Mensa gymnázium

Prediktivní klávesnice

Seminární práce

Jan Slíva
7.11.2021

MENSA GYMNÁZIUM, o.p.s.

SEMINÁRNÍ PRÁCE

Prediktivní klávesnice

Předmět: Informatika
Autor: Jan Slíva

Třída: Kvinta
Školní rok: 2021/2022
Vedoucí práce: Martin Zimen
Rozsah práce (s.):
celkový (včetně této strany, vyjma titulní): 29
vlastní text: 23
přílohy: -
vědecký aparát (poznámky, bibliografie aj.): 3

Formát elektronické podoby práce: .docx, .pdf
Poznámky (vyplývající ze specifik jednotlivých oborů)**: -**

ANOTACE	Tato práce se zabývá vytvořením prediktivní klávesnice, tedy programu, který na základě vstupního textu doporučí uživateli text, který by chtěl pravděpodobně napsat, a tím mu ulehčí psaní. Tuto klávesnici jsem vytvořil v anglickém jazyce a pro vytvoření klávesnice jsem analyzoval data z textového korpusu.
KLÍČOVÁ SLOVA	R, Quanteda, ggplot2, analýza, predikce, datový strom, trie, korpus, měření efektivnosti programů
ANOTACE (AJ)	This work deals with the creation of a predictive keyboard, ie a program that, based on the input text, recommends to the user the text he would probably like to type, and thus makes it easier for him to write. I created this keyboard in English and I analyzed the data from the text corpus to create the keyboard.
KLÍČOVÁ SLOVA (AJ)	R, Quanteda, ggplot2, analysis, prediction, data tree, trie, corpus, program efficiency measurement

1 Obsah

2 Úvod

Prediktivní klávesnice je program, který na základě zadaného textu doporučí následující slova, která by uživatel chtěl pravděpodobně napsat.

Cílem této práce je vytvořit prediktivní klávesnici fungující v angličtině. Slova doporučuji na základě dat z analýzy textového korpusu. Používám při tom programovací jazyk R a jeho knihovny. R jsem si vybral, protože má širokou nabídku knihoven, se kterými je snadné pracovat.

3 Jak to přibližně funguje

Klávesnice by měla fungovat tak, že, když napíšu nějaký text, tak mi to doporučí další slovo, na základě zpracovaných dat z textového korpusu. Pokud napíšu jenom část slova, tak mi to bude doporučovat slova začínající touto částí.

Data zpracovávám tak, že z nich odstraním nepotřebné znaky jako jsou číslice, symboly a nepotřebná interpunkce. Potom nahradím slova s malým výskytem v textu univerzálním slovem, tedy jokerem.

Pak vezmu 1 až 5-gramy, tedy posloupnosti slov o délce 1 až 5 a spočítám jejich výskyt v textu. 2 až 5-gramy přepočítám pomocí podmíněné pravděpodobnosti abych určil, jaká je pravděpodobnost, že když už všechna slova z tohoto n-gramu kromě posledního byla napsána, tak jaká je šance, aby bylo napsáno poslední slovo z tohoto n-gramu. U 1-gramů místo podmíněné pravděpodobnosti vezmu poměrný výskyt v textu, a budu je používat v případě, kdy na vstupu nebude napsáno žádné slovo nebo bude napsána jenom část slova.

Tyto přepočítané n-gramy si ukládám do objektu na bázi stromů, protože hodně n-gramů začíná stejnými slovy. Pro doporučování slov na základě části slova používám trie neboli písmenkové stromy.

Samotná klávesnice funguje tak, že načte textový vstup, který rozdělí na jednotlivá slova a určí, jestli to poslední slovo je část slova nebo dokončené slovo na základě toho, jestli za ním byla napsána mezera. Potom vezme poslední celé slovo, co bylo na vstupu napsáno a bude hledat všechny 2-gramy, které začínají tímto slovem. Pokud na konci vstupu je část slova, tak bude používat trie pro vybrání výsledných 2-gramů. Výsledkem tedy bude seznam posledních slov z dotčených 2-gramů. Potom vezmu dvě, tři, čtyři poslední slova a stejným způsobem budu hledat v 3, 4, 5-gramech. Nakonec mi vyjde seznam posledních slov z n-gramů, z nichž doporučím slova s nejvyšší podmíněnou pravděpodobností.

Seznam 1-gramů používám, pokud je vstup prázdný, nebo pokud je na vstupu pouze část slova – v tom případě použiju trii.

4 Použité technologie

4.1 R

R je interpretovaný dynamický programovací jazyk a prostředí určené pro statistickou analýzu dat a jejich grafické zobrazení. Je vyvíjen R Core Team a používá se zejména pro vývoj statistického softwaru a analýzu dat. V srpnu 2021 dosáhlo R 14. místa v TIOBE indexu[1], což je index, který zhodnocuje popularitu programovacích jazyků. (1)

4.2 C++

C++ je multiparadigmatický kompilovaný programovací jazyk, který vytvořil Bjarne Stroustrup jako rozšíření jazyka C. Existuje více kompilátorů C++, mezi ty nejznámější patří Microsoft Visual C++, Intel

[1] https://www.tiobe.com/tiobe-index/

C++ Compiler a GCC. C++ patří mezi nejpoužívanější programovací jazyky, v TIOBE indexu dosáhlo 4. místa v srpnu 2021. (2)

4.3 Knihovny R

Knihovny R jsou rozšíření jazyka R, které obsahují kód, data a dokumentaci ve standardizované formě. Většinou se instalují pomocí centrální softwarového repozitáře CRAN. Velký počet dostupných knihoven a snadnost jejich instalování a používání je považován za hlavní důvod pro velké rozšíření jazyka R. (3)

4.3.1 Quanteda

Knihovna Quanteda neboli *Quantitative Analysis of Textual Data* je knihovna pro správu a analýzu textových dat vyvíjená Kenneth Benoit, Kohei Watanable a dalšími kontributory. Knihovna je navržena pro použití natural language processing pro všechny typy dat – od dokumentů až po finální analýzu. (4)

4.3.2 ggplot2

ggplot2 je systém pro deklarativní vykreslování dat. Je to implementace přístupu k vytváření grafů *The Grammar of Graphics*, který byl popsán Leland Wilkinson. Zjednodušeně jde o to, že uživatel zadá data a to, jak chce, aby data byla mapována. Další úprava grafu funguje tak, že graf je rozčleněn na části s odlišným významem, např.: osy nebo vrstvy. (5) (6)

4.3.3 rbenchmark

rbenchmark je knihovna určená pro vykonávání testů výkonnosti jakéhokoliv kódu v R. Jejími vývojáři jsou Wacek Kusnierczyk, Dirk Eddelbuettel a Berend Hasselman. Knihovna obsahuje pouze funkci *benchmark*, která spustí kód v požadovaném prostředí a počtu opakování. Výsledky vrátí jako data.frame. (7)

4.3.4 Rcpp

Rcpp je knihovna obsahující R funkce a C++ třídy, které integrují R a C++. Vývojáři jsou Dirk Eddelbuettel, Romain Francois, JJ Allaire a další. Knihovna funguje tak, že všechny datové typy a objekty v R jsou propojeny s odpovídajícími C++ objekty. Například numeric vektory v R jsou namapovány v C++ jako třída Rcpp::NumericVector. (8) (9)

4.3.5 Shiny

Shiny je knihovna, která umožňuje jednoduchý vývoj interaktivních webových aplikací pomocí R. Je vyvíjena RStudio, Inc. (10) Aplikace vytvořené v Shiny se dají zadarmo nahrát na web shinyapps.io, kde jsou pak volně přístupné jako webové stránky.

4.4 Základy R

Protože v práci hodně používám programovací jazyk R a také ukázky kódu v tomto jazyku, tak bych tu chtěl představit jeho základy

V R se přiřazují proměnné pomocí znaku „<-“. Základní datové typy jsou pak vektory, existuje jich 5 typů:

- logical
- integer
- numeric
- complex
- character

Vektor lze vytvořit pomocí funkce *c()*, do které jako argumenty zadám prvky daného typu. Každý vektor má svoji délku, tedy kolik prvků obsahuje, ta se dá zjistit pomocí funkce *length()*. Jednočlenný vektor se dá vytvořit zadáním jednoho prvku.

Listy jsou posloupnosti jakýkoliv objektů – mohou to být vektory, data.frame anebo samostatné listy. Vytvořím je pomocí funkce *list()*, do které zadám prvky listu.

Data.frame je objekt v R, který se používá pro reprezentaci tabulkových dat. Má řádky, které reprezentují jednotlivá pozorování, a sloupce, které reprezentují jednotlivé vlastnosti (např. výskyt slova v textu).

Funkce se definují pomocí přiřazení „<- function()“ k názvu funkce. Hodnoty se pak vrací pomocí „return()“

```
factorial <- function(x){

  if (x <= 1) return(1)

  return(x*factorial(x-1))
}
```

Příklad: Rekurzivní algoritmus pro vypočtení faktoriálu

5 Korpus

Prediktivní klávesnici dělám v anglickém jazyce, protože je to jednodušší – není tam tolik skloňování a časování, které vytváří různé tvary slov, které bych musel v češtině sjednocovat do jednoho tvaru. Textový korpus, který používám jsem si stáhnul z webu HC Corpora[2].

Stáhnul jsem si tedy korpus se články ze zpravodajských serverů, z Twitteru a z blogů. Protože jsem chtěl udělat univerzální prediktivní klávesnici, tak pro finální korpus, který používám pro analýzu kombinuji všechny tři typy a to rovnoměrně.

