PROHLÁŠENÍ

Předkládám tímto k posouzení a obhajobě diplomovou práci zpracovanou na závěr studia na Fakultě aplikovaných věd Západočeské univerzity v Plzni.

Prohlašuji, že jsem diplomovou práci vypracoval samostatně a výhradně s použitím odborné literatury a pramenů, jejichž úplný seznam je její součástí.

V Plzni dne Petr Salajka

Chtěl bych poděkovat především vedoucímu své diplomové práce Ing. Luboši Šmídlovi, Ph.D. a Ing. Janu Švecovi z Katedry kybernetiky Fakulty aplikovaných věd Západočeské univerzity v Plzni, jejichž cenné rady, ale i úkoly, které mi zadávali, mne dovedli až k závěru magisterského studia.

Děkuji samozřejmě i své rodině a také přátelům, bez jejichž pomoci, podpory a pochopení bych se jistě až sem nedostal.

Abstrakt

Diplomová práce zkoumá možnosti jazykového modelování, metody statistické analýzy a modelování jazyka obecně jako prostředku pro prediktivní psaní textu. Jejím cílem je vytvořit aplikaci navrhující vždy další možná slova tak, aby mohla být žádaná věta zadána co nejrychleji a co nejefektivněji a následně převedena na řeč. Díky možnosti provozu aplikace na lehkých mobilních zařízeních by se mohla stát užitečnou pomůckou pro lidi s řečovým handicapem.

Klíčová slova: jazykové modelování, n-gramové modely, prediktivní psaní textu, vážené konečné automaty, WebSocket

Abstract

The thesis examines possibilities of language modeling, methods of statistical analysis and language modeling in general as a means for predictive text writing. Its aim is to create an application proposing words suitable for completing a typed sentence as quickly and efficiently as possible and convert it into speech. The application can be run on lightweight mobile devices and so could become a useful tool for people with speech disabilities.

Keywords: language modeling, n-gram models, predictive text input, weighted finite state automata, WebSocket

Obsah

Kapitola 1

Úvod

Spolu s neustálým vývojem mobilních telefonů a dalších přenosných zařízení (například tablety) vzniká také velké množství rozličných technologií, jejichž cílem je usnadnit uživatelům jejich používání. Mezi tyto technologii patří i řada aplikací umožňujících pohodlnější psaní na malých klávesnicích těchto zařízení; případně zcela bez mechanických kláves, kdy je klávesnice promítána na displeji přístroje.

Paralelně s tím se vědecká pracoviště snaží hledat způsoby, jak pomoci lidem s tělesným postižením překonávat jejich handicap. Takový výzkum probíhá i zde na Katedře kybernetiky (Fakulta Aplikovaných věd na Západočeské univerzitě v Plzni) [1, 2]. Jednou z oblastí, kterým se katedra věnuje, je i pomoc lidem, kteří trpí vadou, popřípadě nejsou vůbec schopni používat řeč. V takové situaci často nalézá uplatnění počítačová syntéza řeči.

Tato práce si klade za cíl spojit obě výše uvedené oblasti a vytvořit aplikaci provozovanou na mobilním zařízení, která by umožnila uživateli neschopnému řeči rychle a efektivně formulovat žádanou větu a tuto následně syntetizovat.

Vzhledem k omezenému výpočetnímu výkonu a prostoru pro ukládání dat na mobilních zařízeních se předpokládá forma aplikace klient–server.

Kapitola 2

Jazykové modelování

Jazykové modelování a jazykové modely nachází celou řadu uplatnění ať už v teoretických či praktických úlohách. Z teoretických můžeme jmenovat například zkoumání přirozeného (lidmi používaného) jazyka; obecně lingvistické úlohy. Mezi praktické můžeme zařadit automatické překladače nebo spellcheckery. Je dobré poznamenat, že český překlad spellcheckingu „kontrola pravopisu“ může být dosti zavádějící. Spellchecker kontroluje, zda jsou hlásky ve slově ve správném (jemu známém) sledu, zda se slovo jako takové v modelu vyskytuje a případně, zda je běžné, aby se slovo vyskytovalo v daném kontextu. Zřídkakdy však kontroluje například správnou shodu přísudku s podmětem, což jistě ke kontrole pravopisu patří.

Další praktickou aplikací jazykových modelů je automatické rozpoznávání řeči. Často zde nalézá uplatnění vztah

$$\hat{W} = \underset{W}{\operatorname{argmax}} P(O|W)P(W) \quad , \tag{2.0.1}$$

kde $\hat{W}$ je nejlepší odhad slova příslušejícího zaznamenané řeči, $P(O|W)$ je akustický model a konečně $P(W)$ je jazykový model, který vnáší do systému apriorní informaci o uvažovaných slovech [3]. Symbolem $P(A)$ obecně označujeme pravděpodobnostní ohodnocení jevu A.

Další úlohou, ve které figurují jazykové modely a se kterou se často setkáváme, jsou prediktory usnadňující psaní. Pravděpodobně nejstarší všeobecně rozšířenou technologií je *T9* (vyvinuta společností Tegic Communications).

Umožňovala snazší psaní textu na mobilních telefonech. Než byla převážně vytlačena novějšími technologiemi (na novějších telefonech), našla si celou řadu svých zastánců, ale i odpůrců. Důvod případného odporu lidí k této technologii měl převážně jediný důvod. Nevyhovoval jim použitý jazykový model, který sekvenci stisknutých kláves přiřazoval slova. Jednoduše řečeno, nabízel jim podle nich samé hlouposti. Volba vhodného jazykového modelu aplikace je ale stěžejní otázkou bez ohledu na dobu.

Další technologií, která je dnes na mobilních telefonech poměrně hojně rozšířena a ve které hraje jazykové modelování důležitou úlohu, je *Swype* (vyvinuta Swype Inc.) a její nadstavby. Umožňuje uživateli psát pouhým tahem prstu po klávesnici. Akustický model v tomto případě nahrazuje ve vztahu (2.0.1) model popisující pravděpodobnost dané trajektorie, pokud chtěl uživatel zadat slovo W.

Každý přirozený jazyk má své zákonitosti a k nim je třeba při volbě jazykového modelu přihlížet. Jazykový model se pak tyto zákonitosti snaží na určité úrovni modelovat. Nejhrubším přístupem je deterministický model, kdy máme jasně dané, jakým způsobem se mohou slova řetězit a připouštíme pouze slova, která model skutečně obsahuje. Příkladem může být gramatika popisující konkrétní formu zápisu dat. Píšeme „prvního ledna, čtvrtého května, patnáctého prosince“, ale v žádném případě nepřipouštíme např. řetězec „prvního závodníka“. I zde bychom mohli jednotlivým promluvám přiřadit pravděpodobnosti; nejspíše bychom ale dospěli k závěru, že jsou všechny promluvy stejně pravděpodobné.

Mnohem volnější je stochastický jazykový model (využívající pravděpodobnostní omezení). V tomto případě již předpokládáme, že jednotlivé promluvy nejsou stejně časté a snažíme se dané sekvenci slov přiřadit vhodnou pravděpodobnost. Předpis stochastického jazykového modelu uvedený na str. 227 v [3] vyhovuje i naší úloze, proto ho zde beze změny uvedeme.

$$\begin{gathered} P(W) = P\left(w_1^K\right) = P(w_1 w_2 w_3 \dots w_k) = \\ = P(w_1)\, P(w_2|w_1)\, P(w_3|w_1 w_2) \dots P(w_K|w_1 w_2 \dots w_{K-1}) = \\ = P(w_1)\, P(w_2|w_1^1)\, P(w_3|w_1^2) \dots P\left(w_K|w_1^{K-1}\right) = \\ = \prod_{i=1}^{K} P\left(w_i|w_1^{i-1}\right) \end{gathered} \tag{2.0.2}$$

Je možné formulovat i jiný rozklad pravděpodobnosti $P(W)$, ovšem tento má pro nás výhodu v tom, že pravděpodobnost slova je dána pouze svou historií. To se pro naši úlohu ukáže jako velmi vhodné.

2.1 Stochastické n-gramové modely

Formulovaný model (2.0.2) však stále trpí značným neduhem. Pro jeho konstrukci bychom potřebovali znát pravděpodobnost $P\left(w_1^K\right)$ všech možných posloupností slov. Tento předpoklad je však vzhledem ke množství potřebných dat zcela nesplnitelný. Proto provedeme určitou aproximaci modelu. Tato aproximace se obecně nazývá *n-gramový model* a provedeme ji tak, že posloupnost $P\left(w_1^K\right)$ nahradíme posloupností $P\left(w_{K-n}^K\right)$ [3]:

$$P\left(w_1^K\right) \approx \prod_{i=1}^{K} P\left(w_i | w_{i-(n-1)}^{i-1}\right) \quad . \tag{2.1.1}$$

Takový model bude tím lepší, čím pevnější je pořadí slov ve větách daného jazyka. Trénování (odhad pravděpodobností) takového modelu je založeno na zjišťování relativních četností sekvencí slov v trénovacích datech; tedy podle [3] platí pro odhad pravděpodobnosti trigramu

$$\bar{P}\left(w_K | w_{K-2} w_{K-1}\right) = \frac{N\left(w_{K-2} w_{K-1} w_K\right)}{N\left(w_{K-2} w_{K-1}\right)} \quad , \tag{2.1.2}$$

kde $N(\ldots)$ značí vypočtenou četnost sekvence tří respektive dvou slov. Symbol $\bar{P}(\ldots)$ upozorňuje na fakt, že se jedná o odhad pravděpodobnosti.

Mějme řetězec

$$a\,b\,a\,b\,b\,a$$

Pak jeho n-gramové modely budou následující

$$\begin{aligned} \bar{P}(a) &= \tfrac{N(a)}{N} = \tfrac{3}{6} = 0{,}5 \\ \bar{P}(b) &= \tfrac{N(b)}{N} = \tfrac{3}{6} = 0{,}5 \end{aligned} \tag{2.1.3}$$

$$\begin{aligned}\bar{P}(b|a) &= \tfrac{N(a\,b)}{N(a)} = \tfrac{2}{3} \doteq 0{,}67\\ \bar{P}(a|b) &= \tfrac{N(b\,a)}{N(b)} = \tfrac{2}{3} \doteq 0{,}67\\ \bar{P}(b|b) &= \tfrac{N(b\,b)}{N(b)} = \tfrac{1}{3} \doteq 0{,}33\end{aligned} \tag{2.1.4}$$

$$\begin{aligned}\bar{P}(a|<s>) &= \tfrac{N(<s>\,a)}{N(<s>)} = \tfrac{1}{1} = 1{,}00\\ \bar{P}(b|a) &= \tfrac{N(a\,b)}{N(a)} = \tfrac{2}{3} \doteq 0{,}67\\ \bar{P}(a|b) &= \tfrac{N(b\,a)}{N(b)} = \tfrac{2}{3} \doteq 0{,}67\\ \bar{P}(b|b) &= \tfrac{N(b\,b)}{N(b)} = \tfrac{1}{3} \doteq 0{,}33\\ \bar{P}(</s>|a) &= \tfrac{N(a\,</s>)}{N(a)} = \tfrac{1}{3} = 0{,}33\end{aligned} \tag{2.1.5}$$

V případě unigramového modelu (2.1.3) je situace celkem jasná. Jedná se o relativní četnosti každého slova v řetězci. V případě bigramového modelu (2.1.4) si však všimneme jisté nesrovnalosti. Suma bigramů s historií a se nerovná jedné; jinými slovy neplatí věta o úplné pravděpodobnosti. Takový model tedy zavrhneme. Pro sestavení modelů bigramem počínaje je totiž vhodné zahrnout do výpočtu i dodatečné symboly (slova) pro začátek ($<s>$) respektive konec ($</s>$) věty. Vhodný bigramový model řetězce tedy bude (2.1.5). Obdobně bychom mohli sestavit i modely vyšších řádů; i v nich by figurovaly symboly začátku a konce věty.

2.2 Vyhlazování

Ačkoliv je n-gramový model pouhou aproximací stochastického jazykového modelu (2.0.2), jeho hlavním neduhem jsou stále vysoké nároky na množství trénovacích dat. Uvažujme řetězec $a\,b\,a\,a\,b\,a$. Příliš se neliší od $a\,b\,a\,b\,b\,a$, který byl použit pro natrénování bigramového modelu (2.1.5). Vyskytuje se v něm však nepozorovaný bigram $a\,a$, což znamená, že takovému řetězci přiřadí náš model nulovou pravděpodobnost. Úpravu modelu, která zajistí, že bude každému řetězci přiřazena pravděpodobnost nenulová, nazýváme vyhlazování. Mezi známé metody vyhlazování patří Bayesova metoda s aditivním vyhlazováním, Goodův-Turingův odhad, metoda odhadu s postupným vynechávání jednoho jevu, Katzův diskontní model, model s absolutním diskontem, model s lineárním diskontem, Kneserův-Neyův model, lineární interpolace

nebo Wittenův-Bellův model [3]. Právě Wittenův-Bellův model je v práci použit, proto se mu budeme věnovat více.

