

VYSOKÁ ŠKOLA EKONOMICKÁ



---

Metody zpracování přirozeného anglicky  
psaného textu v prostředí R pro účely  
statistické reprezentace jazyka

---

Lubomír Štěpánek

červen 2017

(2017) Lubomír Štěpánek, CC BY-NC-SA 3.0 (CZ)



Dílo lze dále svobodně šířit a dokonce i upravovat, ovšem za dodržení stejné licence, tedy s uvedením původního autora a s uvedením stejné licence. Dílo ani jeho derivát není možné šířit komerčně ani s ním jakkoliv jinak nakládat pro účely komerčního zisku. Autor neručí za správnost informací uvedených kdekoliv v předložené práci, přesto vynaložil nezanedbatelné úsilí, aby byla uvedená fakta správná a aktuální, a práci sepsal podle svého nejlepšího vědomí a svých „nejlepších“ znalostí problematiky.

# Obsah

<b>1</b>	<b>Úvod</b>	<b>5</b>
<b>2</b>	<b>Workflow získání a zpracování textu</b>	<b>6</b>
<b>3</b>	<b>Metody získání a zpracování textu</b>	<b>9</b>
3.1	Získání korpusu	9
3.2	Webscraping	9
3.2.1	Webscraping anglické Wikipedie	9
3.3	Dělení textu do vět	11
3.4	Preprocessing textu	12
3.5	Tokenizace	12
3.6	Odstranění stop slov a dalších	13
3.7	<i>n</i> -gramming	13
<b>4</b>	<b>Aplikace the_next_word_prediction</b>	<b>15</b>
4.1	Princip aplikace	15
4.2	Komponenty aplikace	16
<b>5</b>	<b>Závěr a další možné směřování práce</b>	<b>18</b>
<b>6</b>	<b>Literatura</b>	<b>19</b>
<b>7</b>	<b>Rejstřík</b>	<b>20</b>
<b>8</b>	<b>Apendix</b>	<b>21</b>
8.1	Procedura <code>__main__.R</code>	21
8.2	Procedura <code>initialization.R</code>	22
8.3	Funkce <code>helper_functions.R</code>	23
8.4	Procedura <code>webscraping.R</code>	28
8.5	Procedura <code>corpora_loading.R</code>	30
8.6	Procedura <code>text_splitting.R</code>	32
8.7	Procedura <code>preprocessing.R</code>	33
8.8	Procedura <code>tokenization.R</code>	34
8.9	Procedura <code>postprocessing.R</code>	35
8.10	Procedura <code>n_gramming.R</code>	36
8.11	Komponenta <code>ui.R</code>	39
8.12	Komponenta <code>server.R</code>	45
8.13	Komponenta <code>www/style.css</code>	51
8.14	Komponenta <code>my_1_word_vocabulary.txt</code>	52



# 1 Úvod

Účelem této minimalistické práce je nabídnout a popsat jeden z možných způsobů, jak zpracovat relativně rozsáhlý, anglicky psaný text pro účely jeho následné statistické reprezentace.

Autor ve výkladu systematicky popisuje proces zpracování textu přirozeného jazyka od začátku do konce a bez zbytečného formalismu; v podstatě využívá řešení úlohy nad konkrétními daty (korpusem), což posiluje praktický aspekt celé práce.

Výchozím bodem pro zpracování přirozeného (anglického) textu je možnost nakládat s textovým korpusem daného jazyka. Získaný korpus je poté v několika na sebe navazujících fázích postupně zpracováván tak, že z původních vět korpusu nakonec vzniknou uspořádané seznamy slov a kratších sousloví pevných délek, tzv.  $n$ -gramů. Pomocí nich pak lze jazyk tzv. statisticky reprezentovat, což umožňuje řešení několika zajímavých úloh nad přirozeným jazykem. Jedna z nich je rovněž popsána – jde o predikci  $n$ -tého slova, které by v daném jazyce mělo s největší pravděpodobností následovat po zadané  $(n - 1)$ -členné frázi. Pro účely real-time řešení této úlohy byla publikována webová aplikace, jejíž popis a zdrojový kód je rovněž součástí této publikace.

Funkce a procedury, jež řeší dále uvedené fáze zpracování textu korpusu, jsou implementovány v prostředí a programovacím jazyce R. Znalost jazyka R však není nutnou podmínkou pro čtení této práce; na následujících stránkách jsou popsány vesměs obecné principy zpracování textu přirozeného jazyka a jsou uvedeny i obecné algoritmy.

V této práci je současně popsán i způsob, jak získat vlastní textový korpus, opět s využitím již naimplementovaných procedur a funkcí v jazyce R.

Ve snaze posílit „edukační“ rozměr této práce jsou v relativně rozsáhlém apendixu uvedeny veškeré zdrojové kódy opatřené volnotextovými komentáři; kódy postačují pro reprodukci všech uvedených procedur zpracování textu přirozeného jazyka.

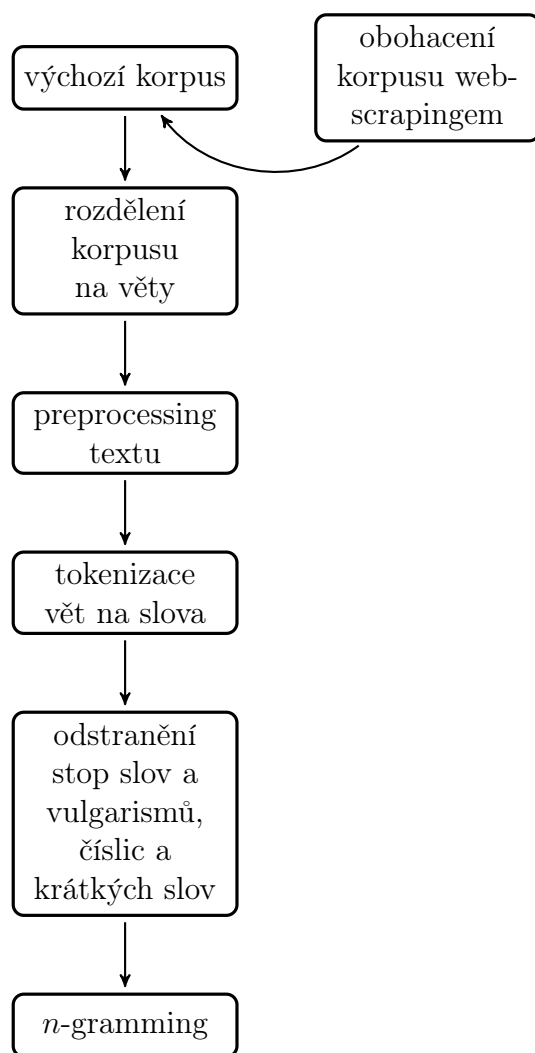
Práce splní svůj smysl, pokud čtenáře zaujme byť i jedna jediná myšlenka, která je v následujícím textu uvedena.

## 2 Workflow získání a zpracování textu

Zpracování textu přirozeného jazyka patří mezi úlohy oblasti *natural language processing* (NLP), kam lze v širším slova smyslu řadit kromě zpracování textů i práci s netextovými záznamy daného jazyka.

My se zde omezíme pouze na práci s textovými zdroji daného přirozeného jazyka, konkrétně na jako reprezentaci formou rozsáhlého souboru (ne nutně konzistentních) vět psaných v daném jazyce, tedy formou tzv. *korpusu*.

Jednotlivé procedury zpracování textu přirozeného jazyka na sebe navazují, jak ukazuje obrázek 1.



Obrázek 1: Schéma (zjednodušené) asynchronních úloh během zpracování textu pro účely jeho statistické reprezentace ( $n$ -grammingu)

V tomto pojednání se zaměřujeme na zpracování textů psaných anglicky, neboť angličtina je analytický jazyk [1], jehož textové a jazykově-statistické zpracování je při dnešních možnostech relativně dobře zvládnutelné, a to především díky minimálnímu zatížení anglických textů morfematičností ohebných slovních druhů; *lemmatizaci* slov, tedy jejich převod na morfologicky základní tvar, lze v anglickém

textu v podstatě vynechat, aniž by byla ohrožena kvalita analýzy textu i konečného  $n$ -gramingu<sup>1</sup> Proto ve schématu na obrázku 1 chybí fáze lemmatizace.

Výchozím bodem je stav, kdy je k dispozici existující korpus anglických textů (v následujících statích bude zpracovávána část tzv. *Helsinki corpora* [6]; jde o anglické texty různého původu – pocházejí z anglických zpravodajských relací, blogů a z twitteru). K nim byl přidán ještě menší korpus získaný vlastním webscrapingem malé části anglicky mluvící Wikipedie.

Korpusy je poté možné dohromady jednotně předzpracovat. Bloky textů korpusů jsou nejdříve rozděleny na věty, a to podle určité heuristiky (viz dále).

Dále je text tvořený větami v rámci preprocessingu očištěn o některé nadbytečné znaky (především interpunkci); v závěru této fáze jsou korpusy tvořeny oddělenými větami, které tvoří písmena malé a velké anglické abecedy a mezery. Velká písmena byla v rámci redukce „variability“ textu převedena bez ztráty informace na malá písmena. Rovněž je v této fázi žádoucí oprostit slova vět o ty, které nesou pouze kontextovou informaci, zde jde především o číslice. Některá velmi krátká slova (obvykle o délce kratší než tři písmena) lze považovat za výsledky odstranění interpunkce (zejména apostrofu v rámci zkratkovým forem, např. kdy z „i’m“ jakožto původně tříznakového „slova“ zbude pouze „i“ a „m“), kdy obvykle vznikají jen oddělené zkrácené části fráze, které má smysl také odstranit.

Následuje fáze tokenizace vět na jednotlivá slova, výsledkem je pro daný korpus množina uspořádaných, různě dlouhých  $k$ -tic slov, kde  $k \geq 1$  je pro každou větu nějaké přirozené číslo označující počet slov této věty. V této fázi je možné z vět odstranit ta slova, která v textech nejsou z nějakého smysluplného důvodu žádoucí. Jde především o vulgarismy a jinak nevhodná slova, dále je možné (ale ne nutné) odstranit tzv. stop slova – jde obvykle o často se objevující slova, většinou neohebných slovních druhů (není pravidlem), která nenesou příliš zajímavou informaci.

Nyní by v případě flekčního jazyka následovala fáze lemmatizace, tedy snížení variability slov (konkrétně ohebných slovních) jejich převodem vždy na základní tvar. Tato podúloha je však netriviální, naštěstí v případě anglického jazyka, jak již bylo zmíněno, ji lze vynechat bez významného snížení kvality předzpracování jazyka. Morfologie anglických slov zahrnuje v podstatě jen přidání koncovek „-s“ či „-es“ v případě plurálu.

