VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
FACULTY OF INFORMATION TECHNOLOGY

ÚSTAV POČÍTAČOVÉ GRAFIKY A MULTIMÉDIÍ
DEPARTMENT OF COMPUTER GRAPHICS AND MULTIMEDIA

KOMUNIKAČNÍ AGENT PRO INFORMACE O BRNĚ

BRNO COMMUNICATION AGENT

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE **JIŘÍ KŘIŠTOF**
AUTHOR

VEDOUCÍ PRÁCE **Doc. RNDr. PAVEL SMRŽ, Ph.D.**
SUPERVISOR

BRNO 2021

Vysoké učení technické v Brně
Fakulta informačních technologií

Ústav počítačové grafiky a multimédií (UPGM) Akademický rok 2020/2021

Zadání bakalářské práce

Student: **Křištof Jiří**
Program: Informační technologie
Název: **Komunikační agent pro informace o Brně**
Brno Communication Agent
Kategorie: Umělá inteligence

Zadání:

1. Seznamte se s existujícími řešeními komunikačních agentů a nástroji pro jejich vytváření
2. Shromážděte data znalostní báze, ze které bude agent čerpat
3. Navrhněte a implementujte systém pro komunikaci s uživatelem v oblasti informací o Brně, aktuálních událostech a dalších informacích.
4. Vyhodnoťte vytvořený systém v testu s prvotními uživateli
5. Vytvořte stručný plakát prezentující práci, její cíle a výsledky

Literatura:

- dle doporučení vedoucího

Pro udělení zápočtu za první semestr je požadováno:

- funkční prototyp řešení

Podrobné závazné pokyny pro vypracování práce viz https://www.fit.vut.cz/study/theses/

Vedoucí práce: **Smrž Pavel, doc. RNDr., Ph.D.**
Vedoucí ústavu: Černocký Jan, doc. Dr. Ing.
Datum zadání: 1. listopadu 2020
Datum odevzdání: 30. července 2021
Datum schválení: 30. října 2020

Abstrakt

Cílem práce je implementace komunikačního agenta poskytujícího informace o Brně. Komunikační agent využívá třívrstvé architektury. Pro vlastní odpovídání na otázky jsou použity techniky strojového učení a neuronových sítí. Na základě provedeného testu bylo se systémem spokojeno 58 % respondentů, s přesností odpovědí poté 84 % uživatelů. Přínosem této práce je usnadnění získávání informací o Brně jeho obyvatelům i návštěvníkům.

Abstract

The aim of this bachelor thesis is the implementation of a communication agent which provides informations about Brno. The communication agent uses three-tier architecture. Machine learning and neural network techniques are used for the question answering. According to user tests the success rate is 84 % and 58 % of the primary users were satisfied with the system. Main benefit of the work is facilitating retrieving of informations about Brno for its residents and visitors.

Klíčová slova

zpracování přirozeného jazyka, odpovídání na otázky, strojové učení, neuronové sítě, klasifikace, získávání informací, třívrstvá architektura

Keywords

natural language processing, question answering, machine learning, neural networks, classification, information retrieval, three-tier architecture

Citace

KŘIŠTOF, Jiří. *Komunikační agent pro informace o Brně*. Brno, 2021. Bakalářská práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce Doc. RNDr. Pavel Smrž, Ph.D.

Komunikační agent pro informace o Brně

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením pana doc. RNDr. Pavla Smrže, Ph.D. Další informace mi poskytl Ing. Martin Fajčík. Uvedl jsem všechny literární prameny, publikace a další zdroje, ze kterých jsem čerpal.

.....
Jiří Křištof
27. července 2021

Poděkování

Rád bych poděkoval vedoucímu práce, panu doc. RNDr. Pavlu Smržovi, Ph.D., a panu Ing. Martinu Fajčíkovi za cenné rady, ochotu a pomoc při tvorbě bakalářské práce.

Obsah

Kapitola 1

Úvod

Díky rozmachu globální sítě internet je dnes získávání informací snadnější, než kdykoliv v historii. Z důvodu velkého množství dat dostupných skrze internet se nedílnou součástí sítě staly vyhledávače informací. V posledních letech můžeme v této oblasti pozorovat významné pokroky, kdy systémy založené na principech umělé inteligence umožňují interakci uživatele s vyhledávačem pomocí lidského jazyka. Díky zdokonalování algoritmů pro rozpoznávání řeči je dále možné komunikovat nejen s pomocí klávesnice, ale i mluveným slovem.

Právě fakt prudkého rozmachu těchto technologií mi byl motivací pro výběr tématu mé bakalářské práce. Mým záměrem bylo vytvořit systém s potenciálem následného každodenního využití. Prostřednictvím praktické části práce byl tento úmysl splněn – byla vytvořena komplexní aplikace komunikačního agenta dostupná na adrese `http://www.stud.fit.vutbr.cz/~xkrist22/` pro město Brno, která je schopna usnadnit život obyvatelům i návštěvníkům města. Pomocí otázek v přirozeném jazyce systém umožňuje zjišťovat informace o objektech města, historii, kultuře, dále vyhledávat spoje hromadné dopravy či poskytovat další užitečné informace, kupříkladu předpověď počasí.

Kapitola 2 obsahuje popis základních principů komunikačních agentů, mechanismus odpovídání na otázky a uvádí další potřebné metody a algoritmy používané v rámci vyvinutého systému. Dále kapitola poukazuje na způsob získávání dat relevantních pro uživatelem položené dotazy. Kapitola 3 se zabývá vlastním návrhem systému – architekturou systému a implementací jednotlivých částí. V kapitole 4 jsou prezentovány použité postupy pro ověření funkčnosti systému a vyhodnocení provedených testů. Kapitola 5 poté obsahuje zhodnocení dosažených výsledků a uvádí další možnosti vývoje agenta.

Kapitola 2

Shrnutí dosavadního stavu

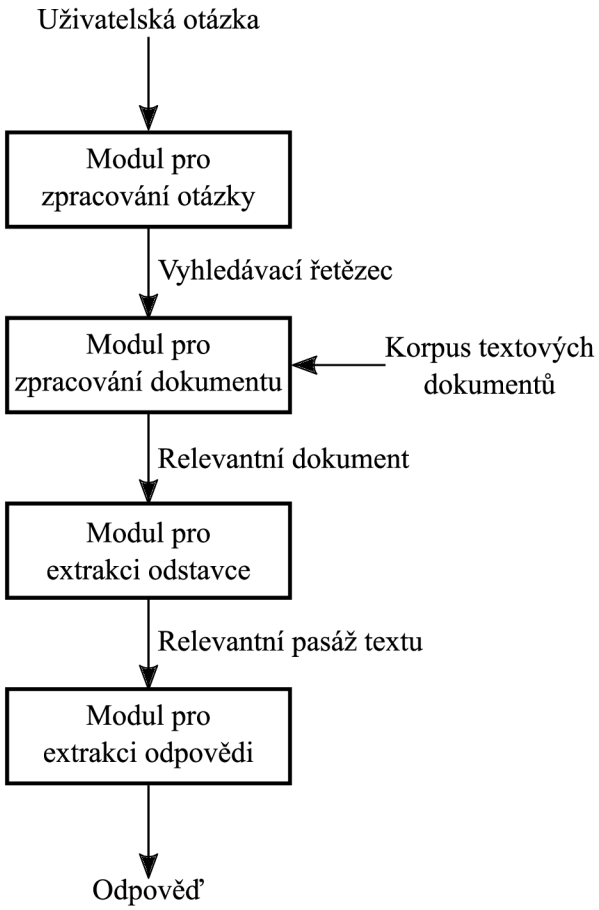
V této kapitole jsou popsány základní aspekty a teoretické principy využívané v rámci systému komunikačního agenta. Text není encyklopedickým přehledem – některé pojmy nemusí být plně vysvětleny, jsou předpokládány znalosti v rozsahu bakalářského studia informačních technologií. Pro případné bližší ujasnění problematiky je nutné dohledat doplňující literaturu. Pokud není uvedeno jinak, je při vysvětlení použit spisovný anglický jazyk.

2.1 Principy systémů pro odpovídání na otázky

Odpovídání na otázky (anglicky question answering) je problematika spadající do oblasti získávání informací. Oblast získávání informací je blíže popsána v sekci 2.7. Úkolem systémů pro odpovídání na otázky (anglicky question answering systems) je nalezení odpovědi na daný uživatelský dotaz v relevantním textovém kontextu. Systémy se dělí na dva hlavní typy dle řešené domény – systémy s **otevřenou** (anglicky open domain systems) a **uzavřenou** (anglicky closed domain systems) doménou. Systémy s doménou uzavřenou umožňují pokládání dotazů z předem limitované řešené problematiky, například lékařské záležitosti či počasí. Tyto systémy typicky využívají předem vytvořenou a nerozšiřující se bázi dat. Naopak systémy s doménou otevřenou dokážou odpovídat na otázky z libovolné oblasti. Tyto obvykle využívají webové vyhledávání dokumentů, čímž není striktně omezena báze dat jako u systémů pracujících s doménou uzavřenou [15].

Systémy odpovídající na uživatelské otázky položené v přirozeném jazyce využívají techniky zpracování přirozeného jazyka (anglicky natural language processing, zkráceně NLP). Obecně je možné tyto systémy rozdělit do čtyř separátních modulů – **modul pro zpracování otázky** (anglicky question processing module), **modul pro zpracování dokumentu** (anglicky document processing module), **modul pro extrakci odstavců** (anglicky paragraph extraction module) a **modul pro extrakci odpovědi** (anglicky answer extraction module). Blokové schéma na obrázku 2.1 znázorňuje tok programu systému a výstupy jednotlivých modulů [15].

Úlohou řešenou modulem pro zpracování otázky je vytvoření vyhledávacího řetězce (anglicky search query) pro systém získávající relevantní dokument (anglicky document retriever). Druhým výstupem modulu je poté typ očekávané odpovědi. Tato informace usnadňuje proces extrakce odpovědi. Výstupy tohoto modulu využívá modul pro zpracování dokumentu. Na základě vyhledávacího řetězce provede modul vlastní vyhledání dokumentu obsahujícího odpověď na položenou otázku z korpusu dat. Korpusem je v této oblasti soubor textových dokumentů určitého lidského jazyka. Modul pro extrakci odstavce se následně



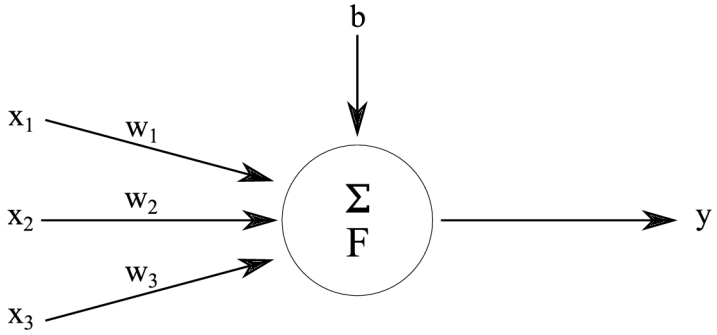
Obrázek 2.1: Blokové schéma modulů systému pro odpovídání na otázky, převzato z [15]

pokouší v poskytnutém dokumentu najít takovou pasáž textu, která obsahuje odpověď na uživatelskou otázku. Tento textový úsek je poté vstupem pro modul extrahující odpověď [15].

2.2 Umělé neuronové sítě

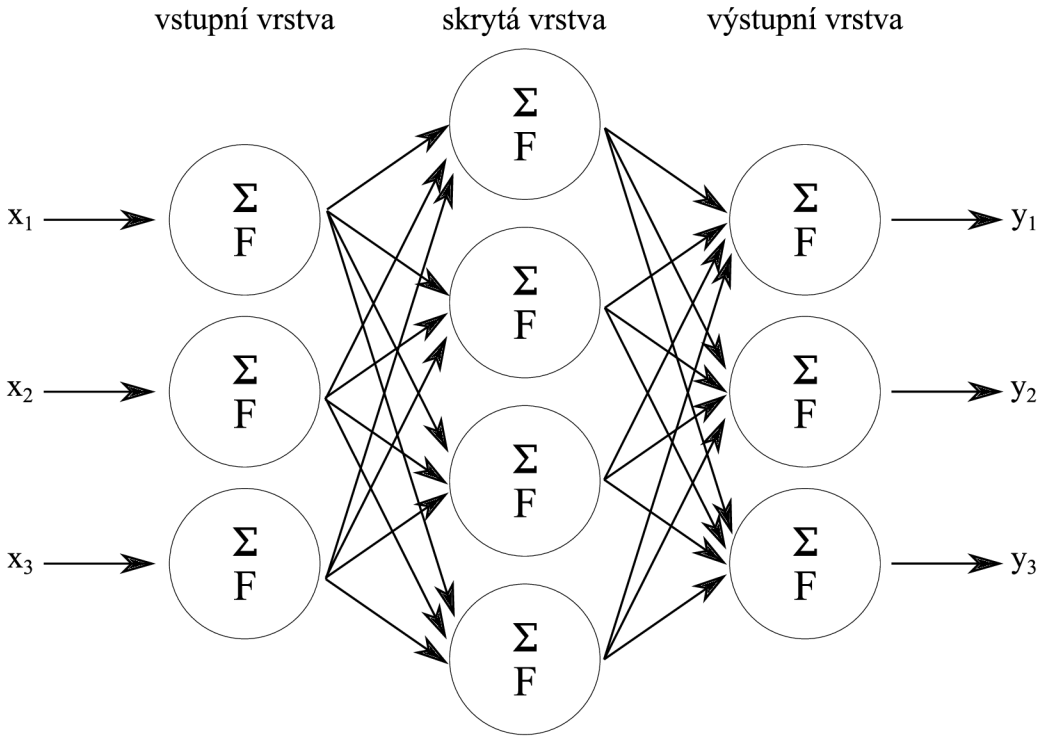
Umělá neuronová síť (anglicky artificial neuron network) je výpočetní systém inspirovaný biologickými vlastnostmi mozku. Neuronové sítě umožňují efektivně řešit problémy v oblasti zpracování přirozeného jazyka, například predikci odpovědi na základě poskytnutého kontextu a otázky. Model neuronové sítě se skládá z **neuronů**, mezi nimiž existují vazby. Obrázek 2.2 ilustruje princip neuronu, vztah 2.1 způsob výpočtu výstupní hodnoty neuronu. Vstupem neuronu jsou číselné hodnoty, které dohromady tvoří **vstupní vektor** $\overrightarrow{x}$ skládající se z n složek. Následně je provedeno vektorové násobení vstupního vektoru a **vektoru vah** $\overrightarrow{w}$. Jednotlivé složky výsledného vektoru jsou poté sečteny a k této hodnotě je přičtena speciální hodnota b nazývaná **bias**. Výsledná hodnota je použita jako vstup pro **aktivační funkci** $F(x)$. Výsledná hodnota aktivační funkce pro danou hodnotu je považována za výstup neuronu y [9].

$$y = F(b + \sum_{i=1}^{n} w_i x_i) \tag{2.1}$$



Obrázek 2.2: Schéma umělého neuronu, převzato z [9]

Dopředná neuronová síť se skládá z jednotlivých neuronů, které jsou organizovány do vrstev. Obrázek 2.3 uvádí ilustrační příklad dopředné neuronové sítě. Vstupní vrstva neuronové sítě slouží pro předání vstupního vektoru $\overrightarrow{x}$ neuronové síti. Následuje minimálně jedna skrytá vrstva neuronů. Jednotlivé výstupy neuronů lokalizovaných ve výstupní vrstvě dohromady tvoří výstupní vektor $\overrightarrow{y}$. Vstupem neuronů ve skrytých vrstvách jsou výstupní hodnoty neuronů předcházející skryté vrstvy, či složky vstupního vektoru $\overrightarrow{x}$ [9].



Obrázek 2.3: Ilustrační příklad dopředné neuronové sítě, převzato z [9]

Trénování neuronové sítě

Před používáním neuronové sítě pro řešení daného problému je nutné provést **trénink sítě**. Podstatou tréninku je určení hodnot jednotlivých vah v rámci sítě tak, aby byla síť schopná korektně predikovat výsledky. Před touto fází je nutné připravit trénovací sadu obsahující trénovací vstupní vektory a očekávaný výstup sítě [9]. Příkladem trénovací sady je datová sada SQuAD určená pro trénování neuronových sítí pro odpovídání na otázky. Datová sada obsahuje kontexty, k nimž se vážou otázky zodpověditelné na základě daného textu, vlastní odpovědi a dodatečné informace [16].

Prvním krokem trénovací fáze je inicializace vah. Každé váze je přiřazena náhodná hodnota s ohledem na určité rozložení. Pro každý trénovací vstup je poté neuronovou sítí generován výstup, který je porovnán vůči očekávanému výstupu [9]. Pro porovnání očekávaného a reálného výstupu jsou používány **nákladové funkce** (alternativně **účelová funkce**, anglicky cost function, alternativně **loss function**) [3]. Příkladem nákladové funkce je **střední kvadratická chyba** e uvedená ve vztahu 2.2, kde y_i je očekávaná výstupní hodnota, $\hat{y}_i$ je skutečná výstupní hodnota a m udává počet složek očekávaného výstupního vektoru a tím i počet složek reálného výstupního vektoru [3].

$$e = \frac{1}{m}\sum_{i=1}^{m}(y_i - \hat{y}_i)^2 \tag{2.2}$$

S využitím hodnoty nákladové funkce a původních hodnot vah je možné provést optimalizaci neuronové sítě. Optimalizací je míněna úprava vah vedoucí k zpřesnění řešení dané úlohy a tím snížení hodnoty nákladové funkce. Pro změnu vah je používán **algoritmus gradientního sestupu** (anglicky gradient descent). Algoritmus umožňuje iterativně pozměňovat hodnoty vah tak, aby se hodnota nákladové funkce přibližovala lokálnímu minimu. Pro tuto změnu je využíván **gradient** – vektor určující směr nejstrmějšího stoupání v daném bodě funkce. Při posunu váhy ve směru vektoru opačného ke gradientu dochází ke snížení hodnoty účelové funkce a tím k optimalizaci sítě [3].

Velikost posunu není konstantní – čím mírnější je pokles funkce, tím menší je velikost posunu. V případě, že není posun prováděn v závislosti na poklesu funkce, nemusí být možné zajistit postupnou konvergenci hodnoty nákladové funkce k minimu, neboť hodnota vah v tomto případě nemusí konvergovat k lokálnímu minimu nákladové funkce [3].