Zpracoval jsem dvě různé velikosti souborů:

- 1,5 MB dat – cca 300 tis. slov
- 15 MB dat – cca 3 mil. slov

6 Zpracování textu

6.1 Načítání textu ze souboru

Text načítám z textového souboru pomocí knihovny readr[3], následně ho tokenizuji do objektu *tokens* knihovny Quanteda. Při tokenizaci vyřazuji z textu všechna čísla, symboly, odkazy url a nepotřebnou interpunkci, tedy znaky na konci věty, čárky mezi slovy a uvozovky neboli znaky, které nejsou součástí slova a výrazně neovlivňují význam tohoto slova nebo způsob, jak se toto slovo zapisuje. Naopak interpunkce, kterou chci zanechat, jsou apostrofy a pomlčky ve slovech (např.: slovo e-mail).

```
raw_text_twitter <- tolower(read_lines("twitter.txt"))
raw_text_news <- tolower(read_lines("news.txt"))
raw_text_blogs <- tolower(read_lines("blogs.txt"))

raw_text <- append(append(raw_text_twitter, raw_text_news),
raw_text_blogs)
```

[2] http://corpora.epizy.com/corpora.html
[3] https://CRAN.R-project.org/package=readr

```
raw_words <- tokens(raw_text, what = "word", remove_symbols = TRUE,
                    remove_numbers = TRUE, remove_punct = TRUE,
                    remove_url = TRUE, padding = FALSE)

filtered_words_all <- tokens_remove(raw_words, "[^[:alpha:][:punct:]]",
valuetype = "regex")
```

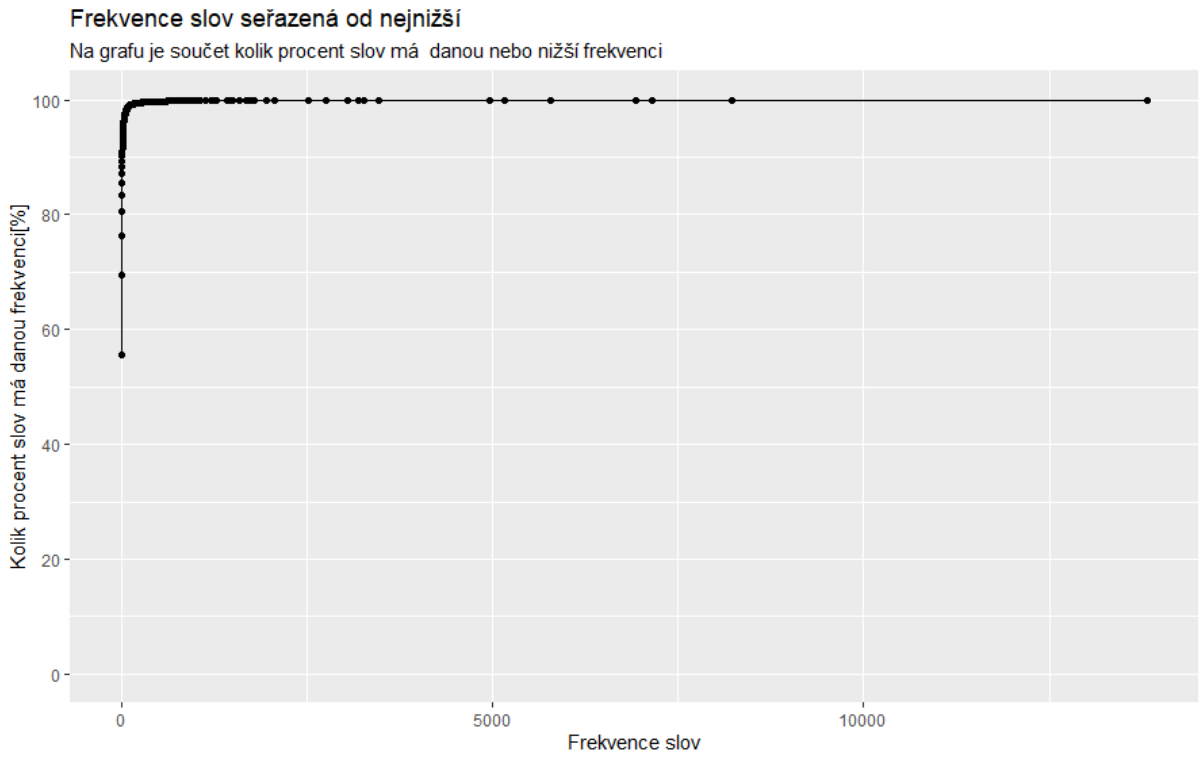
6.1.1 Tokenizace

Tokenizace textu je proces, při kterém je text rozdělen na tokeny a při tom je možné odstranit nepotřebné znaky, například interpunkci. Token je skupina znaků v dokumentu, které jsou seskupeny a spolu vytváří užitečnou významovou jednotku pro zpracování. (11)

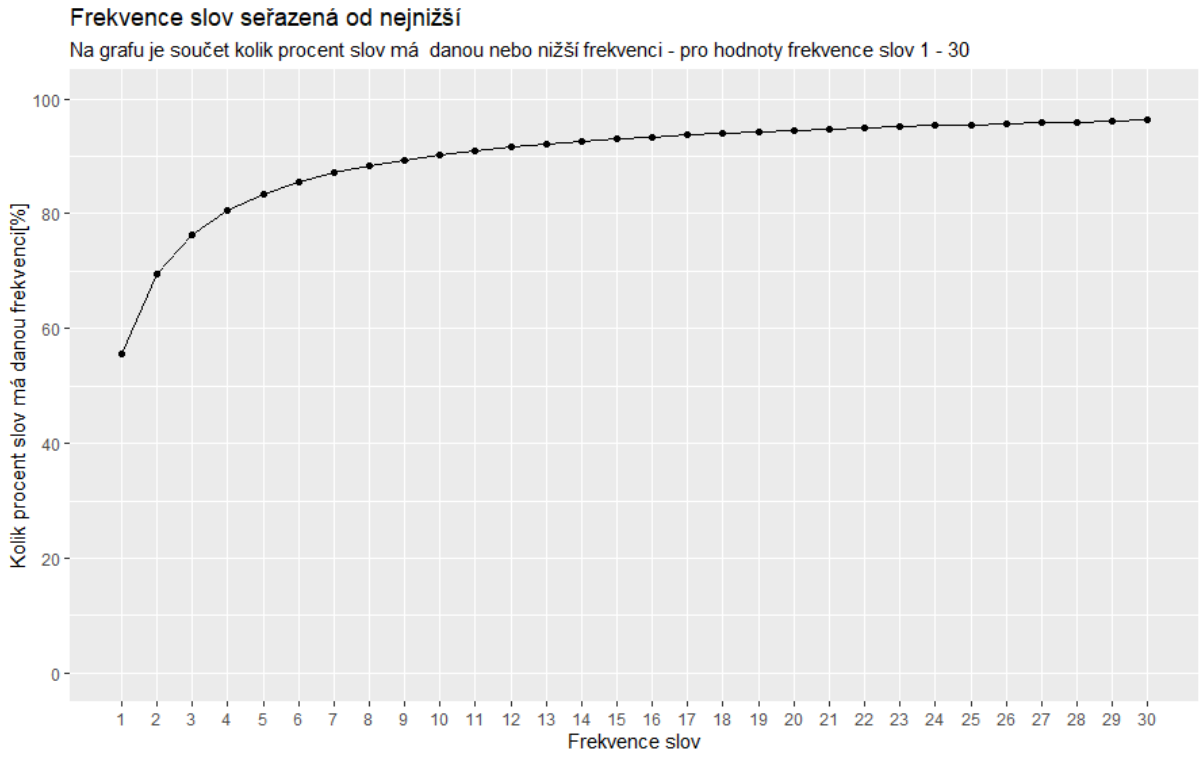
6.2 Nahrazování slov s nízkou frekvencí

Existuje mnoho slov, které mají velmi malé zastoupení celkového textu, a proto je potřeba tyto slova zredukovat, protože je při doporučování slov na prediktivní klávesnici nikdy nepoužiji a budou mi zbytečně zabírat paměť. Tak jsem si vybral určitou hranici, pod kterou tyto slova budu nahrazovat jokery. Jak fungují jokers vysvětluji v kapitole 7 Jak fungují jokers.

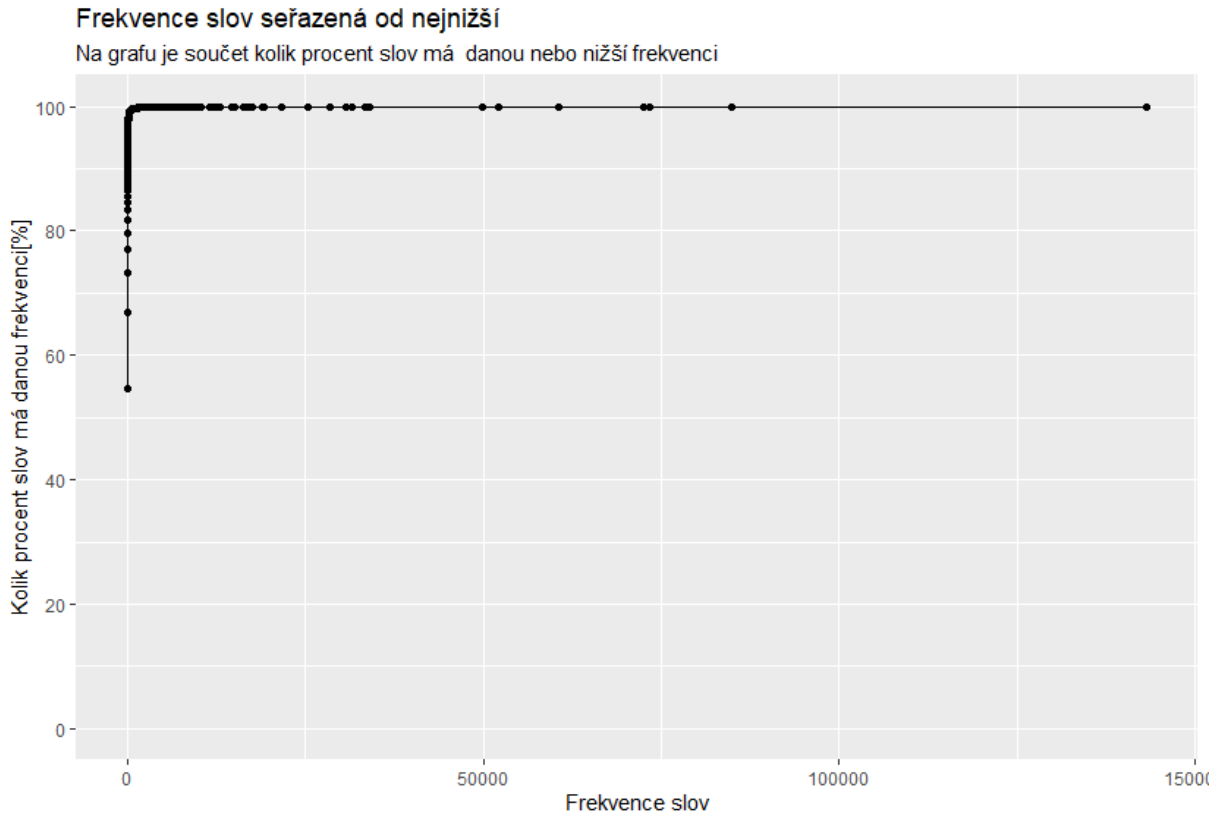
Níže jsou grafy, které ukazují, kolik procent celkových slov zabírají slova s danou a nižší frekvencí výskytu v textu. Pro první dva grafy jsem použil data z 1,5 MB textu, pro druhé dva grafy jsem použil 15 MB textu.