Následně, když jsou korpusy zpracovány do té fáze, že jde o velké množství uspořádaných  $k$ -tic vždy pro přirozená  $k \geq 1$  lišící se mezi původními větami, je možné text korpusů uchopit z pohledu statistického modelování jazyka a nalézt v textu tzv.  $n$ -gramy pro malá  $n$ , typicky  $n \in \{2, 3, 4, 5\}$ , ale i vyšší. Jde o  $n$ -členná sousloví, tedy  $n$ -tice sousedících, po sobě jdoucích slov[2]. Po sestavení  $n$ -gramů je možné vytvořit tabulku četností pro jednotlivá  $n$ -členná slovní spojení a získat tak bodové odhady apriorních pravděpodobností, s jakými se v textu daného přirozeného jazyka vyskytují.

Takové tabulky četností  $n$ -gramů pro  $n \in \{2, 3, 4, 5\}$  či vyšší jsou základem pro algoritmy předpovědi takového slova, které by v textu daného přirozeného jazyka následovalo s největší pravděpodobností po zadaných  $n - 1$  slovech tvořících jazykové

<sup>1</sup>Opakem by byla čeština, tedy flekční jazyk, ve kterém je jazyková analýza prakticky vždy závislá na kvalitě lemmatizace kvůli bohaté morfologii s relativně vysokou mírou nepatternových výjimek.

sousloví.

K tomuto účelu byla vytvořena webová aplikace `the_next_word_prediction`, která uživateli po zadání  $(n - 1)$ -členného slovního spojení v daném jazyce vrátí nejpravděpodobněji následující  $n$ -té slovo. Její principy fungování budou rovněž popsány.



### 3 Metody získání a zpracování textu

V rámci relativně rozsáhlé fáze preprocessingu a processingu textových dat je nutné provést několik asynchronních úloh (tedy „jednorázových předpočítání“), které jsou obvykle náročné na vývojový i exekuční čas. V následujících statích jsou jednotlivé části úlohy naprogramovány a řešeny v prostředí R, které je určeno pro statistické výpočty a následné grafické náhledy [3].

Veškerý komentovaný zdrojový kód je rovněž uveden appendixu publikace. I přes bohatou řadu knihoven pro *text mining* a *natural language processing*, kterou má jazyk R k dispozici, jsou všechny funkce napsány „de novo“ vlastními silami a jen s využitím základních klauzulí jazyka R (a to i ze cvičných důvodů); tedy bez použití dostupných intermediálních či vyšších vestavěných funkcí. Všechny procedury naráz spustí kód 1 a 2, pro všechny skripty viz část 8.

#### 3.1 Získání korpusu

Jak již bylo naznačeno, základním krokem je získání dostatečně velkého textového základu pro další možné zpracování, tedy korpusu.

V současnosti existuje již velká řada anglických korpusů, některé z nich jsou za určitých podmínek k dispozici pro nekomerční a badatelské účely.

Jmenujme např. Corpus of Contemporary American English [4], který nabízí 520 milionů slov, British National Corpus [5] obsahující 100 milionů slov nebo tzv. helsinské korpusy anglických textů postavené na cca 1,5 milionu slov [6].

Obvykle není možné korpusy volně stáhnout. V případě helsinských korpusů však je text, který vychází z jejich částí týkajících se anglických zpravodajských relací, anglicky psaných blogů a anglicky psaných tweetů sociální sítě **twitter**, možné získat v rámci absolvování masivního otevřeného online kurzu (MOOC) *Data Science* na online univerzitě Coursera<sup>®</sup>, který nabízí John Hopkins Bloomberg School of Public Health. Tato malá část původního korpusu je ke stažení např. zde. Vzhledem k tomu, že kurz je volně dostupný a během něj je nakládání s odkazovanými částmi korpusů zcela v režii účastníka kurzu, zdá se, že nakládání s odkazovaným výňatkem korpusu není nelegitimní. Soubor v odkazu obsahuje i korpusy jiných jazyků než angličtiny.

Jsou-li korpusy uloženy jako volný text (`.txt`) ve vhodné složce, jsou do konzole R nahrány kódem 5.

#### 3.2 Webscraping

Smyslem webscrapingu je stáhnout textový obsah nějakého počtu anglicky psaných webových stránek, zde za účelem zvětšení celkového rozsahu použitého korpusu. Zároveň lze předpokládat, že „korpus“ bude obohacen nejen kvantitativně ve smyslu zvětšení jeho rozsahu, ale do jisté míry i kvalitativně – zde samozřejmě záleží na úrovni jazykové kvality stránek, které budou „scrapovány“.

##### 3.2.1 Webscraping anglické Wikipedie

Původní korpus může být rozšířen (nejen) o texty z anglické Wikipedie. Současné mohou tyto texty sloužit i samostatně jako plnohodnotný korpus anglického

jazyka.

Výhodou článků na Wikipedii je fakt, že jejich obsah (a v podstatě i většina formy) je uložen jen pomocí stacionárního HTML (HyperText Markup Language). Formátování pomocí kaskádových stylů (CSS, Cascading Styl Sheets), respektive aditivní javascriptovou funkcionalitu je sice na stránkách Wikipedie možné použít díky sdílení stylů a funkcí na Wikimedia Commons, běžné to rozhodně není; proto lze Wikipedii považovat za dobrý zdroj *hard-typed* obsahu s pravidelnou (HTML) strukturou. Současně lze očekávat, že volný text ze stránek Wikipedie bude mít solidní gramatickou úroveň, půjde často o komplikovaná a dlouhá souvětí s hojným zastoupením idiomatických frází typických i pro akademickou anglickou mluvu. Nehrozí tak, že by mohl být původní korpus obohacením o text z anglicky psané Wikipedie znehodnocen.

Pro účely webscrapingu jedné stránky Wikipedie byla napsána funkce

`webscrapeMyWikipediaPage()`,

jejíž kód v jazyce R je uveden v chunku 3.

Popišme nyní neformálně princip, jakým funkce `webscrapeMyWikipediaPage()` funguje. Jejím vstupem je URL některé libovolné stránky typu článku anglické Wikipedie a výstupem je jeden textový řetězec, který je volným textem extrahovaným z textového obsahu stránky dané URL adresou. Funkce tedy desktopově stáhne veškerý HTML obsah dané URL adresy, poté extrahuje všechny řádky HTML kódu, které jsou ohraničeny HTML tagy `<p>` a `</p>`; ty totiž vymezují třídu HTML textu typu **paragraph** a víceméně jako jediné obsahují v rámci stránky volný text. Naopak ostatní HTML objekty typu nadpisy (vymezené tagy `<h1>` a `</h1>`), tabulky (vymezené tagy `<table>` a `</table>`) apod. volný text neobsahují, resp. si tím nemůžeme být jistí, proto v rámci vysoké specifity scrapování pouze volného textu je obsah mezi tagy jinými než `<p>` a `</p>` ve výstupu vynechán.

Poté, co je extrahován volný text z HTML třídy **paragraph**, je ještě prosycen jinými HTML tagy (`<...>...</...>`), HTML entitami (`&...;`) či wikipedickými tagy (obvykle `[...]`). S aplikací Chomského hierarchie jazyků a Chomského pravidla [7], že jazyk vyšší úrovně je třeba „značkovat“ jazykem nižší úrovně (ve skutečnosti obecnějším, tedy metajazykem), byly už tak relativně obecné HTML klauzule z textu extrahovány pomocí regulárních výrazů. Snadno nahlédneme, že regulární výraz `’<.*?’` odstraní všechny HTML tagy – vyhledá totiž veškerý obsah `(.)` v libovolném množství `(*)` mezi úhlovými závorkami (`<...>`), ale tak, aby např. v řádku `’<p>Ahoj světe</p>’` odstranil pouze HTML tag, nikoliv celý řádek; to vyjádříme otazníkem `?`, který regulárnímu výrazu říká „don’t be greedy“, tedy matchuje pouze nejkratší podřetězec daný regulárním výrazem. Obdobně regulární výraz `<&.*?>` matchuje „nehladově“ všechny HTML entity a výraz `’\\[.*? \\]’` naopak wikipedické tagy. Výsledkem takového očištění textových dat, původně charakteru HTML kódu, je pak volný text obsahující prakticky jen písmena abecedy, interpunkci a číslovky.

Kromě sestavení volného textu funkce z HTML obsahu extrahuje i všechny interní webové linky v rámci anglické Wikipedie; opět pomocí regulárního výrazu matchujícího pattern `href=’/wiki/...’` typický pro wikipedický interní link.

Zmíněnou funkci následuje procedura, viz chunk kódu 4, která na vstupu použije jeden zvolený článek, z něj vyextrahuje všechny interní outlinky vedoucí na jiné stránky typu článek anglické Wikipedie a současně scrapuje text tohoto článku.

Outlinky byly uloženy do globální proměnné `my_links` s linky. V druhé iteraci je vyscrapována stránka (a do stejné globální proměnné s linky jsou uloženy její outlinky) dostupná z prvního linku v globální proměnné s linky. V třetí iteraci je proces opakován se stránkou odkazovanou druhým linkem v globální proměnné s linky `my_links`. Proces je opakován do chvíle, než je získáno určité, uživatelem definované množství vyscrapovaných wikipedických stránek<sup>2</sup>.

Lze však odvodit, že uvedená webscrapovací funkce a procedura mohou v konečném čase teoreticky vytvořit rozsáhlý korpus ze stránek Wikipedie (budou-li tyto vzájemně dostatečně propojené interními linky), a to nejen anglické (algoritmus není citlivý na scrapovaný text).

### 3.3 Dělení textu do vět

Pro tuto úlohu byla implementována funkce

```
splitTextIntoSentences(),
```

jejímž vstupem je textový řetězec celého odstavce a výstupem je určitý počet jednotlivých vět, na které je odstavec funkcí rozdělen 6. Algoritmus využívá regulárních výrazů, a sice je založen na pozorování, že na hranici dvou vět se vyskytuje prakticky vždy jeden z následujících patternů znaků:

- sekvence tečka, mezera (žádná, jedna, nebo i více), velké písmeno – tato sekvence je matchována regulárním výrazem `''\.\s*[A-Z]+''`,
- sekvence otazník, mezera (žádná, jedna, nebo i více), velké písmeno – tato sekvence je matchována regulárním výrazem `''\?\s*[A-Z]+''`,
- sekvence vykřičník, mezera (žádná, jedna, nebo i více), velké písmeno – tato sekvence je matchována regulárním výrazem `''\!\s*[A-Z]+''`,
- sekvence dvojtečka, mezera (žádná, jedna, nebo i více) – tato sekvence odděluje nejspíše větu hlavní od věty vedlejší, což můžeme sémanticky vnímat jako dvě různé věty (alespoň pro účely *n*-grammingu); sekvence je matchována regulárním výrazem `''\:\s*''`.

Funkce relativně spolehlivě rozdělí libovolně dlouhý korpus o libovolném počtu vět na jednotlivé věty. Limitací funkce mohou být např. zkratky titulů, které budou matchovány první uvedenou sekvencí, ale ve skutečnosti nejsou na hranici dvou vět (v případě titulu *Ph.D.* je tento maskou pro pattern `''\.\s*[A-Z]+''`, dvě věty od

<sup>2</sup>Hypoteticky by mohla být procedura nechtěně ukončena i dříve – a to např. tehdy, kdyby volba iniciální stránky nebyla vhodná, protože by tato neobsahovala žádné outlinky, resp. by nějaké obsahovala, ale ty by vedly pouze na malé množství stránek, které by odkazovaly (jako izolovaný orientovaný graf) pouze samy na sebe (fenomén *link farm sink*).

sebe však jistě nerozděluje). Předpokládejme však, že půjde o relativně vzácný jev, který významně nenaruší další analýzu textu a  $n$ -gramming.