Gradient pro váhu w_t je možné získat s pomocí parciální derivace nákladové funkce E vzhledem k váze w_t. Pro zajištění konvergence je nutné dále určit hodnotu **míry učení** ϵ (anglicky learning rate). Gradient g_t pro váhu w_t je možné určit s použitím vztahu 2.3. Novou hodnotu váhy w_{t+1} je pak možné určit pomocí vztahu 2.4 [3].

$$g_t = -\epsilon\frac{\delta E}{\delta w_t} \tag{2.3}$$

$$w_{t+1} = w_t - \epsilon\frac{\delta E}{\delta w_t} \tag{2.4}$$

Při trénování vícevrstvé neuronové sítě je použit speciální případ algoritmu gradientního sestupu – **algoritmus zpětného šíření chyby** (anglicky backpropagation). Prvním krokem je, shodně s předchozím popisem, použití trénovacích vstupů a určení výstupu neuronové sítě. S použitím reálného a očekávaného výstupu je určena hodnota nákladové funkce. Následně je postupně od výstupní vrstvy po první skrytou vrstvu pro každou váhu určen gradient. Při určování gradientu vah ve vrstvách následujících po vrstvě výstupní

je využito řetězové pravidlo pro derivování složené funkce. Následně je pro každou váhu určena její nová hodnota dle vztahu 2.4. Tento postup je opakován pro veškeré trénovací vstupní vektory [9].

Přehled aktivačních funkcí

Následující text se zabývá vybranými aktivačními funkcemi používanými neuronovými sítěmi. První uvedenou aktivační funkcí je **logistická funkce sigmoida** (anglicky logistic sigmoid function). Oborem hodnot této funkce je interval $\langle 0, 1\rangle$. Hodnotu logistické funkce σ pro x je možné vyjádřit pomocí vztahu 2.5 [11].

$$\sigma(x) = \frac{1}{1 + e^{x}} \tag{2.5}$$

Další používanou aktivační funkcí je **hyperbolický tangens**. Obor hodnot této funkce náleží do intervalu $\langle -1, 1\rangle$. Negativní vstupy jsou mapovány na negativní výstupní hodnotu, obdobně pozitivní vstupní hodnoty jsou mapovány na pozitivní výstup. Hodnota funkce hyperbolického tangensu tanh pro x je dána vztahem 2.6 [11].

$$\tanh(x) = \frac{\sinh(x)}{\cosh(x)} = \frac{1 - e^{-2x}}{1 + e^{-2x}} \tag{2.6}$$

ReLU (anglicky plně Rectified Linear Unit) je aktivační funkce, která mapuje vstupní hodnoty do polootevřeného intervalu $\langle 0, \infty)$. Tato funkce je využívána především z důvodu eliminace problému mizejícího gradientu (anglicky vanishing gradient) [11]. Problém mizejícího gradientu algoritmu zpětného šíření chyby je problém, kdy učení vrstev vzdálenějších od výstupní vrstvy je pomalejší z důvodu použití řetězového pravidla derivace složené funkce. Dochází k postupnému snižování hodnoty gradientu a nižšímu kroku při gradientním sestupu [9]. Definice funkce ReLU r je uvedena vztahem 2.7 [11].

$$r(x) = \begin{cases} 0 & \text{pro } x < 0 \\ x & \text{pro } x \geq 0 \end{cases} \tag{2.7}$$

Aktivační funkce ReLU je problematická z důvodu nulové hodnoty pro záporná čísla. Pakliže je tato hodnota použita při gradientním sestupu, je velikost kroku gradientního sestupu rovna 0 a váha tak zůstává nezměněna. Pro eliminaci tohoto problému je možné využít funkci **Leaky ReLU**. Pro záporné vstupy je navrácena nenulová hodnota, čímž je vyřešen výše zmíněný problém. Vztah 2.8 uvádí definici funkce Leaky ReLU r_l pro hodnotu x, kde α je hodnota udávající strmost funkce pro $x < 0$ [11].

$$r_l(x) = \begin{cases} \alpha x & \text{pro } x < 0 \\ x & \text{pro } x \geq 0 \end{cases} \tag{2.8}$$

Transformers

Transformers je architektura neuronových sítí používající mechanismus **self-attention**. Tento mechanismus umožňuje učení souvislostí mezi jednotlivými vstupy modelu. Vlastní model se sestává z **enkodéru** (anglicky encoder) a **dekodéru** (anglicky decoder). Schéma architektury sítě transformers je uvedena na obrázku 2.4 [17].

Vstupem enkodéru jsou vstupní vektory a výstupem jsou interní vektorové reprezentace vstupů. Enkodér se dále dělí na dvě komponenty – hlavu realizující mechanismus

self-attention a dopřednou neuronovou síť používající aktivační funkci ReLU. Self-attention je mechanismus, pomocí něhož je možné detekovat závislosti mezi jednotlivými prvky, například mezi jednotlivými slovy věty, respektive mezi vektory slov vytvořených z původních slov. Princip tvorby vektorů ze slov je popsán v sekci 2.3 [17].

Vstupem hlavy realizující mechanismus self-attention je sekvence vektorů. Tyto vektory je předem nutné upravit tak, aby jejich hodnoty reflektovaly původní pozici v sekvenci. Tento problém řeší **poziční kódování** (anglicky positional encoding). Ke každému vstupnímu vektoru je vytvořen poziční vektor, jehož složky jsou dány funkcemi 2.9, kde r je pozice vstupního vektoru v sekvenci, e je pozice zkoumané složky pozičního vektoru a d_x definuje počet složek vstupního vektoru a tím i vektoru pozičního. Po vytvoření pozičního vektoru pro vstupní vektor jsou tyto sečteny a dále používány mechanismem self-attention [17].

$$\begin{aligned} P_{r,2e} &= \sin\left(\frac{r}{1000^{\frac{2e}{d_x}}}\right) \\ P_{r,2e+1} &= \cos\left(\frac{r}{1000^{\frac{2e}{d_x}}}\right) \end{aligned} \tag{2.9}$$

Na základě vstupních vektorů je vytvořena matice X, kde jednotlivé řádky odpovídají původním vektorům. Z této matice jsou s pomocí **matic vah** W_Q, W_K a W_V vytvořeny **matice dotazů** Q (anglicky query matrix), **matice klíčů** K (anglicky key matrix) a **matice hodnot** (anglicky value matrix). Hodnoty matic vah jsou inicializovány náhodně dle předem daného rozložení a jsou optimalizovány během fáze tréninku. Vztahy pro výpočet matic Q, K a V popisují vzorce 2.10 [17].

$$\begin{aligned} Q &= X \cdot W_Q \\ K &= X \cdot W_K \\ V &= X \cdot W_V \end{aligned} \tag{2.10}$$

Maticovým součinem $Q \cdot K^T$ získáváme **matici skóre** S vyjadřující souvislosti mezi jednotlivými vstupními vektory. Prvky matice S jsou následně vyděleny hodnotou druhé odmocniny dimenze matice dotazů. Tato operace eliminuje problém **explodujícího gradientu** (anglicky exploding gradient) – exponenciálnímu růstu gradientu vedoucího k nestabilní síti. Následně je použita funkce softmax, zobecnění logistické funkce pro více dimenzí. Po aplikaci funkce náleží hodnoty prvků matice S do intervalu $\langle 0, 1\rangle$. Následuje maticový součin $S \cdot V$. Z matice je vytvořen vektor spojením řádků a tento je předán dopředné neuronové síti. Výsledný vektor je rozdělen na vektory o počtu složek d_x [17].

Mechanismus **multi-head attention** využívá pro určení výstupních vektorů více na sobě nezávislých hlav realizujících self-attention. Vstupem každé hlavy je část vstupních vektorů. Výstupní vektory jednotlivých hlav jsou před použitím neuronovou sítí konkatenovány a tyto pak násobeny vektorem vah W_o [17].

Po získání vektorů mechanismem multi-head attention je tento sečten s původním vstupním vektorem. Vektor získaný tímto vektorovým součtem je poté normalizován. Obdobně je k výstupnímu vektoru dopředné neuronové sítě přičten vektor získaný mechanismem multi-head attention a je provedena normalizace výsledného vektoru. Tento mechanismus zefektivňuje fázi trénování. Vrstva realizující normalizaci zabraňuje přílišným změnám hodnot vektoru [17].

Model Transformers může využívat více enkodérů, kde první enkodér využívá vstupních vektorů a další poté výstupní vektory enkodéru předcházejícího. Výstup posledního enkodéru je považován za interní reprezentaci vektorů a je použit jako jedna ze vstupních složek dekodéru. Dále uvažujme diskrétní čas t. Vstupem dekodéru jsou také již predikované vektory. V čase $t = 0$ je namísto predikovaného vektoru použit předem daný specifický vektor. V čase $t = 1$ jsou vstupem krom interní reprezentace vstupních vektorů také počáteční vektor a výstupní vektor získaný v čase $t = 0$. Predikce dalších výstupních vektorů končí při získání specifického předem daného koncového vektoru. Princip mechanismu multi-head attention je obdobný jako pro enkodér. Změnou je nutnost použití **maskování**, které pro vektory, které v daném čase t neexistují, nahradí hodnotu skóre za hodnotu $-\infty$. Maskování je provedeno před aplikací funkce softmax. Tato mapuje hodnotu $-\infty$ na hodnotu 0 – model tak nebere v potaz dosud neexistující vektory. Tento mechanismus je nazýván **masked multi-head attention**. Obdobně jako pro komponentu enkodér je možné použití více dekodérů, kdy vstupem dalších je výstup původního dekodéru a interní reprezentace vstupních vektorů [17].

Model BERT

BERT (anglicky plně Bidirectional Encoder Representations from Transformers) je model založený na architektuře modelu transformers, přičemž využívá pouze komponentu enkodér. Trénování je založeno na **predikci maskovaného slova** (anglicky masked language modeling) a **predikci další věty** (anglicky next sentence prediction). Principem predikce maskovaného slova je určení následujícího slova na základě sekvence slov. Trénování s využitím predikce další věty je způsob učení, kdy vstupem modelu jsou dvě věty a úkolem je určení, zda druhá věta významově následuje první, či nikoliv [17].

Při využití BERT pro odpovídání na uživatelské otázky je vstupem otázka a text obsahující odpověď. Výstupem je počátek a konec části textu, který odpovídá na danou otázku. Tento problém je řešen v několika krocích. Prvním krokem je určení pravděpodobnosti P_s^i, že vybrané slovo i je počátek odpovědi. Tuto pravděpodobnost je nutné vyjádřit pro všechna slova kontextu. Následně je vybráno slovo i s nejvyšší pravděpodobností P_s^i. Obdobně je určeno slovo, které s nejvyšší pravděpodobností P_e^i uzavírá odpověď. Slova jsou v rámci modelu reprezentována vektory R. Tvorba vektorů slov je popsána v sekci 2.3. Schéma modelu je uvedeno na obrázku 2.5 [17].

Mějme vektor S definující vektor slova začínajícího odpověď a vektor E definující vektor slova, jímž odpověď končí. Hodnoty složek vektorů S a E jsou určeny během fáze tréninku. Způsob výpočtu pravděpodobnosti $P_s^(i)$ a $P_e^(i)$ pro slovo i je dán vztahem 2.11 a 2.12, kde R_i je vektorová reprezentace slova i a n je počet slov kontextu.

$$P_s^i = \frac{e^{S \cdot R_i}}{\sum_{j=1}^{n} e^{S \cdot R_j}} \tag{2.11}$$

$$P_s^i = \frac{e^{E \cdot R_i}}{\sum_{j=1}^{n} e^{E \cdot R_j}} \tag{2.12}$$

2.3 Způsoby reprezentace textu lidského jazyka

Pro umožnění strojového zpracovávání dokumentů zapsaných v přirozeném jazyce je nutné nejprve vytvořit takovou reprezentaci textu, kterou je dále možné počítačově zpracovávat. Při řešení problémů v oblasti zpracování přirozeného jazyka mohou používané postupy využívat různé způsoby reprezentace textů v lidském jazyce, nad nimiž je možné aplikovat metody vedoucí k řešení dané úlohy.

Jednou z používaných metod pro vytvoření strojové reprezentace textu je **tokenizace** (anglicky tokenization) – proces, při němž je větší textová jednotka převáděna na uspořádanou množinu menších textových jednotek odpovídajících slovům původního textu nazývaných **tokeny**. Tento postup je částečně jazykově závislý, neboť různé přirozené jazyky mohou využívat nejen různá typografická a stylistická pravidla (typicky se jedná o používání uvozovek a dalších interpunkčních znamének), ale i různé systémy zápisu jazyka [2].

Popisovaná metoda pracuje s principy výstavby textů lidského jazyka, jako oddělování vět interpunkčním znaménkem „tečka“, oddělování slov bílými znaky (mezera či tabulátor) či jinými. Konkrétní příklad tokenizace je uveden v tabulce 2.1. Algoritmy realizující proces tokenizace musí řešit i případné alternativní využití interpunkčních znamének, například znaménko „tečka“ je možné využít při oddělování cifer čísla či zkracování slov. Tokenizace je speciálním případem segmentace. Segmentace je metoda, při níž je původní textová jednotka dělena na menší jednotky, které nemusí nutně odpovídat jednotlivým slovům. Typickým příkladem je vytvoření uspořádané množiny vět z původního textu [2].

Po vytvoření uspořádané množiny tokenů T z původního textového dokumentu je možné z množiny T či její podmnožiny utvořit množinu n-gramů. **N-gram** je posloupnost n po sobě jdoucích tokenů z množiny T. Obsahuje-li určitý n-gram právě dva prvky, pak je nazýván bigram. V případě n-gramu obsahujícího právě tři prvky mluvíme o trigramu. Pro n-gramy o čtyřech a více prvcích je využíván zápis n-gram, kde n symbolizuje celkový počet prvků daného n-gramu [2]. V tabulce 2.1 je uveden příklad množiny bigramů.

Původní dokument	*The cat's on the hot tin roof*
Tokeny	{"*The*", "*cat*", "*'s*", "*on*", "*the*", "*hot*", "*tin*", "*roof*"}
Bigramy	{{"*The*", "*cat*"}, {"*cat*", "*'s*"}, {"*'s*", "*on*"}, {"*on*", "*the*"}, {"*the*", "*hot*"}, {"*hot*", "*tin*", }{"*tin*", "*roof*"}}

Tabulka 2.1: Přehled textových způsobů reprezentací dokumentů v přirozeném jazyce

Jedním z dalších způsobů reprezentace slov jsou **vektory slov** (anglicky word embedding). Existuje více způsobů, jak vytvořit vektorovou reprezentaci původního slova. Jednou z nejjednodušších metod jsou **one-hot vektory**. Tyto vektory jsou vytvářeny na základě předem existujících trénovacích dat. Z těchto dat je vytvořena množina S unikátních slov, kde $|S|$ označuje celkový počet unikátních slov. One-hot vektory obsahují celkem $|S|$ parametrů. Veškeré parametry jsou rovny hodnotě 0, vyjma jediného mající hodnotu 1. Takto je možné těmito vektory indexovat veškerá unikátní slova. [6]

Pro praktické využití jsou one-hot vektory problematické – nijak nezachycují podobnost slov a potencionálně mohou mít velké množství parametrů. Tyto nedostatky eliminují vektory slov vytvořené s pomocí algoritmu Word2vec. Tento přístup využívá neuronové sítě představené v kapitole 2.2. Pro trénování je možné použít dvě základní metody – **Continuous Bag of Words** (zkráceně CBOW) a **Skip Gram**. [6, 12]

Trénování neuronové sítě dle metody CBOW je postaveno na predikci cílového slova pro daný kontext. Kontext pro cílové slovo je vytvořen z původních trénovacích vět pomocí

okénka. Okénko má fixní velikost definující počet slov kontextu cílového slova. Cílové slovo je poté prostřední slovo okénka. Vstupem neuronové sítě jsou one-hot vektory slov vybraných pomocí okénka, vyjma slova cílového. Pro trénování prováděné na základě algoritmu zpětného šíření chyby, popsaného v 2.2, je použit jako očekávaný výstup sítě one-hot vektor cílového slova. [6]

Při získávání vektorů slov jsou tyto získány s pomocí vah dané neuronové sítě. Tyto váhy tvoří matici, kde počet řádků odpovídá hodnotě $|S|$ a počet sloupců je dán počtem skrytých vrstev sítě. Počet skrytých vrstev je zvolen na základě požadovaného počtu složek vektorů slov. Jednotlivé hodnoty prvků poté odpovídají váze uvedené pro daný řádek a skrytou vrstvu. Výsledný vektor slova je pak dán výsledkem maticového součinu matice vah a one-hot vektoru daného slova. [6, 12]

Při použití metody Skip gram pro trénování neuronové sítě je vstupem one-hot vektor prostředního slova okénka. Očekávaným výstupem jsou one-hot vektory slov kontextu. Počet výstupních one-hot vektorů je dán velikostí okénka. Trénování probíhá obdobně jako s využitím přístupu CBOW pomocí algoritmu zpětného šíření chyby. Stejným způsobem také probíhá získávání vektorů slov. [6, 12]

2.4 Využívané operace nad reprezentacemi dokumentů lidského jazyka

Nad možnými reprezentacemi dokumentů v lidském jazyce zmíněných v sekci 2.3 je možné provádět řadu operací, které jsou využívány při získávání odpovědí na uživatelské otázky. Níže uvedené postupy mohou využívat rozdílné reprezentace textových dokumentů. Přehled a popis využívaných reprezentací je blíže popsán v sekci 2.3.

Levenshteinova vzdálenost

Levenshteinova vzdálenost (alternativně **editační vzdálenost**, anglicky Levenshtein distance) slouží pro vyjádření míry podobnosti dvou textových řetězců. Podstata výpočtu vzdálenosti je dána postupným převodem jednoho textového řetězce na druhý, přičemž výchozí řetězec může být transformován do podoby konečného řetězce s využitím tří operací – odebráním, přidáním, či změnou znaku. Každá z těchto operací disponuje svými ohodnoceními, která mohou i nemusí být shodná. Všechny operace jsou typicky ohodnoceny číselnou hodnotou 1. Editační vzdálenost je poté dána součtem ohodnocení všech operací, které bylo nutné provést pro převod výchozího textového řetězce na druhý, konečný [21].

Pro korektní vyjádření míry podobnosti dvou textových řetězců je nutné, aby Levenshteinova vzdálenost používaná pro vyjádření podobnosti byla minimální. **Minimální editační vzdálenost** lze získat pomocí k tomu určené matice L, jejíž prvky jsou postupně vyplňovány podle předem daných pravidel. Mějme dva textové řetězce t_1 a t_2. Označme pak počet znaků těchto řetězců jako $|t_1|$, respektive $|t_2|$. Matice L pro řetězce t_1 a t_2 bude mít rozměr $|t_1|+1 \times |t_2|+1$, případně $|t_2|+1 \times |t_1|+1$. První sloupec matice L bude poté od shora obsahovat hodnoty $\{0, 1, \ldots, |t_1|+1\}$, respektive $\{0, 1, \ldots, |t_2|+1\}$. První řádek bude obdobně obsahovat zleva hodnoty $\{0, 1, \ldots, |t_2|+1\}$, popřípadě $\{0, 1, \ldots, |t_1|+1\}$ pro druhou variantu [21].

