Webscraping a jazykové zpracování korpusů anglicky psaných textů, n-gramming a vývoj aplikace `the_next_word_prediction`, vše v prostředí a jazyce R

4IZ470 Dolování znalostí z webu

Lubomír Štěpánek

31. května 2017

Obsah

1 Zadání úlohy

Doporučený postup úlohou lze shrnout v následujících bodech.

(i) Cílem úlohy je vybrat nástroj[1] spadající do oblasti *web content mining*, *web structure mining*, *web usage mining*, eventuálně do domény zpracování textů přirozeného jazyka, případně dle zájmu vyvinout nástroj vlastní, který bude vycházet z některého publikovaného (případně nově navrženého a dobře popsaného) algoritmu dolování z webu či textů.

(ii) Dále navrhnout vlastní design experimentu s daným nástrojem a vhodnými daty (vlastními či již sesbíranými).

(iii) Průběžné výsledky prezentovat ústně s pomocí slideové prezentace na cvičení předmětu dne 24. 4. 2017.

(iv) Konečné výsledky prezentovat do 22. 5. 2017 formou krátké závěrečné zprávy (5-8 stran formátu A4), kde bude uveden popis nástroje, principy jeho fungování a postup experimentem provedeným nad vhodnými daty pomocí zvoleného nástroje.

2 Řešení úlohy

2.1 Úvod

V rámci nabízených možností je práce tématicky zaměřena především na zpracování textu přirozeného jazyka (*natural language processing*, NLP) a dále na vývoj vlastní aplikace, která poté se zpracovaným textem pracuje.

Nejdříve byly získány již existující korpusy anglických textů různého původu (jde o texty pocházející z anglických zpravodajských relací, blogů a z twitteru), jde o tzv. *Helsinki corpora* [1], které jsou za určitých podmínek a pro určité účely dostupné online. K nim byl přidán ještě

[1] Buďto některý nástroj ze stránky http://www.kdnuggets.com/software, anebo po konzultaci i jiný dle vlastního uvážení.

menší korpus získaný vlastním webscrapingem malé části anglicky mluvící Wikipedie. Korpusy byly poté jednotně předzpracovány. Bloky textů korpusů byly nejdříve rozděleny na věty, a to podle určité heuristiky. Dále byl text vět 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. Velá písmena byla v rámci redukce „variability“ textu převedena bez ztráty infromace na malá písmena. Následovala fáze tokenizace vět na jednotlivá slova, výsledkem tam byly pro každý korpus množina uspořádaných, různě dlouhých k-tic slov, kde $k \geq 1$ je pro každou větu nějaké přirozené číslo. V této fázi je možné odtsranit ta slova, která v textech nejsou z nějakého smysluplného důvodu žádoucí. Jde především o vulgarismy a jinak nevhodná slova (lze je očekávat hlavně v korpusu pocházejícího ze sociální sítě twitter), 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.

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 [2]. Jde o n-členná sousloví, tedy n-tice sousedících, po sobě jdoucích slov. 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. 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.

Angličtina je v rámci úlohy použita zcela záměrně – jde o analytický jazyk [3], jehož textové a jazykově-statistické zpracování je docela dobře možné, především díky minimálnímu zatížení anglických textů morfematikou ohebných slovních druhů; lemmatizaci slov lze v podstatě vynechat, aniž by byla ohrožena kvalita n-grammingu. 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 vyjímek.

2.1.1 Původ již existujících korpusů

Jak již bylo naznačeno, největší část zpracovávaného textu pochází z tzv. *Helsinki corpora*, což jsou rozsáhlé korpusy tematicky různorodých textů. Jsou za určitých podmínek a pro určité účely dostupné online, a sice na webu

http://www.helsinki.fi/varieng/CoRD/corpora/HelsinkiCorpus/.

Volně stáhnout však korpusy nelze. Text, který však z korpusů vychází a je sestaven z těch částí helsinských korpusů, které jsou sesbírány z textů anglických[2] zpravodajských relací,

[2]Vždy jde o americkou angličtinu, nikoliv britskou či jinou.

anglicky psaných blogů a z anglicky psaných tweetů sociální sítě twitter, je možné získat v rámci absolvování masivního otevřeného online kurzu *Data Science* na online univerzitě Coursera®, který nabízí John Hopkins Bloomberg School of Public Health. Tato malá část původního korpusu je ke stažení např. zde

https://d396qusza40orc.cloudfront.net/dsscapstone/dataset/Coursera-SwiftKey.zip.

Vzhledem k tomu, že kurz je volně dostupný a během něj je nakládání s odekzovanými 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.