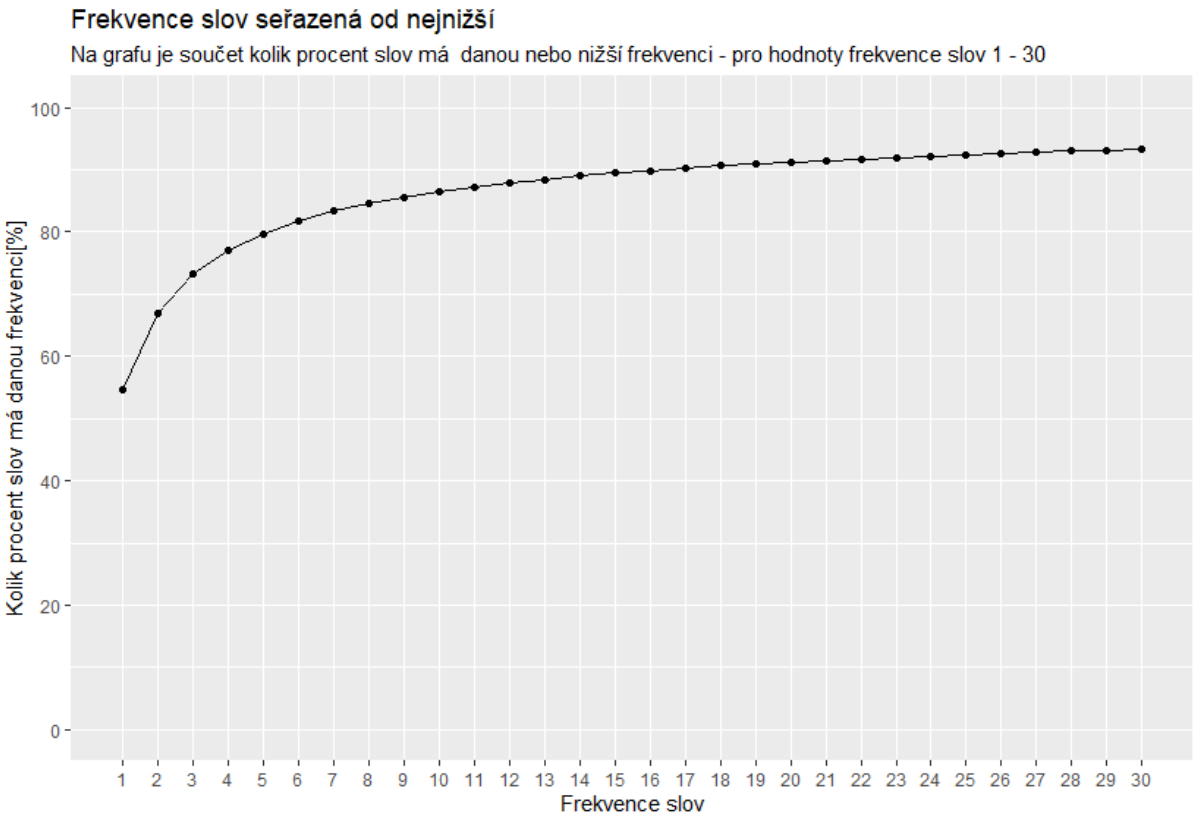
Obrázek 1: Graf pro data z 1.5 MB textu



Obrázek 2: Upravený graf pro data z 1.5 MB textu



Obrázek 3: Graf pro data z 15 MB textu



Obrázek 4: Upravený graf pro data z 15 MB textu

Na grafech je vidět, že slova s nízkou frekvencí výskytu mají o dost vyšší zastoupení než slova s vyšší frekvencí, a proto, když nahradím za jokery větší procento slov, tak to nemá takový vliv na funkci prediktivní klávesnice. Proto jsem se rozhodl nahradit 80 % všech slov, pro 1,5 MB nahrazuji slova s frekvencí výskytu větší rovnou čtyřem, pro 15 MB větší rovnou pěti.

Slova nahrazuji tak, že zjistím seznam všech slov, která chci nahradit, pak už jenom použiju funkci *tokens_replace* z knihovny Quanteda, která nahradí všechna daná slova za jokera v objektu *tokens*, jako jokera jsem použil string "<>", protože jsem tyto konkrétní symboly z textu již odfiltroval, a tak se mi nemůže stát, že by v textu už byli.

```
# determine words to remove
words_to_remove <- (words_freq_all %>% filter(frequency <=
freq_limit))$feature


# replace with jokers
filtered_words <- tokens_replace(filtered_words_all, words_to_remove,
rep(joker, length(words_to_remove)), "fixed")
```

6.3 Převádění objektu *tokens* na tabulku s n-gramy

Pro vypočítání výskytu n-gramů používám knihovnu Quanteda. Výskyty n-gramů si ukládám do *data.*frame, což je vestavěná tabulka v jazyku R. Výskyty 1-gramů, tedy jednotlivých slov počítám následovně:

```
words_dfm <- dfm(filtered_words)

words_freq <- textstat_frequency(words_dfm)
```

```
data_counts <- sum(ntoken(filtered_words))

words_freq$prop <- words_freq$frequency / data_counts

words_freq$order <- 1:length(words_freq$feature)

data_coll <- list()

data_coll[[1]] <- data.frame(order = words_freq$order,
                             ngram = words_freq$feature,
                             freq = words_freq$frequency,
                             prop = words_freq$prop
)
```

V kódu výše je vidět, že pro získávání informací o objektu *tokens* používám funkci dfm, která vrátí objekt document-feature matrix, tedy matici, která obsahuje informace o dokumentu. Z této matice pak pomocí funkce *textstat_frequency* dostávám objekt data.frame se slovy a jejich frekvencemi výskytu. Do tohoto data.frame pak přidávám podmíněnou pravděpodobnost slova a pořadí. Podmíněnou pravděpodobnost vysvětluji v další kapitole [odkaz] a pořadí tam přidávám pro zjednodušení dalšího zpracování. Všechny získané informace pak překopíruji do nového data.frame, který si uložím do listu, kde budou postupně 1 až 5-gramy.

Výskyty 2 až 5-gramů vypočítám následovně za použití funkce *textstats_collocations* z knihovny Quanteda. Jako minimální výskyt n-gramů beru výskyt 2 – tedy, že se daný n-gram aspoň jednou opakuje.

```
data_coll[2:5] <-  mapply(function(x, minCount){

  x_gram <- textstat_collocations(filtered_words, size = x, min_count =
minCount)
  if(nrow(x_gram) > 0){
    ret = data.frame(order = 1:nrow(x_gram), ngram =
x_gram$collocation,
                     freq = x_gram$count, prop = NA)
  }else{
    ret = data.frame(order = c(), ngram = c(),
                     freq = c(), prop = c())
  }
  return(ret)

}, x = 2:5, minCount = rep(2, 4), SIMPLIFY = FALSE)
```

Výstup z funkce *textstat_collocations* je typu data.frame a do nového data.frame si uložím jenom potřebné informace, jen pokud výstup obsahuje alespoň jeden řádek. Do výsledného data.frame si také ukládám informaci o pořadí a zakládám sloupec s pravděpodobnostmi, který prozatímně zaplním NA.

Získal jsem pět tabulek – pro každý 1 až 5-gram, na každém řádku tabulky je uložen samostatný n-gram, jeho výskyt a pořadí tohoto n-gramu.

6.4 Podmíněná pravděpodobnost

Pro správné doporučování slov na prediktivní klávesnici, potřebuji vědět, jaké má slovo pravděpodobnost, že bude napsáno, pokud byla napsána určitá slova, co jsou na vstupu. Proto u

každého 2 až 5-gramu vypočtu jaká je pravděpodobnost, že bude napsáno poslední slovo z n-gramu, pokud již byla napsána všechna předešlá slova z n-gramu.

Postupně pro každý 1 až 4-gram najdu všechny (n+1)-gramy, které začínají tímto n-gramem. Podmíněnou pravděpodobnost posledního slova (n+1)-gramů vypočítám jako poměr výskytu (n+1)-gramu a n-gramu. Vypočtená hodnota je tedy pravděpodobnost toho, že po tom, jak bude na vstupu napsána všechna slova z n-gramu, tak jaká je šance, že bude napsáno poslední slovo (n+1)-gramu. Pravděpodobnost u 1-gramu vypočítám jako podíl výskytu daného 1-gramu na celkovém součtu výskytu všech 1-gramů.

Takto vypočítám pravděpodobnost 1-gramů:

```
data_counts <- sum(ntoken(filtered_words))

words_freq$prop <- words_freq$frequency / data_counts
```

Ostatní pravděpodobnosti vypočítám takto:

```
for (x in 1:4){
  for (index in 1:nrow(data_coll[[x]])){
    prefix = paste0(data_coll[[x]][index,]$ngram, " ")
    thisFreq = data_coll[[x]][index,]$freq

    data_coll[[x + 1]] <- data_coll[[x + 1]] %>% mutate(prop =
                                                          ifelse(
                                                  startsWith(ngram, prefix),
                                                            freq / thisFreq,
                                                            prop))
  }
}
```

Nejdříve vezmu daný n-gram, ke kterému přidám mezeru – toto bude *prefix*. Potom si najdu frekvenci výskytu n-gramu a pak prohledám všechny (n+1)-gramy a najdu takové, které začínají *prefixem*. Jejich pravděpodobnost nastavím jako podíl výskytu (n+1)-gramu a n-gramu.