Dále uvažujeme pouze matici o rozměrech $|t_1|+1 \times |t_2|+1$. Jednotlivé prvky matice, kterým dosud nebyla určena hodnota, jsou nahrazovány shora po řádcích, přičemž každý prvek $L_{x,y}$, kde $x \in \{1, \cdots, |t_1|+1\}$ a $y \in \{1, \cdots, |t_2|+1\}$, je dán nejmenší hodnotou prvků $\{L_{x-1,y}, L_{x,y-1}, L_{x-1,y-1}\}$ inkrementovaných o hodnotu 1 v případě, že x-tý znak jednoho

řetězce a y-tý znak druhého řetězce je rozdílný. V případě shodného x-tého znaku jednoho řetězce a y-tého znaku druhého řetězce je hodnota prvku $L_{x,y}$ dána přímo hodnotou prvku $L_{x-1,y-1}$. Výsledná minimální editační vzdálenost l textových řetězců t_1 a t_2 je poté dána prvkem $L_{|t_1|+1,|t_2|+1}$ [21].

Pro ilustraci uveďme příklad: mějme dva textové řetězce *foggy* a *gym*. Pro tyto dva textové řetězce vypočítejme minimální Levenshteinovu vzdálenost dle výše uvedených pravidel. Prvním krokem je sestrojení matice o 4 řádcích a 6 sloupcích. První řádek a sloupec vyplníme výše uvedeným způsobem, každý další prvek poté získáme aplikováním výše zmíněných pravidel. Po sestrojení matice 2.13 označíme za minimální editační vzdálenost řetězců foggy a gym prvek ležící na čtvrtém řádku v šestém sloupci [21].

$$\begin{array}{c} \\ \text{' '} \\ \text{g} \\ \text{y} \\ \text{m} \end{array} \begin{array}{c} \begin{array}{cccccc} \text{' '} & \text{f} & \text{o} & \text{g} & \text{g} & \text{y} \end{array} \\ \begin{pmatrix} 0 & 1 & 2 & 3 & 4 & 5 \\ 1 & 1 & 2 & 2 & 3 & 4 \\ 2 & 2 & 2 & 3 & 3 & 3 \\ 3 & 3 & 3 & 3 & 4 & 4 \end{pmatrix} \end{array} \tag{2.13}$$

Pro samotné vyhodnocení podobnosti je Levenshteinova vzdálenost nedostatečná, neboť může nabývat hodnot libovolného přirozeného čísla či hodnoty 0. Pro určení podobnosti s řetězců t_1 a t_2 je možné využít vztahu 2.14 využívající značení zavedené v textu výše. Funkce max vybírá větší z hodnot. Výsledná hodnota vždy spadá do intervalu $\langle 0, 1 \rangle$, díky čemuž je možné procentuálně určit míru podobnosti slov [21].

$$s = 1 - \frac{l}{\max(|t_1|, |t_2|)} \tag{2.14}$$

Stematizace

Stematizace (anglicky stemming) je proces, při němž je slovo převáděno na základní tvar (anglicky stem) – část slova, která je shodná i pro další slova získaná z původního termínu pomocí ohýbání. Pro stematizaci slov anglického jazyka je možné využít deterministického algoritmu **Porter Stemming**. Algoritmus v několika krocích prochází slovo, jenž má být převedeno na stem. Pro každý krok jsou definována přepisovací pravidla ve tvaru $s_1 \rightarrow s_2$, kde s_1 je původní přípona a s_2 je přípona, která původní sufix s_1 nahrazuje. Přepisovací pravidlo je možné aplikovat pouze v případě shody přípony zkoumaného slova a přípony s_1 daného pravidla. Pokud není možné aplikovat žádné pravidlo, původní slovo zůstává nezměněno a je proveden další krok. Pakliže došlo ke změně slova, je v dalším kroku používáno modifikované slovo. V rámci každého kroku může být aplikováno nanejvýš jedno pravidlo. Pokud může být v jednom kroku aplikováno více pravidel, pak je vybráno pravidlo modifikující největší část daného slova. Výsledná modifikace slova po všech krocích je označena za stem původního výrazu. Stem slova nemusí mít v lidském jazyce žádný význam, jelikož při stematizaci dochází pouze k postupnému odstraňování sufixů. [1]

Obdobou stematizace je **lemmatizace** (anglicky lemmatisation) – proces, při němž je k určitému slovu přiřazena **lemma** – základní tvar slova uváděný ve slovnících přirozených jazyků a mající význam v lidském jazyce. Lemmatizace je typicky časově a paměťově složitější proces než stematizace, protože je nutné o slově získat dodatečné informace či je nutné slovo vyhledat v databázi. [2]

Kosinová podobnost vektorů slov

Máme-li vektory slov, případně vektory vět či celých dokumentů, pak lze určit procentuální míru podobnosti těchto vektorů s pomocí k tomu určených metrik jako **kosinová podobnost** (anglicky cosine similarity). Určení podobnosti dvou vektorů s využitím tohoto matematického principu je založeno na určení úhlu φ, který svírají dva porovnávané vektory $\overrightarrow{x}$ a $\overrightarrow{y}$, a následného určení kosinu úhlu φ. Princip tvorby vektorů z původních textových dokumentů je popsán v sekci 2.3. [5]

Kosinová podobnost dvou vektorů je definována v intervalu $\langle -1, 1 \rangle$, kde hodnota 1 značí úplnou shodu těchto vektorů a hodnota -1 značí opačný směr porovnávaných vektorů. Čím více se blíží hodnota kosinové podobnosti vektorů slov hodnotě 1, tím více jsou si původní slova reprezentovaná těmito vektory podobná z hlediska vektory kódovaných vlastností. Úhel φ svíraný mezi vektory $\overrightarrow{x}$ a $\overrightarrow{y}$ lze získat pomocí vztahu 2.15. Vlastní kosinová podobnost s vektorů $\overrightarrow{x}$ a $\overrightarrow{y}$ svírajících úhel φ je možné vyjádřit jako hodnotu kosinu vektoru φ. Jelikož jsou funkce cos a arccos inverzní, je možné pro samotný výpočet kosinové podobnosti použít pouze skalární součin vektorů $\overrightarrow{x}$ a $\overrightarrow{y}$ dělený součinem velikostí vektorů $\overrightarrow{x}$ a $\overrightarrow{y}$. Pro výpočet kosinové podobnosti je možné aplikovat vztah 2.16. [5]

$$\varphi = \arccos \frac{\overrightarrow{x} \cdot \overrightarrow{y}}{|\overrightarrow{x}| \cdot |\overrightarrow{y}|} \tag{2.15}$$

$$s \ = \cos \varphi = \frac{\overrightarrow{x} \cdot \overrightarrow{y}}{|\overrightarrow{x}| \cdot |\overrightarrow{y}|} \tag{2.16}$$

Četnost slova v dokumentu, převrácená četnost slova v dokumentech a negativní slovník

Tato sekce pojednává o metodách pro automatickou extrakci **významných slov** (alternativně **klíčových**, anglicky keywords). V rámci této sekce předpokládáme existenci implementované metody tokenizace. Tokeny získané ze zkoumaného dokumentu jsou v této sekci pro zjednodušení nazývány slova.

Jedním z postupů je použití **negativního slovníku** (alternativně **slovník stop-slov**, anglicky stop-words). Negativní slovník obsahuje slova, která mají pro daný kontext nízký význam. Typicky se jedná o spojky, předložky či determinátory. Slova patřící mezi tyto slovní druhy mají obecně pro libovolný kontext vždy nízkou významnost [19].

Při extrakci klíčových slov z dokumentu zapsaného pomocí lidského jazyka s využitím negativního slovníku jsou tak slova, která se vyskytují jak v dokumentu, tak ve slovníku, klasifikována jako nevýznamná. Naopak slova vyskytující se pouze ve zkoumaném dokumentu, nikoliv ve slovníku stop-slov, jsou kategorizována jako významná. Negativní slovník může být vytvořen buď ručně před samotným zahájením klasifikace významnosti, nebo s využitím jiných metod hodnocení významnosti slov pro daný kontext [19].

Výše zmíněný algoritmus extrakce klíčových slov je pak typicky doplněn další metodou klasifikace významnosti slov. Jedním z těchto postupů je hodnocení významnosti slov na základě **metriky tf-idf** (zkratka pojmů term frequency a inverse document frequency). Metrika každému hodnocenému slovu t z dokumentu d přiřazuje číselnou hodnotu v intervalu $\langle 0, 1 \rangle$, která reprezentuje významnost daného slova v uvedeném dokumentu – vyšší hodnota značí vyšší významnost slova. Pro výpočet hodnoty významnosti s využitím metriky tf-idf je nutná existence databáze dokumentů D [20].

Vlastní výpočet hodnoty významnosti pro slovo $t \in d$ je dán součinem dvou hodnot – **četnosti slova ve zkoumaném dokumentu** (anglicky term frequency, značení t_{tf}) a **převrácené četnosti slova ve všech dokumentech** v databázi D (anglicky inverse document frequency, značení t_{idf}). Dále jsou pro výpočet využívány tyto hodnoty:

- počet slov zkoumaného dokumentu d: $|d|$,
- počet výskytů slova t ve zkoumaném dokumentu d: $|\{t \in d\}|$,
- počet dokumentů obsažených v databázi D: $|D|$,
- počet dokumentů c v databázi D, jenž obsahují slovo t: $|\{t \in c; c \in D\}|$.

Výpočet hodnoty významnosti slova pro slovo $t \in d$ je poté dán vztahem 2.17 [20], kde t_{tf} je četnost slova t ve zkoumaném dokumentu d a t_{idf} je převrácená četnost slova t ve všech dokumentech databáze D.

$$t_{tf} \cdot t_{idf} = \frac{|\{t \in d\}|}{|d|} \cdot \log \frac{|D|}{|\{t \in c; c \in D\}|} \tag{2.17}$$

Výsledná významnost v_t slova t s využitím výše popsaných principů je pak dána vztahem 2.18, kde proměnná t_s může nabývat pouze hodnot z množiny $\{0, 1\}$. Proměnná t_s nabývá pro slovo t hodnoty 0 v případě, že se zkoumané slovo t nachází v negativním slovníku, nebo hodnoty 1, pokud se zkoumané slovo t v negativním slovníku nevyskytuje [20].

$$v_t = t_{tf} \cdot t_{idf} \cdot t_s \tag{2.18}$$

2.5 Textové klasifikátory

Klasifikace textů (alternativně **kategorizace**, anglicky text classification, obecněji sequence classification) je úloha, při níž jsou textová data kategorizována na základě jejich vlastností do **tříd** (alternativně **kategorie**, anglicky classes). Dle aplikace klasifikátoru je poté možné rozdělit využívané postupy do dvou skupin. První skupina zahrnuje klasifikátory, které textový dokument vždy **kategorizují do právě jedné kategorie** (anglicky single-category text classification). Druhá skupina algoritmů textová data **klasifikuje do jedné a více skupin** (anglicky multi-category text classification). Tento text se dále zabývá single-category klasifikátory. Tyto lze dále řadit do dvou odlišných podtypů, binární a multinomiální. **Binární klasifikátor** přiřazuje zkoumanému dokumentu právě jednu ze dvou předem daných tříd. Oproti tomu **klasifikátory multinomiální** mohou pro zkoumaný text vybrat ze tří a více kategorií [10].

Blokové schéma 2.6 prezentuje způsob tvorby a využití klasifikátoru na základě principů strojového učení s učitelem. Trénovací data pro klasifikátor jsou dvojice vzniklé kartézským součinem $D \times C$. Množina D obsahuje textové dokumenty $\{d_1, d_2, \cdots, d_i\} \in D$, přičemž hodnota i definuje počet dokumentů množiny D. Množina C obsahuje předem definované třídy $\{c_1, c_2, \cdots, c_j\} \in C$, přičemž hodnota j představuje celkové množství definovaných kategorií. Kategorie jsou symbolická označení definovaná uživatelem klasifikátoru, je tak možné použít libovolné označení kategorií [10].

Trénovací data je nutné před vlastním zahájením učení modelu předzpracovat – je nutné odebrat takové dvojice, které byly vytvořeny autorem trénovací sady chybně či nejsou vhodné z jiných důvodů. Dalším krokem předzpracování trénovacích dat může být úprava

dat určených pro učení – například provedení tokenizace textu či odebrání slov nevýznamných z pohledu daného klasifikátoru [10].

Následuje vlastní učení modelu dle předem vybraného algoritmu. V této sekci je dále popsán způsob trénování modelu s využitím principů **naivní bayesovské klasifikace** (anglicky naive bayes classification). Po dokončení strojového učení je možné využít vzniklý model ke klasifikaci nových dokumentů dle principů využitého algoritmu při tvorbě klasifikátoru. Pro určení kvality vzniklého klasifikačního systému je možné využít evaluačních technik, které jsou blíže popsány v sekci 2.6 [10].

Naivní Bayesovský klasifikátor používá pro klasifikaci **Bayesovu větu** 2.21. Ve fázi strojového učení je analyzována trénovací sada T obsahující dvojice skládající se z textového dokumentu a tomuto přiřazené třídy C_i náležící do množiny tříd C, přičemž $i \in \{1, \cdots, |C|\}$, kde $|C|$ značí kardinalitu množiny C. Pro každou třídu $C_i \in C$ je vytvořena množina F_i, která obsahuje dvojice (t, f), kde t_j je v množině F_i unikátní token a f_j je relativní frekvence výskytu tokenu t_j v textových dokumentech náležících třídě C_i. Výpočet relativní frekvence f_j je dán vztahem 2.19, kde $|t_j|$ značí celkový počet výskytu tokenu v trénovacích dokumentech náležících třídě C_i a $|T_c|$ je celkový počet tokenů získaných z dokumentů náležících třídě C_i. Pro každou třídu C_i je dále vyjádřena hodnota apriorní pravděpodobnosti $P(C_i)$ pomocí vztahu 2.20, kde $|T_{c_i}|$ udává celkový počet textových dokumentů v sadě T náležících třídě C_i a $|T|$ je celkový počet všech dokumentů kategorizovaných do libovolné třídy z množiny C [13].

$$P(C_i) = \frac{|t_j|}{|T_c|} \tag{2.19}$$

$$P(C_i) = \frac{|T_{C_i}|}{|T|} \tag{2.20}$$

Pro vlastní klasifikaci je použita Bayesova věta 2.21, která udává pravděpodobnost, že zkoumaný textový dokument náleží do třídy $C_i \in C$ za předpokladu pozorování určitých vlastností F zkoumaného dokumentu. Tato pravděpodobnost je vypočítána pro veškeré třídy náležící do množiny tříd C a jako třída zkoumaného dokumentu je označena ta, pro níž je hodnota pravděpodobnosti $P(C_i|F)$ největší [13].

$$P(C_i|F) = \frac{P(F|C_i) \cdot P(C_i)}{P(F)} \tag{2.21}$$

Jelikož je hodnota pravděpodobnosti $P(C_i|F)$ použita pro porovnávání a samotná hodnota pravděpodobnosti není podstatná, je možné zanedbat jmenovatel, který je při výpočtu podmíněné pravděpodobnosti $P(C_i|F)$ konstantní pro veškeré třídy z množiny C. Sledovanými vlastnostmi jsou frekvence jednotlivých tokenů pro danou třídu. Před zahájením klasifikace je tak nutné provést tokenizaci zkoumaného dokumentu a pro každý token určit relativní frekvenci tokenu, s níž se vyskytuje v dokumentech dané třídy C_i. Relativní frekvence tokenů pro třídu C_i je možné dohledat s pomocí množiny F_i [13].

Předpokladem této metody je, že sledované vlastnosti, relativní frekvence výskytů tokenů, jsou na sobě nezávislé. Pravděpodobnost $P(F|C_i)$ pro třídu C_i je tak možné definovat jako součin relativních frekvencí tokenů. Místo vztahu 2.21 je možné díky výše popsaným úpravám pro porovnání a tím určení nejpravděpodobnější třídy zkoumaného dokumentu použít vztah 2.22, kde $|T|$ je celkový počet tokenů získaných ze zkoumaného dokumentu a $f_k \in F_i$ je relativní frekvence k-tého tokenu pro třídu C_i [13].

$$P(C_i|F) = P(C_i) \cdot \prod_{k=1}^{|T|} f_k \tag{2.22}$$

Problémem, který musí Bayesovský klasifikátor řešit, je výskyt tokenu ve zkoumaném dokumentu, pro který neexistuje hodnota relativní frekvence f_n v množině F_i pro zkoumanou třídu $C_i \in C$. V těchto případech je frekvence f_n rovna hodnotě 0 a výsledná hodnota pro porovnávání pravděpodobnosti, že zkoumaný dokument náleží třídě C_i, je taktéž rovna této hodnotě. Toto chování není vždy žádoucí a je vhodné jej omezit. Pro neexistující tokeny je tak vhodné určit nenulovou relativní frekvenci. Typickým postupem je při fázi strojového učení přičíst ke každé absolutní hodnotě výskytu všech tokenů konstantní hodnotu, například hodnotu $0,5$, a tuto konstantu poté prohlásit za absolutní hodnotu výskytu tokenů neuvedených v dané trénovací sadě. Relativní frekvence neuvedených tokenů je nízká a může tak výrazně ovlivnit výslednou hodnotu používanou pro klasifikaci, ale zároveň není rovna 0 a nezpůsobí výše popisovaný problém. [2]

Rozpoznávání pojmenovaných entit

Úlohou systémů implementujících rozpoznávání pojmenovaných entit (anglicky named entity recognition) je identifikace pojmenovaných entit (anglicky named entities) – slov či slovních spojení označujících specifické objekty reálného světa. Typicky se jedná o označení jedinců, organizací či geopolitických celků. Tento problém lze dále rozdělit na dvě části – identifikace začátku a konce pojmenované entity a následné přiřazení typu pro danou entitu. [2]

2.6 Evaluační techniky

Evaluace (anglicky evaluation) je proces hodnocení sledovaných vlastností systémů využívajících prvky umělé inteligence [8]. V práci používané evaluační techniky využívají **testovací sady** (anglicky test set, alternativně evaluation set) obsahující testovací vstupy pro evaluovaný systém a očekávaný výstup systému. Na základě porovnání očekávaného výstupu a skutečného výstupu systému jsou následně ohodnoceny jednotlivé sledované vlastnosti. Při porovnávání očekávaného a skutečného výstupu můžeme sledovat čtyři jevy [4]:

- **správná zařazení** do výsledků (anglicky true positives, značeno P_T) – jedná se o případy, které se nachází jak v množině očekávaných výstupních dat, tak ve výstupních datech systému, tedy hodnocený systém zkoumaný prvek korektně zařadil mezi výstupní data;
- **nesprávná zařazení** (anglicky false positives, značeno P_F) – jedná se o případy, které se nachází pouze ve výstupních datech systému, nikoliv v očekávaných výstupních datech, systém tento prvek zařadil mezi výstupní data chybně;
- **nesprávná vynechání** (anglicky false negatives, značeno N_F) – jedná se o případy, které se nachází pouze v množině očekávaných výstupních dat, hodnocený systém je do výstupních dat chybně nezařadil;
- **správná vynechání** (anglicky true negatives, značeno N_T) – jedná se o případy, které se nevyskytují ani v množině očekávaných výstupních dat, ani ve výstupních datech systému, systém tyto prvky korektně vynechal.