2.2.1 Wittenův-Bellův model

Při odhadu pravděpodobností modelu metodou maximální věrohodnosti (obdoba relativních četností) je nepozorovaným jevům přiřazena nulová pravděpodobnost. Přerozdělení pravděpodobností tak, aby měl každý jev nenulovou pravděpodobnost může být provedeno pomocí tzv. *ústupového (backing-off) schématu*. Princip ústupového schématu spočívá v tom, že neznáme-li pravděpodobnost trigramu, „zapomeneme“ nejstarší slovo jeho historie; tedy použijeme na jeho místě bigram atd. Pravděpodobnosti pozorovaných n-gramů se odhadují na základě jejich relativní četnosti, která je ovšem snížena součinitelem $d_{N(h,w)}$, kde $N(h,w)$ je počet výskytů konkrétní posloupnosti historie h a slova w. Číslu $d_{N(h,w)}$ se říká *diskontní součinitel* [3]. Platí

$$P_{BO}(w|h) = \begin{cases} d_{N(h,w)} \frac{N(h,w)}{N(h)} & \text{pro } N(h,w) > 0 \\ B(h)\,\beta\left(w\left|\bar{h}\right.\right) & \text{pro } N(h,w) = 0 \end{cases} \tag{2.2.1}$$

$$B(h) = \frac{1 - \sum_{w:N(h,w)>0} d_{N(h,w)} \frac{N(h,w)}{N(h)}}{\sum_{w:N(h,w)=0} \beta\left(w\left|\bar{h}\right.\right)} \quad , \tag{2.2.2}$$

kde $B(h)$ se nazývá *ústupová váha*, $\bar{h}$ je zobecněná historie (historie h zkrácená o nejstarší slovo) a $\beta\left(w\left|\bar{h}\right.\right)$ je tedy zobecněné pravděpodobnostní rozdělení, které lze získat např. rekurzivním výpočtem $\beta\left(w\left|\bar{h}\right.\right) = P_{BO}\left(w|\bar{h}\right)$.

Poznamenejme, že taková úprava v žádném případě neřeší situaci, kdy je modelu k ocenění předloženo slovo, které nezná. Vyřešena je pouze situace, kdy chceme znát pravděpodobnost slova v kontextu, ve kterém se v trénovacích datech nevyskytovalo.

Wittenův-Bellův model používá diskontní součinitel ve tvaru

$$d_{N(w,h)} = d_h = \frac{N(h)}{N(h) + n(h)} \quad \text{pro } N(h,w) > 0 \quad , \tag{2.2.3}$$

kde $n(h)$ je počet různých slov následujících historii h [3].

Pro odhad pravděpodobností modelu tedy platí

$$P_{WB}\left(w|h\right) = \begin{cases} \frac{N(h,w)}{N(h)+n(h)} & \text{pro } N\left(h,w\right) > 0 \\ \frac{n(h)}{N(h)+n(h)} \frac{\beta\left(w|\bar{h}\right)}{\sum_{w':N\left(h,w'\right)=0} \beta\left(w|\bar{h}\right)} & \text{pro } N\left(h,w\right) = 0 \end{cases} .$$

Během experimentů s dalšími metodami vyhlazování jazykových modelů, jejichž použití knihovna OpenGrm NGram [4] umožňuje, nebylo zaznamenáno žádné zjevné zlepšení. Proto byl zvolen Wittenův-Bellův model, který je v knihovně nastaven jako výchozí.

2.3 Další varianty jazykových modelů

Ačkoliv n-gramové jazykové modely jsou v současné době vzhledem k jejich vlastnostem a možnostem jejich uplatnění nejčastější variantou jazykových modelů, existuje vedle nich celá řada dalších.

Modely založené na třídách slov přiřazují slova do tříd a následně se snaží modelovat vztahy mezi těmito třídami. Na základě metody použité pro přiřazení slov do tříd můžeme tyto modely dále dělit; například *shlukování podle slovních druhů, shlukování založené na lemmatech* nebo *automatické shlukování* [3].

Modely založené na spouštěčích (angl. *trigger models*) předpokládají, že vyskytlo-li se v proudu textu slovo A, je pravděpodobné, že se obecně s proměnným odstupem vyskytne i slovo B. Jeho pravděpodobnost v následujícím textu je tedy třeba adekvátně navýšit. Zásadním problémem těchto modelů je volba pravidel $A \rightarrow B$. Modely založené na spouštěčích jsou často kombinovány s n-gramovými modely metodou maximální entropie [3].

Častým jevem je, že výskyt slova v řeči respektive textu implikuje zvýšenou pravděpodobnost jeho opakování. Jedná se tedy v určitém ohledu o variantu modelu založeného na spouštěčích s pravidly typu $A \rightarrow A$. Takový model obvykle nazýváme *model s krátkodobou pamětí* [3].

Další způsob, jak zlepšit vlastnosti modelu, je jeho *adaptace na doménu*. Jsme-li schopni rozpoznat, jakého tématu se text týká, můžeme modelu dané domény přidat na váze. Nejsnazší je, pokud uživatel sám definuje, která do-

ména je aktuální. Jisté adaptace lze dosáhnout i použitím modelu s krátkodobou pamětí. Adaptace modelu může přinést značné zlepšení jeho funkce, což implikuje důležitost metod automatické detekce domény (tématu). Nutným předpokladem opět je, že jsou k dispozici potřebná data.

Jazyk lze také modelovat na základě jeho strukturálních vlastností. Nejběžnějším příkladem jsou přístupy založené na gramatikách. Vzhledem k rozsahu korpusů se obvykle jedná o stochastické gramatiky a stejně jako v případě modelů založených na spouštěčích tkví problém především v nalezení vhodných pravidel.

V současné době jsou také v úloze jazykového modelování nasazovány neuronové sítě. V případě [5] je podobně jako u n-gramových modelů provedena aproximace

$$f\left(w_t, \ldots w_{t-n+1}\right) = \hat{P}\left(w_t \middle| w_1^{t-1}\right) \quad ,$$

kde $f\left(\ldots\right)$ zastupuje konstruovaný model a $\hat{P}(\ldots)$ je odhad podmíněné pravděpodobnosti slova w_t za podmínky výskytu sekvence slov w_1^{t-1}. Slova nemohou být neuronové síti předkládána přímo, proto je definováno zobrazení C, které každému slovu i ze slovníku V přiřadí reálný (příznakový) vektor $C\left(i\right) \in \mathbb{R}^m$. Dále je definována pravděpodobnostní funkce g, která popisuje podmíněnou pravděpodobnost slova slova i za podmínky výskytu sekvence příznakových vektorů slov v jejich kontextu. Model $f\left(\ldots\right)$ je pak vyjádřen jejich kompozicí, tedy

$$f\left(i, w_{t-1}, \ldots, w_{t-n+1}\right) = g\left(i, C\left(w_{t-1}\right), \ldots, C\left(w_{t-n+1}\right)\right) \quad .$$

Problém konstrukce jazykového modelu je převeden na problém volby zobrazení C a funkce g, která může být reprezentována umělou neuronovou sítí. Dle experimentů [5] byla perplexita[1] testovacího korpusu na modelech s neuronovou sítí vždy nižší.

[1] viz kapitola 2.4

2.4 Posouzení kvality modelu

Pro nalezení vhodného přístupu, jak jazykový model sestavit, je potřeba mít metodu, která by nám umožnila změřit jeho kvalitu. K tomu se obvykle používá *perplexita*. V [3] je perplexita definována vztahem

$$PP = \frac{1}{\sqrt[K]{\bar{P}\left(w_1 w_2 \dots w_K\right)}} \quad . \tag{2.4.1}$$

K-tá odmocnina ve jmenovateli zajišťuje, že delší věty nebudou znevýhodněny. Vyšší hodnota perplexity (česky „komplikovanost“ nebo „složitost“) znamená horší model pro predikci dané věty.

Často se též používá *logaritmus perplexity*. Pak navíc můžeme vyčíslení odhadu pravděpodobnosti upravit a pro trigramový model obdržíme vztah

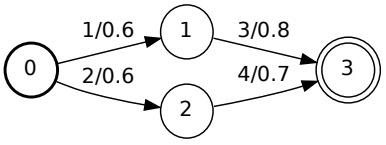
$$LP = -\frac{1}{K}\sum_{i=1}^{K} \log_2 \bar{P}\left(w_i | w_{i-2} w_{i-1}\right) \quad . \tag{2.4.2}$$

Kapitola 3

Vážené konečné automaty

Konečné automaty nacházejí (především ve spojení s knihovnou OpenFst [6]) v reálných úlohách stále větší uplatnění. I v této práci jsou v podstatě základním stavebním kamenem. Nejprve uveďme příklad, který nás dovede k pojmu *polookruh.*

Uvažujme orientovaný graf s hranami ohodnocenými pravděpodobností jejich použití. Hrany jsou navíc pro lepší přehlednost očíslovány. Zajímá nás pravděpodobnost všech cest, které vedou z počátečního do koncového uzlu. Počáteční uzel má číslo 0 a je označen tučnou čarou; koncový uzel má číslo 3 a je označen dvojitou čarou (obr. 3.0.1).



Obrázek 3.0.1: Orientovaný graf s hranami ohodnocenými pravděpodobností

Z počátečního do koncového uzlu vedou dvě cesty a to přes hrany 1 a 3 ($P(1,3)$) a přes hrany 2 a 4 ($P(2,4)$). Platí tedy

$$\begin{aligned} P(1,3) &= P(1) \cdot P(3) = 0{,}6 \cdot 0{,}8 = 0{,}48 \doteq 0{,}4 \\ P(2,4) &= P(2) \cdot P(4) = 0{,}6 \cdot 0{,}7 = 0{,}42 \doteq 0{,}4 \end{aligned} \quad . \tag{3.0.1}$$

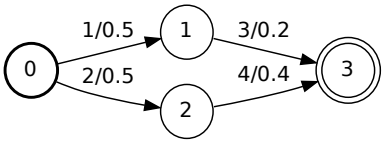
V grafu jsou pravděpodobnosti uvedeny s přesností na jedno desetinné

místo. Každé číslo s plovoucí desetinnou čárkou je v počítači možno reprezentovat jen omezeným počtem číslic. Předpokládejme na chvíli, že jsme schopni uchovat jen jedno desetinné místo tak, jak to ukazuje zaokrouhlení v (3.0.1). Pak ale je pravděpodobnost obou možných cest stejná, což je v rozporu s předpokladem. Došlo k podtečení.

Tento problém částečně vyřešíme, pokud namísto ohodnocení hran pravděpodobností p použijeme váhy w například tak, že

$$w = -\ln p \quad .$$

Tak dostaneme graf na obr. 3.0.2.



Obrázek 3.0.2: Orientovaný graf s hranami ohodnocenými váhou

Použité váhy hran jsou záporné logaritmy původních ohodnocení. Jelikož jsme ty při průchodu cestou násobili, budeme nové váhy sčítat. A tedy

$$\begin{aligned} P(1,3) &= P(1) + P(3) = 0{,}5 + 0{,}2 = 0{,}7 \doteq 0{,}7 \\ P(2,4) &= P(2) + P(4) = 0{,}5 + 0{,}4 = 0{,}9 \doteq 0{,}9 \end{aligned} \quad . \tag{3.0.2}$$

Vzhledem ke způsobu výpočtu vah platí, že cesta s nižší vahou má vyšší pravděpodobnost. I přes zaokrouhlení na jedno desetinné místo výsledek odpovídá předpokladu. Stále hrozí nebezpečí numerického charakteru a to naopak přetečení. Lze však předpokládat, že sčítáním bude, oproti násobení, docházet k mnohem pomalejšímu pohybu desetinné čárky.

Forma a způsob použití ohodnocení hran (přechodů) je ve vážených konečných automatech definována *polookruhem*, nad kterým je daný automat postaven.

3.1 Polookruhy

Pro celou řadu operací nad váženými konečnými automaty je nutné, aby přiřazené váhy měly strukturu polookruhu. Pětice $\left(S, \oplus, \otimes, \bar{0}, \bar{1}\right)$ je polookruh, pokud $\left(S, \oplus, \bar{0}\right)$ je komutativní monoid s identickým (neutrálním) elementem $\bar{0}$, $\left(S, \otimes, \bar{1}\right)$ je monoid s identickým elementem $\bar{1}$, $\otimes$ je distributivní přes $\oplus$ a $\bar{0}$ je absorbční prvek (element) $\otimes : \forall a \in S, a \otimes \bar{0} = \bar{0} \otimes a = \bar{0}$. Příklady polookruhů ukazuje tab. 3.1.1 [7].

