozeného jazyka patří například generování jazyka, porozumění dotazů kladených na hlasové asistenty (Google Home, Apple HomeKit), různé varianty klasifikačních úloh (analýza sentimentu, autorství, detekce spamu), automatizovaná komunikace s lidmi (chatboty), strojový překlad, optimalizace SERP (*search engine results page*) stránky díky zkvalitnění porozumění kladených dotazů, společně s chápáním obsahu stránek a mnoho dalšího. Předmětem experimentů v této diplomové práci je právě prediktivní schopnost systému vzhledem k autorům. Přesněji jde o odhad autora daného textu, který model doposud ještě neviděl.

K řešení velkého množství problémů v doméně zpracování přirozeného jazyka se v posledních letech často používá matematický model neuronových sítí, který obohacuje původní klasické modely o nové dovednosti v rámci porozumění. V posledních pár letech bývá považován za nejmodernější přístup využití neuronových sítí s architekturou Transformer. Tento přístup má možnost být aplikován na široké spektrum úloh zpracování přirozeného jazyka, kde vždy byly zjištěny state-of-art výsledky překonávající doposud nejlepší.

Celá práce je rozdělena do dvou dílčích bloků, teoretické a praktické části. Teoretická část systematicky popisuje proces zpracování přirozeného jazyka v jednotlivých částech od předzpracování, přes získání číselných vektorů až po představení použitelných modelů.

V kapitole o předzpracování jsou čtenářovi demonstrovány základní principy tohoto procesu a dopad na výstup. Další kapitola věnující se transformaci textových dat do číselných vektorů obsáhle popisuje současné varianty přístupu k tomuto problému. Jedná se o představení starších frekvenčních přístupu, přes počátky využití neuronových sítí. Díky brilantního nápadu českého vědce Tomáše Mikulova (Word2Vec), kde ve finální části je rozebrán moderní přístup kontextového vnoření slov (*word embeddings*) v podobě derivátu BERT modelu založeném na zmíněné Transformer architektuře. Po

představení možností vektorizace dokumentů jsou rozebrány detailněji ony klasifikační modely. Klasické (logistická regrese, náhodný les, naive bayes, nejbližší sousedé) a architektury neuronových sítí (konvoluční, rekurentní, dopředné sítě). Uzavření teoretické části je věnováno popisu Transformer architektury.

Po představení teoretického bloku dochází k plynulému seznámení problému autorství od obecného pojetí až po více specifický popis, který byl aplikován v praktické části diplomové práce. Zároveň je věnována část popisu tvorbě potřebné datové sady, provedené explorační analýze nad vytvořenou datovou sadou a následným krátkým popisem aplikací vytěžených informací z explorační analýzy.

Další kapitola, před samotnými experimenty, se věnuje popisem vývojového prostředí, díky kterého bylo možné všechny experimenty provést.

Poslední kapitoly se zaobírají prezentováním navržených experimentů, které byly rozděleny do 3 přístupů. Vektorizaci textu a následné využití klasických modelů, využití architektur neuronových sítí až po architekturu Transformer. V poslední řadě využití derivátu BERT modelu ve spojení s hlubokou dopřednou sítí.

Celou práci uzavírá kapitola shrnující výsledky ze všech 3 přístupů a vyvozený závěr.

Kapitola 2

Předzpracování textové datové sady

Při analytické činnosti [1], na textových datech, je většina dat dostupných skrz webové stránky, sociální sítě, emaily nebo různé chatovací aplikace a mnoho dalších. Většina těchto textových dat jsou v nestrukturovaném formátu, obsahující například prvky značkovacího jazyka, URL řetězce, meta atributy. Proto je žádoucí provést transformaci z nestrukturovaného formátu do strukturovaného, a tím odstranit irelevantní informace.

V současné době existuje několik technik, které se využívají u předzpracování textové sady. Ačkoliv tyto techniky jsou všem známé, neznamená to, že je stejná posloupnost kroků provedena při každém předzpracování. Vždy je nutné mít na paměti, že každé zpracovávané datové sadě je potřeba rozumět a provést takové kroky, které žádným způsobem neznehodnotí možný výsledek.

Pokud zanedbáme speciální přístup k určitým datovým sadám, pak jsou textová data většinou normalizována pomocí odstranění nebo nahrazení nepodstatných slov (HTML tagy, URL řetězce, interpunkční znaménka, stopslova), konverze řetězců do stejného tvaru tak, aby každé slovo mělo pouze malá písmena (slovo jazyk, Jazyk, JAZYK, bude považováno za totožné) a provedení stema-tizačních, lemmatizačních technik, které budou detailněji popsány níže společně s tokenizací.

2.1 Stopslova

Ne všechna slova z dokumentu poskytují informaci. Příkladem takových slov jsou stopslova (a, v, nebo, mě), z pohledu slovního druhu se tedy jedná o předložky, spojky, zájmena. Vlastnost, kterou tato slova obvykle mají je vysoká četnost, ačkoliv hodnota informace, kterou poskytují je vzhledem k frekvenci zanedbatelná, a tím je v analýze produkováno velké množství šumu.

V rámci předzpracování předpokládáme existenci stopslov, a proto provádíme jednu z následujících praktik:

- Odstranění velice frekventovaných slov.
- Odstranění slov z dokumentu v zavilosti na poskytnutém universu reprezentující všechna stopslova.
- Zachování slov, z důvodu výskytu alespoň minimální informativní hodnoty. Dochází k penalizaci váhy popisující četnost slova (vynásobením specifikovaným koeficientem).

2.2 Stematizace

V praxi [2] při analýze jazykového korpusu často narazíme na slova, která se vyskytují v různém tvaru. Výskyt v různých tvarech dochází z důvodu odvozování (*derivaci*) slova, což je základní princip obohacování slovní zásoby jazyka. Technika stematizace je založena na myšlence redukce slova do tvaru reprezentující základní tvar (tzn. *stem*) podobný kořenu daného slova. Tento proces odstraňuje koncovky, přípony, předpony, přičemž se jedná o heuristickou metodu, která nezaručuje nalezené nejlepšího řešení, tudíž výsledkem nemusí být vždy správný tvar slova.

Základem pro správný běh algoritmu, je specifikovaná množina koncovek v daném jazyce, pomocí které dochází k zmíněnému odstranění jednotlivých částí. V současné době existuje několik variant stematizačních algoritmů, příkladem může být Porter [3].

Proces stematizace není dokonalý, mezi nejčastější problémy patří:

- Mnohoznačná slova (zebra, houba).
- Jednoduchost.
- Nebere v potaz pravidla specifikovaná ve vstupním jazyce.
- Vysoká závislosti na vstupním jazyce.
- Heuristický přístup zapříčinující nesprávné zkrácení slova.

2.3 Lemmatizace

Lemmatizace [2] je proces, který sdílí myšlenku redukce slova do základního tvaru slova spolu se stematizací. Výsledkem algoritmu je slovníkový tvar (tzn. *lemma*). V porovnání se stematizační technikou, dochází u lemmatizace k detailnější analýze slova z pohledu jazyka (analýza mluvnických kategorií). Algoritmus tudíž provádí dodatečné výpočty, díky kterých lze dospět k úspěšnějším výsledkům.

Lemmatizace se potýká s následujícími limitacemi:

- Mnohoznačná slova (jaguár, jazyk).
- Víceslovňá spojení, idiomy.
- Výpočetně náročnější operace.

2.4 Tokenizace

Proces, při kterém je vstupní text podroben tokenizaci [4], má za příčinu rozdělení větších elementů na menší. Příkladem může být rozdělení odstavce na věty, případně věty na jednotlivé termy.

V průběhu předzpracování vstupního textu bývá tato operace provedena mezi prvními. Výstupem je sekvence slov, která jsou podrobena dalším operacím z fáze předzpracování textu. Po úspěšném provedení všech metod z této fáze je někdy také provedena oprava možných chyb generovaných z lemmatizačních, stematizačních metod v *postprocessingu*. Takto zpracovaná data jsou využita jako vstup pro jednotlivé modely reprezentující způsob, kterým lze text vektorově popsat.

2.5 Ostatní metody

V kapitolách výše byly popsány metody předzpracování přirozeného jazyka, které jsou skoro vždy využity. Důležité je ale zdůraznit, že se nejedná vždy pouze o ty. Bývá dobrým zvykem vždy provést **explorační analýzu** nad vstupním jazykovým korpusem s ohledem na vizualizaci nejčetnějších slov vzhledem k třídě, identifikaci relevantních slov. Extrahované informace pak využít při definici metod na předzpracování. Po normalizaci textových řetězců zopakovat předchozí kroky a tímto iterativním procesem se pokusit najít nejlepší způsob, jak vstupní textová data předzpracovat. Zároveň je potřeba myslet, že každý z modelů reaguje na předzpracování jiným způsobem. Mezi jiné metody předzpracování mohou patřit například:

- **Odstranění numerických hodnot** se provádí jelikož numerické hodnoty jsou jazykově složitě interpretovatelné, a tak nenesou většinou žádnou relevantní informaci pro modelovaný jazykový model.

- **Odstranění bílých znaků** bývá často jednou z prvních metod, jelikož bývají slova oddělena více než jedním bílým znakem.
- Důvodem **odstranění krátkých slov** bývá myšlenka, že tato krátká slova málokdy mívají relevantní informaci a často se jedná o překlepy.
- **Odstranění definovaných slov** je provedeno po **explorační analýze**, kdy byly analytikem identifikovány textové segmenty, s kterými není potřeba pracovat. Příkladem mohou být například kapitoly v knihách (Kapitola 1).
- **Nahrazení pomocí tokenů** je přístup, který se provádí například u klasifikace sentimentu, kdy pozitivní emoji bývají substituovány přesně definovanou abecedou (*převod smajlíků do jejich vyznámové formy*).
- **Znaková transformace** vychází z myšlenky, že uživatelé využívají stejný znak na vyjádření určité věci. Příkladem mohou být uvozovky, které se využívají ve dvou tvarech ', ".

Kapitola 3

Reprezentace textových dokumentů

Počítačové zpracování přirozeného jazyka se zabývá tvorbou matematických modelů, které jsou schopny zachytit a případně analyzovat přirozený jazyk. Při zpracování textu, v matematických modelech, je potřeba zvolit způsob, kterým budou jednotlivá slova reprezentovaná. Jelikož většina modelů strojového učení není schopna zpracovávat text v normální podobě, je nutné převést slova do podoby číselných vektorů.

Po transformaci nestrukturovaného vstupního formátu textu do strukturovaného získáme základní bloky, na které lze aplikovat některá z níže popsaných technik. Každá z nich přináší určité výhody a nevýhody a měla by být strategicky vybrána. Obecně lze tyto techniky rozdělit do 3 skupin:

- Základní one hot kódování.
- Metody založené na frekvenci výskytu slov (*BoW, TF-IDF*).
- Bezkontextové vnoření slov (*Word2Vec, GloVe*).
- Kontextové vnoření slov (*BERT*).

3.1 One hot kódování

Základní metoda, kterou lze získat numerická reprezentace slova, je one hot kódování. Při této reprezentaci je vytvořen vektor o velikosti slovníku, kde každému slovu je přiřazena jedinečná pozice. Vektor je tedy nastaven na samé nuly, kromě pozice odpovídající pozici daného slova. Na této pozici má vektor vynesenou hodnotu 1.

Takovéto zakódování textového dokumentu obnáší vytvoření slovníku a následnou transformaci každé věty do 2D matice o velikosti (n, m) , kde n je počet tokenů vyskytujících se v dané větě a m velikost slovníku. Tato reprezentace, nikterak nepoopravená, tedy produkuje řídkou matici obsahující malé množství nenulových pozic zároveň s nekonzistentním tvarem matice.

3.2 Bag of words

Bag of words, neboli BoW [5], je základní metoda založená na počtu výskytu slov v daném dokumentu. Metoda přistupuje k reprezentaci podobným přístupem jako one hot kódování, zmíněné v kapitole 3.1 na stránce 21, s rozdílem, že reprezentace je zmáčknuta do jednoho vektoru, rozdíl proti 2D reprezentaci, s tím, že na každé pozici je zaznamenaná informace o četnosti daného tokenu.

Všechny věty jsou reprezentovány stejně velkým vektorem, například řádkovým vektorem $(1, n)$, kde n popisuje velikost slovníku.

3.3 TF-IDF

TF-IDF [6] je metoda reprezentace dokumentu, která se skládá ze dvou složek. První TF (*anglicky Term Frequency*) neboli četnost slova v dokumentu. A druhé složky IDF (*anglicky Inverse Document Frequency*) reprezentující převrácenou četnost slova vzhledem ke všem dokumentům.

TF složka této reprezentace má určitou podobnost s předchozím BoW modelem zmíněným v předchozí kapitole 3.2. Je založena na četnosti slov v dokumentu s tím, že se provádí normalizace vzhledem k velikosti dokumentu. Tímto se předchází zvýhodnění delších dokumentů, jelikož v nich se předpokládá větší četnost slov. Výpočet jednotlivých složek vektoru znázorňuje následující matematická formule, kde t reprezentuje slovo neboli term, a d dokument. Výsledkem je tedy skalár popisující normalizovanou četnost slova vzhledem k dokumentu.

$$tf(t, d) = \frac{\text{četnost } t \text{ v } d}{|d|}$$

IDF složka reprezentuje tedy převrácenou četnost slova vzhledem ke všem vstupním dokumentům. Tato složka nám umožňuje správně určit významnost slova, kdy se předpokládá, že velice frekventovaná slova mívají menší informativní význam než ta, která se vyskytuje méně. Pro představu jsou za tyto slova často považována stop-slova (spojky, předložky).

Výpočet této složky rozdělíme do dvou částí, kde v první dochází k spočtení významnosti slova v rámci celého jazykového korpusu neboli celé množiny vstupních dokumentů. Následující formule detailněji ilustruje výpočet, který je dost podobný výpočtu TF složky s tím rozdílem, že počítáme v kolika dokumentech z jazykového corpusu c najdeme slovo t .

$$df(t) = \text{výskyt } t \text{ v } c$$

Skalár získaný z výpočtu reprezentuje v kolika dokumentech z jazykového korpusu se dané slovo vyskytlo. Účel složky IDF je přesně opačný. Snaží se zvýhodnit přesně taková slova, která se vyskytují v méně dokumentech. Proto je nutné provést inverzní operaci, a tím získat správnou hodnotu.

Dochází tedy k využití předchozího výpočtu, skaláru DF (*Document Frequency*), který je umístěn ve zlomku ve jmenovateli, kdy společně s tím je v čitateli mohutnost jazykového korpusu. Výsledkem výpočtu je složka IDF, která reprezentuje převrácenou četnost významu slova.

$$idf(t) = \log \frac{|c|}{df}$$

Z formule níže můžeme vidět, že pomocí operace násobení mezi složkou TF (četnost slova v dokumentu) a IDF (převrácená četnost slova vzhledem ke všem dokumentům), jsme schopní vytvořit finální vektor, popisující numerickou reprezentaci dokumentu.

$$tf\text{-}idf(t, d) = tf(t, d) * idf(t)$$

3.4 Word2Vec

Doteď výše v kapitolách byly popsány deterministické metody, jak získat numerickou reprezentaci dokumentu. Tyto metody byly využívány do roku 2013, kdy v té době vznikl, dnes již populární, model pro vnoření slov Word2Vec. Za vznikem tohoto modelu stojí Tomáš Mikulov, který představil v publikovaném článku zcela nový pohled na zpracování přirozeného jazyka.

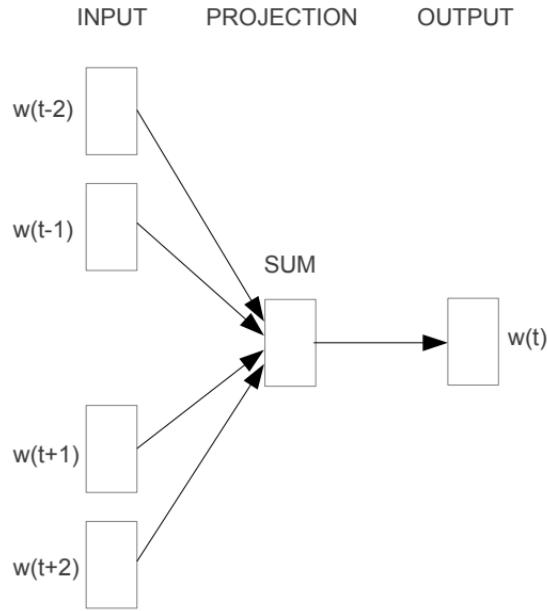
Při porovnání tohoto modelu s předešlými modely, je Word2Vec [7] nedeterministickou metodou, která se snaží zkonstruovat numerickou reprezentaci slova za pomocí neuronových sítí.

Výhodou vytvořených vektorů slov tímto přístupem se ukázalo zachycení sémantické složky jazyka a podobnost mezi různými slovy. Další neméně podstatnými výhodami je zbavení se řídké reprezentace a zároveň menší paměťové a časové nároky, které dovolují trénování na mnohem větším množství dat a získání tak kvalitnějších vektorů.

Algoritmus je tvořen pomocí dvou odlišných architektur, které budou popsány níže v podkapitolách 3.4.1 a 3.4.2. Ve zkratce **Continuous Bag of Words** predikuje slovo na základě jeho okolí, přičemž **Skip Gram** volí opačný přístup predikce okolí v závislosti na vstupním slově.

3.4.1 Continuous Bag of Words

Vektory pro reprezentaci slov jsou získány z prediktivního modelu Continuous Bag of Words (*CBOW*), který z kontextu slova (slov z okolí) předpovídá dané slovo. Model je prezentován neuronovou sítí (viz obrázek č. 3.1). Na vstupu neuronové sítě jsou slova z okolí predikovaného slova, kde velikost okolí je jedním ze vstupních parametrů. Výstupem je vektor reprezentující predikované slovo. Naučením takovéto neuronové sítě získáme požadovanou vektorovou reprezentaci našich slov.

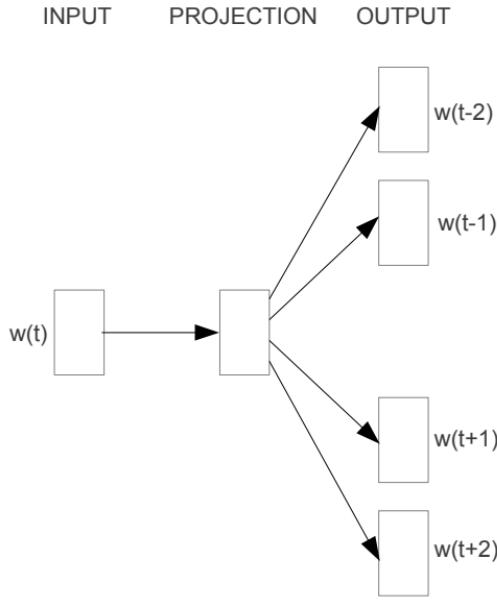


Obrázek 3.1: CBOW architektura [8]

Nevýhodou této architektury můžeme považovat nemožnost zachycení mnohoznačnosti slova. Příkladem může být slovo jaguár. Reprezentuje jak kočkovitou šelmu, tak i automobilovou firmu zabývající se výrobou luxusních a sportovních automobilů. Pro představu výsledkem pak bude vektor umístěn v prostoru mezi oběma slovy.

3.4.2 Skip gram

Druhou možnou architekturou neuronové sítě (viz obrázek č. 3.2) pro získání vektorové reprezentace slov je model Skip gram. Tento model přistupuje k výpočtu odlišně, než bylo zmíněno u modelu CBOW (viz předcházející kapitola 3.4.1). Narozdíl od predikce slova v závislosti na okolí se tento model snaží z centrálního slova predikovat jeho okolí neboli jeho kontext.



Obrázek 3.2: Skip gram architektura [8]

Oproti předešlému přístupu je model schopen zachytit více sémantických významů k jednomu slovu, a tak se vypořádat lépe se mnohoznačnosti slova.

3.5 GloVe

Dalším z modelů pro získání vektorové reprezentace slov je GloVe [9] (*anglicky Global Vectors for Word Representation*). Model je navržen pro optimalizaci informace z matice společného výskytu za účelem hledání významu jednotlivých slov. Myšlenka této metody v určitém směru vylepšuje předchozí model Word2Vec, jelikož v zachycování vztahu mezi slovy je zahrnuta mimo lokální informaci (parametr definující velikost okolí ve Word2Vec modelu) i informace globální.

3.6 Vektorová reprezentace vycházející z architektury Transformeru

Vektorová reprezentace získána pomocí vnoření slov (*word embeddings*) algoritmy Word2Vec (kapitola 3.4) a GloVe (kapitola 3.5) poskytuje pouze informaci o umístění vektoru v mnohodimenzionálním prostoru. I když se některé sofistikovanější modely pokouší zachytit kontextuální vlastnosti slova, přesto bývá vektor fixní, a tak mívaly vektory slov vlastnost podobnosti mezi sebou, nikoliv ale podobnosti mezi stejnými slovy napříč různými větami. Tato kontextuální nedostatečnost je vyřešena představením článku [10] *BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding*.

3.6.1 BERT (Bidirectional Encoder Representations from Transformer)

BERT model vychází z architektury neuronové sítě **Transformer**. V kapitole 5 byla představena na problému strojového překladu, kde část enkodéru čte vstupní sekvenci slov a část dekodéru generuje výstup v podobě sekvence slov v jiném jazyce. BERT model využívá jednu část této architektury, a to enkodér.

Předchozí sekvenční modely měly nedostatečnost právě v sekvenčním čtení vstupu v určitém směru zleva-doprava nebo zprava-doleva (někdy i kombinovaně). Hlavním problémem byla nemožnost paralelizace výpočtu, a tak i když modely měly naučený kvalitní jazykový model, i přesto byly mimořádně limitovány. BERT model umožňuje paralelizovat výpočet a zároveň novým mechanismem s názvem **self-attention** zachytit kontextuální význam slova na základě celé vstupní sekvence.

Tyto vlastnosti byly využity při předtrénovaní BERT modelu na obrovském jazykovém korpusu. Takto natrénovaný model lze následně využít při řešení jiných úloh, kde již vycházíme z předtrénovaného jazykového modelu (*přenesené učení, angl. transfer learning*).

3.6.2 Popis architektury

V článku [10] byly představeny dva typy modelu:

- **BERT-Base** - 12 enkodér vrstev zřetězených za sebou, 768 velikost dimenze, 12 multihead-attention, 110 miliónů parametrů.
- **BERT-Large** - 24 enkodér vrstev, 1024 velikost dimenze, 16 multihead-attention, 340 miliónů parametrů.

Každá z enkodér vrstvy je složena z **multihead self-attention** mechanismu, který následuje **hustě propojená dopředná neuronová síť**. Více v kapitole č. 5.6.

3.6.3 Předtrénování modelu

V článku [10] můžeme najít popis procesu trénování. Proces je založen na dvou různých úlohách strojového zpracování přirozeného jazyka. **Predikci neznámého tokenu** vzhledem k jeho kontextu a **odhadu návaznosti vět**.