Tímto způsobem jsem získal data.frame, kde na každém řádku kromě n-gramu, výskytu a pořadí je také pravděpodobnost, tedy hodnota mezi 0 a 1.

	order	ngram	freq	prop
1	1	<> and <>	510	0.291762014
2	2	on and on	3	0.187500000
3	3	will you be	9	0.642857143
4	4	in and out	4	0.114285714
5	5	that <> the	24	0.083044983
6	6	i the only	6	0.750000000
7	7	am i the	6	0.153846154
8	8	more and more	11	0.733333333
9	9	in love with	15	0.576923077
10	10	for you to	10	0.131578947

Obrázek 5: Ukázka data.frame s 3-gramy

7 Jak fungují jokers

Slova nahrazuji jokerem, což je takové slovo, který při doporučování slov funguje jako jakékoliv nahrazené či neexistující slovo. Prediktivní klávesnice bude fungovat tak, že pokud napsané slovo není na seznamu slov, která se vyskytují v textu po nahrazení za jokery, tak toto slovo bude nahrazeno jokerem pro účely vyhledávání.

8 NGramTree

Jednotlivé n-gramy a jejich pravděpodobnosti si ukládám do speciální datové struktury, kterou jsem vymyslel a naimplementoval a pojmenoval jsem ji NGramTree. Struktura je založená na datovém stromě.

8.1 Popis

Moje datová struktura je založená na buňkách – každá buňka obsahuje jedno slovo. Každé buňce odpovídá určitý n-gram, který získám, když pojedu od kořene stromu až k této buňce a budu dávat za sebe všechna slova, na která narazím. Každá buňka má k sobě přiřazenou pravděpodobnost, že bude napsáno jejich slovo po n-gramu náležející k rodiči této buňky – tedy podmíněnou pravděpodobnost, kterou jsem vypočet v kapitole 6.4 Podmíněná pravděpodobnost.

Každá buňka obsahuje ještě trii, ve které jsou zanesené všechny děti na základě jejich slova. Trie používám v případě, kdy na vstupu dostanu část slova a potřebuji najít slova, která začínají tímto prefixem. Používám je také v případě, kdy chci přecházet mezi buňkami, a to z rodiče na dítě.

V každé buňce je ještě seznam několika buněk se slovy s nejvyšší pravděpodobností, na základě toho, kolik slov na klávesnici doporučuji. Používám ho, v případě, když na vstupu nedostanu část slova. Je to takto rychlejší, protože nemusím vyhledávat v trii.

Samostatný kořen neobsahuje žádné slovo a je využíván v případě, když chci doporučit nějaké slovo, ale na vstupu nic není napsané nebo tam je jenom část slova. Kořen reprezentuji jinou třídou a ukládám v něm taky informace o celé datové struktuře – kolik slov chci maximálně doporučovat a jakým řetězcem reprezentuji jokera.

8.2 Implementace

Pro implementaci jsem použil S4 class system, což je jeden z více class systems jazyka R. Mezi další class systems patří: S3, Reference system (někdy nazývaný R5) a R6, který není interní součástí R a je to samostatná knihovna[4]. Proč jsem si vybral zrovna S4 rozebírám v kapitole 9 Třídění více NGramNodes.

Deklarace datové struktury vypadá takto:

```
setClass("NGramBase", slots = list(children = "list", trie =
"externalptr", Highest = "integer"))

setClass("NGramNode", slots = list(freq = "numeric", name =
"character"), contains = "NGramBase")

setClass("NGramRoot", slots = list(maxResult = "integer", joker =
"character"), contains = "NGramBase")
```

[4] https://CRAN.R-project.org/package=R6

Třída „NgramNode“ reprezentuje buňku stromu a „NGramRoot“ reprezentuje kořen stromu. „NGramNode“ a „NGramRoot“ dědí od abstraktní třídy „NGramBase“.

„children“ reprezentuje list se všemi dětmi, „trie“ je písmenkový strom, který podle náležejícího slova ukládá děti daného objektu, a to jako index v listu „children“. „Highest“ je integer vector s indexy na objekty s nejvyšší frekvencí z listu „children“.

„freq“ reprezentuje pravděpodobnost slova a „name“ reprezentuje slovo samotné.

„maxResult“ reprezentuje kolik slov na klávesnici doporučuji – tedy jaká je délka vektoru „Highest“ u každé buňky/kořene. „joker“ ukládá to, jakým řetězcem je reprezentovaný joker.

Jako implementaci trií používám knihovnu triebeard[5].

8.3 Převádění data.frame na NGramTree

Data.frame na NGramTree převádím tak, že nejdříve vezmu NGramRoot – k němu pak jako jeho děti napojím všechny 1-gramy. Pak vezmu všechny 2-gramy a najdu příslušné NGramNode vzniklé z 1-gramů a k nim napojím konečná slova 2-gramů. Toto opakuji pro 3 až 5-gramy. Přitom se mi nemůže stát, že by předešlý n-gram, na který další (n+1)-gram napojuji neexistoval, protože do původní data.frame jsem zanesl všechna 1-gramy, co byli v textu. 2 až 5-gramy jsem použil s výskytem větší nebo rovným 2. V každém n-gramu si průběžně ukládám jeho pravděpodobnost a vytvářím trii a list Highest.

8.4 Algoritmus vyhledávání v NGramTree

Algoritmus pro nalezení slov s největší pravděpodobností bere na vstupu posloupnost slov a část dalšího nedopsaného slova. Posloupnost slov a část slova mohou být prázdné.

Implementoval jsem čtyři funkce pro vyhledávání:

- GetByNothing – pro případ, kdy na vstupu není žádné slovo ani část slova
- GetByPart – pro případ, kdy je na vstupu jenom část slova
- GetBySeq – pro případ, kdy jsou na vstupu nějaká slova, ale na konci není část slova
- GetBySeqAndPart – pro případ, kdy na vstupu jsou celá slova a na konci je část slova

8.4.1 GetByNothing

Funkce GetByNothing vrátí slova odpovídající listu „Highest“ u kořene stromu

```
GetByNothing <- function(root){

  toSort <- root@children[root@Highest]

  return(sapply(unname(toSort), function(x) x@name))
}
```

Funkce na vstupu bere kořen datového stromu, který chci použít. Potom vezme list „Highest“, který ukládá pozice, na kterých se nacházejí děti s nejvyšší frekvencí a použiji je jako indexy do list „children“, což mi vrátí list s „NGramNode“ objekty. Pak už jenom udělám list se slovy, které odpovídají těmto objektům a vrátím ho.

[5] https://CRAN.R-project.org/package=triebeard

8.4.2 GetByPart

Funkce „GetByPart“ funguje obdobně, akorát místo jednoho daného listu indexů budu mít větší list indexů, který získám jako výsledek hledání v trii. Slova, která dostanu z těchto indexů pak budu muset seřadit podle pravděpodobnosti.

```
GetByPart <- function(root, newPart){

  toSort <- unlist(root@children[prefix_match(root@trie,
newPart)[[1]]])

  return(sapply(SortNGramTree(toSort, root@maxResult), function(x)
x@name))
}
```

Nejdříve provedu hledání v trii, jako výsledek dostanu list objektů „NGramNode“. Potom je seřadím podle frekvence pomocí funkce „SortNGramTree“, kterou popisuji v kapitole 9 Třídění více NGramNodes. Nakonec vrátím slova příslušející objektům s nejvyšší frekvencí.

8.4.3 GetBySeq

Funkce GetBySeq bere na vstupu posloupnost slov, a funguje tak, že nejdříve použije funkci ChangeToJokers, aby to nahradilo málo používaná slova za jokera. Potom vezme poslední čtyři slova a hledá podle nich výrazy s nejvyšší pravděpodobností, která následují po posloupnosti těchto čtyřech slov. Bere tedy slova z listu „Highest“ u odpovídající „NGramNode“, ke které se dostane pomocí procházení stromu, kde další děti NGramNode hledá pomocí vyhledávání v trii. Pak toto samé opakuje i pro poslední tři, dvě a jedno slova. Nakonec ze všech nalezených slov najde ty s největší pravděpodobností a doporučí je na výstupu.

8.4.4 GetBySeqAndPart

Funkce GetBySeqAndPart bere na vstupu posloupnost slov a část slova. Funguje podobně jako GetBySeq akorát že místo slov z listu „Highest“ u odpovídající „NGramNode“ bere slova, která dostane z trie po zadání části slova, kterou jsem dostal na vstupu.

Pokud mi na výstupu nevyjde dostatek slov, tak to doplním slovy s nejvyšší pravděpodobností, které dostanu po vyhledávání vstupní části slova v trii u kořene stromu.

8.4.5 Časová náročnost

Časová náročnost funkce GetByNothing je konstantní, protože vracím už předem vypočítaný list.

Ve funkci GetByPart nejdříve hledám v trii, což má složitost O(n) v závislosti na velikosti vstupní části slova. Potom musím najít x slov s nejvyšší pravděpodobností. Toho jsem docílil s náročností O(n) a popisuji to v kapitole 9 Třídění více NGramNodes.

Ve funkcích GetBySeq a GetBySeqAndPart se pohybuji na stromu, kde každý pohyb provádím pomocí hledání v trii u dané NGramNode. Složitost je lineární v závislosti na délce daného slova. V těchto metodách potom dostanu list s nalezenými slovy. Buď konstantně, kdy vracím už předem vypočítaný list, a to ve funkci GetBySeq, anebo lineárně s délkou vstupní části slova ve funkci GetBySeqAndPart. Nakonec potřebuji získat slova s nejvyššími pravděpodobnostmi, což dokážu se složitostí O(n) v závislosti na počtu slov.

Třídění nalezených slov je ve většině případů více náročná operace než hledání v trii, protože složitost hledání v trii je lineární s délkou hledaného slova a já v trii hledám jenom vstupní anglická slova, která budou mít většinou kolem 5 znaků. Kdežto když třídím slova podle pravděpodobnosti, tak mohu dostat až několik desítek až stovek slov, v závislosti na tom, jak moc se dané slovo či daný prefix

vyskytuje v NGramTree nebo příslušné trii. To je závislé na tom, jak moc se dané slovo či prefix v angličtině využívá a tedy slova, která budou na vstupu hodně často budou mít také větší časovou náročnost.

9 Třídění více NGramNodes

Protože seřazování je časově nejnáročnější operace, kterou provádím při vyhledávání dalšího slova pro doporučení na prediktivní klávesnici, tak jsem se rozhodl najít nejrychlejší způsob a class system pro implementaci NGramTree, a to pomocí testování efektivnosti různých způsobů.