Polookruh	Množina	$\oplus$	$\otimes$	$\bar{0}$	$\bar{1}$
Booleanský	$\{0, 1\}$	$\vee$	$\wedge$	0	1
Pravděpodobnostní	$\mathbb{R}_+ \cup \{+\infty\}$	$+$	$\times$	0	1
Logaritmický	$\mathbb{R}_+ \cup \{-\infty, +\infty\}$	$\oplus_{\log}$	$+$	$+\infty$	0
Tropický	$\mathbb{R}_+ \cup \{-\infty, +\infty\}$	min	$+$	$+\infty$	0

Tabulka 3.1.1: Příklady polookruhů ($x \oplus_{\log} y = -\log\left(e^{-x} + e^{-y}\right)$)

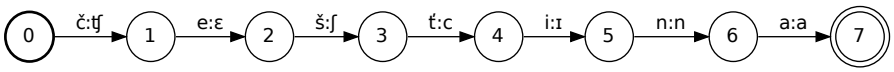
Uvažujme polookruh $\left(S, \oplus, \otimes, \bar{0}, \bar{1}\right)$; potom lze vážený konečný transducer nad tímto polookruhem definovat

$$T = (\Sigma, \Delta, Q, I, F, E, \lambda, \rho) \quad ,$$

kde Σ je konečná množina vstupních symbolů, Δ značící konečnou množinu výstupních symbolů, Q je konečná množina všech stavů, $I \subseteq Q$ je konečná množina počátečních stavů, $F \subseteq Q$ je konečná množina koncových stavů, $E = Q \times (\Sigma \cup \{\epsilon\}) \times (\Delta \cup \{\epsilon\}) \times S \times Q$ je konečná množina přechodů (ϵ je prázdný řetězec), $\lambda : I \to S$ je funkce přiřazující váhu počátečním stavům a $\rho : F \to S$ je funkce přiřazující váhu koncovým stavům [7]. V následujícím textu budeme vždy uvažovat vážené konečné automaty nad tropickým polookruhem, pokud nebude řečeno jinak.

Takový konečný automat se nazývá *transducer* (převodník), protože převádí řetězec α na β. Příkladem je obr. 3.1.1, kde je zobrazen transducer provádějící (bez ohodnocení) fonetickou transkripci slova „čeština“ na jeho ekvivalent ve fonetické abecedě IPA[1].

[1] International Phonetic Alphabet



Obrázek 3.1.1: Příklad transduceru, který převádí slovo „čeština“ na jeho fonetický ekvivalent ve fonetické abecedě IPA

Speciálním případem transducerů jsou ty, které mají na všech přechodech vstupní a výstupní symbol totožný. Neprovádějí tedy žádný převod, pouze odpovídají na otázku, zda-li řetězec patří do jazyka generovaného tímto automatem. Říkáme jim proto *akceptory*. Transducery i akceptory lze obecně nazývat automaty.

Poznamenejme, že dva transducery jsou ekvivalentní, pokud pro každý vstupní řetězec α je výstupem obou stejný řetězec β se stejnou vahou. Výstupní řetězce se ovšem může lišit v počtu použitých přechodů, umístěním symbolů ϵ a rozložením jednotlivých vah na přechodech [8].

Dva řetězce jsou stejné, pokud se liší jen v počtu a umístění symbolů prázdného řetězce ϵ.

3.2 Knihovna OpenFst

Úspěšnou softwarovou implementací vážených konečných automatů je knihovna OpenFst; umožňuje vytvářet, kombinovat, optimalizovat a prohledávat konečné automaty [6]. Ty jsou používány především při rozpoznávání a syntéze řeči, automatickém překladu přirozených jazyků, hledání příznaků, zpracování textu, strojovém učení a obecně vyhledávání informací. Často jsou také nasazovány jako stochastické modely. Knihovna je vyvíjena především skupinou lidí ze společností Google Research a NYU's Courant Institute [6].

3.2.1 Vytvoření FST (Final State Transducer)

OpenFst obsahuje knihovnu funkcí pro použití v programovacím jazyce C++ a také jejich ekvivalenty pro použití z příkazové řádky. Následuje ukázka vytvoření jednoduchého automatu právě z příkazové řádky.

Nejprve si připravíme trojici souborů ve formátu AT&T FSM [9]

text.fst	isyms.txt	osyms.txt
0 1 a x 0.5	<eps> 0	<eps> 0
0 1 b y 1.5	a 1	x 1
1 2 c z 2.5	b 2	y 2
2 3.5	c 3	z 3

Každá řádka v souboru `text.fst` značí jednu hranu grafu. První řádek nám tedy říká, že v grafu povede hrana (přechod) ze stavu 0 do stavu 1 se vstupním symbolem „`a`“, výstupním symbolem „`x`“ a vahou 0,5. Zároveň říká (protože je první), že výchozím stavem automatu bude 0. Poslední řádka oproti tomu značí, že stav 2 patří mezi koncové stavy a má váhu 3,5.

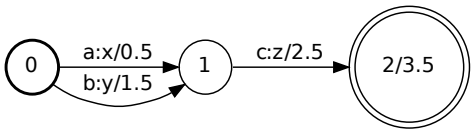
Soubor `isyms.txt` respektive `osyms.txt` definuje množinu vstupních respektive výstupních symbolů. Speciálnímu znaku „`<eps>`“ se dle konvence vždy přiřazuje index 0 a značí prázdný symbol. V abecedě se opět dle konvence uvádí bez ohledu na to, zda ho chceme použít. Některé operace ho totiž vyžadují ke své funkci (např. `Concat` spojí dva transducery právě přechodem „`<eps>:<eps>`“). Následně spustíme příkaz

```
$ fstcompile --isymbols=isyms.txt --osymbols=osyms.txt \
      --keep_isymbols --keep_osymbols text.fst binary.fst
```

který provede sestavení automatu. Příkazem

```
$ fstdraw binary.fst | dot -Tpng -Grotate=-90 -oimage.png
```

získáme soubor `image.png` jako na obr. 3.2.1. Tento transducer převede řetězec „ac“ na řetězec „xz“ s vahou 6,5 s řetězec „bc“ na „yz“ s vahou 7,5. Na tomto místě váhy nemají žádný zjevný užitek a slouží jen pro ilustraci.

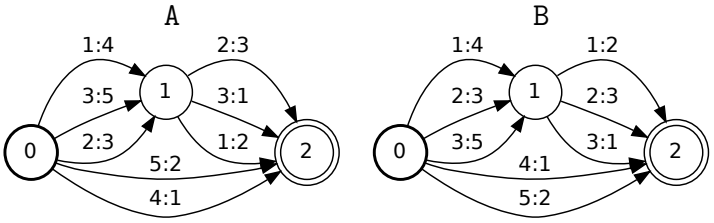


Obrázek 3.2.1: Příklad jednoduchého automatu [10]

3.2.2 Použité operace (v abecedním pořadí)

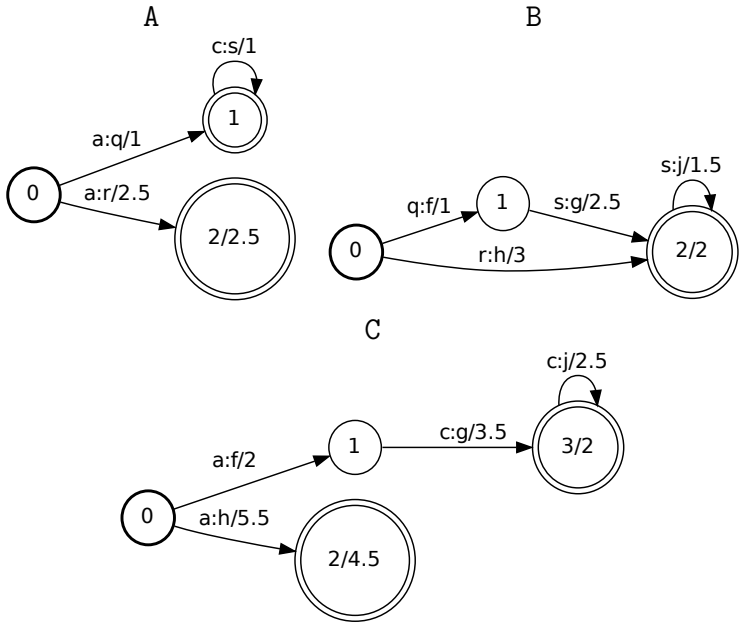
`ArcSort` seřadí hrany vedoucí ze stavu i do stavu j buďto podle vstupních, nebo podle výstupních symbolů a to pro všechny možné kombinace i a j, jako

na obr. 3.2.2, kde `A` je výchozí automat a `B` je automat s přechody seřazenými podle vstupních symbolů.



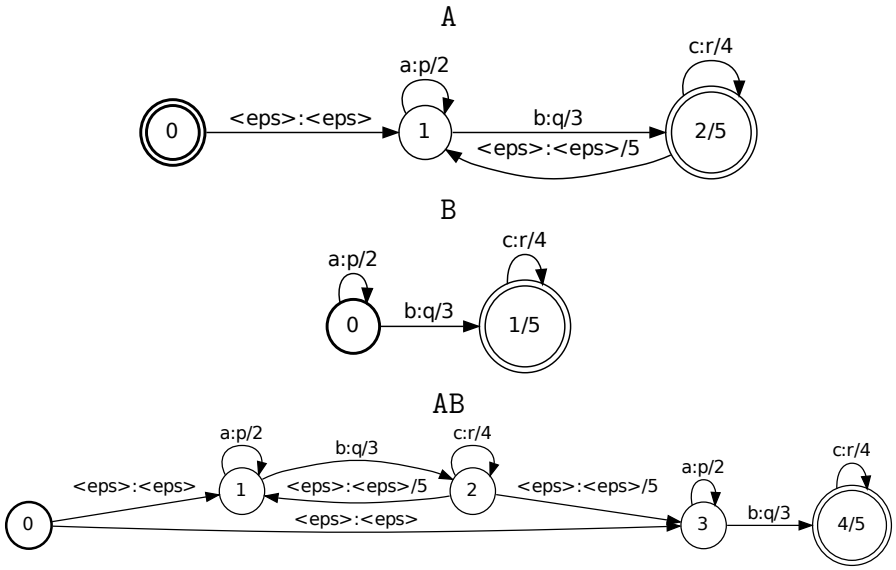
Obrázek 3.2.2: Příklad operace `ArcSort`

Operace `Compose` počítá kompozici dvou transducerů. Ta lze definovat tak, že pokud `A` převádí „x“ na „y“ s vahou a a `B` převádí „y“ na „z“ s vahou b, pak jejich kompozice `C` převádí „x“ na „z“ s vahou $a \otimes b$. Příklad operace ukazuje obr. 3.2.3.



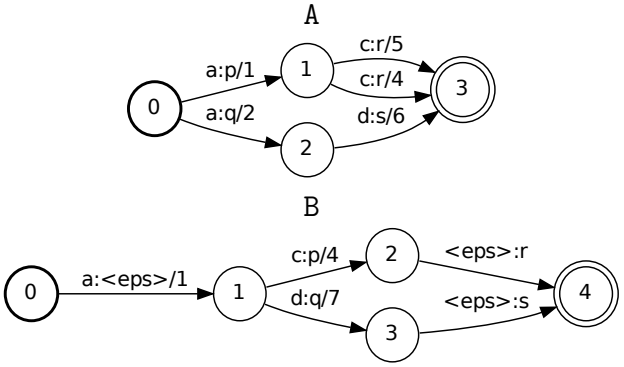
Obrázek 3.2.3: Příklad operace `Compose`

Operace `Concat` je definována tak, že pokud `A` přepisuje „x“ na „y“ s vahou a a `B` přepisuje „w“ na „v“ s vahou b, pak výsledek operace nad těmito transducery je transducer takový, že přepisuje „xw“ na „yv“ s vahou $a \otimes b$. Jednoduše řečeno, dochází k zapojení `B` za `A`, jak ukazuje obr. 3.2.4.



