České vysoké učení technické v Praze
Fakulta jaderná a fyzikálně inženýrská

Využitie skrytých Markovových modelov pre počítačové rozpoznávanie hovorenej českej reči

(Recognition of Czech Speech Using Hidden Markov Models)

autor: Peter Sliacky
školiteľ: Ing. Pavel Bolek
2005/2006

–> Sem príde zadanie <–

Prehlasujem, že som predloženú prácu vypracoval samostatne s použitím literatúry a elektronických zdrojov, ktorých úplný zoznam je jej súčasťou.

V Prahe dňa 27. 5. 2006

_____

Peter Sliacky

Abstrakt

Ľudstvo sa už niekoľko desaťročí snaží o komunikáciu s počítačom v prirodzenom jazyku. Komunikácia smerom počítač – človek je už uspokojivo vyriešená. Opačný smer si razí svoju cestu, hlavný výskum sa deje na jazyku anglickom. Pre češtinu ani slovenčinu neexistuje zatiaľ žiadne implementované riešenie. Existujú pokusy, zaoberajúce sa napríklad rozpoznaním pár povelov v interiéri automobilu.

Podarilo sa mi úspešne implementovať rozpoznávač na báze skrytých Markových modelov pomocou HTK a vytvoriť prostredie pre jeho jednoduchú implementáciu niekým iným. Napísal som skripty na fonetickú transkripciu, podal príklad gramatiky pre rozpoznávanie jednoduchých matematických výrazov. Otestoval som parametre modelov vhodné na tréning foném. Tiež popisujem použité algoritmy a teóriu skrytých Markových modelov.

Na záver konštatujem, že pre rozumne zvolenú gramatiku je tu popísané rozpoznávanie dostatočne presné, problém presúvam do jazykovej roviny – s kvalitným jazykovým modelom by sa úspešnosť zdvihla.

Kľúčové slová: *rozpoznávanie reči, skryté Markovove modely, HTK, čeština, parametre modelu, Viterbi, Baum-Welch*

Abstract

Mankind has been trying to communicate with computers in a natural language for a few decades. The one-way communication between computer and man has already been satisfactorily resolved. The breakthrough in the reverse direction is on its way but the research is done mostly on English language. For either Czech or Slovak language there is no implemented solution.

I have successfully implemented the speech recognizer based on hidden Markov models using HTK and developed a framework for simple implementation by a third person. I have written scripts for phonetic transcription and also presented an example of grammar rules for simple mathematical statements recognition. Model parameters suitable for training of phonemes have also been tested. Moreover, I have described the applied algorithms and the hidden Markov model theory.

At the end of my thesis I state that the presented recognition model is satisfactorily accurate for a reasonably selected grammar. The problem of accuracy is on the language level – with sophisticated language model the accuracy would increase.

Keywords: *speech recognition, hidden Markov models, HTK, Czech language, model parameters, Viterbi, Baum-Welch*

Obsah

Kapitola 1

Úvod

„Poďme sa rozprávať.“ Alebo sa Vám zdá nereálne viesť zmysluplný dialóg s počítačom? Áno, zatiaľ to je nerálne, no snažíme sa k tomu priblížiť. Staviame systémy na rezerváciu leteniek, objednávanie jedla, robíme zápisy z jednaní.

V tejto práci popíšeme vrátane technických detailov, ako skonštruovať rozpoznávač. Ukážeme tiež jeden, nazvime ho použiteľný, príklad rozpoznávania matematických výrazov. Čitateľovi bude poskytnutá teória skrytých Markovových modelov, vysvetlenie tejto metodiky pri rozpoznávaní reči a nevyhnutne aj implementačná časť.

Na svoje si prídu najmä čitatelia technicky založení, spriatelení s linuxovým príkazovým riadkom a majúci dobrý pocit zo skriptov v PERLe. Myslím, že náročnosťou výkladu len miestami prekračujem znalosti prvého ročníka technickej školy. Snažil som sa, aby sa celý text dal stráviť na jedno kontinuálne prečítanie, bez nutnosti neustáleho nazerania na stránky predošlé.

Prajem príjemné čítanie!

1.1 Reč a jej obsah

Prv, než sa pustíme do rozpoznávania reči, popíšeme, čo rečou myslíme a akú má pre nás informačnú hodnotu.

1.1.1 Zvuk, hlas a reč

K reči sa dostaneme cez zvuk. Zvuk je pozdĺžne mechanické vlnenie hmotného prostredia s kmitočtom v rozmedzí približne 16 Hz do 20 kHz, ktoré pôsobí na ľudský sluchový orgán a vyvoláva v ňom subjektívny sluchový vnem.

Zvukové vlny sa od zdroja zvuku šíria v guľových vlnoplochách a s rastúcou vzdialenosťou slabne akustická energia – pokles akustického tlaku o 6 dB pri zdvojnásobení vzdialenosti. Hladina akustického tlaku vytváraného hlasom je pri bežnom rozhovore asi +30 dB, pri kriku zhruba +80 dB.

Hlas je zvuk vytváraný hlasivkami.

Reč je produkcia hlasu, ktorá tvorí lingvistické jednotky daného jazyka a používa sa na komunikáciu medzi rečníkom a poslucháčom.

1.1.2 Informačný obsah fonetickej formy

Za najmenšiu jednotku reči možeme považovať **foném**. Fonémy sa od seba odlišujú v spôsobe vytvorenia, keď je rečový trakt v inej konfigurácii. Sú to zároveň najmenšie jednotky medzi ktorými dokážeme vnímať rozdiely.

Podľa realizovných výskumov sa zistilo, že v existujúcich svetových jazykoch sa aktívne využíva len asi dvanásť akýchsi univerzálnych diferenciálnych príznakov. Ich spojením sa dosahuje definovaného fonému.

Počet fonémov v existujúcich svetových jazykoch sa pohybuje od 12 do 60. Napr. jazyk anglický ich má 42, ruský 40, v českom ich je 36. Fonémy sa spájajú do postupnosti prehovorených celkov, v ktorých môžme nájsť ďalšiu stavebnú jednotku — slabiku. Vyššou jednotkou je slovo, ktoré vzniká spojením slabík podľa pravidiel daného jazyka. Slovanské jazyky napr. používajú približne 2 500 – 3 500 slabík a 45 000 – 50 000 slov.

Pre jednoduchosť predpokladajme počet fonémov 32. Potom priemerná informácia obsiahnutá v jednom foném je

$$2^I = 32 \Rightarrow I = 5$$

bitov. Túto hodnotu môžme ešte znížiť zavedením relatívnych frekvencií výskytu jednotlivých fonémov (niečo ako kompresia na princípe častejšie

sa vyskytujúceho znaku) na približne 4,5 bit. Ďalším obmedzením je, že radenie jednotlivých fonémov je podriadené istým jazykovým pravidlám. Pri skúmaní dvojíc, trojíc fonémov, atď., bude priemerná informácia na jeden foném klesať k úrovni 3 – 3,5 bit.

Pri bežnom rozhovore vysloví človek asi 80 – 130 slov za minútu, čo predstavuje frekvenciu výskytu asi 10 fonémov za sekundu. Ak uvážime priemernú informáciu na jeden foném 3 – 4 bit, dostávame rýchlosť prenosu informácie 30 – 40 bit/s. Táto hodnota je aj v súlade s psychoakustickými testami, pri ktorých bolo zistené, že človek je schopný spracovať informáciu s rýchlosťou maximálne 50 bit/s.

1.1.3 Informačný obsah akustickej formy

Pri vysoko kvalitnom spracovaní signálu hovorenej reči je potrebné, vzhľadom na veľký frekvenčný rozsah frikatív, pracovať vo frekvenčnom pásme 8 – 10 kHz. Tomu zodpovedá frekvencia vzorkovania (Shannonov teorém) $F_V = 16 – 20$ kHz s 12 – 14 bitovým prevodom. Rýchlosť prenosu informácie je teda v tomto prípade $\approx 200\,000$ bit/s.

Vidíme teda obrovskú informačnú redundanciu akustického signálu v porovnaní s jeho fonetickou podobou. Táto redundancia je spôsobená najmä intonáciou a tempom reči, charakteristickým sfarbením hlasu, dialektom, defektami reči a pod.

Človek pri vnímaní akustického signálu potláča nepotrebné údaje v reči a to mu umožňuje zdôrazniť iba niekoľko hlavných zvukových príznakov. Tento proces zatiaľ nie je preskúmaný, preto sa snažíme využiť znalosti o procese tvorenia reči a odhadnúť tak, ktoré charakteristiky akustického signálu nesú potrebnú informáciu.

1.1.4 Proces vytvárania reči človekom

Zdrojom rečových kmitov, ktoré sú fyzikálnou reprezentáciou reči, sú ľudské rečové orgány. Tie sa skladajú z hlasiviek, dutiny hrdla, úst a nosa, podnebia, zubov a jazyka. K týmto orgánom je nutné pripočítať ešte fundamentálny zdroj hlasovej energie, t.j. pľúca a dýchacie svaly.

Zdrojom všetkých znelých zvukov sú kmitajúce hlasivky, ktoré sú umiestnené v hornej časti hrtanu. Priestor medzi hlasivkami tvorí hlasivkovú štrbinu. Ak človek mlčí, štrbina je otvorená, takže ňou vzduch voľne prechádza. Pri vytváraní jednotlivých rečových zvukov sa hlasivky zvierajú a roztáhujú. Pod tlakom vzduchu, ktorý vychádza z pľúc, stiahnuté hlasivky kmitajú.

1.2 Rozpoznávanie reči zo široka

Problém rozpoznávania reči je zložený z čiastkových úloh, počnúc analýzou signálu, výpočtu vhodných parametrov, zaraďovaním týchto parametrov do tried končiac syntaktickou a sémantickou analýzou.

Pod rozpoznávaním, vo všeobecnosti obrazov (to môžu byť skutočné obrázky, ale tiež rôzne objekty), sa rozumie zaraďovanie týchto obrazov do vhodných tried. My sa budeme zaoberať rozpoznávaním fonetických jednotiek, napríklad slov.

Frekvenčnou analýzou rečového signálu najprv každému prehovoru priradíme časovú postupnosť vektorov parametrov nazývaných **príznaky**. Túto postupnosť nazveme **obraz** prehovoru.

Rozpoznávanie slov potom spočíva v zaraďovaní ich obrazov do vopred definovaných tried. Z hľadiska aplikovaných metód rozpoznávania môžeme **klasifikátory izolovaných slov** rozdeliť do troch významnejších skupín:

- Prvú skupinu tvoria klasifikátory, v ktorých sa slovo spracováva ako celok, pričom je zaradené do tej triedy, ku ktorej vzorovému obrazu má najmenšiu vzdialenosť. Kľúčovou otázkou je tu určenie vzdialenosti medzi dvoma obrazmi slov. Táto vzdialenosť je obvykle určovaná na základe aplikácie metódy dynamického programovania, pri ktorej sa hľadá taká nelineárna transformácia časovej osi jedného z obrazov, pri ktorej dôjde k porovnaniu oboch obrazov s najmenšou výslednou vzdialenosťou.

- Pomerne novou skupinou sú systémy pracujúce na princípe neurónových sietí. V procese trénovania nastavíme parametre celej siete – naučíme ju ako reagovať na známe vstupy. Sieť potom dokáže asociovať a tak správne klasifikovať ďalšie slová.

