Embedding dat pro velké jazykové modely

Šimon Nehéz

Bakalářská práce
2024

Univerzita Tomáše Bati ve Zlíně
Fakulta aplikované informatiky

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

Akademický rok: 2023/2024

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

(projektu, uměleckého díla, uměleckého výkonu)

Jméno a příjmení:	**Šimon Nehéz**
Osobní číslo:	**A21271**
Studijní program:	**B0613A140020 Softwarové inženýrství**
Forma studia:	**Prezenční**
Téma práce:	**Embedding dat pro velké jazykové modely**
Téma práce anglicky:	**Embedding Data for Large Language Models**

Zásady pro vypracování

1. Prozkoumejte metodiky převodu textu na matematické vyjádření sémantické podobnosti.
2. Připravte dostatečně rozsáhlý testovací dataset v různých souborových formátech.
3. Vytvořte prototyp webové aplikace v jazyku Python, která bude podporovat embedding dat ve všech připravených formátech sestaveného datasetu.
4. Využijte aplikaci a dataset k získání numerické reprezentace dat. Vyhodnoťte výstupy na základě kriterií jako je rychlost zpracování a paměťová náročnost.
5. Nad získanými výsledky realizujte analýzu s cílem stanovení nejvhodnějšího datového formátu pro extrakci dat za účelem jejich využití ve velkých jazykových modelech.

Forma zpracování bakalářské práce: **tištěná/elektronická**
Jazyk zpracování: **Slovenština**

Seznam doporučené literatury:

1. JURAFSKY, Daniel a James H. MARTIN, 2009. Speech and Language Processing An Introduction to Natural Language Processing, Computational Linguistics, and Speech Recognition. 2nd ed. Pearson Prentice Hall. ISBN 9780131873216.
2. EGGER, Roman, 2022. Applied Data Science in Tourism Interdisciplinary Approaches, Methodologies, and Applications. Springer. ISBN 3030883884.
3. GEORGE, Alexandra, 2022. Python text mining: Perform text processing, word embedding, text classification and Machine Translation. 1. BPB Publications. ISBN 9389898781.
4. Python Docs – Oficiální Dokumentace Programovacího Jazyka Python [online], 2008. 2023 [cit. 2023-11-09]. Dostupné z: https://docs.python.org/3/.

Vedoucí bakalářské práce: **Ing. Jozef Kováč**
Ústav informatiky a umělé inteligence

Datum zadání bakalářské práce: **5. listopadu 2023**
Termín odevzdání bakalářské práce: **13. května 2024**

UNIVERZITA TOMÁŠE BATI VE ZLÍNĚ
Fakulta aplikované informatiky
DĚKAN
L.S.

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

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

Ve Zlíně dne 5. ledna 2024

Prohlašuji, že

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

Prohlašuji,

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

Ve Zlíně, dne 13.5.2024 Šimon Nehéz, v. r.

ABSTRAKT

Bakalárska práca sa zaoberá prevodom rôznych súborových formátov do vektorovej podoby pričom hodnotí kvalitu týchto vektorov. Teoretická časť práce opisuje rôzne metódy prevodu textu do vektorovej podoby, zatiaľ čo praktická časť obsahuje zostavenie testovacieho datasetu, vytvorenie prototypu webovej aplikácie na generovanie embeddingov v jazyku Python a vyhodnotenie experimentu. Experiment ukázal minimálne rozdiely v kvalite embeddingov medzi skúmanými súborovými formátmi (maximálny rozdiel 2.7 percenta) pričom formát CSV bol najefektívnejší z hľadiska časovej náročnosti načítania súborov. Formát TXT dosiahol najnižšiu pamäťovú náročnosť zatiaľ čo formát HTML bol najpomalší a najviac pamäťovo náročný. Práca poskytuje hodnotné poznatky o výkonnosti rôznych súborových formátov pre embeddovanie dát.

Klíčová slova: embedding, veľké jazykové modely, súborové formáty, sémantická podobnosť

ABSTRACT

The bachelor thesis deals with the conversion of various file formats into vectors while evaluating the quality of these vectors. The theoretical part of the thesis describes multiple methods of converting text to vector representation, while the practical part includes the construction of a test dataset, the creation of a prototype web application for generating embeddings in Python and the evaluation of the experiment. The experiment showed minimal differences in the quality of embeddings between the file formats studied (maximum difference of 2.7 percent), with the CSV format being the most efficient in terms of time required to load the files. The TXT format achieved the lowest memory intensity while the HTML format was the slowest and most memory intensive. The work provides valuable insights into the performance of different file formats for data embedding.

Keywords: embedding, large language models, file formats, semantic similarity

Ďakujem môjmu vedúcemu Ing. Jozefovi Kováčovi za cenné rady, podnety, vždy správne nasmerovanie a veľkú ústretovosť počas písania bakalárskej práce.

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

Prohlašuji, že při tvorbě této práce jsem použil nástroj generativního modelu AI ChatGPT; https://chat.openai.com/ za účelem urychlení vývoje prototypu aplikace. Po použití tohoto nástroje jsem provedl kontrolu obsahu a přebírám za něj plnou zodpovědnost.

OBSAH

ÚVOD

Nedávny rapídny rozvoj v oblasti veľkých jazykových modelov zapríčinil, že sa v súčasnosti veľké jazykové modely stávajú stále viac neoddeliteľnou súčasťou rôznych aplikácií a systémov pracujúcimi s textovými dátami. Základným stavebným kameňom týchto modelov je efektívne prevádzanie textu do vektorových reprezentácií, čo umožňuje počítačom lepšie porozumieť a spracovať ľudský jazyk. Táto bakalárska práca, "Embedding dát pre veľké jazykové modely", skúma rôzne spôsoby prevodu textu na vektory a hodnotenie sémantickej podobnosti týchto vektorov.

Cieľom tejto bakalárskej práce je prispieť k hlbšiemu porozumeniu a efektívnemu využitiu embeddingov dát použitých s veľkými jazykovými modelmi, čím sa môže otvoriť cesta k inováciám a zlepšeniu spracovania textu v rôznych aplikáciách a systémoch.

Teoretická časť tejto práce sa zameriava na vysvetlenie rôznych metód prevodu textu do vektorov, vrátane popisu vektorových reprezentácií textu a spôsobov hodnotenia sémantickej podobnosti vektorov. Okrem toho táto časť popisuje spôsob hodnotenia korelácie medzi číselnými hodnotami, ktorý je použitý v praktickej časti.

Cieľom praktickej časti tejto práce je zistiť, či je niektorý z vybraných súborových formátov lepší pre tvorbu embeddingov ako ostatné, pričom sleduje časovú a pamäťovú náročnosť spracovania dát. Zameriava sa na zostavenie vhodného testovacieho datasetu, vrátane výberu, spracovania a prípravy dát. Taktiež je v tejto časti popísaná konzolová aplikácia využitá na úpravu a rozšírenie datasetu do vhodnejšej podoby pre využitie v tejto práci. Obsahuje tiež popis rôznych súborových formátov použitých v experimente a detailný popis aplikácie, ktorá obsahuje dve časti. Prvá časť bola v rámci tejto práce vyvinutá pre zjednodušenie prevodu veľkého množstva textu na vektorové reprezentácie daných textov. Druhá časť prototypu bola vyvinutá pre jednoduché porovnanie kvality vektorových reprezentácií z dát, ktoré vygenerovala prvá časť prototypu.

I. TEORETICKÁ ČÁST

1 TECHNICKÉ POJMY

1.1 Embedding

Embedding je pojem, ktorý označuje slovo, alebo text prevedený do vektorovej podoby so zachovaním istej miery sémantického významu daného slova alebo textu. Tento pojem sa niekedy používa prísnejšie ako názov pre *„dense“* vektory napríklad z metódy *word2vec* a môže byť použitý aj pre *„sparse“* vektory napríklad z metódy *TF-IDF* [1].

1.2 Natural Language Processing

NLP je pojem označujúci oblasť umelej inteligencie, v ktorej počítače užitočne a inteligentne analyzujú, rozumejú a získavajú význam informácií z ľudského jazyka. Pod túto oblasť spadajú témy ako automatická sumarizácia, prekladanie, sémantická analýza a iné [2].

1.3 Large Language Model

LLM je algoritmus hlbokého učenia, ktorý dokáže rozpoznávať, sumarizovať, prekladať, predpovedať a generovať text, alebo iné formy obsahu na základe vedomostí získaných z obrovského počtu dát [3].

1.4 Open source

Originálne je tento pojem nazývaný *„open source software“*. Označuje software, ktorého zdrojový kód je verejne dostupný a ktokoľvek ho môže akokoľvek upraviť a distribuovať [4].

2 SÉMANTICKÁ PODOBNOSŤ

Táto kapitola sa zaoberá spôsobmi, akými sa dá jazyk reprezentovať podobou, ktorej počítače rozumejú. Zameriava sa na objasnenie, prečo je taká podoba potrebná a opisuje rôzne možnosti ako takej podoby dosiahnuť. Tiež sa v tejto kapitole popisuje ako sa hodnostní veľkosť sémantickej podobnosti, medzi reprezentáciami textu, ktorej rozumejú počítače.

2.1 Vektorové reprezentácie textu

Jazyk sa vyvíjal rokmi evolúcie a ľudského používania. Kvôli tomu sú jazyky tak komplexné, že je pre nás ľudí veľmi ťažké pochopiť niektoré z ich častí. Ak je pre nás ťažké porozumieť niektorým častiam prirodzeného jazyka, môžeme predpokladať, že aktuálne technológie s tým majú podobný problém. Keďže počítače pracujú s číslami, musíme v prvom rade previesť jazyk do podoby čísel [5].

Reprezentovať význam slov pomocou vektorov je v oblasti NLP štandard. Základy tohto modelu pochádzajú z 1950 kedy sa prišlo s nápadom použiť bod v trojdimenzionálnom priestore na reprezentáciu konotácie slova. Druhý dôležitý nápad bol definovať význam slova pomocou jeho distribúcii pri jeho používaní, teda pomocou okolitých slov alebo gramatického prostredia daného slova. Pointa tohto nápadu bola, že dve slová, ktoré sa vyskytujú v podobnej distribúcii (ich okolité slová sú podobné) môžu mať podobný význam [1].

Vektory reprezentujúce slová sa volajú embeddingy. Tento pojem sa niekedy používa prísnejšie ako názov pre *„dense“* vektory ako *word2vec* a nie ako názov pre *„sparse“* vektory ako *TF-IDF*. Slovo *„embedding“* sa tu používa ako odvodené z jeho matematického významu, ktorý vraví, že toto slovo znamená mapovať z jedného priestoru, alebo štruktúry do inej. Vektory nazývané „sparse“ (riedke) sú vektory, ktoré majú veľa dimenzií, ktoré sa skladajú prevažne z núl. Vektory nazývané *„dense“* (husté) sú vektory, ktorých hodnoty vo vektore sú prevažne nenulové, a teda môžeme povedať, že uchovávajú viac sémantického významu [1].

Pojem *„vektor“* môže byť vysvetlený ako zoznam zoradených čísel. Prevodom slova, alebo textu na vektor sa docieli, že s ním bude vedieť pracovať počítač, ktorý potom môže napríklad vypočítať akú sémantické podobnosť majú dve slová, alebo kusy textu. Počet číselných hodnôt vo vektore hovorí koľko má dimenzií. Keby mal teda vektor 1000

dimenzií, znamená to, že je v danom vektore umiestnených 1000 hodnôt zoradených v určitom poradí. Počet dimenzií vektoru záleží od toho, aká bola využitá technika na prevod textu do vektorovej podoby [6].

2.2 Metódy prevodu textu na vektor

Táto sekcia rozoberá a popisuje rôzne prístupy ku prevodu slov alebo textu do podoby číselných vektorov. Cieľom tejto kapitoly je priblížiť vývoj metód na prevod textu na číselné vektory a spôsob akým tieto metódy fungujú.

2.2.1 Bag of Words

Najjednoduchší spôsob prevádzania textu na číselné vektory sa nazýva *„Bag of Words“* (BoW). Aby sa previedol text na vektor, musí sa najprv rozdeliť celý text na slová, ktoré sa následne upravia do ich základnej formy procesom nazývaným *„stemming“*. Napríklad slovo *„beží“* by bolo upravené na „beh“. Ďalším krokom je spočítať koľkokrát sa každé slovo nachádza v danom texte. Z týchto čísiel sa vytvorí vektor ktorý reprezentuje daný text [7].

Keby sa na vektor prevádzal text *„to be or not to be“*, najprv by sa vybrali všetky jedinečné slová – *„to“*, *„be“*, *„or“*, *„not“*. Následne by sa vytvoril vektor každého slova vo vete reprezentovaného ako jednotky a nuly. Napríklad slovo *„be“* by bol vektor [0, 1, 0, 0], slovo *„or“* by bolo vektor [0, 0, 1, 0], atď. [6].

	“to”	“be”	“or”	“not”
To	1	0	0	0
be	0	1	0	0
or	0	0	1	0
not	0	0	0	1
to	1	0	0	0
be	0	1	0	0
	2	2	1	1

Obr. 1: Vektor vety *„to be or not to be“* pomocou BoW

Keďže sa vektory slov z veľkej časti skladajú z núl, nazývajú sa *„sparse“*. Ich dimenzia je definovaná počtom unikátnych slov v texte. Celú vetu reprezentuje vektor v ktorom je uložený počet výskytov všetkých unikátnych slov v danej vete ako je vidno na obrázku Obr. 1 [6].

Táto metóda je jednoduchá a neberie do úvahy sémantický význam slov. To znamená že vety ako *„Dievča študuje umelú inteligenciu”* a *„Mladá žena sa učí AI a ML”* nebudú blízko pri sebe aj napriek tomu že významovo sú dané vety podobné [7].

2.2.2 TF-IDF

Metóda *„Term Frequency – Inverse Document Frequency“* vylepšuje metódu *„Bag of Words“*. BoW metóda len vyhodnocuje počet opakovania slov v texte, čo nie je veľmi deskriptívny údaj ak je cieľom správne vyhodnotiť slová, ktoré majú v danom texte najväčší sémantický význam. Slová, ktoré sa často vyskytujú pri sebe (ako napríklad koláč a čerešňa) sú dôležitejšie ako slová, ktoré sa objavia len raz alebo dvakrát. Na druhú stranu slová, ktoré sa v texte objavujú príliš často môžu byť nepodstatné. TF-IDF z časti tento problém prekonáva, pomocou *„Term Frequency“* a *„Inverse Document Frequency“* [1]. TF-IDF vytvára vektory každého slova pomocou týchto dvoch metrík ktorými sa vypočíta váha významu daného slova vzhľadom na celú kolekciu dokumentov. [8] Váha významu daného slova sa vypočíta konkrétne vzorcom uvedeným nižšie.

$$\mathrm{TF-IFD}(t, d, D) = \mathrm{TF}(t, d) \times \mathrm{IDF}(t, D)$$

„Term Frequency“ je frekvencia slova v dokumente. Najčastejšie sa vypočíta ako počet opakovaní slova vydelený počtom unikátnych slov v dokumente, no je viac spôsobov ako túto metriku počítať [7].

$$\mathrm{TF}(t, d) = \frac{\text{počet výskytov slova } t \text{ v dokumente } d}{\text{počet unikátnych slov v dokumente } d}$$

Môže sa počítať napríklad aj ako iba počet opakovaní slov. Taktiež sa môže táto hodnota vypočítať ako logaritmus so základom 10 z počtu opakovaní daného slova. Logaritmus sa môže použiť pretože keď sa nejaké slovo objaví stokrát v dokumente, neznamená to, že je stokrát väčšia pravdepodobnosť, že je dané slovo relevantné pre význam daného dokumentu [1].

„Inverse Document Frequency“ je metrika, ktorá pomáha nájsť slová, ktoré sú naprieč dokumentami zriedkavé a teda by mohli držať veľký sémantický význam pre daný text.

Zároveň táto metrika znižuje váhu hodnotám, ktoré sa často opakujú naprieč dokumentami čo sú väčšinou slová s menším sémantickým významom v rámci daného textu. Počíta sa ako logaritmus so základom 10 z pomeru celkového počtu dokumentov a počtu dokumentov, ktoré obsahujú dané slovo [7].

$$\text{IDF}(t, D) = log\ (\frac{\text{počet dokumentov v korpuse } D}{\text{počet dokumentov ktoré obsahujú slovo } t})$$

Vysoká hodnota IDF naznačuje, že sa slovo vyskytuje v menšom počte dokumentov, čo zvýrazňuje hodnotu sémantického významu daného slova. Keď je naopak IDF nízke, značí to, že sa dané slovo vyskytuje v mnohých dokumentoch a teda hodnota sémantického významu daného slova bude nižšia. [7].

Týmto spôsobom je možné dostať vektory kde veľmi často používané slová ako *„a“*, *„v“* a pod. budú mať nízke váhy zatiaľ čo slová ktoré sa vyskytujú častejšie v rámci jedného dokumentu budú mať vyššie váhy. Vďaka tomu sa získajú lepšie výsledky, no tento spôsob stále nedokáže zachytiť sémantický význam ako taký [7].