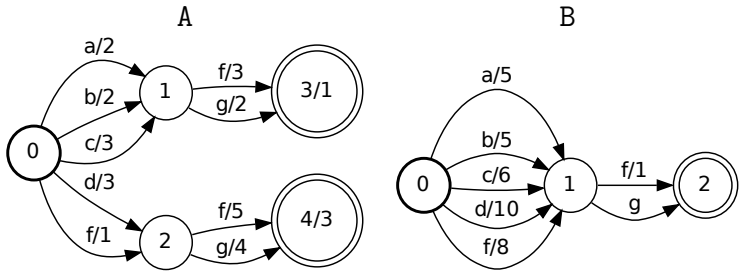
Obrázek 3.2.4: Příklad operace `Concat`

`Determinize` jednoduše provádí determinizaci automatu. Tato operace je velmi důležitá, protože deterministický automat může být minimalizován (viz níže). Deterministický automat má tu vlastnost, že ze žádného uzlu nevedou dvě hrany se stejným vstupním symbolem. Příklad je na obr. 3.2.5.



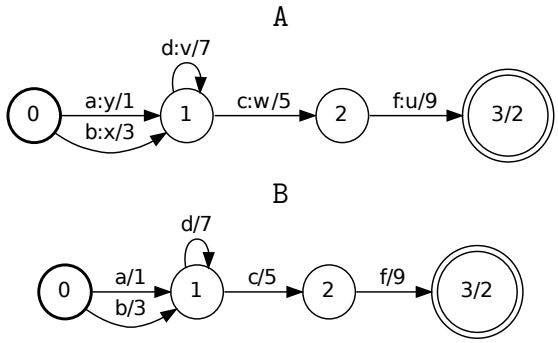
Obrázek 3.2.5: Příklad operace `Determinize`

`Minimize` provádí minimalizaci daného automatu; tedy je-li na vstupu akceptor `A`, na výstupu je akceptor `B` ekvivalentní `A` s minimálním počtem stavů. Situace v případě transducerů je poněkud složitější [11]; obr. 3.2.6.



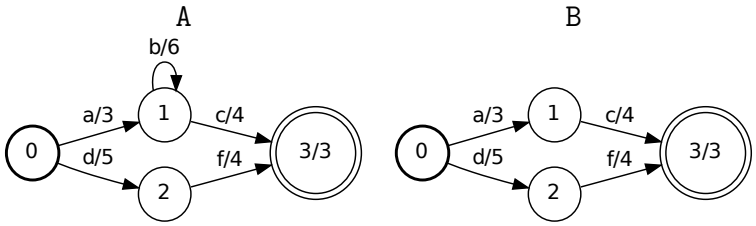
Obrázek 3.2.6: Příklad operace `Minimize`

`Project` převádí transducer `A` na akceptor `B` tak, že zachová na přechodech pouze vstupní respektive výstupní symboly; obr. 3.2.7.

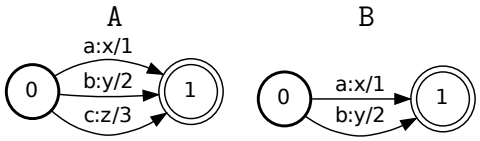


Obrázek 3.2.7: Příklad operace `Project` (vstupní symboly)

`Prune` odstraňuje z automatu nadbytečné stavy a přechody. Konkrétně odstraní všechny stavy a přechody cest, které nepatří k „úspěšné“ cestě (cesta z počátečního do koncového stavu) a které svou vahou přesahují práh $t \otimes$„váha nejkratší cesty grafem“, kde t je předaný parametr a nejkratší cestou je myšlena cesta s nejnižší vahou (cenou); obr. 3.2.8. Obr. 3.2.9 ukazuje jiný příklad, kdy je zvolena hodnota $t = 1{,}5$. Automat je postaven nad tropickým polookruhem, nejkratší cesta grafem („`a:x`“) má váhu 1, tedy dojde k odstranění přechodů cest s vahou větší než $1 + 1{,}5 = 2{,}5$. To splňuje pouze cesta „`c:z`“.

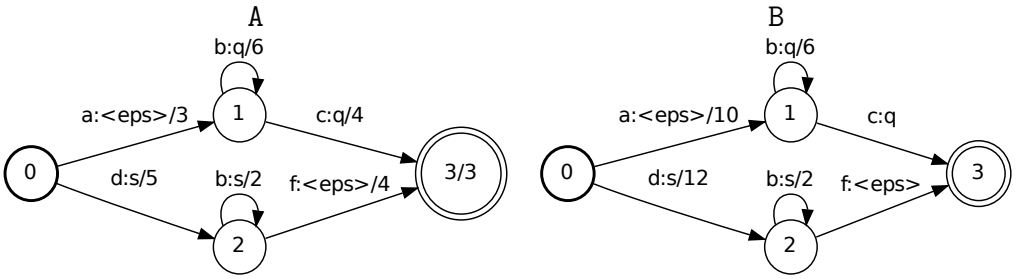


Obrázek 3.2.8: Příklad operace `Prune` (s prahem 3)



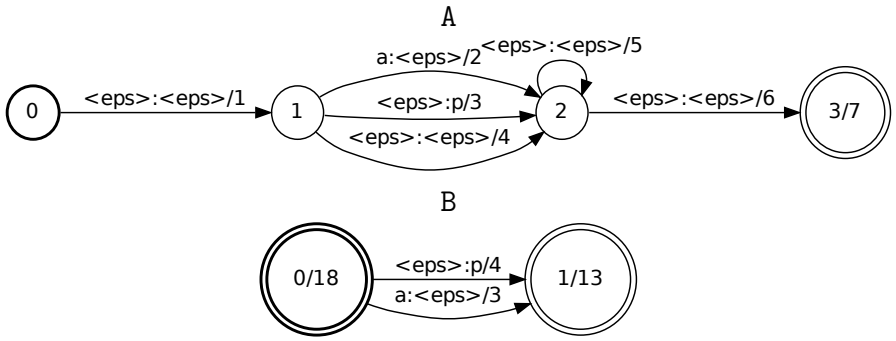
Obrázek 3.2.9: Příklad operace `Prune` (s prahem $1, 5$)

Operace `Push` vytváří ekvivalentní transducer s vahami (a) nebo symboly „stlačenými“ směrem k výchozímu, nebo koncovým stavům. Pro příklad vezměme operaci Push směrem k výchozímu stavu. Pak musí platit, že v každém stavu (kromě výchozího) je součet ($\oplus$) vah odchozích přechodů roven $\bar{1}$. Je třeba si uvědomit, že (dle výchozího nastavení) sestaven nad tropickým pologokruhem, tedy že $\oplus = \min$ a že $\bar{1} = 0$. `B` je transducer, který získáme aplikací operace `Push` (vůči počátečnímu stavu) na transducer `A`; obr. 3.2.10.



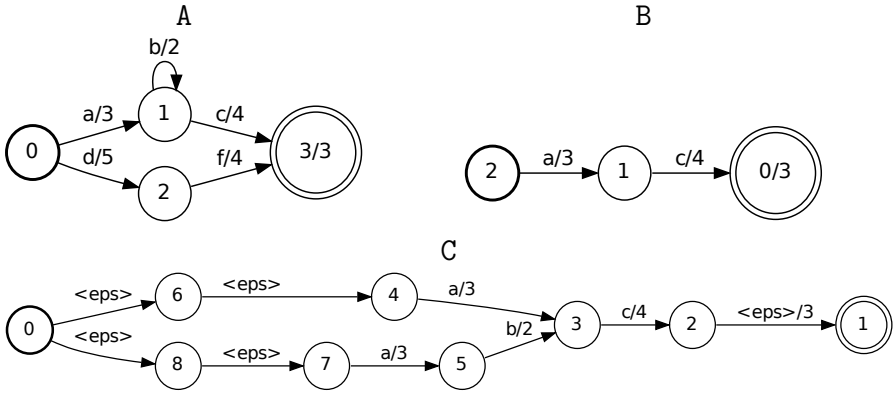
Obrázek 3.2.10: Příklad operace `Push`

`RmEpsilon` odstraňuje z transduceru ϵ-přechody, tedy takové přechody, kde je jako vstupní i jako výstupní symbol použit ϵ („`<eps>`“). Výstupní transducer `B` je ekvivalentní vstupnímu `A`; obr. 3.2.11.



Obrázek 3.2.11: Příklad operace `RmEpsilon`

`ShortestPath` vytváří vážený konečný transducer obsahující n-nejkratších cest (čímž se rozumí n cest s nejnižší vahou). Na obr. 3.2.12 je `A` výchozí transducer, `B` je výsledek operace s $n = 1$ a `C` je výsledek pro $n = 2$.



Obrázek 3.2.12: Příklad operace `ShortestPath`

3.3 Knihovna OpenGrm NGram

Jednou z praktických aplikací knihovny OpenFst je knihovna *OpenGrm Ngram*, která umožňuje snadné vytváření a úpravu n-gramových modelů ve formě vážených konečných transducerů; vyvinuta především lidmi ze společností OHSU a Google Research [4]. Knihovnu si představíme rovnou příkladem použití. Mějme stejně jako v kapitole 2.1 řetězec

$$a\,b\,a\,b\,b\,a$$

Vepíšeme ho do souboru `model.txt` tak, že každý znak bude od dalšího oddělen mezerou. Tím knihovna pozná, že se jedná o jednotlivé znaky a ne jediné slovo. Stejně jako v kapitole 3.2.1 o vytvoření jednoduchého automatu bude naším prvním krokem získání tabulky symbolů.

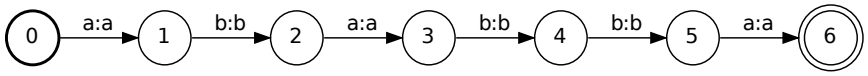
```
$ ngramsymbols --epsilon_symbol="<eps>" model.txt model.syms
```

Tento příkaz vyhledá v souboru model.txt veškeré symboly a uloží je ve známém formátu (kap. 3.2.1) do souboru `model.syms`. Pro symbol ϵ použije označení „`<eps>`“. V souboru je také možné nalézt symbol „`<unk>`“. To proto, že knihovna je schopna pracovat i se slovy mimo slovník a právě tento symbol ve vstupních datech by označoval možné umístění.

Dalším krokem je vytvoření `FAR` archivu. Jedná se o rozšíření OpenFst, kdy v jednom binárním souboru může být uloženo najednou více transducerů.

```
$ farcompilestrings --symbols=model.syms --keep_symbols \
      model.txt model.far
```

V našem případě tento archiv obsahuje pouze jeden transducer; obr. 3.3.1.

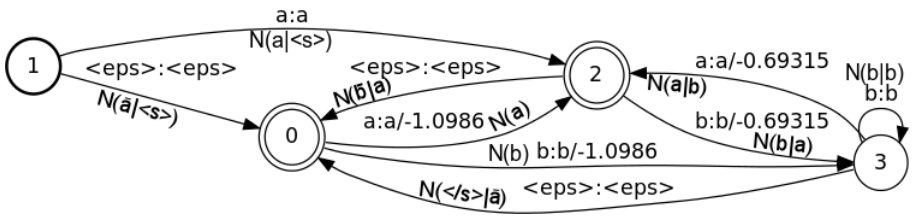


Obrázek 3.3.1: Obsah FAR archivu

Dále vypočteme četnosti všech n-gramů dle zvoleného řádu n. Zvolit řád modelu nám umožní parametr `order` v následujícím příkazu:

```
$ ngramcount --order=2 model.far model.cnts
```

Vytvořili jsme tedy automat, kde jsou hrany ohodnoceny četností daných n-gramů jako $-\ln n$, kde n je právě počet. Tento automat je vykreslen na obr. 3.3.2.



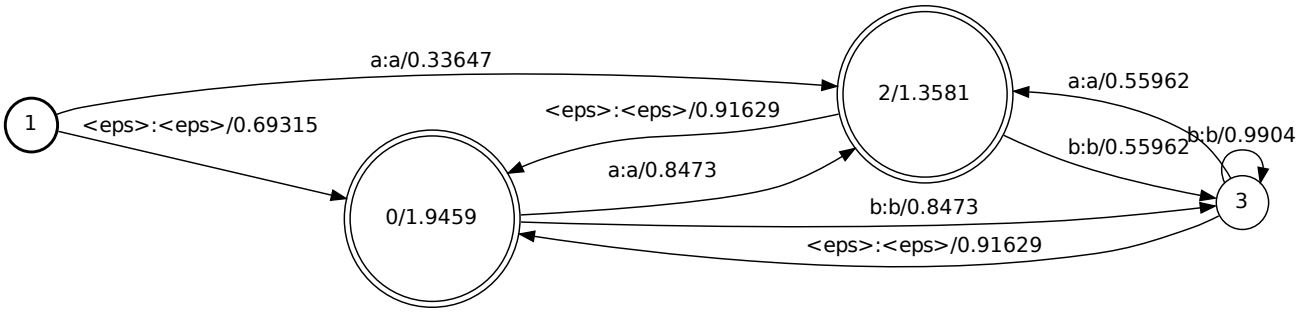
Obrázek 3.3.2: `model.cnts` - četnosti n-gramů ve vstupních datech

Ručně jsou v něm také doplněny popisky `N(...)` vysvětlující, jakému jevu daná hrana přísluší. Pokud u hrany není uvedena váha, znamená to, že její váha je nula (jev se vyskytl jednou). To ovšem neplatí pro přechody „`<eps>:<eps>`“, které jsou v modelu uvedeny jen pro následující vyhlazení. Dobře je ale na nich viděl ústupové schéma, které model používá. Ze stavu 1 vede do stavu 2 hrana „`a:a`“, s váhou nula (četností výskytu jedna), kterou jsme označili „`N(a|<s>)`“. Tento přechod nám tedy říká, že se v textu vyskytl jeden řetězec začínající symbolem „`a`“. Dalším přechodem ze stavu 1 je epsilonová hrana do stavu 0 námi označená jako „`N(ā|<s>)`“, kde ā značí symbol, který není „`a`“. V našem případě je to symbol „`b`“, obecně to však může být i jakýkoliv jiný symbol. Žádný řetězec, který by nezačínal „a“ v trénovacích datech nebyl, proto dle ústupového schématu použijeme ústupovou hranu a váhu dalšího symbolu hledáme jako $(n-1)$-gram, tedy unigram. Jsou to přechody ze stavu 0 označené `N(a)` respektive `N(b)`.