- V systémoch ďalšej skupiny je prístup ku klasifikácii založený na štatistických metódach, v ktorých sú slová modelované pomocou tzv. skrytých Markovových modelov[1]. Pre každú triedu rovnakých slov zo slovníka sú potom v procese trénovania stanovené parametre modelu a neznáme slovo je klasifikované do tej triedy, ktorej model ho generuje s najväčšou pravdepodobnosťou.

V tejto práci sa budeme venovať použitiu skrytých Markovových modelov pre rozpoznávanie reči. Dôvody ich použitia sú nasledovné:

[1] Hidden Markov Model – HMM

- rozpoznávanie pomocou HMM je výpočetne jednoduchšie, rýchle a pomerne účinné,
- trénovanie HMM modelov vyšších jednotiek je možné zjednodušiť ich rozkladom na HMM modely elementárnejších jednotiek (fonémov, trifónov) a zreťazením týchto jednoduchých modelov,
- existujú vývojové nástroje na manipuláciu so skrytými Markovovými modelmi, jedným z nich je sada nástrojov HTK.

1.2.1 Z histórie

Už pred vyše 200 rokmi sa ľudia pokúšali o dialóg so strojom. Spočiatku bola snaha smerovaná k syntéze, keď bol roku 1779 zostrojený prvý mechanický rečový syntetizér. Bola to mašinka, ktorá púšťala zdroj vzduchu cez rôzne prekážky, ktoré následne vyludili tóny.

Postupne sa rozvíjali aj iné postupy, nápomocné pri analýze a syntéze. Veľkým prínosom bola diskrétna Fourierova transformácia, ktorej analógová verzia bola známa už od počiatku 18. storočia, no plne sa uplatnila až s nástupom počítačov. Písal sa rok 1969 a boli prvý krát aplikované metódy dynamického programovania pri klasifikácii slov. Najbližšie desaťročia sa metódy postupne vylepšovali, objavil sa ďalší prístup – modelovanie pomocou skrytých Markovových modelov.

Panoval všeobecný optimizmus, že už čoskoro sa podarí skonštruovať rozpoznávač porovnateľný s kvalitami ľudského posluchu. Lenže niektoré potiaže sa nepodarilo úplne eliminovať.

1.2.2 Komplikuje sa to

Hlavnými príčinami, prečo počítaču spôsobuje rozpoznávanie stále taký problém sú:

- Hlas jednej osoby sa líši od hlasu osôb iných. Je to spôsobené odlišnými parametrami hlasového ústrojenstva a spôsobom artikulácie. Každý človek má potom obvykle inú farbu hlasu, prízvuk, tempo reči a podobne.
- Hlas jedného rečníka môže byť odlišný v rôznych situáciách. Príčinou je najmä premenlivé časovanie, t.j. časovej dĺžky celého slova ako pomernej dĺžky jeho jednotlivých častí miernymi zmenami formantových frekvencií a podobne.

- Meniace sa akustické pozadie, t.j. prítomnosť šumu a rušenia. Problematickými v tomto prípade sú spracovanie slabých frikatív (f, č, ...) a identifikácia začiatku a konca slova.

Dokonca aj jeden rečník v priebehu dňa podľa svalovej tuhosti a únavy mení hlas. Je preto výzvou nájsť vhodné metodiky na elimináciu rušivých vplyvov a odfiltrovanie nedôležitej časti signálu.

Príčinou inej dimenzie je práca mozgu. Ten pracuje ako milióny procesorov s veľmi malou pamäťou. To je napríklad dôvod, prečo nedokážeme robiť rýchle výpočty z hlavy a tromfne nás kalkulačka zo 70-tych rokov. No pri úlohe rozpoznávania, nie je nám počítač rovnocenným partnerom.

Mozog dokáže robiť približné porovnania s tisíckami vzorov, dokáže odfiltrovať nepotrebnú informáciu a rýchlo tak zaklasifikovať aj dva veľmi podobné objekty.

1.2.3 Využitie

V dnešnej dobe (kalendár ukazuje rok 2006) sa komunikácia rečou so strojom používa najmä v úzko špecializovaných dialógových systémoch. Napríklad rezervácia leteniek alebo objednanie jedla. Rozpoznávacie gramatiky sú navrhnuté tak, aby v každom momente existovalo len pár možností, ktoré môžu byť vyslovené.

Rozvíjajúcou sa oblasťou je aj automatizovaný prepis televíznych správ a vytváranie zápisov zo súdnych jednaní. Samé o sebe by to bola náročná úloha, vypomôžeme si teda ďalším človekom – rečníkom, ktorý počuté hovorí stroju. Ten je natrénovaný na jeho hlas. Ako pomôcka pri eliminácii ruchov okolia a tiež eliminácii rušenia rečníkom slúži zariadenie nazývané stenomaska. Vyzerá to ako náustok, prikladá sa pred ústa a vnútri má mikfofón.

Týmto postupom sa dajú tvoriť titulky pre živé prenosy – trénovaný rečník hovorí komentár, ktorý sa ihneď prepisuje divákom ako textové titulky.

Rozpoznávanie sa objavuje aj v programoch na výuku jazykov. Študent sa učí výslovnosť a program ho kontroluje, ako mu to ide.

1.3 Jednoduchá rovnica

Celú úlohu rozpoznávania môžeme uvažovať ako výpočet

$$\underset{i}{\operatorname{argmax}} \{P(w_i|O)\} , \tag{1.1}$$

kde w_i je i-te slovo slovníka, O je rečová reprezentácia neznámeho slova. Hľadáme také i, ktoré maximalizuje pravdepodobnosť, že dané (neznáme) slovo je práve w_i ak na vstupe je O.

Poznamenajme, že výpočtom 1.1 vyriešime problém rozpoznávania.

Pravdepodobnosť $P(w_i|O)$ sa však nedá počítať priamo, použime teda Bayessovo pravidlo

$$P(w_i|O) = \frac{P(O|w_i)P(w_i)}{P(O)} \ .$$

V tomto vzťahu poznáme jednoduché pravdepodobnosti $P(w_i)$– pravdepodobnosť výskytu slova w_i a $P(O)$– pravdepodobnosť výskytu vzorky O (závisí od dimenzie vektoru O). Pravdepodobnosti $P(w_i)$ môžeme zjednodušene považovať za rovné $1/N$, to znamená, že každé z N slov má pravdepodobnosť výskytu v jazyku rovnakú. $P(O)$ nás tiež veľmi nezaujíma, funguje len ako normalizačný faktor.

Hlavným elementom rovnice je člen $P(O|w_i)$, čo vyjadruje pravdepodobnosť, že vyslovením slova w_i by vznikla rečová vzorka O. Predstavme si, že slová w_i sú rečové vzorky natrénovaných slov. Potom výpočet $P(O|w_i)$ je prakticky nerealizovateľný, kedže w_i nereflektuje rôzne deformácie vplývajúce na reč, napríklad tempo meniace sa vrámci slova, prízvuk, ruch na pozadí.

Východiskom je uvažovanie parametrického modelu rečovej produkcie, ktorý dokáže spomínané vplyvy potlačiť. Za predpokladu, že slovo w_i je určené parametrickým modelom λ_i, riešenie 1.1 získame výpočtom $P(O|\lambda_i)$.

Kapitola 2

Skryté Markovove modely

V tejto povinnej kapitole vysvetlíme matematickú teóriu skrytých Markovových modelov, objasníme prívlastok „skrytý“ a zdôvodníme ich použitie pri rozpoznávaní reči.

Poskytnuté budú aj algoritmy na trénovanie modelov a na ich použitie pri klasifikácii. V závere načrtnem, ako modely spájať a umožniť tak tréning na subslovných jednotkách.

Prosím čitateľa o povšimnutie si obrázkov modelov, ktoré majú apelovať na jeho umlecké cítenie nedbalou precíznosťou a majú odľahčiť túto matematickú kapitolu.

2.1 Princíp metódy

Prvé krôčky pri aplikovaní skrytých Markovovych modelov na problematiku rozpoznávania reči podnikol už začiatkom sedemdesiatych rokov Vintsyuk. Väčšieho rozšírenia sa metóda dočkala začiatkom osemdesiatych rokov, keď IBM prezentovalo svoj systém rozpoznávania reči na báze Markovovych modelov – Tangoru.

Princíp metódy modelovania reči Markovovými modelmi vychádza z predstavy o vytváraní reči. Pri artikulácii je celé hlasové ústrojenstvo v dostatočne krátkom okamihu (cca. 20 msec, t.j. 50 krát za sekundu) v jednej z konečného počtu stacionárnych polôh a dá sa opísať vhodnými parametrami. Tak ako sa menia polohy hlasového ústrojenstva, tak sa menia stavy Markovovho modelu – resp. jeho podporného reťazca.

Určite sa každý zarazil nad prívlastkom „**skrytý**“. Dôvod objasníme priamo na modele rečovej produkcie.

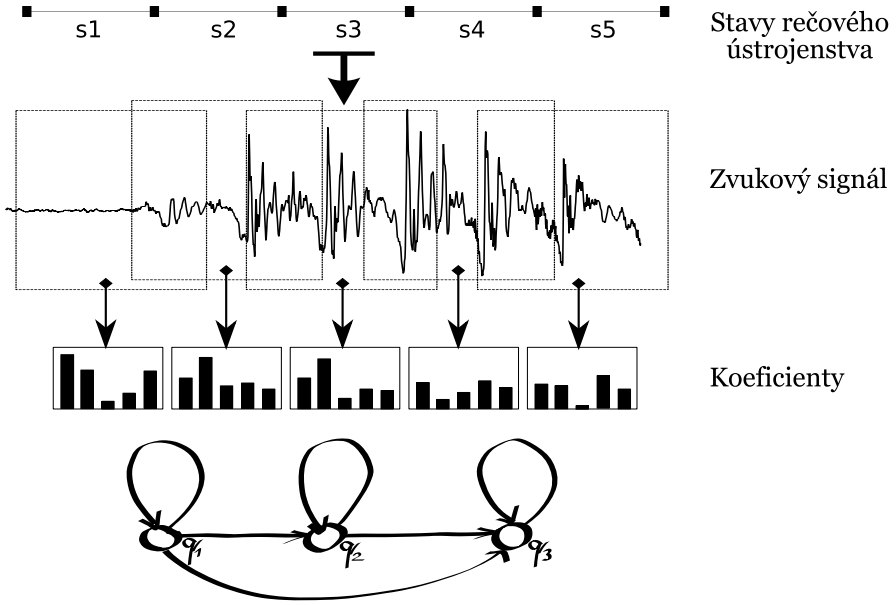
Na vyslovenie jednej hlásky o dĺžke 100 msec môžeme uvažovať 5 polôh rečového ústrojenstva. Postupom času prechádza ústrojenstvo týmito polohami a výstupom je rečový signál.

Skrytý Markovov model je snahou o modelovanie tohto javu. Prechádza zo stavu do stavu a v každom s určitou pravdepodobnosťou generuje výstupnú hodnotu. Pozorovateľ zvonku vidí len výstupné hodnoty. Stavy modelu, rovnako ako stavy rečového ústrojenstva sú mu skryté.

Opodstatnenie použitia skrytých Markovych modelov pri rozpoznávaní reči by mal načrtnúť obrázok 2.1. Zobrazuje proces vytvorenia jednej hlásky **rečovým ústrojenstvom**. To sa nachádzalo v stavoch s_1 až s_5 a bolo príčinou vzniku **zvukového signálu**. Tento sme naokienkovali piatimi okienkami, dosť veľkými, aby sme zabezpečili prekryv. V každom okne napočítame **koeficienty**, ktoré sa snažia odhadnúť parametre rečového ústrojenstva pri vyslovení danej vzorky.