Vstupy do procesu předtrénování byly generovány z kombinace datových zdrojů BooksCorpus (800 milionu slov) a anglické verze Wikipedie (2500 milionu slov).

Maskované modelování jazyka - MLM

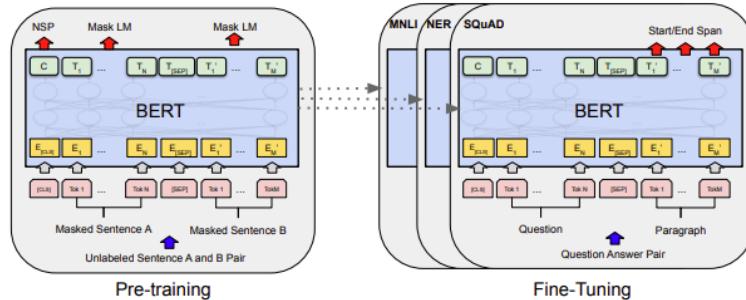
Pro úlohu **predikce neznámého tokenu** se využívá proces maskování, neboli v článku odkazováno na *Cloze task* [10]. Patnáct procent náhodných tokenů ze vstupních záznamů je maskováno, přičemž finální vrstva provádí ve spojení se softmax funkcí predikci, jako je zvykem u standartního jazykového modelování. Článek zároveň popisuje, jakým způsobem dochází k maskování tokenů:

- V 80 % případů nahrazen maskovacím tokenem *[MASK]*.
- 10 % případů je substituováno jiným náhodným tokenem ze slovníku.
- V 10 % je token nezměněn.

Důvodem je předcházení rozdílu mezi předtrénováním a fine-tuningem modelu. Pakliže by byla využita pouze substituce *[MASK]*, je možné, že model by nedosahoval takových výsledků.

Predikce návaznosti vět - NSP

Autoři zvolili druhou úlohu (*anglicky Next Sentence Prediction*) s myšlenkou zachycení vazeb mezi dvěma větami v jazykovém modelu. Důvodem bývá časté využívání této vlastnosti například u systému QA (*Question Answering*). Během trénování na této úloze je vybráno 50 % dvojic, jež jsou ve správném pořadí a 50 %, které nikoliv. Rozlišení obou vět na výstupu je umístěno v speciálním tokenu *[CLS]*, který lze vidět na obrázku č. 3.3.

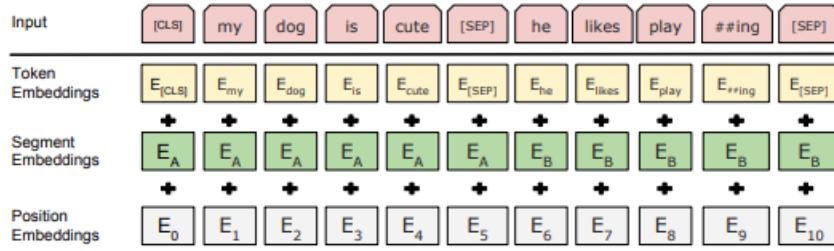


Obrázek 3.3: Způsob předtrénování BERT modelu a jeho využití (*[CLS]* token zkráceně C) [10]

3.6.4 Předzpracování textu

Autoři BERT modelu přišli také se specifickou podobou vstupu, kterou je nutné dodržet. Vstup tvoří kombinace následujících vektorů:

- **Poziční vektor** (*position embeddings*) označující pozici slova ve větě.
- Vektor označující **segment** (*segment embeddings*), který slouží k rozeznání věty u predikce následování dvou vět (*NSP*). Využíváno především u *QA*.
- Vektor **tokenů** (*token embeddings*) obsahující již samotné tokeny generované algoritmem **WordPiece** (sekce 3.6.5) a speciální tokeny.



Obrázek 3.4: Vstupní vektor do BERT modelu [10]

V řádku s označením *Input* (červený řádek) na obrázku č. 10.2 lze pozorovat funkčnost **WordPiece** tokenizace společně s obsažením speciálního tokenu *[CLS]*, který slouží pro ohrazení začátku věty. Token *[SEP]* slouží k označení konce věty nebo také můžeme jeho vlastnost interpretovat jako token, který rozděluje dvě věty od sebe. BERT model má fixně definovaný vstup, a proto podobně jako u ostatních modelů je potřeba využít doplňovací (*padding*) mechanismus a doplnit některé věty o *[PAD]* tokeny. Některé ze vstupu mohou být delší, tudíž mimo *padding* může dojít k operaci zkrácení (*truncation*).

3.6.5 WordPiece tokenizace

V kapitole 2.4 byl popsán krátce mechanismus, který rozděluje větší části textu na menší. Typickým vstupem do neuronových sítí například u algoritmu Word2Vec (kapitola 3.4) bývají samostatné tokeny (slova), díky kterých je postupně získána vektorová reprezentace. V případě BERT modelu nedochází k prostému rozdělení na jednotlivá slova. Je využit komplexnější algoritmus řešící mimo jiné problém **OOV** (*out of vocabulary*). Kvůli absenci slova v trénovací množině slov neexistuje vektorová reprezentace daného slova při produkčním využití. Hyperparametrem WordPiece algoritmu je velikost slovníku, přičemž i s tímto omezením existuje pro většinu slov vektorová reprezentace.

Uživatelem je definován hyperparametr popisující velikost slovníku a dodána trénovací množina s textovými daty. V těchto datech jsou nalezeny nejmenší stavební bloky v podobě samostatných znaků. V každém iteračním kroku je skóre pro bigram vypočteno následovně:

$$\frac{\text{Frekvence bigramu}}{\text{Frekvence první části bigramu} \times \text{Frekvence druhé části bigramu}}$$

Bigram s největším skórem je uložen po inicializaci k samostatným znakům a následně ke všem již uloženým segmentům. Současně jsou všechny bigramy nahrazeny uloženým segmentem. Algoritmus běží dokud aktuální velikost slovníku se nerovná definované velikosti nebo neexistuje žádná další bigram. Na obrázku č. 10.2 si pozorný čtenář může povšimnout výskytu tokenu **##ing**. Toto značí segment, který následuje jiný větný segment. Vznik těchto segmentů zapříčinuje trénovací proces algoritmu. Vstupní jazykový korpus je segmentován prvně na jednotlivé tokeny v podobě slov a následně jsou slova rozdělena na znaky. Pro ukázkou slovo **playing** bude rozděleno na 7 částí ($p, ##l, ##a, ##y, ##i, ##n, ##g$), stejným způsobem jakým pracuje algoritmus **WordPiece**.

3.6.6 Fine-Tuning BERT

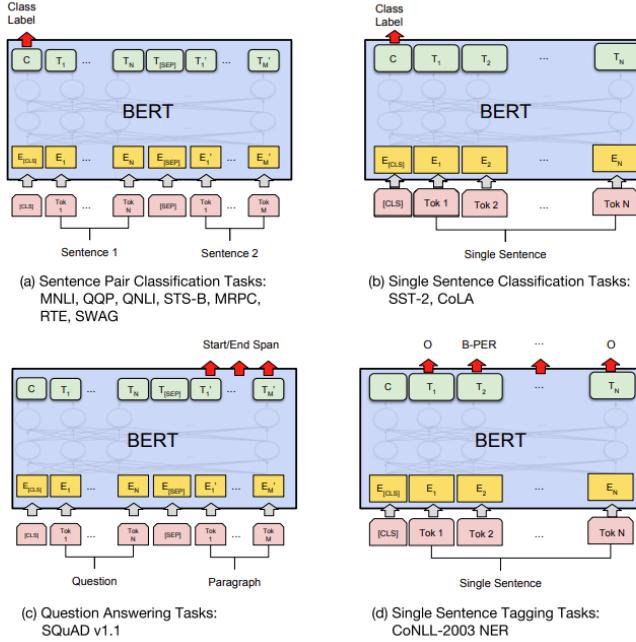
Po předtrénování BERT modelu získáme jazykový model určitého jazyku (někdy i více), jenž můžeme využít na následné problémy. Uživatel tímto způsobem využívá techniku přeneseného učení (*angl. transfer learning*), kdy stačí menší množství dat nato, aby bylo dosaženo kvalitních výsledků. V případě BERT modelu jsou aplikací tohoto přístupu získány state-of-art výsledky u širokého množství úloh zpracování přirozeného jazyka. Například se může jednat o klasifikační úlohu. U ní musí být dodržen specifický vstup BERT modelu, a tímto získány vektory reprezentující vstupní slova. Slova následují do kupříkladu hustě propojené dopředné sítě se sigmoid nebo softmax funkcí na výstupu (dle úlohy).

3.6.7 Možnosti modelu

Propůjčeným obrázkem č. 3.5, z článku [10], můžeme demonstrovat široké spektrum úloh, které lze modelem vyřešit. Způsob využití se liší vstupem a částí výstupu, kterou uživatel využije. Příkladem mohou být:

- **Rozpoznávání entit** (*anglicky Named-entity recognition*) při němž jsou využity vektory jednotlivých tokenů přímo.
- U **klasifikačního problému** je vstupem textový řetězec. Výstup může být využit různý. Například na obrázku vlevo nahoře je využit *[CLS]* token.
- U systému, který **odpovídá na otázky** (*anglicky Question Answering - QA*) položené uživatelem v přirozeném jazyce využijeme na dotrénování vstup dvou vět, otázky a paragrafu.

Použitelný výstup z modelu je druhý segment, který je identifikovatelný díky *segment embeddings*.



Obrázek 3.5: Aplikace BERT modelu u rozličných problémů zpracování přirozeného jazyka [10]

3.6.8 Ostatní modely

Postupnou evolucí BERT modelu začali vznikat deriváty. Většina z nich obohacovaly původní model o nové vlastnosti. V praktické části této diplomové práce bylo experimentováno s 2 deriváty, kterými jsou **ELECTRA** a **DistilBERT**. Kromě zmíněných jich existuje mnoho. Autor práce doporučuje pro experimentální činnost využít knihovnu *HuggingFace* [11] popsanou v kapitole č. 7.4. Ke všem hostovaným modelům zde můžeme najít zároveň popisné informace.

DistilBERT

V článku [12] *DistilBERT, a distilled version of BERT: smaller, faster, cheaper and lighter* byl představen jako jeden derivát, který se liší od originálního BERT modelu v podobě přístupu učení. Je známo, že původní BERT model obsahuje velké množství vah, díky kterých je jeho využití limitující. V některých případech potřebujeme klientovi vrátit odpověď s co nejmenším zpožděním, případně data nemáme možnost posílat na server, a tak je třeba model zprovoznit na koncových zařízeních.

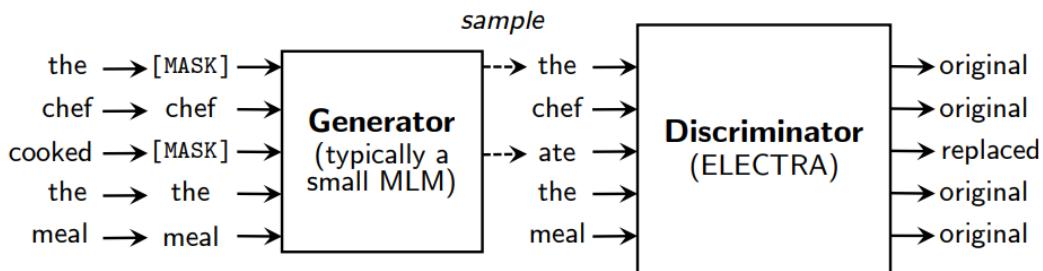
Kvůli zmíněných problémů vznikla odlehčená verze BERT modelu využívající techniku *distillation*, někdy taky známo jako *teacher-student learning*. Myšlenka, která za touto technikou stojí

je učení menšího modelu z chyb modelu většího, a tak zachytit onu generalizaci, která je podchycena ve velkém modelu.

Experimentální činností bylo zjištěno, že tento přístup hraje velký vliv. Díky nějž lze modifikací učení získat lepší výsledky. V případě DistilBERT modelu se jedná o zmenšenou verzi o 40 % proti původnímu BERT modelu se zárukou až 60% zrychlení. I když model obsahuje menší počet vah garantuje z 97 % úspěšnost jako model původní.

ELECTRA

V podkapitole 3.6.4 byla čtenáři představena technika, kterou je originální BERT model předtrénován. Trénování modelu je založeno na dvou různých úlohách. **Predikci maskovaného tokenu** a predikci, **zda věta B následuje větu A** . V článku [13] *ELECTRA: PRE-TRAINING TEXT ENCODERS AS DISCRIMINATORS RATHER THAN GENERATORS* byla představena jiná metodika, kterou lze využít k učení modelu založeném na Transformer architektuře. Obrázek č. 3.6 tento způsob vizualizuje. Nahrazení MLM (*masked language modeling*) za metodu RTD (*replace token detection*) bylo dosaženo lepších výsledků než u originálního modelu. Myšlenkou modelu je využít dvě komponenty **generátor** a **diskriminátor**. Obě založeny na enkodér části Transformeru. **Generátor** z maskovaných vstupů podobně jako u originálu produkuje plný výstup. Ten následně zhodnocuje **diskriminátor** s úmyslem zhodnocení, jestli token je originální nebo byl nahrazen. Autoři popisují, že příčinou lepších výsledků bývá právě úprava vnitřních vah modelu v závislosti na celé vstupní sekvenci a nejen z 15 % maskovaných tokenů.



Obrázek 3.6: ELECTRA [14]

Kapitola 4

Modely pro klasifikaci

Po provedení procesu předzpracování a odvození informativních hodnot ze vstupního jazykového korpusu následuje vybrání modelu takového, který bude řešit námi definovaný problém. V této kapitole budou popsány určité modely určené pro problém klasifikace. V podkapitole 4.1 budou rozebrány klasické modely strojového učení, které jsou založeny na rozdělení prostoru dle určitých datových struktur, práci s pravděpodobností nebo například užití metriky a následného výběru nejpodobnějších nebo nejbližších prvků. Podkapitola 4.2 se bude zabývat deskripcí matematického modelu neuronových sítí.

4.1 Klasické modely

Myšlenkou těchto modelů bývá na základě trénovacích dat definovat interní stav, dle kterého lze s určitou jistotou odhadovat nově vstupující záznamy. Zmíněný vnitřní stav může být například jednoduchá přímka v dvourozměrném prostoru, která rozděluje prostor na dvě části, a tak je schopna predikovat výsledek, dle jeho umístění nad nebo pod přímou. V rámci této diplomové práce byly některé modely využity z účelem následného porovnání s modely neuronových sítí, které byly hlavní doménou práce.

4.1.1 Naive-Bayes

Je algoritmus založen na práci s pravděpodobností. Přesněji na aplikaci Bayesova teorému.

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

Teorém předpokládá nezávislost mezi jednotlivými vlastnostmi. Aplikací tohoto algoritmu při řešení problému v NLP uvažujme jako vlastnost počet výskytu daného slova (viz kapitola 3.3 nebo 3.2). Pokud budeme brát v potaz klasifikační problém odhadu polarity textu, pak algoritmus v trénovací fází počítá pravděpodobnost výskytu slova vzhledem k třídě reprezentující kladný nebo

záporný sentiment. Při predikci, představíme si větu "Mám rád NLP", dochází k výpočtu pravděpodobnosti připadnuti věty k danému sentimentu. Vybrán je sentiment s vyšší pravděpodobností. Samotný výpočet pravděpodobnosti vzhledem k pozitivnímu sentimentu by vypadal následovně.

$$P(\text{Mám rád NLP} \mid \text{pozitivní}) = P(\text{Mám} \mid \text{pozitivní}) * P(\text{rád} \mid \text{pozitivní}) * P(\text{NLP} \mid \text{pozitivní})$$

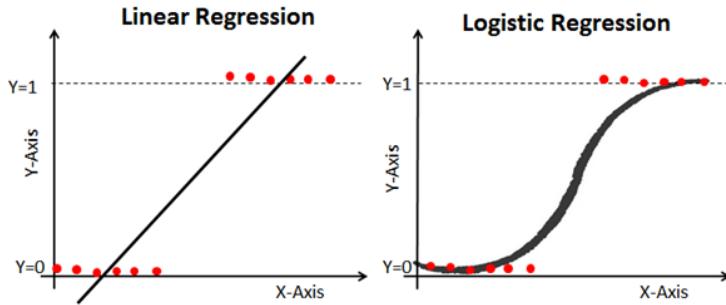
4.1.2 Logistická regrese

Z obrázku č. 4.1 lze pozorovat, že s menší zámenou můžeme definovat řešení pro dva odlišné problémy. Pakliže bereme v potaz lineární regresi, snažíme se (pro zjednodušení) ve dvoudimenziálním prostoru definovat přímku, která minimalizuje rozdíl čtverců od přímky, tak abychom byli schopni odhadnout výstupní spojitou proměnou s co nejmenší odchylkou od přímky vytvořené na trénovacích datech. Typickým využitím mohou být odhady ceny nemovitostí, vzhledem k vlastnostem velikosti bytu, lokalitě, počtu pokojů a tak dále. V doméně problémů textových dat lze například lineární regresí modelovat problém datování textu. Představme si, že každé slovo je definováno n počtem vlastností a po provedení kombinace těchto vlastností slov celého dokumentu jsme schopni s určitou jistotou odhadnout, kdy bylo dílo napsáno. Na pravé straně v obrázku, pak můžeme vidět logistickou regresi, která mimo jiné byla využita v této diplomové práci. Po provedení zámeny za sigmoid funkci lze podobný přístup aplikovat pro zjištění dané třídy. Tento problém se již nesnaží minimalizovat chybu od nadroviny, ale maximalizovat pravděpodobnost. Logistickou regresi lze kategorizovat do 3 skupin [15]:

- **Binomická logistická regrese** se na výstupu rozhoduje mezi dvěma třídami. Příkladem může být klasifikace sentimentu.
- **Multinomická logistická regrese** oproti **binomické** již pracuje s více třídami. A tak ji lze využít pro klasifikaci autorů.
- **Ordinální** pracuje s výstupem, který má ordinální vlastnosti. Tudíž pro něj lze definovat uspořádání. Například jak moc byla recenze pozitivní na škále od 0 do 5.

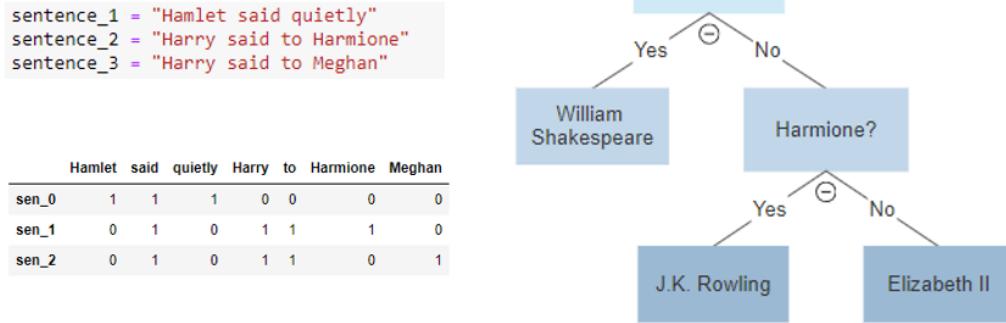
4.1.3 Rozhodovací strom

Hlavní myšlenkou tohoto modelu je rozdělit problémovou doménu pomocí datové struktury stromu. Tento strom je vytvořen na trénovací množině dat, kde v každém kroku musí nalézt takovou kombinaci atributu a hodnoty, která rozděluje problém s ohledem minimalizace entropie vzhledem k jedné části. Z takhle vytvořeného stromu, který se zaměřuje na klasifikační problém, pak lze odhadnout třídu nově vstupujícího záznamu. U našeho problému pracujeme s textovými daty, a tak bude uveden příklad, jak se rozhodovací strom bude vnitřně chovat. Chování stromu bude jednoduše



Obrázek 4.1: Porovnání lineární a logistické regrese [15]

vizualizováno na obrázku č. 4.2 společně se slovním doprovodem. Představme si 3 autory: **Joanne Rowlingovou**, **William Shakespeare** a **královnu Alžbětu II.** Každý z těchto autorů napsal určitý počet knih. Prvně musíme tyto textová data reprezentovat číselně. Obrázek pracuje s BoW reprezentací (viz kapitola 3.2). Z takovéto numerické reprezentace je následně vytvořena stromová struktura se schopností rozeznání autora. Strom po určitém množství kroků dorazí do listového uzlu, v kterém s určitou pravděpodobnosti (zde s jistotou) predikuje autora.



Obrázek 4.2: Vizualizace BoW vektorizace s modelem rozhodovacího stromu

4.1.4 Nejbližší sousedé

Každý textový dokument lze reprezentovat pomocí číselného vektoru o definované velikosti. Tyto vektory lze získat různými přístupy. Příkladem mohou být frekvenční modely BoW (zmíněné v kapitole 3.2 na stránce 22), složitější vektory získané z neuronových sítí bez hlubšího kontextového pochopení Word2Vec (sekce 3.4) a nebo pomocí state-of-art modelu BERT (*anglicky Bidirectional Encoder Representations from Transformers*). Po vektorizaci všech dokumentů vytváříme mnichodimenzionální prostor, ve kterém platí základní vlastnosti metrického prostoru, a tak lze definovat pojem vzdálenosti, nazvaný metrika.

Definice 1 (Metrika) [16] *Nechť X je množina a d funkce přiřazující každé dvojici (x, y) z X nezáporné reálné číslo $d(x, y)$ mající vlastnosti:*

1. pro $x, y \in X$ je $d(x, y) = 0$ právě když $x = y$;
2. (symetrie) $d(x, y) = d(y, x)$ pro každé $x, y \in X$;
3. (trojúhelníková nerovnost) $d(x, y) \leq d(x, z) + d(z, y)$ pro každé $x, y, z \in X$.

Funkce d se pak nazývá **metrika** na X a dvojice (X, d) se nazývá **metrický prostor**.

Nad takto definovaným prostorem lze určit vzdálenosti nebo podobnost mezi dvěma vektory a to využít pro zjištění, jak dva dokumenty jsou od sebe vzdáleny nebo jak jsou si podobné (od teď zmíněna jen vzdálenost). Díky této vlastnosti, pak lze vytvořit systém, kterým jsme schopni odhadnout nejpodobnější dokumenty vzhledem k jasně vybranému dokumentu. Pokud půjdeme dále můžeme tuhle myšlenku využít pro klasifikaci, a tak při řešení problému autorizace předpokládat, že daný dokument bude nejblíže dokumentům, které napsal stejný autor. Případně odhalit prohrešek plagiátorství.

4.2 Modely založené na neuronových sítích

Obecně neuronové sítě jsou typem výpočetního modelu, který je inspirován poznatky o neuronech a neuronových sítích živých organismů. Existují různé typy neuronových sítí, kdy v rámci tohoto projektu byly použity husté neuronové sítě, různé typy rekurentních neuronových sítí a sítě konvoluční. Vstupem pro všechny tyto sítě, v textové analýze, mohou být v jednodušších modelech vektory získané z TF-IDF (kapitola 3.3), BoW (kapitola 3.2) reprezentace ale povětšinou jsou to pak vektory získané pomocí vnoření slov (*word embeddings*).