Naším dalším krokem bude vyhlazení modelu, abychom mohli reálně použít popsané ústupové schéma. Jak jsme již uvedli (kap. 2.2), použijeme k tomu Wittenovu-Bellovu metodu, která je v NGram výchozím nastavením.

```
$ ngrammake --method=witten_bell model.cnts model.mod
```

Na obr. 3.3.3 je již vyhlazený a plně funkční bigramový model natrénovaný z řetězce „ababba“. Právě takové modely (i když značně složitější) jsou základem naší aplikace.



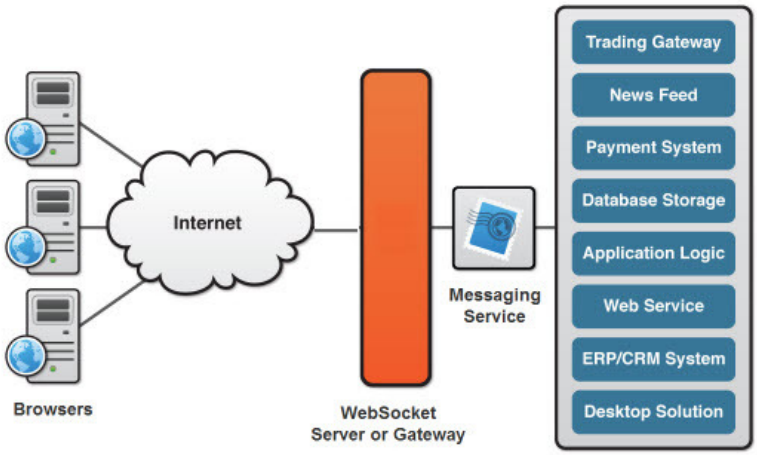
Obrázek 3.3.3: `model.mod` - vyhlazený n-gramový model

Mezi další funkce knihovny, které je dobré zmínit, patří `ngrammerge`, která umožňuje modely slučovat, `ngramshrink` pro prořezávání modelů a také `ngramperplexity` pro výpočet perplexity modelu.

Kapitola 4

Komunikace mezi serverem a klientem

Grafické uživatelské rozhraní našeho systému tvoří jednoduchá webová aplikace. Pro komunikaci se serverem byla zvolena technologie *WebSocket* [12]. Ta byla vyvinuta v rámci iniciativy HTML5 a zajišťuje obousměrnou (full-duplex) komunikaci mezi serverem a webovou stránkou (klientem) s pomocí jediného soketu formou zasílání zpráv.



Obrázek 4.0.1: Architektura WebSocket [12]

Kromě značného snížení provozu mezi serverem a klientem (data se posílají jen pokud jsou potřeba) je další předností technologie WebSocket její řešení omezení způsobovaných firewally a proxy servery. Pokud WebSocket detekuje proxy server, automaticky vytvoří tunel skrz něj, kterým pak komunikace probíhá. Tunel je vytvořen zasláním požadavku „HTTP CONNECT“, pro který proxy server vytvoří TCP/IP spojení a přes něj pak komunikace probíhá. Navíc, protože HTTP/S funguje velmi podobným způsobem, může WebSocket s pomocí SSL navázat také zabezpečené spojení. Jedná se však o velmi mladou technologii a její funkce není prozatím v prohlížečích zcela běžná [12].

4.1 WebSocket protokol

WebSocket protokol byl navržen, aby si dobře poradil s existující webovou infrastrukturou. Proto také vždy začíná spojení jako spojení HTTP, což zajišťuje určitou zpětnou kompatibilitu s technologiemi, které WebSocket neznají.

Komunikaci zahajuje vždy prohlížeč tak, že pošle serveru žádost o změnu protokolu z HTTP na WebSocket. Dělá to pomocí tzv. *Update header*:

```
GET ws://echo.websocket.org/?encoding=text HTTP/1.1
Origin: http://websocket.org
Cookie: __utma=99as
Connection: Upgrade
Host: echo.websocket.org
Sec-WebSocket-Key: uRovscZjNol/umbTt5uKmw==
Upgrade: websocket
Sec-WebSocket-Version: 13
```

Pokud server správně porozumí odsouhlasí změnu protokolu opět pomocí Update header.

```
HTTP/1.1 101 WebSocket Protocol Handshake
Date: Fri, 10 Feb 2012 17:38:18 GMT
Connection: Upgrade
Server: Kaazing Gateway
Upgrade: WebSocket
Access-Control-Allow-Origin: http://websocket.org
Access-Control-Allow-Credentials: true
Sec-WebSocket-Accept: rLHCkw/SKsO9GAH/ZSFhBATDKrU=
Access-Control-Allow-Headers: content-type
```

V této chvíli HTTP spojení končí a je nahrazeno spojením WebSocket. Stále však skrze původně navázané spojení protokolu TCP/IP. Protokol umožňuje posílat zprávy s textovým i s binárním obsahem [12].

4.2 Použití WebSocket API

Pokud je technologie v prohlížeči implementována, může být spojení navázáno zcela jednoduše a to pomocí jediného příkazu JavaScriptu. Na existující objekt `myWebSocket` se následně naváží handlery událostí (events) a komunikace může probíhat. Nakonec pošleme serveru zprávu pomocí metody `send()` a spojení ukončíme.

```
var myWebSocket = new WebSocket("ws://www.websockets.org");
myWebSocket.onopen = function(evt) {
    alert("Connection open ..."); };
myWebSocket.onmessage = function(evt) {
    alert( "Received Message: " + evt.data); };
myWebSocket.onclose = function(evt) {
    alert("Connection closed."); };
myWebSocket.send("Hello WebSockets!");
myWebSocket.close();
```

Obdobně řeší protokol WebSocket i server, ačkoliv to závisí na dané implementaci a použitém jazyku.

Kapitola 5

Aplikace

5.1 Jazykové modely

První jazykový model, který byl převážně využíván při vývoji aplikace, byl sestaven ručně a obsahoval pouze několik málo vět. V aplikaci je označen jako „Uživatelský“, protože vzhledem k jeho velikosti má každý vstup uživatele velký vliv 5.4.

Další jazykový model byl vytvořen z přepisů záznamů telefonních hovorů pořízených původně pro vývoj aplikace Nádraží [13] a má proto název „Nádraží“. Přepisy však často obsahovaly zdvořilostní fráze jako „dobrý den“, „na shledanou“, „děkuji“ a další výplňová slova jako „haló“, „dobře, dobře“, „prostě“, nebo „takže“, která významným způsobem negativně ovlivňovala kvalitu predikce. Proto byla z trénovacích dat ručně odstraněna. V případě reálného nasazení aplikace se předpokládá definování takto často používaných vět jako zkratek (viz kap. 5.3), které bude možné nechat jednoduše vyslovit, ale používaný model ovlivňovat nebudou, nebo jiná metoda, která jejich negativní dopad vyloučí. Vzhledem k rozsahu trénovacích dat byla tato slova odstraňována pomocí regulárních výrazů a nebylo možné zkontrolovat smysluplnost všech pozměněných vět. Z predikcí slov generovaných tímto modelem je zřejmé, že se jedná o přepis přirozené řeči. Slovník modelu má 3 180 slov.

Následně byly přidány ještě modely „Počasí“ (20 336 slov), „Vaření“ (29 257

slov) a „Kultura“ (106 468 slov). Tyto modely pochází z projektu JMZW [14, 15] a byly opět pro aplikaci ručně upraveny (např. vymazání interpunkce). JMZW tvoří korpusy z textu dostupných webových stránek. Proto tyto modely nemusí vyhovovat potřebám uživatele.

Všechny modely byly zpracovány postupem popsaným v kapitole 3.3; tedy byly sestaveny n-gramové modely pro různé hodnoty n (viz kap. 6.1), konkrétně Wittenův-Bellův model využívající ústupové schéma.

Tabulka 5.1.1 zobrazuje preplexitu trénovacích dat jednotlivých modelů v závislosti na zvoleném řádu. Zcela podle předpokladu perplexita s rostoucím řádem modelu ve všech případech klesá. Dozvídáme se zde však pouze to, že budou-li uživatelem zadávané věty odpovídat těm použitým pro konstrukci modelu, bude lépe zvolit modely vyššího řádu. Hodnota této informace se však v kapitole 6.1 ukáže jako mizivá.

Model	Řád modelu	Perplexita trénovacího korpusu
Nádraží	2	18,6417
	3	6,66135
	4	3,94623
	5	3,14479
Počasí	2	38,7657
	3	4,42135
	4	2,13566
	5	1,62383
Vaření	2	53,3702
	3	3,87924
	4	1,87547
	5	1,51118
Kultura	2	107,697
	3	5,92843
	4	2,16918
	5	1,60628

Tabulka 5.1.1: Perplexita trénovacích korpusů použitých jazykových modelů v závislosti na jejich řádu

5.2 Server

Serverová část aplikace je napsána v jazyce *Python* [16]. Tento jazyk byl zvolen jednak proto, že se jeví přehledný a přívětivý k uživateli a jednak také proto, že je zde na katedře kybernetiky (FAV, ZČU) široce používán.

Základ serveru je sestaven nad balíkem *Tornado* (Tornado WebSocket), což je implementace WebSocket serveru. Následuje příklad jednoduchého WebSocket serveru nad balíkem Tornado. Server pouze vypisuje hlášení při připojení a odpojení klienta a na příchozí zprávu reaguje tak, že lehce pozměněnou klientovi vrací (echo - ozvěna).

```
class EchoWebSocket(websocket.WebSocketHandler):
    def open(self):
        print "WebSocket opened"
    def on_message(self, message):
        self.write_message(u"You said: " + message)
    def on_close(self):
        print "WebSocket closed"
application = tornado.web.Application([
    (URI, WSHandler),
])
http_server = tornado.httpserver.HTTPServer(application)
http_server.listen(PORT)
tornado.ioloop.IOLoop.instance().start()
```

Tento server spolu s klientskou částí z kap. 4.2 tvoří plnohodnotnou aplikaci schopnou obousměrné komunikace.

Součástí serveru je také jednoduchý TTS Server umožňující převod zaslaných vět do řeči (ve formě zvukových souborů). Ty jsou následně zasílány zpět prohlížeči, který je přehraje.

Vzhledem k tomu, že knihovna OpenFst je napsána v jazyce C++, její přímé použití v jazyce Python není možné. Existuje projekt *pyopenfst* [17] zpřístupňující funkcionalitu knihovny. Nicméně se jedná skutečně pouze o rozhraní pro použití funkcí, které knihovna nabízí. Namísto toho je v práci použit projekt *pyfst*, který je vyvíjen zde na katedře Kybernetiky (FAV ZČU

v Plzni). Pyfst je v tomto ohledu mnohem „jemnější“ a umožňuje uživateli pracovat s konečnými automaty stejně jako se skutečnými objekty jazyka Python a přesto využívat výhod knihovny OpenFst.