Toto jsou class systems, které jsem testoval:

- Listy
- S3
- S4
- Reference system (R5)
- R6

Pro třídění testuji dva algoritmy – jeden, který funguje na základě několika funkcí v R a popisuji ho v kapitole 9.3 Implementace a druhý, který jsem sám navrhnul a zoptimalizoval. Popisuji ho v kapitole 9.2 Popis vlastního algoritmu a implementuji ho v R a v C++. Testuji tedy celkem tyto tři možnosti:

- Algoritmus používající několik funkcí z jazyka R
- Implementace vlastního algoritmu v jazyce R
- Implementace vlastního algoritmu v C++ a propojení s R pomocí Rcpp

9.1 Popis class systems

9.1.1 Listy

Listy v R mohou obsahovat jakýkoliv objekt – data.frame, vektor anebo jiný list. Ke každému prvku se pak dá přiřadit jméno, podle kterého pak můžu daný prvek v listu najít. Pomocí tohoto pojmenování pak můžu listy používat jako třídy. Vyhledávání v listu podle jmen pak funguje tak, že se prochází všechna jména a výsledkem bude to, které nejvíce odpovídá. (12) Můžu proto v listu hledat jména, která tam nejsou, a přesto dostanu výsledek. Provádí se to tedy se složitostí O(n).

9.1.2 S3

Objekty S3 jsou objekty, které mají atributy, které se volají pomocí operátoru „$“. Definují se pomocí přiřazení řetězce do atributy *class*, který definuje třídu objektu. Tento atribut potom pomáhá generickým funkcím určit, jakou metodu mají spustit pro tento objekt. Metody tříd S3 fungují tak, že třída S3 je zadána do argumentů generické funkce, a tato funkce podle třídy objektu určují, jaká metoda se má spustit. Například funkce *print* je generická, takže pokud napíšu funkci *print(f)*, kde *f* je objekt s S3 třídou *factor*, tak se spustí funkce *print.factor(f)*. (13)

9.1.3 S4

Třídy S4 se definují pomocí funkce *setClass*. Instance třídy se pak vytváří pomocí funkce *new*. S4 mají atributy neboli sloty, které se volají pomocí operátoru „@“. Metody tříd S4 fungují podobně jako u S3. (14)

9.1.4 Reference system – R5

R5 třídy se definují pomocí funkce *setRefClass* a instance se vytváří pomocí metody *new* této třídy. R5 třídy fungují podobně jako třídy v Javě nebo C#, metody se volají na daný objekt. Metody a atributy se zpřístupňují pomocí operátoru „$“. R5 class system je implementovaný pomocí S4 tříd. (15)

9.1.5 R6

R6 class system není součástí základního R, ale je součástí knihovny R6. Definují se pomocí funkce *R6Class*. Instance se vytvářejí pomocí metody *new*. R6 metody fungují podobně jako R5 metody, používá se také operátor „$“. R6 je implementovaný pomocí S3 tříd. (16)

9.2 Popis vlastního algoritmu

Požadavky mého algoritmu jsou, že potřebuji získat seřazený list s konstantním počtem „NGramNode“ s nejvyššími pravděpodobnostmi. Toho s asymptotickou časovou složití O(n) docílím tak, že budu projíždět celý list a budu si pamatovat seřazený list s prvky s nejvyššími pravděpodobnostmi. Vždycky když přijedu na nový prvek, tak ho vezmu a budu ho porovnávat, jestli není větší než poslední prvek listu s nejvyššími pravděpodobnostmi. Pokud není větší, tak přejdu na další prvek listu, pokud je větší, tak ho budu porovnávat s předposledním prvkem listu s nejvyššími pravděpodobnostmi. Po porovnání s předposledním prvkem v případě, že prvek není větší, tak upravím list s nejvyššími pravděpodobnostmi odpovídajícím způsobem, pokud je tak pokračuji nahoru v listu s nejvyššími pravděpodobnostmi. Takto pokračuji nahoru až dokud případně nezjistím, že prvek je větší než nejvyšší prvek listu s nejvyššími pravděpodobnostmi, v tom případě upravím list tak, že nahoru dám porovnávaný prvek a ostatní prvky posunu o jednu pozici dolů.

9.3 Implementace

9.3.1 Class systems

Nejdřív jsem si vytvořil funkci, která vytváří dané třídy, které testuji. Zde je příklad pro S4:

```
CreateS4Nodes <- function(count, seed, maxValue){

  set.seed(seed)
  setClass("S4Node", representation(freq = "integer"))

  lapply(1:count, function(x){
    x <- new("S4Node", freq = as.integer(floor(runif(1, 1, maxValue +
1))))
    return(x)
  })
}
```

V argumentech je *count*, tedy počet, kolik chci vytvořit instancí dané třídy, potom *seed*, podle kterého náhodně generuji pravděpodobnost (*freq*) této třídy (hodnota, podle které chci tuto třídu seřazovat) a *maxValue*, tedy maximální hodnota *freq*.

9.3.2 Způsoby seřazování

První jsem naimplementoval algoritmus používající interní funkce R, pro všechny class systems jsem napsal jinou funkci. Zde je příklad pro seřazování tříd S4:

```
GetTop5S4_Order <- function(S4Nodes){

  S4Nodes[order(sapply(S4Nodes, function(x) x@freq), decreasing =
TRUE)][1:5]
}
```

Na vstupu bere list s S4 objekty. Funguje to tak, že to vezme pravděpodobnosti všech prvků, které potom funkce *order* sestupně seřadí a vrátí indexy seřazených prvků. Tyto indexy potom použiju do listu *S4Nodes*, od kterého vezmu 5 prvních prvků.

Potom jsem naimplementoval výše popsaný algoritmus [přidat odkaz] v R:

```
TopXSort <- function(toSort, parameterFunc, x = 5){
```

```
  topXList <- lapply(1:x, function(x) NA)
  topXListValues <- rep(-.Machine$integer.max, x)

  for (item in toSort){
    value <-  parameterFunc(item)

    if (value > topXListValues[[x]])

      for (i in (x-1):0){
        if( (i == 0) || (value <= topXListValues[[i]] ) ){
          if (x >= (i + 2))
            for (e in x:(i + 2)) {
              topXList[[e]] <-  topXList[[e-1]]
              topXListValues[[e]] <-  topXListValues[[e-1]]
            }
          topXList[[i+1]] <-  item
          topXListValues[[i+1]] <- value
          break
        }
      }
  }
  return(topXList)
}
```

toSort je list na seřazení, *x* je počet, kolik nejvyšších prvků chci najít. Aby to bylo univerzální pro všechny typy class systems, tak jsem tu zavedl do argumentů funkci *paramfunc*, která říká, jak dostat pravděpodobnost daného prvku.

Implementace například pro S4 vypadá takto:

```
GetTop5S4_TopXSort <- function(S4Nodes){

  TopXSort(S4Nodes, function(x) x@freq, 5)
}
```

Pak jsem to naimplementoval v C++:

```
template<typename T>
List TopXSortCpp(Rcpp::List items, std::function<int(T)> paramFunc, int
count){

  int topXListIndeces[count];
  int topXListValues[count];
  for (int i = 0; i < count; i++) topXListValues[i] = INT_MIN;

  for (int index = 0; index < items.length(); index++){
    T item = items[index];
    int value = paramFunc(item);

    if (value > topXListValues[count-1]){
      for ( int i = (count - 1); i >= 0; i--){
        if( (i == 0) || (value <= topXListValues[i-1] ) ){
          if (count >= (i + 2)){
            for (int e = count; e >= (i + 2); e--) {
              topXListIndeces[e-1] =  topXListIndeces[e-2];
```

```
              topXListValues[e-1] =  topXListValues[e-2];
            }
          }
          topXListIndeces[i] =  index;
          topXListValues[i] = value;
          break;
        }
      }
    }
  }
  List ret = List::create();
  for (int i = 0; i < count; i++)
ret.push_back(items[topXListIndeces[i]]);
  return ret;
}
```

Podobně jako ve verzi v R, tak tu jsou 3 argumenty: *items* – tedy list, co chci seřadit. Pomocí Rcpp můžu v C++ používat List z jazyka R. Potom paramFunc, tedy funkce, kterou používám, abych zjistil pravděpodobnost prvku a nakonec *count* – tedy kolik nejvyšších prvků chci najít.

Implementace pro S4 vypadá takto:

```
// [[Rcpp::export]]
List TopXSortCppS4(List items, int count){

  return TopXSortCpp<S4>(items, [](S4 node)->int{ return (int)
node.slot("freq");}, count);
}
```

Propojení s R funguje pomocí Rcpp, kdy nad funkci napíšu „// [Rcpp::export]“

V R jsem udělal funkci, která říká, kolik nejvyšších prvků chci dostat.

```
GetTop5S4_TopXSortRcpp <- function(S4Nodes){

  TopXSortCppS4(S4Nodes, 5)
}
```

9.4 Jak jsem měřil

Udělal jsem dva testy, v prvním jsem testoval všechny popsané metody a u každé jsem provedl 10 samostatných měření, při každém jsem spustil danou metodu 1000 krát. Vyhledával jsem 5 prvků s nejvyšší pravděpodobností v náhodném listu 100 prvků. Při každém měření jsem vygeneroval jiný náhodný list tak, aby všechny metody v rámci jednoho měření měli stejný list.

Při druhém testu jsem testoval jenom metody, které používají Rcpp a u každé jsem udělal 100 měření. Všechny podmínky tohoto testu jsem zanechal stejné jako u prvního testu.

Pro měření jsem používal knihovnu rbenchmark a pro zaznamenávání do grafů jsem použil knihovnu ggplot2.