Dlužno říci, že jde o výpočetně náročnou úlohu: předpokládejme korpus délky  $p$  symbolů, pak pro každou ze čtyř uvažovaných sekvencí mezi větami je tento korpus v lineárním čase proscanován (každá alespoň dvojice sousedních znaků je porovnávána s regulárním výrazem), dostáváme tedy celkem  $3(p - 2)$  porovnání s patternem, pomocí asymptotické časové složitosti tedy  $\Theta(3p) = \Theta(p)$ . Pak při průměrné délce  $q$  symbolů jedné věty je předchozím porovnáváním nalezeno průměrně cca  $\frac{p}{q}$  hranic sousedních vět, které jsou však sortovány jen v rámci sekvence. Je nutné tedy sortovat tři vektory o délce zhruba  $\frac{p}{3q}$  rostoucích čísel dohromady, tím dostáváme výpočetní čas přinejmenším  $\Theta(\frac{p}{q} \log \frac{p}{q})$  (merge sort). Nakonec je v lineárním čase  $\Theta(\frac{p}{q})$  postupně ukrojena vždy „nová“ první věta z postupně se zkracujícího řetězce korpusu, jak je „zpředu“ vždy zkrácen o substring do prvního dalšího indexu označujícího novou hranici vět. Zřejmě tedy rozdělení jednoho korpusu na věty běží v čase lehce náročnějším než lineárním,  $\Theta(\frac{p}{q} \log \frac{p}{q})$ , pro velká  $p$  jde obecně o zdlouhavou proceduru.

### 3.4 Preprocessing textu

V rámci navazujícího preprocessingu textu jsou pomocí procedur skriptu

`preprocessing.R`,

viz též 7, provedeny v získaných větách tyto úpravy:

- odstraněny zkratkové formy obsahující apostrof (např. I'll (I will)); jsou matchovány regulárním výrazem `''[a-zA-Z]+'[a-zA-Z]+'''`;
- odstraněny všechny znaky, které není možné kódovat pomocí UTF-8;
- všechna velká písmena jsou převedena na malá;
- všechny vícenásobné mezery jsou nahrazeny jednoduchou mezerou (`'' ''`);
- odstraněny tzv. leading a trailing spaces (uvozující a koncové mezery); snadno nahlédneme, že ty nejsou předchozím krokem odstraněny;
- nakonec je z vět odstraněna veškerá interpunkce včetně závorek a dalších znaků netypických pro přirozený text (tedy hashtagů, zavináčů, smajlíkových forem apod.).

### 3.5 Tokenizace

Ve fázi, kdy je proveden kvalitní preprocessing a věty obsahují prakticky jen písmena malé anglické abecedy a jednoduché mezery, což vyplývá z operací provedených v rámci preprocessingu, je tokenizace vět na slova relativně snadnou úlohou. V rámci skriptu `tokenization.R` 8 jsou namapovány indexy mezer ve větách a podle nich jsou pak věty rozděleny na slova pomocí uživatelem definované funkce

`splitSentenceIntoWords()` 3. Výsledkem je tedy pro daný korpus tolik uspořádaných  $k$ -tic<sup>3</sup>, kolik obsahuje korpus vět; přirozené  $k \geq 1$  udává pro každou větu počet slov, na kolik je rozdělena.

### 3.6 Odstranění stop slov a dalších

Máme-li věty již rozděleny na slova, je možné některá slova odstranit, je-li k tomu speciální důvod. Na jedné straně je třeba si uvědomit, že odstraněním jednoho nebo více slov z věty, která je nyní reprezentována uspořádanou  $k$ -ticí, kde  $k$  je počet jejích slov, můžeme narušit „sémantickou“ plynulost věty, kterou je nepochybně vhodné udržet pro účely následného  $n$ -grammingu. V poslední době se objevují články, které odstranění stop slov, které bylo ještě donedávna pro některé typy především statistických úloh nad zpracováním přirozených textů rutinní fází, již nedoporučují, nebo ho minimálně zpochybňují, např. [8]. Pravděpodobně to souvisí s rozvojem sofistikovanějších metod pro účely sémantické analýzy (např. sentiment analýzy), kde je vypuštění (stop) slov z vět vnímáno skutečně jako narušení jazykové kontinuity vět a ztrátu jejich „přirozenosti“ či „autentičnosti“, což nakonec zhoršuje výsledky sémantické analýzy. Navíc se běžně uváděné seznamy anglických stop slov významně liší, což je rovněž argument spíše proti rutinnímu odstraňování stop slov z vět reprezentovaných slovy, která jsme získali tokenizací. Programátorsky jde o relativně snadnou úlohu – asymetrický rozdíl dvou uspořádaných seznamů zde vnímaných jako množiny, kdy od uspořádaného seznamu slov věty vnímaného jako množina (v Pythonu typicky `set(list)`) „odečítáme“ množinu stop slov.

Naopak, odstranění vulgární nebo jinak nevhodných slov lze doporučit, ať už jen z pragmatického důvodu určité serióznosti analýzy a na ní postavené webové aplikace. Seznamů nevhodných slov (*swear words lists*, *profanity filters*) je online celá řada, např. zde.

Kromě nevhodných slov byly odstraněny ještě číslice, neboť jejich význam v dané větě je pouze kontextový; jsou-li číselné prvky součástí idiomatické fráze, u které máme dozajista zájem, aby byla součástí  $n$ -gramu, bude takový číselný prvek pravděpodobně vyjádřen číslovkou (slovem).

### 3.7 $n$ -gramming

Vstupem pro  $n$ -gramming je uspořádaná  $k$ -tice slov, která reprezentuje  $k$ -slovnou větu. Korpus je pak složen z nějakého (obvykle velkého) množství takto reprezentovaných vět.  $n$ -gramem rozumíme  $n$ -člennou sekvenci po sobě jdoucích slov některé  $k$ -slovné věty, přičemž předpokládáme, že  $n \leq k$ .

Snadno nahlédneme, že počet  $n$ -gramů, které můžeme získat z  $k$ -slovné věty, je roven  $\epsilon(n, k)$  tak, že

$$\epsilon(n, k) = \begin{cases} k - n + 1, & n \leq k, \\ 0, & n > k. \end{cases}$$

Algoritmus, který z korpusu získá všechny  $n$ -gramy, je relativně jednoduchý. V podstatě každou větu korpusu reprezentovanou uspořádaným seznamem slov pro-

<sup>3</sup> $k$  zde není konstanta, variuje mezi jednotlivými větami.

scanuje „čtecím okénkem“ délky  $n$  (slov) a vždy takové  $n$ -členné slovní spojení uloží do postupně rostoucího seznamu  $n$ -gramů.

Je-li v korpusu  $l$  vět a je-li průměrná délka jedné věty  $k$  slov, kde  $k \geq n$ , pak průměrná časová složitost algoritmu je asymptoticky  $\Theta(l(k - n + 1)) = \Theta(kl - nl)$ , tedy s rostoucím  $n$  klesá.

Nad seznamem  $n$ -gramů lze vytvořit tabulka jejich frekvencí; relativní frekvenci  $n$ -gramu pak můžeme chápat jako bodový odhad pravděpodobnosti, s jakou se daný  $n$ -gram vyskytuje v obecném<sup>4</sup> rozsáhlém textu příslušného přirozeného jazyka. Pro  $n = 1$  získáme *unigramy*, neboli jednotlivá slova. Pro  $n = \{2, 3, 4, 5\}$  hovoříme postupně o *bigramech*, *trigramech*, *quadriramech*, *pentagramech*, respektive.

Seznamy  $n$ -gramů pro  $\forall n = \{2, 3, 4, 5\}$  řeší implementovaná funkce `getNGrams()`, viz též 3 a 10, jejímž vstupem je jednak  $n$ , jednak věta reprezentovaná jako uspořádaná  $k$ -tice slov. Výstupem jsou všechny  $n$ -gramy, které lze ze vstupní věty získat. Skript `n_gramming.R` rovněž počítá tabulky frekvencí pro jednotlivé  $n$ -gramy.

Výstup z procedury `n_gramming.R` pro  $n = 2$ , tedy bigramy, je naznačen v komponentě 14, která slouží zároveň i jako vstup pro aplikaci predikující nejpravděpodobnější  $n$ -té slovo následující za vloženou  $(n - 1)$ -člennou frází. Aplikace pochopitelně používá i  $n$ -grammy vyšších řádů (pro vyšší  $n$ ), viz dále. Pro  $n = 3$  je přehled některých trigramů získaných  $n$ -grammingem nad částmi helsinských korpusů uveden v tabulce 1. Pro ostatní  $n$  získáme obdobné tabulky.

$n$ -gram	první $(n - 1)$ -podřetězec	$n$ -té slovo	frekvence
out of here	out of	here	3
out of his	out of	his	8
out of it	out of	it	9
out of my	out of	my	11
out of our	out of	our	2
out of school	out of	school	3

Tabulka 1: Několik vybraných  $n$ -gramů pro  $n = 3$  (tedy trigramů) první  $(n - 1)$ -podřetězec  $n$ -gramu, jeho  $n$ -té slovo a absolutní frekvence získaná  $n$ -grammingem části helsinských korpusů

<sup>4</sup>Pokud by byl  $n$ -gramming prováděn nad zpracovaným korpusem, který byl silně monotématický či doménově velmi specifický, pak relativní frekvence  $n$ -gramu bodově odhaduje pravděpodobnost výskytu takového  $n$ -členného slovního spojení v textu omezeném daným tématem či doménou.

## 4 Aplikace the\_next\_word\_prediction

Aplikace, podobně jako asynchronní úlohy v rámci zpracování textu korpusu a  $n$ -grammingu, byla vyvinuta v jazyce R prostřednictvím R-kového balíčku **Shiny**.

Smyslem aplikace je nabídnout platformu pro predikci slova, které s největší pravděpodobností následuje zadané anglické  $(n - 1)$ -členné sousloví, kde  $n > 1$ .

### 4.1 Princip aplikace

Zadá-li uživatel do aplikace  $(n - 1)$ -členné anglické sousloví, kde  $n > 1$ , s cílem odhadnout slovo, které by mělo dané  $(n - 1)$ -členné sousloví s největší pravděpodobností následovat, aplikace najde největší takové  $n_0$ , aby  $0 \leq n_0 \leq n - 1$  a zároveň aby v množině všech  $\{2, 3, 4, 5\}$ -gramů existoval alespoň jeden takový  $(n_0 + 1)$ -gram, že posledních  $n_0$  slov uživatelem zadaného sousloví odpovídá postupně prvním, druhému,  $\dots$ ,  $n_0$ -tému slovu tohoto  $(n_0 + 1)$ -gramu. Ze všech takto vyhovujících  $(n_0 + 1)$ -gramů je pak vybrán ten, který má největší relativní četnost (v korpusu, resp. ve frekvenční tabulce  $(n_0 + 1)$ -gramů) a jeho  $(n_0 + 1)$ -té slovo je vráceno uživateli jako to, které nejpravděpodobněji následuje jeho zadanou  $(n - 1)$ -člennou frázi.