4.2.1 Hustá neuronová síť

Pokud zřetězíme několik skrytých vrstev za sebe využíváme architekturu husté neuronové sítě. Tento model funguje na principu hustého propojení všech neuronů. Vstupem do neuronu ve vrstvě $n+1$ jsou všechny výstupy z vrstvy n . Myšlenkou je pouhá závislost neuronu n na výstupech z předchozí vrstvy. Neuron tedy provádí skalární součin mezi vektorem vah a vektorem vstupních hodnot, kdy na tento výsledný skalár je aplikovaná aktivační funkce a výsledek reprezentuje výstup daného neuronu. Hustá neuronová síť, pak pracuje se sekvencí (*větou o n termech*), tak že jednotlivé termíny jsou brány individuálně.

4.2.2 RNN

Architektura rekurentní neuronové sítě (*anglicky Recurrent neural network, neboli zkráceně RNN*) [17] je založena na myšlence závislosti aktuálního výstupu nejen na aktuálním vstupu, ale i na předchozím stavu neuronu. Z této myšlenky byly postupně vyvinuty pokročilejší typy (GRU 4.2.5, LSTM 4.2.3), které budou popsány níže. Oba typy byly vyvinuty ze základního modelu, jednoduché RNN (*anglicky Vanilla RNN*). Úvodem je ještě nutné dodat, že rekurentní sítě jsou vhodné pro zpracování sekvenčních dat, tudíž jsou vhodné pro NLP úkoly.

Jednoduchá RNN pracuje mimo aktuální vstup také s vnitřní pamětí, která je v průběhu zpracování vstupní sekvence postupně modifikována a aplikována na aktuální vstup. Přesněji tedy podle následujících dvou formulí 4.2.2:

$$\begin{aligned} h_i &= f(W_{hh}h_{i-1} + W_{hx}x_i) \\ y_i &= W_{yh}h_i \end{aligned}$$

Neuron pracuje se vstupem v aktuálním čase x_i , a obsahem paměti z předchozího kroku h_{i-1} , pomocí kterých dochází k přepočtu vnitřní paměti s využitím dvou váhových matic W_{hh} a W_{hx} a aktivační funkce f , která ve většině případů je definována jako hyperbolický tangens. Výstup neuronu je pak získán pomocí vnitřní paměti a matice W_{yh} .

4.2.3 LSTM

Složitější typy RNN byly vynalezeny se záměrem předejít problému, které se vyskytovaly u jednoduchých RNN, a to především neschopnosti udržet závislosti v delší sekvenči. Představme si větu "*Byl jsem ve Francii 10 let, užil jsem si tam mnoho zábavy a zároveň se naučil mluvit plynule X*". Jednoduchá rekurentní síť by měla problém odhadnout term X , právě z důvodu zmíněné neschopnosti. LSTM [18] (*anglicky Long short-term memory*) je typ neuronové sítě, který si dokáže zapamatovat tyto závislosti napříč dlouhými vstupy. Tuto vlastnost má díky složitější stavby buňky, která se skládá ze 3 bran umožňující udržet konkrétní informaci pro potřebnou délku času.

- Na základě **zapomínací brány** se rozhoduje kolik informace se má v paměti zapomenout.
- **Vstupní brána** reguluje množství informace, která má být zaznamenána v paměti.
- **Výstupní brána** reguluje množství informace, která má být použita z vnitřní paměti při výpočtu výstupu.

4.2.4 Obousměrná LSTM

Úmysl obousměrné LSTM sítě spočívá v rozšíření o vlastnost zpracování vstupní sekvence v opačném pořadí. Tudíž jde o dvě LSTM sítě vedle sebe, které zpracovávají vstupní sekvenči, každá v jiném pořadí. Výstupem je sloučený vektor podle dané operace. Zpracováním vstupní sekvence z obou stran, tak může tato síť zachytit informaci, kterou by jednosměrná LSTM zachytit nedokázala.

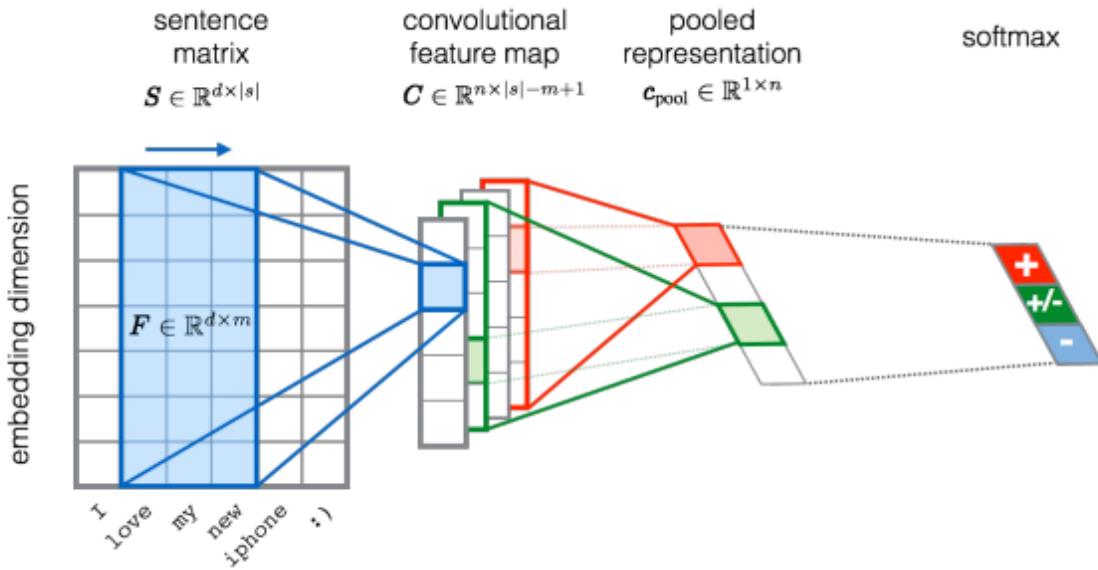
4.2.5 GRU

GRU [19] (*anglicky Gated recurrent unit*) síť sdílí stejnou myšlenku jako předchozí LSTM síť, s rozdílem, že struktura buňky neobsahuje jednotku paměti, ale pouze aktuální stav. Nedosahuje takových možností uchování dlouhých závislostí, ale trénování této sítě bývá mnohem rychlejší, než u sítí LSTM.

4.2.6 Konvoluční síť

Další alternativou [20], jak řešit klasifikační problém v rámci domény zpracování mluveného slova, je architektura neuronové sítě založena na matematickém operátoru konvoluce. Fungování této sítě bude vizualizováno na obrázku 4.3, přičemž vysvětlení bude demonstrováno na zpracování obrázku. Důvodem je čtenářovo jednodušší uchopení problematiky na tomto problému. Dnes jsou konvoluční sítě především využívány u zpracování obrazu a mimo jiné zde mají taky svůj původ.

Představme si problém, kde chceme identifikovat, zda se na obrázku vyskytuje určité zvíře. Například koala. Vstupem do neuronové sítě v takovém případě je obrázek. Pro ulehčení s jedním kanálem, takže matice $n \times m$. Architekt neuronové sítě, pak musí definovat, jakým způsobem bude neuronová síť pracovat. Hlavním parametrem je velikost filtru, který bude aplikován na vstup. Na obrázku můžeme pozorovat modré pole o velikosti 7×3 . Filtr (*jádro*) je postupně posouván po obrázku, a tak generuje nové hodnoty, které tímto filtrem byly vypočteny. Myšlenkou tohoto filtru jsou detekce vzoru, které se v obrázku vyskytují. Pakliže půjdeme od základu, tímto způsobem jsme schopni naučit filtry detektovat hrany nebo okraje. U námi zmíněného zvířete tomu mohou být jednoduché tvary jako jsou oči. Hlouběji v neuronové sítí již dochází po zkombinování jednotlivých filtrů k detekci hlavy, těla až po detekci samotné koaly. Hodnota, která je generována po aplikaci filtru na část obrázku, je vypočtena pomocí skalárního součinu.



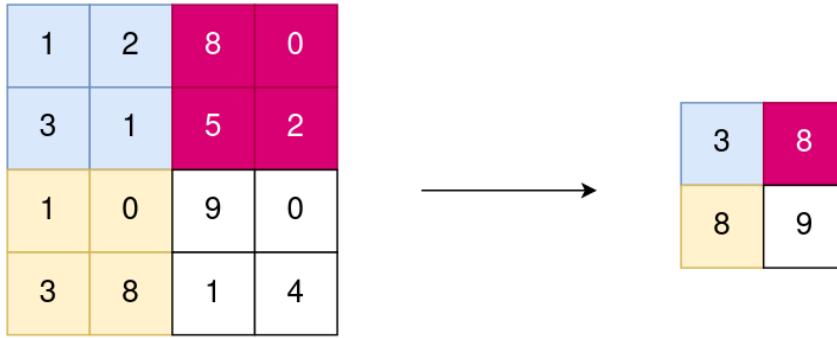
Obrázek 4.3: Konvoluční síť u zpracování mluveného slova [20]

Další důležitý mechanismus, který se vyskytuje u konvolučních sítí je pojem pooling. Postupným posunem filtru po obrázku dostaneme matici, nebo vektor o daných dimenzích. Často bývá zvykem, že na tento výstup je aplikován pooling, který se snaží extrahovat právě pouze ty části, které jsou pro

danou část nejrelevantnější. Pro lepší uchopení je zde uveden obrázek 4.4, na němž je vizualizován maximální pooling. Obecně existuje několik strategií, jak pooling provést a to například:

- **Maximální pooling** z definovaného okénka je vybrána maximální hodnota.
- **Součtový pooling** z definovaného okénka je vypočtena suma.
- **Průměrný pooling** z definovaného okénka je vypočten průměr.

Mimo extrakci důležitých vlastností, má pooling vlastnost redukce dimenze, a tak hlouběji v neuronových sítích dochází k rychlejšímu výpočtu, menším paměťovým nárokům.



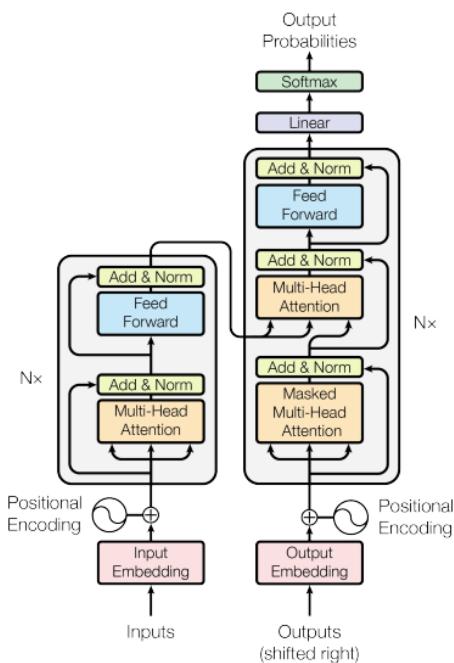
Obrázek 4.4: Vizualiacie mechanismu maximálního poolingu [21]

Zjistilo se, že přístup, který byl aplikován na obrázky, lze podobně aplikovat na číselnou reprezentaci textových dokumentů. Jak již víme, díky vnoření slov (kapitola 3.4 nebo 3.5), získáme pro každé slovo vektorovou reprezentaci o námi definované dimenzi (na obrázku *embedding dimension*). Každá z těchto dimenzí představuje určitou vlastnost slova, tudíž podobně jako u detekce části zvířete se můžeme pokusit nalézt podobné vzory mezi vlastnostmi přirozeného jazyka. Představou pak může být nalezení vazby mezi stylem psaní autora, blízký výskyt slov s pozitivním sentimentem.

Kapitola 5

Transformer

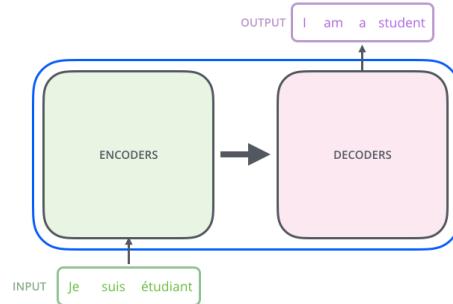
Transformer architektura (viz obrázek č. 5.1) neuronové sítě patří mezi state-of-art přístupy, které jsou aplikovány v doménách zpracování obrazu, a většiny problémů potýkajících se s zpracováním přirozeného jazyka. Zde také dosahuje většinou nadměrně skvělých výsledků. Architektura byla poprvé představena v článku *Attention Is All You Need* [22]. V této kapitole bude detailněji popsána Transformer architektura v několika dílčích částech. Po těchto částech by čtenář měl více rozumět myšlence, která stojí za tímto modelem. Slovní doprovod bude popisovat architekturu z pohledu zpracování přirozeného jazyka, ačkoliv principy fungování lze využít k řešení jiných záležitostí.



Obrázek 5.1: Transformer architektura [22]

5.1 Enkodér a dekodér bloky

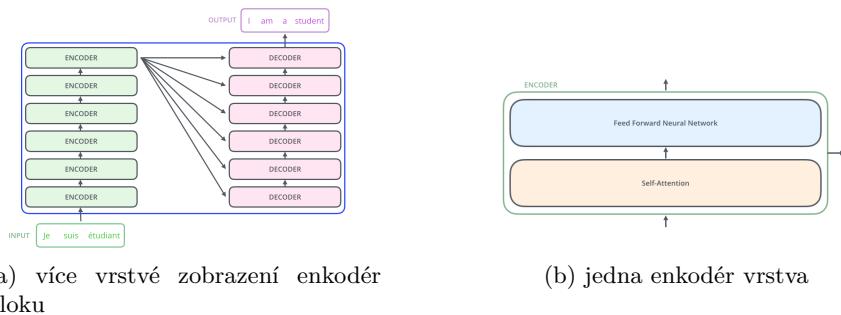
Architektura je založena na dvou stavebních kamenech enkodéru a dekodéru. Obecně enkodér zakóduje vstupní sekvenci do vektoru o více dimenzích a následně z tohoto vektoru část dekodéru dokáže rekonstruovat sekvenci původní. Příkladem zde může být strojový překlad, který vstupní sekvenci o (x_1, \dots, x_n) přemění na výstupní sekvenci (z_1, \dots, z_n) . Jednoduchá vizualizace na obrázku č. 5.2 může čtenářovi pomoci s představou.



Obrázek 5.2: Dekodér a enkodér blok transformer architektury [23]

5.2 Enkodér

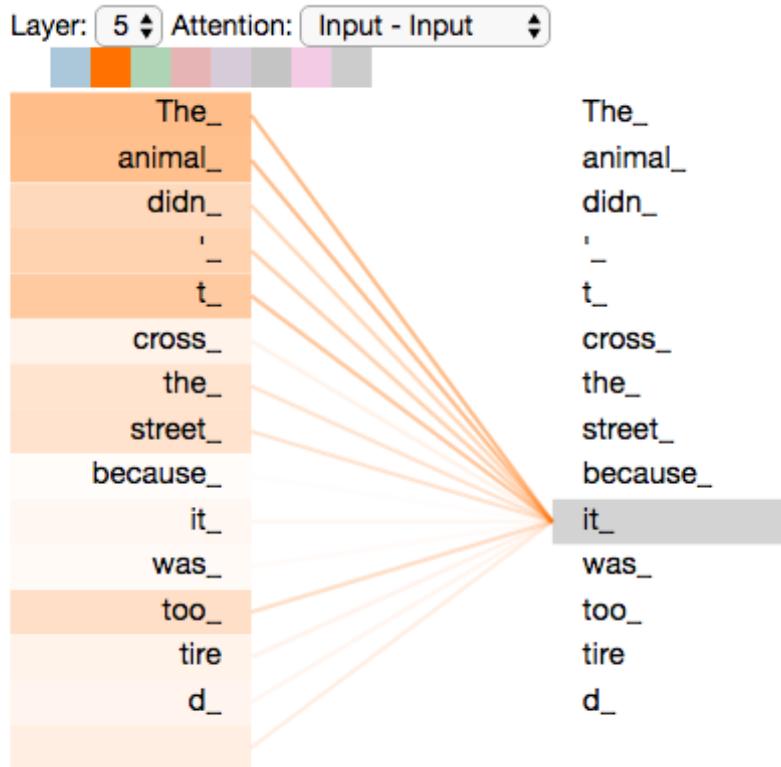
V článku [22] je blok enkodéru složen z více vrstev, které jsou za sebou zřetězené. Tento fakt je vizualizován na obrázku 5.3a. Na obrázku 5.3b, pravé straně, můžeme pozorovat vizualizace jedné samostatné vrstvy enkodéru. Vstupní signály tedy proudí prvně do **self-attention** vrstvy, která umožňuje odhalit vazby mezi signály (více v kapitole č. 5.3). Následně výstup z této vrstvy vstupuje do dopředné sítě (popsána v kapitole 4.2.1). Počet enkodér vrstev je hyperparametr, nad kterým lze provádět experimentální činnost.



Obrázek 5.3: Detailnější vizualizace enkodér bloku [23]

5.3 Self-Attention

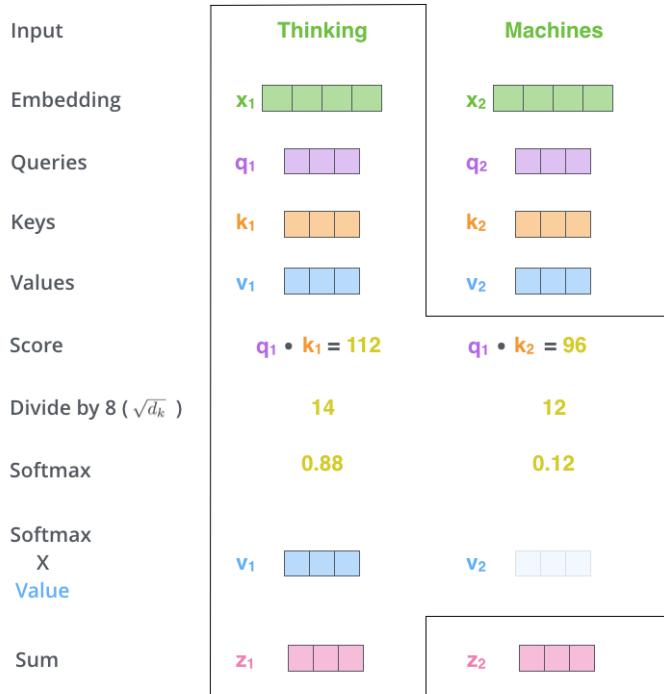
Koncept self-attention je jedním z hlavních bloků Transformer architektury. Vychází z myšlenky získání vektorové reprezentace daného vstupu s závislosti na všech ostatních vstupech. Představme si anglickou větu. *The animal did not cross the street because it was too tired* (zobrazenou na obrázku č. 5.4). Obrovskou nevýhodou předchozích *seq2seq* modelů bylo sekvenční zpracování věty. Výstup v čase x_i byl závislý na ostatních vstupech. Tyto modely měly dva hlavní nedostatky. A to právě sekvenční zpracování a neschopnost zachycení většího kontextového okna. Self-attention mechanismus přichází s jiným pohledem na věc. Řekněme, že chceme získat vektorovou reprezentaci slova *it*. Ta je závislá na všech ostatních vstupech a zároveň na sobě samém. Reprezentace slova bude získána způsobem, který je vizualizován na obrázku č. 5.4. Detailnější pohled s matematickým zaměřením bude popsán v kapitole 5.4.



Obrázek 5.4: Self-Attention mechanismus [23]

5.4 Výpočet v mechanismu Self-Attention

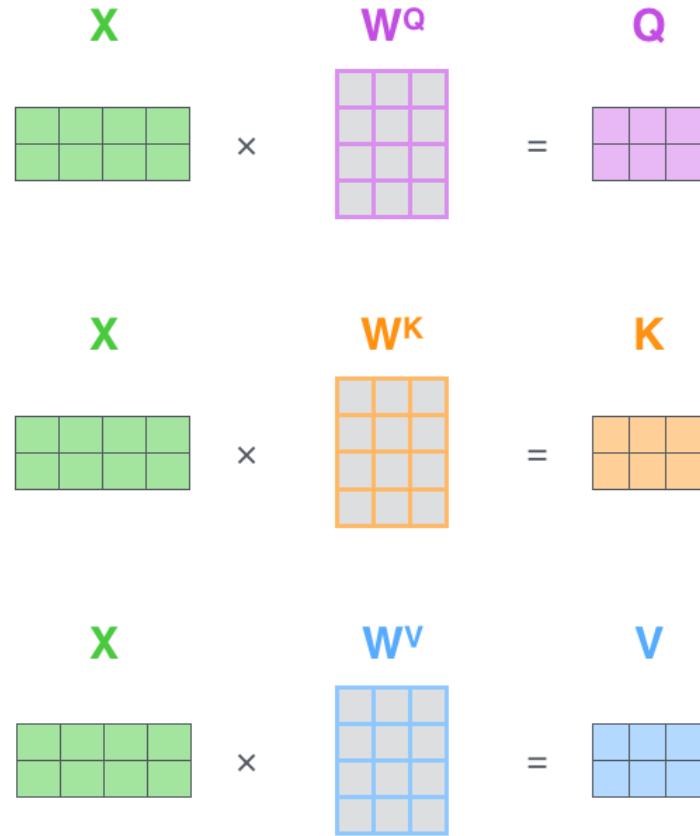
V předchozí kapitole č. 5.3 byl čtenáři popsán self-attention mechanismus obecně s úmyslem zatajení mnoha informací. Především šlo o matematický výpočet vektoru vzhledem k ostatním vstupům. V současné kapitole bude tento prohřešek napraven. Obrázek č. 5.5 popisuje ono fungování. Pro každý vstup existují 3 vektory, **query**, **key** a **value**. Abychom vypočetli aktuální signál daného vstupu, který bude propagován přes síť dále, tak musíme provést skalární součin mezi g_1 , **query** vektorem pro první vstup, a (k_1, \dots, k_n) , tedy všemi **key** vektory pro každý vstup aktuální sekvence. Tímto získáváme **skóre** reprezentující, jak velký vliv má vstup x_i na aktuálně zpracovávaný vstup. Zároveň si můžeme povšimnout, že zmíněné vektory mají jinou dimenzi, než vstupující vektor x_i . Hodnota dimenze je jedním z hyperparametrů tohoto modelu. Při kalkulaci je tato velikost znova využita při binární operaci dělení mezi skóre a odmocninou dané dimenze. Normalizována hodnota skóre je následně znova znormalizována pomocí funkce softmax. Tímto získáváme číselnou hodnotu v rozmezí od 0 do 1. Nutné si povšimnout, že součtem všech těchto hodnot získáváme 1 (*pravděpodobnost*). Tato **míra ovlivnění** se využije v matematické operaci násobení mezi **mírou** a **value** vektorem. Provedením všech těchto operací získáváme množinu (v_i, \dots, v_n) vektorů. Sečtením všech vektorů z množiny získáváme vektorovou reprezentaci daného vstupu x_i , která vstupuje do hluboce propojené dopředné sítě.