Dimenzia (veľkosť) vektorov je definovaná počtom všetkých unikátnych slov naprieč všetkými dokumentami v korpuse. To znamená že výstupné vektory budú veľmi neefektívne, pretože drvivá väčšina hodnôt vo vektore bude nulová [7].

2.2.3 Word2Vec

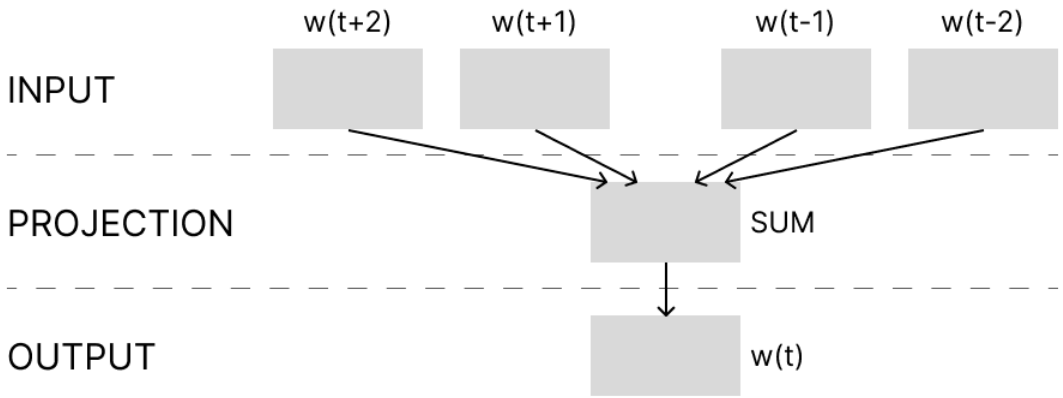
Tento spôsob prevodu textu na vektorovú reprezentáciu bol akousi revolúciou, nakoľko metódy ktoré sú popísané vyššie v tejto práci dokázali vyprodukovať len takzvané *„sparse“* vektory *„Word2Vec“* dokáže produkovať takzvané *„dense“* vektory, čo sú vektory ktoré majú menej dimenzií približne v rozsahu 50-1000. Čísla v týchto vektoroch sú nenulové a môžu byť aj záporné [1].

Tento spôsob bol vyvinutý spoločnosťou Google v roku 2013 a je založený na preddefinovanej dvojvrstvovej neurónovej sieti natrénovanej aby generovala vektorové reprezentácie slov, vďaka ktorým dokáže z textu vziať kontextové a sémantické podobnosti [5].

„Word2Vec“ má dva algoritmy, ktoré sú založené na *„distributional hypothesis“*. Táto hypotéza vraví, že slová ktoré sa vyskytujú v podobnom alebo rovnakom kontexte majú podobný význam. Tieto algoritmy sú založené na umelej neurónovej sieti, ktorá má 3 vrstvy. Táto neurónová sieť dokáže klasifikovať komponenty textu. Slová a ich kontext je

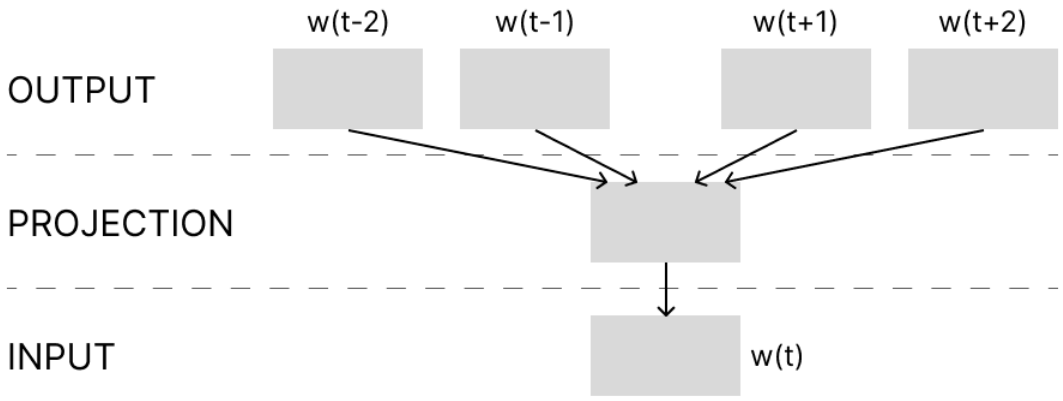
embeddovaný vo *„dense“* vektoroch. Každé slovo má priradený svoj vlastný vektor. Vektory sú potom usporiadané do vektorového priestoru tak, že sa dá vypočítať sémantická podobnosť pomocou kosínovej vzdialenosti. Výsledkom môže byť číslo od -1 do 1 kde čím väčšia hodnota, tým bližší sémantický význam. [8].

Prvý algoritmus sa nazýva *„Continuous Bag od Words“* (CBOW). Táto metóda sa snaží z kontextu okolo slova predpovedať dané slovo. Napríklad ak by sa táto metóda použila na vetu „Algoritmus CBOW je založený na neurónovej sieti.“ a veľkosť kontextového okna by bola nastavená na 2, vznikli by páry ako ([Algoritmus, je], CBOW), ([CBOW, založený], je), ([je, na], založený), ([založený, neurónovej], na) atď. kde prvé dve slová sú kontext a tretie slovo je slovo, ktoré sa algoritmus snaží predpovedať [9].



Obr. 2: Architektúra metódy CBOW

Druhý algoritmus sa nazýva *„Skip-gram“*. Táto metóda je opakom predošlej – snaží sa z jedného slova predpovedať kontext. Ak by sa táto metóda použila na vetu „Algoritmus Skip-gram je založený na neurónovej sieti.“ s veľkosťou kontextového okna nastaveného na 2, v prípade slova *„neurónovej“*, by sa model snažil predpovedať slová *„na“* a *„sieti“* atď. v závislosti od slova. Tento model sa trénuje na dvoch typoch dát – pozitívnych a negatívnych. Pozitívne dáta majú podobu [(slovo, kontext), 1] – jednotkou na konci indikujeme, že ide o relevantný kontext pre dané slovo. Negatívne dáta majú podobu [(slovo, kontext), 0] – v tomto prípade sú slová v kontexte náhodne vybrané a nulou indikujeme, že ide o kontext, ktorý je nerelevantný pre dané slovo. Takéto trénovanie pomáha modelu aby vedel lepšie vyhodnotiť kontextovo relevantné slová a teda vygenerovať podobné embeddingy pre slová s podobným významom [9].



Obr. 3: Architektúra metódy Skip-gram

V prípade, že je dostupným malé množstvo dát, skip-gram model je vhodnejšou voľbou, pretože vie lepšie rozpoznať význam aj menej používaných slov alebo fráz. Na druhej strane trénovanie modelu CBOW je oveľa rýchlejšie a tento model je lepší pre slová, ktoré sa vyskytujú v text často [8].

2.2.4 fastText

„fastText“ je open-source software, ktorý sa dá považovať za pokročilejšiu verziu Word2Vec. Bol vytvorený spoločnosťou Facebook v roku 2016 [8]. Word2Vec považuje každé slovo v korpuse za najmenšiu jednotku, no fastText ide o niečo hlbšie a považuje každé slovo za jednotku zloženú z n-gramov. To znamená, že vygenerované vektory sú založené na sume charakterov jednotlivých n-gramov. Tento prístup je výhodnejší oproti Word2Vec, nakoľko morfologická štruktúra slov často obsahuje dôležité informácie o význame daného slova [5]. V niektorých jazykoch je tento prístup dôležitejší ako v iných, napríklad pre morfologicky bohaté jazyky ako sú nemčina, alebo turečtina prináša tento prístup významné zlepšenia. Ďalšia dôležitá vlastnosť tohto modelu je, že dokáže vygenerovať embeddingy aj pre slová, na ktorých nebol natrénovaný. Word2Vec dokáže generovať len embeddingy pre slová ktoré pozná, teda na ktorých bol daný model trénovaný, no vďaka tomu, že fastText sa pozerá na slová ako na skupiny n-gramov dokáže vygenerovať embeddingy aj pre slová ktoré nepozná [8].

2.2.5 GloVe

„GloVe“ je skratka pre *„Global Vectors“*. Tento model dostal svoje meno, pretože je založený na zachytávaní globálnych štatistík z celého korpusu [1]. Jedna z kritík oboch

architektúr Word2Vec je, že ignorujú rôzny počet výskytov niektorých kontextových slov a taktiež neberú do úvahy globálny kontext, ale len lokálny kontext. GloVe je regresívny model, ktorý sa snaží prekonávať tieto limitácie pomocou kombinácie metód ktoré používa Word2Vec a predošlé jednoduchšie prístupy. GloVe využíva zachytávanie globálnej štatistiky, čo je veľmi efektívne, a kombinuje ju s benefitmi ktoré pramenia z prediktívnych modelov ako Word2Vec. Vďaka tejto elegantnej kombinácii dvoch prístupov sú reprezentácie slov vygenerované modelom GloVe kvalitnejšie ako z predošlých modelov [8].

Spolu výskyt slov v rámci kontextových okien môže obsahovať cenné sémantické informácie. Napríklad slovo „pevný“ má väčšiu pravdepodobnosť, že bude použité pri slove „ľad“ ako pri slove „para“. Na druhú stranu slovo „plynná“ bude pravdepodobne viac použité pri slove „para“ ako pri slove „ľad“. Takéto globálne štatistiky môžu byť vypočítané dopredu a tým sa zvýši rýchlosť trénovanie modelu [10].

Štatistika spolu výskytov sa ukladá do matice spolu výskytov. Primárne ukladá informácie o frekvencii dvoch slov vyskytujúcich sa pri sebe v rámci celého korpusu [11].

Tab. 1: Matica výskytnosti

	X_1	X_2	X_3	X_4
X_1	0	12	18	6
X_2	22	0	22	17
X_3	16	32	0	24
X_4	30	24	10	0

V tabuľke Tab. 1 reprezentujú X_1, X_2, X_3 a X_4 unikátne slová z korpusu. Jednotlivé čísla v matici hovoria ako často sa v korpuse vyskytujú dané dve slová pri sebe. X_{ij} teda reprezentuje počet spolu výskytov slov X_i a X_j. Neurónová sieť teda bude trénovaná smerom k tejto matici. Inak povedané, zo vstupu daného slova sa bude model snažiť predpovedať túto maticu [11].

2.2.6 ELMo

Všetky spôsoby prevodu textu na vektory, ktoré boli doteraz v tejto práci popísané majú spoločnú jednu vec – generujú statické vektory pre každé slovo. Inak povedané, pre každé

slovo v korpuse vytvoria konkrétnu vektorovú reprezentáciu, ktorá sa nezmení ak je rovnaké slovo použité v dvoch rozdielnych kontextoch a to môže byť problematické. *„ELMo“* je skratka pre *„Embeddings from Language Models“*. Tento prístup ku embeddovaniu dokáže vytvoriť lepšie embeddingy, práve vďaka tomu, že je schopný podľa kontextu slova vytvoriť pre dané slovo rôzne embeddingy. Každé slovo v korpuse teda nebude mať jeden statický vektor, ale každé inštancia slova v texte bude mať iný vektor, pretože sa berie do úvahy embeddingu daného slova jeho kontext. Tento algoritmus je taktiež založený na jednotlivých znakoch, čo má za následok, že dokáže úspešne embeddovať aj slová, ktoré sú mimo štandardnú slovnú zásobu [8].

ELMo používa na generovanie kontextovo závislých embeddingov obojsmernú LSTM (*„Long-Short-Term Memory“*) sieť. To znamená, že dané slovo nie je závislé len od predošlých slov, ale berú sa do úvahy aj slová, ktoré sú nasledujúce. Tento spôsob spracovania pomáha modelu lepšie rozumieť významu daného slova v rámci väčšieho kontextu ako keby bola použitá iba jednosmerná LSTM sieť. ELMo taktiež dokáže jednoducho rozpoznať štruktúru slov vďaka tomu, že vstup do modelu je založený na znakoch a nie na celých slovách. To znamená že dokáže postrehnúť rozdiel napríklad medzi slovom „beauty“ a slovom „beautiful“ dokonca aj bez kontextu. Pri testovaní tohto modelu na rôznych úlohách vyšli výsledky všetkých úloh podobne, alebo lepšie ako pri testovaní predošlých metód na rovnakých úlohách [12].

2.2.7 BERT

„BERT“ je skratka pre *„Bidirectional Encoder Representations from Transformers“* a je to model reprezentácie jazyka od spoločnosti Google [13]. Skladá sa z hlbšej neurónovej siete ako ELMo a obsahuje omnoho viac parametrov, vďaka čomu má lepší zmysel pre reprezentáciu textu. BERT dokázal produkovať jedny z najlepších výsledkov a to s potrebou len minimálneho dolaďovania po prvotnom natrénovaní. Taktiež je veľkou výhodou to, že BERT môže byť použitý na mnoho iných úloh v rámci oblasti NLP, nie len na poskytovanie vektorovej reprezentácie slov. BERT poskytol základ pre ďalší rýchly vývoj v podobe modelov ako RoBERTa, Distillbert, alebo Google TransformerXL. Nová architektúra nazývaná *„Transformer“* priniesla nový mechanizmus nazývaný *„Self-attention“* ktorý sa snaží vytvoriť spojenia medzi rôznymi pozíciami v sekvencii, aby lepšie zachytil reprezentáciu danej sekvencie [8].

Architektúra Transformer je postavená na mechanizme nazývanom *„self-attention“*. Tento mechanizmus pomáha vytvoriť spojenia medzi slovami v rámci jednej vety. Napríklad vo vete *„I washed the dishes until they were clean“* by sa zachytilo spojenie slova *„dishes“* a slova *„they“*, pretože slovo *„they“* sa v rámci tejto vety odkazuje na slovo *„dishes“* [14].

Jazykové modely zvyčajne vytvárame trénovaním na nesúvisiacich úlohách, no na takých úlohách, ktoré pomáhajú vyvinúť kontextové chápanie slov v modeli. Častokrát sú to úlohy, ktoré obsahujú predikciu nasledujúceho slova alebo slov v blízkosti pri sebe. Problém je, že takéto trénovanie sa nedá použiť pri obojstranných modeloch, pretože by modelu umožnilo nepriamo vidieť slovo, ktoré má predpovedať. V takejto situácii by model dokázal triviálne predpovedať správne slovo, no aj keby model prebehol kompletným trénovaním, nebolo by zaručené, že sa naučil kontextovo správne sémantické významy slov miesto toho aby sa naučil len optimalizovať triviálne predikcie slov [13].

BERT sa trénuje na dvoch tzv. *„unsupervised“* (to znamená, že model nepotrebuje explicitne zadané správne odpovede) úlohách. Prvá úloha sa volá *„Masked Language Model“* (MLM). V tejto úlohe model dostane vstup, v ktorom je nejaké percento slov (tokenov) náhodne nahradených maskou a model sa snaží predpovedať tieto zamaskované slová (tokeny) na základe zvyšného kontextu vstupu. Pre lepšie fungovanie s dolaďovaním pred trénovaného modelu sa 80 percent náhodne vybraných slov (tokenov) na zamaskovanie zmení na masku, 10 percent sa zmení na náhodné slová (tokeny) a zvyšných 10 percent sa vybrané slovo nezamaskuje vôbec. Druhá úloha sa volá „Next Sentence Prediction“ (NSP) kde je model trénovaný na sérii dvojíc viet, kde v 50 percentách prípadov druhá veta správne nasleduje prvú vetu a v 50 percentách prípadov je druhá veta vybraná náhodne z korpusu. Táto úloha sa snaží model trénovať aby správne rozumel vzťahom medzi dvomi vetami, čo je dôležitá vlastnosť v mnohých NLP úlohách ako napríklad odpovedanie na otázky [13].

2.3 Hodnotenie sémantickej podobnosti

Táto kapitola rozoberá možnosti hodnotenia sémantickej podobnosti dvoch textov. Keďže výsledkom embeddingu sú vektory, môžeme vypočítať vzdialenosť v priestore medzi danými dvomi vektormi. Čím bude menšia vzdialenosť medzi dvomi vektormi tým bude väčšia ich sémantická podobnosť [7].