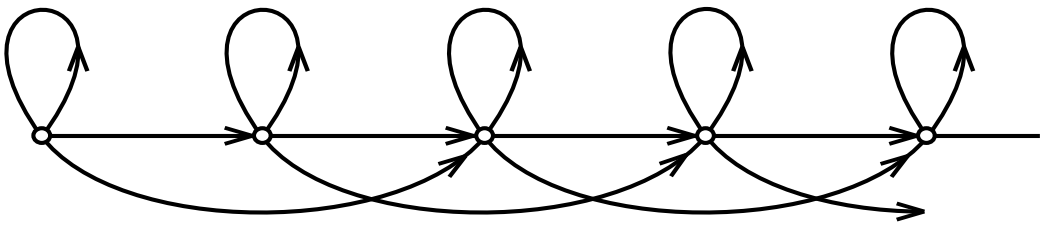
Predpokladajme teraz, že máme natrénované modely všetkých hlások. Spočítame, s akou pravdepodobnosťou ten ktorý model generuje spomínanú postupnosť koeficientov. Tu sa nám to uzatvára a dostávame model, ktorý najlepšie odhaduje stavy rečového ústrojenstva pri vytvorení zvukovej vzorky – to je hľadaný model hlásky.

Pri modelovaní reči s výhodou využívame tzv. **ľavo-pravé Markovove modely**, ktoré sa obzvlášť hodia na modelovanie javov spätých s postupujúcim časom. Ich základnou črtou je fakt, že v modele sa vždy postupuje od stavov s nižšími indexami k vyšším, tzn. ak usporiadame stavy zľava doprava, proces bude postupovať od ľavého (najnižšieho) indexu k pravému, nanajvýš sa v niektorom stave zdrží.



Obr. 2.1: Modelovanie rečovej produkcie

Pôvodný Vintsyukov model mal vysoký počet stavov (40-50), pričom jeden stav približne zodpovedal 10 msec. Bol určený na modelovanie slov, kde pri kratších slovách bolo možné preliezť modelom kratšou cestou (preskočením niektorých stavov) a naopak, pri dlhších sa bolo možné v niektorých stavoch zdržať.



Obr. 2.2: Vintsyukov typ Markovovho modelu slova

Neskôr boli urobené experimenty s menším počtom stavov a Vintsyukov model sa ukázal ako pomerne reduntantý. Počet stavov sa podarilo znížiť až na 5 bez významného poklesu presnosti.

Upozornime ešte, že Markovov model je štatistickým modelom a preto pri jeho trénovaní musíme pracovať so štatisticky významnou vzorkou, čo obvykle znamená natrénovať model každého slova desiatkami vzoriek.

2.2 Matematický základ metódy

Markovov proces G so skrytým Markovovým modelom môžeme vyjadriť päticou

$$G = (\boldsymbol{Q}, \boldsymbol{V}, \boldsymbol{N}, \boldsymbol{M}, \boldsymbol{\pi}),$$

kde:

- $\boldsymbol{Q} = \{q_1, ..., q_N\}$ je súbor N možných stavov Markovovho modelu
- $\boldsymbol{V} = \{v_1, ..., v_L\}$ je abeceda L možných výstupných symbolov[1]
- $\boldsymbol{N} = [n_{ij}]$ je matica prechodu, jej prvky určujú pravdepodobnosť prechodu systému zo stavu q_i do stavu q_j v ktoromkoľvek čase t. Platí:
$$n_{ij} = P(q(t+1) = q_j | q(t) = q_i), \;\; 1 \leq i, j \leq N$$
- $\boldsymbol{M} = [m_{jl}] = [m_j(l)]$ je matica pravdepodobností generovania vzorov, jej prvky určujú pravdepodobnosť generovania l-tého výstupného symbolu. Pre jednoduchosť môžme za výstupné symboly považovať písmená abecedy. Platí:
$$m_j(l) = P(v(t) = v_l | q(t) = q_j), \;\; 1 \leq j \leq N, 1 \leq l \leq L$$
- $\boldsymbol{\pi} = [\pi_i]$ je stĺpcový vektor pravdepodobností počiatočného stavu. Platí:
$$\pi_i = P(q(t) = q_i | t = 0)$$

Zaveďme ešte označenie λ pre súbor parametrov Markovovho modelu

$$\lambda = (\boldsymbol{N}, \boldsymbol{M}, \boldsymbol{\pi})$$

Poznamenajme ešte, že platia normalizačné podmienky

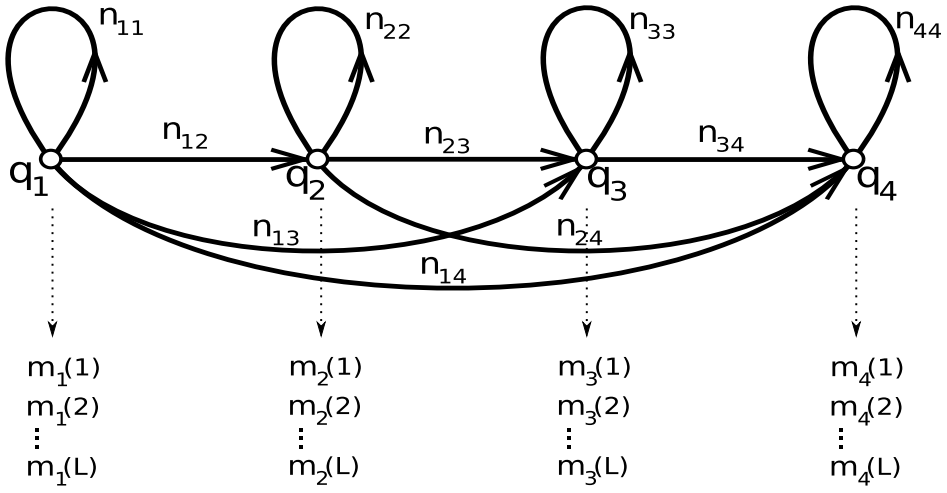
$$\sum_{i=1}^{N} \pi_i = 1$$

$$\sum_{j=1}^{N} n_{ij} = 1, \;\; i = 1..N$$

$$\sum_{l=1}^{L} m_j(l) = 1, \;\; j = 1..N \tag{2.1}$$

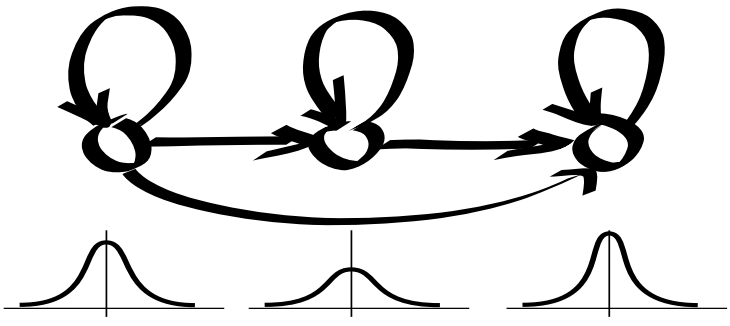
[1] Pre jednoduchosť predpokladajme, že model generuje postupnosť z konečnej sady L kvantizačných symbolov.

Pre ilustráciu ešte uveďme príklad 4-stavového Markovovho modelu, pre ktorý platia podmienky: $n_{ij} > 0$ pre $i \leq j$ a $n_{ij} = 0$ pre $i > j$ (t.j. nie sú povolené prechody sprava doľava). Čísla $m_j(l)$ určujú pravdepodobnosti generovania l-tého symbolu v stave j.



Obr. 2.3: Trojstavový HMM, spojitý prípad

Aby sme neboli až príliš upätý na rovnaký vzhľad modelu, na obrázku 2.4 je ilustrácia 3-stavového skrytého Markovovho modelu so spojitými distribúciami. To znamená, že v každom stave sa generuje vektor koeficientov podľa definovaného rozloženia pravdepodobnosti a stačí si pamätať parametre tohto rozloženia.



Obr. 2.4: Trojstavový HMM, spojitý prípad

Pri použití skrytých Markových modeloch nás bude zaujímať najmä to, ako ich prirovnať k pozorovanej neznámej vzorke a ako ich natrénovať.

2.3 Stanovenie pravdepodobnosti modelu

V tomto odstavci popíšeme, ako vypočítať pravdeopdobnosť, s akou pozorovanú reč, resp. jej obraz generuje konkrétny model λ. Určujeme teda $P(O|\lambda)$, kde pozorovaná rečová vzorka O je vyjadrená postupnosťou $O = \{o_1 o_2 ... o_T\}$, o_t je jeden z výstupných symbolov a T je dĺžka rečovej vzorky.

Pre stanovenie pravdepodobnosti $P(O|\lambda)$ bola navrhnutá metóda, ktorá využíva rekurzívneho výpočtu odpredu alebo odzadu generovanej postupnosti, tzv. forward-backward algorithm.

2.3.1 Forward-backward algorithm

Výpočet odpredu

Pri výpočte odpredu uvažujeme premennú $\alpha_t(i)$ definovanú ako pravdepodobnosť generovania čiastočnej postupnosti $\{o_1 o_2 ... o_t\}$ a zároveň stavu $q(t) = q_i$ pri danom modele λ. Definujme funkciu

$$\alpha_t(i) = P(o_1 o_2 .. o_t, q(t) = q_i|\lambda) \ .$$

Hodnoty $\alpha_t(i)$ je možné počítať rekurzívne:

1. Inicializácia
$$\alpha_1(i) = \pi_i m_i(o_1), \ \ 1 \leq i \leq N \ .$$

2. Rekurzia pre $t = 1, 2, ..., T-1$
$$\alpha_{t+1}(j) = \Big[\sum_{i=1}^{N} \alpha_t(i) n_{ij}\Big] m_j(o_{t+1}), \ \ 1 \leq j \leq N \ . \tag{2.2}$$

3. Výsledná pravdepodobnosť
$$P(O|\lambda) = \sum_{i=1}^{N} \alpha_T(i) \ .$$

Pripomeňme, že π_i je pravdepodobnosť, že počiatočný stav je práve i, $m_i(o_t)$ určuje pravdepodobnosť generovania symbolu o_t v stave i a n_{ij} je pravdepodobnosť prechodu zo stavu i do stavu j.

Finálna pravdepodobnosť $P(O|\lambda)$ potom určuje súčet čiastkových pravdepodobností, že bude vygenerovaná celá postupnosť $O = o_1 o_2 ... o_t$ a záverčný stav $i \in < 1, N >$.

