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

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 navazujcích 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.

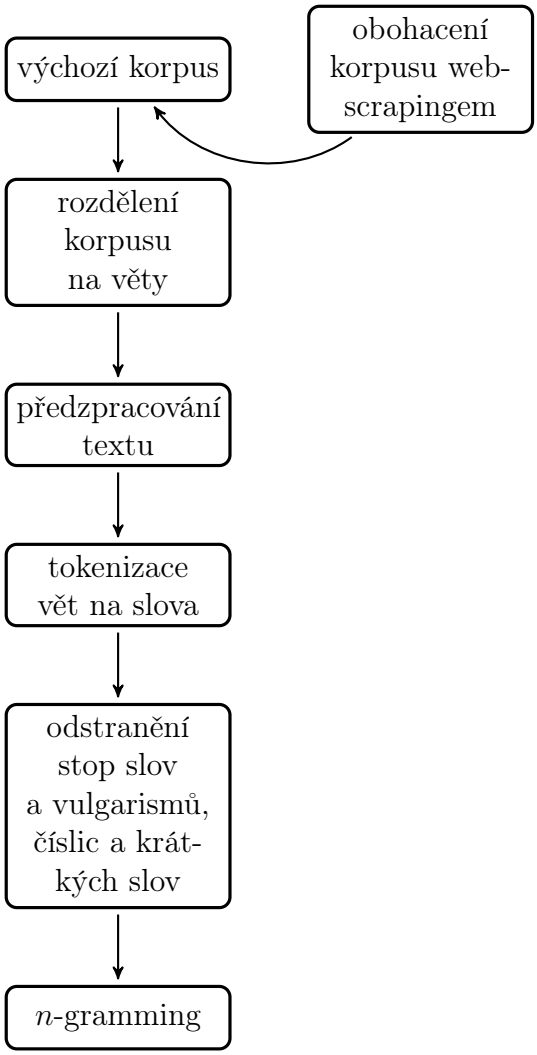
Práce splní svůj smysl, pokud čtenáře zaujme byť i jedna jediná myšlena, která je v následujícím textu uvedena.

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 dobře zvládnutelné, a to především díky minimálnímu zatížení anglických textů morfematikou 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-grammingu[1] 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* [2]; 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 předzpracování 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 druhů) 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[3]. 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 předpovědí takového slova, které by v textu daného přirozeného jazyka následovalo s největší pravděpodobností po zadaných $n-1$ slovech tvořících jazykové

[1] 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 předzpracování 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 [4].

Veškerý komentovaný zdrojový kód je rovněž uveden v příloze publikace (kapitola 8). 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 intermediánních či vyšších vestavěných funkcí. Všechny procedury naráz spustí přiložený kód 1 a 2 v kapitole 8; pro všechny skripty viz kapitolu 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 [5], který nabízí 520 milionů slov, British National Corpus [6] obsahující 100 milionů slov nebo tzv. helsinské korpusy anglických textů postavené na cca 1,5 milionu slov [2].

Obvykle není možné korpusy volně stáhnou. 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®, který nabízí John Hopkins Bloomberg School of Public Health [7]. Malá část tohoto původního korpusu je ke stažení např. na webovém zdroji [8]. 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 přiloženým kódem 5, jak je uvedeno v kapitole 8.

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. Současně 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 statického HTML (HyperText Markup Language). Formátování pomocí kaskádových stylů (CSS, Cascading Style Sheets), respektive přidanou 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 přiloženém kódu 3 v kapitole 8.

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 `<h`*i*`>` a `</h`*i*`>`), tabulky (vymezené tagy `<table>` a `</table>`) apod. volný text neobsahují, resp. si tím nemůžeme být jistí, proto v rámci vysoké specificity 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 [9], 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 ztotožní se pouze s nejkratším podřetězcem daným regulárním výrazem. Obdobně regulární výraz `<&.*?;>` testuje „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 číslice.

Kromě sestavení volného textu funkce z HTML obsahu extrahuje i všechny interní webové odkazy v rámci anglické Wikipedie; opět pomocí regulárního výrazu testujícího vzor `href=''/wiki/...''` typický pro wikipedický interní odkaz.