2.3.1 Euklidovská vzdialenosť

Euklidovská vzdialenosť, tiež známa ako L2 je najviac štandardná metóda na výpočet vzdialenosti dvoch vektorov. Počíta priamu vzdialenosť dvoch bodov v priestore. Je najviac využívaná v bežnom živote, napríklad na výpočet vzdialenosti dvoch miest vzdušnou čiarou [7].

$$\mathrm{L2} = \sqrt{\sum_{i=1}^{n}(x_i - y_i)^2}$$

2.3.2 Manhattanská vzdialenosť

Ďalšia bežne používaná metrika vzdialenosti je Manhattanská vzdialenosť, tiež známa ako L1. Táto metóda bola nazvaná podľa ostrovu Manhattan, ktorý má svoje ulice usporiadané do štvorcov. Najkratšia cesta medzi dvomi bodmi na tomto ostrove by sa dala vypočítať práve pomocou Manhattanskej vzdialenosti [7].

$$\mathrm{L1} = \sum_{i=1}^{n}|x_i - y_i|$$

2.3.3 Skalárny súčin

Ďalšou možnosťou na výpočet vzdialenosti dvoch vektorov je vypočítať ich skalárny súčin. Táto metrika sa trochu zložito interpretuje. Ukazuje, či dané dva vektory smerujú do rovnakého smeru, no výsledky sú z veľkej časti závislé od veľkosti vektorov [7]. Ak by sme porovnávali vektory smerujúce do opačného smeru, v prípade, že by vektory neboli rovnaké, výsledkom by bolo záporné číslo. V prípade, že sú vektory orientované do podobného smeru, výsledkom by bolo kladné číslo. Čím menšia vzdialenosť (teda výsledok bližší nule) medzi dvomi vektormi, tým sú dané vektory sémanticky podobnejšie [15].

$$\vec{x} \cdot \vec{y} = \sum_{i=1}^{n} x_i \times y_i$$

2.3.4 Kosínová podobnosť

Kosínová podobnosť je používaná pomerne často. Táto metrika je skalárny súčin normalizovaný podľa veľkostí vektorov. To odstráni komplikácie samotného skalárneho

súčinu [7]. Táto metrika meria uhol medzi smerom ktorým ukazujú dva vektory v mnoho-dimenzionálnom priestore. Stojí na domnienke, že vektory s podobným sémantickým významom budú smerovať podobným smerom. To je výhodné, pretože ak sú dva vektory podľa Euklidovskej vzdialenosti ďaleko od seba, je stále možné, že budú smerovať podobným smerom. Napríklad ak sa slovo *„ovocie“* vyskytne v jednom dokumente 30-krát a v druhom dokumente sa rovnaké slovo vyskytne 10-krát, je to jednoznačný rozdiel vo veľkosti, no tieto dokumenty stále môžu mať podobný sémantický význam ak sa vezme do úvahy iba uhol medzi vektormi reprezentujúcimi dané dokumenty. Čím menší je uhol medzi nimi, tým je podobnejší sémantický význam [15].

$$\text{Kosínová podobnosť} = \frac{\vec{x} \cdot \vec{y}}{||\vec{x}|| \times ||\vec{y}||}$$

2.4 Hodnotenie korelácie medzi dvomi číslami

Na vyhodnotenie výsledkov v experimente spracovanom v praktickej časti, je potrebný spôsob hodnotenia korelácie medzi dvomi číslami. Na toto vyhodnotenie je použitý Pearsonov korelačný koeficient. Používa sa na odmeranie sily korelácie medzi dvomi listami hodnôt [16].

$$r_{xy} = \frac{\sum_{i=1}^{n}(x_i - \bar{x})(y_i - \bar{y})}{\sqrt{\sum_{i=1}^{n}(x_i - \bar{x})^2}\sqrt{\sum_{i=1}^{n}(y_i - \bar{y})^2}}$$

Výsledkom tohto výpočtu je skóre sily korelácie, s hodnotami od -1 do 1, kde -1 znamená úplná korelácia, ktorá je nepriamo úmerná, 0 znamená žiadna korelácia a 1 znamená úplná korelácia, ktorá je priamo úmerná [16].

II. PRAKTICKÁ ČÁST

3 ZHOTOVENIE TESTOVACIEHO DATASETU

Táto kapitola sa zaoberá procesom výberu, popisu, spracovania a rozvinutia datasetu. Popisuje dataset, ktorý bol pre túto prácu vybraný, popisuje tiež ako bol tento dataset upravený, aby bol vhodnejší pre použitie v tejto práci a taktiež popisuje aplikáciu, ktorá bola vytvorená pre zjednodušenie daných úprav použitého datasetu.

3.1 Výber dát a popis ich charakteristík

Dataset použitý v tejto práci je z github repozitáru *„STS: Semantic Textual Similarity“* používateľa *„nttuyenx“* [17]. Dataset v danom github repozitáre obsahuje výber dát datasetov z ročníkov 2012-2016 akcie SemEval [18]. Tento dataset je vo formáte CSV, má viac ako 5500 riadkov a obsahuje všetky potrebné dáta k relevantnému testu na vyhodnotenie kvality embeddingov vytvorených z rôznych súborových formátov. V tejto práci budú použité dáta z troch stĺpcov v tomto datasete. Prvý stĺpec, ktorý bude použitý obsahuje číslo hodnotiace sémantickú podobnosť dvoch viet, ktoré sa nachádzajú v druhom a treťom stĺpci použitom v tejto práci. Číslo, ktoré hodnotí sémantickú podobnosť daných dvoch viet je stanovené na škále od 0 (žiadna sémantická podobnosť) do 5 (vety sú sémanticky identické). Tieto hodnotenia boli určené ľuďmi [18], preto sa môžu považovať za zlatý štandard, s ktorým sa potom bude porovnávať hodnota sémantickej podobnosti, ktorá sa vypočíta vybranou metódou.

3.2 Spracovanie a príprava datasetu

Dataset o ktorom sa píše v predošlej kapitole je pevný základ pre porovnávanie kvality embeddingov súborov rôznych formátov. Táto práca sa ale zameriava na kvalitu embeddingov pre veľké jazykové modely. V snahe porovnať výsledky embeddingov s menej syntetickými dátami je potrebné dataset pre použitie v tejto práci trochu upraviť a to tak, že jedna z viet bude rozšírená do paragrafu, ktorý bude sémanticky podobný základnej vete a bude v sebe obsahovať nepozmenenú originálnu vetu. Keďže sú v datasete dve vety s ľuďmi určenými hodnotami ich sémantickej podobnosti, bolo by nevhodné vo veľkej miere upravovať sémantický význam daných viet. Pri takto rozsiahlom datasete nepripadá do úvahy rozširovať každú vetu do paragrafu ručne človekom. Preto bol na rozšírenie datasetu použitý jazykový model *„GPT-3.5 Turbo“*. Tento jazykový model je vo väčšine prípadov dostatočne dobrý na pochopenie kontextu vety a na základe danej vety vygenerovať paragraf s kontextom, ktorý je veľmi podobný danej vete [19]. Využitie

jazykového modelu na rozšírenie dát je výrazne rýchlejšie ako rozširovanie dát ručne. Pôvodná veta je ponechaná v paragrafe v originálnom znení, takže vygenerovaný paragraf do istej miery potláča syntetickosť dát. Tým budú dosiahnuté výsledky bližšie realite, ako keby boli použité syntetické dáta ako len dve vety. Bohužiaľ jazykový model *„GPT-3.5 Turbo“* nie je dokonalý [19] a teda sa môže stať že do datasetu zanesie v nízkej miere šum v podobe nesprávne identifikovaného kontextu, alebo náhodného znaku, ktorý sa nehodí do danej vety. Manuálnou úpravou by bolo možné tento šum odstrániť, no znovu by bola problémom časová náročnosť. No šum nemusí jednoznačne znamenať niečo zlé. V realite sú dáta málokedy dokonalé, môžeme preto nad šumom pridaným jazykovým modelom v istom zmysle uvažovať ako nad viac realite zodpovedajúcimi dátami [20]. Niektoré vety nevedel jazykový model spracovať spoľahlivo, preto bolo zopár riadkov z rozšíreného datasetu manuálne zmazaných, aby sa v datasete nenachádzali žiadne jednoznačne chybné dáta. Nový dataset sa skladá zo štyroch stĺpcov. V prvom stĺpci sa nachádza hodnota podobnosti pôvodných dvoch viet. V druhom stĺpci sa nachádza prvá veta z dvojice v nezmenenom znení. V treťom stĺpci sa nachádza nadpis pre paragraf vygenerovaný jazykovým modelom spomínaným vyššie. Nadpis pre paragraf je vygenerovaný tiež a bol pridaný do datasetu z dôvodu priblíženia sa lepšej reprezentácie reálnych dát. V štvrtom, teda poslednom, stĺpci sa nachádza vyššie spomínaný paragraf textu vygenerovaný jazykovým modelom, ktorý v sebe obsahuje druhú vetu z dvojice v pôvodnom znení, alebo ako súčasť dlhšej vety, no tiež v pôvodnom znení.

3.2.1 Aplikácia na spracovanie datasetu

Pre rýchlejšie a efektívnejšie spracovania datasetu bola v rámci tejto práce vytvorená malá aplikácia bez grafického rozhrania, ktorá rozšíri dataset potrebný pre túto prácu. Aplikácia je napísaná v jazyku Python.

3.2.1.1 Použité knižnice

Zdroj. kód 1: Knižnice použité v aplikácii na spracovanie datasetu

```
1  import sys
2  import os
3  import csv
4  import json
5  from openai import OpenAI
```

- *sys*

  Táto knižnica je súčasťou štandardnej knižnice jazyka Python [21]. V tejto aplikácii

je použitá na získanie informácií z príkazového riadku, odkiaľ sa aplikácia spúšťa. Taktiež je použitá na skontrolovanie správneho počtu argumentov zadaných v príkazovom riadku pri spustení aplikácie.

- *os*

  Táto knižnica poskytuje možnosť používať funkcionality operačného systému [22]. V tejto aplikácii bola táto knižnica použitá na kontrolovanie existencie súborov a na manipuláciu s cestami.

- *csv*

  Táto knižnica poskytuje funkcionalitu na prácu so súbormi formátu CSV [23]. V tejto aplikácii bola táto knižnica použitá na čítanie datasetu, ktorý je vo formáte CSV a na zapisovanie nového, rozšíreného, datasetu, ktorý je uložený taktiež vo formáte CSV.

- *json*

  Táto knižnica poskytuje funkcionalitu na prevod formátu JSON na Python slovník. Taktiež poskytuje funkcionalitu na zapisovanie a čítanie súborov formátu JSON [24]. V tejto aplikácii je táto knižnica využitá na zapisovanie dát o spotrebe tokenov pri používaní openai API do súboru formátu JSON.

- *openai*

  Táto knižnica je oficiálnou Python knižnicou pre openai API [25]. Táto knižnica bola použitá na posielanie žiadostí na openai API, pre rozšírenie jednej vety do paragrafu s nadpisom.

3.2.1.2 Konštanty

Nasledujúci zdrojový kód má viackrát rovnaké číslo riadku, čo sa môže zdať ako chyba, no je to tak spravené z dôvodu, aby bolo jasne vidieť ktoré riadky sú zalomené kvôli maximálnej šírke tohto dokumentu. Všetky riadky s rovnakým číslom pod sebou znamenajú, že všetky tieto riadky boli v originálnom zdrojovom kóde len jeden riadok.

Zdroj. kód 2: Konštanty v aplikácii na spracovanie datasetu

```
8  PROMPT = """
9  You will generate a paragraph that will contain a standalone sentence
9  "{sentence}" in the middle with two short sentences before it and two
9  short sentences after it.
10 The sentences you generate should provide context and should highlight
10 the main sentence.
11 Your response should be a JSON with the following structure:
```

```
{{
    'title': Title for the paragraph comes here,
    'text': The paragraph comes here,
}}
"""
DATASET_OUTPUT_PATH = 'transformed_dataset/dataset.csv'
OPENAI_MODEL = 'gpt-3.5-turbo-0125'
```

Konštanta *PROMPT* obsahuje predlohu textu, ktorý bude poslaný ako dotaz modelu, ktorý je definovaný v konštante *OPENAI_MODEL*. V konštante *DATASET_OUTPUT_PATH* je uložená cesta kam sa má zapísať upravený dataset.

3.2.1.3 Funkcie

- *process_csv_file*

Zdroj. kód 3: funkcia *process_csv_file*

```
def process_csv_file(file_path):
    batch_size = 10
    rows_batch = []

    with open(file_path, 'r', encoding='utf-8') as file:
        client = OpenAI()
        for i, line in enumerate(file):
            row = line.strip().split('\t')

            print(i)
            title, text = expand_sentence(row[6], client)

            rows_batch.append([row[4], row[5], title, text])

            if (len(rows_batch) == batch_size):
                print(rows_batch)
                append_rows_to_csv(DATASET_OUTPUT_PATH, rows_batch)
                rows_batch = []
                print(f"Saved {batch_size} rows to file.")

    print(f"Saved {len(rows_batch)} rows to file.")
    append_rows_to_csv(DATASET_OUTPUT_PATH, rows_batch)
```

Táto funkcia prijíma cestu k CSV súboru. Vytvorí objekt pre prácu s openai knižnicou, aby sa tento objekt nemusel inicializovať každú iteráciu. Súbor otvorí a číta ho riadok po riadku. Každý riadok rozdelí podľa znaku tabulátoru, vyberie druhú vetu z dvojice viet a tú nechá rozšíriť na nadpis a paragraf. Keď je veta rozšírená, uloží si do bufferu riadok pre nový dataset, teda skóre sémantickej

podobnosti, prvá veta, nadpis pre paragraf a paragraf. Sleduje počet viet v bufferi a keď dosiahne definovanej hodnoty, zapíše všetky riadky v bufferi do súboru pomocou funkcie *append_rows_to_csv*, aby program nestratil všetky dáta v prípade neočakávaného ukončenia. Táto funkcia taktiež vytlačí pri spracovaní riadku príkazového riadku na ktorej iterácii sa teraz nachádza aby používateľ videl rýchlosť spracovania.

- *is_csv_file*

Zdroj. kód 4: Funkcia *is_csv_file*

```
72 def is_csv_file(file_path):
73     _, file_extension = os.path.splitext(file_path)
74     return file_extension.lower() == ".csv"
```

Táto funkcia prijíma cestu k súboru. Ak je súbor typu CSV, vráti hodnotu *True* a ak typu CSV nie je tak vráti hodnotu *False*. Používa sa pri kontrole argumentov poskytnutých z príkazového riadku.

- *expand_sentence*

Zdroj. kód 5: Funkcia *expand_sentence*

```
50 def expand_sentence(sentence, openai_client):
51     response = openai_client.chat.completions.create(
52         model=OPENAI_MODEL,
53         temperature=0.12,
54         response_format={'type': 'json_object'},
55         messages=[
56             {
57                 'role': 'user',
58                 'content': PROMPT.format(sentence=sentence)
59             },
60         ]
61     )
62 
63     token_usage_logger(response.usage.prompt_tokens,
64                        response.usage.completion_tokens)
65     if response.choices[0].finish_reason == "content_filter":
66         return "Title", "CONTENT_FILTER"
67 
68     response_dict = json.loads(response.choices[0].message.content)
69     return response_dict['title'], response_dict['text']
```

Táto funkcia prijíma vetu na rozšírenie a klienta na použitie openai knižnice, ktorý bol vytvorený vo funkcii *process_csv_file*. Funkcia pošle požiadavku na API

spoločnosti OpenAI pomocou openai knižnice a spracuje výsledok. Pomocou funkcie *token_usage_logger* zapíše do JSON súboru koľko tokenov bolo použitých pri spracovaní požiadavky. Skontroluje, či nebolo generovanie rozšírenia vety prerušené z dôvodu filtrácie obsahu na strane API, v prípade, že bolo, nahradí názov a paragraf za definované hodnoty, aby bolo jednoduché tieto hodnoty nájsť a zmazať vo výslednom datasete. Nakoniec využije JSON knižnicu na získanie hodnôt paragrafu a jeho názvu a tie z funkcie vráti.

- *append_rows_to_csv*

Zdroj. kód 6: Funkcia *append_rows_to_csv*

```
38 def append_rows_to_csv(file_path, rows):
39     if not os.path.exists(file_path):
40         with open(file_path, 'w', newline='') as file:
41             writer = csv.writer(file)
42             writer.writerows(rows)
43         return
44 
45     with open(file_path, 'a', newline='') as file:
46         writer = csv.writer(file)
47         writer.writerows(rows)
```

Táto funkcia prijíma cestu k súboru a riadky, ktoré treba zapísať do daného súboru. Najprv funkcia skontroluje, či súbor v danej ceste existuje a ak nie, vytvorí ho. Následne pripíše do daného súboru dané riadky.