Výpočet odzadu

Pri výpočte odzadu uvažujeme premennú $\beta_t(i)$ definovanú ako pravdepodobnosť generovania čiastočnej postupnosti $\{o_{t+1}o_{t+2}...o_T\}$ pri danom stave $q(t) = q_i$ a modele λ. Upozornime na rozdiely oproti výpočtu odpredu: generujeme postupnosť od $(t+1)$-vého indexu vyššie a stav $q(t) = q_i$ je známy a pevný. Definujme funkciu

$$\beta_t(i) = P(o_{t+1}o_{t+2}..o_T|q(t) = q_i, \lambda) \ .$$

Hodnoty $\beta_t(i)$ je opäť možné počítať rekurzívne:

1. Inicializácia
$$\beta_T(i) = 1, \ 1 \leq i \leq N \ .$$

2. Rekurzia pre $t = T-1, T-2, ..., 1$
$$\beta_t(i) = \sum_{j=1}^{N} n_{ij} m_j(o_{t+1}) \beta_{t+1}(j), \ \ 1 \leq i \leq N \ . \tag{2.3}$$

3. Výsledná pravdepodobnosť
$$P(O|\lambda) = \sum_{i=1}^{N} \pi_i m_i(o_1) \beta_1(i) \ .$$

Najpravdepodobnejší stav v danom čase

Pomocou pravdepodobností $\alpha_t(i)$ a $\beta_t(i)$ môžeme určiť aj najpravdepodobnejší stav Markovovho procesu v čase t. Definujme premennú $\gamma_t(i)$ ako pravdepodobnosť, že v čase t je proces v stave q_i

$$\gamma_t(i) = P(q(t) = q_i) = \frac{\alpha_t(i)\beta_t(i)}{P(O|\lambda)} \ . \tag{2.4}$$

Tu stačí uvážiť, že $\alpha_t(i)\beta_t(i)$ je pravdepodobnosť generovania čiastočnej postupnosti $\{o_1o_2...o_t\}$ a stavu $q(t) = q_i$, násobená pravdepodobnosťou, že za stavu q_i v čase t bude generovaná čiastočná postupnosť $\{o_{t+1}o_{t+2}...o_T\}$. Dokopy to teda dáva pravdepodobnosť generovania celej postupnosti s tým, že v čase t proces prechádzal stavom q_i.

$$P(O|\lambda) = \sum_{i=1}^{N} \alpha_t(i)\beta_t(i) \ .$$

V rovnici 2.4 má táto pravdepodobnosť význam normalizačného faktoru, ktorý zabezpečuje, aby

$$\sum_{i=1}^{N} \gamma_t(i) = 1 \ ,$$

tzn. pravdepodovnosť toho, že proces je v čase t v niektorom z N stavov bola rovná jednej.

Pomocou $\gamma_t(i)$ môžeme aj jednoducho určiť najpravdepodobnejší stav procesu v čase t, označme ho q_{t_i}. Platí:

$$i_t = \underset{i}{\text{argmax}} \left[\gamma_t(i)\right] \ , \quad i = 1..N, \, t = 1..T.$$

2.3.2 Viterbiho algoritmus

Výpočet pravdepodobnosti $P(O|\lambda)$ pomocou forward-backward algoritmu zahŕňal všetky možné postupnosti stavov $q(1), q(2), ..., q(T)$, v ktorých sa proces mohol nachádzať. Pravdepodobnosť generovania postupnosti O modelom λ môžeme alternatívne nahradiť výpočtom pravdepodobnosti optimálnej (maximálne pravdepodobnej) postupnosti stavov za predpokladu pozorovanej postupnosti O a modelu λ. Túto pravdepodobnosť – $P(O|\lambda)$, ako aj optimálnu postupnosť stavov $q(1), q(2), ..., q(T)$ je možné určiť za pomoci Viterbiho algoritmu.

Definujeme najprv veličinu $\delta_t(i)$ ako maximálnu pravdepodobnosť cesty DTW[2] v čase t, ktorá sa pozerá len na prvých t symbolov pozorovanej postupnosti ($o_1 o_2 ... o_t$). Index t nám označí koncový čas a i koncový stav:

$$\delta_t(i) = \max_{q(1),q(2),...,q(t-1)} P(o_1 o_2 .. o_t, q(1), q(2), .. q(t) = q_i|\lambda) \ .$$

Na tomto mieste ešte stojí za zmienku algoritmus DTW. Zjednodušene, je to algoritmus, ako názov napovedá, snažiaci sa o naťahovanie prípadne stláčanie časovej mierky pre zisk maximálnej zhody dvoch postupností.

Majme 2 postupnosti (vektorov) príznakov,

$$A = \{a_1, a_2, ..., a_n, ..., a_I\}$$

$$B = \{b_1, b_2, ..., b_m, ..., b_J\},$$

kde I a J je počet mikrosegmentov (t.j. vektorov príznakov) v slovách A a B.

Algoritmus s funkciou DTW hľadá v rovine (n, m) optimálnu cestu

$$m = \psi(n),$$

[2]Dynamic Time Warping

ktorá minimalizuje funkciu D celkovej vzdialenosti medzi postupnosťami A a B.

$$D(A,B) = \sum_{n=1}^{I} \hat{d}[a_n, b_{\psi(n)}],$$

pričom $\hat{d}[a_n, b_{\psi(n)}]$ je lokálna vzdialenosť medzi n-tým vektorom príznakov postupnosti A a m-tým ($m = \psi(n)$) vektorom príznakov postupnosti B. Požiadavky na túto lokálnu vzdialenosť sú symetria, pozitívna definitnosť a trojuholníková nerovnosť. Pre niektoré lokálne vzdialenosti sa dokonca vypúšťa požiadavka na symetriu.

Výsledkom tohto algoritmu je najlepšie napasovanie jednej postupnosti na druhú. V prípade Markovových modelov budeme triafať na seba postupnosť stavov $(q_1, q_2, ..., q_N)$ a postupnosť pozorovaných symbolov $(o_1 o_2 ... o_T)$. Hľadáme teda takú cestu modelom, ktorá najlepšie ohodnocuje pozorované symboly.

Všeobecný tvar Viterbiho algoritmu teda vyzerá:

1. Inicializácia

$$\begin{aligned} \delta_1(i) &= \pi_i m_i(o_1) \ , \\ \psi_1(i) &= 0 \ , \end{aligned}$$

2. Rekurzia pre $t = 2, 3, ...T$ a $j = 1, 2, ...N$

$$\begin{aligned} \delta_t(j) &= \max_i \left[\delta_{t-1}(i) n_{ij}\right] m_j(o_t) \ , \\ \psi_t(j) &= \operatorname*{argmax}_i \left[\delta_{t-1}(i) n_{ij}\right] \ , \quad i = 1..N \end{aligned}$$

3. Výsledná pravdepodobnosť a index maximálne pravdepodobného stavu v čase $t = T$ sa rovná

$$\begin{aligned} P^* &= \max_i \left[\delta_T(i)\right] \ , \\ i_t^* &= \operatorname*{argmax}_i \left[\delta_T(i)\right] \ , \quad i = 1..N \end{aligned}$$

Funkcia $\psi_t(j)$ si počas kroku 2 ukladala stavy, ktorými prechádzal algoritmus DTW. Preto optimálnu postupnosť stavov určíme jednoducho spätným stopovaním (backtracking) tejto funkcie. Pre $t = T-1, T-2, ..., 1$ môžeme indexy hľadaných stavov určiť zo vzťahu

$$i_t^* = \psi_{t+1}(i_{t+1}^*) \ .$$

2.4 Trénovanie parametrov modelu

V predchádzajúcej sekcii sme opísali, ako stanovit' pravdepodobnosť, s ktorou daný model generuje pozorovanú postupnosť. Teraz si ukážeme, ako tento model $\lambda = (\boldsymbol{N}, \boldsymbol{M}, \boldsymbol{\pi})$ natrénovat' tak, aby ním bola maximalizovaná pravdepodobnosť generovania pozorovanej postupnosti O.

Zatiaľ nebola predložená žiadna analytická metóda, ktorá by zaručovala dosiahnutie globálneho maxima tejto pravdepodobnosti. Existujú ale iteratívne postupy, ktoré zaručujú aspoň lokálne maximum. Najznámejším a najpoužívanejším je Baum-Welchov algoritmus. Jedná sa o reestimačný proces (cyklicky opakovaný odhad), pre ktorý je možné na základe trénovacích dát a momentálneho modelu určit' nový odhad $\hat{\lambda} = (\hat{\boldsymbol{N}}, \hat{\boldsymbol{M}}, \hat{\boldsymbol{\pi}})$.

2.4.1 Baum-Welchov algoritmus

Pre popis tohto algoritmu si definujeme premmennú $\xi_t(i,j)$, ktorá vyjadruje pravdepodobnosť, že daný proces prejde v intervale t až $t+1$ zo stavu q_i do stavu q_j za predpokladu pozorovanej postupnosti O a daného modelu λ.

$$\xi_t(i,j) = P(q(t) = q_i, q(t+1) = q_j | O, \lambda) \ .$$

Ďalej platí, že $P(O|\lambda)\xi_t(i,j)$ je pravdepodobnosť generovania postupnosti O za prepokladu modelu λ a podmienky, že v intervale t až $t+1$ proces prechádzal zo stavu q_i do stavu q_j. Z toho plynie

$$\xi_t(i,j) = \frac{\alpha_t(i) n_{ij} m_j(o_{t+1}) \beta_{t+1}(j)}{P(O|\lambda)} \ . \tag{2.5}$$

Ako sme si mohli uvedomit' už vyššie, $P(O|\lambda)$ tu má funkciu normalizačného faktoru.

Pre ďalší vzťah si požičiame rovnice 2.3, 2.4 a 2.5. Dosadíme 2.3 do 2.4 a nahradíme zlomok v sume vzťahom 2.5

$$\begin{aligned}
\gamma_t(i) &= \frac{\alpha_t(i) \sum_{j=1}^{N} n_{ij} m_j(o_{t+1}) \beta_{t+1}(j)}{P(O|\lambda)} \\
\gamma_t(i) &= \sum_{j=1}^{N} \frac{\alpha_t(i) n_{ij} m_j(o_{t+1}) \beta_{t+1}(j)}{P(O|\lambda)} \\
\gamma_t(i) &= \sum_{j=1}^{N} \xi_t(i,j) \ .
\end{aligned}$$

Posledná rovnosť vyjadruje nasledovný fakt: Pravdepodobnosť, že proces je v čase t v stave i dostaneme ako sumáciu cez všetky pravdepodobnosti prechodov z pevného stavu i do ľubovoľného stavu j v čase t.

Ak prevedieme sumáciu $\gamma_t(i)$ podľa času ($t = 1..T-1$), dostaneme veličinu, ktorú môžeme interpretovať ako očakávaný počet prenosov realizovaných **zo** stavu q_i (do ľubovoľného iného). Obdobnú časovú sumáciu veličiny $\xi_t(i,j)$ by sme mohli interpretovať ako očakávaný počet prenosov zo stavu q_i do stavu q_j.

Teraz už môžeme vysloviť vzťahy pre reestimáciu parametrov Markovovho modelu $\hat{\lambda} = (\hat{\boldsymbol{N}}, \hat{\boldsymbol{M}}, \hat{\boldsymbol{\pi}})$:

- $\hat{\pi}_i$ sa nechá definovať ako pomer očakávaného počtu stavov q_i v čase $t = 1$ k celkovému počtu všetkých možných stavov v čase $t = 1$

  $$\hat{\pi}_i = \gamma_1(i) \ ,$$

  Pripomeňme len, že veličinou $\gamma_t(i)$ sme označili pravdepodobnosť, že proces je v čase t v stave q_i.