Zmíněnou funkci následuje procedura, viz přiložený kód 4 v kapitole 8, která na vstupu použije jeden zvolený článek, z něj vyextrahuje všechny webové odkazy vedoucí na jiné stránky typu článek anglické Wikipedie a současně scrapuje text tohoto článku.

Odkazy byly uloženy do globální proměnné `my_links`. V druhé iteraci je vyscrapována stránka (a do stejné globální proměnné `my_links` jsou uloženy její odkazy) dostupná z prvního odkazu v globální proměnné `my_links`. V třetí iteraci je proces opakován se stránkou odkazovanou druhým odkazem v globální proměnné s odkazy `my_links`. Proces je opakován do chvíle, než je získáno určité, uživatelem definované množství vyscrapovaných wikipedických stránek[2].

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 odkazy), 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, viz přiložený kód 6 v kapitole 8. Algoritmus využívá regulárních výrazů, a sice je založen na pozorování, že na hranici dvou vět se vyskytuje prakticky vždy jeden z následujících vzorů znaků:

- sekvence tečka, mezera (žádná, jedna, nebo i více), velké písmeno – tato sekvence je testována regulárním výrazem `''\\.\\s*[A-Z]+''`,
- sekvence otazník, mezera (žádná, jedna, nebo i více), velké písmeno – tato sekvence je testována regulárním výrazem `''\\?\\s*[A-Z]+''`,
- sekvence vykřičník, mezera (žádná, jedna, nebo i více), velké písmeno – tato sekvence je testována 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 testována regulárním výrazem `''\\:\\s*''`.

[2]Hypoteticky by mohla být procedura nechtěně ukončena i dříve – a to např. tehdy, kdyby volba výchozí stránky nebyla vhodná, protože by tato neobsahovala žádné odkazy, 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*).

Funkce relativně spolehlivě rozdělí libovolně dlouhý korpus o libovolném počtu vět na jednotlivé věty. Omezením funkce mohou být např. zkratky titulů, které budou spárovány s první uvedenou sekvencí, ale ve skutečnosti nejsou na hranici dvou vět (v případě titulu `Ph.D.` je tento maskou pro vzor `''\\.\\s*[A-Z]+''`, dvě věty od 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 porovnána s regulárním výrazem), dostáváme tedy celkem $3(p-2)$ porovnání se vzorem, 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 Předpracování textu

V rámci navazujícího předzpracování textu jsou pomocí procedur skriptu

`preprocessing.R`,

provedeny v získaných větách tyto úpravy (viz též přiložený kód 7 v kapitole 8):

- odstraněny zkratkové formy obsahující apostrof (např. `I'll` (I will)); jsou testovány 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 provedeno kvalitní předzpracování a věty obsahují prakticky jen písmena malé anglické abecedy a jednoduché mezery, což vyplývá z operací provedených v rámci předzpracování, je tokenizace vět na slova relativně snadnou úlohou. V rámci skriptu `tokenization.R`, viz přiložený kód 8 v kapitole 8, jsou namapovány indexy mezer ve větách a podle nich jsou pak věty rozděleny na slova pomocí autorem definované funkce `splitSentenceIntoWords()`, viz přiložený kód 3. Výsledkem je tedy pro daný korpus tolik uspořádaných k-tic[3], 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, jež bylo ještě donedávna pro některé typy především statistických úloh nad zpracováním přirozených textů rutinní fází, již nedoporučují, nebo ho minimálně zpochybňují, např. [10]. 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 vulgárních 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. na webovém zdroji [11].

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 jsoucích slov některé k-slovné věty, přičemž předpokládáme, že $n \leq k$.

[3] k zde není konstanta, obecně se mezi jednotlivými větami liší.

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 proscanuje „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 tabulku jejich frekvencí; relativní frekvenci n-gramu pak můžeme chápat jako bodový odhad pravděpodobnosti, s jakou se daný n-gram vyskytuje v obecném[4] 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 kódy 3 a 10 v kapitole 8, 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`, uvedený v části 10 kapitoly 8 rovněž počítá tabulky frekvencí pro jednotlivé n-gramy.