- *token_usage_logger*

Zdroj. kód 7: Funkcia *token_usage_logger*

```
20 def token_usage_logger(prompt_tokens, completion_tokens):
21     usage_file = 'usage.json'
22 
23     if not os.path.exists(usage_file):
24         with open(usage_file, 'w') as file:
25             data = {'prompt_tokens': 0, 'completion_tokens': 0}
26             json.dump(data, file)
27 
28     with open(usage_file, 'r') as file:
29         data = json.load(file)
30 
31     data['prompt_tokens'] += prompt_tokens
32     data['completion_tokens'] += completion_tokens
33 
34     with open(usage_file, 'w') as file:
```

```
35          json.dump(data, file)
```

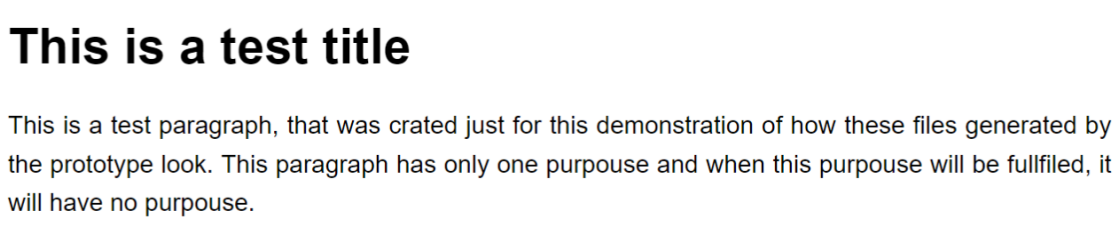
Táto funkcia prijíma dve hodnoty – počet tokenov v prompte a počet vygenerovaných tokenov. Má na starosti zapisovanie použitých tokenov do JSON súboru pre jednoduché sledovanie počtu použitých tokenov.

3.3 Popis formátov súborov

V tejto kapitole sú popísané spôsoby vytvorenia súborov a štruktúry týchto súborov pre použitie na embeddovanie. Každý súbor bude obsahovať nadpis a paragraf. Tieto dáta boli vygenerované pomocou aplikácie popísanej v predošlej kapitole. Generovanie súborov je súčasťou prototypu hlavnej webovej aplikácie pre túto prácu.

3.3.1 PDF

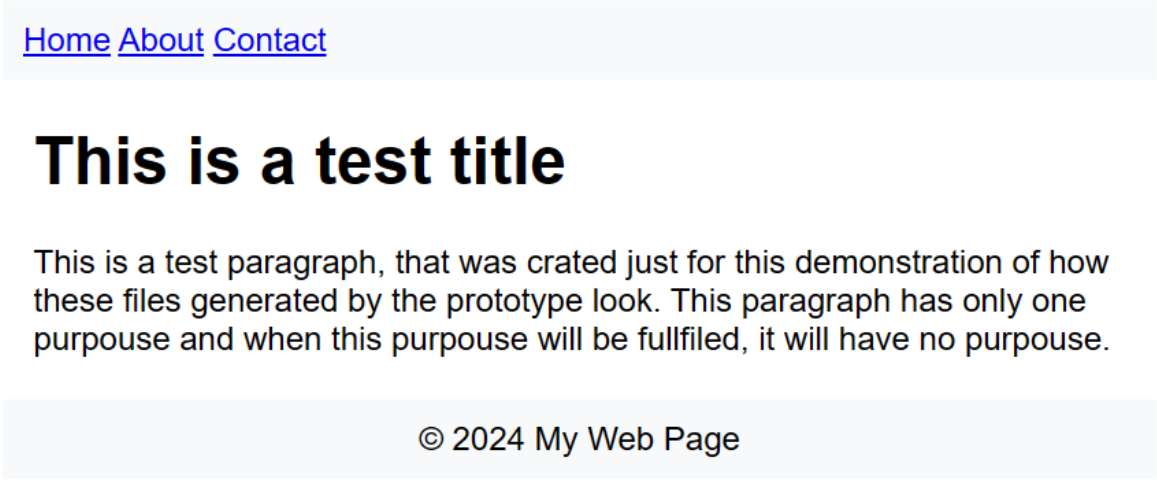
Na vytvorenie súboru PDF, ktorý obsahuje jeden nadpis a jeden paragraf bola použitá Python knižnica nazývaná *„FPDF“*. Táto knižnica podporuje jednoduché vytvorenie PDF súboru a naformátovanie obsahu tohto súboru [26]. Bola použitá na vytvorenie PDF súboru, ktorý má vždy jednu stranu, na ktorej sa nachádza nadpis a paragraf. Nadpis aj paragraf používajú rovnaký font písma – *„Arial“*. Nadpis má veľkosť písma 24 a je hrubým písmom. Pod nadpisom je pridaná medzera a pod medzerou sa nachádza paragraf. Paragraf má veľkosť písma nastavenú na 12. Vygenerovaný súbor sa vždy volá *„data.pdf“*.



Obr. 4: Vygenerovaný PDF súbor

3.3.2 HTML

Na vytvorenie súboru HTML nebola použitá žiadna špeciálna knižnica. Používa sa tu jednoduchá predloha HTML stránky ktorá obsahuje základné tagy ako *head*, *title*, *style*, *nav*, *main*, *footer* a iné. V tagu *main* sa nachádzajú tagy *h1* a *p*. V týchto tagoch sa nachádza nadpis a paragraf. Nadpis paragrafu sa taktiež nachádza v tagu *title*. Vygenerovaný súbor sa vždy volá *„data.html“*.



Obr. 5 Vygenerovaný HTML súbor

3.3.3 CSV

Na vytvorenie súboru CSV bola použitá knižnica csv. Táto knižnica poskytuje zjednodušenie práce so súbormi CSV [23]. CSV súbor má na prvom riadku vždy názvy stĺpcov. Prvý stĺpec má názov *„title“* a druhý stĺpec má názov *„text“*. Po tom ako sa vytvorí tento riadok s názvami, pridá sa druhý riadok, ktorý obsahuje nadpis a paragraf. Ako rozdeľovací znak bola použitá čiarka *„,“*. Vygenerovaný súbor sa vždy volá *„data.csv“*.

Zdroj. kód 8: Obsah vygenerovaného CSV súboru

```
1  title,text
2  This is a test title,"This is a test paragraph, that was crated just for
2  this demonstration of how these files generated by the prototype look.
2  This paragraph has only one purpouse and when this purpouse will be
2  fullfiled, it will have no purpouse."
```

3.3.4 TXT

Vytvorenie tohto súboru nepotrebovalo žiadne špeciálne knižnice. Do súboru sa dáta zapíšu do jedného riadku v jednoduchej štruktúre: *„Nadpis - paragraf“*. Vygenerovaný súbor sa vždy volá *„data.txt“*.

Zdroj. kód 9: Obsah vygenerovaného TXT súboru

```
1  This is a test title - This is a test paragraph, that was crated just
1  for this demonstration of how these files generated by the prototype
1  look. This paragraph has only one purpouse and when this purpouse will
1  be fullfiled, it will have no purpouse.
```

4 PROTOTYP WEBOVEJ APLIKÁCIE

Táto kapitola popisuje prototyp webovej aplikácie, ktorá vznikla v rámci tejto práce. Od výberu technológií, knižníc a nástrojov na vývoj až po konkrétne implementácie častí kódu. Taktiež sú v tejto kapitole popísané dôvody, prečo boli dané technológie, knižnice a nástroje vybrané.

4.1 Technológie

Na implementáciu prototypu webovej aplikácie bol použitý programovací jazyk Python. Toto rozhodnutie bolo spravené kvôli tomu, že Python podporuje rozsiahly výber balíčkov (knižníc), nakoľko má veľkú komunitu vývojárov. Vďaka tomuto faktu nebol žiadny problém nájsť všetky potrebné balíčky pre túto prácu. Ďalším dôvodom na výber jazyka Python je jeho jednoduchosť a rýchlosť práce s ním, vďaka čomu sa dajú prototypy v tomto jazyku implementovať pomerne rýchlo, nakoľko má jednoduchšiu syntax ako mnoho iných programovacích jazykov [27].

4.1.1 Použité knižnice

Táto kapitola sa zameriava na popis všetkých knižníc, ktoré boli v rámci prototypu webovej aplikácie využité. Má za úlohu vysvetliť prečo a ako boli dané knižnice použité.

4.1.1.1 streamlit

Vďaka tejto knižnici bolo vyvíjanie grafického prostredia webového prototypu veľmi jednoduché. Táto knižnica podporuje budovanie web aplikácii priamo v jazyku Python, bez separátnych súborov ktoré by sa museli vyvíjať len pre grafické prostredie [28]. To je dôvod prečo bola táto knižnica vybraná v tejto práci.

4.1.1.2 csv

Knižnica je popísaná v kapitole 3.2.1.1.

4.1.1.3 json

Knižnica je popísaná v kapitole 3.2.1.1.

4.1.1.4 langchain_community

Táto knižnica obsahuje integrácie tretích strán, ktoré implementujú základné triedy definované v LangChain Core knižnici [29]. Táto knižnica sa v prototype používa na

načítanie potrebných informácií z vygenerovaných súborov formátov PDF, HTML, CSV a TXT.

4.1.1.5 langchain_core

Táto knižnica obsahuje základné abstrakcie na ktorých je postavený zvyšok ekosystému LangChain knižníc. Tieto abstrakcie sú navrhnuté aby boli čo najviac modulárne [30]. V prototype sa knižnica používa pri načítaní textu z TXT súboru, kvôli štandardizácii návratových hodnôt zo všetkých funkcií v triede.

4.1.1.6 langchain_openai

Táto knižnica integruje OpenAI do LangChain prostredia [31]. V prototype bola táto knižnica použitá na generovanie embeddingov súborov za pomoci OpenAI API.

4.1.1.7 os

Knižnica je popísaná v kapitole 3.2.1.1.

4.1.1.8 copy

Táto knižnica poskytuje možnosť plytkého a hlbokého kopírovania objektov v jazyku Python. Niekedy je túto knižnicu potrebné použiť, keďže Python pri priraďovaní nevytvára kópie, ale len odkazy na objekt [32]. V prototype bola táto knižnica použitá na zjednodušenie a skrátenie kódu v miestach, kde bolo vhodné ju použiť.

4.1.1.9 numpy

Táto knižnica poskytuje podporu pre multifunkčné objekty n-dimenzionálnych polí, rôzne pomocné funkcie s lineárnou algebrou, náhodnými číslami a iné [33]. V prototype bola použitá pri výpočtoch sémantickej podobnosti.

4.1.1.10 fpdf

Táto knižnica poskytuje rozhranie na prácu PDF súbormi prostredníctvom jazyka Python [26]. V prototype bola táto knižnica použitá na generovanie PDF súboru, ktorý bol následne embeddovaný.

4.1.1.11 time

Táto knižnica poskytuje rôzne funkcie spojené s časom [34]. V prototype bola táto knižnica využitá na presné meranie časovej náročnosti embedovania jednotlivých súborových formátov.

4.1.1.12 tracemalloc

Táto knižnica poskytuje nástroje na stopovanie pamäťových blokov, ktoré boli alokované Pythonom [35]. V prototype bola táto knižnica použitá na meranie pamäťovej náročnosti embeddovania jednotlivých súborových formátov.

4.1.1.13 pandas

Táto knižnica poskytuje rýchle, ohybné a expresívne dátové štruktúry, navrhnuté tak, aby boli kompatibilné s relačnými alebo označkovanými dátami a aby práca s nimi bola jednoduchá a intuitívna [36]. V prototype bola táto knižnica použitá na uloženie dát kompatibilným spôsobom, aby knižnica streamlit mohla z týchto dát vykresliť graf.

4.1.2 Nástroje

Táto kapitola sa venuje popisu nástrojom ktoré boli použité pri vytváraní prototypu popisovanej aplikácie. Cieľom tejto kapitoly je vysvetliť prečo a ako boli dané nástroje použité.

4.1.2.1 Visual Studio Code

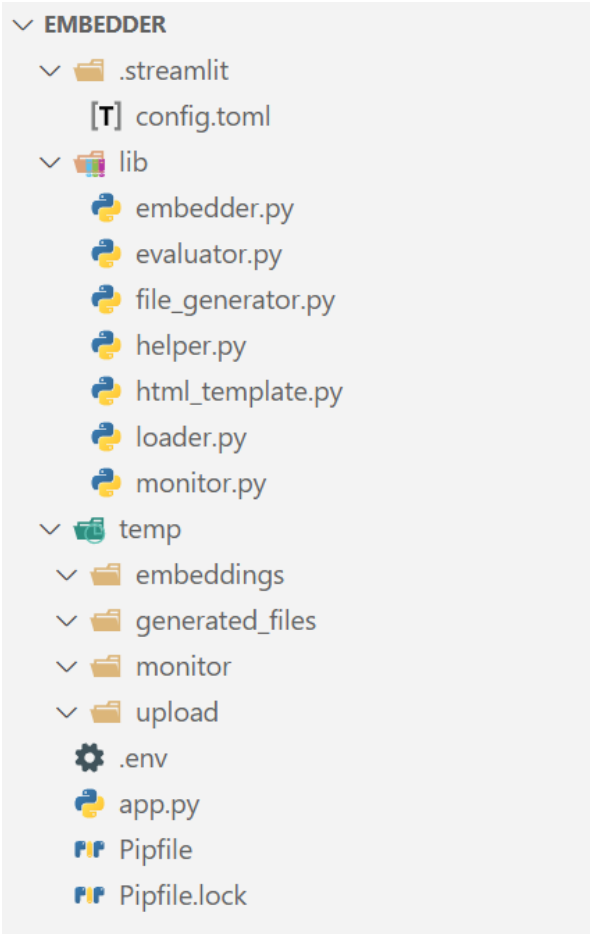
Visual Studio Code je výkonný editor zdrojového kódu. Bol vybraný na vývoj prototypu kvôli jeho širokej podpore, veľkej komunite, ktorá v prípade problémov môže pomôcť, kvôli vysvecovaniu zdrojového kódu a inteligentnému dokončovaniu kódu. Taktiež poskytuje vstavané nástroje pre ladenie kódu, čo umožňuje pri vyvíjaní zdrojového kódu efektívne identifikovať a opraviť chyby v kóde. Nakoniec bol tento editor vybraný kvôli jeho rozšíriteľnosti. Má mnoho rozšírení, ktorými sa dá editor v mnohých smeroch upraviť a teda si ho každý vývojár môže upraviť presne pre svoje potreby [37].

4.1.2.2 Pipenv

Pipenv je nástroj na správu virtuálnych prostredí pre Python. Tento nástroj automaticky vytvorí a spravuje virtuálne prostredie pre projekty v ktorých je použitý. Taktiež vygeneruje dva súbory, ktoré obsahujú zoznam balíčkov, ktoré boli nainštalované v danom

projekte a teda poskytuje jednoduchý spôsob ako oddeliť požiadavky balíčkov pre jednotlivé projekty. Taktiež tým poskytuje jednoduchý spôsob ako získať potrebné balíčky v prípade, že je potrebné daný projekt spustiť na inom prostredí [38]. Tento nástroj bol použitý práve kvôli tomu, že poskytuje jednoduchý spôsob ako zhrnúť všetky balíčkové požiadavky projektu na jedno miesto.

4.2 Štruktúra



Obr. 6: Adresárová štruktúra prototypu webovej aplikácie

Adresár, v ktorom sa celý prototyp nachádza obsahuje adresár *lib*, *temp* a súbory *.env*, *app.py*, *Pipfile* a *Pipfile.lock*. Prototyp sa spúšťa pomocou súboru *app.py*, v ktorom je vďaka knižnici *streamlit* vytvorené jednoduché užívateľské prostredie a v tom istom súbore je aj zdrojový kód na spracovanie všetkých interakcií s prvkami, ktoré sú interaktívne. V zložke *.streamlit* je súbor *config.toml*, ktorý obsahuje nastavenie maximálnej veľkosti súboru, ktorý je možné nahrať do prototypu. V súbore *.env* je uložený kľúč na OpenAI API. Súbory *Pipfile* obsahujú balíčky potrebné pre fungovanie tohto prototypu.

V zložke *lib* sa nachádza hlbšia logika, ktorá bola oddelená zo súboru *app.py* kvôli prehľadnejšiemu zdrojovému kódu. Táto hlbšia logika je rozdelená do viacerých knižníc v tejto zložke (funkcionalita týchto knižníc je popísaná v nasledujúcej kapitole).

V zložke *temp* sa nachádzajú ďalšie 4 zložky – *embeddings, generated_files, monitor* a *upload*. Tieto zložky slúžia ku dočasnému uchovávaniu dát potrebných pre beh prototypu. Do zložky *upload* sa uloží súbor, ktorý používateľ nahrá do prototypu, aby k nemu mal prototyp prístup keď bude s daným súborom pracovať. V zložke *generated_files* sa budú počas embeddovania postupne vymieňať súbory, vygenerované z každého riadku poskytnutého datasetu. Z týchto súborov sa budú následne generovať embeddingy. V zložke *embeddings* bude po ukončení embeddovania vygenerovaný JSON súbor s vygenerovanými embeddingmi a inými dátami potrebnými na vyhodnotenie výsledkov experimentu. V zložke *monitor* sa bude po skončení embeddovania nachádzať JSON súbor, v ktorom budú údaje o časovej a pamäťovej náročnosti procesu embeddovania.

4.3 Funkcionalita

Táto kapitola popisuje a vysvetľuje všetku funkcionalitu naprogramovanú v prototype aplikácie. Prechádza každý súbor, ktorý aplikácia obsahuje a popisuje zdrojový kód, ktorý sa v nich nachádza. Cieľom je priblížiť, ako prototyp funguje vo svojom jadre.