Obrázek 5.5: Detailní vizualizace kalkulace vektoru v Self-Attention mechanismu [23]

5.5 Optimální výpočet Self-Attention

Abychom zajistili optimální výpočet, všechny zmíněné operace z předchozí kapitoly 5.4 se provádí jako maticové operace. Opět bude využit popisný obrázek, který čtenáři udělá dobrou představu (obrázek č. 5.6).



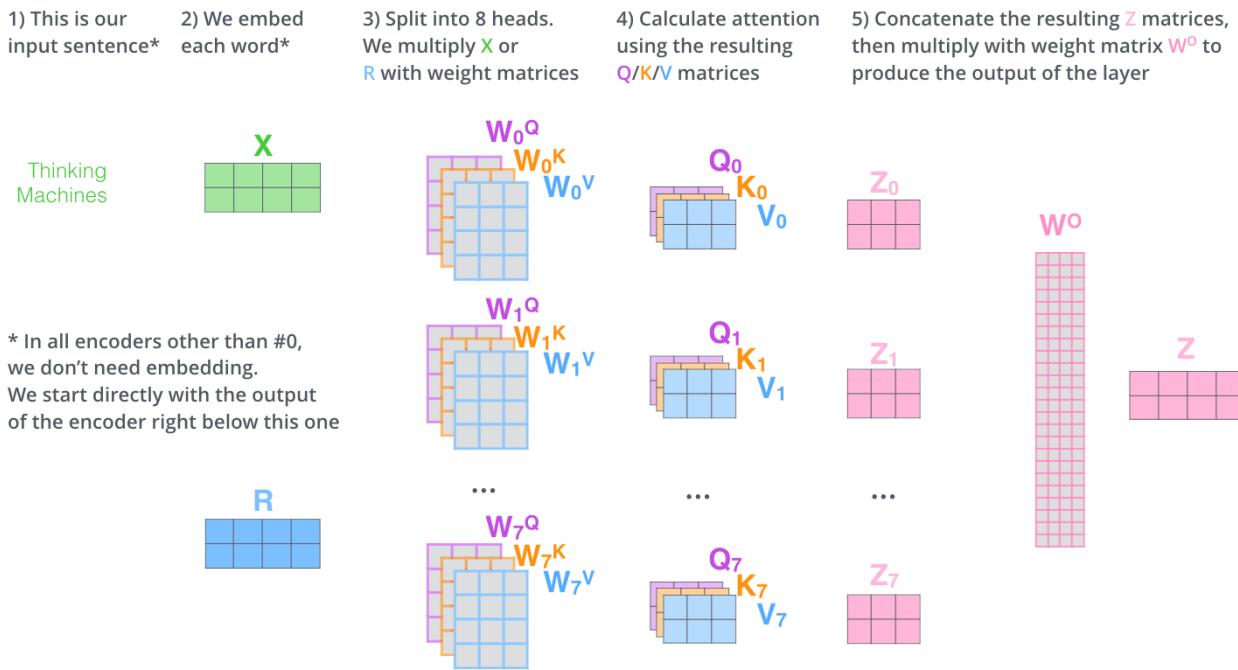
Obrázek 5.6: Maticový výpočet self-attention mechanismu [23]

Z obrázku je patrné, že architektura využije vnitřně uchované váhové matice W^Q , W^K , W^V , aby pro aktuální vstup získala zmíněné **query**, **key** a **value** vektory. Vypočtené matice, \mathbf{Q} , \mathbf{K} a \mathbf{V} reprezentují tedy všechny potřebné vstupy pro výpočet vektoru z_i (obrázek č. 5.5). Obdobně využijeme matematický výpočet popsaný následujícím vzorcem, *Scaled Dot-Product Attention* [22], kde výstupem je vektor z_i .

$$\text{Attention}(\mathbf{Q}, \mathbf{K}, \mathbf{V}) = \text{softmax}\left(\frac{\mathbf{Q}\mathbf{K}^T}{\sqrt{d_k}}\right)\mathbf{V}$$

5.6 Multihead-Attention

Článek zároveň [22] přinesl návrh vylepšení attention vrstvy. Jde o myšlenku využití multiheaded self-attention mechanismu. Tudíž pracujeme s několika váhovými maticemi $((W^Q), (W^K), (W^V))_1, \dots, (W^Q), (W^K), (W^V))_n$. Popis bude demonstrován obrázkem č. 5.7. Každá enkodér vrstva obsahuje i váhových matic, takže dochází k mechanismu self-attention i krát. U zpracování přirozeného jazyka to může značit například zachycení jiné části vlastnosti jazyka v každé hlavě. Stylistická, sémantická, syntaktická stránka. Z multihead-attention vrstvy je získán vektor z po provedení skalárního součinu mezi spojenými vektory (z_i, \dots, z_n) a váhovou maticí W^Q .



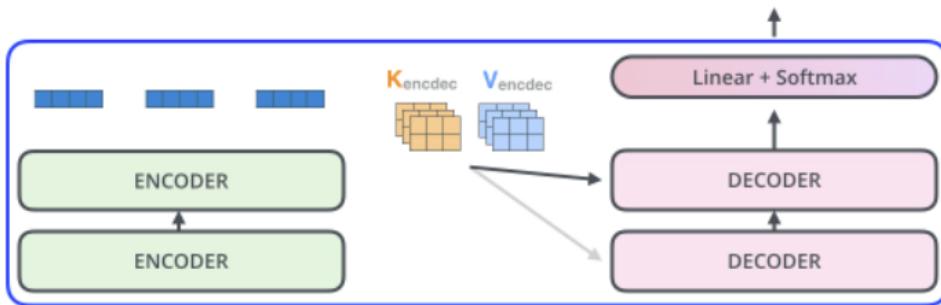
Obrázek 5.7: Multi-head attention [23]

5.7 Dopředná síť

Každá z vrstev enkodéru, tak i dekodéru obsahuje plně propojenou dopřednou neuronovou síť, kde jak už víme vstupem je vektor z_i o definované dimenzi. Výstupem je vektor o stejně dimenzi (hyperparametr popisující dimenzi bývá závislý na typu modelu).

5.8 Dekodér

Doposud byla vysvětlena strana enkodéru, ačkoliv jak je z obrázku č. 5.1 patrné, tak část dekodéru má většinu komponent stejnou jako část enkodéru. Doplňme akorát informace, jak vrstvy společně kooperují. K tomu využijeme vizualizaci č. 5.8.



Obrázek 5.8: Spolupráce enkodéru a dekodéru [23]

Výstupem enkodér bloku je množina attention vektorů \mathbf{K} a \mathbf{V} . Ty dekodér využívá při výpočtu "pozornosti" mezi vstupy dekodéru a výstupy enkodér bloku. U strojového překladu tímto přístupem jsme schopni generovat kvalitní překlad. Důležitým doplněním je zajištění, aby dekodér měl při zpracování vstupu v čase t_i přístup k výstupům v stejném čase z enkodéru. Budoucí hodnoty bývají většinou maskovány pomocí hodnoty $-inf$.

5.9 Lineární a softmax vrstva

Posledním nepopsaným blokem je lineární a softmax vrstava, která provádí jednoduchou projekci do pravděpodobnosti. Příkladem může být výběr následujícího slova ze slovníku o velikosti n . Největší pravděpodobnost ve výsledném vektoru najdeme právě na pozici x_i , kde se vyskytuje vhodné slovo.

Kapitola 6

Definice problému

Úkolem této práce je prozkoumání metod rozpoznání autorství u textových dat. Systém má mít schopnost se, ze vstupní množiny dokumentů, naučit rozpoznávat neznámé dokumenty, u kterých má s určitou jistotou určit autora.

Takto definovaný systém lze rozdělit dle problému do 3 kategorií:

- **Rozpoznání autora** (*autorství*). Kdy po trénování na množině dokumentů od vybraných autorů jsme schopni s určitou jistotou odhadnout autora neznámého díla.
- U **ověření autora** jsme schopni odpovědět na otázku s jakou pravděpodobností dané dílo bylo napsáno nebo nebylo zadáným autorem.
- Při **profilování autora** jsme schopni do určité míry modelovat vlastnosti textu, kterými se autor vyhraňuje.

6.1 Obecný popis řešení

Jelikož se ve všech případech jedná o typ problému řešitelný strojovým učením, pak obdobně jako u definice můžeme kategorizovat přístupy s částečnou korelací k definovanému problému:

- Řešení pomocí modelů založených na **strojovém učení** (*logistická regrese, náhodný les*). Učení vychází z techniky **učení s učitelem** (*supervised learning*).
- Řešení pomocí sofistikovanějších modelů založených na **hlubokých neuronových sítích** (*rekurentní, konvoluční*). Jako v předchozím případě se jedná o techniku **učení s učitelem**.
- **Učení bez učitele** (*unsupervised learning*), při kterém se snažíme najít nejfektivnější klastry s minimální chybou a určitými stylistickými vlastnostmi, které jde následně definovanými technikami extrahovat.

6.2 Popis vlastního řešení

Prvním krokem bylo vytvoření datové sady, nad kterou by šla provést experimentální činnost v podobě jednoho ze zmíněných problémů. V rámci diplomové práce bylo vytvořena datová sada prostřednictvím *Projekt Gutenberg* [24]. Jedná se o dobrovolnickou snahu digitalizovat kulturní díla. Detailnější tvorba bude popsána v kapitole č. 6.3.

Nad touto vytvořenou datovou sadou byly provedeny experimenty zmíněné v kapitolách 8, 9 a 10. Všechny tyto experimenty se snažily řešit problém **rozpoznání autora** (tj. *autorství*). Po zamýšlení je možné tento problém interpretovat jako **klasifikační** problém, se snahou námi definovaným modelem odhadnout autora textového řetězce, který model ještě neviděl vzhledem k autorům, nad kterými byl model trénován.

6.3 Datová sada

K řešení problému autorství bylo potřeba vytvořit datovou sadu, která obsahuje textová data společně s autorem těchto dat. Jak již bylo zmíněno v kapitole č. 6.2 k tomu byl využit *Projekt Gutenberg* [24]. Tato digitální knihovna obsahuje až 60000 děl v různých formátech (*epub, pdf, mobi, txt*).

6.3.1 Získání datové sady

U tvorby datové sady byl využit programovací jazyk R společně s knihovnou *gutenbergr* popsanou v kapitole 7.5. Ačkoliv knihovna obsahuje kolem 60000 digitalizovaných děl, ne všechna bylo možné využít. V rámci diplomové práce byla především věnována pozornost anglickým textům, jelikož většina modelů byla původně předtrénovaná na anglických textech.

Prvním krokem bylo provést explorační analýzu nad metadaty v podobě nalezení počtu anglických děl. Těch bylo nalezeno 42719. Ke každému záznamu existovalo několik proměnných. Například titulek, jazyk, boolean proměnná jestli ke knížce existuje text, autor a podobně (více obrázků č. 6.1). Předtím než byly moci být díla stažena došlo k filtrace dle:

title	author.x	language	gutenberg_bookshelf	rights	has_text	subject_type	subject	author.y	alias	birthdate
Amendments to the United States Constitution	United States	en	Politics/United States	Public domain in the USA. TRUE	loc	ICF	United States	N/A	N/A	
Major Pictures of Rotating Earth	United States	en	N/A	Public domain in the USA. TRUE	loc	World maps	United States	N/A	N/A	
Radar Map of the United States	United States	en	N/A	Public domain in the USA. TRUE	loc	G	United States	N/A	N/A	
The United States Constitution	United States	en	American Revolutionary War/Politics/United States/United S...	Public domain in the USA. TRUE	loc	ICF	United States	N/A	N/A	
The United States Bill of Rights The Ten Original Amendm... 1995 United States Congressional Address Book	United States	en	American Revolutionary War/Politics/United States/Law	Public domain in the USA. TRUE	loc	United States: Constitution, 1st-10th Amendments	United States	N/A	N/A	
The Papers And Writings Of Abraham Lincoln — Volume 7 ...	Lincoln, Abraham	en	N/A	Public domain in the USA. TRUE	loc	United States: Congress	United States	N/A	N/A	
Speeches & Letters of Abraham Lincoln, 1832-1865	Lincoln, Abraham	en	US Civil War	Public domain in the USA. TRUE	loc	Lincoln, Abraham, 1809-1865 -- Correspondence	Lincoln, Abraham	N/A	1809	
The Papers And Writings Of Abraham Lincoln — Volume 8 ...	Lincoln, Abraham	en	US Civil War	Public domain in the USA. TRUE	loc	E458	Lincoln, Abraham	N/A	1809	
The Lincoln Year Book: Axioms and Aphorisms from the Gre...	Lincoln, Abraham	en	N/A	Public domain in the USA. TRUE	loc	E458	Lincoln, Abraham	N/A	1809	
Abraham Lincoln's First Inaugural Address	Lincoln, Abraham	en	US Civil War	Public domain in the USA. TRUE	loc	E458	Lincoln, Abraham	N/A	1809	

Obrázek 6.1: Gutenberg v jazyce R

- Kniha musela být v anglickém jazyce.
- Ke knize musel existovat text.

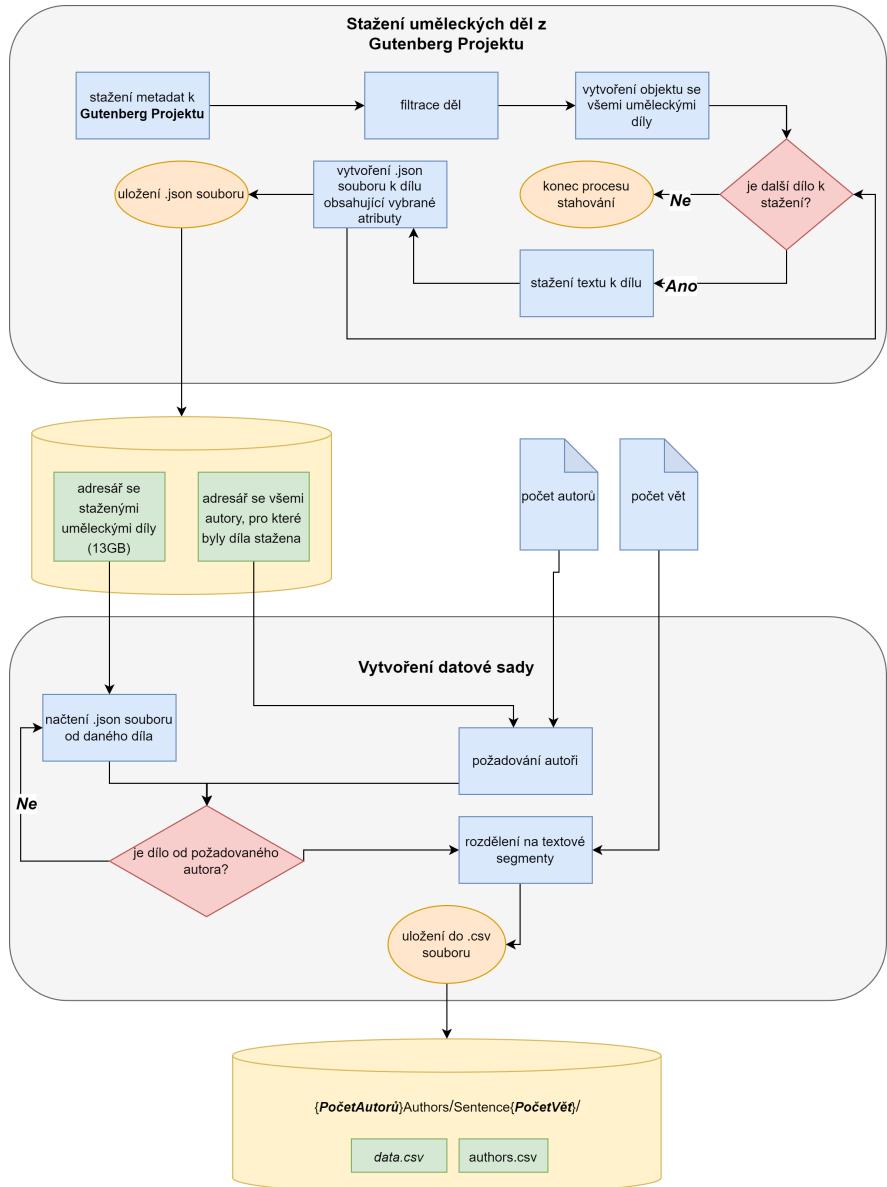
- Kniha musela mít **jasně** definovaného autora. Některé knihy totiž obsahovaly referenci v podobě identifikátoru na autora, ačkoliv tento autor byl neznámý nebo dílo spadalo do kategorie děl, které mají více autorů.

Po provedení této filtrace zůstalo **34413** děl, která mohla být stažena a využita pro experimenty.

6.3.2 Automatizovaná tvorba datové sady

Po odhalení validních anglických děl byly implementovány automatizované skripty, které prováděly proces popsaný v bodech níže. Fungování také může poohlít obrázek č. 6.2.

1. Stažení děl do specifikovaného adresáře. Do podadresáře *data* dle konfiguračního souboru. Finální adresář obsahoval 13 GB dat.
 - Každé dílo bylo staženo do samostatného *.json* souboru s vybranými atributy (více obrázek č. 6.3).
2. Uložení informací o všech autorech, pro které byla stažena díla do souboru *authors.csv*. Stejně jako u děl do specifikovaného adresáře, do podadresáře *authors*. Každý záznam autora obsahoval jméno autora, identifikátor autora a počet děl, která mohou být pro autora stažena.
3. Po specifikování počtu autorů a počtu vět dochází k načtení identifikátorů z *authors.csv* (autoři jsou seřazeni dle počtu možných děl) a následnému sekvenčnímu průchodu adresáře *data*.
 - Pokud autor aktuálně zpracovávaného *.json* souboru **nepatří** do aktuálně požadované množiny autorů, je soubor přeskočen.
 - Pokud autor aktuálně zpracovávaného *.json* souboru **patří** do aktuálně požadované množiny, je text díla rozsegmentován na textové segmenty o definovaném počtu vět a každý takový záznam uložen do patřičného *.csv* souboru.



Obrázek 6.2: Automatizovaná tvorba datové sady

Popsaným způsobem vzniklo několik *.csv* souborů popisující vždy unikátní datovou sadu. Záznamy datové sady lze pozorovat na obrázku č. 6.4. Můžeme vidět, že se jedná o záznam, kde v prvním sloupci je text o dané velkosti *k* vět a v sloupci dva *identifikátor* autora, ke kterému text patří. Takto je možné experimentovat v modelech s množstvím autorů a zároveň proměnlivým textovým vstupem.

JSON
Text
0 "December, 1776 [Elevit #1] The Project Gutenberg Elevit of The Declaration of Independence. All of the original Project Gutenberg Elevits from the 1970's were produced in ALL CAPS, no lower case. The computers we used then didn't have lower case at all. This is a retranscription of one of the first Project Gutenberg Elevits."
Title
0 "The Declaration of Independence of the United States of America"
Author
0 "Jefferson, Thomas"
Subject
0 "E201"
Alias
0 "United States President (1801-1809)"
Birthdate
0 1743
Deathdate
0 1826
Aliases
0: null
Id
0: 1
AuthorId
0: 1638

Obrázek 6.3: .json formát jednoho staženého díla

KLOOF A gap or pass in mountains. KRALA Native hut also a walled inclosure for cattle. KRAANZES Rocky precipices.	1285
MEERKAT A small fox-like mammal.	1285
RODINEK Literally, "red-neck" term applied to British soldiers by the Boers. SCHANSES Intrenchments (or fissures on hills).	1285
SAMBO A stick or whip made from elephant or rhinoceros hide. SPROUT A small stream. STOEP Veranda of a Dutch house.	1285
TAAL South African Dutch. TREK To move from place to place with belongings. VELD An open grassy plain.	1285
VELDSCHIJEN Rough untamed leather shoes. VERDOMDE Damned. VIERKLEUR The national flag (four colours) of the late South African Republics.	1285
VOORTREKKER Pioneer. VROUW Wife.	1285
The School Friends Nothing New By WHG Kingston Illustrations by E. Evans Published by George Routledge and Sons, London. The School Friends, by WHG Kingston.	8659
BOOK I - THE SCHOOL FRIENDS NOTHING NEW CHAPTER I. Lance Loughton and Emery Dulman were brought up together at Elmerston Grammar School. They were both in the upper or sixth form but Lance was nearly at the head, while Emery was at the bottom, of the form.	8659
They were general favourites, though for different causes. Lance was decidedly best liked by the masters. He was steady, persevering, and studious, besides being generous, kind-hearted, and brave—ever ready to defend the weak against the strong, while he would never allow a little boy t	8659
Emery had talents, but they were more showy than solid. He was good-natured and full of life and spirits, and having plenty of money, spent it freely. He was, however, easily led, and had in consequence done many foolish things, which got him into trouble, though he managed, on the whic	8659
Lance and Emery were on friendly terms and Lance, who thought he saw good qualities in his companion, would gladly have won his confidence, but Emery did not like what he called Lance's lectures, and there was very little or no interchange of thought between them. Without its real frien	8659
Emery was somewhat of a fine gentleman in his way. His father was a tradesman in the place, and wished his son to assist him in his business, but Emery often spoke of entering the army or one of the liberal professions. He therefore considered himself equal to those whose fathers held a h	8659
His father's style of life encouraged him in this. Mr Dulman had a handsome house, and gave dinners and parties and at elections took a leading part, and entertained the proposed member and his friends, and indeed sometimes talked of entering Parliament himself, and altogether did a go-	8659
Lance, however, very frequently refused Emery's pressing invitations. "I never met such a stay-at-home fellow as you are," exclaimed the latter, when on one occasion Lance had declined attending a gay party Mr and Mrs Dulman were about to give. "We shall have half the neighbourhood pr	8659

Obrázek 6.4: .csv soubor s vytvořenou datovou sadou

6.4 Explorační analýza nad vytvořenými soubory

Po vytvoření datových sad byla vždy provedena explorativní analýza s účelem získání širšího povědomí o nově vytvořené datové sadě. Z těchto vytěžených informací se mohlo postupně přejít k návrhu kvalitních metod na předzpracování textu (viz kapitola 6.5 na stránce 53), porovnání počtu záznamů mezi autory nebo nalezení správných hyperparametrů, které budou vstupem do neuronové sítě (*délka sekvence*). Útržek ze statistického výstupu si může čtenář prohlédnout na obrázku č. 6.5. Příklady pomocných tříd k explorativní analýze jsou:

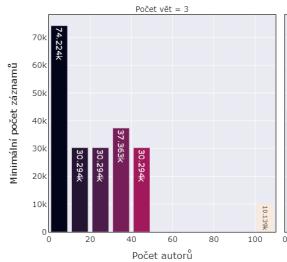
PreprocessingType	NormalizationValue	SubsetType	TransformerName	Authors	TextExample	LabelCounter	LabelTokenCounter	SentenceLength	TokenC
Raw	15000	all	Nada	{"AuthorId;Author": {"("761:Lytton", "Edward B..."}, "Tophet s..."}	I was all sick to my teeth as my poor topeth s...	{"0": {"1285": 15000, "520": 15000, "28": 1500...}}	{"Avg": {"1285": 51.0, "520": 24.0, "28": 66.0...}}	{"1285": 154.0, "28": 31...}	{"128": 8129, "28": 1135}
CaseInterpunction	15000	all	Nada	{"AuthorId;Author": {"("761:Lytton", "Edward B..."}, "Tophet s..."}	i was all sick to my teeth as my poor topeth s...	{"0": {"1285": 15000, "520": 15000, "28": 1500...}}	{"Avg": {"1285": 51.0, "520": 26.0, "28": 67.0...}}	{"1285": 147.0, "28": 30...}	{"128": 11214, "28": 1174}
Raw	NaN	all	Nada	{"AuthorId;Author": {"("761:Lytton", "Edward B..."}, "Tophet s..."}}	THE TRAGEDY OF PUDD'NHEAD WILSON by Mark Twai...	{"0": {"53": 73025, "520": 147048, "1865": 1865...}}	{"Avg": {"53": 27.0, "520": 629.0, "1865": 18...}}	{"53": 3477.0, "28": 9...}	{"53": "TRAC", "28": "7, "OF", "1865": "9..."}
CaseInterpunction	NaN	all	Nada	{"AuthorId;Author": {"("761:Lytton", "Edward B..."}, "Tophet s..."}}	the tragedy of pudd'nhead wilson by mark twain...	{"0": {"53": 73025, "520": 147048, "1865": 1865...}}	{"Avg": {"53": 28.0, "520": 645.0, "1865": 18...}}	{"53": 3444.0, "28": 8...}	{"53": "136.0, "28": 2, "1865": "8..."}

Obrázek 6.5: Statistický výstup

- Autoři, kteří se v datové sadě vyskytují (**Authors**).
- Náhodný záznam po provedení předzpracování (**TextExample**).