Výstup z procedury `n_gramming.R` pro $n = 2$, tedy bigramy, je naznačen v komponentě 14 kapitoly 8, 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

[4]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 prvnímu, druhému, ..., 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

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

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 uživatelovu frázi $w_1 w_2 \dots w_{n-1}$, je w^*_n takové, že

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

Jde o MAP (maximum aposteriori probability) princip, kdy je jako nejpravděpodobnější n-té slovo následující uživatelovu $(n-1)$-člennou frázi vybráno poslední slovo takového (n_0+1)-gramu, který má prvních n_0 slov totožných s posledními n_0 slovy uživatelovy 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 uživatelovu 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 uživatelovy fráze, vrací obvykle nejčastější unigram (v angličtině člen *the*).

Kromě MAP principu existují i další, sofistikovanější přístupy, jak vybrat nejpravděpodobnější n-té slovo doplňující uživatelovu $(n-1)$-člennou frázi. Někdy se případy chybějících frází ošetřují obyčejným Laplaceovým „add-one“ vyhlazováním [12], kdy je každému n-gramu (i chybějícímu, poprvé se objevivšímu až při uživatelově 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 [13], které pro danou $(n-1)$-člennou frázi, jež zadává uživatel, prohledává i po nalezení dobré shody mezi posledními n_0 slovy uživatelovy fráze a některým (n_0+1)-gramem ještě všechny (n_0+1-j)-gramy pro $\forall j \in \{1, 2, \ldots, n_0\}$ a zkoumá četnosti dalších shodujících se (n_0+1-j)-gramů s posledními (n_0-j) slovy uživatelovy fráze – pokud by některá četnost některého (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

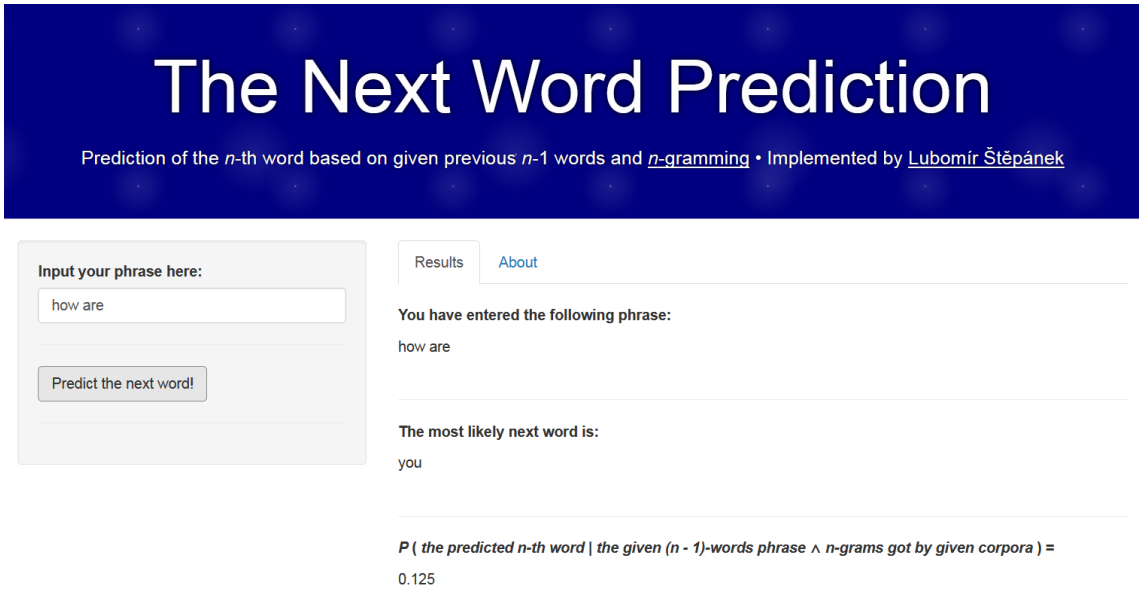
http://shiny.statest.cz:3838/the_next_word_prediction/.

Na první záložce *Results* lze vložit vstupní frázi o délce 1-3 slov (či delší). 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.

Popišme nyní detailněji jednotlivé části aplikace.

`ui.R` Viz též přiložený kód 11 v kapitole 8. Název souboru je zkratkou *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



Obrázek 2: Uživatelský interface domovské stránky aplikace

na výstupu vrací čisté HTML. 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.