5.2.1 Získávání návrhů slov

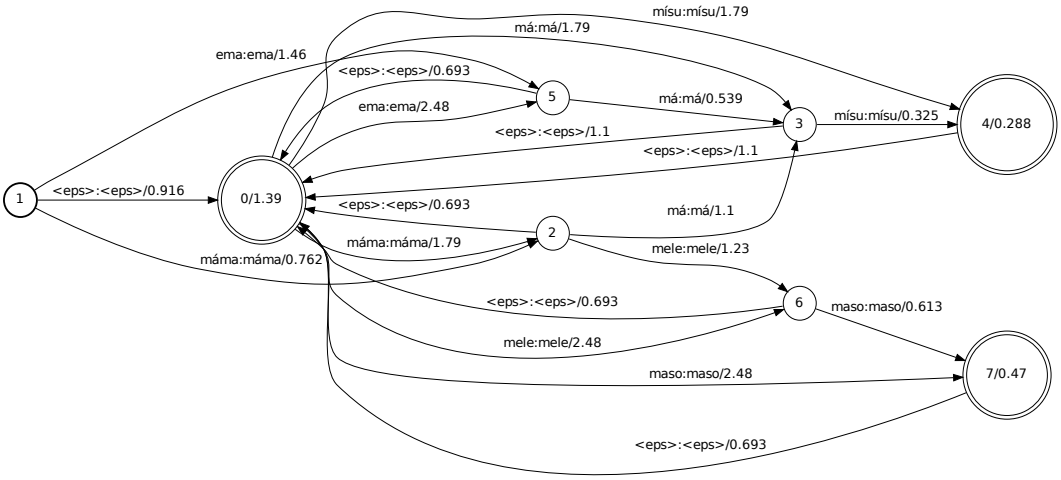
Jakmile se klient připojí k serveru, zašle mu zprávu, který model má být načten, a protože ještě uživatel nezadal žádná slova, je třeba sestavit apriorní návrhy sekvencí slov. Dostáváme se tedy k otázce, jak získat z jazykového modelu m nejpravděpodobnějších sekvencí o $1, 2, \ldots, n$ slovech. Proměnlivá délka sekvencí slov (vět) byla zvolena záměrně namísto hledání m nejpravděpodobnějších slov, aby bylo možné vkládat do textu rovnou i delší promluvy, pokud je pravděpodobnost jejich umístění v promluvě dostatečně vysoká. Každá z m sekvencí může mít obecně proměnnou délku (maximálně n slov), přičemž tato délka je dána pouze vahou cesty, které sekvence odpovídá.

Předpokládejme, že máme k dispozici bigramový jazykový model M reprezentovaný váženým konečným transducerem, sestavený z vět

```
máma má mísu
ema má mísu
máma mele maso
```

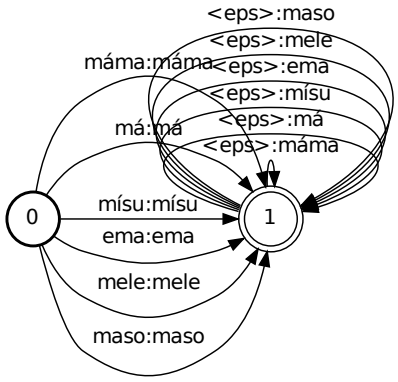
a chceme nalézt jediné slovo, kterým bude věta uživatele s největší pravděpodobností začínat. Tento model je na obr. 5.2.1. Příklad jak takový dotaz Q může vypadat je na obr. 5.2.2. Následně provedeme kompozici tohoto dotazu s modelem $(Q \circ M)$, a tak získáme hledaná slova.

Prozatím nemáme k dispozici žádnou historii, proto z počátečního stavu 0 (obr. 5.2.2) vede do stavu 1 rovnou několik hran a to přesně jedna pro každý symbol ze slovníku modelu. Tím docílíme, že prostor mezi těmito stavy bude po kompozici vyplněn vždy prvním slovem z možných cest modelem. Pokud přechod prvního slova cesty modelu vede do koncového stavu, úspěšnou cestu jsme již našli a pokračujeme v hledání další. Pokud však ještě v koncovém stavu nejsme, umožňují hrany vedoucí z a zároveň do stavu 1 pokračovat v cestě přes libovolný symbol, dokud koncový stav nenalezneme. Váhy těchto



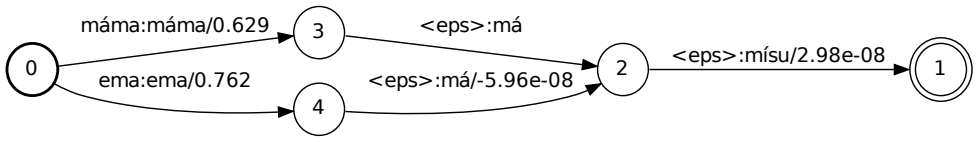
Obrázek 5.2.1: Sestavený bigramový model

přechodů se započítají do váhy nalezené cesty, ale protože jako vstupní symbol přechodu je symbol „`<eps>`“, mezi navrhovanými slovy je nenajdeme (viz níže).



Obrázek 5.2.2: Dotaz modelu na jedno slovo, pokud je historie prázdná (začátek nové věty)

Provedeme tedy kompozici dotazu (obr. 5.2.2) s modelem (obr. 5.2.1); z výsledného transduceru vybereme dvě nejlepší cesty (`ShortestPath`), odstraníme epsilonové hrany (`RmEpsilon`) a provedeme operaci `Push` s normováním odchozích hran počátečního stavu. Váhy přechodů $4 \rightarrow 2$ a $2 \rightarrow 1$ o málo větší než nula jsou důsledkem nedokonalosti operace Push pro transducery [18]. Výsledek je na obr. 5.2.4.



Obrázek 5.2.3: Výsledek dotazu (obr. 5.2.2) do modelu (obr. 5.2.1)

Výběrem pouze vstupních symbolů (`Project`) obdržíme dva řetězce:

Řetězec	Váha	Pravděpodobnost
máma	0,629	0,533
ema	0,762	0,467

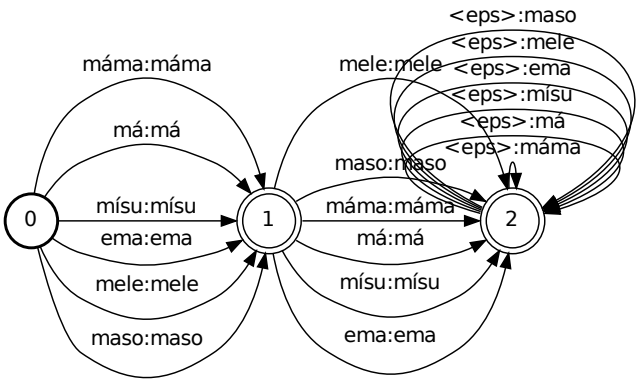
Odpověď na dotaz tedy je, že nová věta může začínat slovy „máma“ nebo „ema“, přičemž slovo „máma“ je pravděpodobnější; což odpovídá modelu. Připomeňme, že

$$\text{váha} = -\ln \text{pravděpodobnost} \quad \text{a tedy, že}$$
$$\text{pravděpodobnost} = e^{-\text{váha}} \quad .$$

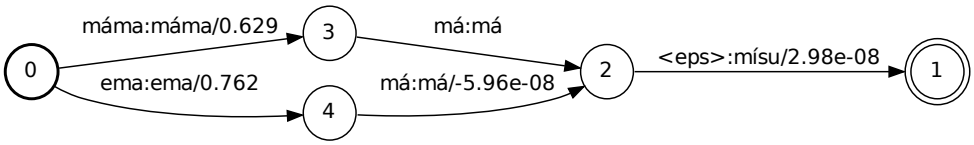
Příklad dotazu na jedno až dvě slova je pak na obr. 5.2.4. Tento se oproti obr. 5.2.2 jen nepatrně pozměnil. Levá část je zde nyní dvakrát (pro každé žádané slovo jednou). Navíc byl ale stav 1 označen jako koncový (v obrázku dvojitou čarou). Tím vlastně říkáme, že cesta může obsahovat jedno až dvě slova. Dotaz zpracujeme obdobným způsobem a jeho výsledek je na obr. 5.2.5. Tentokrát jsme získali řetězce:

Řetězec	Váha	Pravděpodobnost
máma má	0,629	0,533
ema má	0,762	0,467

což opět odpovídá předpokladu.



Obrázek 5.2.4: Dotaz modelu na jedno až dvě slova, pokud je historie prázdná (začátek nové věty)

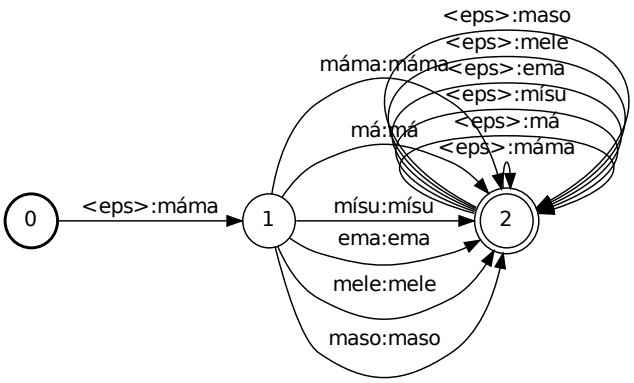


Obrázek 5.2.5: Výsledek dotazu na jedno až dvě slova, pokud je historie prázdná (začátek nové věty)

Maximální počet slov n, které chceme vyhledat, je v této chvíli pevně nastaven a vhodná volba hodnoty tohoto parametru bude předmětem dalšího zkoumání.

5.2.2 Dotaz na návrhy slov s historií

Předpokládejme, že uživatel již začal větu slovem „máma“ a my mu chceme nabídnout slovo, kterým by mohl pokračovat. Je tedy třeba sestavit dotaz, ve kterém již zadané slovo bude zaneseno. Námi zvolené řešení dotazu je na obr. 5.2.6. Použití hrany „`<eps>:máma`“ zastupující zde zastupuje historii má tu výhodu, po kompozici se na jejím vstupu prázdný symbol „`<eps>`“ uchová. Proto projekcí vstupních symbolů vymizí a v návrzích, které chceme uživateli předat nám nebude překážet.

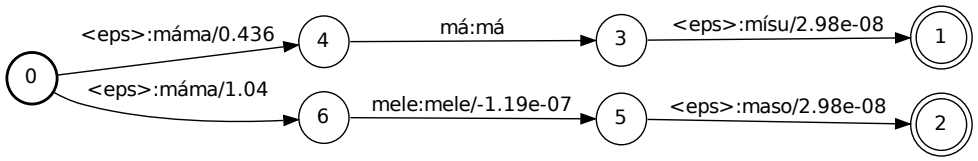


Obrázek 5.2.6: Dotaz na jedno slovo, pokud již bylo zadáno slovo „máma“

Výsledek dotazu je na obr. 5.2.7. Stejně jako v předchozích případech vezmeme pouze vstupní symboly a obdržíme návrhy slov, které by mohly podle modelu následovat po slově „máma“:

Řetězec	Váha	Pravděpodobnost
má	0,436	0,647
mele	1,04	0,353

Výsledek odpovídá předpokladu a stejně jako v kap. 5.2.1 bychom mohli provést takové hledání pro libovolný počet slov.

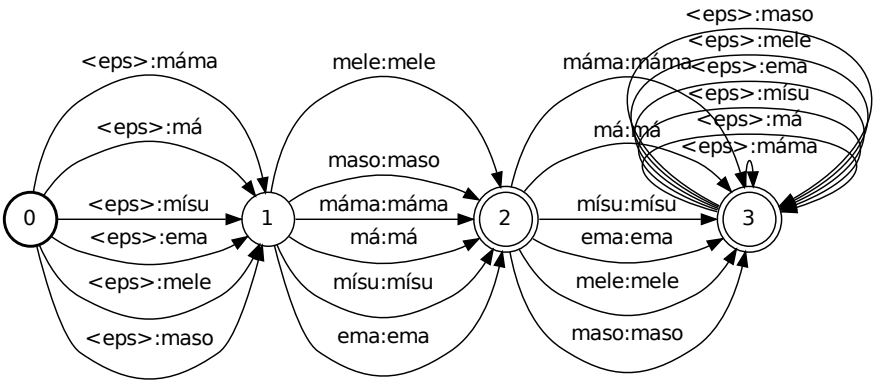


Obrázek 5.2.7: Výsledek dotazu na jedno slovo, pokud již bylo zadáno slovo „máma“

5.2.3 Slova mimo slovník

Kromě vyhledávání nad slovy, které slovník zná, jsme se pokusili implementovat i jednoduché řešení vyhledávání návrhů slov pro případ, kdy se v historii (již zadaném textu) vyskytuje slovo, které ve slovníku modelu není. Představme si, že chceme vyhledat jedno až dvě slova, která by mohla následovat po slově „karel“. Takové slovo ve slovníku není. Budeme postupovat podobně

jako v případě dotazu na obr. 5.2.6, ovšem slovo „karel“, které náš model nezná nahradíme sérií přechodů „`<eps>:*`“, kde `*` zastupuje postupně všechna slova slovníku. Vznikne tak dotaz na obr. 5.2.8.