9.5 Označení měřených metod

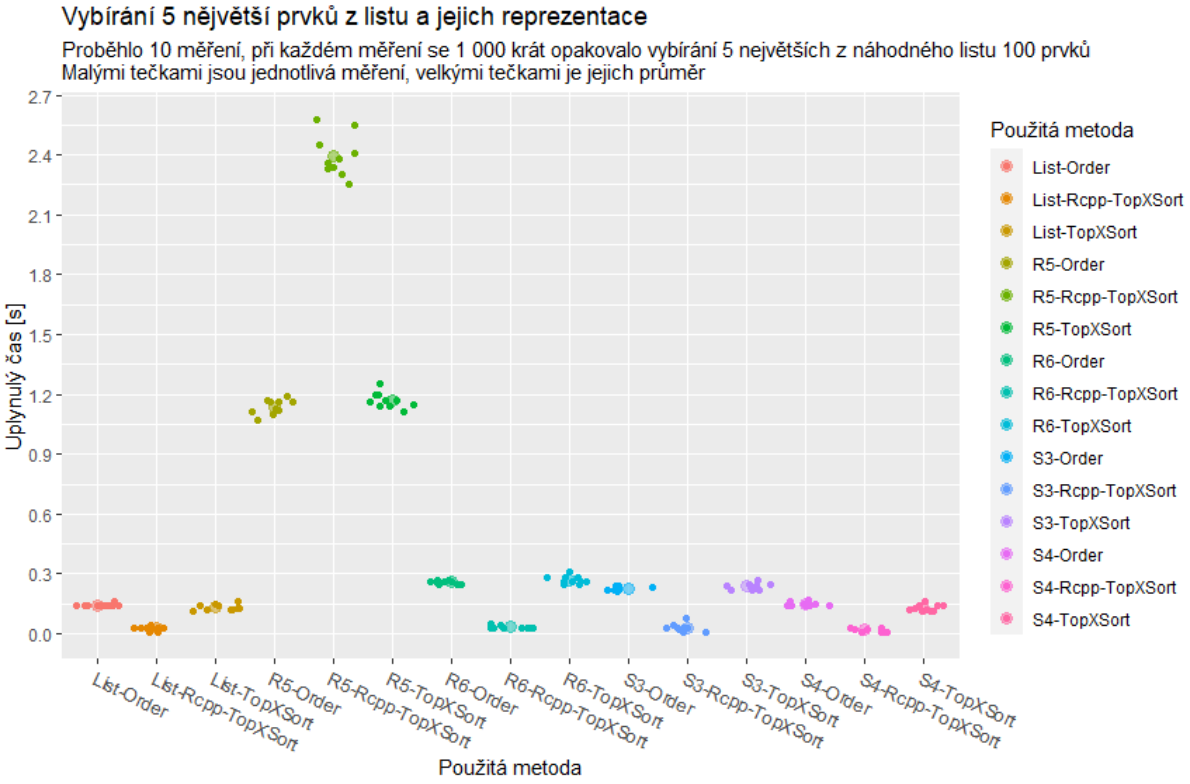
V grafech používám označení jednotlivých měření podle toho, jaký class system používám a podle použité funkce pro třídění, označení je následovné:

- Order = Algoritmus používající několik funkcí z jazyka R

- TopXSort = Implementace vlastního algoritmu v jazyce R
- Rcpp-TopXSort = Implementace vlastního algoritmu v C++ a propojení s R pomocí Rcpp

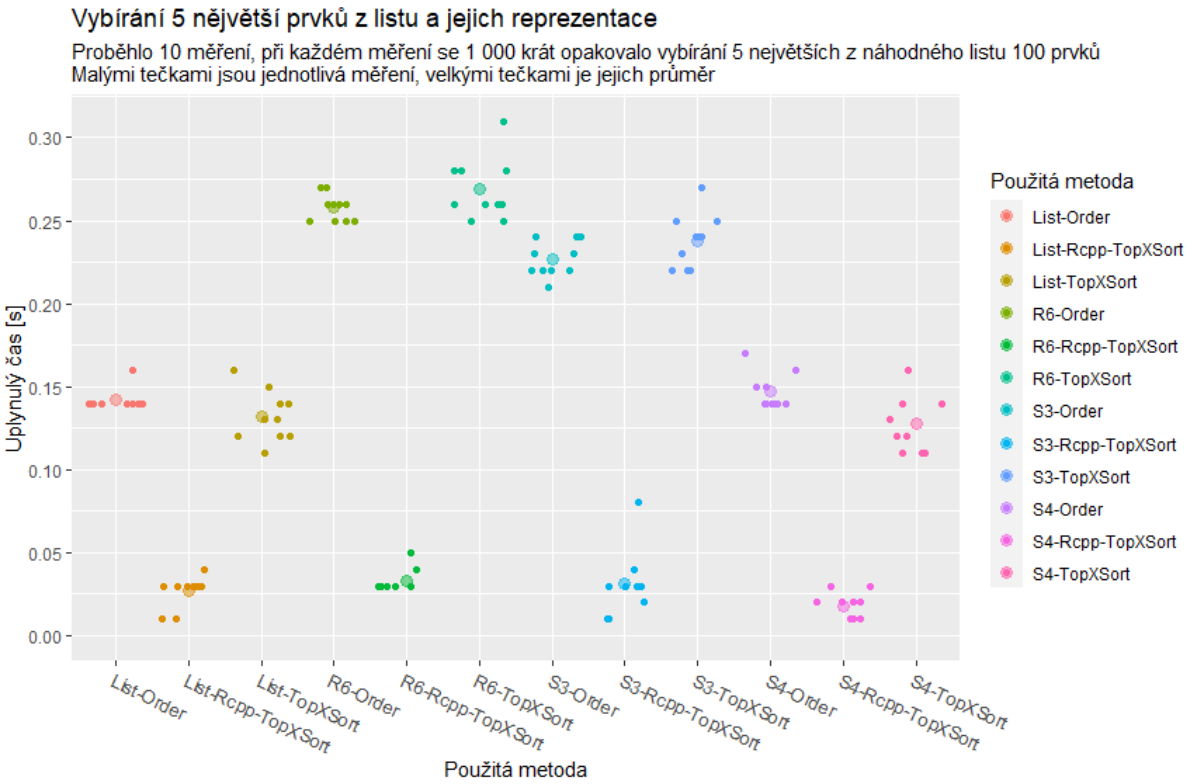
9.6 Výsledky 1. měření

Toto jsou celkové výsledky z 1. měření zanesené do grafu:



Obrázek 6: Graf se všemi záznamy v rámci měření 1

Na grafu je vidět, že použití R5 je značně neefektivní a také kvůli tomu nejsou vidět rozdíly mezi ostatními metodami. Udělal jsem proto druhý graf, kde jsem to vyřadil:

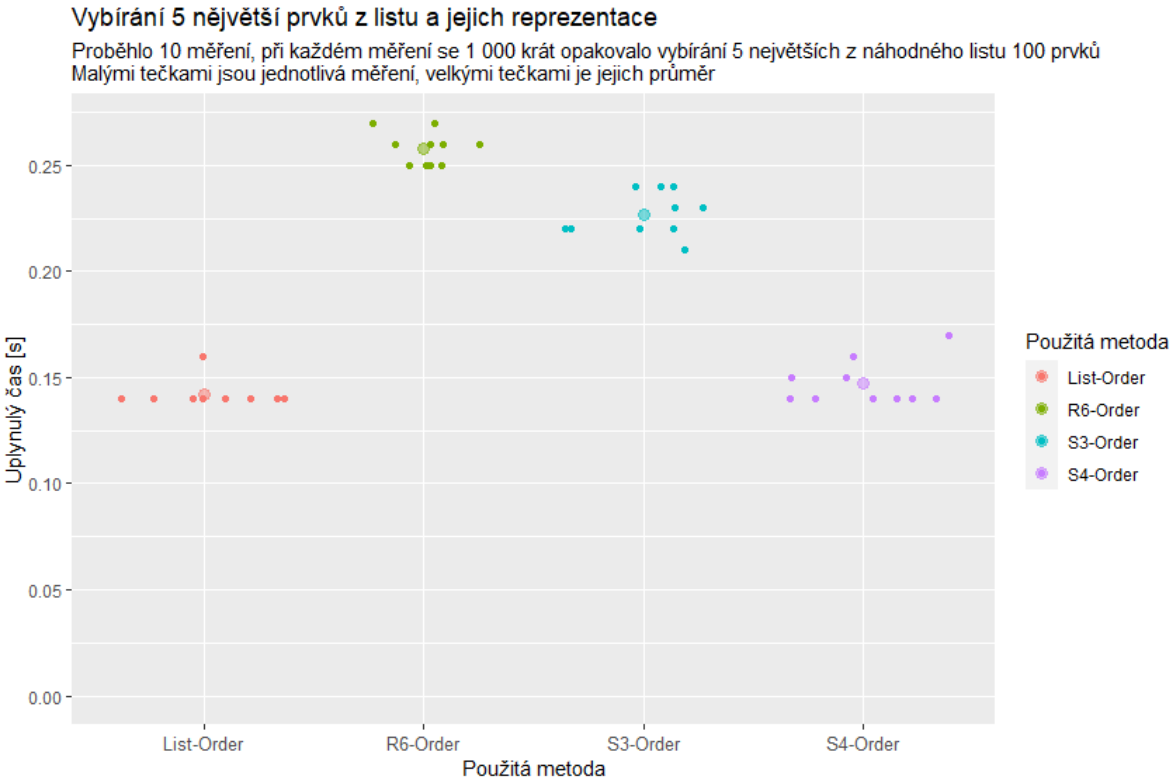


Obrázek 7: Graf se všemi záznamy kromě R5 v rámci měření 1

Z grafu jde poznat, že Rcpp je nejrychlejší způsob, což dává smysl, protože je to přímá implementace v C++.

9.6.1 Metoda order

Zde je graf zobrazující použití metody order.