`server.R` — Viz též přiložený kód 12 v kapitole 8. Jádro celé aplikace, obsahuje workhorse funkce, především implementaci backoff modelu výběru tak, jak je popsán ve stati 4.1.

slovníky n-gramů — Viz též tabulku 1 a komponentu 14 v kapitole 8. Slovníky tvoří základ pro back-off vyhledávání nejpravděpodobnějšího n-tého slova následujícího $(n-1)$-člennou frázi.

`style.css` — Viz též přiložený kód 13 v kapitole 8. 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 frázi, 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 vypsat nikoliv jen první a nejpravděpodobnější slovo následující po zadané $(n-1)$-členné frázi, ale vypsat 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] 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.

[3] Christopher Manning a Hinrich Schuetze. *Foundations of Statistical Natural Language Processing*. The MIT Press, 1999. ISBN: 0-262-13360-1.

[4] 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/.

[5] 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.

[6] 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.

[7] Coursera a John Hopkins Bloomberg School of Public Health. *Data Science Specialization*. 2017. URL: https://www.coursera.org/specializations/jhu-data-science/ (cit. 18. 06. 2017).

[8] John Hopkins Bloomberg School of Public Health. *Part of Helsinki Corpora*. 2017. URL: https://d396qusza40orc.cloudfront.net/dsscapstone/dataset/Coursera-SwiftKey.zip (cit. 18. 06. 2017).

[9] 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.

[10] 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.

[11] BannedWordList.com. *Banned Word List*. URL: http://www.bannedwordlist.com/ (cit. 18. 06. 2017).

[12] 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.

[13] 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

8 Příloha

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

8.1 Procedura `__main__.R`

```
################################################################################
################################################################################
################################################################################

## nastavuji pracovní složku ------------------------

while(!grepl("asynchronni_ulohy_seminarni_prace$", getwd())){

    setwd(choose.dir())

}

mother_working_directory <- getwd()


## -------------------------------------

################################################################################

## spouštím sekvenci skriptů ------------------------

for(my_script in c(

    "initialization",       ## spouštím inicializaci procedur
    "helper_functions",     ## pomocné funkce
    "webscraping",          ## webscraping některých stránek anglické
        Wiki
    "corpora_loading",      ## loaduji již existující korpusy, stop
        slova
                            ## a nevhodná slova
    "text_splitting",       ## rozděluji texty korpusů na věty (i
        vedlejší)
    "preprocessing",        ## očišťuji věty o interpunkci a bílá místa
    "tokenization",         ## rozděluji věty na slova
    "postprocessing",       ## odstaňuji stop slova, nevhodná slova,
                            ## číslovky, příliš krátká slova
    "n_gramming"            ## vytvářím n-gramy pro nízká n

)){

    setwd(mother_working_directory)

    source(
        file = paste(my_script, ".R", sep = ""),
        echo = TRUE,
        encoding = "UTF-8"
```

```
    )

}


## ------------------------------------

################################################################################
################################################################################
################################################################################
```

Kód 1: Procedura `__main__.R`

8.2 Procedura `initialization.R`

```
################################################################################
################################################################################
################################################################################

\## instaluji a inicializuji balíčky ----------------------

for(package in c(
                  "openxlsx",
                  "xtable",
                  "tm"
                  )){

    if(!(package %in% rownames(installed.packages()))){

        install.packages(
            package,
            dependencies = TRUE,
            repos = "http://cran.us.r-project.org"
        )

    }

    library(package, character.only = TRUE)

}


## ------------------------------------

################################################################################

## nastavuji handling se zipováním v R --------------------

Sys.setenv(R_ZIPCMD = "C:/Rtools/bin/zip")


## ------------------------------------

################################################################################

```

```
## zakládám podsložku "výstupy" -----------------------

for(my_directory in c("vystupy")){

    if(!file.exists(my_directory)){

        dir.create(file.path(

            mother_working_directory, my_directory

        ))

    }

}


## -------------------------------------

################################################################################
################################################################################
################################################################################
```