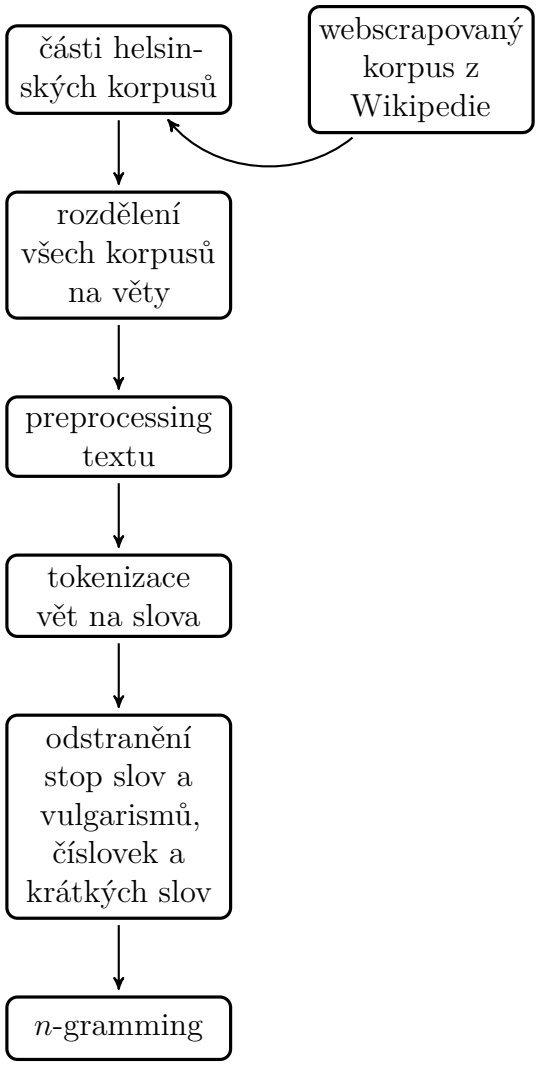
2.2 Metody zpracování textu

V rámci relativně rozsáhlé fáze preprocessingu a processingu textových dat bylo nutné provést několik asynchronních úloh („jednorázová předpočítání“), které byly náročné na vývojový i exekuční čas. Veškeré části úlohy byly naprogramovány a řešeny v prostředí R, které je určeno pro statistické výpočty a následné grafické náhledy [4]. Současně existuje online repozitář umístěný na platformě GitHub a obsahující všechny zdrojové kódy, kde je možné je i upravovat; je dostupný zde

https://github.com/LStepanek/4IZ470_Dolovani_znalosti_z_webu/.

Veškerý komentovaný zdrojový kód je rovněž uveden ve stati „Implementace řešení asynchronních a synchronních úloh v R“. I přes bohatou řadu knihoven pro *text mining* a *natural language processing*, kterou má jazyk R k dispozici, byly prakticky všechny funkce napsány „de novo“ vlastními silami a jen s využitím základních klauzulí jazyka R (i ze cvičných důvodů); tedy bez použití dostupných intermediánních či vyšších vestavěných funkcí.

Schéma pořadí jednotlivých asynchronních úloh tak, jak byly v rámci zpracování textu prováděny, je naznačeno na obrázku 1.



Obrázek 1: Schéma (zjednodušené) asynchronních úloh během zpracovní textu pro účely n-grammingu

2.2.1 Webscraping Wikipedie

Smyslem webscrapingu je stáhnout textový obsah nějakého počtu anglicky psaných stránek Wikipedie za účelem zvětšení celkového rozsahu všech použitých korpusů. Zároveň lze předpokládat, že vzhledem k tematickému zaměření části helsinských korpusů, kterou máme k dispozici, totiž texty zpravodajských relací (stručnější, gramaticky správné anglické fráze), blogů (jazykově bohaté, rozvité věty) a twitteru (krátká sdělení s velmi relaxovanou gramatikou), bude „korpus“ získaný sloučením volných textů určitého množství stránek anglické Wikipedie nejen kvantitativním obohacením celkového užitého korpusu, ale do jisté míry i obohacením kvalitativním – 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.

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 Style 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.

Pro účely webscrapingu jedné stránky Wikipedie byla napsána funkce

`webscrapeMyWikipediaPage()`,

jejímž 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 `<hi>` a `</hi>`), tabulky (vymezené tagy `<table>` a `</table>`) apod. bolný 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 entitatmi (`&...;`) či wikipedickými tagy (obvykle `[...]`). S aplikací Chomského hierarchie jazyků a Chomského pravidla [5], 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.

Následovala procedura (ve skriptu `webscraping.R`), která na vstupu použila jeden zvolený článek, z něj vyextrahovala všechny interní outlinky vedoucí na jiné stránky typu článek anglické Wikipedie a současně scrapovala text tohoto článku. Outlinky byly uloženy do globální proměnné s linky. V druhé iteraci byla vyscrapována stránka (a do stejné globální proměnné s linky uloženy její outlinky) dostupná z prvního linku v globální proměnné s linky. V třetí iteraci byl proces opakován se stránkou odkazovanou druhým linkem v globální proměnné s linky. Proces byl opakován do chvíle, než bylo získo určité, uživatelem definované

množství vyscrapovaných wikipedických stránek[3].

S volbou jednoho iniciálního anglicky psaného článku na Wikipedii, v našem případě konkrétně[4]

https://en.wikipedia.org/wiki/World_War_II.

a s podmínkou, že chceme vyscrapovat právě 900 wikipedických anglických stránek typu článek, byl nakonec sestaven vlastní (byť malý) „korpus“ wikipedických stránek, uložený v souboru `en_US.wikipedia.txt`. Jak bude dále naznačeno, dostupné části helsinského korpusu jsou samy o sobě natolik velké, že provedení asynchroních úloh nad textovými daty korpusu vyžaduje extermní množství výpočetního času.

Lze však dovodit, 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).

2.2.2 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. Algoritmu 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 (žádá, jedna, nebo i více), velké písmeno – tato sekvence je matchována regulárním výrazem `"\\.\\s*[A-Z]+"`,
- sekvence otazník, mezera (žádá, jedna, nebo i více), velké písmeno – tato sekvence je matchována regulárním výrazem `"\\?\\s*[A-Z]+"`,
- sekvence vykřičník, mezera (žádá, jedna, nebo i více), velké písmeno – tato sekvence je matchována regulárním výrazem `"\\!\\s*[A-Z]+"`,
- sekvence dvojtečka, mezera (žádá, 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.`

[3]Hypoteticky by mohla být procedura nechtěně ukončena i dříve – a to např. tehdy, kdyby volba iniciální strány 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*).