Jinými slovy a formálněji, uvažujme, že uživatel zadá do aplikace na vstupu  $(n - 1)$ -člennou frázi ve tvaru  $w_1 w_2 \dots w_{n-1}$ , kde  $w_j$  je  $j$ -té slovo dané fráze postupně pro všechna  $j \in \{1, 2, \dots, n - 1\}$ , s cílem zjistit, jaké slovo  $w_n$  bude nejpravděpodobněji následovat po zadaných  $n - 1$  slovech ve frázi. Buď  $\mathcal{G}_n = \{v_1 v_2 \dots v_n \mid n\}$  množina všech známých  $n$ -gramů, postupně pro všechna  $n \in \{2, 3, 4, 5\}$ . Hledáme  $n_0$  takové, aby

$$n_0 = \arg \max_{i \in \{0, 1, \dots, n-1\}} \{i : (\exists v_1 v_2 \dots v_{i+1} \in \mathcal{G}_{i+1})(\forall j \in \{1, 2, \dots, i\})(v_j = w_{n-i+j-1})\}.$$

Je-li  $n_0 = 0$ , aplikace vrátí nejčastější unigram (slovo *the*). Je-li však  $n_0 > 0$ , pak označme  $\mathcal{G}_{n_0+1}^*$  množinu všech  $(n_0 + 1)$ -gramů, které jsou tvaru  $v_1 v_2 \dots v_{n_0+1} = w_{n-n_0} w_{n-n_0+1} \dots w_{n-1}$ . Potom slovo, které nejpravděpodobněji následuje uživateli frázi  $w_1 w_2 \dots w_{n-1}$ , je  $w_n^*$  takové, že

$$w_n^* = \arg \max_{v_1 v_2 \dots v_n \in \mathcal{G}_{n_0+1}^*} \{\hat{P}(w_n^* = v_{n_0+1} \mid v_1 v_2 \dots v_{n_0+1} \in \mathcal{G}_{n_0+1}^*)\}.$$

Jde o MAP (maximum aposteriori probability) princip, kdy je jako nejpravděpodobnější  $n$ -té slovo následující uživateli frázi vybráno poslední slovo takového  $(n_0 + 1)$ -gramu, který má prvních  $n_0$  slov totožných s posledními  $n_0$  slovy uživateli fráze a zároveň je mezi takovými  $(n_0 + 1)$ -gramy nejčastěji zastoupen. Je-li více takových nejčastěji zastoupených  $(n_0 + 1)$ -gramů, je jako  $n$ -té slovo následující uživateli frázi vybráno to, které je první v abecedě (anebo je vybráno poslední slovo náhodného z těchto  $n$ -gramů). Někdy se též popsanému přístupu říká back-off model, neboť algoritmus v podstatě nejdříve vyhledává shodu se zadanou frází mezi  $n$ -gramy; pokud ji nenajde, pokračuje mezi  $(n - 1)$ -gramy, opět pokud ji nenalezne, přistoupí k  $(n - 2)$  gramům, a tak dále, až nakonec prohledává 1-gramy

(unigramy). Když nenalezne shodu mezi žádným unigramem a posledním slovem uživatelské fráze, vrací obvykle nejčastější unigram (v angličtině člen *the*).

Kromě MAP principu existují i další, sofistikovanější přístupy, jak vybrat nejpravděpodobnější  $n$ -té slovo doplňující uživatelskou  $(n - 1)$ -člennou frázi. Někdy se případy chybějících frází ošetřují obyčejným Laplaceovým „add-one“ vyhlazováním [9], kdy je každému  $n$ -gramu (i chybějícímu, poprvé se objevivšímu až při uživatelském zadání) uměle zvýšena absolutní četnost ve frekvenčních tabulkách  $n$ -gramů o 1. Obecně se tento přístup dnes již nedoporučuje. Známé je také např. Kneser-Neyovo vyhlazování [10], které pro danou  $(n - 1)$ -člennou frázi, jež zadává uživatel, prohledává i po nalezení dobré shody mezi posledními  $n_0$  slovy uživatelské fráze a některým  $(n_0 + 1)$ -gramem ještě všechny  $(n_0 + 1 - j)$ -gramy pro  $\forall j \in \{1, 2, \dots, n_0\}$  a zkoumá četnosti dalších shodujících se  $(n_0 + 1 - j)$ -gramů s posledními  $(n_0 - j)$  slovy uživatelské fráze – pokud by některá četnost některé  $(n_0 + 1 - j)$ -gramu byla výrazně vyšší než byla četnost nejčastějšího  $(n_0 + 1)$ -gramu, jako uživatelem hledané slovo je (poněkud paradoxně) vráceno poslední slovo tohoto (i podstatně kratšího)  $(n_0 + 1 - j)$ -gramu. Kneser-Neyovo vyhlazování totiž řeší i kontextové zařazení některých frází.

## 4.2 Komponenty aplikace

Aplikace se skládá z následujících částí:

- `ui.R`
- `server.R`
- slovníky  $n$ -gramů, tedy `my_i_word_vocabulary` pro  $i \in \{1, 2, 3\}$
- složka `www`:
  - `style.css`

Uživatelský layout aplikace je relativně jednoduchý a intuitivní, viz obrázek 2. Online lze aplikaci používat snadno na adrese

[http://shiny.statest.cz:3838/the\\_next\\_word\\_prediction/](http://shiny.statest.cz:3838/the_next_word_prediction/).

Na první záložce *Results* lze vložit vstupní frázi o délce 1-3 slov. Po kliknutí na tlačítko *Predict the next word!* se v pravém panelu zobrazí slovo, které by mělo pravděpodobně jazykově následovat po vložené  $(n - 1)$ -členné frázi.

Na druhé záložce *About* je zmíněno pár slov o účelu a principu aplikace.

Popíšme nyní detailněji jednotlivé části aplikace.

`ui.R` Viz též 11. Název vyplývá ze zkratky *user interface*. Definuje veškeré grafické a ovládací prvky aplikace, které lze napsat pomocí jazyka HTML (*HyperText Markup Language*). I přesto je však psána pomocí příkazů jazyka R; balíček *Shiny* totiž definuje placeholderové funkce (aliasy), které mají na vstupu kód srozumitelný prostředí R, ale na výstupu vrací čisté HTML.



**The Next Word Prediction**

Prediction of the  $n$ -th word based on given previous  $n-1$  words and  $n$ -gramming • Implemented by [Lubomír Štěpánek](#)

Input your phrase here:

how are

Predict the next word!

Results About

You have entered the following phrase:

how are

The most likely next word is:

you

$P(\text{the predicted } n\text{-th word} \mid \text{the given } (n - 1)\text{-words phrase} \wedge n\text{-grams got by given corpora}) = 0.125$

Obrázek 2: Uživatelský interface domovské stránky aplikace

Některé grafické prvky však byly napsány přímo pomocí syntaxe HTML – balíček **Shiny** této syntaxi rozumí a v případě, že má uživatel znalost i značkovacího jazyka HTML, je pak práce snazší přímo pomocí HTML, nikoliv R-kových aliasů či wrapperů.

<code>server.R</code>	Viz též 12. Jádro celé aplikace, obsahuje workhorse funkce, především implementaci backoff modelu výběru tak, jak je popsán ve statí 4.1.
slovníky $n$ -gramů	Viz též tabulka 1 a komponenta 14. Slovníky tvoří základ pro back-off vyhledávání nejpravděpodobnějšího $n$ -tého slova následujícího $(n - 1)$ -člennou frází.
<code>style.css</code>	Viz též 13. Kaskádové styly, které definují rozměry, barvy a další parametry některých prvků aplikace, především headeru.

## 5 Závěr a další možné směřování práce

Byla vytvořena „pipeline“ zpracování textu přirozeného analytického jazyka, konkrétně angličtiny, a základní model statistické reprezentace korpusu daného jazyka, tedy  $n$ -gramming pro  $n \in \{2, 3, 4, 5\}$ . Ve všech výše uvedených případech jde o asynchronní úlohy, které je možné „předpočítat“ dopředu.

Rovněž byla implementována webová aplikace, která staví na extrahovaných  $n$ -gramech a predikuje ( $n$ -té) slovo, které s největší pravděpodobností následuje ( $n - 1$ )-člennou frází, kterou zadává uživatel aplikace. Zde se již jedná o „on-the-fly“ úlohu, která musí proběhnout takřka v reálném čase (uživatel v podstatě nesmí čekat).

Princip, který aplikace využívá, je obdobou různých našeptávačů v messaging aplikacích a v prohlížečích, kdy různé formy pop-up našeptávačů nabízejí uživateli okamžité možnosti toho, jaké slovo se s velkou pravděpodobností nabízí jako vhodné (kontextové) pokračování slovní fráze, kterou do textového pole již zadal.

Do budoucna se nabízí eventuální vytvoření  $n$ -gramů pro vyšší  $n$ . Zároveň je možné implementovat Kneser-Neyovo vyhlazování, které balancuje rozdíly mezi  $n$ -gramy a jejich kontextem pro malá a velká  $n$ . V aplikaci je možné nechat vypsát nikoliv jen první a nejpravděpodobnější slovo následující po zadané ( $n - 1$ )-členné frází, ale vypsát těchto pravděpodobných slov hned několik a řadit je sestupně dle pravděpodobnosti.

## 6 Literatura

- [1] Manfred Görlach. *Introduction to Early Modern English*. Cambridge University Press, 1991. ISBN: 0521325293.
- [2] Christopher Manning a Hinrich Schuetze. *Foundations of Statistical Natural Language Processing*. The MIT Press, 1999. ISBN: 0-262-13360-1.
- [3] R Core Team. *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing. Vienna, Austria, 2016. URL: <https://www.R-project.org/>.
- [4] M. Davies. „The Corpus of Contemporary American English as the first reliable monitor corpus of English“. In: *Literary and Linguistic Computing* 25.4 (říj. 2010), s. 447–464. DOI: 10.1093/llc/fqq018. URL: <https://doi.org/10.1093/llc/fqq018>.
- [5] Jeremy H. Clear. „The Digital Word“. In: ed. George P. Landow a Paul Delany. Cambridge, MA, USA: MIT Press, 1993. Kap. The British National Corpus, s. 163–187. ISBN: 0-262-12176-x. URL: <http://dl.acm.org/citation.cfm?id=166403.166418>.
- [6] Merja Kytö. *Manual to the diachronic part of the Helsinki corpus of English texts : coding conventions and lists of source texts*. Helsinki: Dept. of English, University of Helsinki, 1996. ISBN: 951-45-7470-2.
- [7] N. Chomsky. „Three models for the description of language“. In: *IEEE Transactions on Information Theory* 2.3 (zář. 1956), s. 113–124. DOI: 10.1109/tit.1956.1056813. URL: <https://doi.org/10.1109/tit.1956.1056813>.
- [8] Hassan Saif, Miriam Fernández, Yulan He et al. „On stopwords, filtering and data sparsity for sentiment analysis of twitter“. In: *The 9th International Conference on Language Resources and Evaluation*. 2014.
- [9] Stanley F. Chen a Joshua Goodman. „An Empirical Study of Smoothing Techniques for Language Modeling“. In: *Proceedings of the 34th Annual Meeting on Association for Computational Linguistics*. ACL '96. Santa Cruz, California: Association for Computational Linguistics, 1996, s. 310–318. DOI: 10.3115/981863.981904. URL: <http://dx.doi.org/10.3115/981863.981904>.
- [10] Dan Jurafsky. *Speech and language processing : an introduction to natural language processing, computational linguistics, and speech recognition*. Upper Saddle River, N.J: Pearson Prentice Hall, 2009. ISBN: 978-0131873216.