Kód 2: Procedura `initialization.R`

8.3 Funkce `helper_functions.R`

```
################################################################################
################################################################################
################################################################################

## definuji pomocné funkce -------------------------

################################################################################

#### funkce na dělení textu do vět ----------------------

splitTextIntoSentences <- function(

    my_text

){

    # '''
    # Textový řetězec "my_text" o jedné či více větách rozdělí
    # s určitou mírou spolehlivosti na samostatné věty.
    # '''

    split_indices <- NULL
    my_sentences <- NULL

    for(stop_mark in c(
        "\\.\\s*[A-Z]+",     ## tečka, mezera (>= 0), velké písmeno
        "\\?\\s*[A-Z]+",     ## otazník, mezera (>= 0), velké písmeno
        "\\!\\s*[A-Z]+",     ## vykřičník, mezera (>= 0), velké písmeno
```

```
        "\\:\\s*"             ## dvojtečka, mezera (>= 0)
    )){

        split_indices <- c(

            split_indices,
            gregexpr(
                pattern = stop_mark,
                text = my_text
            )[[1]] + 1

        )

    }

    ordered_split_indices <- split_indices[split_indices > 0][
        order(split_indices[split_indices > 0])
    ]

    if(length(ordered_split_indices) > 0){

        ordered_split_indices <- c(
            1,
            ordered_split_indices,
            nchar(my_text)
        )

        for(i in 1:(length(ordered_split_indices) - 1)){

            my_sentences <- c(

                my_sentences,
                substr(
                    my_text,
                    ordered_split_indices[i],
                    ordered_split_indices[i + 1]
                )

            )

        }

    }else{

        my_sentences <- my_text

    }

    for(j in 1:length(my_sentences)){

        while(substr(my_sentences[j], 1, 1) == " "){

            my_sentences[j] <- substr(
                my_sentences[j], 2, nchar(my_sentences[j])
            )

```

```
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
116     my_sentence
117
118 ){
119
120     # '''
121     # Rozděluje větu "my_sentence" na jednotlivá slova.
122     # '''
123
124     return(
125         strsplit(
126             x = my_sentence,
127             split = " "
128         )[[1]]
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 vrací textový řetezec 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     my_raw_paragraphs <- my_html[
214         grepl("<p>", my_html) & grepl("</p>", my_html)
215     ]
216
217
218     ## očišťuji text paragrafů o HTML tagy, HTML entity a wikipedické tagy
            --
219
220     my_paragraphs <- gsub("<.*?>", "", my_raw_paragraphs)
221     my_paragraphs <- gsub("&.*?;", "", my_paragraphs)
222     my_paragraphs <- gsub("\\[.*?\\]", "", my_paragraphs)
223     my_paragraphs <- gsub("\t", "", my_paragraphs)  ## zbavuji se
            tabulátorů
224
225
226     ## vytvářím jeden dlouhý řetězec (odstavec) ----------------
227
228     my_text_output <- paste(my_paragraphs, collapse = " ")
229
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                                        ## wikipedických stránek
19 how_many_pages_i_would_like_to_scrape <- 900
20                                        ## počet wikipedických
```

```
                                                    stránek,
                                                ## který chci scrapovat


my_links <- c(my_links, my_initial_page_url)

while(number_of_scraped_pages < how_many_pages_i_would_like_to_
    scrape){

    my_webscrape <- webscrapeMyWikipediaPage(
            my_links[number_of_scraped_pages + 1]
    )

    my_wikipedia_corpus <- c(

        my_wikipedia_corpus,
        my_webscrape[["text_stranky"]]

    )

    my_links <- unique(c(

        my_links,
        my_webscrape[["outlinky_stranky"]]

    ))

    number_of_scraped_pages <- number_of_scraped_pages + 1

    flush.console()
    print(
        paste(
            "Právě vyscrapováno ",
            format(
                round(
                    number_of_scraped_pages /
                    how_many_pages_i_would_like_to_scrape *
                        100,
                    digits = 2
                ),
                nsmall = 2
            ),
            " % z požadovaného počtu stránek.",
            sep = ""
        )
    )


}

my_wikipedia_corpus <- iconv(
    my_wikipedia_corpus,
    to = "UTF-8"
)
```

```
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              encoding = "UTF-8"
30          )
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
45  setwd(mother_working_directory)
46
47
48  #### -----------------------------------
49
50  ###############################################################################
51
52  #### loaduji množiny stop slov a nevhodných slov ---------------
53
54  setwd(paste(mother_working_directory, "vstupy", sep = "/"))
55
56  for(my_word_group_name in c(
57
58      "stop_words",
59      "swear_words"
60
61  )){
62
63      assign(
64          my_word_group_name,
65          readLines(
66              con = paste(my_word_group_name, ".txt", sep = ""),
67              encoding = "UTF-8"
68          )
69      )
70
71  }
72
73  empty_words <- c("")     \## definuji množinu "prázných" slov
74
75  setwd(mother_working_directory)
76
77
78  #### -----------------------------------
79
80  ###############################################################################
81  ###############################################################################
```

```
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 #### inicializuji 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(
```

```
            my_sentences,
            splitTextIntoSentences(my_corpus[i])
        )

        flush.console()
        print(
            paste(
                "Korpus '",
                my_corpus_source,
                "' (jeho první ",
                format(
                    round(
                        my_proportion * 100,
                        digits = 1
                    ),
                    nsmall = 1
                ),
                " %): ",
                "proces hotov z ",
                format(
                    round(
                        i / ceiling(
                            length(my_corpus) * my_proportion
                        ) * 100,
                        digits = 2
                    ),
                    nsmall = 2
                ),
                " %.",
                sep = ""
            )
        )

    }

}


#### -----------------------------------

################################################################################
################################################################################
################################################################################
```