[4]Volba právě tohoto článku jako iniciálního nemá žádný hlubší smysl, ale článek obsahuje relativně velké množství volného textu.

je tento maskou pro pattern `"\\.\\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 proskenován (každá alespoň dvojice sousedních znaků je porovnána 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 zkracujího řetězce korpusu, jak „zpředu“ vždy zkrácen o substring do prvního dalšího indexu označujícího novou hranici vět. Zřejme 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.

Aktuálně je touto asynchronní úlohou zpracováno pouze cca 0,5 % dostupného korpusu – není problémem rozdělit na věty celý korpus, avšak pro účely aplikace, která pak v daném seznamu n-gramů, jež roste s objemem vstupního korpusu, vyhledává vhodné zástupce n-gramů *on-the-fly*, znamená příliš velký korpus velké zpomalení chodu aplikace.

2.2.3 Preprocessing textu

V rámci navazujícího preprocessingu textu jsou pomocí skriptu `preprocessing.R` 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 intepunkce včetně závorek a dalších znaků netypických pro přirozený text (tedy hashtagů, zavináčů, smajlíkových forem apod.).

2.2.4 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 preporcessingu, je tokenizace vět na slova relativně snadnou úlohou. V rámci skriptu `tokenization.R`

jsou namapovány indexy mezer ve větách a podle nich jsou pak věty rozděleny na slova pomocí uživatelem definované funkce `splitSentenceIntoWords()`. Výsledkem je tedy pro každý korpus tolik uspořádaných k-tic[5], kolik obsahuje korpus vět; přirozené $k \geq 1$ udává pro každou větu počet slov, na které byla rozdělena.

2.2.5 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. [6]. 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 jejich „přirozenosti“, což nakonec zhoršuje výsledky sémantické analýzy. Navíc se běžně uváděné seznamy anglických stop slov významně liší. Z tohoto důvodu nebyla stop slova z vět v naší analýze odstraněna[6]. Progamrátorsky jde ale 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, odstraněna byla vulgární nebo jinak nevhodná slova, už jen z pragmatického důvodu určité serióznosti analýzy a na ní postavené aplikace. Nelze totiž vyloučit, že se nemohou nevohdná slova vyskytnout především v části korpusu pocházejícího ze sociální sítě twitter. 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 součástí idiomatické fráze, u které máme zájem, aby byla součástí n-gramu, bude pravděpodobně vyjádřena číslovkou (slovem).

2.2.6 n-gramming

Vstupem pro n-gramming je uspořádaná k-tice slov, která reprezentuje k-slovnou větu. Korpus je pak složen z nějakého množství takto reprezentovaných vět. n-gramem rozumíme n-člennou sekvenci po sobě jsoucích 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

[5] k zde není konstanta, variuje mezi jednotlivými větami.

[6] Byť je pro to ve skriptu `postprocessing.R` vytvořena procedura.

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

Algoritmus, který z korpusu získá všechny n-gramy, he relativně jednoduchý. V podstatě každou větu korpusu reprezentovanu uspořádaným seznamem slov proskenuje „čtecím okénkem“ délky n slov a vždy takový 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 frekvence pak můžeme chápat jako bodové odhady pravděpodobností, s jakými se dané n-gramy vyskytují v 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()`, jejímž vstupem je jednak n, jednak věta reprezentovaná jako uspořádaná k-tice slov. Výstupem jsou všechny n-gramy, které lze z věty na vstupu získat. Skript `n_gramming.R` počítá tabulky frekvencí pro jednotlivé n-gramy.

2.3 Aplikace `the_next_word_prediction`

Aplikace, podobně jako asynchronní úlohy v rámci zpracování textu korpusů 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, $n > 1$.

2.3.1 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 vyhovujících (n_0+1)-gramů je pak vybrán ten, který má největší relativní četnost (v korpusu) a jeho (n_0+1)-té slovo je vráceno uživateli jako to, které nejpravděpodobněji následuje jeho zadanou $(n-1)$-člennou frázi.

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

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

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

$$w^*_{n_0+1} = \underset{w_{n-n_0} w_{n-n_0+1} \ldots w_{n-1} w_{n_0+1} \in \mathcal{G}^{(n_0+1)}_{w_1 w_2 \ldots w_{n-1}}}{\arg\max} \{\hat{P}(w_1 w_2 \ldots w_{n-1} w_{n_0+1} \mid w_1 w_2 \ldots w_{n-1})\}.$$

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. Někdy se též popsanému přístupu říká back-off model, neboť v podstatě nejdříve vyhledává shodu se zadanou frází mezi 5-gramy (pentagamy); pokud ji nenajde, pokračuje mezi 4-gramy (quadrigramy) atd. až k 1-gramům (unigramům).