Tento text dále užívá značení výše zavedené ve smyslu počtu těchto jevů. Na základě počtu těchto jevů je možné vyjádřit **matici záměn** (alternativně **konfusní** nebo **chybová** matice, anglicky confusion matrix). Příklad 2.23 prezentuje konfusní matici pro výše uvedené sledované jevy. Matici záměn je možné zobecnit pro více pozorovaných jevů – toto je vhodné zejména při evaluaci multinomiálních single-category klasifikátorů. Konfusní matice K je vždy čtvercová, přičemž počet řádků a sloupců je roven celkovému počtu kategorií n přiřazovaných zkoumaným klasifikátorem. Každá buňka $K_{i,j}$, kde $i, j \in \{0, 1, \cdots, n\}$ určuje, jak často byla pro korektní třídu j predikována třída i. Prvky na hlavní diagonále matice určují procentuální míru správných predikcí, ostatní prvky poté míru nesprávných klasifikací. Je nutné předem určit, zda očekávanou, respektive systémem získanou, třídu reprezentují sloupce či řádky. [4]

$$\begin{array}{lcc} & \text{Očekáváno zařazení} & \text{Očekáváno vynechání} \\ \text{Provedeno zařazení} & P_T & P_F \\ \text{Provedeno vynechání} & N_F & N_T \end{array} \tag{2.23}$$

S využitím konfusní matice můžeme vyjádřit několik metrik hodnotících zkoumaný systém. Jedna z používaných metrik je **přesnost** (anglicky accuracy). Přesnost vyjadřuje poměr mezi počtem výstupů, které systém určil správně, a počtem všech předpokládaných testovacích výstupů. Předpokládáme-li testovací sadu o n testovacích vstupech, pak přesnost a zkoumaného systému je možné určit pomocí vztahu 2.24 [18, 4].

$$a = \frac{P_T + N_T}{n} \tag{2.24}$$

Další používanou metrikou pro vyhodnocení zkoumaného systému je **preciznost** (anglicky precision). Preciznost systému je definována jako poměr počtu relevantních a veškerých výstupů získaných při analýze. Míru preciznosti p je možné určit s využitím vztahu 2.25 [18, 4].

$$p = \frac{P_T}{P_T + P_F} \tag{2.25}$$

Metrika **výtěžnost** (anglicky recall) vyjadřuje poměr počtu získaných relevantních výstupů a počtu všech očekávaných relevantních výstupů. Pro zkoumaný systém je možné definovat výpočet výtěžnosti vztahem 2.26 [18, 4].

$$r = \frac{P_T}{P_T + N_F} \tag{2.26}$$

Poslední zde uvedenou metrikou používanou pro ohodnocení zkoumaného systému je **F1 skóre** (anglicky F1 score, alternativně F1 measure). Vlastní výpočet 2.27 skóre f_1 je harmonickým průměrem hodnot preciznosti a výtěžnosti pro daný zkoumaný systém [18, 4].

$$f_1 = 2 \cdot \frac{p \cdot r}{p + r} \tag{2.27}$$

2.7 Získávání informací

Získávání informací (anglicky information retrieval, zkráceně IR) je úloha řešící vyhledávání dat v předem dané množině, způsob tvorby této množiny a vkládání nových dat do

této. Data jsou v oblasti získávání informací nazývána **dokumenty**. Množina ukládající dokumenty je poté označována jako **index**. Index je datová struktura typicky optimalizována za účelem zefektivnění vyhledávání. [14]

Příkladem techniky optimalizace indexu může být **obrácený index** (anglicky inverted index). Takto vytvořený index mapuje jednotlivé termy na seznam dokumentů, v nichž se term nachází. V porovnání s **dopředným indexem** (anglicky forward index), jenž mapuje dokumenty na seznam termů vyskytujících se v daném dokumentu, je vyhledávání efektivnější, neboť není potřeba procházet seznam termů každého dokumentu, ale pouze dohledat dokumenty, v nichž se každý term nachází. Tabulka 2.2 ilustruje princip dopředného a obráceného indexu. [14]

Obrácený index

Term	ID dokumentu
Tramway	1, 2
Station	1

Dopředný index

ID dokumentu	Term
1	Tramway, Station
2	Tramway

Tabulka 2.2: Ilustrační příklad dopředného a obráceného indexu, příklad převzat z [14]

Jedním z přístupů pro získávání informací je fulltextové vyhledávání (anglicky full-text search). V tomto případě jsou dokumenty ukládány v textové podobě. Při požadavku na vyhledání uživatel uvádí termy popisující hledaný dokument. Z množiny uvedených termů je vytvořen vyhledávací řetězec (anglicky search query). S využitím vyhledávacího řetězce je poté provedeno hledání v indexu a jsou navráceny relevantní dokumenty. Vyhledané dokumenty jsou typicky řazeny dle relevance určené na základě vypočítaného skóre relevance. Toto skóre indikuje míru shodnosti dokumentu a vyhledávacího řetězce [14].

Tvorba vyhledávacího řetězce se typicky skládá z několika kroků. Prvním krokem je tokenizace, po níž může následovat filtrace. V rámci filtrace mohou být tokeny převedeny na základní tvar s pomocí stematizace či lemmatizace, může dojít k převodu všech znaků na malá písmena či k odebrání tokenů nacházejících se ve slovníku stop-slov. V případě použití obráceného indexu jsou pro jednotlivé tokeny vyhledány seznamy dokumentů, v nichž se tokeny vyskytují. Následně je proveden průnik všech seznamů, čímž získáváme dokumenty relevantní pro daný vyhledávací řetězec. Následně je možné provést hodnocení relevance na základě k tomu určených technik [14].

Vyhledávač Elasticsearch

Elasticsearch[1] je fulltextový vyhledávač implementovaný s využitím programovacího jazyka Java a open-source knihovny Apache Lucene. Komunikace s vyhledávačem probíhá pomocí aplikačního rozhraní fungujícího dle principů architektury rozhraní REST (plně Representational State Transfer) [14].

Jednotlivé uložené elementy v systému Elasticsearch jsou nazývány **dokumenty** (anglicky documents). Dokumenty se skládají z **polí** (anglicky fields) – párů klíčů a hodnot. Hodnota pole může nabývat libovolného typu. Dokumenty jsou prezentovány s pomocí formátu JSON. Definice polí, které dokumenty mohou obsahovat, je nazývána **mapování** (anglicky mapping). Elasticsearch podporuje dva způsoby mapování – dynamické a explicitní. Při **dynamickém mapování** je schéma dokumentů tvořeno odvozeno z vkládaných

[1] Dokumentace systému Elasticsearch je dostupná online na adrese https://www.elastic.co/guide/index.html

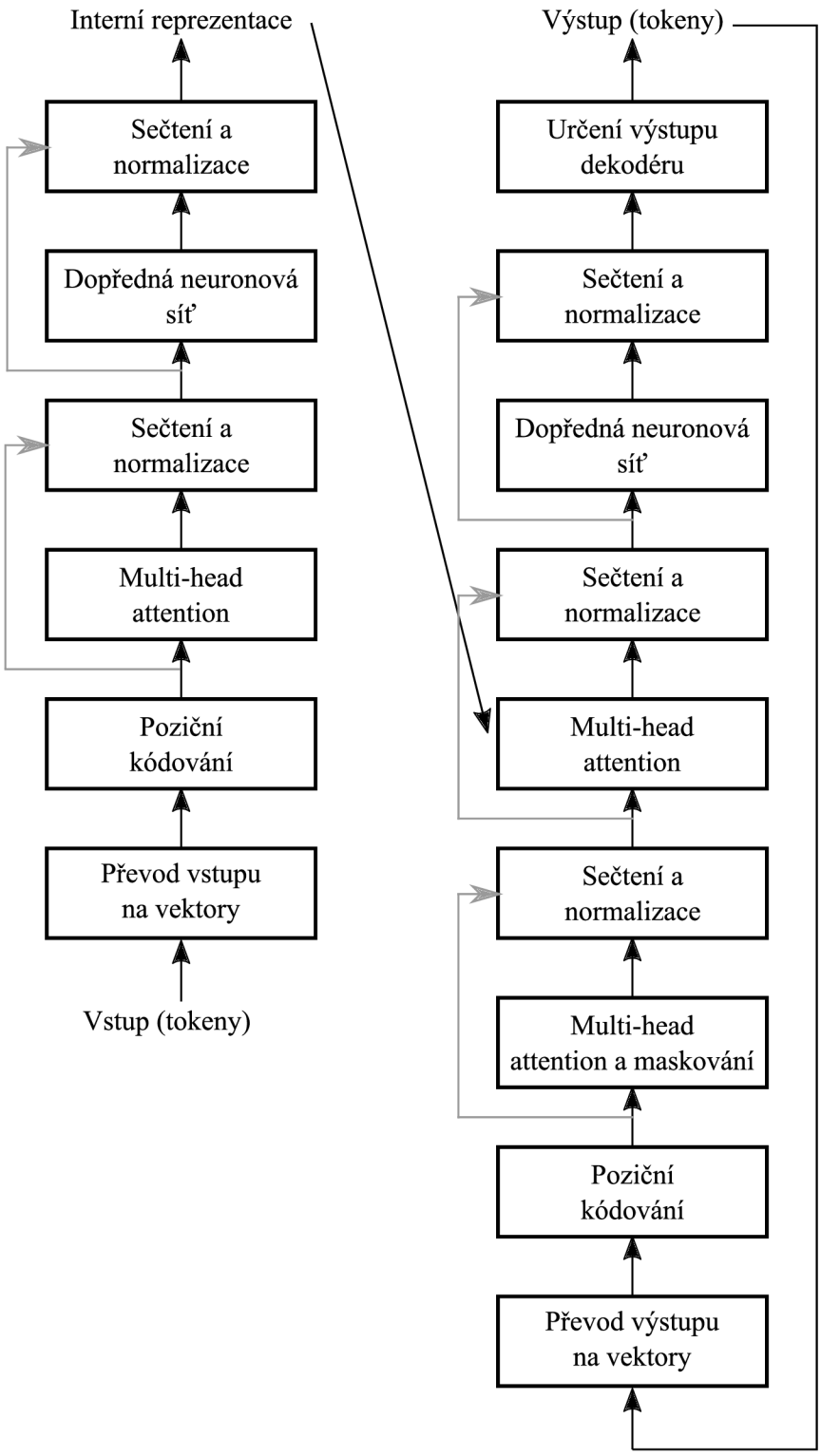
dat. **Mapování explicitní** naopak umožňuje předem definovat přípustná pole a jejich typy. [7]

Dokumenty jsou ukládány do **indexů**. Elasticsearch využívá techniky obrácených indexů. Dokumenty uložené v jednom indexu spolu typicky souvisí a mají shodné mapování. Indexy jsou ukládány v paměti **uzlů** (anglicky nodes) – fyzických či virtuálních počítačů, na nichž je instalován a spuštěn systém Elasticsearch. Jelikož indexy mohou pro uložení vyžadovat více paměťového prostoru, než je schopen poskytnout jeden uzel, je možné uložení indexů rozložit mezi více uzlů propojených sítí a tím tak systém Elasticsearch horizontálně škálovat. V případě uložení indexu na více uzlech je index rozdělen do dvou či více částí nazývaných **shards**. Počet shards může být explicitně určen, implicitně je pak dáno rozdělení na pět shards [7].

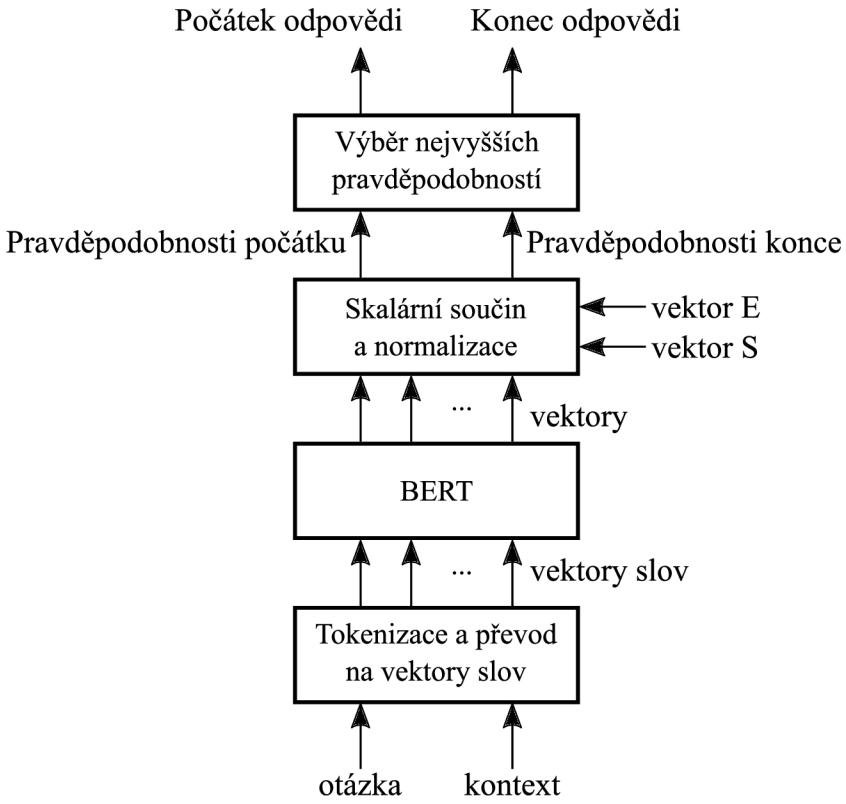
Množina uzlů, které se podílí na uložení všech indexů používaných v rámci aplikace, se nazývá **cluster**. Uzel, případně uzly, na nichž je index, ve kterém má probíhat vyhledávání, je automaticky určen systémem. Odpadá tak nutnost výběru uzlu, na němž má probíhat vyhledávání [7].

Pro zajištění dostupnosti dat je možné vytvářet **repliky** (anglicky replicas) z shards. Replika je kopií shard uloženou na jiném uzlu a je tak možné ji použít v případě poruchy některého uzlu a následné ztráty dat. Repliku daného shard je také možné použít pro rozložení zátěže mezi uzly. Ve výchozím nastavení je pro každý shard vytvořena právě jedna replika. Možný způsob realizace uložení dat pomocí systému Elasticsearch ilustruje obrázek 2.7 [7].

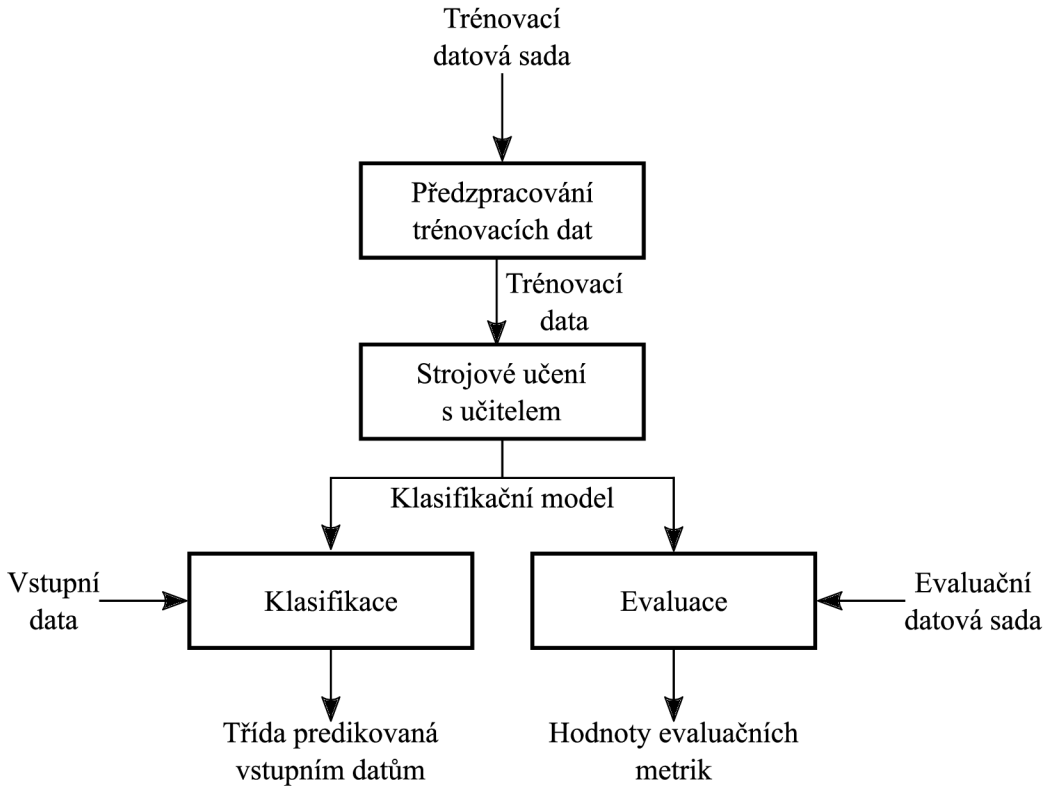
Při vyhledávání dat je použit princip vyhledávání v obrácených indexech. Následně jsou výsledné relevantní dokumenty ohodnoceny pomocí skóre TF/IDF. Popis výpočtu tohoto skóre je uveden v sekci 2.4. Systém Elasticsearch umožňuje při výpočtu hodnoty skóre dokumentu přidělovat váhu jednotlivým polím – relevantnost dokumentu tak může být více ovlivňována vybranými poli. Systém také podporuje výpočet hodnoty skóre bez využití frekvence termu ve vyhledávacím řetězci [14].