4.3.1 helper.py

Tento súbor obsahuje pomocné funkcie, ktoré logicky nespadajú priamo pod žiadnu z knižníc.

4.3.1.1 save_uploaded_file

Zdroj. kód 10: Funkcia *save_uploaded_file*

```
5  def save_uploaded_file(file, output_dir_path):
6      local_path = os.path.join(output_dir_path, file.name)
7      with open(local_path, "wb") as f:
8          f.write(file.getbuffer())
9      return local_path
```

Táto funkcia slúži na uloženie súboru, ktorý používateľ nahrá do prototypu webovej aplikácie. Funkcia uloží súbor do zložky, ktorá je špecifikovaná a následne vráti cestu k uloženému súboru.

4.3.1.2 get_csv_row_count

Zdroj. kód 11: Funkcia *get_csv_row_count*

```
12 def get_csv_row_count(file_path, has_header=False):
13     with open(file_path, "r", encoding="utf-8") as file:
14         csv_reader = csv.reader(file)
15         row_count = len(list(csv_reader))
16         return row_count - 1 if has_header else row_count
```

Táto funkcia slúži na získanie počtu riadkov v súbore formátu CSV. Funkcia otvorí súbor v danej ceste a podľa špecifikácie vráti počet riadkov buď s odpočítaným riadkom na ktorom sú názvy stĺpcom, alebo bez odpočítania tohto riadku.

4.3.2 app.py

V tomto súbore nájdeme celú časť aplikácie, ktorá je priamo dostupná užívateľovi cez webové rozhranie. Vďaka knižnici *streamlit* je v jednom súbore obsah stránky a zároveň zdrojový kód zodpovedný za spracovanie interakcií so stránkou. Tento súbor nemá jednotlivé funkcie, no sú tu dve časti zdrojového kódu, ktoré by mali byť ukázané.

4.3.2.1 Spracovanie embeddovania

Zdroj. kód 12: Časť aplikácie zodpovedná za spracovanie kliknutia na tlačidlo, ktoré spustí generovanie embeddingov

```
34 if generate_embeddings:
35     if uploaded_dataset is None:
36         first_section_container.write(
37             "You have to upload a csv dataset first."
38         )
39         st.stop()
40 
41     uploaded_data_file_path = save_uploaded_file(
42         uploaded_dataset, UPLOAD_FOLDER_PATH
43     )
44     Embedder.create_embeddings_json(uploaded_data_file_path)
45     os.remove(uploaded_data_file_path)
46 
47     first_section_container.info(
48         f'Embeddings saved to:
48 "{os.path.abspath(Embedder.EMBEDDINGS_OUTPUT_PATH)}"'
49     )
```

Tento zdrojový kód ukazuje ako sa aplikácia správa, keď používateľ klikne na tlačidlo na generovanie embeddingov. Aplikácia v prvom rade skontroluje, či je vo formulári nahraný

CSV súbor. Ak je vo formulári nahraný CSV súbor, aplikácia tento súbor uloží lokálne do zložky definovanej v konštante *UPLOAD_FOLDER_PATH*. Následne sa zavolá trieda *Embedder*, ktorá vytvorí JSON súbor s požadovanými dátami. Následne sa lokálne uložený CSV súbor vymaže a v grafickom prostredí sa vypíše hláška o úspešnom vytvorení súboru s embeddingami.

4.3.2.2 Spracovanie reportu

Zdroj. kód 13: Časť aplikácie zodpovedná za spracovanie kliknutia na tlačidlo, ktoré spustí vyhodnocovanie poskytnutých embeddingov.

```
51  if generate_report:
52      if uploaded_embeddings is None:
53          st.write(
54              "You have to upload the JSON file containing the generated
54  embeddings first."
55          )
56          st.stop()
57
58      uploaded_embeddings_file_path = save_uploaded_file(
59          uploaded_embeddings, UPLOAD_FOLDER_PATH
60      )
61      pearson_scores = Evaluator.evaluate_all(
62          uploaded_embeddings_file_path,
63      )
64      os.remove(uploaded_embeddings_file_path)
65
66      bottom_container.info(
66          "The embeddings have been evaluated using the Pearson
67  correlation coefficient."
68      )
69      bottom_col1.json(pearson_scores)
70      download_pearson_scores = bottom_col1.download_button(
71          "Download JSON",
72          data=json.dumps(pearson_scores),
73          file_name="pearson_scores.json",
74          mime="text/csv",
75      )
76      chart_data = pd.DataFrame.from_dict(
77          pearson_scores,
78          orient="index",
79      )
80      bottom_col2.bar_chart(chart_data)
```

Tento zdrojový kód ukazuje, čo aplikácia spraví, keď používateľ stlačí tlačidlo, ktoré vygeneruje report o podobnosti vektorov. V prvom rade aplikácia skontroluje, či je vo

formulári nahraný JSON súbor. Ak je, aplikácia ho uloží lokálne pre neobmedzený prístup. Následne pomocou triedy *Evaluator* vyhodnotí dáta z daného JSON súboru. Lokálne uložený JSON súbor sa zmaže, vypíše sa hláška o úspešnom dokončení vyhodnotenia, na stránke sa ukáže blok s výslednými hodnotami vo formáte JSON a vedľa neho sa ukáže graf s rovnakými hodnotami pre lepšiu vizualizáciu.

4.3.3 html_template.py

Tento súbor obsahuje iba predlohu HTML stránky do ktorej sa vkladajú údaje názov stránky, nadpis a paragraf. Názov a nadpis je vždy rovnaký.

Zdroj. kód 14: Predloha obsahu generovaného HTML súboru

```
1  html_template = """
2  <!DOCTYPE html>
3  <html lang="en">
4  <head>
5      <meta charset="UTF-8">
6      <meta name="viewport" content="width=device-width, initial-
6  scale=1.0">
7      <title>{title}</title>
8      <style>
9          body {{font-family: Arial, sans-serif;}}
10         nav {{background-color: #f8f9fa; padding: 10px;}}
11         main {{margin: 15px;}}
12         footer {{background-color: #f8f9fa; text-align: center; padding:
12 10px; margin-top: 20px;}}
13     </style>
14 </head>
15 <body>
16     <nav>
17         <a href="#home">Home</a>
18         <a href="#about">About</a>
19         <a href="#contact">Contact</a>
20     </nav>
21     <main>
22         <h1>{heading}</h1>
23         <p>{text}</p>
24     </main>
25     <footer>
26         &copy; 2024 My Web Page
27     </footer>
28 </body>
29 </html>
30 """
```

4.3.4 file_generator.py

Tento súbor obsahuje triedu *FileGenerator*. Táto trieda obsahuje niekoľko funkcií a dve konštanty – *OUTPUT_FOLDER_PATH* a *SUPPORTED_FORMATS*.

Zdroj. kód 15: Trieda *FileGenerator* a jej atribúty

```
8  class FileGenerator:
9      OUTPUT_FOLDER_PATH = "temp/generated_files/"
10     SUPPORTED_FORMATS = [
11         "pdf",
12         "html",
13         "csv",
14         "txt",
15     ]
```

4.3.4.1 generate_all

Zdroj. kód 16: Metóda *generate_all*

```
17 @classmethod
18 def generate_all(cls, title, text, output_folder=OUTPUT_FOLDER_PATH):
19     cls._generate_pdf(title, text, output_folder)
20     cls._generate_html(title, text, output_folder)
21     cls._generate_csv(title, text, output_folder)
22     cls._generate_plaintext(title, text, output_folder)
```

Táto metóda spája takmer všetky ostatné metódy v tejto triede. Vygeneruje všetky súbory podporované touto triedou na základe zadaného nadpisu a textu. Tieto súbory uloží do zadanej zložky, alebo do *OUTPUT_FOLDER_PATH*, ak žiadna špecifická zložka nebola zadaná.

4.3.4.2 generate_pdf

Zdroj. kód 17: Metóda *generate_pdf*

```
36 @staticmethod
37 def _generate_pdf(title, text, output_folder):
38     pdf = FPDF()
39     pdf.add_page()
40
41     # Title
42     pdf.set_font("Arial", size=24, style="B")
43     pdf.multi_cell(0, 12, txt=title)
44     pdf.ln(4)
45
46     # Text
```

```
47      pdf.set_font("Arial", size=12)
48      pdf.multi_cell(0, 6.5, txt=text)
49
50      pdf.output(f"{output_folder}data.pdf")
```

Táto metóda je súkromná v rámci triedy FileGenerator. Je zodpovedná za generáciu súboru PDF podľa poskytnutého nadpisu a textu. Nadpis je naformátovaný tak, aby bol väčší (veľkosť písma 24) a zároveň hrubý. Po nadpise sa pridá medzera a po medzere nasleduje paragraf textu, ktorý je menší ako nadpis (veľkosť písma 12) a nie je hrubý. PDF súbor je teda naformátovaný tak, aby bol čitateľný pre človeka. Následne sa PDF súbor uloží do zložky, ktorej cesta je poskytnutá metóde.

4.3.4.3 generate_html

Zdroj. kód 18: Metóda *generate_html*

```
52  @staticmethod
53  def _generate_html(title, text, output_folder):
54      with open(f"{output_folder}data.html", "w", encoding="utf-8") as f:
55          f.write(
56              html_template.format(title=title, heading=title, text=text)
57          )
```

Táto metóda je tiež súkromná. Má na starosti generovanie HTML súboru podľa poskytnutého nadpisu a textu. V tomto prípade nie je metóda rozsiahla, nakoľko sa používa HTML predloha zo súboru *html_template.py*, do ktorej sa jednoducho doplnia texty poskytnuté metóde a výsledný HTML súbor sa zapíše do poskytnutej zložky.

4.3.4.4 generate_csv

Zdroj. kód 19: Metóda *generate_csv*

```
59  @staticmethod
60  def _generate_csv(title, text, output_folder):
61      with open(f"{output_folder}data.csv", "w", newline="",
61  encoding="utf-8") as f:
62          writer = csv.writer(f)
63          writer.writerow(["title", "text"])
64          writer.writerow([title, text])
```

Táto metóda je súkromná, rovnako ako predošlé dve. Je určená na generovanie CSV súboru podľa poskytnutého nadpisu a textu. Funguje tak, že vytvorí nový CSV súbor v poskytnutej zložke, do prvého riadku zapíše názvy stĺpcov a do druhého riadku zapíše poskytnutý nadpis a text.

4.3.4.5 generate_plaintext

Zdroj. kód 20: Metóda *generate_plaintext*

```
66  @staticmethod
67      def _generate_plaintext(title, text, output_folder):
68          with open(f"{output_folder}data.txt", "w", encoding="utf-8") as
68  f:
69              f.write(f"{title} - {text}")
```

Táto metóda je v tejto triede posledná a je taktiež súkromná. Funguje tak, že vytvorí TXT súbor v poskytnutej zložke a do tohto súboru zapíše riadok, vo formáte *„Nadpis - Paragraf“*.

4.3.5 loader.py

Tento súbor obsahuje triedu *Loader*. Táto trieda má jednu konštantu a niekoľko metód. Zodpovednosť tejto triedy je načítavať textový obsah súborov, ktorý bude následne embedovaný. Na implementáciu metód v tejto triede bola použitá knižnica *langchain_community* popísaná v sekcii 4.1.1. Konštanta v tejto triede sa nazýva *SUPPORTED_FILETYPES* a je to zoznam podporovaných súborov, kde prvá položka je názov funkcie po prefixe *„load_“* a druhá položka je prípona súboru, ktorý bude spracovaný.

Zdroj. kód 21: Trieda *Loader* a jej atribúty

```
9   class Loader:
10      SUPPORTED_FILETYPES = [
11          ("pdf", "pdf"),
12          ("html", "html"),
13          ("csv", "csv"),
14          ("plaintext", "txt"),
15      ]
```

4.3.5.1 load_all

Zdroj. kód 22: Metóda *load_all*

```
17  @classmethod
18  def load_all(cls, folder_path):
19      monitor = Monitor()
20      texts = []
21      for filetype, file_extension in cls.SUPPORTED_FILETYPES:
22          func = getattr(cls, f"load_{filetype}")
23          doc = monitor.run(
```

```
            "text_loading",
            filetype,
            func,
            f"{folder_path}data.{file_extension}",
        )
        texts.append(doc.page_content)

    return texts
```

Táto metóda načíta texty zo všetkých podporovaných súborov v poskytnutej zložke a vráti ich ako list usporiadaný v rovnakom poradí, aké je poradie typov súborov v konštante *SUPPORTED_FILETYPES*. Pomocou tejto konštanty táto metóda postupne prejde každú metódu v triede na načítanie textu zo súboru (ktorá má správny formát názvu) a pošle túto metódu aj s potrebnými argumentami do triedy *Monitor* (bude popísaná neskôr), ktorá monitoruje časovú a pamäťovú náročnosť načítania textov každého typu súboru.

4.3.5.2 load_pdf

Zdroj. kód 23: Metóda *load_pdf*

```
@staticmethod
def load_pdf(file_path):
    loader = PyPDFLoader(file_path)
    document = loader.load_and_split()[0]
    return document
```

Metóda zodpovedaná za načítanie textového obsahu PDF súboru. Používa *PyPDFLoader* z knižnice *langchain_community*.

4.3.5.3 laod_html

Zdroj. kód 24: Metóda *load_html*

```
@staticmethod
def load_html(file_path):
    loader = UnstructuredHTMLLoader(file_path)
    document = loader.load()[0]
    return document
```

Metóda zodpovedná za načítanie textového obsahu HTML súboru. Používa *UnstructuredHTMLLoader* z knižnice *langchain_community*.

4.3.5.4 load_csv

Zdroj. kód 25: Metóda *load_csv*

```
45  @staticmethod
46  def load_csv(file_path):
47      loader = CSVLoader(file_path)
48      document = loader.load()[0]
49      return document
```

Metóda zodpovedná za načítanie textového obsahu CSV súboru. Používa *CSVLoader* z knižnice *langchain_community*.

4.3.5.5 load_plaintext

Zdroj. kód 26: Metóda *load_plaintext*

```
51  @staticmethod
52  def load_plaintext(file_path):
53      with open(file_path, "r", encoding="utf-8") as file:
54          text = file.read()
55      return Document(page_content=text, metadata={"source": file_path})
```

Metóda zodpovedná za načítanie obsahu textového súboru. Táto metóda jednoducho načíta celý obsah súboru a naformátuje ho to *Document* objektu, nakoľko tento objekt je návratovým typom všetkých ostatných funkcií v tejto triede, ktoré načítavajú obsah súboru a teda pre zachovanie konzistentnosti je aj v tejto metóde návratová hodnota objekt typu *Document*.

4.3.6 embedder.py

Tento súbor obsahuje triedu *Embedder*. Táto trieda je zodpovedná za získavanie vektorových reprezentácií textov jednotlivých súborov, ktoré boli získané pomocou triedy *Loader*. Vektorové reprezentácie sa generujú využitím OpenAI API a ich modelu na generovanie embeddingov. Táto trieda obsahuje dve konštanty a tri metódy. Prvá konštanta sa nazýva *EMBEDDINGS_OUTPUT_PATH* a je v nej uložená cesta k súboru, do ktorého sa vygenerované embeddingy majú uložiť. Druhá konštanta sa volá *OPENAI_API_KEY*. V tejto konštante je uložený API kľúč.

Zdroj. kód 27: Trieda *Embedder* a jej atribúty

```
13  class Embedder:
14      EMBEDDINGS_OUTPUT_PATH = "temp/embeddings/embeddings.json"
15      OPENAI_API_KEY = os.environ["OPENAI_API_KEY"]
```

4.3.6.1 create_embeddings_json

Zdroj. kód 28: Metóda *create_embeddings_json*

```
17  @classmethod
18  def create_embeddings_json(
19      cls, data_file_path, output_path=EMBEDDINGS_OUTPUT_PATH
20  ):
21      all_embeddings = cls._generate_embeddings(data_file_path)
22
23      with open(data_file_path, "r", encoding="utf-8") as file:
24          csv_reader = csv.reader(file)
25
26          embeddings_json = []
27          for i, row in enumerate(csv_reader):
28              similarity_score = row[0]
29
30              embeddings_json_row = {}
31              embeddings_json_row["expected_similarity"] =
31  similarity_score
32              embeddings_json_row["files"] = {}
33
34              for filetype, embeddings in all_embeddings.items():
35                  if filetype == "search_sentences":
36                      embeddings_json_row["search_term_embedding"] = (
37                          embeddings[i]
38                      )
39                  else:
40                      embeddings_json_row["files"][filetype] =
40  embeddings[i]
41
42              embeddings_json.append(embeddings_json_row)
43
44      with open(output_path, "w+") as file:
45          json.dump(embeddings_json, file)
46
47      return embeddings_json
```

Táto metóda má na starosti generovanie finálneho JSON súboru so správne naformátovanými dátami. Na začiatku zavolá metódu *generate_embeddings*, tá vráti všetky embeddingy, s ktorými následne táto metóda pracuje. Po načítaní všetkých embeddingov, metóda spáruje každý riadok v originálnom datasete, ktorý je načítaný z poskytnutej cesty, s embeddingom každého typu súboru a zapíše tieto dáta do pomocného listu, ktorý reprezentuje finálny JSON súbor v Pythone. Následne sa do tohto

listu ešte pridá pre každý záznam predpokladaná miera podobnosti z originálneho datasetu. Následne sa daný list uloží ako JSON súbor do cesty, ktorá bola poskytnutá.