2.3.2 Komponenty aplikace

Aplikace se skládá z následujících částí:

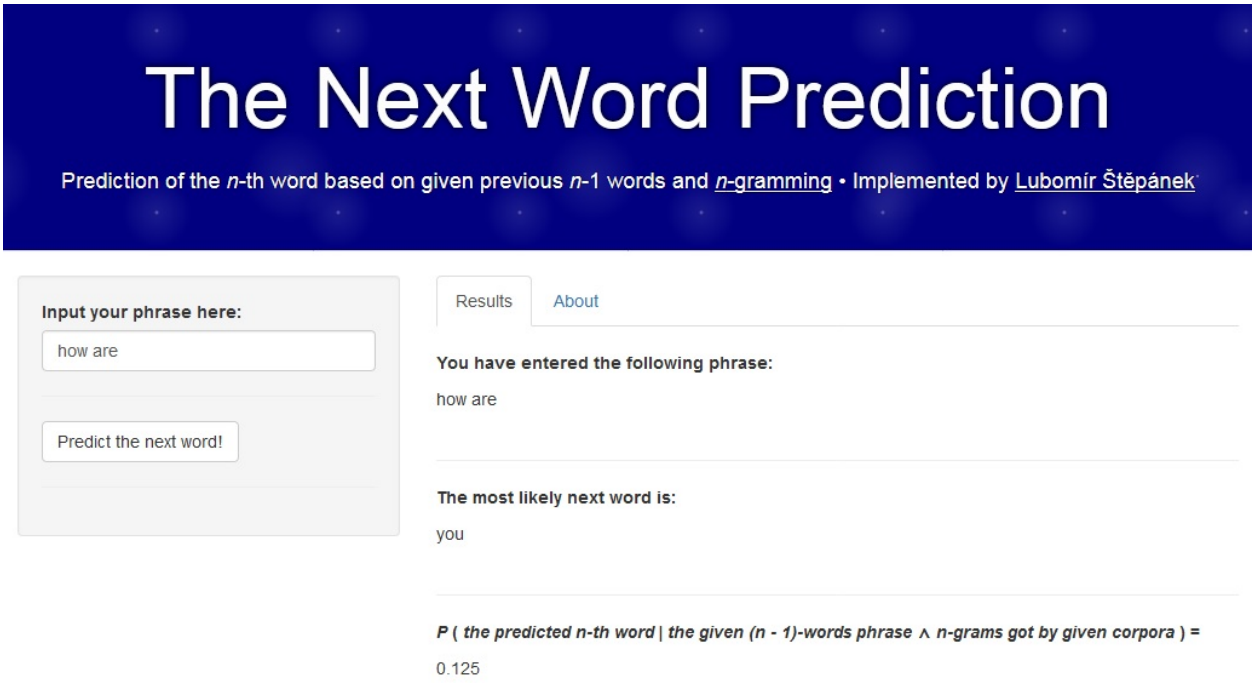
- `ui.R`
- `server.R`
- slovníky n-gramů, tedy `my_`i`_word_vocabulary` pro $i \in \{1,2,3\}$
- složka `www`:
  - `style.css`

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. 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.



Obrázek 2: Uživatelský interface domovské stránky aplikace

Popišme nyní detailněji jednotlivé části aplikace.

`ui.R` — Název vyplývá z fráze *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. 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.

`server.R` — Jádro celé aplikace, obsahuje workhorse funkce, především implementaci backoff modelu výběru tak, jak je popsán ve stati „Princip aplikace“.

`style.css` — Kaskádové styly, které definují rozměry, barvy a další parametry některých prvků aplikace, především headeru.

2.4 Další možné směřování práce

Nabízí se zpracování celého korpusu, eventuálně 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 frázi, ale vypsat těchto slov hned několik a řadit je sestupně dle pravděpodobnosti.

2.5 Implementace řešení asynchronních a synchronních úloh v R

V následujících statích je uveden R-kový zdrojový kód asynchronních úloh a rovněž R-kový zdrojový kód webové aplikace `the_next_word_prediction`.

2.5.1 __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"
    )

}


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

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

2.5.2 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

        ))

    }

}


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

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

2.5.3 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])
            )