## 7 Rejstřík

$(n - 1)$ -členná fráze, 5, 14–18

$n$ -gram, 5–7, 11–16, 18

bigram, 14, 17

pentagram, 14

quadrigram, 14

tabulka četností, 7, 14–16, 18

trigram, 14, 17

unigram, 14–16

$n$ -té slovo, 5, 8, 14, 15, 18

predikce, 15, 16

back-off, 15, 17

úloha

asynchronní, 9, 15, 18

synchronní, 18

číslovka, 10

člen *the*, 15, 16

R, 5, 9, 10, 15, 16

Shiny, 15–17

algoritmus, 7, 10, 11, 13

greedy, 10

maximum a posteriori, 15, 16

analýza, 6

sentimentu, 13

angličtina, 6, 7, 10, 18

aplikace

`the_next_word_prediction`, 8,

13, 15, 16

CSS, 10, 16, 17

graf

orientovaný, 11

hashtag, 12

HTML, 10, 16, 17

entita, 10

klauzule, 10

tag, 10

interpunkce, 7

javascript, 10

jazyk

analytický, 6

Chomského hierarchie, 10

flekční, 7

metajazyk, 10

kódování

UTF-8, 12

korpus, 6, 7, 9, 11, 13–15, 18

British National Corpus, 9

Corpus of Contemporary

American English, 9

helsinský, 7, 9, 14

lemmatizace, 6, 7

link

farm sink, 11

interní, 10

outlink, 11

webový, 10

mezera

leading, 12

trailing, 12

morfematika, 6, 7

pravděpodobnost, 14, 15, 18

odhad, 14, 15

processing, 9, 15

natural language processing, 6, 9

preprocessing, 7, 9, 12

regulární výraz, 10–12

slova, 13–15

nevhodná, 7, 13

stop slova, 7, 13

vulgarismy, 7, 13

sorting, 12

tokenizace, 7, 12, 13

URL, 10

výpočetní čas

lineární, 12

nelineární, 12, 14

vyhlazování

Kneser-Neyovo, 16, 18

Laplaceovo (add-one), 16

webscraping, 7, 9–11

Wikipedie, 7, 9–11

tag, 10

## 8 Apendix

Zde jsou uvedeny kódy všech funkcí a procedur v jazyce RindexR, které řeší uvedené fáze zpracování přirozeného jazyka a současně i kód všech komponent aplikace `the_next_word_prediction`.

### 8.1 Procedura `__main__.R`

```

1  #####
2  #####
3  #####
4
5  ## nastavuji pracovní složku -----
6
7  while(!grepl("asynchronni_ulohy_seminarni_prace$", getwd())){
8
9      setwd(choose.dir())
10
11 }
12
13 mother_working_directory <- getwd()
14
15
16 ## -----
17
18 #####
19
20 ## spouštím sekvenci skriptů -----
21
22 for(my_script in c(
23
24     "initialization",      ## spouštím inicializaci procedur
25     "helper_functions",    ## pomocné funkce
26     "webscraping",         ## webscraping některých stránek anglické
27     "corpora_loading",     ## loaduji již existující korpusy, stop
28     "text_splitting",      ## a nevhodná slova
29     "preprocessing",       ## rozděluji texty korpusů na věty (i
30     "tokenization",        ## vedlejší)
31     "postprocessing",      ## očišťuji věty o interpunkci a bílá místa
32     "n_gramming",          ## rozděluji věty na slova
33     "n_gramming",          ## odstraňuji stop slova, nevhodná slova,
34                             ## číslovky, příliš krátká slova
35                             ## vytvářím n-gramy pro nízká n
36 )){
37
38     setwd(mother_working_directory)
39
40     source(
41         file = paste(my_script, ".R", sep = ""),
42         echo = TRUE,
43         encoding = "UTF-8"

```

```

44 |         )
45 |
46 |     }
47 |
48 |
49 |     ## -----
50 |
51 |     #####
52 |     #####
53 |     #####

```

Kód 1: Procedura \_\_main\_\_.R

## 8.2 Procedura initialization.R

```

1 | #####
2 | #####
3 | #####
4 |
5 | \## instaluji a inicializuji balíčky -----
6 |
7 | for(package in c(
8 |     "openxlsx",
9 |     "xtable",
10 |     "tm"
11 | )){
12 |
13 |     if(!(package %in% rownames(installed.packages()))){
14 |
15 |         install.packages(
16 |             package,
17 |             dependencies = TRUE,
18 |             repos = "http://cran.us.r-project.org"
19 |         )
20 |
21 |     }
22 |
23 |     library(package, character.only = TRUE)
24 |
25 | }
26 |
27 |
28 | ## -----
29 |
30 | #####
31 |
32 | ## nastavuji handling se zipováním v R -----
33 |
34 | Sys.setenv(R_ZIPCMD = "C:/Rtools/bin/zip")
35 |
36 |
37 | ## -----
38 |
39 | #####
40 |

```

```

41  ## zakládám podsložku "výstupy" -----
42
43  for(my_directory in c("výstupy")){
44
45      if(!file.exists(my_directory)){
46
47          dir.create(file.path(
48
49              mother_working_directory, my_directory
50
51          ))
52      }
53  }
54
55 }
56
57
58 ## -----
59
60 #####
61 #####
62 #####

```

Kód 2: Procedura initialization.R

### 8.3 Funkce helper\_functions.R

```

1  #####
2  #####
3  #####
4
5  ## definuji pomocné funkce -----
6
7  #####
8
9  #### funkce na dělení textu do vět -----
10
11 splitTextIntoSentences <- function(
12
13     my_text
14
15 ){
16
17     # '''
18     # Textový řetězec "my_text" o jedné či více větách rozdělí
19     # s určitou mírou spolehlivosti na samostatné věty.
20     # '''
21
22     split_indices <- NULL
23     my_sentences <- NULL
24
25     for(stop_mark in c(
26         "\\\\.\\s*[A-Z]+",      ## tečka, mezera (>= 0), velké písmeno
27         "\\?\\s*[A-Z]+",    ## otazník, mezera (>= 0), velké písmeno
28         "\\!\\s*[A-Z]+",    ## vykřičník, mezera (>= 0), velké písmeno

```

```

29         "\\:\\s*"          ## dvojtečka, mezera (>= 0)
30     )){
31
32         split_indices <- c(
33
34             split_indices,
35             gregexpr(
36                 pattern = stop_mark,
37                 text = my_text
38             )[[1]] + 1
39
40         )
41
42     }
43
44     ordered_split_indices <- split_indices[split_indices > 0][
45         order(split_indices[split_indices > 0])
46     ]
47
48     if(length(ordered_split_indices) > 0){
49
50         ordered_split_indices <- c(
51             1,
52             ordered_split_indices,
53             nchar(my_text)
54         )
55
56         for(i in 1:(length(ordered_split_indices) - 1)){
57
58             my_sentences <- c(
59
60                 my_sentences,
61                 substr(
62                     my_text,
63                     ordered_split_indices[i],
64                     ordered_split_indices[i + 1]
65                 )
66
67             )
68
69         }
70
71     }else{
72
73         my_sentences <- my_text
74
75     }
76
77     for(j in 1:length(my_sentences)){
78
79         while(substr(my_sentences[j], 1, 1) == " "){
80
81             my_sentences[j] <- substr(
82                 my_sentences[j], 2, nchar(my_sentences[j])
83             )
84

```



```

85     }
86
87     while(substr(
88         my_sentences[j],
89         nchar(my_sentences[j]),
90         nchar(my_sentences[j])
91     ) == " "){
92
93         my_sentences[j] <- substr(
94             my_sentences[j],
95             1,
96             (nchar(my_sentences[j]) - 1)
97         )
98     }
99 }
100
101 }
102
103 return(my_sentences)
104 }
105 }
106
107 ##### -----
108 #####
109 #####
110 #####
111 #####
112 ##### funkce na rozdělení věty na slova -----
113
114 splitSentenceIntoWords <- function(
115     my_sentence
116 ){
117
118     # ""
119     # Rozděluje větu "my_sentence" na jednotlivá slova.
120     # ""
121
122     return(
123         strsplit(
124             x = my_sentence,
125             split = " "
126         )[[1]]
127     )
128 }
129
130 }
131
132 ##### -----
133 #####
134 #####
135 #####
136 #####
137 #####
138 ##### funkce pro tvorbu n-gramů -----
139
140 getNGrams <- function(

```

```

141
142     my_splitted_sentences ,
143     n = 2
144
145 ){
146
147     # '''
148     # Nad větou rozdělenou na slova "my_splitted_sentences" vytvoří
149     # všechny n-gramy pro zadané "n".
150     # '''
151
152     output <- NULL
153
154     if(length(my_splitted_sentences) >= n){
155
156         for(i in 1:(length(my_splitted_sentences) - n + 1)){
157
158             output <- c(
159
160                 output ,
161                 paste(
162                     my_splitted_sentences[i:(i + n - 1)],
163                     collapse = " "
164                 )
165
166             )
167
168         }
169
170     }
171
172     return(output)
173
174 }
175
176
177 ##### -----
178
179 #####
180
181 ##### funkce pro webscraping jedné stránky Wikipedie (typu článek)
182 ##### a pro následnou úpravu formátu do podoby volného textu -----
183
184 webscrapeMyWikipediaPage <- function(
185
186     page_url
187
188 ){
189
190     # '''
191     # Funkce stáhne statický HTML obsah jedné stránky z (anglické)
192     # Wikipedie, která je pod odkazem "page_url". Poté extrahuje jen
193     # odstavcové statě ohraničené HTML tagy <p>...</p>.
194     # Z nich pak odstraní veškeré další HTML tagy, HTML entity či
195     # wikipedické tagy.
196     # Nakonec vrátí textový řetězec odpovídající jen přirozenému