- $\hat{n}_{ij}$ môžeme definovať ako pomer očakávaného počtu prechodov zo stavu q_i do stavu q_j k očakávanému počtu prechodov modelu zo stavu q_i (do ľubovoľného stavu)

  $$\hat{n}_{ij} = \frac{\sum_{t=1}^{T-1} \xi_t(i,j)}{\sum_{t=1}^{T-1} \gamma_t(i)} \ , \tag{2.6}$$

- $\hat{m}_j(l)$ definujeme ako pomer očakávaného počtu okamihov, keď je model v stave q_j a zároveň je generovaný l-tý symbol (v_l) k očakávanému počtu okamihov, keď je model v stave q_j

  $$\hat{m}_j(l) = \frac{\sum_{t=1, o_t = v_l}^{T} \gamma_t(j)}{\sum_{t=1}^{T} \gamma_t(j)} \ .$$

Baum dokázal, že pre každý nový odhad $\hat{\lambda} = (\hat{\boldsymbol{N}}, \hat{\boldsymbol{M}}, \hat{\boldsymbol{\pi}})$ parametrov modelu vždy platí $P(O|\hat{\lambda}) > P(O|\lambda)$ až na prípad dosiahnutia optimálneho nastavenia parametrov, t.j. $P(O|\hat{\lambda}) = P(O|\lambda)$. Ako sme ale poznamenali v úvode, Baum-Welchovým algoritmom sa dosahuje len lokálne maximum, preto je dôležitá vhodná voľba počiatočných hodnôt modelu.

2.4.2 Normalizácia

Pri výpočte premmených $\alpha_t(i)$ a $\beta_t(j)$ vo forward-backward algoritme narážame často na problém s aritmetikou počítača – hodnoty sú blízke nule.

Preto zavádzame normalizáciu premenných $\alpha_t(i)$ a $\beta_t(j)$, ktorou sa problému s malými hodnotami vyhneme.

Definujme koeficient

$$C_{t+1} = \frac{1}{\sum_{i=1}^{N} \alpha_{t+1}(i)} , \tag{2.7}$$

ktorým vynásobíme premennú $\alpha_{t+1}(i)$

$$\hat{\alpha}_{t+1}(j) = C_{t+1}\alpha_{t+1}(j) \tag{2.8}$$

kde $\hat{\alpha}_{t+1}(j)$ sme označili normalizovanú premennú. Pre efektívnejší výpočet zavedieme normalizačné koeficienty c_t, pre ktoré platí

$$C_t = \prod_{\tau=1}^{t} c_\tau , \quad t = 1..T$$

Odvodíme 2 vzťahy, pomocou ktorých sa dajú napočítat'normalizované premenné $\hat{\alpha}_{t+1}(j)$. Najprv si uvedomíme, že

$$c_{t+1} = \frac{C_{t+1}}{C_t} ,$$

dosadením 2.7 získame

$$c_{t+1} = \frac{1}{C_t[\sum_{i=1}^{N} \alpha_{t+1}(i)]} ,$$

C_t vtiahneme do sumy a dosadíme 2.2,

$$c_{t+1} = \frac{1}{\sum_{i=1}^{N}\sum_{j=1}^{N} \frac{\alpha_t(i)}{C_t} n_{ij} m_j(o_{t+1})} .$$

Napokon užijeme 2.8, tým dostaneme prvý zo sľubovaných vzťahov vedúcich k poznaniu $\hat{\alpha}_{t+1}(j)$:

$$c_{t+1} = \frac{1}{\sum_{i=1}^{N}\sum_{j=1}^{N} \hat{\alpha}_t(i) n_{ij} m_j(o_{t+1})} . \tag{2.9}$$

Samotné $\hat{\alpha}_{t+1}(j)$ potom vypočítame zo vzťahu 2.8 roznásobením $\frac{C_t}{C_t}$ a dosadením 2.2:

$$\hat{\alpha}_{t+1} = c_{t+1}\Big[\sum_{i=1}^{N} \hat{\alpha}_t(i) n_{ij}\Big] m_j(o_{t+1}) .$$

Obdobným postupom sa nechajú určiť normalizované premenné $\hat{\beta}_t(i)$, pre $t = T, T-1, ..., 1$. Tu je ale rozdiel v tom, že využívame normalizačné koeficienty c_t napočítané skôr. Platí

$$\hat{\beta}_t(i) = (\prod_{\tau=t}^{T} c_\tau)\beta_t(i) = D_t\beta_t(i). \tag{2.10}$$

Ešte zostáva overiť, že použitím normalizovaných premenných o nič neprídeme a odhady parametrov Markovovho modelu budú rovnako dobré. Vyjdeme zo vzťahu pre odhad parametru $\hat{n}_{ij}$ (2.6), dosadíme doň za $\gamma_t(j)$ 2.4 a za $\xi_t(i,j)$ 2.5. Celé to označíme n^*_{ij}:

$$n^*_{ij} = \frac{\sum_{t=1}^{T-1} \frac{\alpha_t(i) n_{ij} m_j(o_{t+1}) \beta_{t+1}(j)}{P(O|\lambda)}}{\sum_{t=1}^{T-1} \frac{\alpha_t(i)\beta_t(i)}{P(O|\lambda)}} .$$

Členy $P(O|\lambda)$ sa vyrušia a za $\beta_t(i)$ v menovateli dosadíme 2.3:

$$n^*_{ij} = \frac{\sum_{t=1}^{T-1} \alpha_t(i) n_{ij} m_j(o_{t+1}) \beta_{t+1}(j)}{\sum_{t=1}^{T-1} \alpha_t(i) \sum_{j=1}^{N} n_{ij} m_j(o_{t+1}) \beta_{t+1}(j)}$$

Použitím vzťahov 2.8 a 2.10 nahradíme $\alpha_t(i)$ a $\beta_{t+1}(j)$

$$n^*_{ij} = \frac{\sum_{t=1}^{T-1} \frac{\hat{\alpha}_t(i)}{C_t} n_{ij} m_j(o_{t+1}) \frac{\hat{\beta}_{t+1}(j)}{D_{t+1}}}{\sum_{t=1}^{T-1} \sum_{j=1}^{N} \frac{\hat{\alpha}_t(i)}{C_t} n_{ij} m_j(o_{t+1}) \frac{\hat{\beta}_{t+1}(j)}{D_{t+1}}} .$$

Už stačí vyrušiť členy vyskytujúce sa v čitateli aj menovateli

$$C_t D_{t+1} = \prod_{\tau=1}^{t} c_\tau \prod_{\tau=t+1}^{T} c_\tau = C_T$$

a dostaneme výsledný tvar, ktorý formálne zodpovedá počítaniu s normalizovanými premennými oproti pôvodným, čo znamená, že $n^*_{ij} = \hat{n}_{ij}$.

To znamená, že použitím normalizovaných premenných skutočne „o nič neprídeme“ a vyhneme sa problémom s aritmetikou počítača. Pre úplnosť dodajme, že pomocou normalizovaných premenných je možné odhadovať aj $\hat{m}_j(l)$.

2.4.3 Problém chýbajúcich dát

Všeobecne sa pre kvalitné natrénovanie Markovovho modelu odporúča veľa rečových vzoriek každého typu, tzn. napríklad pri trénovaní modelov

slov viac vzoriek každého slova. Často sa stáva (najmä pri modelovaní zreťazených menších jednotiek), že trénovacia množina nepokryje dostatočne všetky možné výskyty zvolených jednotiek.

Obtiaž je spôsobená nulovými prvkami v matici $\boldsymbol{M}$. Problém sa rieši prepočítaním tejto matice za splnenia podmienky 2.1 nasledovne: pre všetky prvky matice $\boldsymbol{M}$ menšie ako vhodne zvolená konštanta κ, nastavíme tieto na hodnotu κ. Ostatné prvky potrebujeme prepočítať aby bola splnená spomenutá podmienka. Platí teda

$$\tilde{m}_j(l) = \frac{(1 - s\kappa)m_j(l)}{\sum_{i=1}^{N-s} m_j(i)} ,$$

kde s je počet prvkov matice $\boldsymbol{M}$ menších ako κ. Tento vzťah len hovorí toľko, že zvyšné prvky majú dokopy dať pravdepodobnosť zníženú u pôsobenie „κ-prvkov".

2.5 Klasifikácia

Prešli sme procesom trénovania modelov a tiež vieme, ako vypočítať pravdepodobnosť, že pozorovaná postupnosť je generovaná daným Markovovým modelom s parametrami $\lambda = (\boldsymbol{N}, \boldsymbol{M}, \boldsymbol{\pi})$.

Predpokladajme, že máme natrénovaných R modelov. Úlohou klasifikátora je teraz rozhodnúť o tom, do ktorej triedy zaradiť pozorovanú postupnosť O. Pre každý model λ_r ($r = 1..R$) sa vypočíta pravdepodobnosť $P(O|\lambda_r)$. Klasifikátor jednoducho vyberie maximálnu pravdepodobnosť a pozorovanú postupnosť zaradí do príslušnej triedy.

Výpočet pravdepodobností môžme previesť štandardným algoritmom forward-backward. Lenže aj v tomto prípadne, podobne ako pri Baum-Welchovej reestimácii, hrozí reálne nebezpečenstvo, že narazíme na problém s aritmetikou počítača pri nízkych hodnotách $\alpha_t(i)$ a $\beta_t(i)$.

Tu nám opäť pomôžu normalizačné koeficienty c_t počítané podľa 2.9. Platí:

$$\begin{aligned}
\sum_{i=1}^{N} \hat{\alpha}_T(i) &= 1 \\
C_T \sum_{i=1}^{N} \alpha_T(i) &= 1 \\
C_T P(O|\lambda) &= 1 \\
\Big[\prod_{t=1}^{T} c_t\Big] P(O|\lambda) &= 1
\end{aligned}$$

Osamostatníme $P(O|\lambda)$ a zlogaritmujeme:

$$\log P(O|\lambda) = \log\Big[\prod_{t=1}^{T} c_t\Big]^{-1} = -\sum_{t=1}^{T} \log c_t.$$

Problémom so spracovaním malých čísel pri výpočte pravdepodobností sa vyhneme ak použijeme modifikovanú verziu Viterbiho algoritmu. Pôvodný algoritmus pracuje len s operáciami súčinu a výberu maxima, preto je pomerne jednoducho transformovateľný na operácie súčtov logaritmov.

2.5.1 Modifikovaný Viterbiho algoritmus

1. Inicializácia
$$\Phi_1(i) = \log \pi_i + \log m_i(o_1) \ ,$$

2. Rekurzia pre $t = 2, 3, ...T$ a $j = 1, 2, ...N$
$$\Phi_t(j) = \max_i \left[\Phi_{t-1}(i) + \log n_{ij}\right] + \log m_j(o_t) \ ,$$

3. Výsledná pravdepodobnosť
$$\log P^* = \max_i \left[\Phi_T(i)\right] \ ,$$

   kde vo všetkých vzťahoch $i = 1..N$.

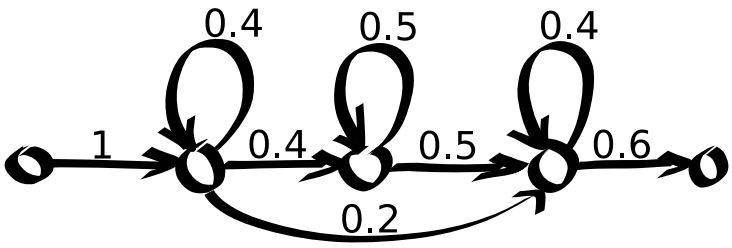
Poznamenajme, že hodnoty $\log n_{ij}$ môžu byť vypočítané vopred a pokiaľ pracujeme s konečným počtom diskrétnych parametrov $m_j(l)$ tak aj všetky hodnoty $\log m_j(o_t)$.