4.3.6.2 generate_embeddings

Zdroj. kód 29: Medóda *generate_embeddings*

```
49  @classmethod
50  def _generate_embeddings(cls, data_file_path, batch_size=10):
51      rows_count = get_csv_row_count(data_file_path, has_header=False)
52
53      with open(data_file_path, "r", encoding="utf-8") as file:
54          csv_reader = csv.reader(file)
55
56          embeddings = {
57              filetype: []
58              for filetype, file_extension in Loader.SUPPORTED_FILETYPES
59          }
60          embeddings["search_sentences"] = []
61
62          batches = copy.deepcopy(embeddings)
63          for i, row in enumerate(csv_reader):
64              print(i)
65
66              search_sentence = row[1]
67              text_title = row[2]
68              text = row[3]
69
70              FileGenerator.generate_all(text_title, text)
71              texts = Loader.load_all(FileGenerator.OUTPUT_FOLDER_PATH)
72              FileGenerator.delete_all()
73              to_embed = texts + [search_sentence]
74
75              for text, type in zip(to_embed, batches.keys()):
76                  batches[type].append(text)
77
78              if ((i + 1) % batch_size == 0) or (i + 1 == rows_count):
79                  for type, batch in batches.items():
80                      if type == "search_sentences":
81                          new_embeddings = cls._embed(batch)
82                      else:
83                          monitor = Monitor()
84                          new_embeddings = monitor.run(
85                              "embedding", type, cls._embed, batch
86                          )
87
88                      embeddings[type].extend(new_embeddings)
89                      batches[type] = []
```

```
90
91      return embeddings
```

Táto metóda je súkromná a je jadrom celej triedy. Prijíma cestu CSV súboru a veľkosť jednej várky. Najprv si metóda získa počet riadkov, ktorý obsahuje CSV súbor v prijatej ceste. Následne sa tento súbor otvorí, pripravia sa potrebné štruktúry na ukladanie dát a začne sa prechádzať jednotlivými riadkami v danom CSV súbore. Pri tomto prechádzaní sa vždy vypíše číslo iterácie, na ktorej sa program nachádza, aby bolo vidieť rýchlosť akou program napreduje, alebo či program nenarazil na nejaký problém, ktorý by ho zasekol. Pri prechádzaní každého riadku daného CSV súboru sa pre každý riadok vygenerujú všetky podporované typy súborov pomocou triedy *FileGenerator*, následne sa z nich načítajú texty pomocou triedy *Loader* a hneď po načítaní sa tieto súbory zmažú, aby sa mohli v nasledujúcej iterácii vygenerovať nové pre dáta na nasledujúcom riadku. Dáta načítané z vygenerovaných súborov pri každej iterácii sa ukladajú do pomocnej premennej (aktuálna várka). Pri každej iterácii sa tiež sleduje, či bol dosiahnutý limit pre veľkosť jednej várky. V prípade, že bol tento limit dosiahnutý, spustí sa samostatne pre každý typ súboru a potom samostatne pre typ *„search_sentences“* (prvá veta z dvojice, na ktorej sa bude neskôr robiť výpočet vzdialenosti vektorov) metóda *embed*, ktorá vráti vygenerované embeddingy pre jednotlivé várky, rozdelené podľa typu súboru. Táto metóda sa spustí cez metódu *run* z triedy *Monitor*, aby bola zaznamenaná pamäťová a časová náročnosť tohto spracovania. Po spracovaní všetkých riadkov sa z tejto metódy vrátia všetky vygenerované embeddingy.

4.3.6.3 Embed

Zdroj. kód 30: Medóda *embed*

```
106 @classmethod
107 def _embed(cls, batch):
108     embeddings_model = OpenAIEmbeddings(
109         model="text-embedding-3-large", api_key=cls.OPENAI_API_KEY
110     )
111     return embeddings_model.embed_documents(batch)
```

Táto metóda je súkromná a je zodpovedná za získavanie vektorových reprezentácií prijatého textu pomocou OpenAI API. Na generovanie embeddingov sa používa model *„text-embedding-3-large“*. Táto metóda prijíma list textov a vracia list vektorov, ktoré sú v rovnakom poradí ako boli dané texty.

4.3.7 evaluator.py

Tento súbor obsahuje triedu *Evaluator*, ktorá obsahuje 3 funkcie. Táto trieda je zodpovedaná za spracovanie dát v JSON súbore, ktorý bol vygenerovaný triedou *Embedder* a poskytovanie výsledkov porovnania predpokladanej sémantickej podobnosti a skutočne vypočítanej sémantickej podobnosti na základe týchto dát.

4.3.7.1 openai_cosine_similarity

Zdroj. kód 31: Metóda *openai_cosine_similarity*

```
6   @staticmethod
7   def _openai_cosine_similarity(vec_1, vec_2):
8       vec_1 = np.array(vec_1)
9       vec_2 = np.array(vec_2)
10      dot_product = np.dot(vec_1, vec_2)
11      return dot_product
```

Táto metóda je zodpovedná za počítanie sémantickej podobnosti pomocou *„cosine similarity“*. Podľa dokumentácie od spoločnosti OpenAI [39] sa pre zjednodušenie táto metrika môže vypočítať ako bodový súčin dvoch vektorov.

4.3.7.2 pearson_correlation

Zdroj. kód 32: Metóda *pearson_correlation*

```
13  @staticmethod
14  def _pearson_correlation(list_1, list_2):
15      list_1 = np.array(list_1)
16      list_2 = np.array(list_2)
17
18      diffs_1 = list_1 - np.mean(list_1)
19      diffs_2 = list_2 - np.mean(list_2)
20
21      numerator = np.sum(diffs_1 * diffs_2)
22      denominator = np.sqrt(np.sum(diffs_1**2) * np.sum(diffs_2**2))
23
24      return numerator / denominator
```

Táto metóda je zodpovedná za počítanie Pearsonovho korelačného koeficientu medzi dvomi listami hodnôt. V prototype sa pomocou tejto metódy počíta výsledná metrika podľa ktorej sa porovnávajú kvality embeddingov jednotlivých súborov.

4.3.7.3 evaluate_all

Zdroj. kód 33: Metóda *evaluate_all*

```
26  @classmethod
27  def evaluate_all(cls, embeddings_file_path):
28      with open(embeddings_file_path) as file:
29          data = json.load(file)
30
31      cosine_similarities = {
32          filetype: [] for filetype in data[0]["files"].keys()
33      }
34      for item in data:
35          for filetype, vector in item["files"].items():
36              cosine_similarities[filetype].append(
37                  cls._openai_cosine_similarity(
38                      item["search_term_embedding"], vector
39                  )
40              )
41
42      expected_similarities = [
43          float(item["expected_similarity"]) for item in data
44      ]
45      pearson_scores = {
46          filetype: 0 for filetype in cosine_similarities.keys()
47      }
48      for filetype, cos_similarity_scores in cosine_similarities.items():
49          pearson_scores[filetype] = cls._pearson_correlation(
50              expected_similarities, cos_similarity_scores
51          )
52
53      return pearson_scores
```

Táto metóda má za úlohu vyhodnotiť kvalitu embeddingov jednotlivých typov súborov. Prebieha to tak, že sa najskôr vypočítajú *„cosine similarity“* metriky, pre vektor zo súboru v porovnaní s neupravenou vetou z datasetu. Všetky tieto metriky sa uložia do rôznych listov pre každý typ súboru. Následne sa pomocou Pearsonovho korelačného koeficientu pre každý typ súboru separátne vypočíta sila korelácie medzi všetkými hodnotami predpokladanej sémantickej podobnosti a všetkými hodnotami podobnosti vypočítanými pomocou *„cosine similarity“* metriky. Tak sa získajú koncové hodnoty kvality embeddinov pre každý typ súboru.

4.3.8 monitor.py

Tento súbor obsahuje triedu *Monitor*, ktorá je zodpovedná za monitorovanie a zaznamenávanie časovej a pamäťovej náročnosti istých úsekov zdrojového kódu. Táto trieda využíva návrhový vzor nazývaný *„Singleton“*. Na zaznamenávanie dát sa používa JSON súbor rozdelený na kategórie (čas a pamäť) a v rámci daných kategórií sa delí na jednotlivé typy súborov.

Zdroj. kód 34: Trieda *Monitor* a jej atribút

```
7  class Monitor:
8      _instance = None
```

4.3.8.1 __new__

Zdroj. kód 35: Špeciálna metóda *__new__*

```
10 def __new__(cls):
11     if cls._instance is None:
12         cls._instance = super().__new__(cls)
13         cls._instance.OUTPUT_FILE_PATH = (
14             f"temp/monitor/monitor-{int(time.time())}.json"
15         )
16     return cls._instance
```

Táto metóda je špeciálna, pretože sa vykonáva pri vytvorení novej inštancie tejto triedy. Ak žiadna inštancia neexistuje, vytvorí sa nová, kde sa nastaví *OUTPUT_FILE_PATH* konštanta na špecifickú cestu s časom v názve pre rozoznanie rôznych dát z monitorovania. Následne sa táto inštancia uloží do súkromnej triednej premennej *_instance*. V prípade že sa pri pokuse o vytvorenie novej inštancie zistí, že už je v premennej *_instance* nejaká inštancia uložená, metóda vráti inštancia uloženú v danej premennej. Vďaka tomu sa trieda správa ako Singleton.

4.3.8.2 run

Zdroj. kód 36: Metóda *run*

```
18 def run(self, category, filetype, func, *args, **kwargs):
19         func_return = self._monitor_time(
20             category,
21             filetype,
22             self._monitor_memory,
23             category,
24             filetype,
```

```
            func,
            *args,
            **kwargs,
        )
        return func_return
```

Táto metóda je len obaľovacia metóda, ktorá slúži na spustenie monitorovania časovej aj pamäťovej náročnosti naraz.

4.3.8.3 monitor_memory

Zdroj. kód 37: Metóda *monitor_memory*

```
def _monitor_memory(self, category, filetype, func, *args, **kwargs):
    tracemalloc.start()
    func_return = func(*args, *kwargs)
    current, peak = tracemalloc.get_traced_memory()
    tracemalloc.stop()

    self._update_monitoring_json(category, "memory", filetype, peak)

    return func_return
```

Táto metóda je zodpovedná za sledovanie pamäťovej náročnosti v určitom úseku zdrojového kódu. Metóda prijíma kategóriu a typ súboru pre zápis dát do JSON súboru. Taktiež metóda prijíma funkciu, ktorej vykonávanie má monitorovať s argumentami a pomenovanými argumentami. Na začiatku spustí monitorovanie pamäte a zavolá funkciu, ktorá má byť monitorovaná, s potrebnými argumentami. Po dokončení vykonávania danej funkcie táto metóda získa aktuálne využitie a maximálne využitie pamäte v rámci vykonávania danej funkcie. Hneď po tom sa sledovanie pamäte zastaví a do JSON súboru sa zapíše maximálna hodnota používania pamäte. Na koniec metóda vráti hodnotu, ktorá vrátila monitorovaná funkcia.

4.3.8.4 monitor_time

Zdroj. kód 38: Metóda *monitor_time*

```
41  def _monitor_time(self, category, filetype, func, *args, **kwargs):
42      start_time = time.perf_counter()
43      func_return = func(*args, **kwargs)
44      end_time = time.perf_counter()
45
46      elapsed_time = end_time - start_time
47      self._update_monitoring_json(
48          category,
49          "time",
50          filetype,
51          elapsed_time,
52      )
53
54      return func_return
```

Táto metóda je zodpovedná za meranie času, ktorý je potrebný na dokončenie spracovania danej funkcie. Funkcia prijíma kategóriu a súborový typ na zápis dát do JSON súboru. Okrem toho metóda prijíma funkciu, ktorú má sledovať a všetky jej argumenty. Metóda si najskôr uloží aktuálny čas, spustí funkciu na sledovanie a počká kým sa dokončí. Akonáhle sa daná funkcia dokončí, metóda sa znovu pozrie na aktuálny čas. Rozdiel času pred spustením funkcie a po dokončení funkcie sa zapíše do JSON súboru pod danú kategóriu a typ súboru. Na koniec metóda vráti hodnotu, ktorú vrátila monitorovaná funkcia.

4.3.8.5 update_monitoring_json

Zdroj. kód 39: Metóda *update_monitoring_json*

```
56  def _update_monitoring_json(
57      self,
58      category,
59      monitor_type,
60      filetype,
61      value,
62  ):
63      if not os.path.exists(self.OUTPUT_FILE_PATH):
64          with open(self.OUTPUT_FILE_PATH, "w") as file:
65              data = {}
66              json.dump(data, file)
67
68      with open(self.OUTPUT_FILE_PATH, "r") as file:
69          data = json.load(file)
70
```

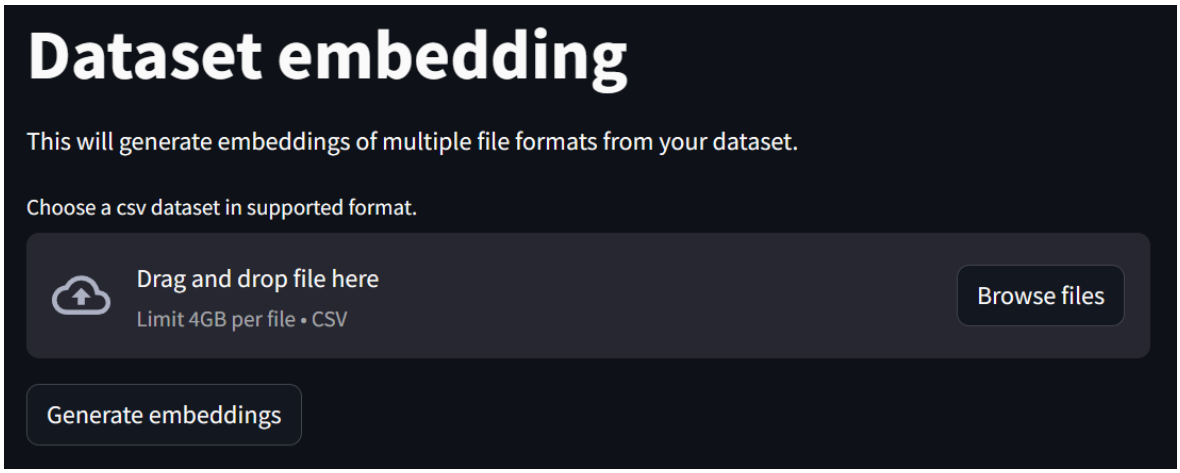
```
71      if category not in list(data.keys()):
72          data[category] = {"time": {}, "memory": {}}
73
74      if filetype not in list(data[category][monitor_type].keys()):
75          data[category][monitor_type][filetype] = []
76
77      data[category][monitor_type][filetype].append(value)
78
79      with open(self.OUTPUT_FILE_PATH, "w") as file:
80          json.dump(data, file)
```

Táto metóda je zodpovedná za zapisovanie potrebných údajov z monitorovania do JSON súboru. Prijíma kategóriu, typ monitoru, typ súboru a hodnotu. V prípade, že daný JSON súbor ešte neexistuje, vytvorí ho a zapíše prvé dáta. V prípade, že už daný súbor existuje súbor sa otvorí, načítajú sa z neho dáta a prebehnú kontroly či už existujú kľúče požadované na zapísanie danej hodnoty. V prípade že neexistujú, vytvoria sa. Požadovaná hodnota sa zapíše do načítaných dát a všetky dáta sa následne uložia do daného JSON súboru.

4.4 Grafické prostredie

Grafické prostredie prototypu tejto webovej aplikácie je rozdelené na dve časti. V prvom rade je potrebný formulár na spracovanie datasetu z CSV súboru. V druhom rade je potrebný formulár na spracovanie vektorov vygenerovaných v prvej sekcii.