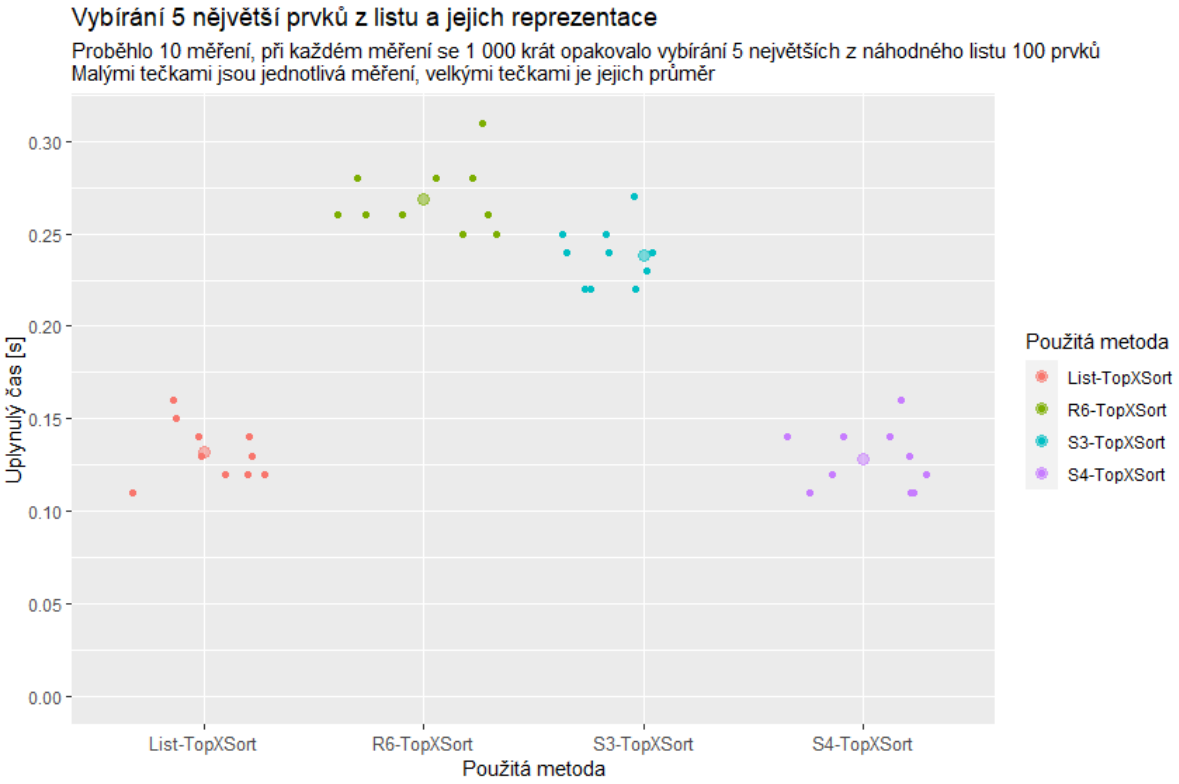


Obrázek 8: Graf se záznamy metody Order v rámci měření 1

Je zde vidět, že použití S4 a List je efektivnější než používání S3 a R6.

9.6.2 Metoda TopXSort

Vyrendroval jsem graf zobrazující použití metody TopXSort.

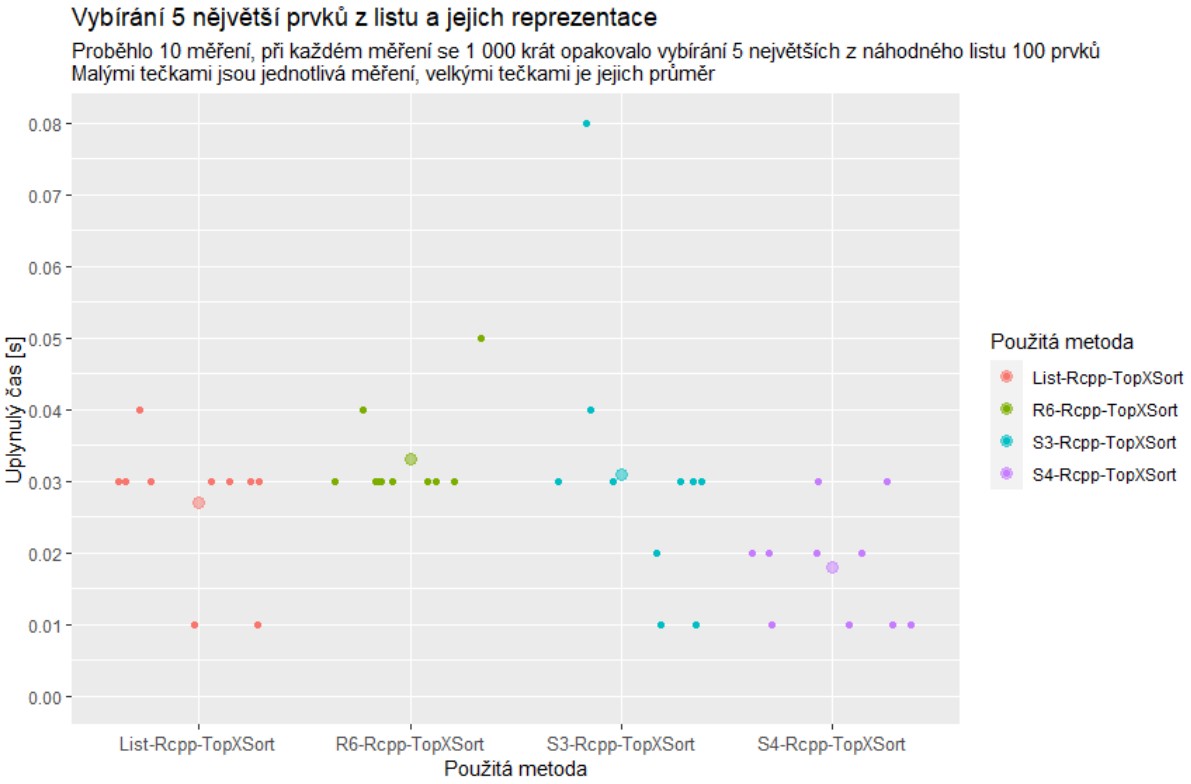


Obrázek 9: Graf se záznamy metody TopXSort v rámci měření 1

Je zde vidět, že S4 a List jsou značně efektivnější než R6 a S3.

9.6.3 Metoda Rcpp-TopXSort

Teď rozeberu výsledky měření metody Rcpp-TopXSort

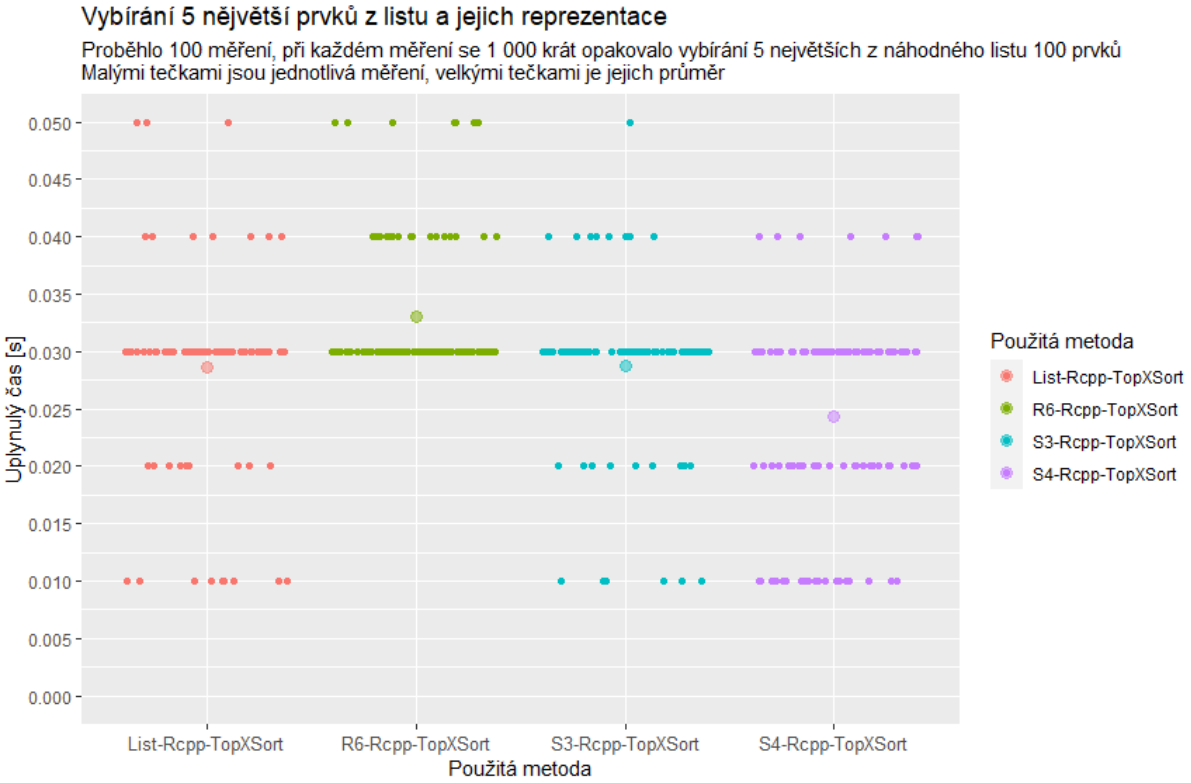


Obrázek 10: Graf se záznamy metody Rcpp-TopXSort v rámci měření 1

Tento graf mi připadá dost nepřesný a nevypovídající, protože jednotlivá měření mají mezi sebou až moc velké rozdíly, a proto se mi zdá, že by mohlo jít o chybu měření, která by v tomto případě ovlivnila výsledky až moc. Proto jsem udělal 2. měření, kde jsem toto zopakoval s větším počtem měření.

9.7 Výsledky 2. měření

Ve druhém měření jsme měřil použití metody Rcpp-TopXSort:



Obrázek 11: Graf měření 2

Z grafu je vidět, že S4 je o něco efektivnější než ostatní metody, zároveň R6 je o něco méně efektivnější.

9.8 Zhodnocení měření

Vybral jsem si S4 class system, protože byl ve všech metodách nejrychlejší a algoritmus naprogramovaný v Rcpp, protože byl rychlejší než ostatní. Do algoritmu v Rcpp jsem doplnil funkci, která vrací pravděpodobnost funkce.