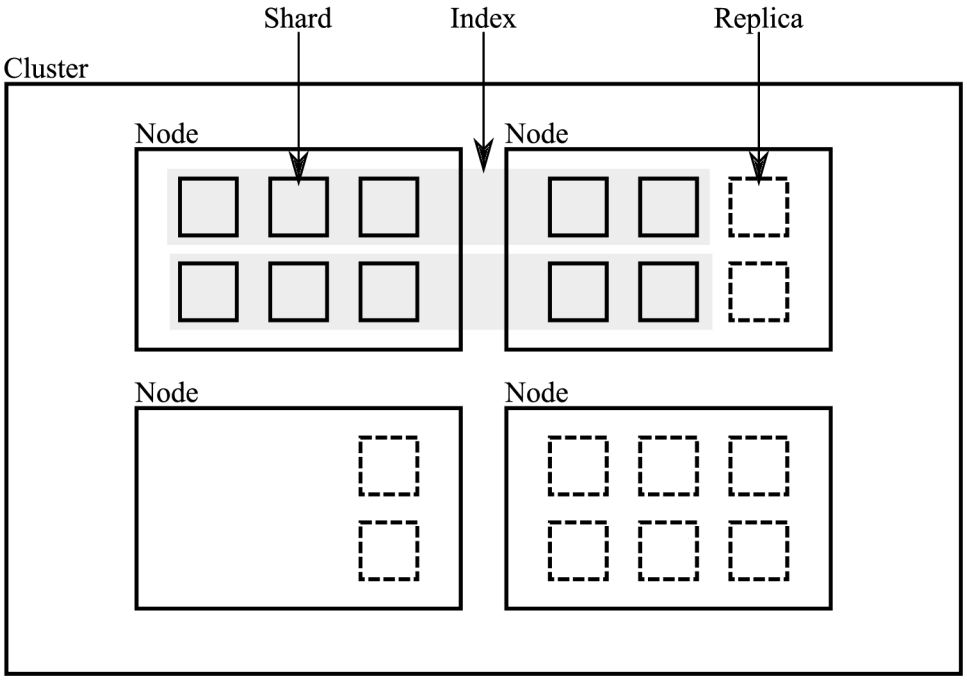
Obrázek 2.4: Schéma architektury neuronové sítě transformers, převzato z [17]



Obrázek 2.5: Schéma architektury modelu BERT pro odpovídání na otázky, převzato z [17]



Obrázek 2.6: Blokové schéma principu klasifikátoru, převzato z [10]



Obrázek 2.7: Možné schéma uložení dat systémem Elasticsearch, převzato z [14]

Kapitola 3

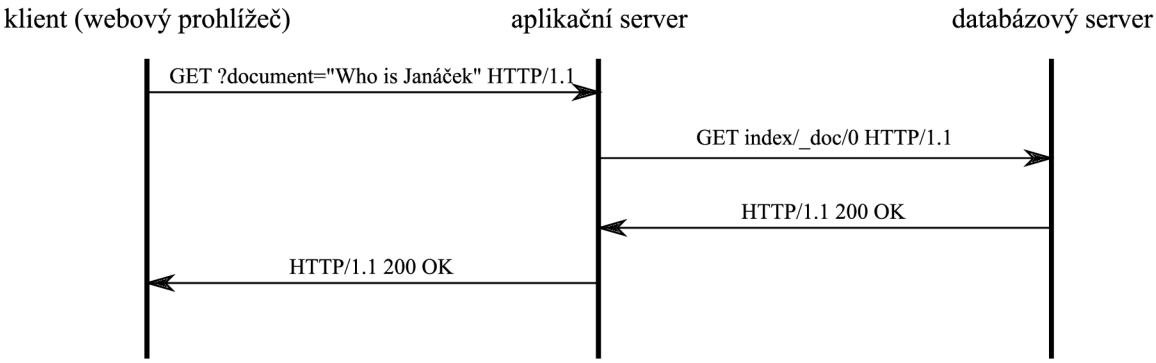
Návrh řešení

Tato kapitola obsahuje popis vlastního návrhu a implementace systému komunikačního agenta. Při vývoji byly využity programovací jazyky Python3 a JavaScript. Projekt dále využívá značkovacích jazyků HTML a CSS. V následujících sekcích jsou detailně popsány jednotlivé implementační aspekty celého systému. Kapitola nepřináší podrobný popis veškerých implementačních problematik, jsou předpokládány znalosti v rozsahu bakalářského studia informačních technologií. Dále jsou využívány teoretické poznatky popsané v kapitole 2. Způsob spuštění systému komunikačního agenta je uveden v příloze A.

3.1 Architektura systému

Systém komunikačního agenta pro informace o Brně je implementován dle principů **třívrstvé architektury** (anglicky three-tier architecture). Agent se skládá z vrstvy **prezentační**, webového grafického rozhraní umožňujícího interakci uživatele a systému, vrstvy **aplikační** realizující vlastní logiku agenta a vrstvy **datové** poskytující data vrstvě aplikační. Vrstvy aplikační a datová jsou implementovány servery, mezi nimiž je síťová komunikace realizována s využitím protokolu **Hypertext Transfer Protocol** (zkratka HTTP). Prezentační vrstva komunikuje s vrstvou aplikační asynchronně pomocí HTTP.

Schéma síťové komunikace klientů a serverů tvořících komunikačního agenta je uvedeno na obrázku 3.1. Při zadání uživatelského dotazu zasílá prezentační vrstva asynchronní požadavek aplikačnímu serveru protokolem HTTP, metodou **GET**. Cílová URL (Uniform Resource Locator) adresa aplikačního serveru obsahuje **URL parametr** (anglicky URL parameter, alternativně query string) uvádějící uživatelskou otázku. Aplikační server přijímá klientský požadavek a zahajuje proces získání odpovědi na otázku, během něhož dojde ke kontaktování serveru datové vrstvy s využitím metody GET protokolu HTTP. V roli serveru datové vrstvy může být lokální databázový server, webový vyhledávač, případně jiný server poskytující užitečná data. Po získání odpovědi z dat poskytnutých serverem datové vrstvy zasílá aplikační server klientovi, prezentační vrstvě, odpověď na HTTP požadavek. Příchozí odpověď je vizualizována uživateli pomocí webového grafického rozhraní.



Obrázek 3.1: Schéma síťové komunikace vrstev systému pomocí protokolu HTTP

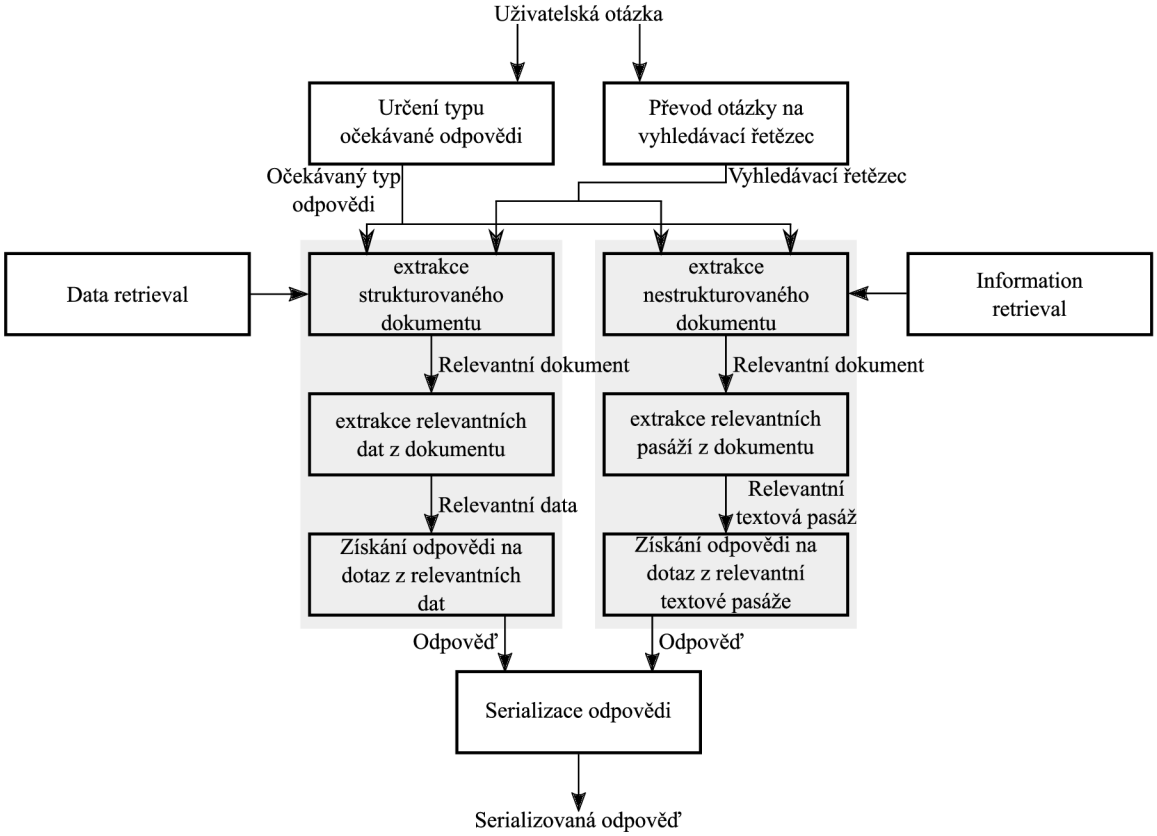
Aplikační vrstvu implementuje server podporující HTTP komunikaci metodou GET. Tento server využívá pro zodpovídání otázek dva separátní subsystémy pro odpovídání na uživatelské dotazy, přičemž první subsystém realizuje odpovídání s využitím dat nestrukturovaných a druhý pomocí dat strukturovaných. Podrobné implementační detaily jsou uvedeny v sekci 3.6. Po provedení extrakce odpovědi je tato serializována s využitím formátu JSON a v rámci HTTP odpovědi tuto zasílá server klientovi. Pro determinování subsystému, který bude použit pro zodpovězení dotazu, je po přijetí HTTP požadavku na server nutné otázku klasifikovat. Bližší realizační informace týkající se tohoto klasifikátoru jsou uvedeny v sekci 3.5. Schéma na obrázku 3.2 blíže popisuje systém realizující aplikační vrstvu komunikačního agenta.

Pro možnost snadného rozložení zátěže je systém na aplikační vrstvě horizontálně škálován. Subsystém pro nalezení faktické odpovědi využívající principy neuronových sítí je implementován jako samostatný server. Tento server komunikuje s aplikačním serverem pomocí HTTP protokolu metodou GET. Při zaslání požadavku je nutné mezi URL parametry zařadit samotnou uživatelskou otázku a relevantní text, z něhož je extrahována odpověď. Komunikace aplikačního serveru a popsaného serveru pro extrakci faktické odpovědi probíhá na lokální síti, proto není nutné mít dvě veřejné IP adresy pro zařízení, na nichž je serverová aplikace spuštěna. Na obrázku 3.3 je ilustrován příklad komunikace výše zmíněných serverů implementujících systém aplikační vrstvy.

Vrstvu datovou z hlediska síťové architektury implementuje server realizující internetové vyhledávání webových serverů poskytujících relevantní data pro uživatelskou otázku, kontaktované vzdálené webové servery, z nichž jsou extrahovány textové dokumenty, servery poskytující aplikační rozhraní umožňující přímé získávání informací o Brně ve strukturované podobě pomocí HTTP požadavků a lokální databázový server ElasticSearch umožňující ukládání a znovuvyužití již extrahovaných textových nestrukturovaných dokumentů. Bližší implementace získávání relevantních dat je popsána v sekci 3.6.

3.2 Implementace grafického uživatelského rozhraní

Prezentační vrstva je realizována webovým **grafickým uživatelským rozhraním** (zkratka GUI) a komunikačním subsystémem umožňujícím asynchronní komunikaci s aplikačním serverem prostřednictvím protokolu HTTP. Přijatá data serverové odpovědi následně subsystém vizualizuje uživateli. Při vývoji GUI byl využit programovací jazyk JavaScript implementující komunikační modul a další podpůrné skripty, značkovací jazyk HTML vytvářející



Obrázek 3.2: Schéma systému realizujícího aplikační vrstvu komunikačního agenta

základní strukturu rozhraní a stylovací jazyk CSS definující grafickou podobu webového rozhraní. Webové rozhraní je možné používat s pomocí libovolného webového prohlížeče, doporučenými prohlížeči jsou Google Chrome, případně Mozilla Firefox.

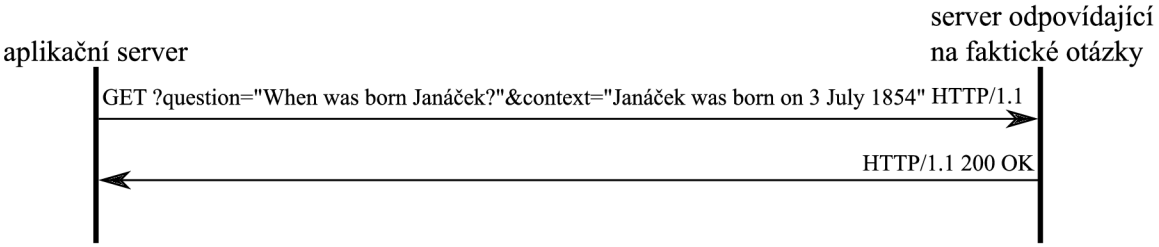
Uživatel může komunikovat s agentem skrze grafické rozhraní v textové a hlasové podobě. Hlasové zadávání dotazu je implementováno s využitím aplikačního rozhraní `webkitSpeechRecognition` pro jazyk JavaScript. K 25. dubnu 2021 je používané aplikační rozhraní podporováno internetovým webovým prohlížečem Google Chrome minimální verze 33[1].

Následná textová část odpovědi na dotaz může být prezentována jak v textové, tak v hlasové podobě. Tato část odpovědi je syntetizována do lidské řeči s pomocí API `speechSynthesis` pro jazyk Javascript. K 25. dubnu 2021 je aplikační rozhraní podporováno dnes nejpoužívanějšími internetovými webovými prohlížeči – Google Chrome minimální verze 33, Mozilla Firefox minimální verze 49 a další[2].

Pro omezení nutnosti opakovaného znovu-načítání rozhraní při přijetí odpovědi z aplikačního serveru probíhá komunikace mezi GUI a aplikačním serverem asynchronně. Pro implementaci asynchronního zasílání HTTP dotazů je použito aplikační rozhraní XMLHttpRequest. Server odpověď na uživatelský dotaz serializuje pomocí serializačního formátu JSON. Bližší popis metadat serializované odpovědi zasílané aplikačním serverem je uveden v sekci 3.7.

[1] Bližší informace o kompatibilitě `webkitSpeechRecognition` API je možné dohledat na webové stránce https://caniuse.com/speech-recognition

[2] Bližší informace o kompatibilitě `speechSynthesis` API je možné dohledat na webové stránce https://caniuse.com/speech-recognition



Obrázek 3.3: Schéma síťové komunikace serverů implementujících aplikační vrstvu

3.3 Implementace subsystémů pro tvorbu strojové reprezentace textů

Teoretické principy strojové reprezentace dat jsou uvedeny v sekci 2.3. Není-li uvedeno jinak, pro implementaci metod převádějících texty lidského jazyka do strojově zpracovatelné podoby je využit jazyk Python3. Pro implementaci algoritmů umožňujících vytvoření vektorů slov byly využity předem trénované a volně dostupné modely, případně je převod realizován v rámci knihovních funkcí.

Proces tokenizace je implementován třídou `tokenizer`.Pro vlastní tokenizaci jsou využívány třídy a metody implementované knihovnou `textblob`[3]. V rámci subsystému pro tvorbu strojově zpracovatelných dat jsou využity následující třídy:

- `textblob.blob.TextBlob`: objekty této třídy poskytují veřejné členské proměnné, pomocí kterých je možné přistupovat k strojovým reprezentacím původního dokumentu – tokenům utvořených z jednotlivých slov, částí slov či vět;
- `textblob.blob.Word`: objekty instanciované s pomocí této třídy reprezentují jednotlivé tokeny vytvořené ze slov či částí slov původního dokumentu;
- `textblob.blob.Sentence` – objekty třídy reprezentují jednotlivé věty původního textu.

3.4 Implementace subsystému pro extrakci klíčových slov

Teoretické principy extrakce klíčových slov z textového dokumentu jsou popsány v sekci 2.4. Tato metoda předpokládá implementovanou třídu `tokenizer` pro tokenizaci dokumentů. Vlastní subsystém pro získávání klíčových slov je implementován třídou `keyword_search`.

Subsystém využívá principu strojového učení bez učitele. Fáze učení vyžaduje existenci datové sady obsahující pouze vstupy systému. Z důvodu specifického zaměření tohoto subsystému na extrakci klíčových slov z uživatelských otázek obsahuje trénovací datová sada uživatelské dotazy. Princip učení spočívá ve vytvoření množiny M dvojic (t, t_{idf}), kde t je token nacházející se minimálně v jednom dotazu trénovací sady a t_{idf} je hodnota převrácené četnosti tokenu v trénovacích sadě dotazů.

Pro tokeny nevyskytující se v trénovací sadě uživatelských otázek je nutné určit hodnotu t_{idf} jiným způsobem. Tato hodnota je jednotná pro veškeré v množině M dosud neexistující tokeny a je využívána při vlastní extrakci klíčových slov z otázky pro tokeny, které se

[3]Dokumentace knihovny `textblob` pro jazyk Python je dostupná online na https://textblob.readthedocs.io/en/dev/

nachází v uživatelském dotazu, ale ne v množině dvojic M. V rámci vyvinutého subsystému je hodnota u_{idf} pro neexistující token u pevně dána vztahem 3.1, kde t_{max} je maximální existující hodnota převrácené četnosti pro určitý token z trénovací sady dotazů.

$$u_{idf} = t_{max} \cdot 1,5 \tag{3.1}$$

Po dokončení fáze strojového učení je možné vzniklý model používat pro extrakci klíčových slov z uživatelských dotazů. V rámci procesu hodnocení významnosti jednotlivých tokenů t zkoumané otázky je nutné určit počet výskytů jednotlivých tokenů t_{tf} ve analyzovaném dokumentu. Dále je pro každý token nutné určit hodnotu t_{idf}, kterou je možné získat s použitím množiny M vytvořené při fázi učení. Pro tokeny vyskytující se pouze ve zkoumaném dotazu je použita hodnota u_{idf}. V případě, že se token t nachází v negativním slovníku, je tomuto přiřazena hodnota významnosti 0.

Po zjištění obou potřebných hodnot pro každý token dotazu jsou tyto vynásobeny a výsledná hodnota je prohlášena za významnost daného tokenu. Následně je nutné určit hranici významnosti – slova s významností vyšší než je tato hranice jsou označena jako klíčová, slova s hodnotou významnosti nižší jsou prohlášena za neklíčová. Tuto hranici systém určuje jako aritmetický průměr všech hodnot významnosti.

Výhodou tohoto přístupu je možnost stálého rozšiřování množiny dvojic M o nově se vyskytující tokeny v uživatelských dotazech a upřesňování hodnot převrácené četnosti pro již existující tokeny. Možnou nevýhodou používaného algoritmu je pak snižování hodnoty t_{idf} a tím i výsledného skóre pro často se vyskytující tokeny t, které jsou ovšem považovány za klíčové. Evaluace zde popsaného subsystému je uvedena v sekci 4.1.

3.5 Implementace klasifikátoru pro určení typu odpovědi

Teoretické principy využívané při realizaci klasifikátoru typu odpovědi na uživatelskou otázku jsou uvedeny v sekci 2.5. Tento text dále předpokládá implementované třídy `tokenizer` a `keyword_search`. Popis implementace těchto tříd je uveden v sekci 3.3, respektive 3.4. Klasifikátor je implementován třídou `qtd`. Diagram tříd je uveden na obrázku 3.7.