Kód 6: Procedura `text_splitting.R`

8.7 Procedura `preprocessing.R`

```
################################################################################
################################################################################
################################################################################

## provádím preprocessing -------------------------

################################################################################
```

```
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 #### odtraňuji vícenásobné mezery ve větách ------------------
26
27 my_sentences <- gsub("\\s+", " ", my_sentences)
28
29
30 #### odtraňuji 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 #### odtraňuji 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,
```

```
    splitSentenceIntoWords
)


#### ----------------------------------

################################################################################
################################################################################
################################################################################
```

Kód 8: Procedura `tokenization.R`

8.9 Procedura `postprocessing.R`

```
################################################################################
################################################################################
################################################################################

## provádím postprocessing -------------------------

################################################################################

#### odstaňuji číslovky ----------------------------

my_splitted_sentences <- lapply(
    my_splitted_sentences,
    function(x) gsub("[0-9]+", "", x)
)


#### nahrazuji některé nečitelné znaky --------------------

my_splitted_sentences <- lapply(
    my_splitted_sentences,
    function(x) gsub("[^a-z ']", "", x)
)


#### odstraňuji stop slova, nevhodná slova a "prázdná" slova ("") -------

for(my_word_group_name in c(

    "stop_words",
    "swear_words",
    "empty_words"

)){

    my_splitted_sentences <- lapply(
        my_splitted_sentences,
        function(x) setdiff(x, get(my_word_group_name))
    )

}

```

```
42
43 #### odstraňuji číslovky ---------------------------
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", "
                        hepta"),
22                  c(2, 3, 4, 5, 6, 7)
23              )[as.character(n)],
24              "grams",
25              sep = "_"
26          ),
27          n_grams
28      )
29
30      flush.console()
31      print(
32          paste(
33              n,
34              "-gramy jsou zkompletovány.",
35              sep = ""
36          )
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", "
                        hepta"),
54                  c(2, 3, 4, 5, 6, 7)
55              )[as.character(n)],
56              "grams",
57              sep = "_"
58          )
59      )
60
61      my_table <- table(n_grams)
62
63      assign(
64          paste(
65              setNames(
```

```
                c("bi", "tri", "quadri", "penta", "hexa", "
                    hepta"),
                c(2, 3, 4, 5, 6, 7)
            )[as.character(n)],
            "grams_table",
            sep = "_"
        ),
        data.frame(
            "n_gram" = names(my_table[order(names(my_table))]),
            "subphrase" = gsub(
                "(.*) (.*)",
                "\\1",
                names(my_table[order(names(my_table))])
            ),
            "last_word" = gsub(
                "(.*) (.*)",
                "\\2",
                names(my_table[order(names(my_table))])
            ),
            "frequency" = as.numeric(
                unname(my_table[order(names(my_table))])
            ),
            stringsAsFactors = FALSE
        )
    )

    flush.console()
    print(
        paste(
            "Tabulka pro ",
            n,
            "-gramy je hotová.",
            sep = ""
        )
    )

}


#### -----------------------------------

################################################################################

#### ukládám tabulky n-gramů ------------------------

setwd(paste(mother_working_directory, "vystupy", sep = "/"))

for(n in 2:5){

    n_grams_table <- get(
        paste(
            setNames(
                c("bi", "tri", "quadri", "penta", "hexa", "
                    hepta"),
                c(2, 3, 4, 5, 6, 7)
            )[as.character(n)],
```

```
            "grams_table",
            sep = "_"
        )
    )

    write.table(
        x = n_grams_table,
        file = paste(
            setNames(
                c("bi", "tri", "quadri", "penta", "hexa", "
                   hepta"),
                c(2, 3, 4, 5, 6, 7)
            )[as.character(n)],
            "_grams_table.txt",
            sep = ""
        ),
        row.names = FALSE,
        col.names = TRUE,
        sep = ";"
    )

    flush.console()
    print(
        paste(
            "Tabulka pro ",
            n,
            "-gramy je uložena.",
            sep = ""
        )
    )

}

setwd(mother_working_directory)


#### -----------------------------------

################################################################################
################################################################################
################################################################################
```