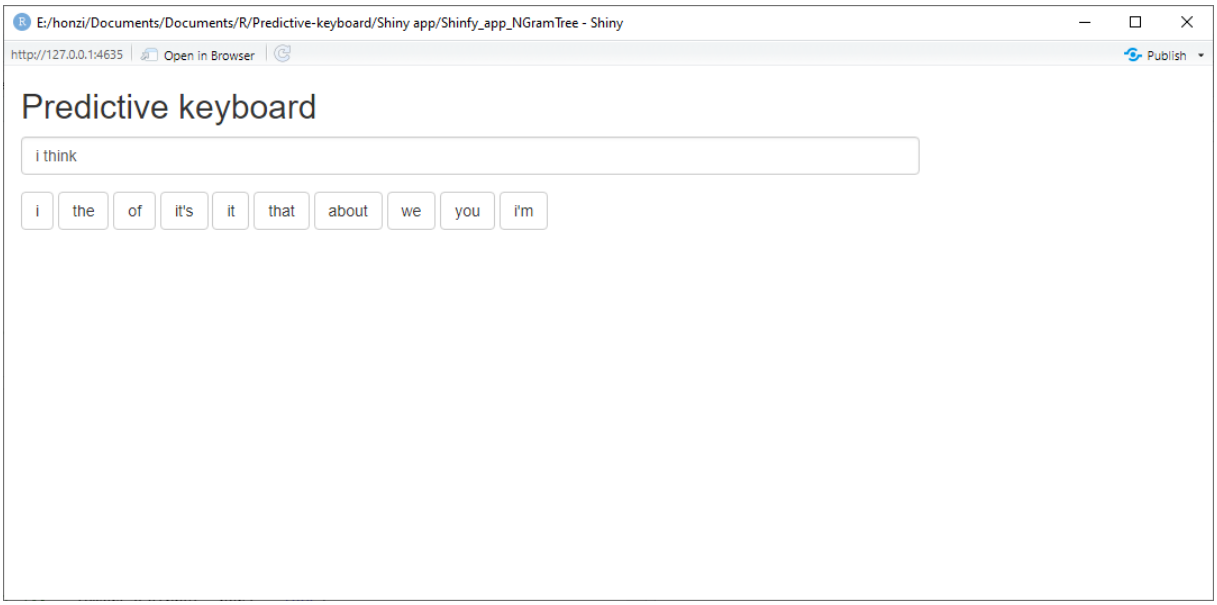
```
// [[Rcpp::export]]
List SortNGramTree(Rcpp::List items, int itemsCount){

  int count = std::min((int) itemsCount, (int) items.length());

  int topXListIndeces[count];
  double topXListValues[count];
  for (int i = 0; i < count; i++) topXListValues[i] = DBL_MIN;

  for (int index = 0; index < items.length(); index++){
    S4 item = items[index];
    double value = item.slot("freq");

    if (value > topXListValues[count-1]){
      for ( int i = (count - 1); i >= 0; i--){
        if( (i == 0) || (value <= topXListValues[i-1] ) ){
          if (count >= (i + 2)){
```

```
            for (int e = count; e >= (i + 2); e--) {
              topXListIndeces[e-1] =  topXListIndeces[e-2];
              topXListValues[e-1] =  topXListValues[e-2];
            }
          }
          topXListIndeces[i] =  index;
          topXListValues[i] = value;
          break;
        }
      }
    }
  }
  List ret = List::create();
  for (int i = 0; i < count; i++)
ret.push_back(items[topXListIndeces[i]]);
  return ret;
}
```

10 Shiny App

Pro interakci s uživatelem jsem napsal aplikaci pomocí knihovny Shiny. Aplikace bere na vstupním zadávacím poli text a jako výstup zobrazuje 10 nabízených slov. Pokud uživatel klikne na nabízené slovo, tak aplikace na vstup napíše nebo dokončí toto slovo a napíše za ním mezeru a potom obnoví výstupová slova. Vždy, když uživatel napíše něco na vstupu, tak se obnoví nabídka slov.

Aplikace vezme text na vstupu, který potom rozdělí na jednotlivá slova. Jako jedno slovo považuji posloupnost znaků, která v sobě nemá mezery. Potom zkontroluje, jestli někde nebyla ukončená věta – jestli ano, tak nechá jenom slova, která jsou za ukončenou větou.



Obrázek 12: Ukázka implementace UI prediktivní klávesnice v Shiny

Aplikaci pro 1,5 MB jsem nahrál na server shinyapps.io, takže je teď přístupná na následujícím odkazu: https://jan-sliva.shinyapps.io/Predictive_keyboard_1-5MB/ . Data v aplikaci ukládám jako tabulky Csv, jejichž načtení do NGramTree trvá asi 30 vteřin.

Verzi pro15 MB se mi nepodařilo nahrát na web, protože tam bylo příliš mnoho dat, jejichž načtení do NGramTree by trvalo až moc dlouho. Limit na webu shinyapps.io je 1 minuta.

11 Závěr

Prediktivní klávesnice vytvořená z 1,5 MB funguje na webu shinyapps.io. Vytvořil jsem také verzi z 15 MB, kterou se mi nepodařilo nahrát na internet.

Vývoj aplikace byl bez větších komplikací, díky použití vhodných technologií. Pracování s knihovnami R bylo jednoduché, protože jsem měl vždy k dispozici dokumentaci, která dané funkce či jiné téma popisovala stručně a jasně.

Zaměřil jsem se spíše na návrh, testování a implementaci datové struktury než na statistickou optimalizaci dat, které jsem použil.

Prediktivní klávesnice pro 1,5 MB je dostupná na následujícím odkazu:
https://jan-sliva.shinyapps.io/Predictive_keyboard_1-5MB/

Všechny zdrojové kódy jsou dostupné na: https://github.com/Jan-Sliva/Predictive-keyboard

12 Citovaná literatura

1. **R (programming language). *Wikipedia.* [Online] 2021. [Citace: 19. října 2021.] https://en.wikipedia.org/wiki/R_(programming_language).**

2. C++. *Wikipedia.* [Online] 2021. [Citace: 23. října 2021.] https://en.wikipedia.org/wiki/C%2B%2B.

3. R package. *Wikipedia.* [Online] 2021. [Citace: 20. října 2021.] https://en.wikipedia.org/wiki/R_package.

4. quanteda: Quantitative Analysis of Textual Data. *quanteda.* [Online] 2021. [Citace: 20. října 2021.] https://quanteda.io/.

5. ggplot2. *Wikipedia.* [Online] 2021. [Citace: 22. října 2021.] https://en.wikipedia.org/wiki/Ggplot2.

6. *ggplot2.* [Online] 2021. [Citace: 22. října 2021.] https://ggplot2.tidyverse.org/.

7. rbenchmark: Benchmarking routine for R. *CRAN.* [Online] 2021. [Citace: 22. října 2021.] https://CRAN.R-project.org/package=rbenchmark.

8. Rcpp: Seamless R and C++ Integration. *CRAN.* [Online] 2021. [Citace: 23. října 2021.] https://CRAN.R-project.org/package=Rcpp.

9. Rcpp: Seamless R and C++ Integration. *Dirk Eddelbuettel.* [Online] 30. června 2020. [Citace: 23. října 2021.] http://dirk.eddelbuettel.com/code/rcpp.html.

10. *Shiny.* [Online] 2020. [Citace: 23. října 2021.] https://shiny.rstudio.com/.

11. Tokenization. *nlp.stanford.* [Online] 7. dubna 2009. [Citace: 23. října 2021.] https://nlp.stanford.edu/IR-book/html/htmledition/tokenization-1.html.

12. Reinhart, Alex. R's Lists and its Detestable Dearth of Data-Structures. *refsmmat.* [Online] 12. září 2016. [Citace: 6. listopadu 2021.] https://www.refsmmat.com/posts/2016-09-12-r-lists.html.

13. Wickham, Hadley. S3. *Advanced R.* [Online] 23. října 2019. [Citace: 6. listopadu 2021.] https://adv-r.hadley.nz/s3.html.

14. —. S4. *Advanced R.* [Online] 8. ledna 2021. [Citace: 6. listopadu 2021.] https://adv-r.hadley.nz/s4.html.

15. —. Reference classes. *Advanced R.* [Online] 2021. [Citace: 6. listopadu 2021.] http://adv-r.had.co.nz/R5.html.

16. —. R6. *Advanced R.* [Online] 8. ledna 2021. [Citace: 6. listopadu 2021.] https://adv-r.hadley.nz/r6.html#r6-classes.

13 Citace použitého korpusu

English (US). *HC Corpora.* [Online] květen 2010. [Citace: 9. 11 2021.]
http://corpora.epizy.com/corpora.html.

14 Seznam Obrázků

15 Použitý software

R Core Team (2021). R: A language and environment for statistical computing. R Foundation for Statistical Computing, Vienna, Austria. URL https://www.R-project.org/.

Benoit K, Watanabe K, Wang H, Nulty P, Obeng A, Müller S, Matsuo A (2018). "quanteda: An R package for the quantitative analysis of textual data." Journal of Open Source Software, 3(30), 774. doi: 10.21105/joss.00774, https://quanteda.io.

H. Wickham. ggplot2: Elegant Graphics for Data Analysis. Springer-Verlag New York, 2016.

Wacek Kusnierczyk (2012). rbenchmark: Benchmarking routine for R. R package version 1.0.0. https://CRAN.R-project.org/package=rbenchmark

Dirk Eddelbuettel and Romain Francois (2011). Rcpp: Seamless R and C++ Integration. Journal of Statistical Software, 40(8), 1-18. URL https://www.jstatsoft.org/v40/i08/.

Winston Chang, Joe Cheng, JJ Allaire, Carson Sievert, Barret Schloerke, Yihui Xie, Jeff Allen, Jonathan McPherson, Alan Dipert and Barbara Borges (2021). shiny: Web Application Framework for R. R package version 1.6.0. https://CRAN.R-project.org/package=shiny

Dean Attali (2020). shinyjs: Easily Improve the User Experience of Your Shiny Apps in Seconds. R package version 2.0.0. https://CRAN.R-project.org/package=shinyjs

Hadley Wickham and Jim Hester (2020). readr: Read Rectangular Text Data. R package version 1.4.0. https://CRAN.R-project.org/package=readr

Oliver Keyes, Drew Schmidt and Yuuki Takano (2016). triebeard: 'Radix' Trees in 'Rcpp'. R package version 0.3.0. https://CRAN.R-project.org/package=triebeard

Winston Chang (2020). R6: Encapsulated Classes with Reference Semantics. R package version 2.5.0. https://CRAN.R-project.org/package=R6

Hadley Wickham (2019). stringr: Simple, Consistent Wrappers for Common String Operations. R package version 1.4.0. https://CRAN.R-project.org/package=stringr

Hadley Wickham, Romain François, Lionel Henry and Kirill Müller (2021). dplyr: A Grammar of Data Manipulation. R package version 1.0.7. https://CRAN.R-project.org/package=dplyr

Lionel Henry and Hadley Wickham (2020). purrr: Functional Programming Tools. R package version 0.3.4. https://CRAN.R-project.org/package=purrr

Hadley Wickham and Dana Seidel (2020). scales: Scale Functions for Visualization. R package version 1.1.1. https://CRAN.R-project.org/package=scales

SO/IEC. (2014). ISO International Standard ISO/IEC 14882:2014(E) – Programming Language C++. Geneva, Switzerland: International Organization for Standardization (ISO). Retrieved from https://isocpp.org/std/the-standard