Po získání uživatelské otázky z prezentační vrstvy je nutné na úrovni vrstvy aplikační určit typ očekávané odpovědi. Určený typ dále ovlivňuje způsob získání dat pro zodpovězení dotazu, určení odpovědi a způsob prezentace dat odpovídajících na položený dotaz. Implementovaný klasifikátor rozlišuje celkem 8 různých kategorií odpovědí – faktografická odpověď, definiční přehled s ilustračním obrázkem, vyhledání spoje hromadné dopravy, lokalizace místa na mapě, informace o ulici, seznam provozoven poskytujících určitou službu, informace o kulturních akcích a událostech a předpověď počasí. Po určení způsobu zodpovězení otázky jsou pro samotné získání odpovědi využity k tomu realizované subsystémy blíže popsané v sekci 3.6.

Klasifikátor typu odpovědi na otázku je implementován s využitím třídy `classifiers.NaiveBayesClassifier` balíčku `textblob`. Před vlastní klasifikací je nutné provést fázi strojového učení s učitelem. Pro strojové učení je vyžadována existence trénovací datové sady K skládající se z dvojic (q, c), kde q představuje uživatelskou otázku a c očekávanou kategorii odpovědi na danou otázku. Tato trénovací data byla sesbírána od uživatelů prostřednictvím webového dotazníku.

Před instanciací objektu klasifikátoru s využitím otázek shromážděných od uživatelů jsou tyto dotazy předem modifikovány tak, aby byl co nejvíce omezen vliv faktografických údajů, jako názvy budov, osobností či míst. Modifikace spočívá v převedení otázky na

množinu tokenů a následném nahrazení klíčových tokenů za zástupný token s. Sekvence více po sobě jdoucích zástupných tokenů s je pak nahrazena právě jedním tokenem s.

Objekty instanciované z třídy `qtd` dále využívají spolu s výše popsaným klasifikátorem alternativní přístup využívající tokeny klíčové, neboť pro určení typu odpovědi lze v některých případech tyto tokeny využít. Jedná se například o tokeny související s tématem počasí či kultury. Pro klasifikaci na základě těchto tokenů slouží subsystém pro rozpoznávání pojmenovaných entit. Tento systém je trénován s využitím sady shodné pro trénování naivního bayesovského klasifikátoru. Pro realizaci rozpoznávání pojmenovaných entit je použito funkcionalit knihovny `spacy`[4]. Entity, respektive klíčové tokeny, jsou kategorizovány do tříd uvedených pro původní otázku.

V případě nalezení více než dvou rozdílných klíčových tokenů, které systém pro rozpoznávání pojmenovaných entit kategorizoval do více než jedné kategorie, jsou výsledky této metody označeny za neprůkazné a na tento výstup není brán zřetel. V případě nenalezení žádné pojmenované entity klasifikátor rozhoduje obdobně, pouze na základě klasifikace naivního bayesovského klasifikátoru.

3.6 Implementace subsystému pro vyhledávání relevantních dat a pro odpovídání na dotazy

Tato sekce pojednává o implementaci metod pro práci s daty strukturovanými i polostrukturovanými. Zde uvedená realizace je využívána v sekci 3.6. Aplikované teoretické postupy jsou uvedeny v sekci 2.7.

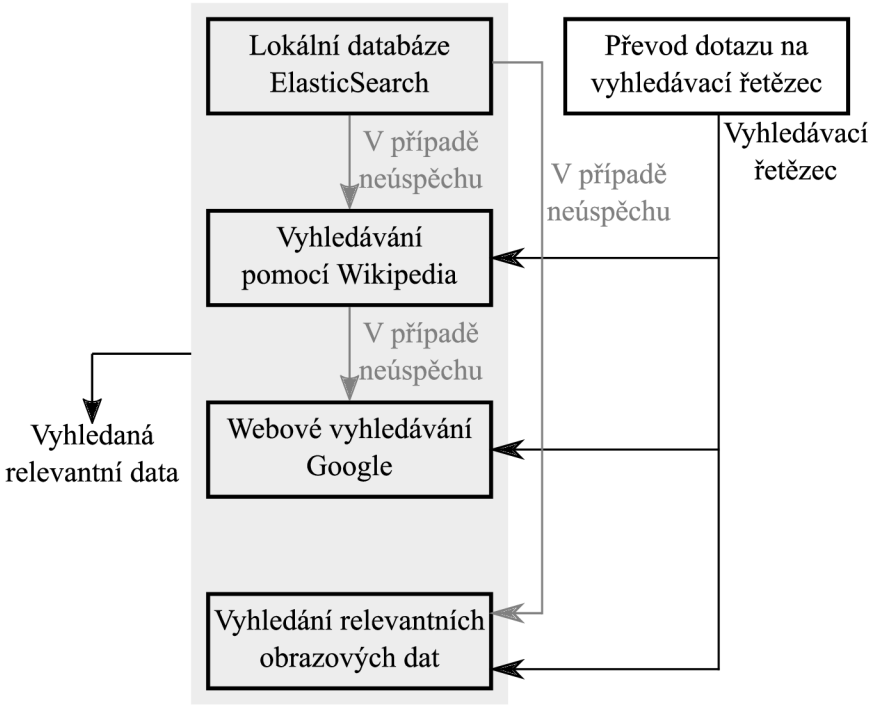
Vyhledávání a ukládání polostrukturovaných dat

Subsystém pro odpovídání na dotazy s využitím polostrukturovaných textových dokumentů tyto vyhledává s užitím dvou základních postupů. Prvním krokem je vyhledávání pro otázku relevantního dokumentu s pomocí systému Elasticsearch. V případě, že touto cestou nebyl nalezen žádný s tématem otázky související dokument, je provedeno vyhledávání dokumentu s využitím webového vyhledávače. Schéma ilustrující princip vyhledávání relevantních polostrukturovaných dat je uvedeno na obrázku 3.4.

Prvním krokem při získávání textu potencionálně obsahujícího odpověď je pokus o vyhledání relevantního dokumentu s využitím systému Elasticsearch. Vyhledávání je prováděno objektem třídy `lookup_table`. Diagram tříd je uveden na obrázku 3.5. Pro vyhledávání je použit vyhledávací řetězec **More like this**, který umožňuje vyhledání nejvíce podobných dokumentů. Každý dokument ukládá relevantní text, otázky vázající se k textu, pro dotaz relevantní obrazová data a URL adresu odkazující na zdroj textu. Pro určení relevantnosti je pak využita původní uživatelská otázka a této nejvíce podobná otázka. Pomocí objektu třídy `keyword_search` popsané v sekci 3.4 jsou pro tyto otázky vytvořeny množiny klíčových slov Q_u, obsahující klíčová slova uživatelského dotazu, a Q_e, obsahující klíčová slova otázky získané pomocí vyhledávání. Následně je provedena stematizace obou množin klíčových slov. První složka výsledného skóre r_k pro hodnocení relevantnosti je pak dána vztahem 3.2, kde čitatel je kardinalita průniku množin Q_u a Q_e a jmenovatel je kardinalita sjednocení těchto množin.

$$r_k = \frac{|Q_u \cap Q_e|}{|Q_u \cup Q_e|} \tag{3.2}$$

[4]Dokumentace knihovny `spacy` pro jazyk Python je dostupná online na https://spacy.io/



Obrázek 3.4: Schéma principu vyhledávání relevantních polostrukturovaných dat

Druhou složkou tvořící výsledné skóre pro hodnocení relevantnosti je pak kosinová podobnost vektorů vytvořených z uživatelského a získaného dotazu. Způsob výpočtu kosinové podobnosti je uveden v sekci 2.3, tvorba vektoru v sekci 3.3. Výsledné skóre pro hodnocení relevantnosti je dáno aritmetickým průměrem výše uvedených složek. Skóre je definováno v intervalu $\langle 0; 1 \rangle$. Za relevantní je dokument označen v případě, že skóre hodnotící relevantnost je vyšší než předem určená hodnota. V opačném případě je dokument označen za nerelevantní a je tak nutné provést další kroky vedoucí k zisku relevantního dokumentu.

V případě nenalezení relevantního dokumentu je provedeno vyhledání dokumentu s využitím webového vyhledávače. Webové vyhledávání je implementováno objektem třídy `source_search`. Tato třída pro vyhledávání používá funkcionality knihoven `wikipedia` umožňující vyhledávání encyklopedických článků v internetové encyklopedii Wikipedia, `google-search` provádějící dohledávání webových serverů obsahující texty relevantní pro danou otázku s využitím vyhledávače Google a knihovny `google_images_download` umožňující vyhledání obrazových dat relevantních pro dotaz.

Před vlastním vyhledáním relevantních dat je potřeba sestavit vyhledávací řetězec. Vyhledávací řetězec je vytvořen z klíčových slov otázky, která jsou uvedena v řetězci ne dle významnosti, ale v pořadí, v němž se objevují v původní otázce. První vyhledání dat je provedeno pomocí funkcionalit knihovny `wikipedia`. V případě neexistence dokumentu odpovídajícího danému vyhledávacímu řetězci je provedeno vyhledání dokumentu s využitím metod knihovny `googlesearch`. Použitá metoda navrací seznam webových serverů obsahujících textové dokumenty relevantní pro zadanou otázku. Pro získání dokumentu je nutné provést HTTP požadavek zasílaný vybranému webovému serveru s cílem získání textu. K tomuto účelu jsou použity funkcionality poskytnuté knihovnou `urllib` umožňující zasílání HTTP dotazů. Po obdržení webového dokumentu ve formátu HTML je tento syntakticky analyzován (anglicky parsing) s využitím knihovny `bs4.BeautifulSoup`. Po vy-

hledání textového dokumentu jsou s využitím vyhledávacího řetězce pro danou uživatelskou otázku dohledána obrazová data, která mohou doplnit odpověď na dotaz.

Objekty tříd `source_search` a `lookup_table` jsou používány objektem třídy `data_provider`, který s využitím metod implementovaných objekty dvou výše popsaných tříd tvoří aplikační rozhraní umožňující získávání dat. Diagram tříd je uveden na obrázku 3.5. Dále objekty třídy `data_provider` automaticky provádí ukládání dat získaných webovým vyhledávačem do indexu pro ukládání těchto dokumentů, realizují vkládání dat do zmíněného indexu z prvotní datové sady SQuAD či implementují metodu umožňující zrušení indexu.

Odpovídání na dotazy s využitím polostrukturovaných dat

Na základě polostrukturovaných dat je možné vytvářet odpovědi definiční, textový popis charakterizující objekt reálného života, a faktické: přesný údaj, číselná hodnota a další. Při vytváření definiční odpovědi systém po získání relevantních dat uživateli prezentuje krátký textový popis, první odstavec či jeho část z nalezeného textu, obrazová data ilustrující objekt reálného světa, jehož se dotaz týká, a odkaz na zdroj dat obsahující více podrobností. Metoda pro definiční odpovídání je implementována objektem třídy `sentence_answer`. Nad oběma způsoby odpovídání poskytuje aplikační rozhraní třída `qas`. Diagram tříd je uveden na obrázku 3.5.

Pro získávání faktických odpovědí na uživatelské dotazy jsou používány objekty a metody implementované knihovnou `adaptNLP`[5]. Pro vlastní proces odpovídání využívá knihovna modelu `bert-large-uncased-whole-word-masking-finetuned-squad` poskytnutého komunitou vývojářů huggingface. Používané teoretické poznatky a principy jsou uvedeny v sekcích 2.1 a 2.2. Vlastní volání knihovních metod je prováděno objektem třídy `factoid_answer`. UML diagram tříd je zobrazen na obrázku 3.5.

Vyhledávání a ukládání strukturovaných dat a odpovídání na dotazy

Pro zodpovídání určitých typů otázek je vhodné použití dat strukturovaných. Zdrojem dat jsou typicky internetová aplikační rozhraní či datové sady obsahující užitečné informace o Brně. Následuje výčet používaných zdrojů v rámci výsledného systému komunikačního agenta pro informace o Brně:

- Aplikační rozhraní Mapy API[6] verze 4.13 poskytující možnost geografické lokace objektů na základě jejich textového popisu;
- Internetová encyklopedie Brna[7] uvádějící podrobné informace o ulicích Brna;
- Vyhledávač spojů[8] hromadné dopravy Integrovaného dopravního systému Jihomoravského kraje (zkráceně IDS JMK);
- Datová sada[9] obsahující seznam zastávek obsluhovaných dopravci, kteří jsou zapojeni do systému IDS JMK;

[5] Dokumentace knihovny `adaptNLP` je dostupná na adrese https://novetta.github.io/adaptnlp/

[6] Dokumentace Mapy API je dostupná na adrese http://api.mapy.cz/doc/index.html

[7] Encyklopedie ulic Brna je dostupná na adrese https://encyklopedie.brna.cz/home-mmb/?acc=ulice

[8] Webové rozhraní vyhledávače spojů je dostupné na adrese https://www.idsjmk.cz/connection-finder/search

[9] Datová sada je dostupná online na https://data.brno.cz/datasets/0fc06562018b4e6880b5514fb333ca6f_0/about

- Aplikační rozhraní Yelp Fusion[10] poskytující seznamy služeb evidovaných na území města Brna;
- Internetová stránka BRNO Daily[11] umožňující vyhledání poskytovaných služeb na území města Brna;
- Aplikační rozhraní One Call API[12] poskytované organizací OpenWeather zveřejňující podrobné informace o aktuálním počasí a předpovědi pro území Brna;
- Datová sada[13] kulturních akcí pořádaných na území města Brna.

Na rozdíl od dat polostrukturovaných, která mají v rámci vyvinutého systému encyklopedický charakter a jsou tak v průběhu času typicky neměnná, data strukturovaná se mohou v průběhu času měnit. Proto tato data systém neukládá, případně definuje časovou platnost dat. Časová platnost dat je určena před spuštěním aplikačního serveru v nastavení, detaily jsou uvedeny v sekci 3.7. Tyto hodnoty uvádí počet dní, po které jsou data považována za aktuální. Pro každý datový zdroj je možné nastavit jinou hodnotu časové platnosti. Při získání dat je vytvořen perzistentní objekt instanciovaný z třídy `datetime.datetime`. Tento objekt obsahuje datum expirace těchto dat. Při každém přístupu k datům je provedeno porovnání aktuálního a expiračního datumu. Pokud došlo k expiraci dat, jsou tato data nahrazena aktuálními a pro tato je uloženo nové datum expirace. Pro zajištění perzistence objektu je použit serializační formát pickle. Nad všemi objekty tříd implementujících získávání relevantních dat a odpovídání na základě těchto vytváří rozhraní třída `sas`. Diagram tříd je uveden na obrázku 3.6.

Při geolokaci je pro hledaný objekt reálného světa zjišťována zeměpisná šířka a délka, které nejsou systémem ukládány. Před vlastním hledáním je nutné z uživatelské otázky získat název hledaného objektu – tato úloha je realizována s pomocí objektu třídy `keyword_search` – z uživatelského dotazu jsou získána klíčová slova, která jsou následně seřazena dle pořadí výskytu v dotazu. Následně s pomocí rozhraní Mapy API je mapa se zjištěnou pozicí objektu vizualizována v grafickém rozhraní. Zaslání HTTP požadavku aplikačnímu rozhraní je realizováno objektem třídy `geo_data`. Diagram tříd je zobrazen na obrázku 3.6.

V případě vyhledávání spojů hromadné dopravy zapojených do systému IDS JMK jsou získávána data o cestě – číslo linky či linek, které obsluhují uživatelem dané zastávky, dobu jízdy, čas odjezdu a příjezdu a další. Tato data získaná pro uživatelský dotaz nejsou ukládána. Aplikační rozhraní komunikuje s pomocí protokolu HTTP metodou GET, kdy s pomocí parametrů URL jsou předány názvy výchozí a cílové zastávky. Tyto je nutné získat z uživatelského dotazu. Při detekci zastávek je uživatelský dotaz převeden na množinu tokenů T, z níž je následně vytvořena množina n-gramů, kde $n \in \{0, 1, \cdots, |T|\}$. Následně je každý n-gram převeden na vyhledávací řetězec (jednotlivé tokeny jsou propojeny do řetězce a odděleny mezerou) a je provedeno porovnání vůči jménům zastávek uvedených v množině S. V případě, že se daný vyhledávací řetězec v množině S nachází, pak je určen název zastávky. Tato metoda umožňuje vyhledat zastávky i v případě chybného zápisu s pomocí Levenshteinovy vzdálenosti a podobnosti (tato problematika je popsána v sekci 2.4). Pokud není poměr mezi vyhledávacím řetězcem a názvem zastávky menší než předem určená

[10] Dokumentace Yelp Fusion API je dostupná online na adrese https://www.yelp.com/developers/documentation/v3

[11] Vyhledávač služeb je dostupný na adrese https://brnodaily.com/search-listings/

[12] Dokumentace a uživatelská příručka je dostupná online na https://openweathermap.org/api/one-call-api

[13] Datová sada je dostupná online na https://data.brno.cz/datasets/8df323e4e66e41fe8e62e325cef2644c_0/about

hodnota, pak je nalezen správný název zastávky. Množina obsahující jména zastávek je vytvořena s využitím datové sady obsahující informace o zastávkách. Tato datová sada je od doby získání aktuální po jeden rok. V tomto odstavci uvedené principy jsou implementovány třídou `public_transport`. Diagram tříd je uveden na obrázku 3.6.

Pokud mají být nalezena data o službách poskytovaných na území města Brna, je možné použít dva zdroje dat. Primárním zdrojem je aplikační rozhraní Yelp Fusion. V případě nenalezení žádné provozovny či nedostatečného počtu je pro doplnění seznamu použit zdroj sekundární – webová stránka BRNO Daily. Před vlastním vyhledáním je nutné určit název vyhledávané služby. Pro určení názvu je využit postup obdobný jako při geolokaci objektů reálného světa. Pro každou provozovnu poskytující hledanou službu je uveden její název, uživatelské hodnocení, ilustrační obrázek a odkaz na zdroj dat. Zde popsané postupy získání dat jsou implementované objektem třídy `service_search`. Diagram tříd je zobrazen na obrázku 3.6.

Data o ulicích jsou získána z internetové encyklopedie Brna. Na základě v ní uvedených dat byla skriptem vytvořena datová sada pro každou ulici definující její název, původ jména, lokaci, datum vzniku a případného zániku a odkaz na zdroj dat. Pro tuto datovou sadu není definováno datum expirace, jelikož se jedná o encyklopedická data. Před vyhledáváním dat je nutné z uživatelského dotazu získat název ulice. Tohoto je docíleno totožným způsobem jako při vyhledávání názvů zastávek z dotazu požadujícího informace o spojení MHD. Vyhledávání dat o ulicích je implementováno objektem třídy `geo_data` uvedené v diagramu tříd na obrázku 3.6.

Kulturní akce ukládá systém do k tomu určeného indexu systému Elasticsearch. Pro vlastní vyhledávání nejrelevantnější kulturní, případně sportovní akce je použit vyhledávací řetězec more like this. Datová sada je ve výchozím nastavení periodicky získávána jednou za týden. Akce v indexu již evidované nejsou opětovně vkládány. Během přidávání nových akcí je spuštěn proces, který z indexu odebírá již proběhlé akce. Zde popsané funkcionality jsou implementovány objektem instanciovaným z třídy `culture_action`. Diagram tříd je uveden na obrázku 3.6.