- Čítač záznamů pro daného autora ([LabelCounter](#)).
- Zaznamenání klouzavého průměru, maximálního a minimálního počtu tokenů vyskytujících se ve větách patřících k danému autorovi ([LabelTokenCounter](#)).
- Zaznamenání klouzavého průměru, maximální a minimální délky textového záznamu pro daného autora ([SentenceLength](#)).
- Zaznamenání čítače jednotlivých tokenů pro daného autora ([TokenCounter](#)).
- 50% a 75% kvantil počtu tokenů pro daného autora.

Náhodný záznam a čítač tokenů byly především využity k vytvoření různých způsobů přezpracovacích metod (kapitola 2 a 6.5). Díky čítače záznamů k danému autorovi byly zkonztruovány grafy (viz grafy na obrázku č. 6.6), a tak vybrány hodnoty, dle kterých bylo provedeno náhodné podvzorkování. Důvodem bylo především zrychlení výpočtu. Vedlejším produktem pak vyvážení datové sady a možnost využít metriky přesnosti (*accuracy*) (kapitola č. 6.6).



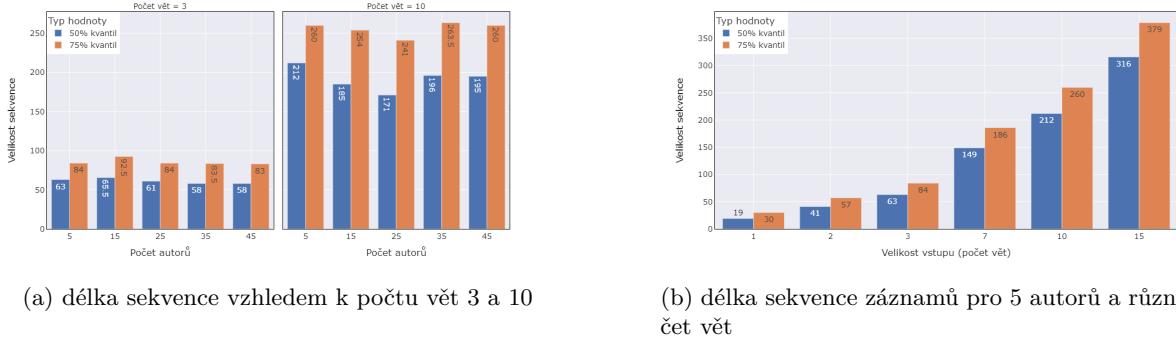
(a) počet autorů vzhledem k počtu vět 3 a 10



(b) počet záznamů pro 5 autorů a různý počet vět

Obrázek 6.6: Vizualizace počtu záznamů

Vstupujícím parametrem do neuronových sítí bývá někdy velikost vstupní sekvence. Textové záznamy dle této hodnoty bývají zkracovány (*truncation*) nebo doplňovaný (*padding*) o určitou hodnotu (θ , $[PAD]$). V statistických výpočtech jsem se snažil zaznamenat několik hodnot, z kterých následně byla vybrána výsledná hodnota vstupující do neuronové sítě. Grafy, které můžeme vidět na obrázku č. 6.7 s výběrem pomohly také.



Obrázek 6.7: Detailnější pohled na vybrané kvantily vzhledem k délce sekvencí

6.5 Předzpracování textové sady

Po provedené explorativní analýze byly zkonstruovány metody na předzpracování. S těmito metodami bylo experimentováno:

- **PreprocessingDefault** - v kterém dochází k odstranění vybraných úseků, transformaci do malých písmen, odstranění interpunkce, normalizaci všech bílých znaků, odstranění numerickej znaků, odstranění stopslov, odstranění slov kratších jak délka 3 a lemmatizace textu.
- **PreprocessingLowercase** - dochází pouze k transformaci do malých písmen.
- **PreprocessingRaw** - text zůstavá v původní formě.
- **PreprocessingCaseInterpunction** - odstranění vybraných úseků, transformace do malých písmen, odstranění interpunkce, odstranění přebytečných bílých znaků a odstranění numerickej znaků.

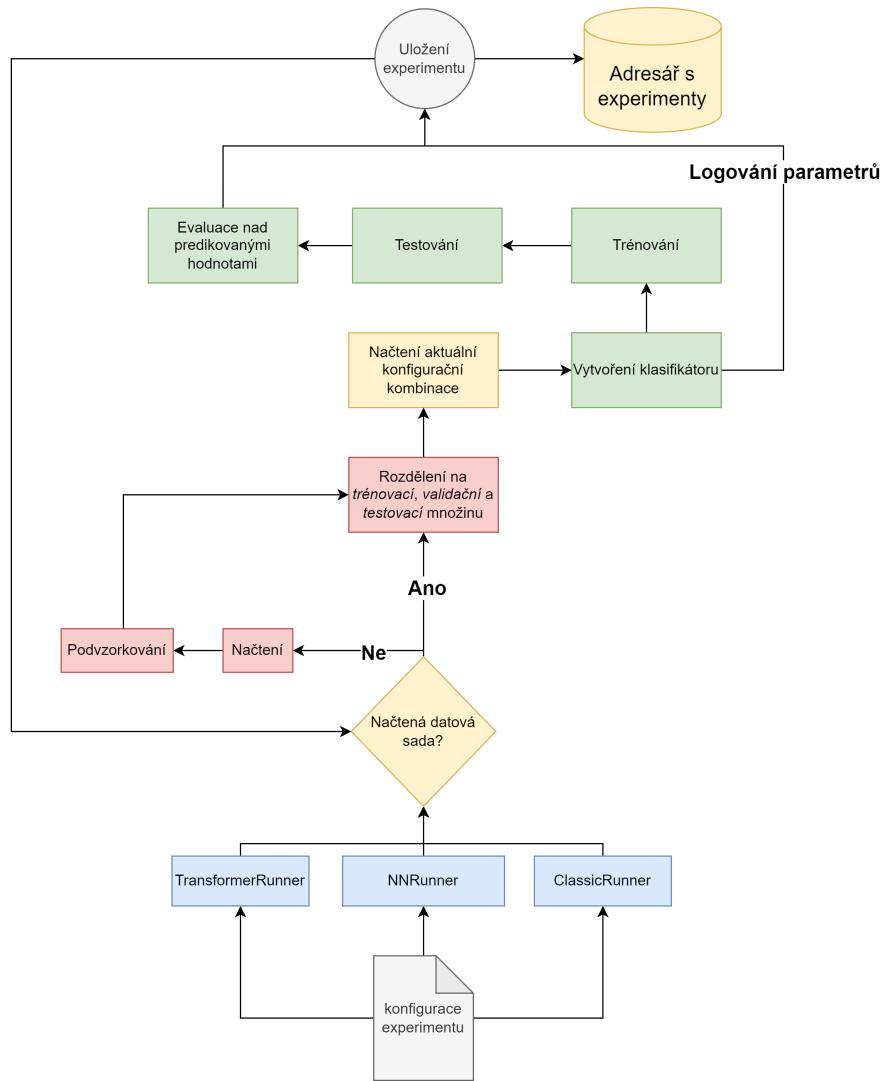
6.6 Kvalita modelů

Výsledek klasifikace pro jednotlivé modely byl ověřen pomocí metriky přesnosti (*anglicky accuracy*). Ta se vypočte pomocí formule $\frac{(TP+TN)}{(TP+FP+FN+TN)}$, kde v čitateli je mohutnost úspěšně predikovaných a ve jmenovateli mohutnost všech záznamů. Důvodem, proč daná metrika mohla být v experimentech

využita bylo dodržení vyváženosti vzhledem ke všem třídám. Více informací, proč datové sady byly vyváženy, nalezneme v následující kapitole 6.7.

6.7 Popis spouštění experimentů

Úkolem bylo především provést porovnání mezi jednotlivými přístupy. Kvůli možnosti provedení většího množství experimentů s následným vyhodnocením byla zprovozněna automatizace skriptů. Vývojový diagram na obrázku č. 6.8 tento proces vizualizuje.



Obrázek 6.8: Automatizované spouštění experimentů

Prvně dochází k definování konfigurací pro dané experimenty. Zde musí uživatel specifikovat počet autorů, velikost sekvence, typ předzpracování a normalizační hodnotu na podvzorkování.

Současně hyperparametry pro klasifikátory, které mají být použity. Po definici dochází k spouštění skriptů, v kterých dle typu experimentu dojde k podobnému procesu:

Přes konfigurační objekt se načte požadovaná datová sada. Datová sada se rozdělí na 3 potřebné množiny. Trénovací, testovací a validační (velikosti jdou definovat v konfiguračních proměnných projektu). Po načtení a rozdelení datové sady se vytvoří všechny možné kombinace ze specifikovaných nastavení klasifikátorů. Postupně pro všechny konfigurace jsou spouštěny experimenty. Prvně dojde k vektorizaci textových dat, natrénovaní klasifikátoru. Natrénovaný klasifikátor následně provede predikci na testovací množině. V průběhu výpočtu dochází k logování všech možných hodnot (viz kapitola č. 6.7.1).

6.7.1 Logování výsledků

Abychom měli možnost porovnávat klasifikátory mezi sebou, tak bylo vytvořeno rozhraní, kterým po každém ukončeném experimentu bylo uloženo vše do několika *.csv* souborů.

- **Shrnutí experimentu** - zaznamenaní důležitých časů běhu predikce, učení, vektorizace a mimo jiné uložení počtů jednotlivých záznamů v trénovací, validační a testovací množině.
- **Deskripce experimentu** - počet autorů, počet vět, typ klasifikace, délka sekvence, normalizační hodnota na podvzorkování, nastavení neuronové sítě.
- **Log hodnot neuronových sítí** - *log.csv* k dispozici pro neuronové sítě, kde se vyskytovaly hodnoty po každé proběhlé iteraci (přesnost na trénovací množině, přesnost na validační množině, chyba na trénovací množině, chyba na validační množině).

Kapitola 7

Vývojové prostředí

Práce byla vytvořena v programovacím jazyce Python a R společně s využitím mnoha užitečných knihoven pro vizualizaci dat, načtení a uchování dat v paměti, knihoven poskytujících klasifikační modely, knihoven pro vytváření modelů neuronových sítí nebo knihoven poskytující metody pro práci s nestrukturovaným formátem textových dat. Bylo vybráno pár knihoven, které byly detailněji rozebrány v následujících podkapitolách.

7.1 Natural Language Toolkit

NLTK (*Natural Language Toolkit*) [25] je knihovna poskytující rozhraní k práci s textovými daty. Mezi toto rozhraní patří například proces tokenizace, stematizace, lemmatizace (detailněji popsané v kapitole 2).

7.2 Keras a TensorFlow

TensorFlow [26] je veřejně dostupná knihovna, která uživateli usnadní vytváření modelů pro strojové učení. Společně s jednoduchým uživatelským rozhraním, TensorFlow poskytuje flexibilní ekosystém, užitečné nástroje, společně s možností jednoduše načíst některé z často používaných datových sad.

Keras [27] je uživatelské rozhraní, které poskytuje abstrakci nad vytvářením komplikovaných matematických modelů neuronových sítí. Je zaměřeno na rychlé vytváření modelů společně s možností využití v produkci. Zároveň od nové verze TensorFlow 2 je Keras součástí knihovny TensorFlow.

7.3 Gensim

Gensim [28] je veřejně dostupná knihovna, která poskytuje algoritmy a další nástroje pro zpracování přirozeného jazyka. Příkladem mohou být algoritmy pro vytvoření vnoření slov jako Word2Vec (popsaná detailněji v kapitole 3.4). Mezi zmíněné nástroje patří například algoritmy pro provedení stematizace, obecně předzpracování textu.

7.4 HuggingFace

HuggingFace [11] je veřejně dostupná knihovna, která vznikla z myšlenky vytvořit chatovacího bota v roce 2016. Postupem času z nápadu vytvoření bota vznikla dnes jedna z nejpopulárnějších knihoven nabízející předučené state-of-art modely především pro problémy strojového zpracování jazyka a mimo to zpracování obrazu.

7.5 gutenbergr

Gutenbergr [29] je veřejně dostupný balíček k práci s kolekcí dokumentů z *Projekt Gutenberg*. Balíček obsahuje metody k výpočtu statistických atributů společně s nástroji pro jednoduché stáhnutí knih.

7.6 Výpočetní prostředí

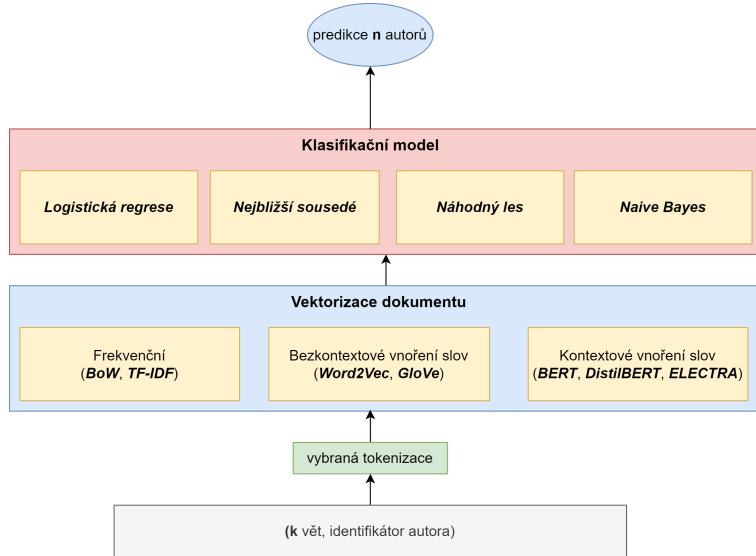
Pro praktickou činnost byly využity zdroje VŠB-TUO (*Vysoká škola báňská - Technická univerzita Ostrava*). Byly využity servery, kde se nachází hardwarové vybavení 40 CPU a 4 GPU (GeForce GTX 1070).

Kvůli nekompatibilních verzích knihovny tensorflow a CUDA (*anglicky Compute Unified Device Architecture*) knihoven, běžel výpočet pouze na CPU. Toto omezení částečně limitovalo v provedení experimentů, ačkoliv i přesto bylo provedeno dostatečné množství experimentů pro zpracování diplomové práce.

Kapitola 8

Experiment 1 - vektorizace textu a klasické modely

První experiment pracuje s přístupem vektorizace vstupních textových dat pomocí více typů instancí a následným využitím klasických klasifikátorů s různým nastavením. Obrázek popisující experiment lze najít na obrázku č. 8.1.



Obrázek 8.1: Architektura klasických modelů

8.1 Popis řešení

Čtenářovi budou představeny přesněji způsoby vektorizace, které byly využity:

- **Bag of words** vektorizace.
- **TF-IDF** vektorizace.
- Vektorizace s využitím předučených vektorů z **Word2Vec** modelu. Vektory byly předučeny nad daty z Google news. Parametrem pro dimenzi byla hodnota 300.
- Vektorizace s využitím předučených vektorů z **GloVe** modelu. Vektory byly předučeny nad kombinaci dat z Wikipedie 2014 a Gigaword 5. Parametrem, jak v předchozím případě, byla hodnota 300.
- Extrakce vektorové reprezentace z derivátu BERT modelu (**ELECTRA**, **DitilBERT**, **BERT**).

V rámci vnoření slov byla vektorová reprezentace získána průměrem ze všech tokenů dokumentu. Po provedení této vektorizace byl dokument reprezentován číselným vektorem o velikosti 300.

Pro BERT model existuje více strategií, jak extrahovat vektorovou reprezentaci. Více informací o extrakci s výsledky experimentů zaměřených na různý způsob extrakce bude prezentováno v kapitole č. 10. V tomto experimentu šlo o extrakci reprezentace dokumentu z */CLS/* tokenu. Respektive doporučovaný přístup z článku [10], kde byl BERT model představen.

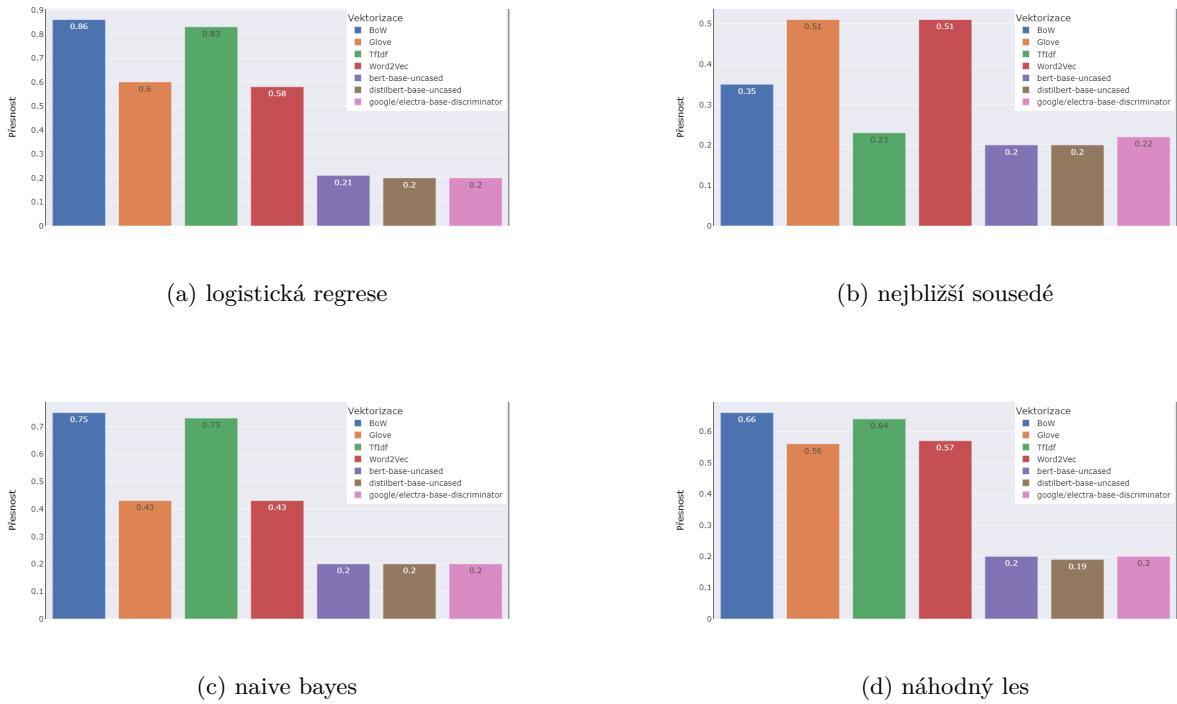
Modelů pro klasifikaci bylo vybráno pár takových, které jsou použitelné na úkol textové klasifikace. Použité metody byly detailněji teoretičky popsány v kapitole 4.1. Šlo o **naive bayes**, **logistickou regresi**, **nejbližší sousedy**, **náhodný les**.

8.2 Výsledky

Dohromady bylo provedeno 203 experimentů zaměřující se na všechny možné kombinace mezi typy vektorizace a klasifikátorů. Některé z experimentů byly provedeny vícekrát. Do analýzy byly zahrnutý experimenty pro danou kombinaci s nejvyšší přesností. Vizualizace výsledků je zaznamenána v grafech na obrázku č. 8.2.

Po analýze byla vybrána logistická regrese jako nejkvalitnější klasický model. Tento model především skvěle pracoval s jednoduchou BoW a TF-IDF vektorizaci, kde bylo dosaženo výsledků přes 83 %. Sestupné seřazení dle kvality klasifikátoru je následující: logistická regrese, naive bayes, náhodný les, nejbližší sousedé. Pro všechny modely, až na nejbližší sousedy platilo pravidlo nejkompatibilnější vektorizace s BoW a TF-IDF.

Výjimkou byl model nejbližších sousedů, kde Word2Vec a GloVe vektorizace překonala BoW a TF-IDF až o 30 %.



Obrázek 8.2: Grafová vizualizace klasických modelů s různými typy vektorizace

8.2.1 Časová náročnost

Mimo měření úspěšnosti modelů byl zároveň měřen čas potřebný k vykonání vektorizace, učení modelu a následné predikce. Zaznamenané časové nároky jednotlivých běhů jsou zobrazeny v tabulce č. 8.1. Můžeme pozorovat potvrzení výkonnosti BoW a TF-IDF vektorizace a klasifikátoru logistické regrese. Po provedení normalizace BoW lze zredukovat čas výpočtu, ačkoliv musíme přepokládat větší časové náklady na vektorizaci u TF-IDF, jelikož TF-IDF provádí více výpočtu než BoW.

Bezkontextové vnoření slov dosahovalo rychlých výpočtů oproti ostatním vektorizačním technikám. Ačkoliv vzhledem k přesnosti nedává smysl tento typ vektorizace využít.