        }

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

```
            my_sentences[j] <- substr(
                my_sentences[j],
                1,
                (nchar(my_sentences[j]) - 1)
            )

        }

    }

    return(my_sentences)

}


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

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

#### funkce na rozdělení věty na slova ----------------------------------------

splitSentenceIntoWords <- function(

    my_sentence

){

    # '''
    # Rozděluje větu "my_sentence" na jednotlivá slova.
    # '''

    return(
        strsplit(
            x = my_sentence,
            split = " "
        )[[1]]
    )

}


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

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

#### funkce pro tvorbu n-gramů ------------------------------------------------
```

```
getNGrams <- function(

    my_splitted_sentences,
    n = 2

){

    # '''
    # Nad větou rozdělenou na slova "my_splitted_sentences" vytvoří
    # všechny n-gramy pro zadané "n".
    # '''

    output <- NULL

    if(length(my_splitted_sentences) >= n){

        for(i in 1:(length(my_splitted_sentences) - n + 1)){

            output <- c(

                output,
                paste(
                    my_splitted_sentences[i:(i + n - 1)],
                    collapse = " "
                )

            )

        }

    }

    return(output)

}


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

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

#### funkce pro webscraping jedné stránky Wikipedie (typu článek)
#### a pro následnou úpravu formátu do podoby volného textu ------------------

webscrapeMyWikipediaPage <- function(

    page_url
```

```
){

    # '''
    # Funkce stáhne statický HTML obsah jedné stránky z (anglické)
    # Wikipedie, která je pod odkazem "page_url". Poté extrahuje jen
    # odstavcové statě ohraničené HTML tagy <p>...</p>.
    # Z nich pak odstraní veškeré další HTML tagy, HTML entity či
    # wikipedické tagy.
    # Nakonec vrací textový řetezec odpovídající jen přirozenému
    # textu v odstavcích dané stránky Wikipedie.
    # Kromě toho ještě z textu stránky extrahuje interní webové odkazy
    # na další stránky Wikipedie, které je poté možné scrapovat.
    # '''


    ## stahuji statický HTML obsah ------------------------------------------

    my_html <- readLines(
        con = page_url,
        encoding = "UTF-8"
    )


    ## extrahuji jen odstavcové statě ohraničené HTML tagy <p>...</p> ---------

    my_raw_paragraphs <- my_html[
        grepl("<p>", my_html) & grepl("</p>", my_html)
    ]


    ## očišťuji text paragrafů o HTML tagy, HTML entity a wikipedické tagy ----

    my_paragraphs <- gsub("<.*?>", "", my_raw_paragraphs)
    my_paragraphs <- gsub("&.*?;", "", my_paragraphs)
    my_paragraphs <- gsub("\\[.*?\\]", "", my_paragraphs)
    my_paragraphs <- gsub("\t", "", my_paragraphs)  ## zbavuji se taublátorů


    ## vytvářím jeden dlouhý řetězec (odstavec) -------------------------------

    my_text_output <- paste(my_paragraphs, collapse = " ")


    ## extrahuji z textu všechny webové interní linky na další stránky
    ## Wikipedie --------------------------------------------------------------
```

```
    my_links <- paste(
        "http://en.wikipedia.org",
        gsub(
            '\\"',
            "",
            gsub(
                '(.*)(href=)(\\"/wiki/.*?\\")(.*)',
                "\\3",
                my_raw_paragraphs[
                    grepl("href=", my_raw_paragraphs)
                ]
            )
        ),
        sep = ""
    )


    ## odstraňuji nesmyslné outlinky -- ty, co obsahují mezeru, eventuálně
    ## ty, co odkazují na celý portál nebo kategorii (jsou o obvykle jen
    ## seznamy hesel, tedy nevhodné pro sestavené korpusu) --------------------

    my_links <- my_links[!grepl(" ", my_links)]
    my_links <- my_links[!grepl("Portal:", my_links)]
    my_links <- my_links[!grepl("Category:", my_links)]


    ## vracím výstup ----------------------------------------------------------

    return(
        list(
            "text_stranky" = my_text_output,
            "outlinky_stranky" = my_links
        )
    )


}


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

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

2.5.4 webscraping.R

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

## vytvářím malý korpus z nějakého počtu stránek anglicky psané
## Wikipedie ---------------------------------------------------------------

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

#### vytvářím zatím prázdný vektor linků na wikipedické anglické stránky
#### a vektor korpus pro jejich texty,
#### oba budu postupně populovat ------------------------------------------

my_links <- NULL
my_wikipedia_corpus <- NULL
my_initial_page_url <- "https://en.wikipedia.org/wiki/World_War_II"
number_of_scraped_pages <- 0           ## počet aktuálně scrapovaných
                                       ## wikipedických stránek
how_many_pages_i_would_like_to_scrape <- 900
                                       ## 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"
)


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

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

#### ukládám wikipedický korpus ------------------------------------------

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

writeLines(
    text = my_wikipedia_corpus,
    con = "en_US.wikipedia.txt"
)


setwd(mother_working_directory)


#### -----------------------------------------------------------------------
```

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

2.5.5 corpora_loading.R

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

## loaduji korpusy a množiny stop slov a nevhodných slov ----------------------

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

#### loaduji korpusy ------------------------------------------------------------

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

for(my_corpus_source in c(

    "blogs",
    "news",
    "twitter"

)){

    # '''
    # nahrávám všechny korpusy a ukládám je pod originálními jmény
    # '''

    assign(
        paste(my_corpus_source, "corpus", sep = "_"),
        readLines(
            con = paste("en_US.", my_corpus_source, ".txt", sep = ""),
            encoding = "UTF-8"
        )
    )

    flush.console()
    print(
        paste(
            "Byl nahrán korpus '",
            my_corpus_source,
            "'.",
            sep = ""
        )
```

```
    )

}

setwd(mother_working_directory)


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

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

#### loaduji množiny stop slov a nevhodných slov -----------------------------

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

for(my_word_group_name in c(

    "stop_words",
    "swear_words"

)){

    assign(
        my_word_group_name,
        readLines(
            con = paste(my_word_group_name, ".txt", sep = ""),
            encoding = "UTF-8"
        )
    )

}

empty_words <- c("")    ## definuji množinu "prázných" slov

setwd(mother_working_directory)


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

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

2.5.6 text_splitting.R

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

## text splitting --------------------------------------------------------------

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

#### zadávám, kolik prvních "my_proportion" * 100 procent každého korpusu
#### bude zpracováno pro účely sestavení vektoru s větami ---------------------

my_proportion <- 0.005    ## zřejmě je 0 <= my_proportion <= 1


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

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

#### inicializuji vektor "my_sentences", do kterého se budou ukládat
#### věty všech korpusů -------------------------------------------------------

my_sentences <- NULL


#### nyní dělím všechny korpusy na jednotlivé věty (i vedlejší) ---------------

for(my_corpus_source in c(

    "blogs",
    "news",
    "twitter"

)){

    # '''
    # postupně upravuji a očišťuji daný korpus
    # '''

    ## loaduji korpus ------------------------------------------------------------

    my_corpus <- get(paste(my_corpus_source, "corpus", sep = "_"))