2.5.2 Zreťazenie modelov

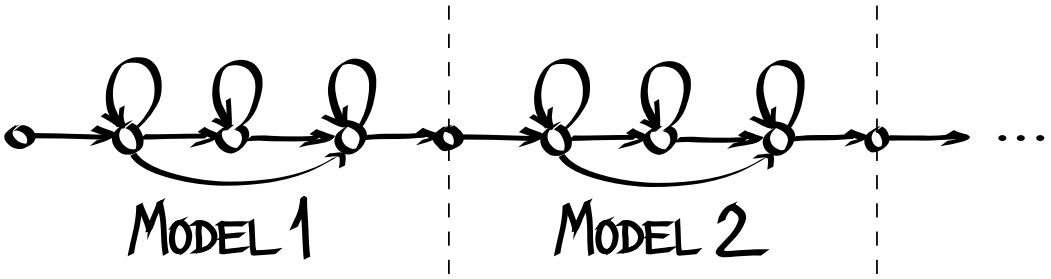
Pôvodným cieľom bol tréning subslovných jednotiek, no vyslovovaných prirodzene v slovách. Preto nevyhnutnou súčasťou tréningu aj rozpoznávania sa stalo spájanie príslušných modelov aby tvorili jeden väčší skrytý Markovov model. Pri tréningu vždy spojíme modely foném, aby tvorili celé slovo a trénujeme tento „jeden“ model. Modely rozpojíme a následne spojíme inak, aby formovali ďalšie slovo.

Početným opakovaním získame dobre naladené modely foném.

Aby sme vytvorili ostrejšie hranice medzi modelmi, pridáme technický prvok – počiatočný a koncový stav, ktoré budú neemitujúce a ich úloha je



Obr. 2.5: Pridanie počiatočného a koncového stavu



Obr. 2.6: Spojenie modelov

slúžiť pri spájaní modelov. Obrázok 2.5 ukazuje, ako vyzerajú tieto pridané stavy a ako nastavíme príslušné prechodové pravdepodobnosti.

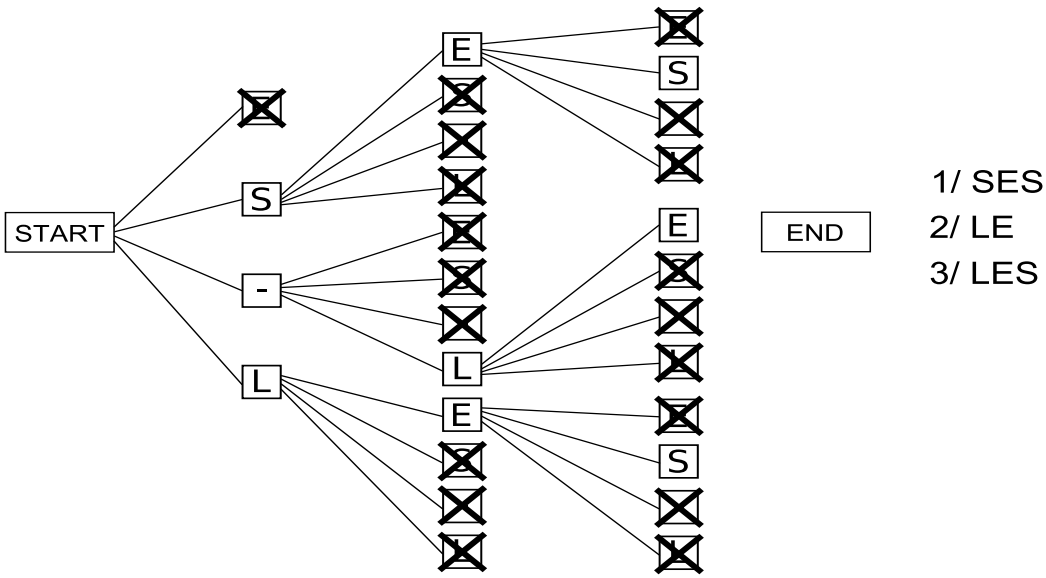
Spojenie samotné realizujeme stotožnením koncového stavu modelu č.1 s počiatočným stavom modelu č.2, ako ukazuje obrázok 2.6.

Teraz sme ale postavený pred ďalší problém. Máme natrénovaných N foném a chceme rozpoznať neznáme slovo. Skúšať všetky kombinácie by bolo výpočtovo neúnosné. Naznačím, ako sa tento problém rieši metódou zamietania neperspektívnych ciest.

Popíšem prechod rozpoznávacou sieťou pre gramatiku zloženú z písmen {`E, S, L, -`} (pomlčka značí pauzu). Gramatika je úplne voľná, sú povolené prechody z každého do každého písmena, nie je obmedzená ani dĺžka slov.

Pozeráme sa na obrázok 2.7 a začíname v políčku `START`. Rozvinieme všetky možnosti a spočítame, s akou pravdepodobnosťou by dané modely hlások odhadovali úvodný úsek neznámej postupnosti. Každý z modelov v tomto strome ohodnotíme a najlepší zvolíme za referenčný. Tie z modelov, ktoré sú príliš nepravdepodobné – referenčná pravdepodobnosť mínus prah – zamietneme.

Zvyšné modely rozvinieme podľa gramatiky ďalej a proces opakujeme až do momentu, keď sme dočítali vstupnú postupnosť. Potom sa pozrieme,



Obr. 2.7: Rozpoznávacia sieť

ktoré uzly prežili až do konca a prečítame, aké slová tvoria. Na obrázku 2.7 vytrvali do konca slová `SES, LE, LES`.

Neznamená to, že by sme sa nevedeli rozhodnúť, tvrdíme len toľko, že tieto tri slová sú dostatočne pravdepodobné (nepadli pod prah zamietnutia). To zohľadníme v kontexte širšej gramatiky.

Vysvetlili sme teóriu skrytých Markovových modelov, dôvod ich použitia, algoritmy na prácu s nimi. Ale či to skutočne funguje uvidíme, až pri pokuse o implementáciu.

Kapitola 3

Konštrukcia rozpoznávača

Od teraz až do konca práce bude nasledovať popis mojich pokusov a skriptov, vedúcich k postaveniu rozpoznávača. V úvode je zrejmá inšpirácia tutoriálom v [Htk05], ktorý sa stal veľmi užitočným v počiatočnej fáze.

Kapitola sa snaží popísať metodiky až k technickým detailom a tiež je snahou podať náhľad zvrchu – od spracovania dát z mikrofónu po konečné rozpoznanie. Podávam návod, ako skonštruovať rozpoznávač s rozumnou gramatikou. Vyprodukoval som množstvo skriptov, ktorých význam bude objasnený a ak niekomu poslúžia, radosť na mojej strane.

Ako výhodná sa ukazuje znalosť linuxového príkazového riadku, ktorý nás bude správadzať celú kapitolu. Príkazy a výpisy súborov som pre lepšiu orientáciu typograficky oddelil.

Posledných pár strán je venovaných experimentom s parametremi modelu. Očakáva sa, že nasledovná kapitola bude stráviteľnejšia ako predošlá.

3.1 Cieľ

V úvode pochopiť skryté Markovove modely a ich význam pri rozpoznávaní reči. Aplikovať znalosti a skonštruovať reálne použiteľný rozpoznávač. Napokon vyladiť parametre modelu a rozpoznávača pre rozpoznávanie izolovaných slov s veľkosťou slovníka cez tisíc slov a potlačiť závislosť na jednom rečníkovi.

3.2 Prostriedky

Rozpoznávač na báze skrytých Markovových modelov. Ponúkalo sa začať programovať na kolene, ale už po úvodných pokusoch a zisteniach, že tadeto cesta nevedie, som to zavrhol a poobzeral sa po niečom hotovom.

Skryté Markovove modely sú pomerne populárnou záležitosťou, preto existuje viacero knižníc pre prácu s nimi. Jednou z nich je aj HTK[1]. Okrem knižníc ponúka aj utility, ktoré s nimi pracujú. Je to dobre zdokumentovaný multiplatformný projekt, knižnice a utility sa šíria v podobe zdrojových kódov v jazyku C. Cieľom projektu je poskytnúť vývojárom prostredie na experimentovanie so skrytými Markovovými modelmi. Viac informácií na stránkach projektu [Htk].

V licenčnej dohode na HTK sa zaväzujeme nešíriť akékoľvek súčasti distribúcie HTK. Inak máme povolené robiť kópie, meniť zdrojové kódy, používať ich v iných produktoch, výlučne však vrámci svojej organizácie.

Utility projektu HTK sú pomenované podľa konvencie `HNazov`, kde `Nazov` je určením funkcie prípadne implementované algoritmu.

3.3 Začiatky

Prvé kroky boli nesmelé, všade okolo seba som chŕlil nepresnosti, ale časom sa to poddalo a podarilo sa mi viac preniknúť do princípov.

Čokoľvek chceme robiť s rozpoznávaním, potrebujeme vstupné dáta – rečové vzorky. Môj optimizmus s nahrávaním vzoriek pomocou štandardných systémových nástrojov[2] rýchlo vypršal, keď nahrávanie svojou pracnosťou a nekvalitou bolo už v počiatočných fázach neúnosné.

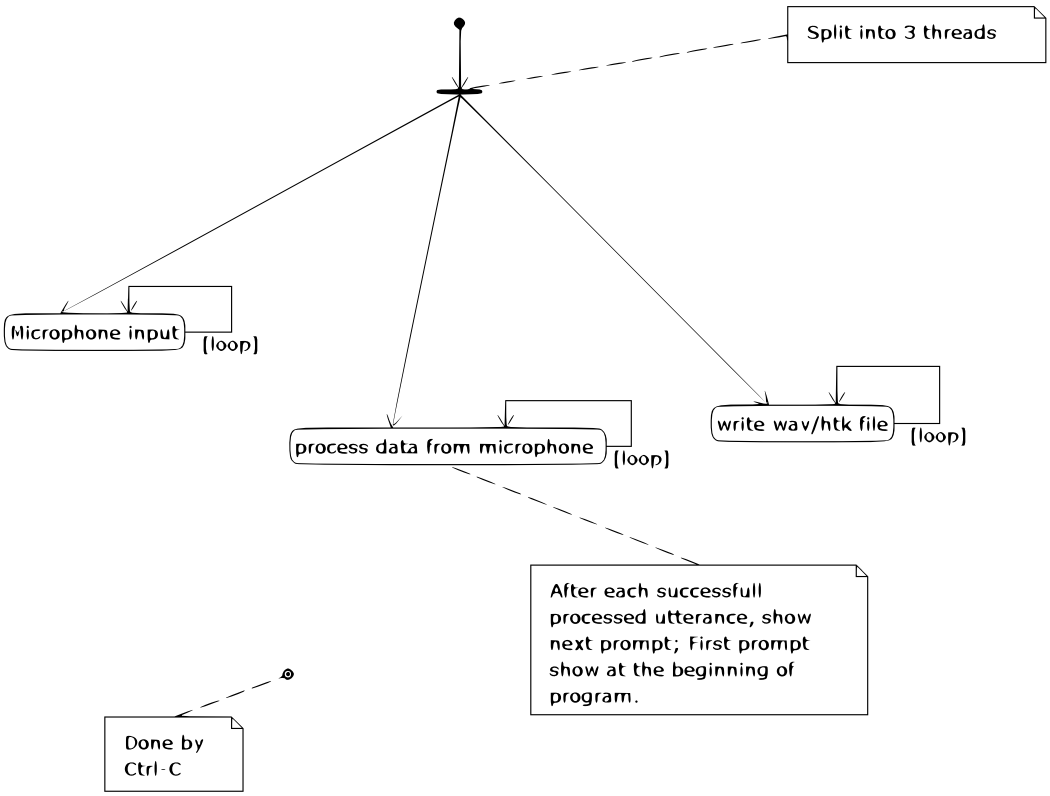
Snahou v úvodných momentoch bol pokus o rozpoznanie dvoch slov, `áno` a `nie`. Po krátkej dobe sa mi to aj podarilo. Hneď som chcel roz-

[1] The Hidden Markov Model Toolkit
[2] Napríklad linuxový KRecord

poznávač rozšírit' na štyri slová, ale výsledky neboli podľa očakávania, čo spôsobovala nekvalita trénovacích vzoriek.