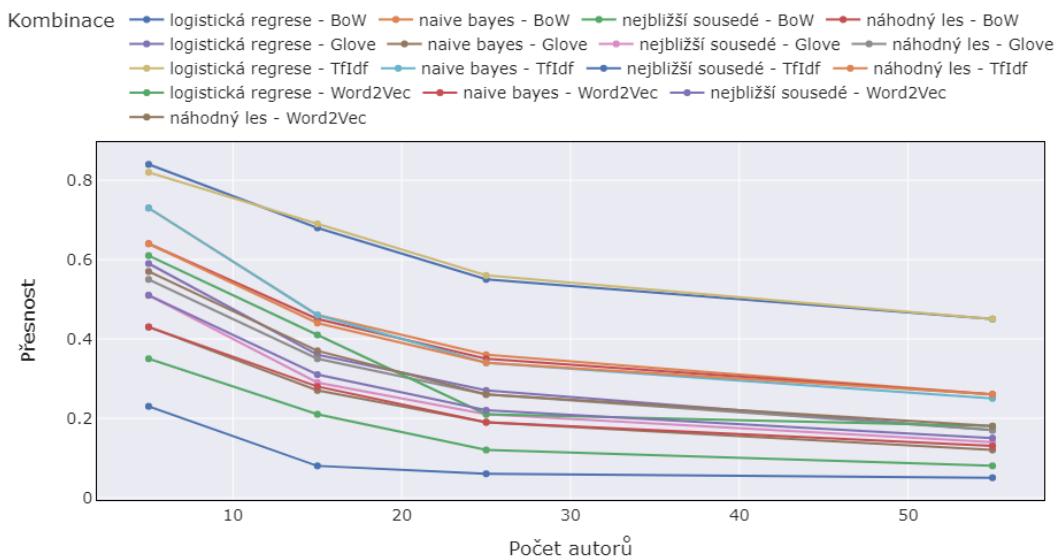
Kontextové vnoření slov a derivace BERT modelu měli podobné výpočetní nároky jako u frekvenčních vektorizačních technik.

Tabulka 8.1: Časové nároky na vektorizaci v kombinaci s klasickými modely

Vektorizace	log. regrese		naive bayes		nej. sousedé		náh. les	
	tré.	test.	tré.	test.	tré.	test.	tré.	test.
<i>BoW</i>	121s	2s	20s	10s	1s	154s	340s	6s
<i>TF-IDF</i>	92s	1s	13s	6s	2s	148s	313s	3s
<i>GloVe</i>	5s	1s	1s	1s	1s	21s	153s	1s
<i>Word2Vec</i>	9s	1s	1s	1s	1s	31s	179s	1s
<i>ELECTRA</i>	112s	1s	1s	1s	1s	119s	290s	1s
<i>DistilBERT</i>	108s	1s	1s	1s	1s	111s	290s	1s
<i>BERT</i>	113s	1s	1s	1s	1s	108s	291s	1s

8.2.2 Různé kombinace vstupu

Z předchozích provedených analýz bylo zjištěno, že transformer vektorizace není prospěšné kombinovat s klasickými modely. Proto v této analýze byly již transformery opomenuty. Obrázek č. 8.3 srovnává bezkontextové vnoření slov a klasické frekvenční vektorizace ve spojení s klasickými modely. Především si lze povšimnout, že logistická regrese kombinovaná s BoW a TF-IDF, na tomto typu úlohy, dosahovala mnohem vyšších výsledků než ostatní kombinace (přes 80 % na 5 autorech). Experimenty proběhly nad velikosti věty 3 a počtem autorů (5, 15, 25, 55). U většiny zmíněných modelů byl zaznamenán rozdíl mezi 55 a 5 autory kolem 30 % přesnosti.

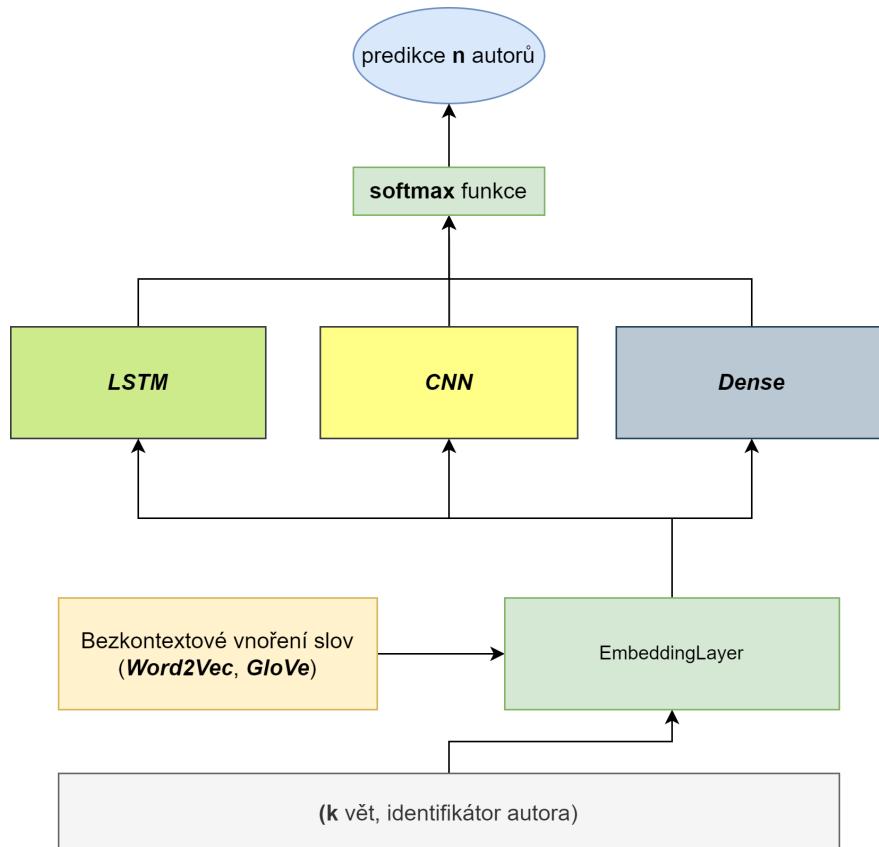


Obrázek 8.3: Různý počet autorů ve spojení s klasickými modely

Kapitola 9

Experiment 2 - neuronové sítě

Druhý blok experimentů testoval různé typy neuronových sítí ve spojení s vrstvou `EmbeddingLayer` z knihovny TensorFlow (sekce 7.2). Obrázek č. 9.1 vizualizuje způsob experimentu.



Obrázek 9.1: Diagram popisující druhý experimentální blok

9.1 Popis řešení

Z obrázku je tedy patrné, že byly využity 3 typy architektur neuronových sítí **konvoluční síť** (*CNN*), **rekurentní síť** (*LSTM*) a **jednoduché dopředné síť** (*Dense*). Jak již bylo zmíněno neuronové sítě nepracují s textovými vstupy, nýbrž s číselnými vektorovými vstupy, které byly získány pomocí [EmbeddingLayer](#) objektu importovaného z knihovny TensorFlow. Tato vstva umožnuje inicializaci pomocí již předučených vektorů. Dle způsobu inicializace byly využity 3 přístupy:

- [EmbeddingLayer](#) s **prázdnou** inicializační hodnotou, kde dochází k učení slovní reprezentace od základu. Vektory byly reprezentovány dynamicky vybraným dimenzionálním prostorem n .
- Inicializce pomocí vektorů, které byly naučeny algoritmem **Word2Vec**. Vektory byly modelovány dimenzí 300.
- Inicializce pomocí vektorů, které byly naučeny algoritmem **GloVe**. Vektory byly modelovány dimenzí 300.

9.2 Výsledky

Experimenty byly provedeny nad datovou sadou s 5 autory a velikosti věty 3. U neuronových sítích dochází k povinnosti definice velikosti slovníku, s kterou kódovací vrstva bude pracovat. Nenalezené slova budou transformovány do *[UNK]* tokenu. Tato velikost byla definována hodnotou 60000 a velikost věty hodnotou 100. Finální výsledky včetně zaznamenání doby učení, velikosti dimenze slovníku a počtu nenalezených vektorů zaznamenává tabulka č. 9.1.

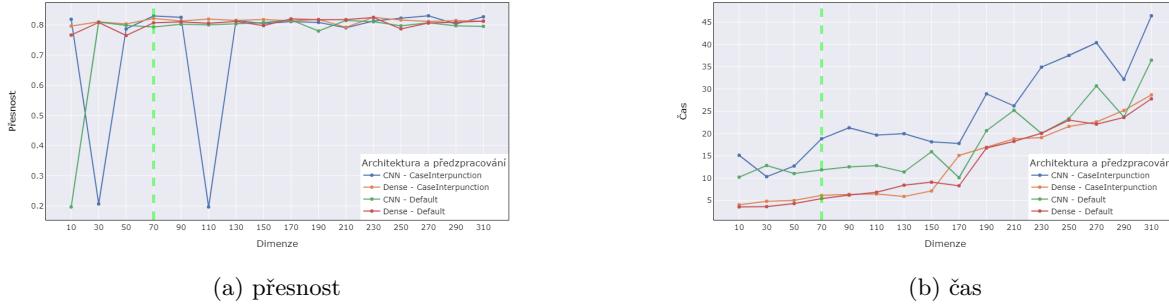
Tabulka 9.1: Výsledky experimentu s větným oknem

typ architektury	inicializace	dimenze	nенalezených vek.	přesnost (%)	čas (min)
<i>CNN</i>	-	50	60000	81,75	8
<i>CNN</i>	-	150	60000	80,90	14
<i>CNN</i>	-	200	60000	81,28	20
<i>CNN</i>	Word2Vec	300	10576	81,07	26
<i>CNN</i>	GloVe	300	15657	82,65	36
<i>Dense</i>	-	50	60000	82,36	6
<i>Dense</i>	-	150	60000	79,54	9
<i>Dense</i>	-	200	60000	81,79	16
<i>Dense</i>	Word2Vec	300	10576	72,50	30
<i>Dense</i>	GloVe	300	15657	79,61	23
<i>LSTM</i>	-	50	60000	19,67	51
<i>LSTM</i>	-	150	60000	19,67	57
<i>LSTM</i>	-	200	60000	19,67	58
<i>LSTM</i>	Word2Vec	300	10576	19,67	58
<i>LSTM</i>	GloVe	300	15657	19,67	63

U GloVe vektorizace nebylo nalezeno 15657 slov, přičemž Word2Vec byl úspěšnější a nebylo nalezeno jen 10576. Hodnotu by bylo možné optimalizovat po ladění procesu předzpracování, ačkoliv k tomu v experimentech nedocházelo. Konvoluční síť dosáhla nejvyššího výsledku ve spojení s GloVe vektorizací, a to 82,65 %. Druhý nejvyšší výsledek měla hluboce propojená dopředná síť s výsledkem přesnosti 82,36 %. Tato architektura dosahovala nejlepších výsledků v kombinaci s ne-inicializovanou kódovací vrstvou ([EmbeddingLayer](#)). Rekurentní síť měla tendenci v experimentech stagnovat a nebylo blíže zjištěno, proč k tomuto jevu docházelo. Vzhledem k modelování dimenzionální prostoru pomocí kódovací vrstvy je možné pozorovat, že dostačuje dimenze 50. Detailnější zkoumání je popsáno v kapitole 9.2.1.

9.2.1 Modelování potřebné velikosti vektoru

Vzhledem k předchozímu experimentu byl navržen další, který byl zaměřen na modelování velikosti mnohodimenzionálního prostoru potřebného k zachycení vlastností, dle kterých neuronová síť dokáže identifikovat autora. Konfigurace sítě zůstaly stejné jako v předchozím experimentu, akorát byla opomenuta LSTM architektura. Byly testovány hodnoty od 10 do 320 s velikostí kroku 20. Následující obrázek znázorňuje výsledky nad testovací sadou.



Obrázek 9.2: Modelování velikosti mnohodimenzionálního prostoru pro reprezentaci slov

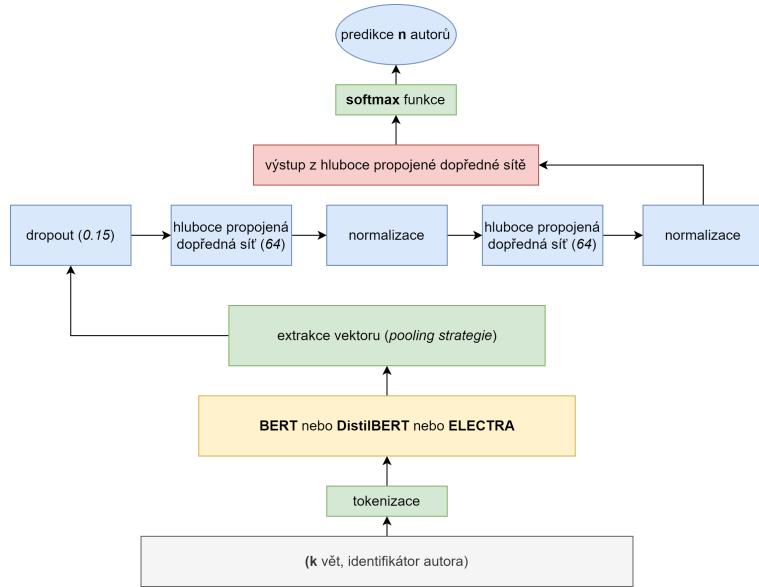
Výsledky na obrázku č. 9.2 znázorňují výsledky přesnosti vzhledem k testovací množině a dobu potřebnou k provedení celého procesu naučení modelu na trénovací množině, validovaní nad validační množinou, následné získání predikcí nad testovací množinou a vyhodnocení těchto výsledků. Byly provedeny experimenty nad dvěma architekturami CNN (*konvoluční síť*) a Dense (*hluboce propojená dopředná síť*). Zároveň byly využity dva přístupy k předzpracování **CaseInterpunction** a **Default** (viz kapitola o předzpracování 6.5). Vzhledem k proměnným přesnosti (obrázek 9.2a vlevo) a času (obrázek 9.2b vpravo) byla vybrána hodnota 70 (světle zeleně přerušovaná čára na obrázku č. 9.2), která vystačí neuronovým sítím k schopnosti následně predikovat autora textu s vysokou přesnosti. Se zvyšující hodnotou pro mnohodimenzionální reprezentaci tokenů od hodnoty 70, již nedocházelo k mimořádnému zvýšení přesnosti. Naopak docházelo pouze k zbytečnému zvýšení výpočetních nároků (komplexnosti) na neuronovou síť, jelikož byla nucena pracovat s vícedimenzionálními vektory, naučit se většího množství vah, což také vyžadovalo větší nároky na čas.

Kapitola 10

Experiment 3 - BERT

Tento experiment pracoval s hlubokými neuronovými sítěmi, které vycházely z architektury Transformeru. Šlo o hlavní zaměření diplomové práce, kde bylo účelem porovnat kvalitu BERT modelu nad klasifikačním problémem.

Základní BERT model obsahuje velké množství parametrů, které je potřeba naučit, abychom měli k dispozici jazykový model. Díky něj, pak můžeme řešit široké spektrum různých úloh. Jelikož naučení takového modelu bývá velice drahé na výpočet, přistupuje se k technice přeneseného učení (*transfer learning*). Ta vychází z myšlenky využití modelu, který má v sobě naučený jazykový model. Tento model je na naší úloze doladěn, a tak máme možnost využít state-of-art přístupy i přes ”malou” výpočetní námahu.



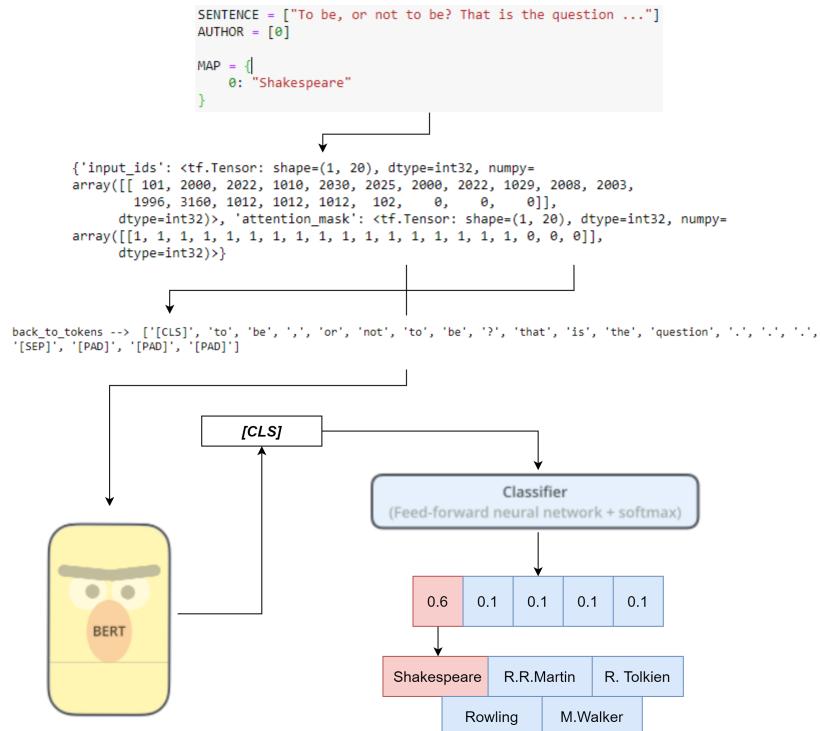
Obrázek 10.1: Architektura BERT modelu

10.1 Popis řešení

V experimentech bylo využito přenesené učení (*transfer learning*) s vyzkoušením 3 typů derivátu z BERT modelu. Půjde o původní BERT model, rychlejší destilovaný DistilBERT a jeden z nejnovějších modelů ELECTRA.

BERT model nám umožňuje získat vektorovou reprezentaci vět, tokenů, obecně textových segmentů. Tyto segmenty jsou následně použity v hluboce propojené dopředné neuronové síti k získání předpovědi autora. Není tedy potřeba nechat modelu tolík času k učení, jelikož vychází z určitého stavu porozumění jazyku. Váhy bývají v průběhu trénování pouze doupraveny, tak aby byly schopny produkovat vektorovou reprezentaci modelující styl psaní daného autora. Obecně pak nemusí jít pouze o zachycení stylu psaní autora, ale například zachycení polarity textu, schopnost rozeznání spamu a tak dále. Na obrázku č. 10.1 je zachycen diagram popisující využití BERT modelu.

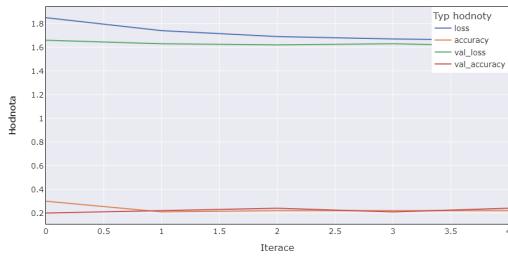
V prvním kroku šlo o vytvoření adekvátního vstupu do BERT modelu. Respektive procesu tokenizace, kde textová složka záznamu autora byla transformována do slovníkového tvaru (více na obrázku č. 10.2) dle definice zmíněné v kapitole 3.6.4. Takto předzpracovaný vstup byl díky enkodér vrstvám transformován do vektorové číselné reprezentace o dimenzi n . Z BERT modelu byl vybrán vektor na 0 pozici, $[CLS]$ token, určený ke klasifikaci (více způsobu výběru rozebráno v kapitole č. 10.2.2) a ten vstupoval do hluboké neuronové sítě, kde výstup po aplikaci softmax funkce představoval s jakou pravděpodobností napsal autor X_i záznam i .



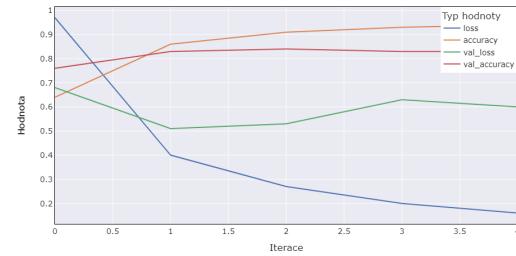
Obrázek 10.2: Detailní průběh záznamů při klasifikaci

10.2 Výsledky

Po předtrénovaní je v modelu uložen jazykový model, který je schopen rozumět vlastnostem reálného světa. V jednom z prvních testovaných schopností předučeného modelu byla kvalita jazykového porozumění. Na obrázku č. 10.3 je zaznamenán běh modelu v dvou konfiguracích. V jedné z nich byl model doladěn na aktuální problém a v druhém mu byla dovolena aktualizace vah jen v dopředné vrstvě. Model bez fine-tuning procesu celého modelu nelze využít, jelikož není schopen porozumět v našem případě vlastnostem autora.



(a) trénování pouze dopředné vrstvy

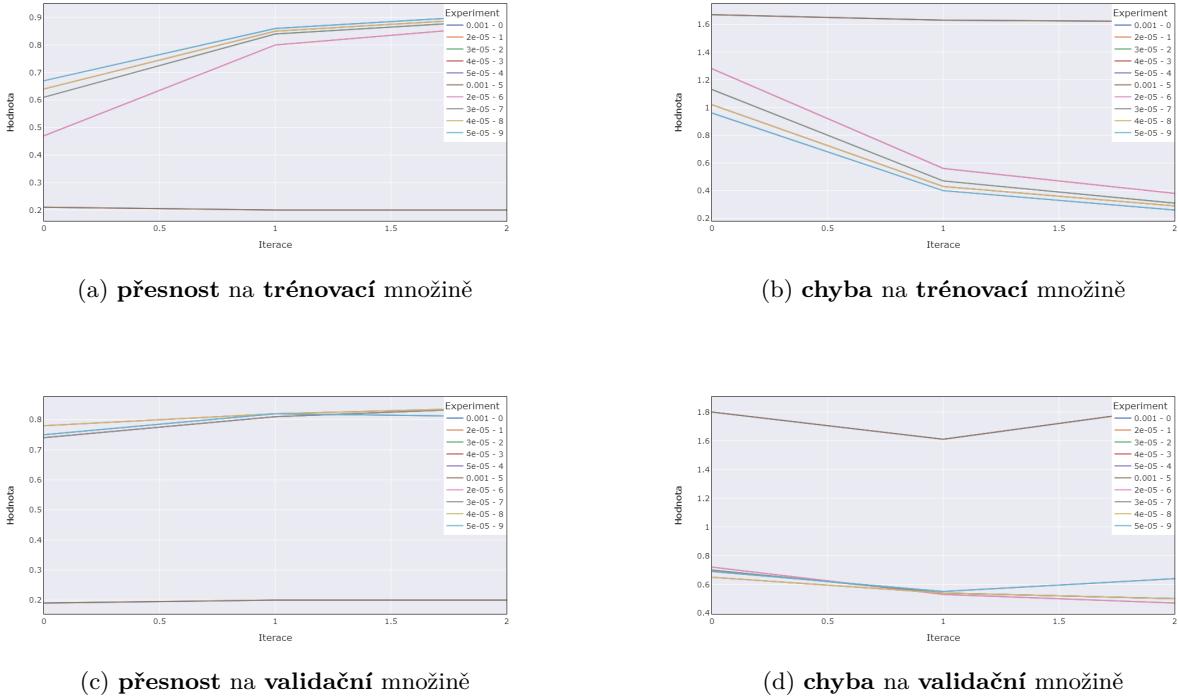


(b) trénování celého modelu

Obrázek 10.3: Kvalita obecného jazykového DistilBERT modelu

10.2.1 Učící konstanta

Učící konstanta je hodnota, kterou lze regulovat míru chyby na aktualizaci vah modelu. V článku [10] *BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding* autoři zmiňují hodnoty učící konstanty dle kterých model dosahuje nejlepších výsledků. S těmito hodnotami byly provedeny experimenty. Výsledky lze pozorovat na obrázku č. 10.4.