```

```

197 # textu v odstavcích dané stránky Wikipedie.
198 # Kromě toho ještě z textu stránky extrahuje interní webové odkazy
199 # na další stránky Wikipedie, které je poté možné scrapovat.
200 # ""
201
202
203 ## stahuji statický HTML obsah -----
204
205 my_html <- readLines(
206   con = page_url,
207   encoding = "UTF-8"
208 )
209
210
211 \## extrahuji jen odstavcové statě ohraničené HTML tagy <p>...</p>
212 -----
213
214 my_raw_paragraphs <- my_html[
215   grepl("<p>", my_html) & grepl("</p>", my_html)
216 ]
217
218 ## očišťuji text paragrafů o HTML tagy, HTML entity a wikipedické tagy
219 --
220
221 my_paragraphs <- gsub("<.*?>", "", my_raw_paragraphs)
222 my_paragraphs <- gsub("&.*?;", "", my_paragraphs)
223 my_paragraphs <- gsub("\\[.*?\\]", "", my_paragraphs)
224 my_paragraphs <- gsub("\\t", "", my_paragraphs) ## zbavuji se
225   tabulátorů
226
227 ## vytvářím jeden dlouhý řetězec (odstavec) -----
228
229 my_text_output <- paste(my_paragraphs, collapse = " ")
230
231 ## extrahuji z textu všechny webové interní linky na další stránky
232 ## Wikipedie -----
233
234 my_links <- paste(
235   "http://en.wikipedia.org",
236   gsub(
237     '\\\"',
238     "",
239     gsub(
240       '(.*)(href=)("\\\"/wiki/.*?\\\")(.*)',
241       '\\3',
242       my_raw_paragraphs[
243         grepl("href=", my_raw_paragraphs)
244       ]
245     )
246   ),
247   sep = " "
248 )
249

```

```

250
251     ## odstraňuji nesmyslné outlinky - ty, co obsahují mezeru, eventuálně
252     ## ty, co odkazují na celý portál nebo kategorii (jsou o obvykle jen
253     ## seznamy hesel, tedy nevhodné pro sestavené korpusu) -----
254
255     my_links <- my_links[!grepl(" ", my_links)]
256     my_links <- my_links[!grepl("Portal:", my_links)]
257     my_links <- my_links[!grepl("Category:", my_links)]
258
259
260     ## vracím výstup -----
261
262     return(
263         list(
264             "text_stranky" = my_text_output,
265             "outlinky_stranky" = my_links
266         )
267     )
268
269
270 }
271
272
273 ##### -----
274
275 #####
276 #####
277 #####

```

Kód 3: Funkce helper\_functions.R

## 8.4 Procedura webscraping.R

```

1  #####
2  #####
3  #####
4
5  ## vytvářím malý korpus z nějakého počtu stránek anglicky psané
6  ## Wikipedie -----
7
8  #####
9
10 ##### vytvářím zatím prázdný vektor linků na wikipedické anglické stránky
11 ##### a vektor korpus pro jejich texty,
12 ##### oba budu postupně populovat -----
13
14 my_links <- NULL
15 my_wikipedia_corpus <- NULL
16 my_initial_page_url <- "https://en.wikipedia.org/wiki/World_War
    _II"
17 number_of_scraped_pages <- 0                ## počet aktuálně
    scrapovaných
18
19                                     ## wikipedických stránek
20 how_many_pages_i_would_like_to_scrape <- 900
    ## počet wikipedických

```

```
21                                     stránek,  
22                                     ## který chci scrapovat  
23  
24 my_links <- c(my_links, my_initial_page_url)  
25  
26 while(number_of_scraped_pages < how_many_pages_i_would_like_to_  
27       scrape){  
28     my_webscrape <- webscrapeMyWikipediaPage(  
29       my_links[number_of_scraped_pages + 1]  
30     )  
31  
32     my_wikipedia_corpus <- c(  
33  
34       my_wikipedia_corpus,  
35       my_webscrape[["text_stranky"]]  
36     )  
37  
38     my_links <- unique(c(  
39  
40       my_links,  
41       my_webscrape[["outlinky_stranky"]]  
42     ))  
43  
44     number_of_scraped_pages <- number_of_scraped_pages + 1  
45  
46     flush.console()  
47     print(  
48       paste(  
49         "Právě vyscrapováno ",  
50         format(  
51           round(  
52             number_of_scraped_pages /  
53             how_many_pages_i_would_like_to_scrape *  
54             100,  
55             digits = 2  
56           ),  
57           nsmall = 2  
58         ),  
59       " % z požadovaného počtu stránek.",  
60       sep = "  
61     )  
62   )  
63 )  
64  
65 }  
66  
67 my_wikipedia_corpus <- iconv(  
68   my_wikipedia_corpus,  
69   to = "UTF-8"  
70 )  
71  
72  
73
```

```

74  ### -----
75
76  #####
77
78  ### ukládám wikipedický korpus -----
79
80  setwd(paste(mother_working_directory, "vstupy", sep = "/"))
81
82  writeLines(
83      text = my_wikipedia_corpus,
84      con = "en_US.wikipedia.txt"
85  )
86
87
88  setwd(mother_working_directory)
89
90
91  ### -----
92
93  #####
94  #####
95  #####

```

Kód 4: Procedura webscraping.R

## 8.5 Procedura corpora\_loading.R

```

1  #####
2  #####
3  #####
4
5  ## loaduji korpusy a množiny stop slov a nevhodných slov -----
6
7  #####
8
9  ### loaduji korpusy -----
10
11  setwd(paste(mother_working_directory, "vstupy", sep = "/"))
12
13  for(my_corpus_source in c(
14
15      "blogs",
16      "news",
17      "twitter"
18
19  )){
20
21      # '''
22      # nahrávám všechny korpusy a ukládám je pod originálními jmény
23      # '''
24
25      assign(
26          paste(my_corpus_source, "corpus", sep = "_"),
27          readLines(
28              con = paste("en_US.", my_corpus_source, ".txt", sep

```

```

29         = ""),
30         encoding = "UTF-8"
31     )
32 )
33 flush.console()
34 print(
35     paste(
36         "Byl nahrán korpus '",
37         my_corpus_source,
38         "'.",
39         sep = ""
40     )
41 )
42 }
43
44 setwd(mother_working_directory)
45
46 ##### -----
47 #####
48 ##### loaduji množiny stop slov a nevhodných slov -----
49
50 setwd(paste(mother_working_directory, "vstup", sep = "/"))
51
52 for(my_word_group_name in c(
53     "stop_words",
54     "swear_words"
55 )){
56     assign(
57         my_word_group_name,
58         readLines(
59             con = paste(my_word_group_name, ".txt", sep = ""),
60             encoding = "UTF-8"
61         )
62     )
63 }
64
65 empty_words <- c("") \## definuji množinu "prázdných" slov
66
67 setwd(mother_working_directory)
68
69 ##### -----
70 #####
71 #####

```

82 || #####

## Kód 5: Procedura corpora\_loading.R

### 8.6 Procedura text\_splitting.R

```

1  #####
2  #####
3  #####
4
5  ## text splitting -----
6
7  #####
8
9  #### zadávám, kolik prvních "my_proportion" * 100 procent každého korpusu
10 #### bude zpracováno pro účely sestavení vektoru s větami -----
11
12 my_proportion <- 0.005      ## zřejmě je 0 <= my_proportion <= 1
13
14
15 #### -----
16
17 #####
18
19 #### inicializují vektor "my_sentences", do kterého se budou ukládat
20 #### věty všech korpusů -----
21
22 my_sentences <- NULL
23
24
25 #### nyní dělím všechny korpusy na jednotlivé věty (i vedlejší) -----
26
27 for(my_corpus_source in c(
28
29     "blogs",
30     "news",
31     "twitter"
32
33 )){
34
35     # '''
36     # postupně upravuji a očišťuji daný korpus
37     # '''
38
39     ## loaduji korpus -----
40
41     my_corpus <- get(paste(my_corpus_source, "corpus", sep = "_
42                        "))
43
44     ## dělím korpus na věty -----
45
46     for(i in 1:ceiling(length(my_corpus) * my_proportion)){
47
48         my_sentences <- c(

```



```

49         my_sentences ,
50         splitTextIntoSentences(my_corpus[i])
51     )
52
53     flush.console()
54     print(
55         paste(
56             "Korpus '",
57             my_corpus_source,
58             "' (jeho první ",
59             format(
60                 round(
61                     my_proportion * 100,
62                     digits = 1
63                 ),
64                 nsmall = 1
65             ),
66             " %): ",
67             "proces hotov z ",
68             format(
69                 round(
70                     i / ceiling(
71                         length(my_corpus) * my_proportion
72                     ) * 100,
73                     digits = 2
74                 ),
75                 nsmall = 2
76             ),
77             " %.",
78             sep = " "
79         )
80     )
81 }
82
83 }
84
85
86
87 ##### -----
88
89 #####
90 #####
91 #####

```

Kód 6: Procedura text\_splitting.R

## 8.7 Procedura preprocessing.R

```

1 #####
2 #####
3 #####
4
5 ## provádím preprocessing -----
6
7 #####

```

```

8
9  #### odstraňuji zkrácená slova s apostrofem -----
10
11 my_sentences <- gsub("[a-zA-Z]+'[a-zA-Z]+", "", my_sentences)
12
13
14 #### převádím věty korpusu na kódování UTF-8 (některé znaky i přes
15 #### kódování zdrojových souborů v UTF-8 mohou zůstat "nečitelné") -----
16
17 my_sentences <- iconv(my_sentences, to = "UTF-8")
18
19
20 #### převádím všechna písmena všech vět na malá -----
21
22 my_sentences <- tolower(my_sentences)
23
24
25 #### odstraňuji vícenásobné mezery ve větách -----
26
27 my_sentences <- gsub("\\s+", " ", my_sentences)
28
29
30 #### odstraňuji leading a trailing spaces (uvozující a koncové mezery) -----
31
32 my_sentences <- gsub("^\\s+", "", my_sentences)
33 my_sentences <- gsub("\\s+$", "", my_sentences)
34
35
36 #### odstraňuji z vět veškerou interpunkci -----
37
38 my_sentences <- gsub("[[:punct:]]", "", my_sentences)
39
40
41 #### -----
42
43 #####
44 #####
45 #####

```

Kód 7: Procedura preprocessing.R

## 8.8 Procedura tokenization.R

```

1  #####
2  #####
3  #####
4
5  ## provádím tokenizaci -----
6
7  #####
8
9  #### dělím všechny věty na slova -----
10
11 my splitted_sentences <- lapply(
12     my_sentences,

```

```

13     splitSentenceIntoWords
14 )
15
16
17 ##### -----
18
19 #####
20 #####
21 #####

```

Kód 8: Procedura tokenization.R

## 8.9 Procedura postprocessing.R

```

1 #####
2 #####
3 #####
4
5 ## provádím postprocessing -----
6
7 #####
8
9 ##### odstraňuji číslouky -----
10
11 mySplittedSentences <- lapply(
12   mySplittedSentences,
13   function(x) gsub("[0-9]+", "", x)
14 )
15
16
17 ##### nahrazuji některé nečitelné znaky -----
18
19 mySplittedSentences <- lapply(
20   mySplittedSentences,
21   function(x) gsub("[^a-z ']", "", x)
22 )
23
24
25 ##### odstraňuji stop slova, nevhodná slova a "prázdná" slova ("") -----
26
27 for(myWordGroupName in c(
28   "stop_words",
29   "swear_words",
30   "empty_words"
31 )){
32
33 }{
34
35   mySplittedSentences <- lapply(
36     mySplittedSentences,
37     function(x) setdiff(x, get(myWordGroupName))
38   )
39
40 }
41