Posledním zdrojem informací pro uživatele je předpověď počasí. Při dotazu na počasí je uživateli prezentován aktuální stav počasí a výhled na dva dny dopředu. V rámci odpovědi je zobrazen textový popis počasí, ilustrační piktogram, teplota a síla větru. Tato data jsou systémem ukládána, přičemž data jsou platná vždy do půlnoci dne, kdy byla obdržena. Metody realizující v tomto odstavci popsané postupy jsou implementovány třídou `weather_data`. Diagram tříd je poté zobrazen na obrázku 3.6.

3.7 Implementace aplikačního serveru

Výše uvedené postupy, s výjimkou vyhledávání faktické odpovědi na uživatelský dotaz s použitím polostrukturovaných dat, dohromady implementují aplikační server. Diagram tříd pro aplikační server je uveden na obrázku 3.7. Pro odpovídání využívá aplikační server objektů třídy `sas` a `qas` popsané v sekci 3.6. Pro určení typu odpovědi na přijatý dotaz je využit objekt třídy `qtd` uvedený v sekci 3.5. Popis síťové komunikace aplikačního serveru je uveden v kapitole 3.1.

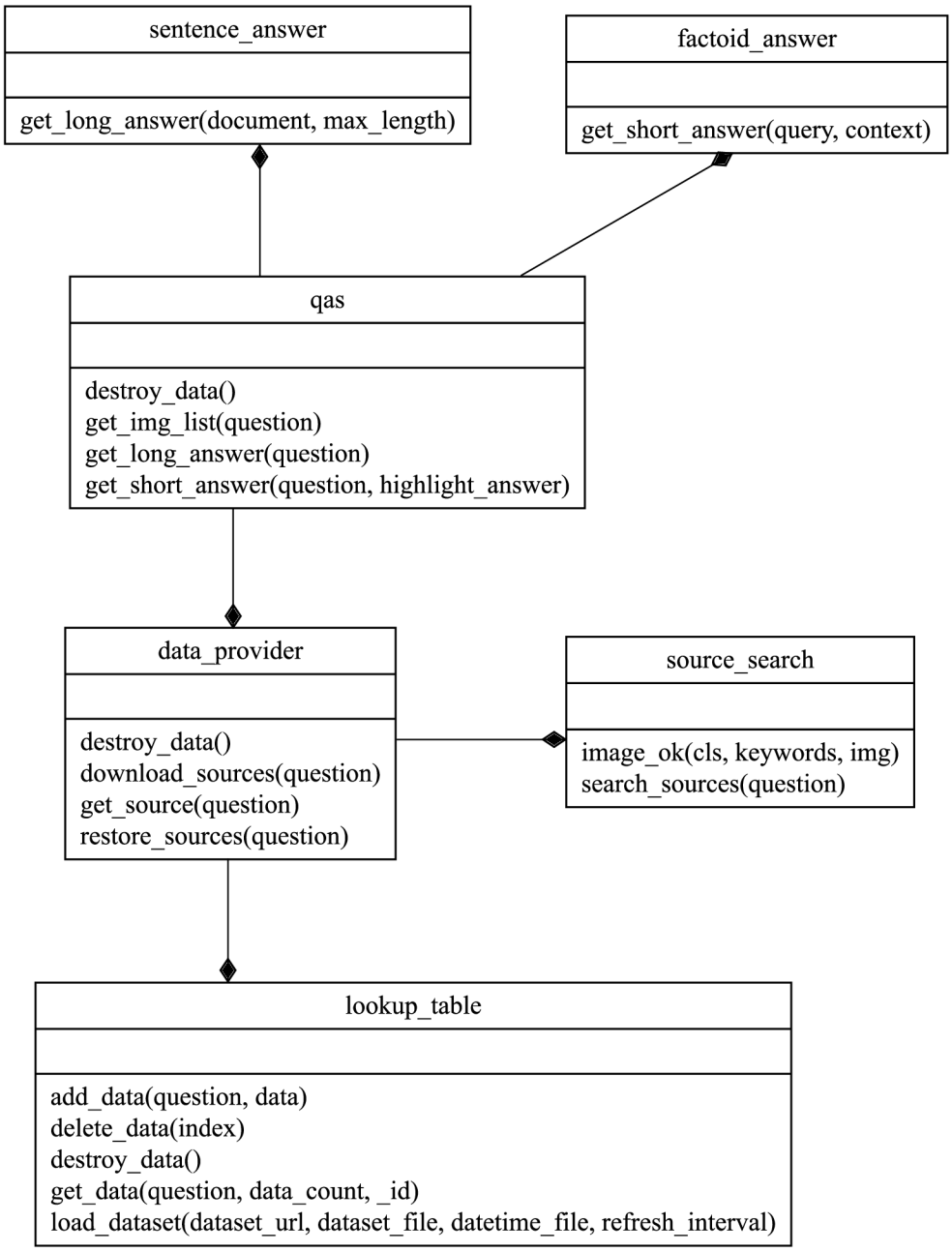
Po prvotním spuštění je serverem zajištěno získávání datových sad a dalších zdrojů, které jsou následně ukládány s pomocí systému Elasticsearch či jako perzistentní objekty s definovanou dobou expirace. Objekty realizující modely pro klasifikaci otázek a vyhledávání klíčových slov jsou implementovány jako perzistentní. Při opakovaném spuštění tak

není nutné opětovně provádět fázi strojového učení. Obdobně nedochází ke znovu získávání datových sad v případě, že nedošlo k jejich expiraci.

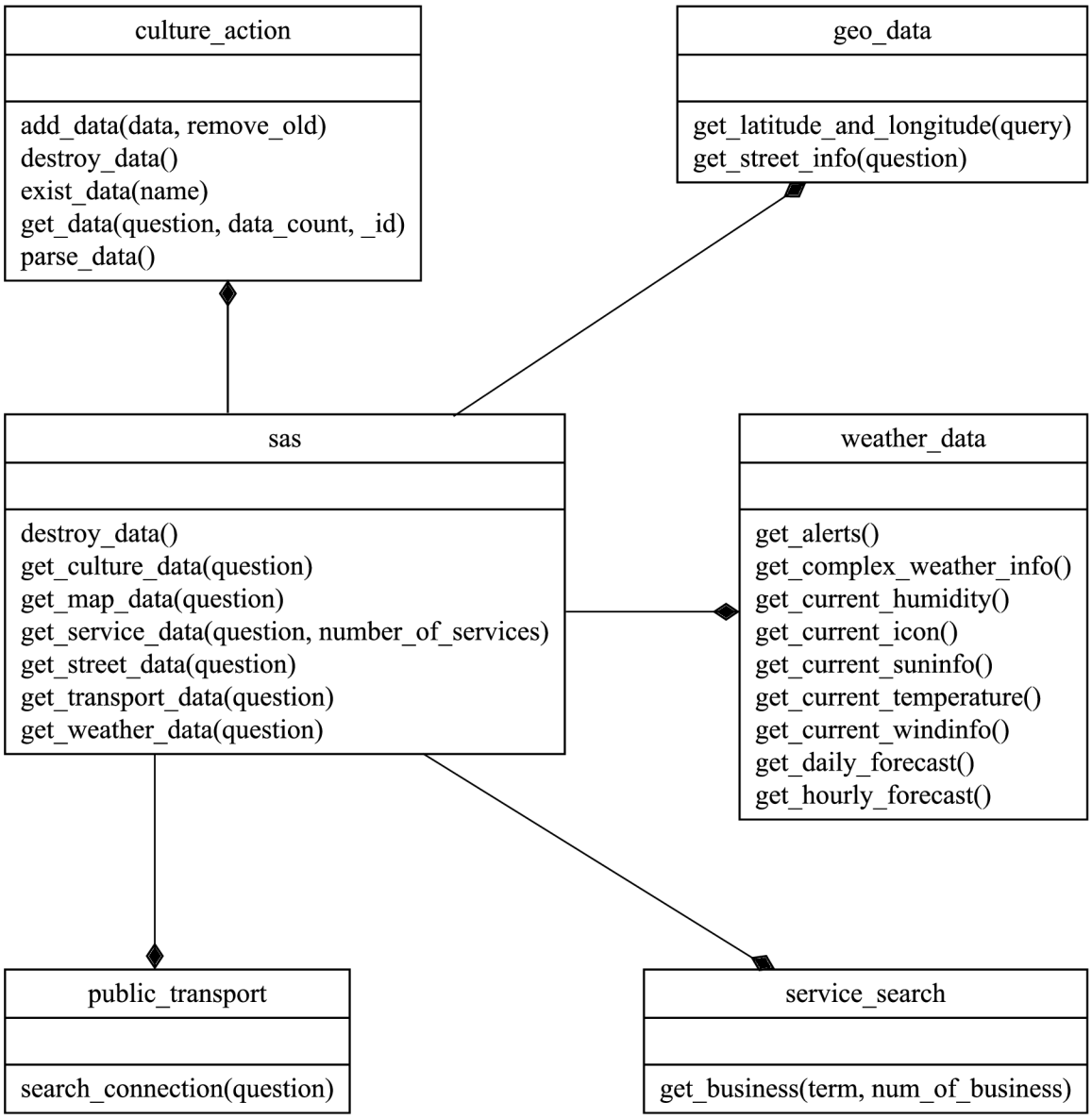
Před spuštěním je nutná existence serverového nastavení, které uvádí URL adresy, z nichž je možné získat používané datové sady, doby experací jednotlivých sad, názvy souborů pro uložení perzistentních objektů a další. Podrobný přehled nastavení serveru je uveden v příloze A. Kontrolu správnosti nastavení a případné doplnění volitelných polí realizuje objekt třídy `src`, který následně data zpřístupňuje skrze veřejné členské proměnné.

Serverová odpověď na HTTP dotaz obsahuje buď data odpovídající na otázku, nebo popis zjištěné chyby. Mezi možné chyby, na něž je možné narazit, se řadí nenalezení přesné faktické odpovědi, nemožnost geolokace objektu reálného světa v mapě, neexistence dat pro požadovanou ulici, nenalezení seznamu služeb pro danou otázku a neidentifikování výchozí a cílové zastávky hromadné dopravy. Vyjma těchto chyb je dále možné detekovat případné chyby vlastního serveru.

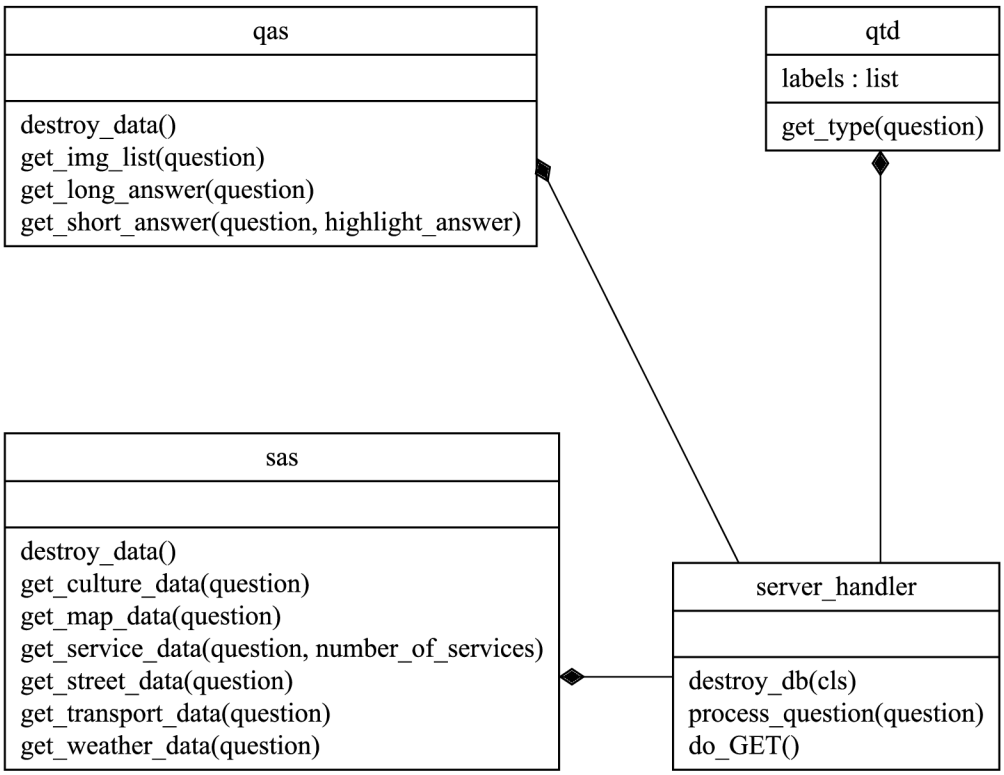
Data odpovídající na dotaz jsou serializována jako kolekce struktur, kde položkami struktury je interní typ dat a vlastní odpověď. Server podporuje následující interní typy dat: textová data, odkaz na zdroj dat, obrazová data, informace o počasí, zeměpisná šířka a délka, informace o spoji MHD, seznam služeb, hlasová data a data ve formě tabulky. Data nesoucí odpověď pak mohou být vložena do HTML fragmentu.



Obrázek 3.5: Diagram tříd subsystému pro odpovídání na dotazy s využitím polostrukturovaných dat



Obrázek 3.6: Diagram tříd subsystému pro odpovídání na dotazy s využitím strukturovaných dat



Obrázek 3.7: Diagram tříd aplikačního serveru

Kapitola 4

Experimenty a vyhodnocení

Tato kapitola uvádí provedené experimenty, které mají za úkol otestovat a zhodnotit funkcionalitu výsledného systému komunikačního agenta pro informace o městě Brně. Teoretické principy, na nichž agent staví, jsou blíže popsány v kapitole 2. Implementační detaily jednotlivých částí agenta jsou poté uvedeny v kapitole 3. Použité evaluační techniky popisuje sekce 2.6.

4.1 Evaluace systému pro extrakci klíčových slov

Systém pro extrakci klíčových slov z uživatelských otázek byl evaluován s použitím metrik accuracy, precision, recall a F1 skóre. Pro stanovení hodnot metrik pro zkoumaný systém byla použita testovací datová sada čítající 75 otázek a k nim vázajících se seznamů klíčových slov. Testovací datová sada obsahuje otázky ze všech tematických okruhů, na něž dokáže komunikační agent odpovídat. Datová sada použitá pro strojové učení modelu je shodná s datovou sadou použitou pro učení modelu v rámci vlastního systému komunikačního agenta. Shodné jsou taktéž používané slovníky stop-slov. Výsledky jsou uvedeny v tabulce 4.1.

Metrika	**Hodnota**
accuracy	0.9417
precision	0.9395
recall	0.9352
f1 skóre	0.9374

Tabulka 4.1: Výsledky evaluace systému pro extrakci klíčových slov

4.2 Evaluace systému pro klasifikaci uživatelských otázek

Evaluace systému klasifikujícího typy odpovědí na uživatelské otázky je založena na metrikách accuracy, precision, recall a F1 skóre. Při tomto procesu byla použita testovací datová sada obsahující 96 otázek a k nim odpovídajících tříd očekávaných odpovědí. Testovací datová sada obsahuje pro každou kategorii 12 testovacích vstupů. Datová sada určená pro strojové učení modelu je totožná s datovou sadou využitou při učení modelu v rámci systému komunikačního agenta. Výsledky evaluace jsou uvedeny v tabulce 4.2.

Metrika	**Hodnota**
accuracy	0,9271
precision	0,9271
recall	0,9539
f1 skóre	0,9403

Tabulka 4.2: Výsledky evaluace systému pro klasifikaci uživatelských otázek

4.3 Uživatelské testování

Za účelem otestování použitelnosti výsledného komunikačního agenta pro informace o Brně bylo provedeno uživatelské testování. Do uživatelského testování byla zahrnuta skupina devíti osob ve věkovém rozmezí 16 až 23 let. Účastníci testování mají trvalé či přechodné bydliště v Brně, případně do města denně dojíždí. Participanti dále ovládají anglický jazyk alespoň na úrovni A1 dle společného evropského referenčního rámce. Samotné testování bylo rozděleno na dílčí testy oddělené časovým odstupem. Tato sekce blíže uvádí popis provedených testů a jejich vyhodnocení.

Testy použitelnosti grafického rozhraní probíhaly s využitím prototypu výsledného systému. Před vlastním testováním GUI byla uživatelům podílejícím se na testování popsána řešená doména, způsob komunikace a byly uvedeny tematické okruhy otázek, pro které dokáže agent nalézt odpověď. Poté probíhalo vlastní testování s využitím metody moderovaného pozorování uživatele při práci s rozhraním, po němž následovalo individuální interview. Úkolem participantů testování bylo vyhledat informace dle předem daných scénářů. Pozorovanými vlastnostmi byly zejména rychlost pochopení užití grafického rozhraní a přehlednost grafické prezentace odpovědi. Bezprostředně po ukončení testování proběhla skupinová diskuze (anglicky focus group), na základě které byl vyvozen seznam dalších možných vylepšení grafického rozhraní.

Druhou fází uživatelského testování byly testy výsledného komunikačního agenta pro informace o Brně. Testy probíhaly s využitím grafického rozhraní odladěného a upraveného na základě první testovací fáze. Pro tuto etapu testování byla využita shodná skupina účastníků jako při fázi testování GUI. Zúčastněným uživatelům byly znovu popsány tematické okruhy, neboť došlo k jejich rozšíření. Testy probíhaly jako při první části s využitím metod moderovaného pozorování práce uživatele a individuálního interview. V rámci testování byly připraveny scénáře situací, v nichž je možné využít komunikačního agenta pro získání informací. Sledovanými jevy byly přehlednost upravené reprezentace odpovědí a spokojenost uživatele s odpovědí. Po dokončení proběhla skupinová diskuze na téma možných vylepšení komunikačního agenta z uživatelského pohledu.

Třetí fází testování bylo poskytnutí funkční webové aplikace participantům testování pro použití v každodenním životě. Po jednom týdnu bylo provedeno dotazníkové šetření

s hlavním cílem zjistit spokojenost uživatelů při používání v reálném světě. Mezi dalšími sledovanými hodnotami byla poté spokojenost s reakční dobou komunikačního agenta či tematický okruh nejčastěji pokládaných otázek. Do třetí etapy testování se zapojili tři další účastníci – návštěvníci města Brna.