Pristúpil som teda k programovaniu vlastného nahrávača slov, ktorý by mal na vstupe zoznam slov a v spolupráci s užívateľom by mal vyprodukovat' rečové vzorky.

3.4 Nahrávanie

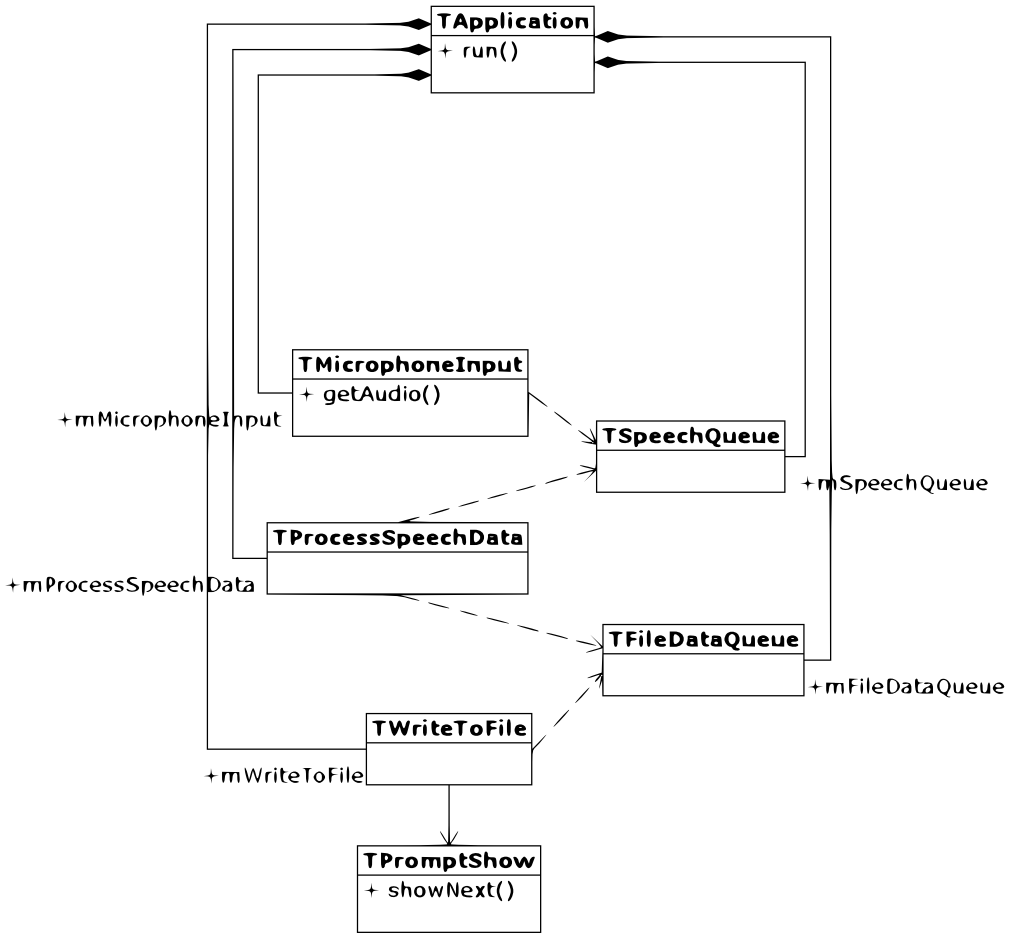


Obr. 3.1: UML activity diagram

Pri návrhu programu na nahrávanie reči som sa snažil o to, aby čítanie dát z mikrofónu nebolo brzdené ďalším spracovaním. Preto sa program na začiatku rozdelí na tri vlákna, pričom prvé neustále číta dáta z mikrofónu, druhé tieto dáta spracúva a tretie sa stará o zápis na disk vo formáte wav.

Celú myšlienku som vymodeloval pomocou UML diagramov, pre lepší náhľad. Na obrázku 3.1 je diagram aktivity.

Detailnejší pohľad na vnútro programu poskytuje ďalší z UML diagramov – diagram tried (obr. 3.2).



Obr. 3.2: UML class diagram

Snaha o jednoduché rozhranie ma priviedla na myšlienku vytvoriť len jeden objekt triedy `TApplication` a všetkú logiku implementovat doň. Telo programu je potom veľmi strohé:

```
int main(int argc, char* argv[])
{
   TApplication app(argc, argv);
   return app.run();
}
```

Následne sa vytvoria inštancie zvyšných tried (`TMicrophoneInput` až `TWriteToFile`) a pri konštrukcii objektu triedy `TWriteToFile` napokon aj inštancia triedy `TPromptShow`. V troch z nich sa vyvolajú metódy na vytvorenie vlákna, ako ukazuje fragment kódu:

```
mpMicrophoneInput->run();
```

```
mpProcessSpeechData->run();
mpWriteToFile->run();
```

Ako bolo naznačené vyššie, už to potom beží „samo“ a to nasledovne. Inštancia triedy `TMicrophoneInput` neustále číta dáta z mikrofónu a plní frontu typu FIFO[3], čo je inštancia triedy `TSpeechQueue`. Inštancia triedy `TProcessSpeechData` je semafórom zamknutá pokiaľ sa, z jej pohľadu vo vstupnej, fronte neobjavia dáta. V tom okamihu začne spracovávať dáta a čistiť vstupnú frontu (inštancia `TSpeechQueue`) a výsledky svojej práce ukladať do výstupnej fronty `TFileDataQueue`.

Teraz prichádza na rad vlákno č.3 – inštancia triedy `TWriteToFile`, ktoré na základe momentálne zobrazenej výzvy (poskytovanej triedou `TPromptShow`) zapíše dáta z fronty `TFileDataQueue` do wav súboru. Kvôli synchronizácii s výzvami sa až teraz zobrazí ďalšia výzva, všetky dáta nahrané doteraz boli považované za patriace výzve predchádzajúcej.

Ako sa ukázalo neskôr, tento mnou navrhnutý model nebol až taký scestný, pretože som našiel ďalšie opisy, ako podobnú aplikáciu pracujúcu s mikrofónom poskladať a bolo to veľmi podobné.

Ukázalo sa tiež, že týmto nahrávaním som si mohol regulovať parametre ako som potreboval, napríklad orezávanie nepotrebného ticha na začiatku a konci vzorky, čo prispelo k zautomatizovaniu a skvalitneniu získavania vstupných vzoriek.

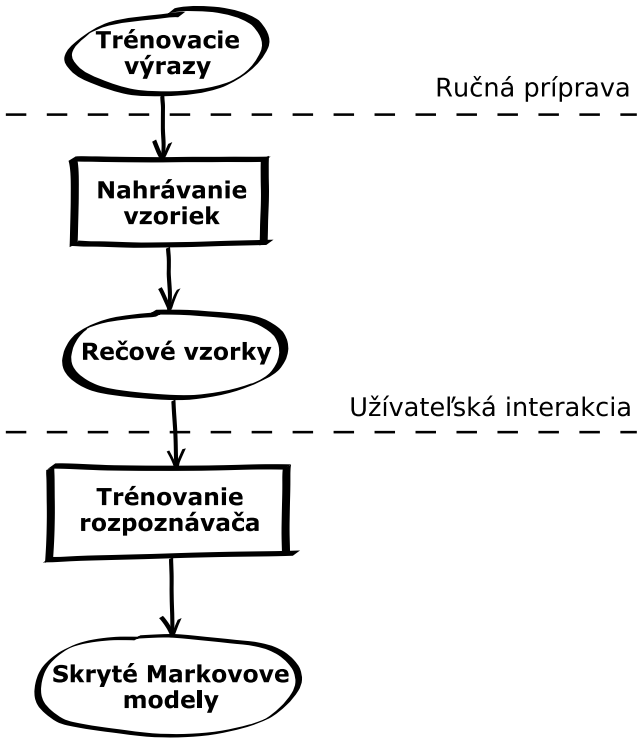
3.5 Príprava trénovacích dát

Desiatky experimentov a hodiny písania skriptov prispeli k pochopeniu princípov rozpoznávania reči na báze skrytých Markových modelov. Pustil som sa teda do konštrukcie rozpoznávača s cieľom minimálnych vstupov a čo najväčšej automatizácie.

Hrubý pohľad na štruktúru trénovania rozpoznávača je na obrázku 3.3.

V prvej fáze si pripravíme dáta na tréning modelov. Budeme potrebovať výrazy v textovej a v zodpovedajúcej rečovej podobe, ďalej definujeme výslovnosť a spočítame koeficienty pre lepšiu reprezentáciu rečových vzoriek.

[3] First In First Out



Obr. 3.3: Tréning rozpoznávača

3.5.1 Vstupné dáta

Na začiatku potrebujeme definovať textový súbor obsahujúci na každom riadku jeden trénovací výraz (slovo, skupinu slov, vetu), pomenujeme ho `subor_s_vyrazmi`. Jednoduchým skriptom očíslujeme každý z riadkov a výstup zapíšeme do súboru `trainprompts`.

```
./number_prompts.pl subor_s_vyrazmi > trainprompts
```

Tieto potom nahráme pomocou programu z predošlej sekcie – nazval som ho `record_prompts`.

```
./record_prompts -d data_train -p trainprompts
```

Výstupom nahrávača budú wav súbory v adresári `data_train`, pre každý výraz jeden, pomenované číselne skriptom `number_prompts.pl`.

3.5.2 Slovník

Na základe súboru so zoznamom výrazov (`trainprompts`) vytvoríme zoznam slov obsiahnutých v trénovacej množine. Tento zoznam bude pôsobiť

ako pomocná štruktúra pre ďalší krok.

```
./prompts2wlist trainprompts wlist
```

Teraz vytvoríme dôležitý prvok – výslovnostný slovník. Ten nám určí, z akých foném sa to ktoré slovo skladá. Pre jednoduchosť som v predošlom predpokladal, že existuje jeden súbor s trénovacími výrazmi, no skripty si poradia aj s viacerými, pomenovanými podľa nasledovnej konvencie: (train|test)_popis.cz, napríklad: train_set1.cz, test_sada2.sk.

Prefixy train respektíve test určujú, či je sada slov určená k trénovaniu alebo testovaniu rozpoznávača. Postfixy `cz` a `sk` sa používajú k rozhodnutiu, výslovnostné pravidlá ktorého jazyka sa majú použiť.

```
cat wlist | ./wlist2dict_piped.pl > dict.generic
```

Výsledný slovník je uložený v súbore `dict.generic`. Ten obsahuje výslovnosť všetkých slov trénovacej sady. Pri zápise som sa inšpiroval najmä fonetickou abecedou SAMPA[4] [smpCz], nejaké pravidlá som si pridal, pretože sa mi zdala neúplná. České fonetické pravidlá som získal na [phoCz].