Obrázek 10.4: Experimentální ověření učící konstanty

Byl proveden 2krát stejný experiment s různými učícími konstantami (0.001 , $2e-5$, $3e-5$, $4e-5$, $5e-5$). Na obrázku č. 10.4 s grafy můžeme pozorovat stav přeучení po 1 iteraci u hodnoty $5e-5$. Zároveň hodnota 0.001 má tendenci způsobit stagnaci modelu. Důvodem je příliš vysoká hodnota, při které model není schopen upravit váhy tak, aby vnitřně modelovaly jazyk vzhledem k našemu problému. Z výsledných přesností u této učící konstanty lze pozorovat jev nazvaný *Catastrophic Forgetting*, který způsobí absolutní destrukci všech vědomostí o jazyku právě při procesu učení vědomostí nových. Po 3 iteracích nejlépe na **validační** množině dopadla učící konstanta $4e-5$. Vzhledem k chybě je prospěšnější hodnota $2e-5$, u které dochází k rychlejšímu spádu chyby. Finální výsledky predikce na **testovací** množině zaznamenává tabulka č. 10.1. Ze získaných informací můžeme zhodnotit hodnotu $2e-5$ jako nejvhodnější.

Tabulka 10.1: Experimentální výsledky nad testovací množinou vzhledem k učící konstantě

Učící konstanta	běh 1.	běh 2.
	přesnost	
0.001	21,5 %	21,5 %
$2e-5$	84,7 %	84,7 %
$3e-5$	84,7 %	84,7 %
$4e-5$	84,3 %	84,3 %
$5e-5$	81,8 %	81,8 %

10.2.2 Strategie extrakce vektorů

Vzhledem k obrázku č. 3.5 dochází k využití *[CLS]* vektoru reprezentující vektorovou reprezentaci celého vstupního textového segmentu. V této kapitole budou rozebrány odlišné strategie, jak z derivátu BERT modelu extrahovat vektorovou reprezentaci použitelnou ke klasifikaci. V implementaci objektu `BaseModelOutputWithPooling` [30] dostupného z knihovny HuggingFace jsou k dispozici 3 typy výstupu:

- **last_hidden_state** - s výstupem (*batch_size*, *sequence_length*, *hidden_size*).
- **hidden_states** - s výstupem (*number_of_layers*, *batch_size*, *sequence_length*, *hidden_size*).
- **pooler_output** - s výstupem (*batch_size*, *hidden_size*).

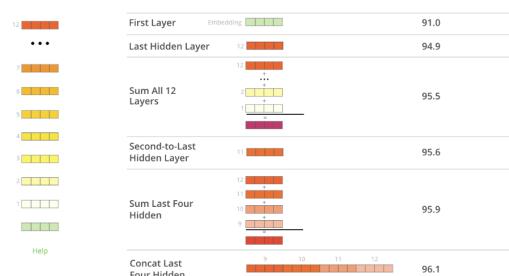
Kde hodnota *batch_size* reprezentuje počet vstupních záznamů v jedné dálce. *Sequence_length* velikost věty, která bývá ovlivněna definicí **tokenizeru**. Je potřeba definovat strategii, dle které bude docházet k operaci **doplnění** (*padding*) a **zkrácení** (*truncation*) u každého vstupujícího záznamu. *Hidden_size* je závislá na modelu derivátu. Například DistilBERT pracuje s velikostí 768. Hodnota je hyperparametr určený v době učení modelu. Poslední nepopsaný parametr *number_of_layers*

značí počet vrstev enkodéru. Příkladem obdobně bude uveden DistilBERT, kde je obsaženo 6 na sobě navazujících vrstev.

Byla implementována třída [BertPoolingLayer](#) (viz příloha B) ve frameworku TensorFlow sloužící právě k různým extrakcím větné reprezentace dokumentu pomocí odlišně definovaných strategií. [TransformerPooling](#) je definovaný typ navázáný na výstupy z [BaseModelOutputWithPooling](#) [30]. Typ [TransformerPoolingStrategy](#) mohl nabývat 4 následujících hodnot:

- [CLS](#) - výběr 0 pozice z výstupu enkodér bloku (pozice s CLS tokenem), výstup je reprezentován (*batch_size, hidden_size*).
- [Average](#) - průměr nad celým výstupem z modelu, respektive dochází k operaci sečtení všech vektorových reprezentací tokenů a následnou normalizaci dle velikosti věty, výstup má tvar (*batch_size, hidden_size*).
- [ConcatCLS](#) - spojení CLS tokenu z jednotlivých vrstev, výstup (*batch_size, hidden_size * number_of_layers*).
- [ConcatAverage](#) - spojení průměrů reprezentující větu z jednotlivých vrstev, výstup (*batch_size, hidden_size * number_of_layers*).

Z těchto typů šlo nadále zkonztruovat různé kombinace extrakce vektorů (strategie). Experimentováno bylo se zobrazenými na obrázku č. 10.5b. Obrázek 10.5a slouží k vizualizaci strategií.



(a) vizualizace extrakce

```

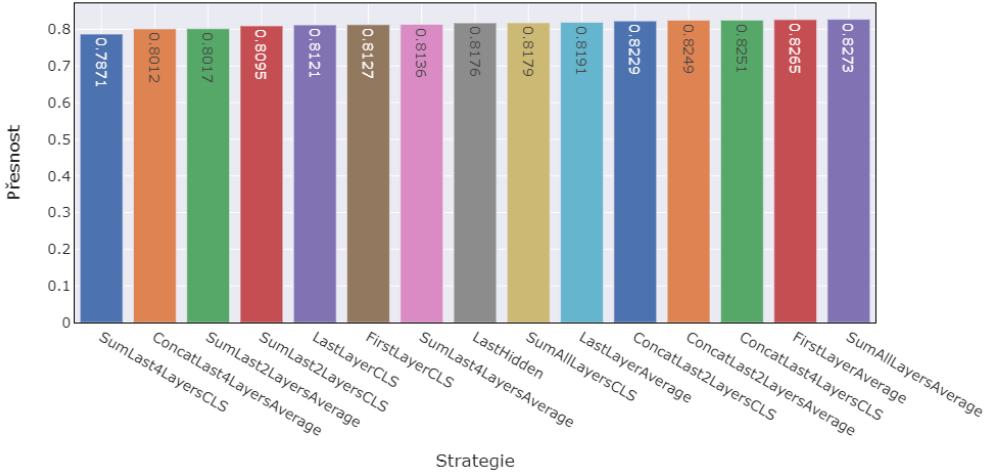
0 -> TransformerPoolingStrategySelection.SumAllLayersCLS
1 -> TransformerPoolingStrategySelection.SumLast4LayersCLS
2 -> TransformerPoolingStrategySelection.ConcatLast4LayersCLS
3 -> TransformerPoolingStrategySelection.LastLayerCLS
4 -> TransformerPoolingStrategySelection.FirstLayerCLS
5 -> TransformerPoolingStrategySelection.SumLast2LayersCLS
6 -> TransformerPoolingStrategySelection.ConcatLast2LayersCLS
7 -> TransformerPoolingStrategySelection.SumAllLayersAverage
8 -> TransformerPoolingStrategySelection.SumLast4LayersAverage
9 -> TransformerPoolingStrategySelection.ConcatLast4LayersAverage
10 -> TransformerPoolingStrategySelection.LastLayerAverage
11 -> TransformerPoolingStrategySelection.FirstLayerAverage
12 -> TransformerPoolingStrategySelection.SumLast2LayersAverage
13 -> TransformerPoolingStrategySelection.ConcatLast2LayersAverage
14 -> TransformerPoolingStrategySelection.Pooler
15 -> TransformerPoolingStrategySelection.LastHidden

```

(b) vlastní implementované metody [31]

Obrázek 10.5: Strategie získání číselných vektorů

Vzhledem k naměřeným přesnostem vycházela nejlépe strategie sečtení všech vrstev (viz obrázek č. 10.6). Experimenty byly prováděny na DistilBERT modelu nad 5 autory, 3 větami, s učící konstantou 5e-05, velikostí batche 64 a 5 iteracemi.

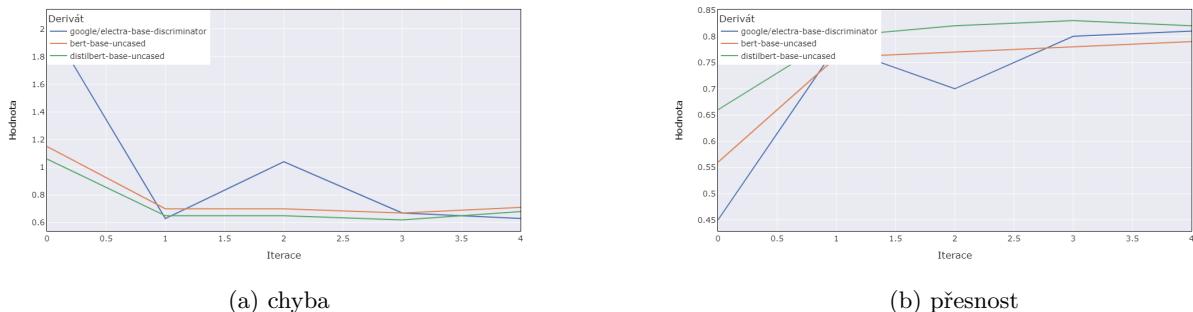


Obrázek 10.6: Výsledek přesnosti nad 5 autory, 3 větami a různými typy strategií

10.2.3 Typ derivátu

Další z experimentů testoval sílu kvalitu typu derivátu na problému autorství. Byly porovnány 3 deriváty ELECTRA, BERT a DistilBERT. Finální výsledek zaznamenaný na obrázku č. 10.7 ukazuje, že DistilBERT dosahuje nejlepších výsledků po 5 iteracích, i když ponecháním delšího učení by výsledek mohl dopadnout jinak.

Experimenty byly provedeny s nastavením velikost batche 64, učící konstanta 2e-05, předzpracování [CaseInterpunction](#), vstupující délka sekvence o 130 znacích.



Obrázek 10.7: Průběh derivátů

10.2.4 Větné okno

BERT model má vlastnost zachycení dlouhých závislostí oproti LSTM neuronové sítí. S tímto předpokladem byl navržen experiment rozličné velikosti vstupu v podobě počtu vět. Zároveň byl cíl dodržení pravidla nechat neuronové sítí vždy stejný počet textových dat. Tudíž pro vyšší počet vět byla použita nižší hodnota na podvzorkování vzhledem k danému autorovi (viz tabulka č. 10.2).

Tabulka 10.2: Počet trénovacích, testovacích a validačních záznamů v experimentu s větným oknem

Počet vět	počet záznamů v množině				hod. podvzorkování
	trénovací	validační	testovací		
1	162562	28688	33750		45000
2	81281	14344	16875		22500
3	54187	9563	11250		15000
7	23481	4144	4875		6500
10	16256	2869	3375		4500
15	10837	1913	2250		3000

Experiment byl proveden s modelem DistilBERT, velikosti batche s hodnotou 64, učící konstantou 5e-05 a povoleném počtu iterací neuronové sítě s hodnotou 5. Výsledky společně s dalšími hodnotami jsou zaznamenány v tabulce č. 10.3.

Tabulka 10.3: Výsledky experimentu s větným oknem

počet vět	přesnost	čas potřebný na (minuty)			velikost vstupu
		trénování	testování	vyhodnocení	
1	65,80 %	432	27	1,4	30
2	77,04 %	394	17	0,7	60
3	83,40 %	432	14	0,5	100
7	88,94 %	467	11	0,3	190
10	91,17 %	534	7	0,2	260
15	91.11 %	594	11	0,2	380

Z výsledků zobrazených v tabulce č. 10.3 lze pozorovat potvrzení schopnosti zachycení delších asociací ze vstupu, které jsou pro model prospěšné, a právě díky nim lépe odhaduje finálního autora.

10.2.5 Více autorů

Poslední zkoumanou vlastností sítě byla prediktivní schopnost nad větším množstvím autorů. Dospud byly výsledky experimentů prezentovány nad 5 autory. Experimenty byly provedeny nad 3 množinami, a to 5, 15 a 25 autory, více informací je zaznamenáno v tabulce č. 10.4.

Velikost sekvence vychází z hodnot získaných z explorativní analýzy tj. 90 pro 3 věty a 260 pro 10 vět (viz kapitola č. 6.4).

Tabulka 10.4: Informace o množinách k různému počtu autorů

		počet záznamů v množině				
počet autorů	počet vět	trénovací	validační	testovací	hod. podzorkování	velikost sekvence
5	3	54187	9563	11250	15000	90
5	10	16256	2869	3375	4500	260
15	3	54187	9563	11250	5000	90
15	10	54187	9563	11250	5000	260
25	3	54187	9563	11250	3000	90
25	10	54187	9563	11250	3000	260

Sít byla konfigurována parametry 64 velikost batche, učící konstanta 5e-05, model DistilBERT, s předzpracováním [CaseInterpunction](#). Výsledky jsou zaznamenány v tabulce č. 10.5

Tabulka 10.5: Výsledky experimentu s více autory

		čas potřebný na (minuty)			
Počet autorů	Počet vět	přesnost	trénování	testování	vyhodnocení
5	3	82,78 %	409	11	0,4
5	10	90,04 %	397	8	0,2
15	3	44,31 %	386	15	0,5
řž 15	10	83,29 %	1609	41	0,8
25	3	38,45 %	465	16	0,6
25	10	65,55 %	1500	25	0,8

V první řadě je možné poukázat na zkvalitnění modelu v případě zvýšení počtu vstupujících vět pro jeden záznam reprezentující kousek díla daného autora. Vzhledem k 5 autorům šlo o zlepšení 7 %. U 15 autorů došlo k zlepšení až o 40 %. U 25 autorů podobně vysoké zlepšení přibližně 30 %.

Kapitola 11

Finální zhodnocení

Z každého experimentálního bloku byl vybrán přístup, který dosahoval nejvyšších výsledků. Následně bylo provedeno srovnání těchto 3 přístupů, viz tabulka č. 11.1. Výsledky byly vybrány s podmínkou provedení experimentů nad datovou sadou s 5 autory (*Lyton, Edward Bulwer Lyton, Baron; Ebers, Georg, Twain, Mark; Kingston, William Henry Giles; Parker, Gilbert*) a 3 vstupními větami.

Tabulka 11.1: Výběr nejlepších modelů

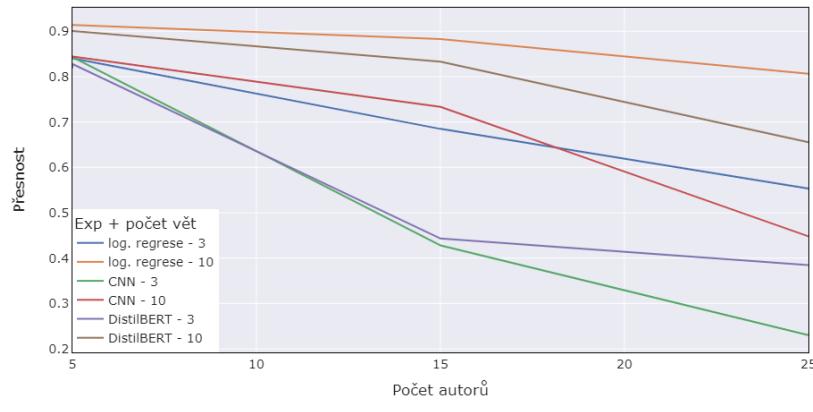
experimentální blok	typ	přesnost (%)
1	log. regrese + BoW	85,94
2	CNN + GloVe	84,28
3	DistilBERT	84,28

Porovnání nejlepších modelů nad zvýšeným počtem autorů, přesněji (5, 15 a 25) dopadlo jak znázorňuje obrázek č. 11.1. Zvětšením počtu vět v jednom záznamu, začaly všechny 3 modely fungovat s vyšší přesností. Sestupné seřazení dle přesnosti modelů je: logistická regrese, DistilBERT a poslední CNN. Časová náročnost experimentu je zobrazena na obrázku 11.1b, kde můžeme pozorovat pochopitelnou větší časovou náročnost u BERT modelu.

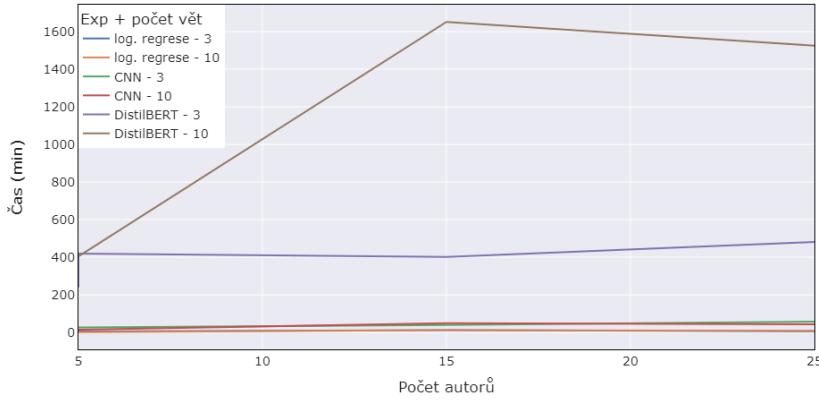
Kromě obrázku jsou výsledky z experimentů nad větším počtem autorů, provedených na testovací množině, zaznamenány v tabulce č. 11.2.

Tabulka 11.2: Výsledky nejlepších modelů u experimentu s větším počtem autorů

počet autorů	5		15		25	
	3	10	3	10	3	10
model						
<i>log. regrese + BoW</i>	84,04 %	91,38 %	68,47 %	88,28 %	55,31 %	80,62 %
<i>CNN + GloVe</i>	84,28 %	84,41 %	42,78 %	73,43 %	23,01 %	44,79 %
<i>DistilBERT</i>	82,78 %	90,04 %	44,31 %	83,29 %	38,45 %	65,55 %



(a) přesnost



(b) čas

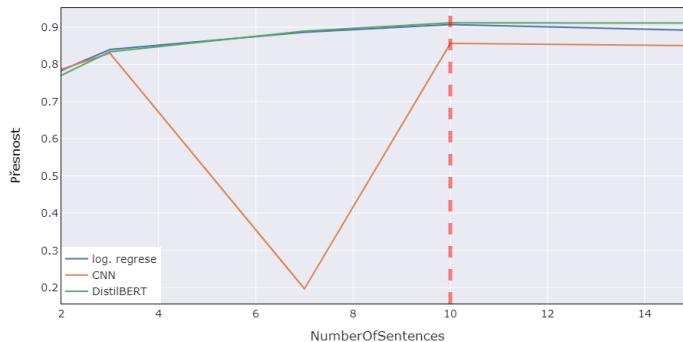
Obrázek 11.1: Výsledky vybraných modelů u více autorů

Stejně jako v předchozím případě byly vybrané nejlepší modely srovnány na problémku klasifikace nad 5 autory s variabilním větným oknem (každý záZNAM je reprezentován k větami). Výsledky zaznamenává tabulka č. 11.3.

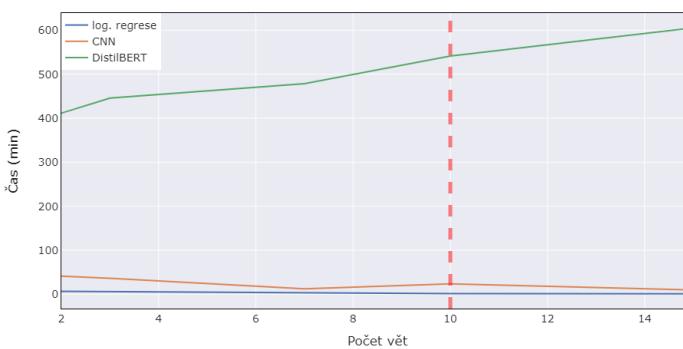
Tabulka 11.3: Výsledky nejlepších modelů nad variabilním větným oknem

model	5				
	2	3	7	10	15
<i>log. regrese + BoW</i>	78,38 %	83,98 %	88,64 %	90,70 %	89,16 %
<i>CNN + GloVe</i>	78,68 %	82,99 %	19,65 %	85,66 %	85,02 %
<i>DistilBERT</i>	77,04 %	83,40 %	88,94 %	91,17 %	91,11 %

Výsledky zmíněného experimentu jsou zachyceny na obrázku č. 11.2. Na obrázku je zaznačena hodnota velikosti větného okna (počtu vět reprezentující jeden záznam), kde DistilBERT dosahuje lepších výsledků než logistická regrese (tj. hodnota 10). Zároveň můžeme pozorovat, že na zmíněné hodnotě se mění trend modelů. Od 10 již nastává chování, kde změna přesnosti není tak vysoká jako v předchozích krocích.



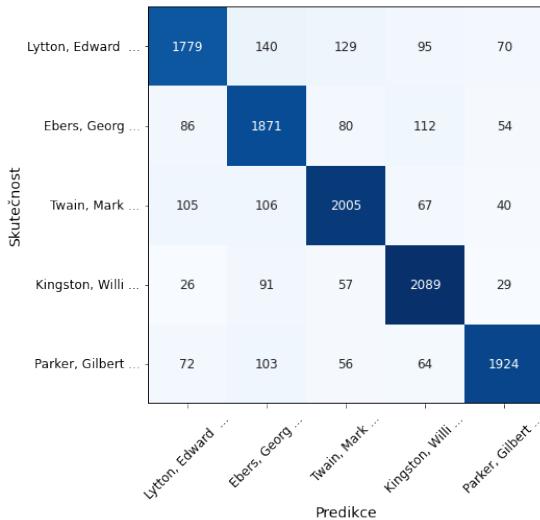
(a) přesnost



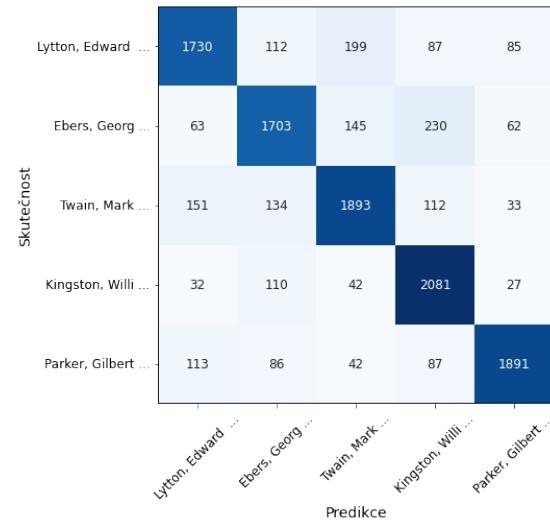
(b) čas

Obrázek 11.2: Výsledky vybraných modelů u experimentu s variabilním větným oknem

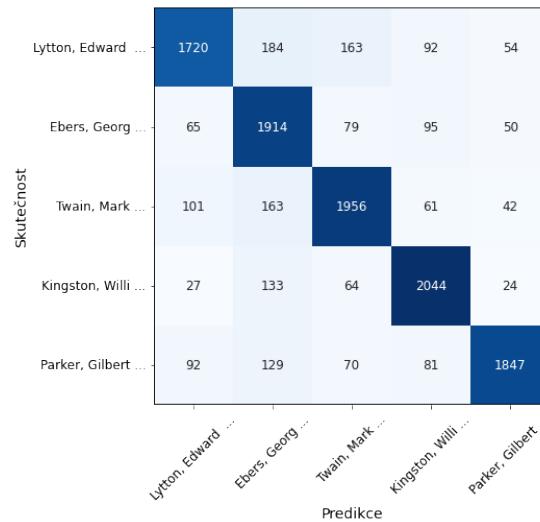
Na obrázku č. 11.3 jsou zobrazeny tři chybové matice, tj. jedna pro každý testovaný model. Na všech maticích je možno pozorovat podobný vzor chování v predikci. Dochází ke stejnemu seřazení úspěšnosti, chybovosti predikce autorů.