Kód 10: Procedura `n_gramming.R`

8.11 Komponenta `ui.R`

```
################################################################################
################################################################################
################################################################################

library(shiny)


################################################################################
################################################################################
```

```
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$script(type = "text/javascript",
27                   src = "busy.js")
28
29     )
30
31   ),
32
33   div(id = "header",
34       div(id = "title", "The Next Word Prediction"),
35       div(id = "subsubtitle",
36
37           HTML("Prediction of the <i>n</i>-th word based on
                given
38               previous <i>n</i>-1 words and "),
39
40           tags$a(
41             href = "http://en.wikipedia.org/wiki/N-gram",
42             HTML("<i>n</i>-gramming"),
43             target = "_blank"
44           ),
45
46           HTML("&bull;"),
47
48           "Implemented by",
49
50           tags$a(
51             href = "http://www.fbmi.cvut.cz/user/stepalu2",
52             "Lubomír Štěpánek",
53             target = "_blank"
54           )
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            ################################################################
86
87            ## druhá záložka
88
89          conditionalPanel(
90
91            condition = "input.conditionedPanels == 2",
92
93            strong(paste("Here you can find some pieces of
                 information",
94                      "about the application.",
95                      sep=" "
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
                prediction of", em("n"), "-th",
171             "word most likely following the previous (", em("n -
                1"), ") words words inputted by user",
172             "Last, but not least piece of purpose is to pass 4
                IZ470 Web Data Mining subject practised",
```

```
173             "at University of Economics, Prague"),
174
175         br(),
176
177         p(h4("Analysis, preprocessing before algorithm
                deployment")),
178
179         p(paste("- data originate from well-known HC corpora ('
                Helsinki Corpora'),",
180                 "blog, twitter and news corpus datasets were
                       used", sep = " ")),
181
182         p(paste("- before analysis, all vulgarisms, swear and
                stop words were",
183                 "removed from the datasets",sep = " ")),
184         p("-", em("n"), "-grams for", em("n"), HTML("&\#x2208;"
                ), "{2, 3, 4}",
185           "were worked out"),
186         p(paste("- 2-grams, 3-grams and 4-grams contain at all
                45 354,",
187                 "23 989, and 5 570 phrases, respectively", sep
                       = " ")),
188         p("- a naive frequentist probability model based on (",
                em("n"), "+ 1)-gram",
189           "for predicting of (", em("n"), "+ 1)-th word
                  according to",
190           "previous", em("n"), "inputted words was conducted"),
191
192         br(),
193
194         p(h4("Pseudocode of the algorithm")),
195
196         code("i-gram <- database of pairs [i-words phrase, its
                (i + 1)-th likely following word]"),
197         br(),
198         code("n <- min(3, number of words of the phrase
                inputted by user)"),
199         br(),
200         code("while n > 0 do"),
201         br(),
202         code("...phrase <- last n words of the phrase inputted
                by user"),
203         br(),
204         code("...{M} <- {0}"),
205         br(),
206         code("...for all pairs in n-gram do"),
207         br(),
208         code("......if phrase in pair then"),
209         br(),
210         code(".........{M} <- pair"),
211         br(),
212         code("...if {M} <> {0} then"),
213         br(),
214         code("......return (n + 1)-th following word of the
                most frequent pair"),
215         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
                  of",
229             "a next word"),
230
231           br(),
232
233           br(),
234
235           p(h4("Usage of the application, conclusions")),
236
237           p("- the application s available at http://shiny.
                statest.cz:3838/the_next_word_prediction/"),
238           p("- a user can intuitively input her phrase which
                contains at least one word"),