```

```

42
43 #### odstraňuji číslouky -----
44
45 my_splitted_sentences <- lapply(
46   my_splitted_sentences,
47   function(x) gsub("[0-9]+", "", x)
48 )
49
50
51 #### odstraňuji slova kratší než tři znaky -----
52
53 my_splitted_sentences <- lapply(
54   my_splitted_sentences,
55   function(x) x[nchar(x) > 2]
56 )
57
58
59 #### odstraňuji "prázdná" slova -----
60
61 my_splitted_sentences <- lapply(
62   my_splitted_sentences,
63   function(x) setdiff(x, "")
64 )
65
66
67 #### ponechávám jen neprázdné věty (některé prázdné mohly nově vzniknout
68 #### kvůli očištění o předchozí skupiny slov) -----
69
70 my_splitted_sentences <- my_splitted_sentences[lapply(
71   my_splitted_sentences,
72   length
73 ) > 0]
74
75
76 #### -----
77
78 #####
79 #####
80 #####

```

Kód 9: Procedura postprocessing.R

## 8.10 Procedura n\_gramming.R

```

1 #####
2 #####
3 #####
4
5 ## vytvářím n-gramy pro n z 2, 3, 4, 5 -----
6
7 #####
8
9 #### vytvářím n-gramy -----
10
11 for(n in 2:5){

```

```

12
13     n_grams <- unlist(lapply(
14         my_splitted_sentences,
15         function(x) getNGrams(x, n)
16     ))
17
18     assign(
19         paste(
20             setNames(
21                 c("bi", "tri", "quadri", "penta", "hexa", "
22                     hepta"),
23                 c(2, 3, 4, 5, 6, 7)
24             )[as.character(n)],
25             "grams",
26             sep = "_"
27         ),
28         n_grams
29     )
30
31     flush.console()
32     print(
33         paste(
34             n,
35             "-gramy jsou zkompletovány.",
36             sep = " "
37         )
38     )
39 }
40
41
42 #### -----
43
44 #####
45
46 ### vytvářím tabulky n-gramů řazených abecedně -----
47
48 for(n in 2:5){
49
50     n_grams <- get(
51         paste(
52             setNames(
53                 c("bi", "tri", "quadri", "penta", "hexa", "
54                     hepta"),
55                 c(2, 3, 4, 5, 6, 7)
56             )[as.character(n)],
57             "grams",
58             sep = "_"
59         )
60     )
61
62     my_table <- table(n_grams)
63
64     assign(
65         paste(
66             setNames(

```

```

66         c("bi", "tri", "quadri", "penta", "hexa", "
67           hepta"),
68         c(2, 3, 4, 5, 6, 7)
69       )[as.character(n)],
70       "grams_table",
71       sep = "_"
72     ),
73     data.frame(
74       "n_gram" = names(my_table[order(names(my_table))]),
75       "subphrase" = gsub(
76         "(.*) (.*)",
77         "\\1",
78         names(my_table[order(names(my_table))])
79       ),
80       "last_word" = gsub(
81         "(.*) (.*)",
82         "\\2",
83         names(my_table[order(names(my_table))])
84       ),
85       "frequency" = as.numeric(
86         unname(my_table[order(names(my_table))])
87       ),
88       stringsAsFactors = FALSE
89     )
90   )
91   flush.console()
92   print(
93     paste(
94       "Tabulka pro ",
95       n,
96       "-gramy je hotová.",
97       sep = ""
98     )
99   )
100 }
101
102
103
104 #### -----
105
106 #####
107
108 #### ukládám tabulky n-gramů -----
109
110 setwd(paste(mother_working_directory, "vystupy", sep = "/"))
111
112 for(n in 2:5){
113
114   n_grams_table <- get(
115     paste(
116       setNames(
117         c("bi", "tri", "quadri", "penta", "hexa", "
118           hepta"),
119         c(2, 3, 4, 5, 6, 7)
120       )[as.character(n)],

```

```

120         "grams_table",
121         sep = "-"
122     )
123 )
124
125 write.table(
126     x = n_grams_table,
127     file = paste(
128         setNames(
129             c("bi", "tri", "quadri", "penta", "hexa", "
130                 hepta"),
131             c(2, 3, 4, 5, 6, 7)
132         )[as.character(n)],
133         "_grams_table.txt",
134         sep = ""
135     ),
136     row.names = FALSE,
137     col.names = TRUE,
138     sep = ";"
139 )
140 flush.console()
141 print(
142     paste(
143         "Tabulka pro ",
144         n,
145         "-gramy je uložena.",
146         sep = ""
147     )
148 )
149 }
150
151 setwd(mother_working_directory)
152
153
154
155 #### -----
156
157 #####
158 #####
159 #####

```

Kód 10: Procedura n\_gramming.R

## 8.11 Komponenta ui.R

```

1 #####
2 #####
3 #####
4
5 library(shiny)
6
7
8 #####
9 #####

```

```

10 #####
11
12 shinyUI(fluidPage(
13
14   #titlePanel("The Next Word Prediction"),
15
16   ## zavádím graficky hezky vypadající header -----
17
18   tagList(
19
20     tags$head(
21
22       tags$link(rel = "stylesheet",
23                 type = "text/css",
24                 href = "style.css"),
25
26       tags$script(type = "text/javascript",
27                   src = "busy.js")
28
29     )
30
31   ),
32
33   div(id = "header",
34       div(id = "title", "The Next Word Prediction"),
35       div(id = "subsubtitle",
36
37         HTML("Prediction of the <i>n</i>-th word based on
38             given
39             previous <i>n</i>-1 words and "),
40
41         tags$a(
42           href = "http://en.wikipedia.org/wiki/N-gram",
43           HTML("<i>n</i>-gramming"),
44           target = "_blank"
45         ),
46
47         HTML("&bull;"),
48
49         "Implemented by",
50
51         tags$a(
52           href = "http://www.fbmi.cvut.cz/user/stepalu2",
53           "Lubomír Štěpánek",
54           target = "_blank"
55         )
56
57     ),
58
59   sidebarLayout(
60
61     sidebarPanel(
62
63       #####
64

```



```

65     ## první záložka
66
67     conditionalPanel(
68
69         condition = "input.conditionedPanels == 1",
70
71         textInput(inputId = "my_inputted_phrase",
72                 label = "Input your phrase here:"
73                 ),
74
75         tags$hr(),
76
77         actionButton(inputId = "my_button",
78                     label = "Predict the next word!"),
79
80         tags$hr()
81     ),
82
83
84     #####
85     ## druhá záložka
86
87     conditionalPanel(
88
89         condition = "input.conditionedPanels == 2",
90
91         strong(paste("Here you can find some pieces of
92                     information",
93                     "about the application.",
94                     sep=" "
95                     ),
96
97         ),
98
99         tags$hr()
100
101     #####
102
103
104     )
105
106
107
108     ),
109
110     #####
111     #####
112     #####
113
114     mainPanel(
115
116         tabsetPanel(
117
118             #####
119

```

```

120     ## první záložka
121
122     tabPanel(
123         title = "Results",
124
125         br(),
126
127         p(strong("You have entered the following phrase:")),
128
129         textOutput("inputted_phrase"),
130
131         br(),
132         tags$hr(),
133
134         p(strong("The most likely next word is:")),
135
136         textOutput("predicted_word"),
137
138         br(),
139         tags$hr(),
140
141         p(strong(
142             em("P"),
143             "(",
144             em("the predicted n-th"),
145             em("word"),
146             "|",
147             em("the given (n - 1)-words phrase"),
148             HTML("&#x2227;"),
149             em("n-grams got by given corpora"),
150             ") = "
151         )),
152
153         textOutput("my_conditional_probability"),
154
155         value = 1
156     ),
157
158
159     #####
160
161     ## druhá záložka
162
163     tabPanel(
164         title = "About",
165
166         br(),
167
168         h4("Introduction"),
169
170         p("- purpose of this application is to offer a tool for
171           prediction of", em("n"), "-th",
172           "word most likely following the previous (" , em("n -
173             1"), ") words words inputted by user",
174           "Last, but not least piece of purpose is to pass 4
175             IZ470 Web Data Mining subject practised",

```

```

173         "at University of Economics , Prague"),
174
175     br(),
176
177     p(h4("Analysis, preprocessing before algorithm
178         deployment")),
179
180     p(paste("- data originate from well-known HC corpora ('
181         Helsinki Corpora'),",
182         "blog, twitter and news corpus datasets were
183         used", sep = " ")),
184
185     p(paste("- before analysis, all vulgarisms, swear and
186         stop words were",
187         "removed from the datasets", sep = " ")),
188     p("-", em("n"), "-grams for", em("n"), HTML("&\#x2208;"
189         ), "{2, 3, 4}",
190         "were worked out"),
191     p(paste("- 2-grams, 3-grams and 4-grams contain at all
192         45 354,",
193         "23 989, and 5 570 phrases, respectively", sep
194         = " ")),
195     p("- a naive frequentist probability model based on ("
196         , em("n"), "+ 1)-gram",
197         "for predicting of (" , em("n"), "+ 1)-th word
198         according to",
199         "previous", em("n"), "inputted words was conducted"),
200
201     br(),
202
203     p(h4("Pseudocode of the algorithm")),
204
205     code("i-gram <- database of pairs [i-words phrase, its
206         (i + 1)-th likely following word]"),
207     br(),
208     code("n <- min(3, number of words of the phrase
209         inputted by user)"),
210     br(),
211     code("while n > 0 do"),
212     br(),
213     code("...phrase <- last n words of the phrase inputted
214         by user"),
215     br(),
216     code("...{M} <- {0}"),
217     br(),
218     code("...for all pairs in n-gram do"),
219     br(),
220     code(".....if phrase in pair then"),
221     br(),
222     code(".....{M} <- pair"),
223     br(),
224     code("...if {M} <> {0} then"),
225     br(),
226     code(".....return (n + 1)-th following word of the
227         most frequent pair"),
228     br(),

```

```

216     code("...else"),
217     br(),
218     code(".....n <- n - 1"),
219
220
221     br(),
222
223     br(),
224
225     p(h4("In progress")),
226
227     p("- Kneser-Ney smoothing with back-off model will be",
228       "eventually applied to predict the smooth probability
229         of",
230       "a next word"),
231
232     br(),
233
234     br(),
235
236     p(h4("Usage of the application, conclusions")),
237
238     p("- the application s available at http://shiny.
239       statest.cz:3838/the_next_word_prediction/"),
240     p("- a user can intuitively input her phrase which
241       contains at least one word"),
242     p("- after pushing the 'Predict the next word!' button
243       the most likely following word is predicted
244       and the conditional probability of correctness of
245       the prediction, given the inputted phrase
246       and given 'static' n-grams based on provided
247       datasets, is returned"),
248
249     br(),
250
251     p(h4("About the author")),
252
253     p("Lubomir Stepanek, M. D."),
254     p("- Department of Biomedical Informatics"),
255     p("- Faculty of Biomedical Engineering"),
256     p("- Czech Technical University in Prague"),
257     HTML("<a href='mailto:lubomir.stepanek[AT]fbmi.cvut.cz
258       '>
259       lubomir.stepanek[AT]fbmi[DOT]cvut[DOT]cz</a>"),
260     br(),
261
262     value = 2
263   ),
264
265   #####
266
267   id = "conditionedPanels"
268
269   #####