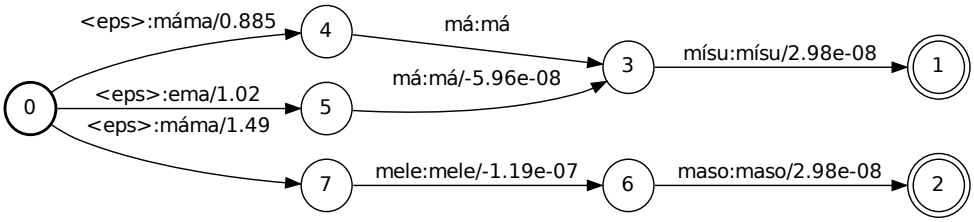


Obrázek 5.2.8: Dotaz na jedno až dvě slova, pokud již bylo zadáno jedno neznámé slovo

Provedeme dotaz a tentokrát vezmeme za výsledek tři nejlepší cesty. Pro slovo „karel“ tedy získáme návrhy:

Řetězec	Váha	Pravděpodobnost
má	0,885	0,413
má	1,02	0,361
mele	1,49	0,225

Jak je vidět, výsledek, který jsme obdrželi je poměrně dobrý. K duplicitě výrazů (jako zde „má“) může docházet. Aplikace řeší takovou situaci tak, že předává klientovi pouze unikátní návrhy a vždy ty s lepší vahou.



Obrázek 5.2.9: Výsledek dotazu na jedno až dvě slova, pokud již bylo zadáno jedno neznámé slovo

Takové řešení je však poněkud naivní. Pokud bychom například chtěli vyhledat jedno slovo pro historii „máma dobře“, obdržíme návrhy „maso“ a „mísu“. Metoda bere v úvahu pouze umístění slova ve větě, nikoliv jeho smysl. Zde by pravděpodobně našly do budoucna uplatnění jazykové modely založené na třídách respektive na slovních druzích (viz kap. 2.3). V kapitole 5.4 si ukážeme jak i tato věta obsahující slovo mimo slovník náš model obohatí.

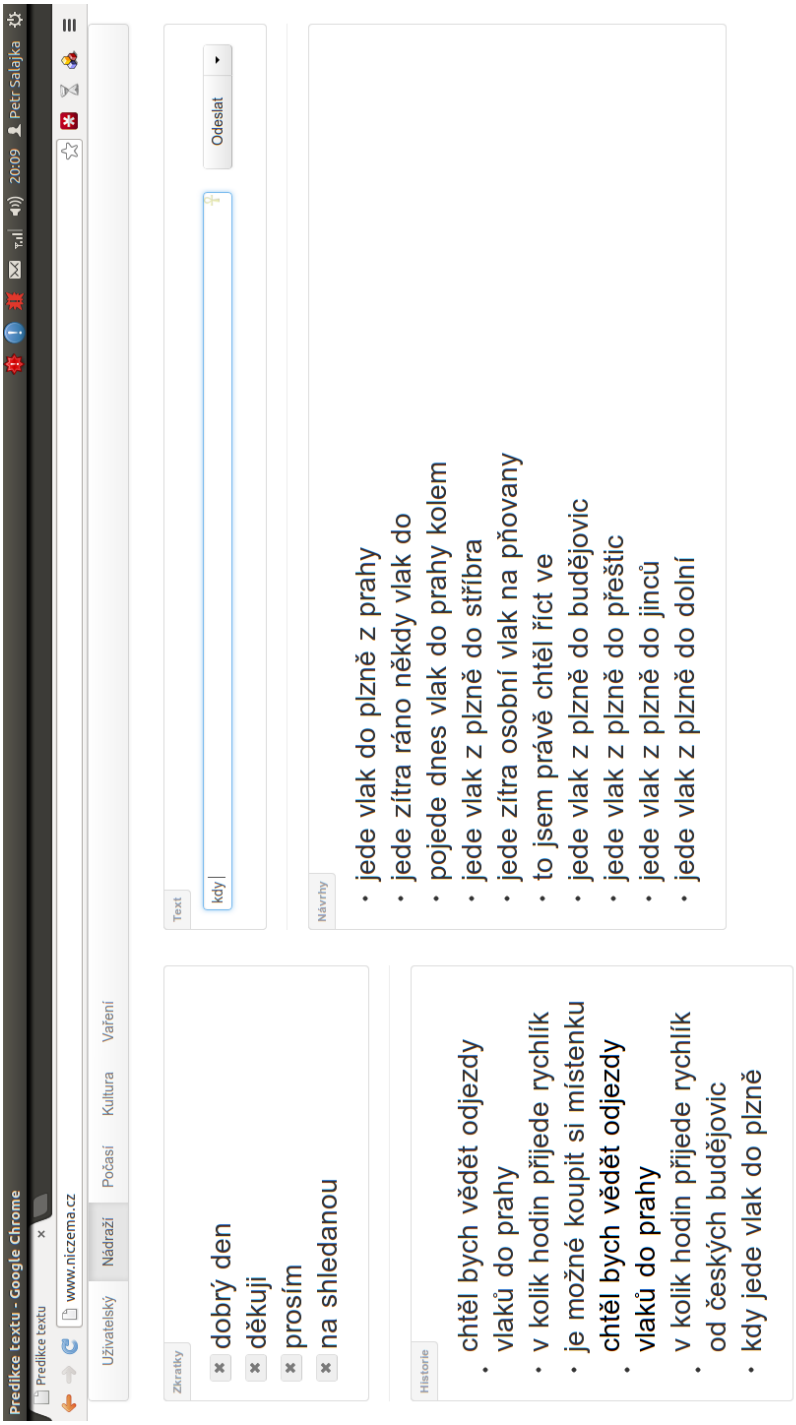
5.3 Klient - grafické uživatelské rozhraní

Grafické uživatelské rozhraní (obr. 5.3.1) je řešeno jako jednoduchá webová aplikace, komunikující se serverem pomocí protokolu WebSocket.

V horní části stránky je navigační lišta, kde jsou uvedeny dostupné jazykové modely. Aktivní model je vyznačen tmavší barvou a uživatel může mezi nimi jednoduše přepínat. Dále je stránka členěna na levou část (postranní panel) a pravou (hlavní panel).

Postranní panel se dále dělí na oblast zkratek a oblast historie. Zkratky si volí uživatel a jsou (krom jeho vlastních zásahů) neměnné. Oblast historie zobrazuje vždy posledních deset vět odeslaných k vyslovení.

Hlavní panel je rozdělen na oblast textu a oblast návrhů. V oblasti textu je editační pole, do kterého může klasickým způsobem psát. Vždy když vloží mezeru (stiskne mezerník - oddělovač slov), nabídne mu server další návrhy sekvencí slov, kterými by mohla věta pokračovat. Ty se zobrazí v oblasti „Návrhy“. Pokud je uživatel s větou spokojen, stiskne tlačítko „Odeslat“, zadaná věta je syntetizována do řeči, uloží se na první místo v oblasti „Historie“ a jsou zobrazeny nové návrhy sekvencí slov, kterými by mohla nová věta začínat.



Obrázek 5.3.1: Grafické uživatelské rozhraní

Uživatelské zkratky i historie se ukládají jako *cookie*[1]. Navíc se také jako cookie ukládá historie vět, které uživatel napsal, pro každý model zvlášť. Vzhledem k omezení maximální velikosti cookie na 4KB dat, je pro každý model uloženo vždy tolik posledních vět, které uživatel zadal, jejichž objem nepřesahuje tuto kvótu. Jedná se o sběr dat pro jednoduchou implementaci jazykového modelu s krátkodobou pamětí.

5.4 Adaptace na uživatele

Historie n posledních vět, které uživatel v rámci zvoleného jazykového modelu zadal, je vždy po její změně odeslána na server. Zde se z ní sestaví jazykový model a ten se pomocí `ngrammerge` spojí s již existujícím modelem. Pro generování návrhů je poté používáno toto spojení, čímž dochází k adaptaci jazykového modelu na uživatele.

5.5 Řešení duplicit sousloví v návrzích

Poměrně často dochází k jevu, kdy se v odpovědi na dotaz do modelu nachází sousloví jako

kdy
kdy jede vlak
kdy jede
kdy pojede vlak
kdy pojede vlak do

Zobrazení návrhů v tomto formátu je velmi neefektivní vzhledem k využití místa na stránce. Proto server před odesláním takové duplicity slučuje.

Dostupné návrhy sousloví se seřadí sestupně podle počtu slov, která obsahují. Následně je tento seznam procházen a kratší sousloví, která jsou již

[1] V protokolu HTTP označuje cookie malé množství dat, které se ukládá na počítači uživatele

obsažena v delších, jsou odstraněna. Zároveň se z vah odstraněného a ponechaného sousloví vezme ta lepší a ta je nastavena jako nová váha ponechaného sousloví.

Dále je pročištěný seznam seřazen podle vah a jako návrhy je uživateli odesláno n nejlepších a ty jsou zobrazeny. Prostor na stránce je tak mnohem lépe využit a navíc jsou krátká sousloví, která byla ohodnocena vysokou pravděpodobností uživateli stále k dispozici. Jakmile označí slovo kliknutím (v případě tabletu dotekem prstu), je do textového pole vložena část návrhu od začátku až po slovo, které zvolil. Původní seznam návrhů by tedy vypadal

kdy jede vlak
kdy pojede vlak do

a uživateli bude přesto umožněno zadat libovolné z výše uvedených sousloví.

Kapitola 6

Hodnocení kvality

6.1 Perplexita modelu

Pro výpočet perplexity modelu „Nádraží“ jsme sestavili 15 testovacích vět (69 slov), které by podle nás mohl uživatel chtít použít:

chtěl bych vědět jaké je spojení na prahu
dobrý den
v kolik přijíždí v sobotu večer rychlík
chtěla bych spojení z prahy na olomouc po deváté ráno
a v kolik jede další
děkuji
děkuji moc
děkuji vám
já bych potřebovala vlak do chebu
ano
ne
jede tam i osobní vlak
v kolik hodin jste říkala
v kolik jede nějaký vlak do prahy
v kolik hodin jede rychlík do stříbra

Nejnižší hodnotu perplexity jsme získali pro bigramový model a to

$$PP = 39{,}0203 \quad .$$

Sestavit reálný testovací korpus pro ostatní modely se nám jevilo poněkud problematické. I tak je ale vidět, že výsledek experimentu jde zcela proti závěru, vyvozeném z hodnot tab. 5.1.1.

Pro reálné zhodnocení prediktivních schopností aplikace tedy tato hodnota nemá téměř žádný význam a to ani ve smyslu volby řádu n-gramového modelu. Číslo, které jsme vypočetli říká, že při použití bigramového modelu budou generované návrhy pro uvedené věty nejlepší. To však navíc platí pouze pro první větu, kterou uživatel zadá. Hned na to totiž dojde k přepočítání modelu s právě zadanou větou (adaptaci) a pro další větu bude již hodnota perplexity a tím i kvalita predikce jiná.

Jako jediná možnost reálného zhodnocení aplikace se jeví subjektivní názor skupiny testovacích uživatelů, kteří by aplikaci používali.

6.2 Subjektivní hodnocení

Během experimentů se ukázalo, že prediktivní schopnosti aplikace jsou velmi dobré pro malé modely (modely „Uživatelský“ a „Nádraží“). Vzhledem k jejich velikosti je adaptace na uživatele poměrně rychlá a značná omezenost tématu v případě „Nádraží“ zajišťuje generování užitečných návrhů.

Bohužel v případě velkých modelů (viz kap. 5.1) je použitelnost aplikace mizivá. Vzhledem k velikosti modelů (a rozsahu jejich slovníků) je reakce serveru značně pomalá. Navíc poměr užitečných slov a slov výplňových způsobuje, že většinu předaných návrhů tvoří spojky, předložky a velmi běžná spojení jako „je to“. Aplikaci bude zjevně potřeba pro použití s modely s takovým rozsahem nastavit.

Kapitola 7

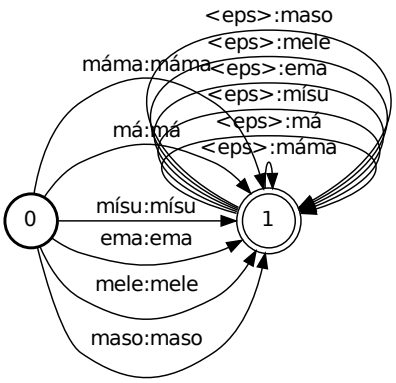
Další úlohy k řešení

Úlohou s jedno z nejvyšších priorit, jak ukázala kapitola 6, bude zřejmě hledání metody hledání metody porovnávání kvality použitých adaptujících se modelů. Budeme se také soustředit na nalezení metody pro vhodnou volbu řádu modelu.