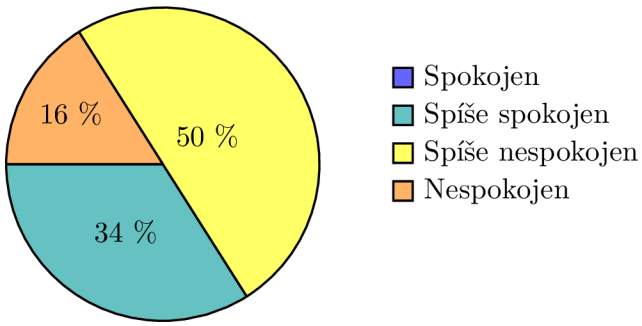
Na základě uživatelského testování grafického rozhraní byly implementovány funkcionality vylepšující uživatelský zážitek z interakce s grafickým rozhraním. Následuje výčet úprav provedených na základě testováním získaných poznatků:

- Uživatelé mohou používat nejen výchozí světlé barevné schéma, ale i alternativní schéma tmavé;
- Nastavení rozhraní (barevné schéma, aktivnost nastavení hlasového odpovídání) je ukládáno s využitím souboru cookies;
- Zobrazení seznamu služeb místo čistě textové podoby prezentuje vybrané služby i s pomocí ilustračního obrázku a uživateli udělenými hodnoceními;
- Pro prezentaci předpovědi počasí je využito rozložení do tabulky a jsou použity ilustrační piktogramy;
- Vyhledaná spojení městské hromadné dopravy jsou prezentována strukturovaně místo původního čistě textového zobrazení.

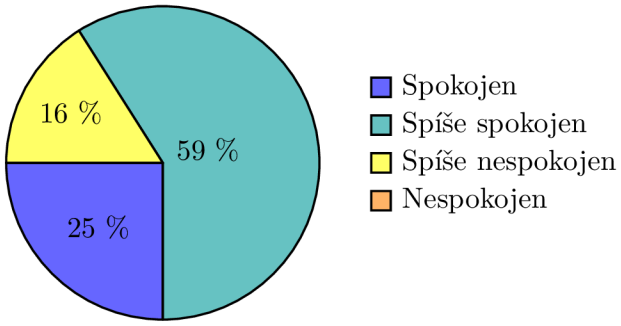
Na základě druhé fáze testování systému s pomocí prvotních uživatelů byly provedeny úpravy grafického rozhraní vedoucí k vyšší přehlednosti prezentace odpovědí na dotazy. Následuje seznam úprav grafického rozhraní:

- Zvýraznění zjištěné faktografické odpovědi ve větě získané z původního dokumentu;
- Přidání tlačítka odkazujícího na možnost přímého nákupu jízdního dokladu pomocí SMS zprávy;
- Realizace strukturované prezentace informací o ulicích a kulturních akcích.

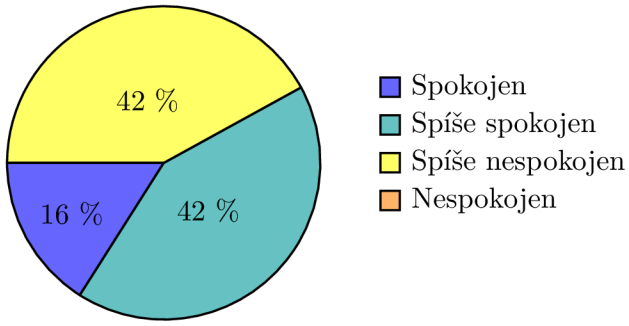
Po třetí fázi byla vyhodnocena spokojenost uživatelů se systémem. V rámci hodnocení spokojenosti byla sledována subjektivní ohodnocení času odezvy systému, přesnosti odpovědi a celkové spokojenosti. Dále byly zjišťovány typy uživateli nejčastěji pokládaných dotazů. Vyhodnocení třetí etapy je uvedeno v grafech na obrázcích 4.1, 4.2, 4.3 a 4.4.



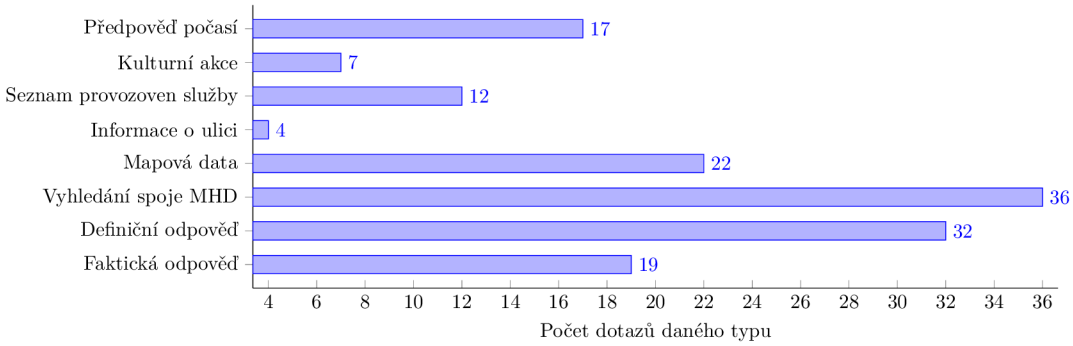
Obrázek 4.1: Subjektivní spokojenost uživatelů s časovou odezvou



Obrázek 4.2: Subjektivní spokojenost s přesností odpovědi



Obrázek 4.3: Celková subjektivní spokojenost se systémem



Obrázek 4.4: Frekventovanost typů odpovědí

Aplikační server, server pro odpovídání na faktické otázky a systém Elasticsearch byly nasazeny na serveru používajícím operační systém Windows 10 s procesorem Intel Core i7. Výpočty byly urychlovány s pomocí grafického procesoru GeForce MX250.

Uživateli nejčastěji zmiňovaný problém v druhé fázi je časová odezva systému při získávání odpovědi na faktické a definiční otázky. Tento problém je dán nutností kontaktování vyhledávače a stažení dat z poskytnutých webových zdrojů. Tento problém se vyskytoval méně častěji u participantů, kteří prováděli testování mezi posledními – systém měl požadované dokumenty uloženy v indexu systému Elasticsearch. V rámci skupinové diskuze druhé fáze uživatelé nejvíce ocenili vyhledávač spojů hromadné dopravy a mapových podkladů, jejichž časová odezva je hodnocena kladně.

Na základě uživatelského testování třetí fáze byly potvrzeny výsledky druhé etapy – nejčastěji zmiňovaným problémem je délka doby časové odezvy při získávání faktických a definičních odpovědí. Čtyři prvotní uživatelé dále poukázali na občasnou chybnou klasifikaci typu cílené odpovědi, zejména záměnu vyhledávání kulturních akcí a služeb. Polovina participantů po třetí fázi navrhla možnost kombinování typů odpovědí, nejčastěji byla zmiňována varianta zobrazení trasy mezi zastávkami při vyhledání spojení hromadné dopravy.

Kapitola 5

Závěr

Zadáním bakalářské práce bylo seznámení se s principy komunikačních agentů, návrh systému pro odpovídání na otázky týkající se města Brna a jeho ověření pomocí uživatelského testování. Tento cíl byl splněn prostřednictvím studia dané problematiky, implementace komunikačního agenta a návrhu a provedení testů s prvotními uživateli.

Prvním krokem práce bylo prostudování technologií umožňujících získávání a ukládání informací a dále mechanismů strojového učení a umělých neuronových sítí. Následně byla s využitím internetových zdrojů shromážděna prvotní data znalostní báze týkající se reálií Brna a poté vytvořen systém umožňující případné dohledávání dat. Další fází byl návrh a implementace systému využívající principy třívrstvé architektury. Text práce blíže objasňuje použité postupy a teoretické principy využívané pro realizaci jednotlivých částí systému. Komunikační agent pro informace o Brně byl vyhodnocen na základě uživatelského testování s prvotními uživateli. Grafické rozhraní umožňující dotazování je dostupné na URL adrese `http://www.stud.fit.vutbr.cz/~xkrist22/`. Jednotlivé komponenty aplikační vrstvy jsou nasazeny na školním serveru `merlin.fit.vutbr.cz`. Cíle a výsledky jsou stručně prezentovány pomocí doprovodného plakátu.

Komunikační agent umožňuje na základě dotazovaných dat poskytnout osm odlišných typů odpovědí, například poskytnutí popisu brněnské reálie, nalezení spoje MHD či vyhledání faktické odpovědi na otázku. S přesností odpovědí na otázky bylo spokojeno 84 % prvotních uživatelů. Spokojenost se systémem jako celkem činila 58 %.

Díky této práci jsem získal nové poznatky v oblasti strojového učení a seznámil se s principy zpracování přirozeného jazyka. Práce mi umožnila získat zkušenosti s plánováním práce na dlouhodobém projektu, návrhem komplexnějších systémů a tvorbou odborného textu.

Výslednou práci bych rád obohatil o poskytování složitějších mapových podkladů, například vyhledávání tras. Systém bych dále rozvinul o možnost psaní recenzí, například hodnocení služeb, dojmy z turistických bodů, a poskytnutí podrobných informací o zastávkách MHD, konkrétně bezbariérovost, dostupnost prodejního automatu jízdenek či přístřešku. Dalším zdokonalením agenta by mohlo být využití zpětné vazby od uživatelů, na základě které by bylo možné optimalizovat subsystémy agenta využívající prvky umělé inteligence. V rámci dalšího vývoje by agent mohl zjišťovat geografickou polohu uživatelů, na základě které by byly poskytovány služby dostupné v blízkém okolí. Dále by agent mohl pro uživatele vést historii vyhledávání, s pomocí které by bylo možné omezit zobrazování duplicitních informací.

Literatura

[1] Ali, N. H. a Ibrahim, N. S. Porter stemming algorithm for semantic checking. In: *Proceedings of 16th international conference on computer and information technology*. 2012, s. 253–258. Dostupné z: https://www.researchgate.net/profile/Noraida-Haji-Ali/publication/260385215_Porter_Stemming_Algorithm_for_Semantic_Checking/links/5584e9d708ae7bc2f448474f/Porter-Stemming-Algorithm-for-Semantic-Checking.pdf.

[2] Bird, S. *Natural language processing with Python*. Beijing Sebastopol, CA: O'Reilly Media Inc, červen 2009 [cit. 2021-04-07]. ISBN 978-0-596-51649-9.

[3] Buduma, N. a Buduma, N. *Fundamentals of deep learning: designing next-generation machine intelligence algorithms*. O'Reilly Media, září 2021. ISBN 9781492082187.

[4] Cvrček, V. a Richterová, O. *Příručka ČNK*. Listopad 2014 [cit. 2021-05-18]. Dostupné z: https://wiki.korpus.cz/doku.php/pojmy:precision.

[5] Dangeti, P. *Statistics for machine learning : build supervised, unsupervised, and reinforcement learning models using both Python and R*. Birmingham, UK: Packt Publishing, červenec 2017. ISBN 9781788291224.

[6] Ekman, M. *Learning Deep Learning: Theory and Practice of Neural Networks, Computer Vision, NLP, and Transformers using TensorFlow*. City: Addison-Wesley Professional, 2021. ISBN 9780137470198.

[7] Elasticsearch, B. *Terminology* [online]. Elasticsearch, B.V., 2021 [cit. 2021-07-03]. Dostupné z: https://www.elastic.co/guide/en/elastic-stack-glossary/current/terms.html.

[8] Fajčík, M. *BISSIT_19_NLP_excercise_QA_student*. 2019 [cit. 2021-01-08]. Dostupné pro studenty FIT VUT.

[9] Géron, A. *Hands-on machine learning with Scikit-Learn and TensorFlow : concepts, tools, and techniques to build intelligent systems*. O'Reilly Media, březen 2018. ISBN 978-1491962299.

[10] Kowsari, K., Jafari Meimandi, K., Heidarysafa, M., Mendu, S., Barnes, L. et al. Text Classification Algorithms: A Survey. *Information*. 2019, sv. 10, č. 4. DOI: 10.3390/info10040150. ISSN 2078-2489. Dostupné z: https://www.mdpi.com/2078-2489/10/4/150.

[11] Lederer, J. Activation functions in artificial neural networks: A systematic overview. *ArXiv preprint arXiv:2101.09957*. Leden 2021, [cit. 15. 5. 2021]. Dostupné z: https://arxiv.org/pdf/2101.09957.pdf.

[12] McCormick, C. *Blog McCormick Chris*. Duben 2016 [cit. 2021-02-27]. Dostupné z: http://mccormickml.com/2016/04/19/word2vec-tutorial-the-skip-gram-model/.

[13] Mézl, M. a Sekora, J. Bayesian Methods for Data Mining. In: *Student EEICT 2009* [print]. VUT v Brně, FEKT & FIT: VUT v Brně, FEKT & FIT, April 2009, kap. 32064, s. 1–3.

[14] Olsson, J. *Using Elasticsearch for full-text searches on unstructured data* [online]. Uppsala universitet, duben 2019 [cit. 2021-04-22]. Dostupné z: https://www.diva-portal.org/smash/get/diva2:1363672/FULLTEXT01.pdf.

[15] Pundge, A. M., Khillare, S. a Mahender, C. N. Question answering system, approaches and techniques: A review. *International Journal of Computer Applications*. Foundation of Computer Science. 2016, sv. 141, č. 3, s. 0975–8887.

[16] Rajpurkar, P., Zhang, J., Lopyrev, K. a Liang, P. Squad: 100,000+ questions for machine comprehension of text. *ArXiv preprint arXiv:1606.05250*. Říjen 2016.

[17] Ravichandiran, S. *Getting started with Google BERT : build and train state-of-the-art natural language processing models using BERT*. Packt Publishing, leden 2021. ISBN 9781838826239.

[18] Sokolova, M. a Lapalme, G. A systematic analysis of performance measures for classification tasks. *Information Processing & Management*. 2009, sv. 45, č. 4, s. 427–437. DOI: https://doi.org/10.1016/j.ipm.2009.03.002. ISSN 0306-4573. Dostupné z: https://www.sciencedirect.com/science/article/pii/S0306457309000259.

[19] Strossa, P. *Počítačové zpracování přirozeného jazyka* [online]. Praha: Oeconomica, 2011. ISBN 978-80-245-1777-3.

[20] Uzun, Y. Keyword extraction using naive bayes. In: *Bilkent University, Department of Computer Science, Turkey www. cs. bilkent. edu. tr/~ guvenir/courses/CS550/Workshop/Yasin_Uzun. pdf*. 2005 [cit. 2021-03-05]. DOI: 10.1.1.103.2128. Dostupné z: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.103.2128&rep=rep1&type=pdf.

[21] Zhang, S., Hu, Y. a Bian, G. Research on string similarity algorithm based on Levenshtein Distance. In: *2017 IEEE 2nd Advanced Information Technology, Electronic and Automation Control Conference (IAEAC)*. 2017, s. 2247–2251. DOI: 10.1109/IAEAC.2017.8054419.

Příloha A

Manuál

Pro korektní instalaci závislostí a spuštění aplikačního serveru a serveru pro odpovídání na faktické otázky je nutné mít nainstalován interpret jazyka Python3 minimální verze `3.8.6` a utilitu `make`. V kořenovém adresáři projektu je vytvořen skript `Makefile`. Tento umožňuje spuštění systému pod operačním systémem Microsoft Windows (testováno na osobním počítači s operačním systémem Windows 10) a GNU/Linux (testováno na školním serveru merlin). Skript implementuje příkaz `install-win` a `install-unix`. Ten provede stažení spustitelných souborů systému Elasticsearch a závislostí s pomocí utility `pip`. Závislosti systému jsou uvedeny v souboru `requirements.txt` umístěném v kořenovém adresáři. Systém Elasticsearch je distribuován na k tomu určené webové stránce https://www.elastic.co/downloads/elasticsearch. Před vlastní instalací je doporučeno vytvoření virtuálního prostředí.

Veškeré komponenty komunikačního agenta je možné spustit s pomocí příkazu `make run-win`, respektive `make run-unix`. Při spuštění je nutné uvést následující parametry:

- `application-server-ip`: IP adresa přidělená aplikačnímu serveru (IP adresa počítače, případně `localhost`);
- `application-server-port`: port, na němž bude aplikační server naslouchat;
- `factoid-server-port`: port, na němž bude naslouchat server pro faktické odpovídání.

Skript `Makefile` zahájí spouštění systému Elasticsearch a serveru pro odpovídání na faktické otázky. V případě nedostupnosti rozhraní Elasticsearch využívá agent přímo vyhledávání webových zdrojů pomocí vyhledávače Google a neumožňuje poskytování informací o kulturních akcích. Před zahájením spouštění aplikačního serveru je nutné úspěšně uvést do chodu alespoň server pro odpovídání na faktické otázky.

Provedení testů je možné vyvolat příkazem `make test-win` pro operační systém Windows 10, případně `make test-unix` pro operační systémy typu GNU/Linux. Před vlastním spuštěním testů je nutné provést instalaci závislostí. Testování se týká základních utilit systému. Následně je provedena evaluace systémů využívajících prvky umělé inteligence – klasifikátoru typu odpovědi a subsystému pro extrakci klíčových slov z uživatelských dotazů.

Při spouštění aplikačního serveru je používán konfigurační soubor, v němž jsou uvedeny URL odkazy, z nichž je možné získat datové sady. Dále konfigurační soubor uvádí názvy souborů, do nichž jsou datové sady uloženy a názvy souborů ukládající datum expirace jednotlivých datových sad a další perzistentní objekty systému. Dále soubor obsahuje klíče nutné pro používání některých aplikačních rozhraní a další užitečné informace. Data konfiguračního souboru jsou serializována s pomocí formátu JSON. Jméno souboru musí být

nastaveno na `server_setup.json` a tento musí být lokalizován v adresáři, v němž se nachází program aplikačního serveru. Na přiloženém paměťovém médiu je uveden konfigurační soubor `src/application/server_setup.json`, který je možné použít pro spuštění. Tento soubor navíc uvádí veškerá povinná i volitelná data.

V případě použití webového grafického rozhraní je možné nastavit URL adresu serveru redefinováním hodnoty proměnné `application_server_url` v souboru `src/presentation/scripts/source.js`. Tento soubor ukládá pouze zmíněnou proměnnou.

Příloha B

Obsah přiloženého paměťového média

Tato kapitola popisuje obsah přiloženého paměťového média (karty SD). Následující seznam uvádí základní přehled obsahu:

- `doc/technical_report`: zdrojové soubory textu bakalářské práce;
- `files`: složka obsahující datové sady vytvořené a využívané v rámci projektu;
- `src/application`: složka obsahující zdrojové soubory systému realizující aplikační vrstvu v jazyce Python3;
- `src/presentation`: adresář obsahující zdrojové soubory prezentační vrstvy implementované jazyky HTML, CSS a JavaScript;
- `src/application/server_setup.json`: konfigurační soubor aplikačního serveru;
- `doc/poster/poster.svg`: plakát prezentující práci ve formátu `svg`;
- `doc/poster/poster.pdf`: plakát prezentující práci ve formátu `pdf`;
- `Makefile`: skript umožňující instalaci závislostí a spouštění serveru s pomocí utility `make`;
- `README.md`: soubor README obsahující základní popis projektu a uživatelskou příručku;
- `requirements.txt`: soubor obsahující seznam závislostí;
- `src/presentation/scripts/source.js`: soubor obsahující proměnnou definující adresu aplikačního serveru.