239           p("- after pushing the 'Predict the next word!' button
                the most likely following word is predicted
240               and the conditional probability of correctness of
                     the prediction, given the inputted phrase
241               and given 'static' n-grams based on provided
                     datasets, is returned"),
242
243           br(),
244
245           p(h4("About the author")),
246
247           p("Lubomir Stepanek, M. D."),
248           p("- Department of Biomedical Informatics"),
249           p("- Faculty of Biomedical Engineering"),
250           p("- Czech Technical University in Prague"),
251           HTML("<a href='mailto:lubomir.stepanek[AT]fbmi.cvut.cz
                '>
252               lubomir.stepanek[AT]fbmi[DOT]cvut[DOT]cz</a>"),
253           br(),
254
255           value = 2
256         ),
257
258
259         ###################################################################
260
261         id = "conditionedPanels"
262
263
264         ###################################################################
```

```
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     ## loaging 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 = " ")
              [[1]])
129
130       if(k >= n_of_words){
131
132         return(my_phrase)
133
134       }else{
135
136         return(
137           paste(
138             strsplit(my_phrase, split = " ")[[1]][
139               (n_of_words - k + 1) : n_of_words
140               ],
141             collapse = " "
142           )
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][
                  my_index]
169             )
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][
                  my_index]
189             )
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
```

```
            which_to_choose <- which(
              my_1_word_vocabulary()$phrase == my_phrase
              )
            my_index <- which.max(
              my_1_word_vocabulary()$frequency[which_to_choose]
              )[1]
            return(
              my_1_word_vocabulary()$next_word[which_to_choose
                  ][my_index]
              )

          }else{

            return("the")

          }

        }

    }


    ######################################################################

    ## helper function for my conditional probability

    getMyConditionalProbability <- function(my_phrase){

      my_phrase <- getEndOfMyPhrase(my_phrase, 3)

      if(length(strsplit(my_phrase, split = " ")[[1]]) == 3){
        if(my_phrase %in% my_3_word_vocabulary()$phrase){

          which_to_choose <- which(
            my_3_word_vocabulary()$phrase == my_phrase
          )
          my_numerator <- my_3_word_vocabulary()$frequency[
              which.max(
            my_3_word_vocabulary()$frequency[which_to_choose]
          )[1]]
          return(min(1,
            my_numerator / sum(
              my_3_word_vocabulary()$frequency[which_to_choose]
              ))
          )

        }else{

          my_phrase <- getEndOfMyPhrase(my_phrase, 2)

        }
      }

      if(length(strsplit(my_phrase, split = " ")[[1]]) == 2){
        if(my_phrase %in% my_2_word_vocabulary()$phrase){

```

```
255             which_to_choose <- which(
256               my_2_word_vocabulary()$phrase == my_phrase
257             )
258             my_numerator <- my_2_word_vocabulary()$frequency[
                    which.max(
259               my_2_word_vocabulary()$frequency[which_to_choose]
260             )[1]]
261             return(min(1,
262               my_numerator / sum(
263                 my_2_word_vocabulary()$frequency[which_to_choose]
264                 ))
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()$frequency[
                    which.max(
281               my_1_word_vocabulary()$frequency[which_to_choose]
282             )[1]]
283             return(min(1,
284               my_numerator / sum(
285                 my_1_word_vocabulary()$frequency[which_to_choose]
286                 ))
287             )
288
289           }else{
290
291             return("> 0")
292
293           }
294
295         }
296
297       }
298
299
300       ####################################################################
301
302       output$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,
          transparent 40px),
28     radial-gradient(white, rgba(255,255,255,.15) 1px,
          transparent 30px),
```

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

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)