    ## dělím korpus na věty ------------------------------------------------------
```

```
    for(i in 1:ceiling(length(my_corpus) * my_proportion)){

        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 = ""
            )
        )

    }

}


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

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

2.5.7 `preprocessing.R`

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

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

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

#### odstraňuji zkrácená slova s apostrofem -----------------------------------

my_sentences <- gsub("[a-zA-Z]+'[a-zA-Z]+", "", my_sentences)


#### převádím věty korpusu na kódování UTF-8 (některé znaky i přes
#### kódování zdrojových souborů v UTF-8 mohou zůstat "nečitelné") ------------

my_sentences <- iconv(my_sentences, to = "UTF-8")


#### převádím všechna písmena všech vět na malá -------------------------------

my_sentences <- tolower(my_sentences)


#### odtraňuji vícenásobné mezery ve větách -----------------------------------

my_sentences <- gsub("\\s+", " ", my_sentences)


#### odtraňuji leading a trailing spaces (uvozující a koncové mezery) ---------

my_sentences <- gsub("^\\s+", "", my_sentences)
my_sentences <- gsub("\\s+$", "", my_sentences)


#### odtraňuji z vět veškerou interpunkci -------------------------------------

my_sentences <- gsub("[[:punct:]]", "", my_sentences)


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

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

2.5.8 tokenization.R

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

## provádím tokenizaci ---------------------------------------------------------

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

#### dělím všechny věty na slova ----------------------------------------------

my_splitted_sentences <- lapply(
    my_sentences,
    splitSentenceIntoWords
)


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

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

2.5.9 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("â€", "'", x)
)

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


#### odstraňuji stop slova, nevhodná slova a "prádná" 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))
    )

}


#### odstraňuji číslovky ------------------------------------------------------

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


#### odstraňuji slova kratší než tři znaky ------------------------------------

#my_splitted_sentences <- lapply(
#    my_splitted_sentences,
#    function(x) x[nchar(x) > 2]
#)


#### odstraňuji "prázdná" slova -----------------------------------------------

my_splitted_sentences <- lapply(
```

```
    my_splitted_sentences,
    function(x) setdiff(x, "")
)


#### ponechávám jen neprázdné věty (některé prázdné mohly nově vzniknout
#### kvůli očištění o předchozí skupiny slov) ---------------------------------

my_splitted_sentences <- my_splitted_sentences[lapply(
    my_splitted_sentences,
    length
) > 0]


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

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

2.5.10 n_gramming.R

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

## vytvářím n-gramy pro n z {2, 3, 4, 5} -------------------------------------

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

#### vytvářím n-gramy -----------------------------------------------------

for(n in 2:5){

    n_grams <- unlist(lapply(
        my_splitted_sentences,
        function(x) getNGrams(x, n)
    ))