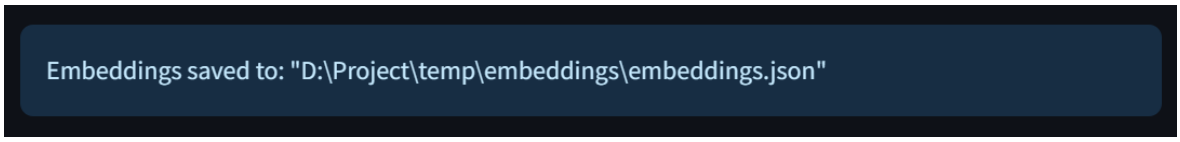
4.4.1 Dataset embedding



Obr. 7: Prvá časť aplikácie, ktorá slúži na generovanie embeddingov

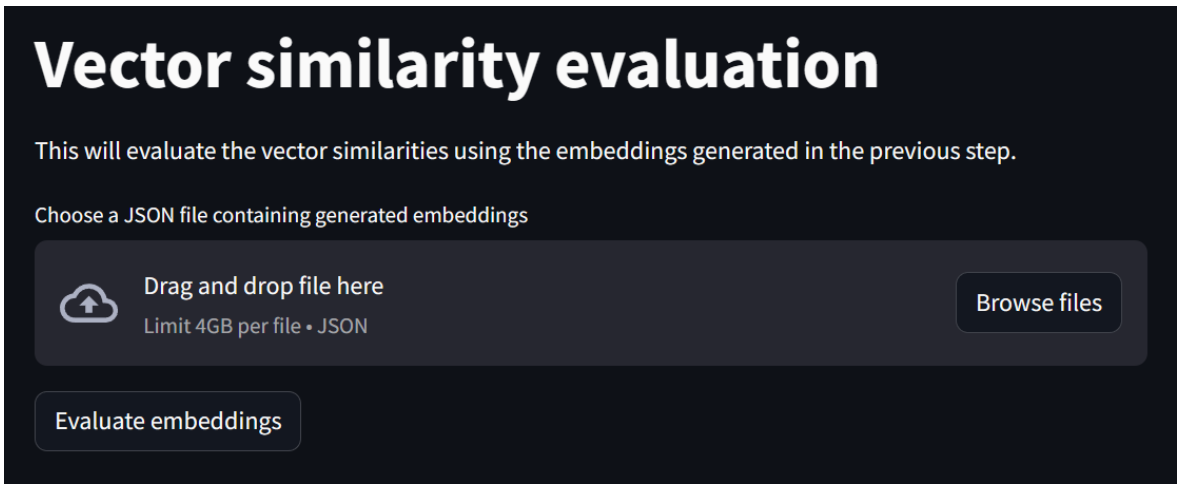
Táto sekcia obsahuje nadpis, podnadpis, vstup na nahratie súboru a tlačidlo na spustenie generovania embeddingov. V prípade, že používateľ na tlačidlo klikne bez toho aby bol

v prototype nahraný CSV súbor, vypíše sa chybová hláška. V prípade, že je v prototype nahraný súbor typu CSV a používateľ klikne na tlačidlo, spustí sa proces generovania embeddingov, čo môže zabrať niekoľko jednotiek až desiatok minút. V grafickom prostredí nie je žiaden indikátor toho, koľko už prototyp spracoval dát, no je vidieť, že stále pracuje. To znamená, že používateľ bude vidieť, že sa niečo deje. Tento prototyp je taktiež určený na lokálne používanie, takže používateľ bude mať prístup k príkazovému riadku z ktorého prototyp spustil. V tomto príkazovom riadku sa postupne ukazujú čísla riadkov, ktoré už prototyp spracoval. Po úspešnom dokončení generovania sa v grafickom prostredí ukáže hláška o uložení embeddingov.



Obr. 8: Hláška o úspešnom dokončení generovanie embeddingov

4.4.2 Vector similarity evaluation



Obr. 9: Druhá časť aplikácie, ktorá slúži na vyhodnotenie embeddingov

Táto sekcia má rovnaké interakčné prvky ako predošlá sekcia, no v tejto sekcii je potrebné do prototypu nahrať súbor typu JSON, ktorý bol vygenerovaný v predošlej sekcii. Ak v prototype nie je nahraný potrebný súbor a používateľ klikne na tlačidlo *„Evaluate embeddings“*, pod tlačidlom sa zobrazí chybová hláška. V prípade, že je v prototype nahraný potrebný súbor, kliknutím na tlačidlo, sa spustí výpočet podobností vektorov v danom JSON súbore. Po úspešnom vyhodnotení výsledkov sa pod touto sekciou zobrazí hláška informujúca o tom, že proces bol úspešný. Taktiež sa zobrazia výsledné dáta vo

formáte JSON a vedľa sa zobrazí stĺpcový graf ukazujúci rovnaké dáta. Pod dátami vo formáte JSON je taktiež tlačidlo na stiahnutie daných dát do JSON súboru.

Obr. 10: Hláška o úspešnom dokončení vyhodnotenia a výsledné dáta

5 EXPERIMENT A VÝSLEDKY

Táto kapitola popisuje, ako prebiehal experiment pre túto prácu krok po kroku. Taktiež je zameraná na popis a zhodnotenie výsledkov podľa dát, získaných v rámci vykonávania daného experimentu.

5.1 Proces experimentu

Popis získania a spracovania pôvodného datasetu bol popísaný v kapitole 3. V tejto kapitole sa teda počíta s hotovým a pripraveným datasetom na embedding.

V prvom rade je potrebné otvoriť prototyp webovej aplikácie, ktorý je v tejto práci popísaný. Do prvej časti prototypu je potrebné nahrať dataset vo formáte CSV. Dataset použitý v tejto práci má 5726 riadkov. Po úspešnom nahraní datasetu do prototypu je potrebné kliknúť na tlačidlo s textom *„Generate embeddings“*. V pravom hornom rohu aplikácie sa zobrazí text *„RUNNING...“* s meniacimi sa ikonkami. Dĺžka tohto kroku záleží od veľkosti datasetu. Embedovanie celého datasetu pre túto prácu aj s monitorovaním časovej a pamäťovej náročnosti trvalo 1 hodinu a 15 minút. Po úspešnom dokončení tohto kroku bude vygenerovaný súbor typu JSON so všetkými embeddovanými dátami, ktoré sú potrebné pre nasledujúci krok. Tento súbor má veľkosť takmer 2 gigabajty. Taktiež bude vygenerovaný súbor typu JSON, ktorý obsahuje všetky dáta o časovej a pamäťovej náročnosti jednotlivých typov súborov.

Druhý krok vyžaduje nahranie súboru vygenerovaného v predošlom kroku do prototypu. Nahranie súboru môže trvať zopár sekúnd, nakoľko môže byť tento súbor pomerne rozsiahly. Po úspešnom nahraní daného JSON súboru, je potrebné kliknúť na tlačidlo s textom *„Evaluate embeddings“*. Po kliknutí na toto tlačidlo sa začne vykonávať vyhodnocovanie výsledkov. To bude sprevádzané rovnakým textom *„RUNNING...“* v pravom hornom rohu ako v predošlom kroku. Tento krok zaberie jednoznačne menej času, ako ten predošlý, no keďže prototyp pracuje s veľkým objemom dát, výsledky nebudú vypočítané ihneď. S datasetom použitým v tejto práci trvalo vyhodnotenie výsledkov 1 minútu. Na konci tohto kroku sa v grafickom prostredí zobrazia výsledné hodnoty Pearsonovho korelačného koeficientu.

5.2 Zhodnotenie výsledkov

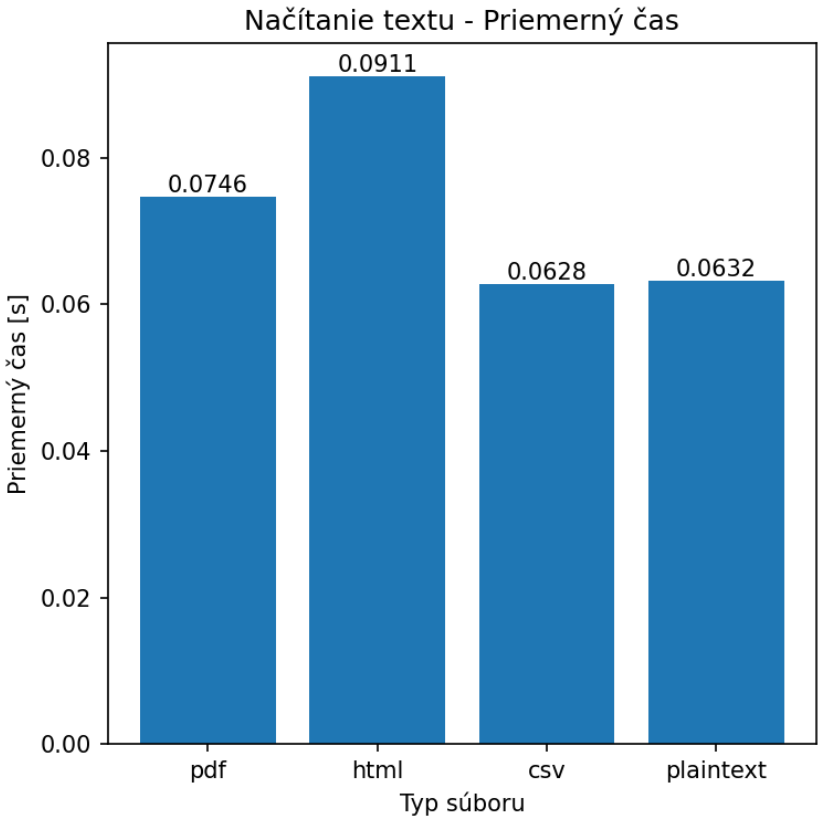
Táto sekcia rozoberá dáta získané z experimentu popísaného v predošlej kapitole. Taktiež sú v tejto kapitole utvorené závery na základe týchto dát.

5.2.1 Časová náročnosť

Časová náročnosť bola meraná na dvoch miestach – pri načítaní obsahu súboru a pri embeddovaní obsahov súboru. Z oboch miest boli zaznamenané dáta, z ktorých bol následne vytvorený graf.

5.2.1.1 Načítanie obsahu súborov

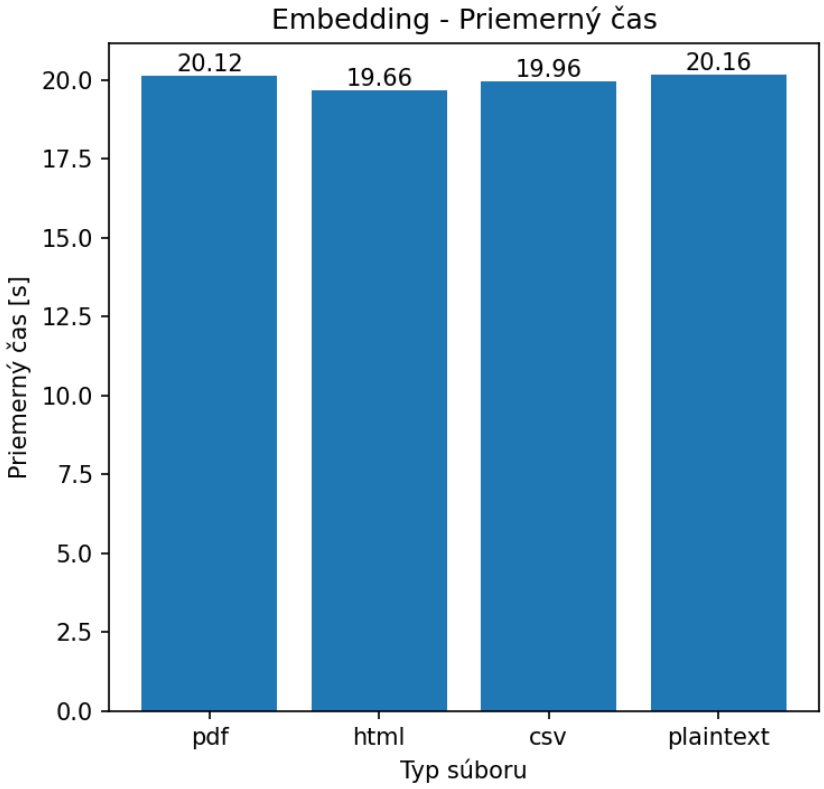
Najlepší výsledok časovej náročnosti dosiahol súborový formát CSV. V prípade, že by sa o priemernom čase načítania textu tohto súborového formátu uvažovalo ako o 100 percentách, ostatné hodnoty by mohli byť porovnané v percentuálnej podobe. Jednoznačne najhorší priemerný čas na načítanie textu zo súboru mal súborový typ HTML, ktorý mal o 45 percent vyšší priemer ako súborový formát CSV. Hneď po ňom je najpomalší súborový typ PDF s priemerným časom načítania textového obsahu o 18 percent vyšším ako súborový formát CSV. Súborový formát TXT mal veľmi podobnú priemernú hodnotu času načítania textového obsahu, keďže bol len o 0.6 percenta pomalší ako súborový formát CSV.



Obr. 11: Graf priemerného času načítania textu jednotlivých súborových formátov

5.2.1.2 Embeddovanie

Monitorovanie časovej náročnosti pri embeddovaní malo iné výsledky. Embeddovanie bolo robené v požiadavkách na OpenAI API každých 500 riadkov datasetu. To je dôvod, prečo sú priemerné časy embeddingu tak vysoké. Najkratší priemerný čas na embedding mali dáta zo súborového formátu HTML. Priemerný čas na embeddovanie 500 záznamov textov načítaných zo súboru HTML bude považovaný za 100 percent pre porovnávanie voči ostatným formátom. Najhorší priemerný čas embeddovania mali dáta načítané zo súborového formátu TXT. Tieto dáta boli embeddované v priemere o 2.5 percenta pomalšie. Hneď po formáte TXT nasledovali dáta zo súborového formátu PDF, ktoré sa v priemere embeddovali o 2.3 percenta pomalšie. Dáta zo súborového formátu CSV sa v priemer embeddovali o 1.5 percenta pomalšie.



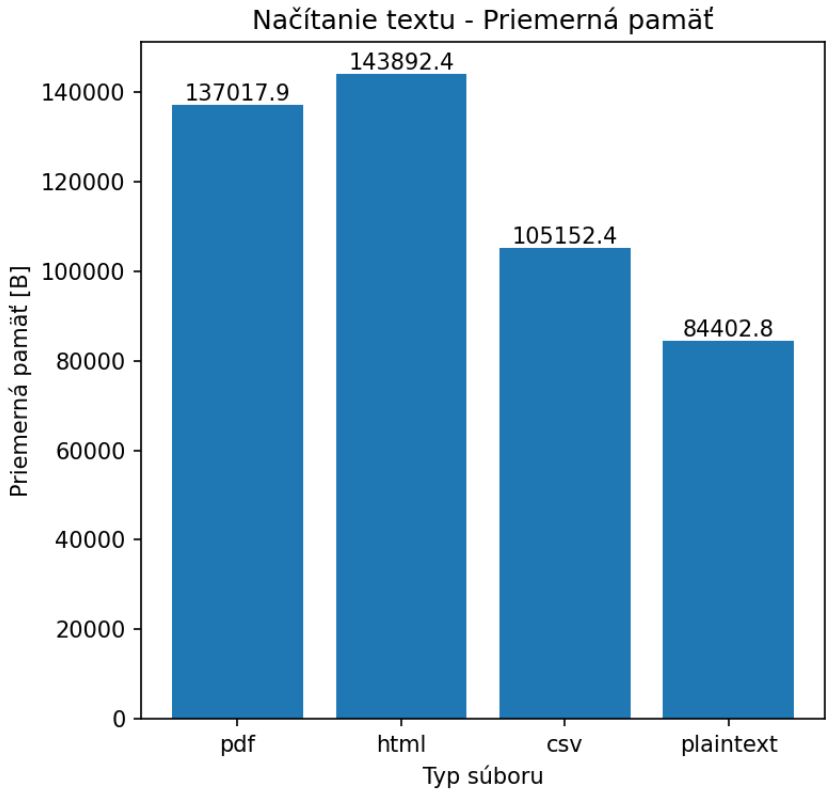
Obr. 12: Graf priemerného času generovania embeddingov pre jednotlivé súborvé formáty

5.2.2 Pamäťová náročnosť

Pamäťová náročnosť bola (rovnako ako časová náročnosť) meraná na dvoch miestach – pri načítaní obsahu súborov a pri embeddovaní obsahov daných súborov.

5.2.2.1 Načítanie obsahu súborov

Najlepšiu, a teda najnižšiu, pamäťovú náročnosť pri načítaní obsahu súborov mal súborový typ TXT. Pre porovnanie s ostatnými súborovými typmi bude priemerná hodnota pamäťovej náročnosti súborového typu TXT považovaná za 100 percent. Súborový typ, ktorý mal druhú najlepšiu priemernú pamäťovú náročnosť pri načítaní obsahu zo súboru bol typ CSV. Nárast v pamäťovej náročnosti súborového typu CVS oproti typu TXT bol 24.5 percenta. Tretí bol súborový typ PDF, ktorý mal priemernú pamäťovú náročnosť o 62 percent väčšiu ako formát TXT. Najhoršiu pamäťovú náročnosť mal súborový formát HTML, ktorý dosiahol priemeru o 70.5 percenta vyššieho ako súborový formát TXT.