```

```

265 |
266 |     )
267 |
268 |   )
269 |
270 |   )
271 |
272 | ))
273 |
274 |
275 |
276 |
277 | #####
278 | #####
279 | #####

```

Kód 11: Komponenta ui.R

## 8.12 Komponenta server.R

```

1 | #####
2 | #####
3 | #####
4 |
5 | library(shiny)
6 |
7 |
8 | #####
9 | #####
10 | #####
11 |
12 | shinyServer(
13 |
14 |   function(input, output){
15 |
16 |     #####
17 |
18 |     ## loading of my vocabularies with 1-, 2- and 3-grams
19 |
20 |     my_1_word_vocabulary <- reactive({
21 |
22 |       data <- read.csv("my_1_word_vocabulary.txt",
23 |                       header = TRUE,
24 |                       sep = " "
25 |                       )
26 |
27 |       for(i in 1:3){data[, i] <- as.character(data[, i])}
28 |       data[, 3] <- as.numeric(data[, 3])
29 |
30 |       return(data)
31 |
32 |     })
33 |
34 |
35 |     my_2_word_vocabulary <- reactive({

```

```

36
37     data <- read.csv("my_2_word_vocabulary.txt",
38                     header = TRUE,
39                     sep = " ")
40 )
41
42     for(i in 1:3){data[, i] <- as.character(data[, i])}
43     data[, 3] <- as.numeric(data[, 3])
44
45     return(data)
46
47 })
48
49
50 my_3_word_vocabulary <- reactive({
51
52     data <- read.csv("my_3_word_vocabulary.txt",
53                     header = TRUE,
54                     sep = " ")
55
56     )
57
58     for(i in 1:3){data[, i] <- as.character(data[, i])}
59     data[, 3] <- as.numeric(data[, 3])
60
61     return(data)
62
63 })
64
65
66 #####
67
68 ## outputting of inserted phrase
69
70 output$inputted_phrase <- renderText({
71
72     if(is.null(input$my_inputted_phrase)){
73
74         return(" ")
75
76     }else{
77
78         return(input$my_inputted_phrase)
79
80     }
81
82 })
83
84
85
86 #####
87
88 ## I am getting a reactive event according to inputted phrase and
89 ## clearing the phrase as well
90
91 my_clear_phrase <- eventReactive(input$my_button, {

```

```

92
93     inputted_phrase <- tolower(input$my_inputted_phrase)
94
95     while(grepl(" ", inputted_phrase)){
96
97         inputted_phrase <- gsub(" ", " ", inputted_phrase)
98
99     }
100
101     if(substr(inputted_phrase, 1, 1) == " "){
102         inputted_phrase <- substr(inputted_phrase,
103                                   2,
104                                   nchar(inputted_phrase)
105                                   )
106     }
107
108     if(substr(inputted_phrase,
109               nchar(inputted_phrase),
110               nchar(inputted_phrase)) == " "){
111         inputted_phrase <- substr(inputted_phrase,
112                                   1,
113                                   nchar(inputted_phrase)
114                                   )
115     }
116
117     return(inputted_phrase)
118 }
119
120
121
122 #####
123
124 ## helper function for getting of last k words of the phrase
125
126 getEndOfMyPhrase <- function(my_phrase, k){
127
128     n_of_words <- length(strsplit(my_phrase, split = " ")[
129                               [[1]]
130     )
131
132     if(k >= n_of_words){
133
134         return(my_phrase)
135     }else{
136
137         return(
138             paste(
139                 strsplit(my_phrase, split = " ")[[1]][
140                     (n_of_words - k + 1) : n_of_words
141                 ],
142                 collapse = " "
143             )
144         )
145     }
146

```

```

147     }
148
149
150     #####
151
152     ## helper function for prediction of the next word
153
154     getMyLikelyNextWord <- function(my_phrase){
155
156         my_phrase <- getEndOfMyPhrase(my_phrase, 3)
157
158         if(length(strsplit(my_phrase, split = " ")[[1]]) == 3){
159             if(my_phrase %in% my_3_word_vocabulary()$phrase){
160
161                 which_to_choose <- which(
162                     my_3_word_vocabulary()$phrase == my_phrase
163                 )
164                 my_index <- which.max(
165                     my_3_word_vocabulary()$frequency[which_to_choose]
166                 )[1]
167                 return(
168                     my_3_word_vocabulary()$next_word[which_to_choose][
169                         my_index]
170                 )
171             }else{
172
173                 my_phrase <- getEndOfMyPhrase(my_phrase, 2)
174
175             }
176         }
177
178         if(length(strsplit(my_phrase, split = " ")[[1]]) == 2){
179             if(my_phrase %in% my_2_word_vocabulary()$phrase){
180
181                 which_to_choose <- which(
182                     my_2_word_vocabulary()$phrase == my_phrase
183                 )
184                 my_index <- which.max(
185                     my_2_word_vocabulary()$frequency[which_to_choose]
186                 )[1]
187                 return(
188                     my_2_word_vocabulary()$next_word[which_to_choose][
189                         my_index]
190                 )
191             }else{
192
193                 my_phrase <- getEndOfMyPhrase(my_phrase, 1)
194
195             }
196         }
197
198         if(length(strsplit(my_phrase, split = " ")[[1]]) == 1){
199             if(my_phrase %in% my_1_word_vocabulary()$phrase){
200

```



```

201         which_to_choose <- which(
202             my_1_word_vocabulary()$phrase == my_phrase
203         )
204         my_index <- which.max(
205             my_1_word_vocabulary()$frequency[which_to_choose]
206         )[1]
207         return(
208             my_1_word_vocabulary()$next_word[which_to_choose
209                 ][my_index]
210         )
211     }else{
212
213         return("the")
214     }
215 }
216
217 }
218
219 }
220
221
222 #####
223
224 ## helper function for my conditional probability
225
226 getMyConditionalProbability <- function(my_phrase){
227
228     my_phrase <- getEndOfMyPhrase(my_phrase, 3)
229
230     if(length(strsplit(my_phrase, split = " ")[[1]]) == 3){
231         if(my_phrase %in% my_3_word_vocabulary()$phrase){
232
233             which_to_choose <- which(
234                 my_3_word_vocabulary()$phrase == my_phrase
235             )
236             my_numerator <- my_3_word_vocabulary()$frequency[
237                 which.max(
238                     my_3_word_vocabulary()$frequency[which_to_choose]
239                 )][1]
240             return(min(1,
241                 my_numerator / sum(
242                     my_3_word_vocabulary()$frequency[which_to_choose]
243                 ))
244             )
245         }else{
246
247             my_phrase <- getEndOfMyPhrase(my_phrase, 2)
248
249         }
250     }
251
252     if(length(strsplit(my_phrase, split = " ")[[1]]) == 2){
253         if(my_phrase %in% my_2_word_vocabulary()$phrase){
254

```

```

255     which_to_choose <- which(
256       my_2_word_vocabulary()$phrase == my_phrase
257     )
258     my_numerator <- my_2_word_vocabulary()$frequency[
259       which.max(
260         my_2_word_vocabulary()$frequency[which_to_choose]
261       ) [1]]
262     return(min(1,
263       my_numerator / sum(
264         my_2_word_vocabulary()$frequency[which_to_choose]
265       ))
266     )
267   }else{
268
269     my_phrase <- getEndOfMyPhrase(my_phrase, 1)
270
271   }
272 }
273
274 if(length(strsplit(my_phrase, split = " ")[[1]]) == 1){
275   if(my_phrase %in% my_1_word_vocabulary()$phrase){
276
277     which_to_choose <- which(
278       my_1_word_vocabulary()$phrase == my_phrase
279     )
280     my_numerator <- my_1_word_vocabulary()$frequency[
281       which.max(
282         my_1_word_vocabulary()$frequency[which_to_choose]
283       ) [1]]
284     return(min(1,
285       my_numerator / sum(
286         my_1_word_vocabulary()$frequency[which_to_choose]
287       ))
288     )
289   }else{
290
291     return("> 0")
292
293   }
294 }
295 }
296
297 }
298
299
300 #####
301
302 output$predicted_word <- renderText({
303
304   getMyLikelyNextWord(my_clear_phrase())
305
306 })
307
308

```

```

309 #####
310
311 output$my_conditional_probability <- renderText({
312
313     getMyConditionalProbability(my_clear_phrase())
314
315 })
316
317 #####
318
319
320
321 }
322
323 )
324
325 #####
326 #####
327 #####
328 #####

```

Kód 12: Komponenta server.R

### 8.13 Komponenta www/style.css

```

1  div.busy {
2      position: absolute;
3      top: 11.9%;
4      left: 88.0%;
5      margin-top: -100px;
6      margin-left: -50px;
7      display: none;
8      background: rgba(230, 230, 230, .8);
9      text-align: center;
10     padding-top: 20px;
11     padding-left: 30px;
12     padding-bottom: 40px;
13     padding-right: 30px;
14     border-radius: 5px;
15 }
16
17 #header {
18     text-align: center;
19     color: #fdfdfd;
20     text-shadow: 0 0 1px #000;
21     padding: 30px 0 45px;
22     border-bottom: 1px solid #ddd;
23     margin: 0 -30px 20px;
24     /* pozadí staženo z http://lea.verou.me/css3patterns/ */
25     background-color: #000080;
26     background-image:
27         radial-gradient(white, rgba(255,255,255,.2) 2px,
28             transparent 40px),
29         radial-gradient(white, rgba(255,255,255,.15) 1px,
30             transparent 30px),

```

```

29     radial-gradient(white, rgba(255,255,255,.1) 2px,
30         transparent 40px),
31     radial-gradient(rgba(255,255,255,.4), rgba(255,255,255,.1)
32         2px, transparent 30px);
33 background-size: 550px 550px, 350px 350px, 250px 250px, 150px
34     150px;
35 background-position: 0 0, 40px 60px, 130px 270px, 70px 100px;
36 }
37
38 #title {
39     font-size: 5em;
40     text-shadow: 0 0 5px #000;
41     margin-bottom: 5px
42 }
43
44 #subsubtitle {
45     font-size: 1.3em;
46 }
47
48 #subsubtitle a {
49     color: #fdddfd;
50     text-decoration: underline;
51 }

```

Kód 13: Komponenta `www/style.css`

## 8.14 Komponenta `my_1_word_vocabulary.txt`

```

1  "id"      "phrase" "next_word" "frequency"
2  "22381"  "may"    "well"    3
3  "22382"  "may"    "when"    4
4  "22383"  "may"    "with"    4
5  "22384"  "may"    "you"     3
6  "22385"  "may"    "your"    2
7  "22386"  "maybe"  "a"       3
8  "22387"  "maybe"  "but"     3
9  "22388"  "maybe"  "even"    2

```

Kód 14: Komponenta `my_1_word_vocabulary.txt` (její část)