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

```
            sep = "_"
        ),
        n_grams
    )

    flush.console()
    print(
        paste(
            n,
            "-gramy jsou zkompletovány.",
            sep = ""
        )
    )

}


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

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

#### vytvářím tabulky n-gramů řazených abecedně -------------------------------

for(n in 2:5){

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

    my_table <- table(n_grams)

    assign(
        paste(
            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)


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

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

2.5.11 ui.R

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

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

library(shiny)


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

shinyUI(fluidPage(

  #titlePanel("The Next Word Prediction"),

  ## zavádím graficky hezky vypadající header ---------------------------------

  tagList(

    tags$head(

      tags$link(rel = "stylesheet",
                type = "text/css",
                href = "style.css"),

      tags$script(type = "text/javascript",
                  src = "busy.js")

    )

  ),

  div(id = "header",
      div(id = "title", "The Next Word Prediction"),
      div(id = "subsubtitle",

          HTML("Prediction of the <i>n</i>-th word based on given
               previous <i>n</i>-1 words and "),

          tags$a(
            href = "http://en.wikipedia.org/wiki/N-gram",
            HTML("<i>n</i>-gramming"),
            target = "_blank"
          ),

          HTML("&bull;"),

          "Implemented by",
```

```
        tags$a(
          href = "http://www.fbmi.cvut.cz/user/stepalu2",
          "Lubomír Štěpánek",
          target = "_blank"
        )

    )
),

sidebarLayout(

  sidebarPanel(

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

    ## první záložka

    conditionalPanel(

      condition = "input.conditionedPanels == 1",

      textInput(inputId = "my_inputted_phrase",
                label = "Input your phrase here:"
                ),

      tags$hr(),

      actionButton(inputId = "my_button",
                   label = "Predict the next word!"),

      tags$hr()

    ),


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

      ## druhá záložka

    conditionalPanel(

      condition = "input.conditionedPanels == 2",

      strong(paste("Here you can find some pieces of information",
                   "about the application.",
                   sep=" "
                   )
```

```
      ),

      tags$hr()


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


  )


),

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

mainPanel(

  tabsetPanel(

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

    ## první záložka

    tabPanel(
      title = "Results",

      br(),

      p(strong("You have entered the following phrase:")),

      textOutput("inputted_phrase"),

      br(),
      tags$hr(),

      p(strong("The most likely next word is:")),

      textOutput("predicted_word"),

      br(),
      tags$hr(),

      p(strong(
        em("P"),
        "(",
```

```
        em("the predicted n-th"),
        em("word"),
        "|",
        em("the given (n - 1)-words phrase"),
        HTML("&#x2227;"),
        em("n-grams got by given corpora"),
        ") ="
      )),

      textOutput("my_conditional_probability"),

      value = 1
    ),


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

    ## druhá záložka

    tabPanel(
      title = "About",

      br(),

      h4("Introduction"),

      p("- purpose of this application is to offer a tool for prediction
        of", em("n"), "-th",
        "word most likely following the previous (", em("n - 1"), ") words
        words inputted by user",
        "Last, but not least piece of purpose is to pass 4IZ470 Web
        Data Mining subject practised",
        "at University of Economics, Prague"),

      br(),

      p(h4("Analysis, preprocessing before algorithm deployment")),

      p(paste("- data originate from well-known HC
              corpora ('Helsinki Corpora'),",
              "blog, twitter and news corpus datasets were used", sep = " ")),

      p(paste("- before analysis, all vulgarisms, swear
              and stop words were",
            "removed from the datasets",sep = " ")),
      p("-", em("n"), "-grams for", em("n"), HTML("&#x2208;"), "{2, 3, 4}",
        "were worked out"),
```

```
        p(paste("- 2-grams, 3-grams and 4-grams contain at all 45 354,",
                "23 989, and 5 570 phrases, respectively", sep = " ")),
        p("- a naive frequentist probability model based on
          (", em("n"), "+ 1)-gram",
          "for predicting of (", em("n"), "+ 1)-th word according to",
          "previous", em("n"), "inputted words was conducted"),

        br(),

        p(h4("Pseudocode of the algorithm")),

        code("i-gram <- database of pairs [i-words phrase, its (i + 1)-th
             likely following word]"),
        br(),
        code("n <- min(3, number of words of the phrase inputted by user)"),
        br(),
        code("while n > 0 do"),
        br(),
        code("...phrase <- last n words of the phrase inputted by user"),
        br(),
        code("...{M} <- {0}"),
        br(),
        code("...for all pairs in n-gram do"),
        br(),
        code("......if phrase in pair then"),
        br(),
        code(".........{M} <- pair"),
        br(),
        code("...if {M} <> {0} then"),
        br(),
        code("......return (n + 1)-th following word of the most
             frequent pair"),
        br(),
        code("...else"),
        br(),
        code("......n <- n - 1"),


        br(),

        br(),

        p(h4("In progress")),

        p("- Kneser-Ney smoothing with back-off model will be",
          "eventually applied to predict the smooth probability of",
          "a next word"),
```

```
        br(),

        br(),

        p(h4("Usage of the application, conclusions")),

        p("- the application s available at
          http://shiny.statest.cz:3838/the_next_word_prediction/"),
        p("- a user can intuitively input her phrase which contains
          at least one word"),
        p("- after pushing the 'Predict the next word!' button the most
          likely following word is predicted and the conditional
          probability of correctness of the prediction, given the inputted
          phrase and given 'static' n-grams based on provided datasets,
          is returned"),

        br(),

        p(h4("About the author")),

        p("Lubomir Stepanek, M. D."),
        p("- Department of Biomedical Informatics"),
        p("- Faculty of Biomedical Engineering"),
        p("- Czech Technical University in Prague"),
        HTML("<a href='mailto:lubomir.stepanek@fbmi.cvut.cz'>
             lubomir.stepanek[AT]fbmi[DOT]cvut[DOT]cz</a>"),
        br(),

        value = 2
      ),


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

      id = "conditionedPanels"


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

    )

  )

  )

))
```

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

2.5.12 server.R

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

library(shiny)


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

shinyServer(

  function(input, output){

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

    ## loaging of my vocabularies with 1-, 2- and 3-grams

    my_1_word_vocabulary <- reactive({

      data <- read.csv("my_1_word_vocabulary.txt",
                       header = TRUE,
                       sep = " "
                       )

      for(i in 1:3){data[, i] <- as.character(data[, i])}
      data[, 3] <- as.numeric(data[, 3])

      return(data)

    })


    my_2_word_vocabulary <- reactive({
```

```
  data <- read.csv("my_2_word_vocabulary.txt",
                   header = TRUE,
                   sep = " "
  )

  for(i in 1:3){data[, i] <- as.character(data[, i])}
  data[, 3] <- as.numeric(data[, 3])

  return(data)

})


my_3_word_vocabulary <- reactive({

  data <- read.csv("my_3_word_vocabulary.txt",
                   header = TRUE,
                   sep = " "

  )

  for(i in 1:3){data[, i] <- as.character(data[, i])}
  data[, 3] <- as.numeric(data[, 3])

  return(data)

})


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

## outputting of inserted phrase

output$inputted_phrase <- renderText({

  if(is.null(input$my_inputted_phrase)){

    return(" ")

  }else{

    return(input$my_inputted_phrase)