(a) chybová matice klasického experimentu



(b) chybová matice experimentu neuronových sítí



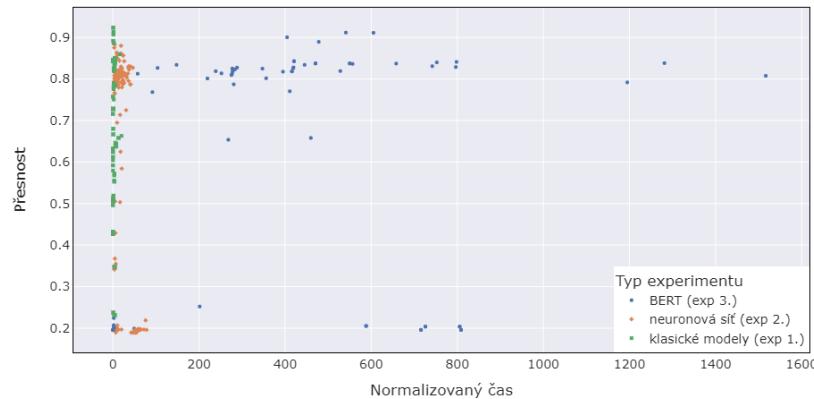
(c) chybová matice BERT modelu

Obrázek 11.3: Srovnání chybových matic nejlepších modelů

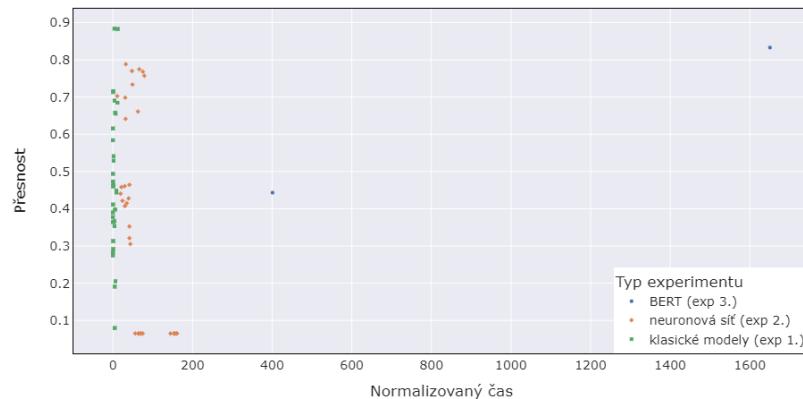
Srovnání přesnosti a výpočetní náročnosti zkoušených modelů

Aby bylo možné srovnat výsledky z jednotlivých experimentů byla vytvořena vizualizace zobrazena na obrázku č. 11.4. Vizualizace, kde na vertikální ose je nanesena výsledná přesnost z testovací množiny a na horizontální normalizovaný čas (*učení a následná predikce*) do intervalu od 0 do 1.

Grafy reprezentují všechny provedené experimenty a jejich výsledky v dvoudimenzionálním prostoru popsaném časem a přesnosti modelu.



(a) 5 autorů



(b) 15 autorů

Obrázek 11.4: Srovnání všech experimentů nad 5 a 15 autory

Je možné především vyzdvihnout vysoké přesnosti u BERT modelů. Ačkoliv některé klasické modely dosahovaly podobných, neli lepších výsledků v lepším čase. Poukázalo se na nedostatečnost neuronových sítí (*CNN, Dense, LSTM*), které nedokázaly konkurovat předchozím dvou zmíněným. Tvrzení platí, pokud bereme v potaz provedení procesu nalezení správných hyperparametrů. Z

předchozích experimentů totiž víme, že DistilBERT model dosahoval vysokých přesností až po zvětšení větného okna, kde porážel logistickou regresi. S odkazem na předchozí experimenty 11.2 jsme mohli pozorovat, že menší větné okno prospívalo 2. kategorií experimentů (tj. neuronové sítě).

Shrnutí

Po provedení vyhodnocení prvního bloku experimentů, který se zabýval provedením pokusů nad kombinacemi mezi typem vektorizace ve spojení s klasickými klasifikátory bylo zjištěno, že vektorová reprezentace získána z BERT modelu a následné využití těchto vektorů u klasifikace nedosahovalo vysokých výsledků. Můžeme přepokládat že tento mnohodimenzionální prostor dokumentů produkovaný state-of-art přístupy je pro klasické modely nevhodný, jelikož nad ním modely nejsou schopny vytvořit pravidla, kterými by následně byly schopny správně predikovat třídu. Ačkoliv lepších, i tak ne dostatečně dobrých výsledků dosahovala vektorová reprezentace získána z GloVe a Word2Vec algoritmu. Z obrázku č. 8.1 je možné pozorovat, že klasifikátor nejbližších sousedů s těmito vektorovými reprezentacemi byl nejkompatibilnější. V ostatních případech klasická BoW a TF-IDF reprezentace dosahovala nejvyšších výsledků. V průběhu experimentování bylo dosaženo přesnosti 86 % nad 5 autory a 3 větami.

Druhý experimentální blok byl zaměřen na prozkoumání neuronových sítí až po architekturu Transformer (tj. *CNN*, *LSTM*, *Dense*). Prvně byla textová data transformována do vektorové reprezentace pomocí vrstvy s názvem *EmbeddingLayer* s následným závěsem architektury, která extra-hovala informace ze získaných vlastností slov a ty využila při zjištění *autorství*. Provedením experimentů nad různými kombinacemi velikosti dimenze popisující vlastnosti jednoho tokenu, typem architektury a inicializační hodnotou pro zmíněnou *EmbeddingLayer*, byly vybrány architektury CNN a Dense, vzhledem k přesnosti a času, jako nejhodnější. Výsledná přesnost na 5 autorech a 3 větách se pohybovala kolem 82,50 % u obou architektur. Identifikovatelný rozdíl mezi architekturami byla ona inicializační hodnota. Pro CNN dosahovala nejvyšších výsledků GloVe incializace, kdyžto pro Dense inicializace prázdná. Vzhledem k modelování potřebné velikosti vektoru nad *EmbeddingLayer* s prázdnou inicializací byla nalezena hodnota dimenze 70 jako nejhodnější kandidát.

Poslední experimentální blok cílil na hlavní zaměření této diplomové práce, a to deriváty BERT modelu. Experimenty byly provedeny nad různými deriváty, konfiguracemi a strategiemi extrakce vektoru z BERT modelu. Zároveň při tomto experimentálním bloku byly navrženy experimenty, které využívaly variabilní velikost větného okna pro jednotlivé záznamy, kde šlo pozorovat pozitivní dopad v podobě zvýšení přesnosti modelu. Během provedení mnoha experimentů byly nalezené vhodné hodnoty, které by mohly být nevhodnější volbou nastavení modelu, když bychom chtěli dosáhnout nejvyšších přesností:

- učící konstanta - $2e-5$
- velikost batche - 64
- strategie - *SumAllLayersAverage*
- typ derivátu - *DistilBERT*
- velikost větného okna - 10

Konfigurace byla efektivní při klasifikaci 5 autorů. Výsledná přesnost by měla dosahovat až 90% přesnosti.

Závěrem byly vybrány nevhodnější modely z každého experimentálního bloku a nad nimi provedeny experimenty s větším počtem autorů a variabilním větným oknem. Přece jen když referenční výsledky BERT modelu nad různými problémy zpracování přirozeného jazyka dosahují nejlepších výsledků v problému řešící autorství mu nad datovou sadou získanou z *Gutenberg Projekt* vysoce konkurovala logistická regrese ve spojení s BoW. Výše v tabulkách 11.2 a 11.1 je možné pozorovat, pro jednotlivé problémy, dosaženou maximální přesnost u právě zmíněných kombinací. Vysoká kvalita BERT modelu nelze u klasifikačního problému jednoznačně pozorovat, jelikož se jedná o jednodušší problém v doméně zpracování přirozeného jazyka, ale i přesto BERT model dosáhl na problému nejvyšší přesnosti 91,17 %.

Kapitola 12

Závěr

Cílem této diplomové práce bylo seznámení s problematikou hlubokých neuronových sítí v doméně zpracování přirozeného jazyka. Hlavním záměrem bylo zaměření na architekturu Transformer, na které je založen moderní jazykový model BERT. Tomuto modelu byla věnována velká část práce.

Značný úsek práce byl věnován jednotlivým komponentám z procesu zpracování přirozeného jazyka. Úvodní teoretické části popisovaly základní principy předzpracování textových dat, starší vektorizační techniky transformující textová data do mnohodimenzionálních číselných vektorů a různé typy klasifikátorů. Tyto základní principy jsou základem pro pochopení fungování state-of-art modelu BERT založeném na Transformer architektuře. Právě mimo popsané základní principy teoretická část rozebírá postupně jednotlivé části z Transformer architektury, a tak poodhaluje i fungování BERT modelu. Kromě obecného popisu Transformer architektury je BERT model rozbrán z pohledu vstupujících dat, procesu předučení, možnosti aplikace, a následná derivace v podobě ELECTRA a DistilBERT modelu.

Potom co byla podchycena teorie, pak byl k modelům definován problém autorství, nad kterým byly navrženy 3 postupy. Ty byly experimentálně prověřeny v praktické části. Abychom experimenty mohli provádět, byla využita veřejně dostupná digitální knihovna *Projekt Gutenberg*, z které byly automatizovaně staženy vybraná umělecká díla. Z těchto stažených uměleckých děl bylo obdobně s pomocí automatizace vytvořeny různé datové sady s definovanými parametry, a to počtem autorů a počtem vět.

Následně co byla dostupná datová sada a navrženy experimentální postupy, přistoupilo se k automatizovanému spouštění experimentů s systematickým uložením výsledků. Tyto výsledky posloužily ve finální části k vyhodnocení kvality navržených postupů.

Experimenty byly rozděleny do třech částí s vlastností inkrementálního zesložitění. Tudíž v prvním bloku byly prozkoumány kvality kombinace klasických modelů s rozličnými přístupy k vektorizaci dokumentů. V druhém kvalita neuronových sítí s kódovací vrstvou. A nakonec, v třetím, hlavní doména diplomové práce, a to BERT model s hlubokou dopřednou sítí.

Z navržených postupů nejlépe dopadla kombinace logistické regrese ve spojení s BoW vektorizací, která byla schopna často konkurovat DistilBERT derivátu odvozeného z originální BERT modelu. Ačkoliv nejlepšího výsledku bylo dosaženo pomocí DistilBERT modelu.

Pokračováním této práce by se mohlo soustředit na navržení sofistikovanějšího přístupu k hledání hyperparametrů u BERT modelu. Případně aplikací BERT modelu nad jiným typem problému. Příkladem by mohla být summarizace textu, datování textu nebo případná aplikace modelů při optimalizaci vyhledávače. Zejména je otázkou, zda by na jiných problémech šlo pozorovat hodnotu BERT modelu jednoznačněji.

Literatura

1. AGGARWAL, Charu C. *Machine Learning for Text*. Springer Publishing Company, Incorporated, 2008. ISBN 978-3-319-73530-6.
2. *Stemming and lemmatization* [online]. c2008 [cit. 2021-03-05]. Dostupné z: <https://nlp.stanford.edu/IR-book/html/htmledition/stemming-and-lemmatization-1.html>.
3. *The Porter Stemming Algorithm* [online] [cit. 2022-04-23]. Dostupné z: <https://tartarus.org/martin/PorterStemmer/>.
4. *A General Approach to Preprocessing Text Data* [online]. c2021 [cit. 2021-03-05]. Dostupné z: <https://www.kdnuggets.com/2017/12/general-approach-preprocessing-text-data.html>.
5. HUILGOL, Purva. *Quick Introduction to Bag-of-Words (BoW) and TF-IDF for Creating Features from Text* [online]. c2013-2020 [cit. 2021-03-05]. Dostupné z: <https://www.analyticsvidhya.com/blog/2020/02/quick-introduction-bag-of-words-bow-tf-idf/>.
6. SCOTT, William. *TF-IDF from scratch in python on real world dataset* [online] [cit. 2021-03-05]. Dostupné z: <https://towardsdatascience.com/tf-idf-for-document-ranking-from-scratch-in-python-on-real-world-dataset-796d339a4089>.
7. RONG, Xin. *word2vec Parameter Learning Explained* [online] [cit. 2021-03-05]. Dostupné z: <https://arxiv.org/pdf/1411.2738.pdf>.
8. MIKOLOV, Tomas. *Efficient Estimation of Word Representations in Vector Space* [online] [cit. 2021-03-05]. Dostupné z: <https://arxiv.org/pdf/1301.3781.pdf>.
9. *GloVe: Global Vectors for Word Representation*. Dostupné také z: <https://nlp.stanford.edu/projects/glove/>.
10. DEVLIN, Jacob; CHANG, Ming-Wei; LEE, Kenton; TOUTANOVA, Kristina. *BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding*. arXiv, 2018. Dostupné z DOI: 10.48550/ARXIV.1810.04805.
11. *HuggingFace*. Dostupné také z: <https://huggingface.co/>.

12. SANH, Victor; DEBUT, Lysandre; CHAUMOND, Julien; WOLF, Thomas. *DistilBERT, a distilled version of BERT: smaller, faster, cheaper and lighter*. arXiv, 2019. Dostupné z DOI: 10.48550/ARXIV.1910.01108.
13. CLARK, Kevin; LUONG, Minh-Thang; LE, Quoc V.; MANNING, Christopher D. *ELECTRA: Pre-training Text Encoders as Discriminators Rather Than Generators*. arXiv, 2020. Dostupné z DOI: 10.48550/ARXIV.2003.10555.
14. *ELECTRA Vs BERT*. Dostupné také z: <https://bijular.medium.com/electra-vs-bert-a-comparative-study-36f07ba88e61>.
15. *Logistic Regression in Machine Learning*. Dostupné také z: <https://medium.com/analyticsvidhya/logistic-regression-in-machine-learning-f3a90c13bb41>.
16. *METRICKÉ PROSTORY*. [B.r.]. Dostupné také z: <https://matematika.cuni.cz/dl/analyza/29-mtr/lekce29-mtr-pmax.pdf>.
17. *Recurrent Neural Networks*. Dostupné také z: <https://andrew.gibiansky.com/blog/machine-learning/recurrent-neural-networks/>.
18. *Recurrent Neural Networks*. Dostupné také z: <https://colah.github.io/posts/2015-08-Understanding-LSTMs/>.
19. *Understanding GRU Networks*. Dostupné také z: <https://towardsdatascience.com/understanding-gru-networks-2ef37df6c9be>.
20. *Convolutional Neural Network in Natural Language Processing*. Dostupné také z: <https://towardsdatascience.com/convolutional-neural-network-in-natural-language-processing-96d67f91275c>.
21. *NLP with CNNs*. Dostupné také z: <https://jalammar.github.io/illustrated-transformer/>.
22. VASWANI, Ashish; SHAZER, Noam; PARMAR, Niki; USZKOREIT, Jakob; JONES, Llion; GOMEZ, Aidan N.; KAISER, Lukasz; POLOSUKHIN, Illia. *Attention Is All You Need*. arXiv, 2017. Dostupné z DOI: 10.48550/ARXIV.1706.03762.
23. ALAMMAR, Jay. *The Illustrated Transformer*. Dostupné také z: <https://jalammar.github.io/illustrated-transformer/>.
24. *Project Gutenberg*. Dostupné také z: <https://www.gutenberg.org/>.
25. *Natural Language Toolkit* [online]. c2021 [cit. 2021-03-05]. Dostupné z: <https://www.nltk.org/>.
26. *TensorFlow* [online] [cit. 2021-03-05]. Dostupné z: <https://www.tensorflow.org/>.
27. *Keras* [online] [cit. 2021-03-05]. Dostupné z: <https://keras.io/about/>.

28. ŘEHŮŘEK, Radim; SOJKA, Petr. Software Framework for Topic Modelling with Large Corpora. In: *Proceedings of the LREC 2010 Workshop on New Challenges for NLP Frameworks*. Valletta, Malta: ELRA, 2010-05, s. 45–50. <http://is.muni.cz/publication/884893/en>.
29. ROBINSON, David. *gutenbergr: Search and download public domain texts from Project Gutenberg*. Dostupné také z: <https://cran.r-project.org/web/packages/gutenbergr/vignettes/intro.html>.
30. *TensorFlow* [online] [cit. 2022-04-22]. Dostupné z: https://huggingface.co/docs/transformers/main/en/main_classes/output#transformers.modeling_outputs.BaseModelOutputWithPooling.
31. ALAMMAR, Jay. *The Illustrated BERT, ELMo, and co. (How NLP Cracked Transfer Learning)*. Dostupné také z: <https://jalammar.github.io/illustrated-bert/>.

Příloha A

Struktura projektu

- **jupters** - *.ipynb* soubory se pouštěnými experimenty, načtení všech experimentů z adresáře *experiment_results*, *.ipynb* soubor se statistikou pro datové sady
- **experiment_results** - adresář s všemi proběhlými experimenty
- **gutenberg_downloaded** - adresář, kde jsou díla z **Projektu Gutenberg** staženy (*konfiderenciálné*)
- **src** - adresář se zdrojovými kódy
 - **analysis** - adresář použitelných kódů pro následnou analýzu výsledků, případnou explorační analýzu
 - * **experiments** - kód načítá rekurzivně soubory z proběhlých experimentů
 - * **stats** - načítá všechny cesty k datovým sadám a spustí výpočet základní statistiky nad soubory
 - **authors** - adresář s objektem který načítá a vybírá n autorů
 - **callbacks** - umístění všech *callback* objektu, které jsou využity v neuronových sítích
 - **config** - konfigurační soubory projektu, umístění uložení datové sady, velikost trénovací množiny a podobně
 - **data_loading** - pomocné metody pro načtení datových sad
 - **data_peparing** - skripty pro automatické vytvoření, zpracování datové sady
 - * **build_dataset** - skripty, které vytvářejí *.csv* soubor pro datovou sadu o n autorech a k větách
 - * **split** - skripty, které dokáží rozdělit *.csv* soubor na *train.csv*, *test.csv*, *valid.csv*
 - **defined_types** - definované typy použité napříč projektu
 - **download** - skripty pro stažení datové sady z **Projektu Gutenberg**

- * **r** - umístění r skriptů s knihovnou *gutenberger*
- * **python** - .py skripty spouštějící .R skripty
- **encoder** - umístění metody, která vytváří enkodér identifikátorů
- **experiments** - adresář s pomocnými metodami pro spouštění experimentů
 - * **descriptions** - metoda pro vytvoření *description.csv* souborů
 - * **experiments_scripts**
 - **classic** - skripty spouštějící experimenty pro klasické klasifikátory společně s různou vektorizací textu
 - **experiment_configurations** - konfigurace pro experimenty
 - **neural_nets** - skripty spouštějící experimenty pro neuronové sítě a transformery
 - **test** - skripty spouštějící testovací experimenty pro všechny typy
 - **types** - definované typy experimentů
 - * **helpers** - pomocné objekty, které slouží pro uchování hodnot z experimentů
 - * **results** - evalvační metody
 - * **settings** - objekt s nastavením pro neuronovou síť
- **models** - adresář s definovanými modely k experimentům
 - * **classic** - obsažení klasickým modelů (*NB, RandomForest*)
 - * **embedding** - definice vrstvy pro získání vektorové reprezentace v neuronové síti
 - * **nets** - definice architektur neuronových sítí
 - * **transformer** - skript pro vytvoření transformer sítě společně s vrstvou, která extractuje vektorou reprezentaci z derivátů založených na BERT modelu
- **preprocessing** - adresář s metodami pro předzpracování textu
- **statistic** - adresář se spustitelnými statistickými kalkulacemi
 - * **instances** - různé objekty provádějící kalkulace
 - * **types** - definice typů pro instance
 - * **utils** - pomocné metody využití u statistiky
- **testing** - adresář s testovací datovou sadou
- **tokenizers** - adresář s tokenizorem pro transformer modely
- **types** - adresář s definovanými typy, které jsou využívány v projektu
- **utils** - kombinace různých pomocných metod
- **vectorizers** - adresář s objekty, které extrahují z textu číselné vektory
 - * **classic** - klasické přístupy (*BoW, TfIdf*)

- * **embedding** - přístup extrakce v podobě neuronové sítě (předučené pomocí *Word2Vec* a *GloVe*)
- * **transformer** - vektorizace pomocí *ELECTRA*, *DistilBERT* a *BERT* modelu
- **visualization** - pomocné metody sloužící k vizualizaci

Příloha B

BERTPoolingLayer

```
class BertPoolingLayer(tf.keras.layers.Layer):
    def call(
        self,
        inputs,
        pooling_type: TransformerPooling,
        transformer_pooling_strategy: typing.Union[
            None, TransformerPoolingStrategy
        ] = None,
        transformer_start_index: typing.Union[
            int, typing.Callable[[int], int]
        ] = lambda x: 0,
        transformer_end_index: typing.Union[
            int, typing.Callable[[int], int]
        ] = lambda x: 0,
    ):
        if pooling_type == TransformerPooling.LastHiddenState:
            last_hidden_state = inputs[TransformerPooling.LastHiddenState.value]
            return tf.reduce_mean(last_hidden_state, axis=1)
        if pooling_type == TransformerPooling.Pooler:
            pooler = inputs[TransformerPooling.Pooler.value]
            return pooler
        if pooling_type == TransformerPooling.HiddenStates:
            selector = inputs[TransformerPooling.HiddenStates.value]
            number_of_layers = len(selector) - 1
            index_start_from_behind = number_of_layers - transformer_start_index(
                number_of_layers
```

```
)  
index_end_from_behind = (  
    number_of_layers - transformer_end_index(number_of_layers) + 1  
)  
selector = selector[index_start_from_behind:index_end_from_behind]  
if transformer_pooling_strategy in [  
    TransformerPoolingStrategy.ConcatAverage,  
    TransformerPoolingStrategy.ConcatCLS,  
]:  
    concatenated = tf.concat(selector, axis=2)  
    if transformer_pooling_strategy == TransformerPoolingStrategy.  
        ConcatCLS:  
        cls = concatenated[:, 0, :]  
        return cls  
    else:  
        averaged_sentence = tf.reduce_mean(concatenated, axis=1)  
        return averaged_sentence  
else:  
    tf_tensor = tf.convert_to_tensor(selector)  
    averaged = tf.reduce_mean(tf_tensor, axis=0)  
    if transformer_pooling_strategy == TransformerPoolingStrategy.CLS:  
        cls = averaged[:, 0, :]  
        return cls  
    else:  
        averaged_sentence = tf.reduce_mean(averaged, axis=1)  
        return averaged_sentence
```