7.1 Speciální symboly ve vážených konečných automatech

Jak již bylo řečeno v kap. 6.2, odezva serveru s rostoucí velikostí slovníku modelu značně klesá. Jedním z možných důvodů je způsob generování dotazů.

Dotaz, jehož kompozicí s modelem se predikce slov provádí, obsahuje jeden přechod pro každé slovo ze slovníku modelu. Tyto přechody jsou vytvářeny ve `FOR` cyklu, což způsobuje s rostoucím slovníkem značné výpočetní nároky (viz obr. 7.1.1). Tento problém by mohl být vyřešen pomocí *speciálních symbolů* [19].



Obrázek 7.1.1: Dotaz modelu na jedno slovo, pokud je historie prázdná (začátek nové věty)

Jedním ze speciálních symbolů se kterým jsme se již setkali je ϵ (epsilon). V případě kompozice dvou transducerů $T_1 \circ T_2$ může být přechod T_2 se vstupním symbolem ϵ realizován a to bez přijetí výstupního symbolu odpovídajícího přechodu T_1. Knihovna OpenFst (v použité verzi) podporuje celkem 4 takové speciální symboly.

Předpokládejme, že chceme provést kompozici transducerů $T_1 \circ T_2$, pak

- ϵ odpovídá prázdnému symbolu. Při kompozici může být přechod T_2 s ϵ na vstupu realizován bez přijetí jakéhokoliv symbolu odpovídajícího přechodu T_1. Důsledkem je, že dva řetězce jsou si rovny, pokud se liší jen v počtu a umístění symbolů ϵ. Přechody s ϵ na vstupu se tedy mohou, ale také nemusí realizovat, což vede k nedeterminizmu automatu [19].

- σ odpovídá při kompozici libovolnému symbolu. σ symbol na vstupu přechodu T_2 tedy přijme libovolný symbol na výstupu odpovídajícího přechodu T_1 a navíc, má-li přechod v T_2 σ i na svém výstupu, je přijatý symbol kopírován i na výstup kompozice [19].

- ρ odpovídá formulaci „`else`“. Pokud není možné v T_2 použít žádný jiný přechod, ρ výstupní symbol odpovídajícího přechodu T_1 přijme a navíc, má-li přechod v T_2 ρ i na svém výstupu, je přijatý symbol kopírován i na výstup kompozice [19].

- ϕ je obdobou symbolu ρ, ovšem na rozdíl od něj nepřijímá z výstupu T_1 žádný symbol. Realizuje se, pokud z daného uzlu není možné pokračovat jinou cestou [19].

Shoda při kompozici	Akceptuje symbol	Neakceptuje symbol
Shoduje se vždy	σ	ϵ
Pokud není jiná shoda	ρ	ϕ

Tabulka 7.1.1: Speciální symboly ve vážených konečných automatech [19, 20]

7.2 Modely založené na třídách slov

Přirozená řeč bývá značně proměnlivá. Mění se pořadí slov ve větě a ani pravidla jazyka nebývají příliš dodržována. Oproti tomu náš písemný projev bývá mnohem kultivovanější. To je důvod, proč by ke zlepšení prediktivních schopností systému mohlo přispět zapojení modelů založených na třídách slov; především se zařazením do tříd podle slovních druhů.

N-gramový model si dokáže s pořadím slov ve větě poměrně dobře poradit. Pokud byl v trénovacích datech dodržen správný slovosled a pokud je řád modelu n dostatečně vysoký, budou mít správný slovosled i predikovaná sousloví a to i vzhledem k již zadaným slovům.

Zcela jiná situace však nastává, pokud chceme vyřešit situaci zadání slova mimo slovník. Model řeší pouze pozici slova ve větě a pokud se tedy uživatel rozhodne vložit například přídavné jméno na pozici, kde se v trénovacích datech nikdy nenacházelo, bude predikce s největší pravděpodobností nesmyslná. Pokud bychom však měli k dispozici například opět n-gramový model modelující sled slovních druhů ve větě a dokázali neznámému slovu jeho slovní druh správně přiřadit, kvalita predikce by se mohla značně zlepšit.

7.3 Zvýhodnění dlouhých slov

Jak jsme již uvedli v kap. 6.2, při experimentech s většími modely jsme narazili na problém, kdy obrovská relativní četnost výplňových slov respektive

sousloví („je to“, „ono je“, „a ono“, „a tak“, atd.) smysluplnou predikci zcela znemožnila.

Chceme-li uživateli umožnit psát co nejrychleji, bylo by vhodné nabízet mu spíše slova delší; taková, která je obtížnější napsat. Bylo by tedy potřeba delší slova oproti těm kratším nějakým způsobem zvýhodnit. Informace o délce slov však není v našich modelech přímo dostupná. Bylo by tedy potřeba nalézt jinou metodu konstrukce jazykových modelů, která by ocenění slov na základě jejich délky umožňovala. Tento problém je v současné době otevřený.

Poznamenejme jen, že experiment s nastavením (krom maximální) také minimální délky hledaných sousloví nepřinesl žádné zřejmé výsledky.

Kapitola 8

Závěr

Pro vypracování dané úlohy jsme prostudovali metody statistické analýzy použitelné pro jazykové modelování. Byli použity n-gramové modely s vyhlazováním pomocí ústupového schématu. Konkrétně byl využit Wittenův-Bellův model. Pro záznam byla využita forma vážených konečných automatů, což umožňuje úpravy přímo za běhu a také další možnosti rozšíření aplikace do budoucna. Při experimentech s dalšími metodami vyhlazování, které použitá knihovna OpenGrm NGram nabízí, jsme nezaznamenali významné zlepšení kvality predikce.

Zkonstruovali jsme serverovou aplikaci, která na základě daného jazykového modelu a předložených slov navrhuje uživateli další možná slova. Počet návrhů a množství slov v nich je dáno kombinací nastavení aplikace a výsledku dotazu do modelu. Pro přístup k serveru jsme též sestavili jednoduché uživatelské rozhraní ve formě webové aplikace. Ta se serverem komunikuje pomocí WebSocket protokolu.

Věty, které uživatel zadá, jsou odesílány zpět na server, kde je počítačová syntéza převede na řeč. Tato řeč je následně jako zvukový signál uživateli přehrána. Navíc jsou tyto věty zaznamenávány a použity k úpravě jazykového modelu a jeho postupné adaptace na uživatele.

Objektivní vyhodnocení kvality predikce se však ukázalo jako značně problematické. Pro jeden z modelů jsme sice sestavili testovací korpus vět a vypočetli jeho perplexitu. Jak jsme však uvedli v kapitole 6.1, má tento výsledek

vzhledem k neustále probíhající adaptaci na uživatele pomíjivou hodnotu a ani vhodný řád modelu nelze tímto způsobem objektivně určit. Jedinou možností ohodnocení tak zůstává subjektivní názor uživatelů.

Ačkoliv je sestavená aplikace plně funkční a reálně použitelná, zůstává zde řada úloh, jejichž vyřešení by mohlo její funkcionalitu pozitivně ovlivnit.

Literatura

[1] Grant. Vedoucí: Prof. Ing. Psutka Josef, CSc. *Eliminace jazykových bariér handicapovaných diváků České televize II.* Zadavatel: TA ČR TA01030476, Dostupné z: `http://www.kky.zcu.cz/cs/grants` [cit. 2013-5-15].

[2] Grant. Vedoucí: Doc. Ing. Matoušek Jindřich, Ph.D. *Speciální vzdělávací pomůcky k podpoře výuky slabozrakých žáků.* Zadavatel: MŠMT CZ.1.07/1.2.31/02.0019. Období řešení: 2013–2015. Dostupné z: `http://www.kky.zcu.cz/cs/grants` [cit. 2013-5-15].

[3] Josef Psutka, Luděk Müller, Jindřich Matoušek, Vlasta Radová. *Mluvíme s počítačem česky.* Praha: Academia, 2006. 1. vyd. 752 s. ISBN 80-200-1309-1.

[4] Brian Roark, Richard Sproat, Cyril Allauzen, Michael Riley, Jeffrey Sorensen, and Terry Tai. The OpenGrm open-source finite-state grammar software libraries. In *Proceedings of the ACL 2012 System Demonstrations*, pages 61–66, Jeju Island, Korea, July 2012. Association for Computational Linguistics. `http://www.opengrm.org` Dostupné z: `http://www.aclweb.org/anthology/P12-3011` [cit. 2013-5-15].

[5] Yoshua Bengio, Réjean Ducharme, Pascal Vincent, and Christian Janvin. A neural probabilistic language model. *Journal of Machine Learning Research*, 3:1137–1155, 2003. Dosupné z: `http://www.jmlr.org/papers/v3/bengio03a.html` [cit. 2013-5-15].

[6] Cyril Allauzen, Michael Riley, Johan Schalkwyk, Wojciech Skut, and Mehryar Mohri. OpenFst: A general and efficient weighted finite-state transducer library. In *Proceedings of the Ninth International Conference on Implementation and Application of Automata, (CIAA 2007)*, volume 4783 of *Lecture Notes in Computer Science*, pages 11–23. Springer, 2007. Dostupné z: `http://www.openfst.org/twiki/pub/FST/FstBackground/ciaa.pdf` [cit. 2013-5-15].

[7] Mehryar Mohri, Manfred Droste, Werner Kuich and Heiko Vogler. *Weighted Automata Algorithms*, pages 213–254. Theoretical Computer Science, 2009. Dostupné z: `http://www.cs.nyu.edu/~mohri/pub/hwa.pdf` [cit. 2013-5-15].

[8] Definice ekvivalence - OpenFst Glosary, 2013. Dostupné z: `http://www.openfst.org/twiki/bin/view/FST/FstGlossary#EquivalentDef` [cit. 2013-5-15].

[9] Fernando C. N. Pereira Mehryar Mohri and Michael D. Riley. At&t fsm library^{TM} – finite-state machine library. Dostupné z: `http://www2.research.att.com/ fsmtools/fsm/man4/fsm.5.html` [cit. 2013-5-15].

[10] Openfst quick tour, 2013. Dostupné z: `http://www.openfst.org/twiki/bin/view/FST/FstQuickTour/` [cit. 2013-5-15].

[11] Minimize - dokumentace operací OpenFst, 2013. Dostupné z: `http://www.openfst.org/twiki/bin/view/FST/MinimizeDoc` [cit. 2013-5-15].

[12] Websocket.org, 2013. Oficiální stránka WebSocket protokolu. Dostupné z: `http://www.websocket.org/` [cit. 2013-5-15].

[13] Tomáš Valenta, Jan Švec, and Luboš Šmídl. Spoken dialogue system design in 3 weeks. In Petr Sojka, Aleš Horák, Ivan Kopeček, and Karel

Pala, editors, *Text, Speech and Dialogue*, volume 7499 of *Lecture Notes in Computer Science*, pages 624–631. Springer Berlin Heidelberg, 2012. Dostupné z: http://dx.doi.org/10.1007/978-3-642-32790-2_76 [cit. 2013-5-15].

[14] Jan Švec, Jan Hoidekr, Daniel Soutner, and Jan Vavruška. Web text data mining for building large scale language modelling corpus. In Ivan Habernal and Václav Matoušek, editors, *Text, Speech and Dialogue*, volume 6836 of *Lecture Notes in Computer Science*, pages 356–363. Springer Berlin Heidelberg, 2011. Dostupné z: http://dx.doi.org/10.1007/978-3-642-23538-2_45 [cit. 2013-5-15].

[15] Lucie Skorkovská, Pavel Ircing, Aleš Pražák, and Jan Lehečka. Automatic Topic Identification for Large Scale Language Modeling Data Filtering. *Text, Speech and Dialogue*, 6836:64–71, 2011. Dostupné z: http://dx.doi.org/10.1007/978-3-642-23538-2_9 [cit. 2013-5-15].

[16] Python programming language, 2013. Dostupné z: http://www.python.org/ [cit. 2013-5-15].

[17] pyopenfst - Python Bindings for the OpenFST Library, 2012. Dostupné z: https://code.google.com/p/pyopenfst/ [cit. 2013-5-15].

[18] Push operation documentation, 2013. Dostupné z: http://www.openfst.org/twiki/bin/view/FST/PushDoc [cit. 2013-5-15].

[19] Jan Švec. *Dizertační práce, KKY, Plzeň*. PhD thesis, KKY FAV ZČU v Plzni, 2013. Kapitola 5.2.5 Speciální symboly ve vážených konečných automatech.

[20] Matchers (openfst advanced usage), 2013. Dostupné z: http://www.openfst.org/twiki/bin/view/FST/FstAdvancedUsage♯Matchers [cit. 2013-5-15].