  }


})
```

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

## I am getting a reactive event according to inputted phrase and
## clearing the phrase as well

my_clear_phrase <- eventReactive(input$my_button, {

  inputted_phrase <- tolower(input$my_inputted_phrase)

  while(grepl("  ", inputted_phrase)){

    inputted_phrase <- gsub("  ", " ", inputted_phrase)

  }

  if(substr(inputted_phrase, 1, 1) == " "){
    inputted_phrase <- substr(inputted_phrase,
                              2,
                              nchar(inputted_phrase)
                              )
  }

  if(substr(inputted_phrase,
            nchar(inputted_phrase),
            nchar(inputted_phrase)) == " "){
    inputted_phrase <- substr(inputted_phrase,
                              1,
                              nchar(inputted_phrase)
                              )
  }

  return(inputted_phrase)

})


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

## helper function for getting of last k words of the phrase

getEndOfMyPhrase <- function(my_phrase, k){

  n_of_words <- length(strsplit(my_phrase, split = " ")[[1]])

  if(k >= n_of_words){
```

```
      return(my_phrase)

    }else{

      return(
        paste(
          strsplit(my_phrase, split = " ")[[1]][
            (n_of_words - k + 1) : n_of_words
            ],
          collapse = " "
        )
      )

    }

  }


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

  ## helper function for prediction of the next word

  getMyLikelyNextWord <- 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()$phrase == my_phrase
          )
        my_index <- which.max(
          my_3_word_vocabulary()$frequency[which_to_choose]
          )[1]
        return(
          my_3_word_vocabulary()$next_word[which_to_choose][my_index]
          )

      }else{

        my_phrase <- getEndOfMyPhrase(my_phrase, 2)

      }
    }
```

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

        which_to_choose <- which(
          my_2_word_vocabulary()$phrase == my_phrase
          )
        my_index <- which.max(
          my_2_word_vocabulary()$frequency[which_to_choose]
          )[1]
        return(
          my_2_word_vocabulary()$next_word[which_to_choose][my_index]
          )

      }else{

        my_phrase <- getEndOfMyPhrase(my_phrase, 1)

      }
    }

    if(length(strsplit(my_phrase, split = " ")[[1]]) == 1){
      if(my_phrase %in% my_1_word_vocabulary()$phrase){

          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){

      which_to_choose <- which(
        my_2_word_vocabulary()$phrase == my_phrase
      )
      my_numerator <- my_2_word_vocabulary()$frequency[which.max(
        my_2_word_vocabulary()$frequency[which_to_choose]
      )[1]]
      return(min(1,
        my_numerator / sum(
          my_2_word_vocabulary()$frequency[which_to_choose]
          ))
      )

    }else{

      my_phrase <- getEndOfMyPhrase(my_phrase, 1)

    }
```

```
    }

    if(length(strsplit(my_phrase, split = " ")[[1]]) == 1){
      if(my_phrase %in% my_1_word_vocabulary()$phrase){

        which_to_choose <- which(
          my_1_word_vocabulary()$phrase == my_phrase
        )
        my_numerator <- my_1_word_vocabulary()$frequency[which.max(
          my_1_word_vocabulary()$frequency[which_to_choose]
        )[1]]
        return(min(1,
          my_numerator / sum(
            my_1_word_vocabulary()$frequency[which_to_choose]
            ))
        )

      }else{

        return("> 0")

      }

    }

  }


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

  output$predicted_word <- renderText({

    getMyLikelyNextWord(my_clear_phrase())

  })


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

  output$my_conditional_probability <- renderText({

    getMyConditionalProbability(my_clear_phrase())

  })


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

```
  }

)


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

2.5.13 www/style.css

```
div.busy {
  position:absolute;
  top: 11.9%;
  left: 88.0%;
  margin-top: -100px;
  margin-left: -50px;
  display:none;
  background: rgba(230, 230, 230, .8);
  text-align: center;
  padding-top: 20px;
  padding-left: 30px;
  padding-bottom: 40px;
  padding-right: 30px;
  border-radius: 5px;
}

#header {
  text-align: center;
  color: #fdfdfd;
  text-shadow: 0 0 1px #000;
  padding: 30px 0 45px;
  border-bottom: 1px solid #ddd;
  margin: 0 -30px 20px;
  /* pozadí staženo z http://lea.verou.me/css3patterns/ */
  background-color: #000080;
  background-image:
    radial-gradient(white, rgba(255,255,255,.2) 2px, transparent 40px),
    radial-gradient(white, rgba(255,255,255,.15) 1px, transparent 30px),
    radial-gradient(white, rgba(255,255,255,.1) 2px, transparent 40px),
    radial-gradient(rgba(255,255,255,.4), rgba(255,255,255,.1) 2px, transparent 30px);
  background-size: 550px 550px, 350px 350px, 250px 250px, 150px 150px;
  background-position: 0 0, 40px 60px, 130px 270px, 70px 100px;
}
```

```
#title {
  font-size: 5em;
  text-shadow: 0 0 5px #000;
  margin-bottom: 5px
}

#subsubtitle {
  font-size: 1.3em;
}

#subsubtitle a {
  color: #fdfdfd;
  text-decoration: underline;
}
```

3 Reference

[1] KYTÖ, Merja. *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.

[2] MANNING, Christopher a Hinrich SCHUETZE. *Foundations of Statistical Natural Language Processing*. B.m.: The MIT Press, 1999. ISBN 0-262-13360-1.

[3] GÖRLACH, Manfred. *Introduction to Early Modern English*. B.m.: Cambridge University Press, 1991. ISBN 0521325293.

[4] R CORE TEAM. *R: A Language and Environment for Statistical Computing* [online]. Vienna, Austria: R Foundation for Statistical Computing, 2016. Dostupné z: https://www.R-project.org/

[5] CHOMSKY, N. Three models for the description of language. *IEEE Transactions on Information Theory* [online]. 1956, **2**(3), 113–124. Dostupné z: doi:10.1109/tit.1956.1056813

[6] SAIF, Hassan, Miriam FERNÁNDEZ, Yulan HE a Harith ALANI. On stopwords, filtering and data sparsity for sentiment analysis of twitter. 2014.