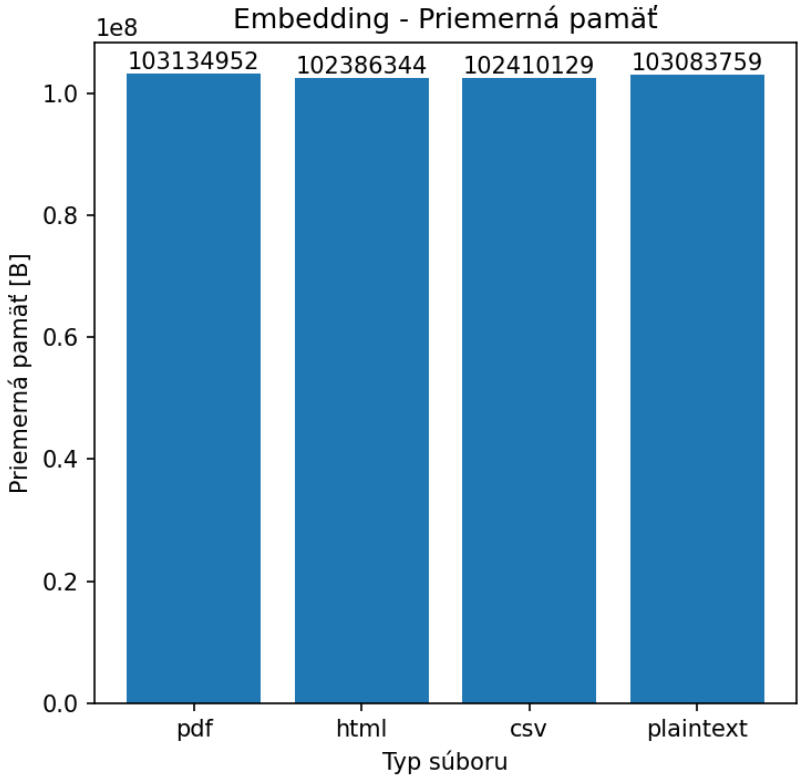


Obr. 13: Graf priemernej veľkosti použitej pamäte pri načítaní textu z jednotlivých súborových formátov

5.2.2.2 Embedding

Pamäťová náročnosť pri embeddingu bola meraná na každých 500 záznamov. Preto sú priemerné hodnoty tejto pamäťovej náročnosti tak vysoké. Z týchto dát je viditeľné, že najnižšiu pamäťovú náročnosť mali dáta zo súborového typu HTML. Najvyššiu priemernú pamäťovú náročnosť mal súborový typ PDF. Rozdiel medzi súborovými typmi HTML a PDF a teda medzi najvyššou a najnižšou pamäťovou náročnosťou v týchto dátach je 0.7

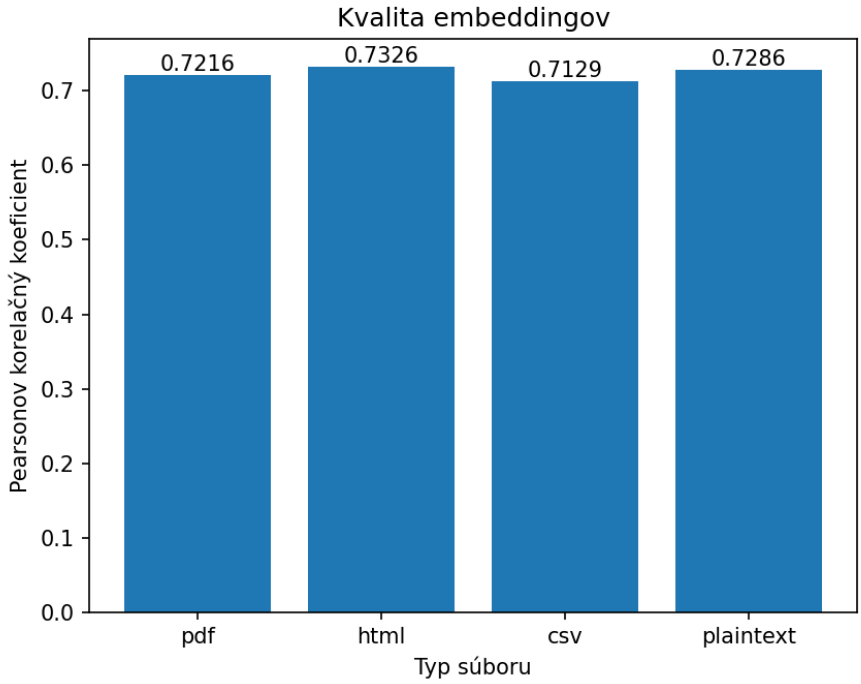
percenta. Priemerné pamäťové náročnosti súborových typov CSV a TXT sa nachádzajú medzi vyššie spomínanými typmi HTML a PDF.



Obr. 14: Graf priemernej veľkosti použitej pamäte pri generovaní embeddingov jednotlivých súborových formátov

5.2.3 Kvalita embeddingov

Kvalita embeddingov bola počítaná pomocou metriky *„cosine similarity“*. Táto metrika bola použitá, pretože OpenAI ju odporúča pre porovnávanie vzdialenosti dvoch vektorov generovaných ich modelmi [39]. Táto metrika bola vypočítaná pre každý pár (pôvodná veta – text zo súboru) vektorov. Tým boli získané hodnoty sémantickej podobnosti daných párov v rámci každého typu súboru. So získanými hodnotami sémantickej podobnosti vygenerovaných vektorov je možné ich porovnať s pôvodnými *„golden standard“* hodnotami sémantickej podobnosti. Na porovnanie korelácie medzi danými dvomi hodnotami bol použitý Pearsonov korelačný koeficient, ktorého základné vysvetlenie je popísané v kapitole 2.4.



Obr. 15: Kvalita embeddingov jednotlivých súborových formátov vyhodnotená pomocou Pearsonovho korelačného koeficientu

Dáta, ktoré sú výstupom prototypu webovej aplikácie vytvorenej v rámci tejto práce naznačujú, že žiadny z testovaných súborových formátov nemá výrazne lepšiu koreláciu vypočítanej sémantickej podobnosti s pôvodnými hodnotami sémantickej podobnosti. Najhoršiu koreláciu mal súborový typ CSV. Najlepšiu koreláciu mal súborový typ HTML. Rozdiel medzi týmito dvomi hodnotami je 2.7 percenta. Oba súborové formáty PDF a TXT majú veľmi podobné hodnoty s rozdielom len 1 percento. Tieto dáta naznačujú záveru, že žiaden z testovaných súborových formátov nie je výrazne lepší ako ostatné pre embedding obsahu súborov daného formátu.

ZÁVĚR

Experiment uskutočnený v rámci tejto bakalárskej práce priniesol cenné údaje o kvalite a efektivite rôznych súborových formátov pre tvorbu embeddingov dát vo veľkých jazykových modeloch. Z výsledkov je zrejmé, že rozdiel v kvalite vygenerovaných embeddingov medzi skúmanými súborovými formátmi je veľmi malý, len 2.7 percenta medzi najlepším a najhorším variantom. Tento malý rozdiel naznačuje, že z hľadiska kvality embeddingov nie je nutné uprednostňovať konkrétny súborový formát spomedzi skúmaných formátov.

Pri načítaní obsahu súborov boli zistené značné rozdiely v časovej a pamäťovej náročnosti. Súborový formát CSV preukázal najlepšiu časovú efektivitu, pričom bol jednoznačne rýchlejší ako ostatné formáty. Na druhej strane, najhorší časový výsledok dosiahol súborový formát HTML, ktorý bol o 45 percent pomalší v porovnaní s formátom CSV. Pokiaľ ide o pamäťovú náročnosť, najlepšie obstál súborový formát TXT, zatiaľ čo formát HTML bol z hľadiska pamäťovej náročnosti najhorší.

Pri embeddovaní dát pomocou OpenAI API sa ukázalo, že časová aj pamäťová náročnosť bola podobná pre všetky súborové formáty, bez výrazných rozdielov. Tieto zistenia naznačujú, že pre potreby embeddovania je možné využívať rôzne formáty bez výraznej straty efektivity v tomto kroku.

SEZNAM POUŽITÉ LITERATURY

[1] JURAFSKY, Daniel a James H. MARTIN, 2009. Speech and Language Processing An Introduction to Natural Language Processing, Computational Linguistics, and Speech Recognition. 2nd ed. Pearson Prentice Hall. ISBN 9780131873216.

[2] WAKEUPCODERS, 2020. Part-1: Introduction to Natural Language Processing (NLP). MEDIUM. Medium [online]. [cit. 2024-05-03]. Dostupné z: https://medium.com/analytics-vidhya/part-1-introduction-to-natural-language-processing-nlp-a66ad8773b3

[3] LEE, Angie, 2023. What Are Large Language Models Used For? NVIDIA. NVIDIA [online]. [cit. 2024-05-03]. Dostupné z: https://blogs.nvidia.com/blog/what-are-large-language-models-used-for/

[4] RED HAT, 2019. What is open source? RED HAT. Red Hat [online]. [cit. 2024-05-03]. Dostupné z: https://www.redhat.com/en/topics/open-source/what-is-open-source

[5] GEORGE, Alexandra, 2022. Python text mining: Perform text processing, word embedding, text classification and Machine Translation. 1. BPB Publications. ISBN 9389898781.

[6] ÇELIK, Tuana, 2022. The Beginner's Guide to Text Embeddings. DEEPSET. Deepset [online]. [cit. 2024-04-03]. Dostupné z: https://www.deepset.ai/blog/the-beginners-guide-to-text-embeddings

[7] MANSUROVA, Mariya, 2024. Text Embeddings: Comprehensive Guide - towards Data Science. Medium [online]. [cit. 2024-04-02]. Dostupné z: https://towardsdatascience.com/text-embeddings-comprehensive-guide-afd97fce8fb5

[8] EGGER, Roman, 2022. Applied Data Science in Tourism Interdisciplinary Approaches, Methodologies, and Applications. Springer. ISBN 3030883884.

[9] DUTTA, Mimi, 2023. Word2Vec For Word Embeddings -A Beginner's Guide. Analytics Vidhya [online]. [cit. 2024-04-10]. Dostupné z: https://www.analyticsvidhya.com/blog/2021/07/word2vec-for-word-embeddings-a-beginners-guide/

[10] ZHANG, Aston et al., 2023. Dive into Deep Learning. Cambridge University Press. ISBN 1009389432.

[11] PERAMBAI, Abhishek, 2020. GLoVE: Theory and Python Implementation. Medium [online]. [cit. 2024-04-11]. Dostupné z: https://medium.com/analytics-vidhya/glove-theory-and-python-implementation-b706aea28ac1

[12] AKHTAR, Zuhaib, 2021. ELMo: Deep contextualized word representations. OpenGenus IQ [online]. [cit. 2024-04-13]. Dostupné z: https://iq.opengenus.org/elmo/

[13] KULSHRESTHA, Ria, 2020. Keeping up with the BERTs. Towards Data Science [online]. [cit. 2024-04-14]. Dostupné z: https://towardsdatascience.com/keeping-up-with-the-berts-5b7beb92766

[14] KULSHRESTHA, Ria, 2020. Transformers. Towards Data Science [online]. [cit. 2024-04-14]. Dostupné z: https://towardsdatascience.com/transformers-89034557de14

[15] CARDENAS, Erika, 2023. Distance Metrics in Vector Search. Weaviate [online]. [cit. 2024-04-30]. Dostupné z: https://weaviate.io/blog/distance-metrics-in-vector-search

[16] BOBBITT, Zach, 2019. Pearson Correlation Coefficient. Statology [online]. [cit. 2024-04-28]. Dostupné z: https://www.statology.org/pearson-correlation-coefficient/

[17] NGUYEN, Tuyen, 2017. GitHub - nttuyenx/STS: Semantic Textual Similarity. GitHub [online]. [cit. 2024-04-09]. Dostupné z: https://github.com/nttuyenx/STS

[18] AGIRRE, Eneko, 2018. Semantic Textual Similarity Wiki. Stswiki [online]. [cit. 2024-04-09]. Dostupné z: https://ixa2.si.ehu.eus/stswiki/

[19] OPENAI et al., 2024. GPT-4 Technical Report [online]. V6. [cit. 2024-05-03]. 10.48550/arxiv.2303.08774. Dostupné z: https://arxiv.org/abs/2303.08774

[20] GANESAN, Kavita, c2024. All you need to know about text preprocessing for NLP and Machine Learning. KDnuggets [online]. [cit. 2024-04-14]. Dostupné z: https://www.kdnuggets.com/2019/04/text-preprocessing-nlp-machine-learning.html

[21] Sys — System-specific parameters and functions, c2001-2024. Python documentation [online]. [cit. 2024-04-17]. Dostupné z: https://docs.python.org/3.11/library/sys.html

[22] Os — Miscellaneous operating system interfaces, c2001-2024. Python documentation [online]. [cit. 2024-04-17]. Dostupné z: https://docs.python.org/3.11/library/os.html

[23] Csv — CSV File Reading and Writing, c2001-2024. Python documentation [online]. [cit. 2024-04-17]. Dostupné z: https://docs.python.org/3.11/library/csv.html

[24] Json — JSON encoder and decoder, c2001-2024. Python documentation [online]. [cit. 2024-04-17]. Dostupné z: https://docs.python.org/3.11/library/json.html

[25] ELETI, Atty et al., 2024. OpenAI Python API library. Python Package Index (PyPI) [online]. [cit. 2024-04-17]. Dostupné z: https://pypi.org/project/openai/

[26] REINGART, Mariano, 2012. FPDF for Python. PyFPDF [online]. [cit. 2024-04-18]. Dostupné z: https://pyfpdf.readthedocs.io/en/latest/

[27] VARTANIAN, Erica, 2022. Why learn Python? 5 advantages and disadvantages. Educative [online]. [cit. 2024-04-19]. Dostupné z: https://www.educative.io/blog/why-learn-python

[28] ROES, Jon et al., 2024. Streamlit. Python Package Index (PyPI) [online]. [cit. 2024-04-19]. Dostupné z: https://pypi.org/project/streamlit/

[29] CHASE, Harrison, 2024. LangChain Community. Python Package Index (PyPI) [online]. [cit. 2024-04-19]. Dostupné z: https://pypi.org/project/langchain-community/

[30] CHASE, Harrison, 2024. LangChain Core. Python Package Index (PyPI) [online]. [cit. 2024-04-19]. Dostupné z: https://pypi.org/project/langchain-core/

[31] FRIIS, Erick, 2024. Langchain-openai. Python Package Index (PyPI) [online]. [cit. 2024-04-19]. Dostupné z: https://pypi.org/project/langchain-openai/

[32] Copy — Shallow and deep copy operations, c2001-2024. Python documentation [online]. [cit. 2024-04-17]. Dostupné z: https://docs.python.org/3/library/copy.html

[33] HARRIS, Charles R. et al., 2024. NumPy. Python Package Index (PyPI) [online]. [cit. 2024-04-19]. Dostupné z: https://pypi.org/project/numpy/

[34] Time — Time access and conversions, c2001-2024. Python documentation [online]. [cit. 2024-04-17]. Dostupné z: https://docs.python.org/3/library/time.html

[35] Tracemalloc — Trace memory allocations, c2001-2024. Python documentation [online]. [cit. 2024-04-17]. Dostupné z: https://docs.python.org/3/library/tracemalloc.html

[36] DATAPYTHONISTA et al., 2024. Pandas. Python Package Index (PyPI) [online]. [cit. 2024-04-19]. Dostupné z: https://pypi.org/project/pandas/

[37] MICROSOFT, c2024. Visual Studio Code. Visual Studio Code [online]. [cit. 2024-04-20]. Dostupné z: https://code.visualstudio.com/

[38] REITZ, Kenneth a PYPA, c2020. Pipenv: Python Dev Workflow for Humans. Python Packaging Authority [online]. [cit. 2024-04-20]. Dostupné z: https://pipenv.pypa.io/en/latest/

[39] OPENAI. Embeddings. OPENAI. OpenAI developer platform [online]. [cit. 2024-04-28]. Dostupné z: https://platform.openai.com/docs/guides/embeddings

SEZNAM POUŽITÝCH SYMBOLŮ A ZKRATEK

TF-IDF	Term Frequency - Inverse Document Frequency
NLP	Natural Language Processing
LLM	Large Language Model
BoW	Bag of Words
CBOW	Continuous Bag of Words
GLoVe	Global Vectors
ELMo	Embeddings from Language Models
LSTM	Long-Short-Term Memory
BERT	Bidirectional Encoder Representations from Transformers
MLM	Masked Language Model
NSP	Next Sentence Prediction
STS	Semantic Textual Similarity
API	Application Programming Interface

SEZNAM OBRÁZKŮ

SEZNAM TABULEK

SEZNAM ZDROJOVÝCH KÓDŮ

SEZNAM PŘÍLOH

PŘÍLOHA P I: CD

Priložené CD obsahuje:

Bakalársku prácu vo formáte DOCX:	BP_SimonNehez_A21271.docx
Bakalársku prácu vo formáte PDF:	BP_SimonNehez_A21271.pdf
Zdrojové kódy aplikácií:	BP_PRAKTICKA_SimonNehez_A21271.zip