Pre ilustráciu, ako preklad na fonémy vyzerá:

```
běhat [b j e h a t]
vzpomínka [w s p o m i: N k a]
čtyřicet [tS t i P/ i ts e t]
```

V slove běhat sa nahradí písmeno `ě` na fonémy `j` a `e`. Fonéma `w` v slove vzpomínka značí výslovnosť písmena `v` na začiatku slova, fonéma `P/` je výslovnosť českého `ř`, ak pred ním nie je jedno z písmen `t,v,p,b,d,k` – vtedy sa výslovnosť mení a je nutné uvažovať inú fonému.

Za zmienku možno stojí ešte fonéma `N`. Svojou výslovnosťou sa líši od klasického `n`. Preto, aby sme zamedzili tréningu jedného modelu fonémy vzorkami s rôznymi výslovnosťami, vytvoríme pre písmeno `n` pred písmenom `k` nový model.

Kompletné pravidlá prevodu, ktoré som použil, sú v dodatku A, implementuje ich skript `sampa_translate_cz.pl`.

Výnimky. Existujú vo výslovnostných pravidlách, napríklad v menách alebo slovách prevzatých z cudzích jazykov. Preto ich definujeme oddelene v slovniku `dict.exceptions` a následne spojíme oba slovníky (`dict.generic` a `dict.exceptions`) do slovníka `dict`, kde prioritne použijeme výslovnosť z `dict.exceptions`.

[4]Speech Assessement Methods Phonetic Alphabet

```
HDMan -m -w wlist -n monophones1 dict
      dict.exceptions dict.generic
```

Vedľajším a zároveň potrebným produktom, je zoznam všetkých foném (`monophones1`) vyskytujúcich sa v trénovacej sade.

3.5.3 Popisný súbor

Opäť nám ako vstup bude slúžiť zoznam výrazov (`trainprompts`) a názov adresára, kde sa nachádzajú rečové vzorky. Vytvoríme z tohto popisný súbor `train.mlf`, ktorá vzorka čo znamená.

```
./prompts2mlf train.mlf trainprompts data_train
```

Ilustračne, popisný súbor (`train.mlf`) vyzerá nasledovne

```
#!MLF!#
"data_train/set1.cz/0001.lab"
šedý modrá
.
"data_train/set1.cz/0002.lab"
jedna
.
```

Prvý riadok (`#!MLF!#`[5]) len určuje, že súbor je hlavným popisným súborom a obsahuje všetky prepisy trénovacích vzoriek. Na ďalších riadkoch sú postupne meno popisku, ktoré je odvodené od mena súboru zámenou prípony z `wav` na `lab`, popisok samotný a napokon oddeľovač vzoriek, ktorým je bodka na prázdnom riadku.

Predošlým sme získali popisný súbor na úrovni slov. My ale chcem trénovať rozpoznávač na úrovni foném, preto potrebujeme vytvoriť ďalší popisný súbor – `phones0.mlf`. Cennú službu nám poskytne `HLEd`.

```
HLEd -l '*' -d dict -i phones0.mlf
     phones0.led train.mlf
```

Utilita `HLEd` je editor popisných súborov[6] obsiahnutý v HTK, ktorý koná podľa inštrukcií v konfiguračnom súbore `phones.led`. Ten vyzerá nasledovne:

[5] Master Label File
[6] label editor

```
EX
IS sil sil
DE sp
```

Príkaz `EX` expanduje všetky slová zo vstupného popisného súboru `train.mlf`, príkaz `IS` vloží modely ticha (`sil`) na začiatok a koniec každej vzorky. Napokon príkaz `DE` vymaže model krátkej pauzy medzi slovami (`sp`[7]), ktorá je pre túto chvíľu nežiadúca.

Ďalej sa používa výslovnosť zo slovníka `dict` vytvorenú v predchádzajúcich krokoch. Parameter `-l '*'` značí implicitnú cestu k súborom, pretože nepotrebujeme popisky ukladať na iné miesto.

Fonémický popisný súbor potom vyzerá nasledovne:

```
"*/0001.lab"
sil
S
e
d
i:
sil
.
"*/0002.lab"
...
```

3.5.4 Vektorizácia vzoriek reči

Posledným prípravným krokom pred použitím skrytých Markovových modelov je výpočet vhodných koeficientov z rečových vzoriek. Používanie samotných rečových vzoriek nie je dobrou voľbou, pretože sú pod veľkým vplyvom rôznych deformácií.

Vektorizáciu signálu budeme rozumieť transformáciu zvukového signálu na postupnosť vektorov koeficientov.

Existujú viaceré metodiky, ako potlačiť v signále nepotrebnú informáciu a vyzdvihnúť takú, ktorá rozhoduje o význame. Obvyklou praktikou je prechod do spektra a napočítanie koeficientov tam. Uvediem príklad výpočtu jedného, pravdepodobne najpoužívanejšieho, typu koeficientov – MFCC[8].

Podľa pozorovaní, ľudské ucho nevníma zvukové frekvencie lineárne,

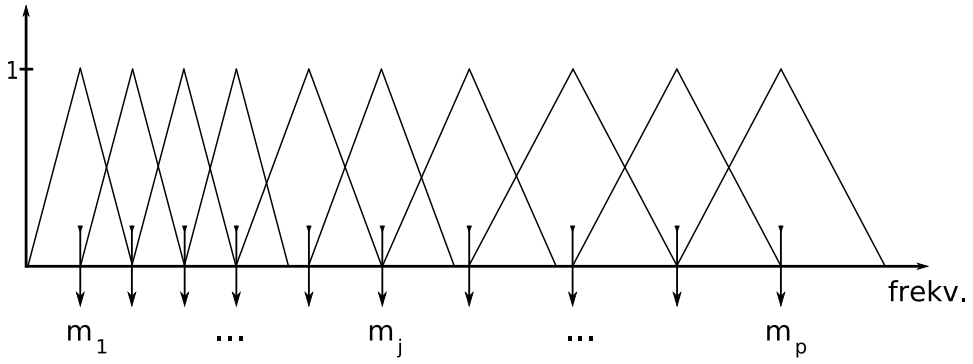
[7] sp=short pause
[8] Mel-Frequency Cepstral Coefficients

ale v pásme vyšších frekvencií je menej citlivé. Ukazuje sa, že zohľadnením tohto faktu sa dosahuje lepšia presnosť rozpoznania.

Implementačne sa to rieši zavedením stupnice Mel, ktorá odlinarizuje pôvodné frekvencie.

$$\text{Mel}(f) = 2595 \log_{10}(1 + \frac{f}{700})$$

Proces aplikácie banky filtrov začína naokienkovaním signálu a prevedením Fourierovej transformácie, čím získame amplitúdové spektrum. Následne po spektre rozložíme trojuholníkové filtre, ako je naznačené na obrázku 3.4.



Obr. 3.4: Mel-škála

Každý z koeficientov m_j je energia v danom pásme, ktorá vznikne násobením zložiek spektra so zložkami filtra a následnou sumáciou. Tiež je možné v HTK nastaviť parameter `USEPOWER`, ktorý vypočítanú energiu vydelí dĺžkou okienka a určí tak výkon signálu (energia za čas).

Samotné Mel-kepstrálne koeficienty sa potom v HTK počítajú podľa vzťahu

$$c_i = \sqrt{\frac{2}{p}} \sum_{j=1}^{p} m_j \cos(\frac{\pi i}{p}(j - 0.5))$$

kde p je počet kanálov v banke filtrov, ktorý je možné nastaviť premennou `NUMCHANS`.

Zdá sa, že voľba koeficientov MFCC je opodstatnená, ako ukazuje článok [Com04]. Pri rozpoznávaní závislom na rečníkovi sa s ňou dosahujú najlepšie výsledky a v porovnaní s koeficientami LPC[9] výrazne víťazí. Na druhej strane, LPC je výpočtovo nenáročné a je obsiahnuté napríklad aj v norme GSM[10], kde má za cieľ redukovať dátový tok.

[9] Linear Predictive Coding
[10] Global System for Mobile Communications

HTK dokáže pracovať ako s koeficientami MFC tak aj s LPC. Osobne si však myslím, že tieto sú už vo väčšine úloh prekonané, ich hlavnou devízou je výpočtová nenáročnosť.

V HTK je na počítanie koeficientov pripravený nástroj `HCopy`, ktorý v závislosti na konfiguračnom súbore kopíruje vstupné vzorky na výstupné, pričom prevádza ich vektorizáciu podľa parametrov zadaných v konfiguračnom súbore `hcopy.cfg`.

```
HCopy -C hcopy.cfg vzorka.wav vzorka.mfc
```

Konfiguračný súbor má nasledovnú podobu:

```
SOURCEKIND     = WAVEFORM
SOURCEFORMAT   = WAV

TARGETKIND     = MFCC_E
TARGETFORMAT   = HTK
TARGETRATE     = 100000
WINDOWSIZE     = 250000
```

Hovorí toľko, že vstupom sú vzorky vo formáte `wav` a chceme z nich napočítať koeficienty MFCC ku ktorým pridávame jeden koeficient značiaci energiu v danom okienku. HTK používa pre meranie časových úsekov 100ns jednotky, takže okienka dĺžky 25msec posúvame po signále s krokom 10msec. Dochádza teda k značnému prekryvu.

Existuje aj pohodlnejší spôsob ako konvertovať veľké množstvá vzoriek – použitím skriptu so zoznamom dvojíc súborov. Dvojice súborov `wav,mfc` vytvoríme štandardným unixovým nástrojom `find` a výstup zapíšeme do súboru `train.scp`.

```
find data_train -iname '*.wav' |
  sed 's/\(.*\)\.wav/\1.wav \1.mfc/' > train.scp
```

Pohľad do súboru `train.scp` ukazuje

```
data_train/0001.wav data_train/0001.mfc
data_train/0002.wav data_train/0002.mfc
...
```

Koeficienty teraz spočítame zavolaním

```
HCopy -C hcopy.cfg -S train.scp
```

Tým sa uzatvára fáza prípravy na nastavovanie parametrov skrytých Markovových modelov. Máme pripravené rečové vzorky vo forme MFCC koeficientov (`*.mfc`), popis k týmto rečovým vzorkám (`phones0.mlf`), zoznam foném (`monophones1`) a slovník s výslovnosťou (`dict`).

3.6 Tréning modelov

Máme pripravené koeficienty rečových vzoriek, vieme, z akých foném sa skladajú, môžeme teda pristúpiť k tvorbe modelov.

3.6.1 Inicializácia modelov

Prvým krokom pri tréningu modelov je ich formálna definícia. Zatiaľ je jedno, aké hodnoty parametrov sa použijú, ide nám len o určenie topológie modelu.

```
./MakeProtoHMMSet models.pcf
```

Skript `MakeProtoHMMSet` napísaný v PERL-e neznámym autorom som rozšíril o tvorbu komplikovanejších topológií a s výhodou ho použil.

Tento prečíta definičný súbor `models.pcf` a na jeho základe zapíše topológiu modelu do súboru `proto`. Definícia modelu je vykonaná jednoduchým nastavením. Pár riadkov súboru `models.pcf`:

```
nStates: 2
parmKind: MFCC_E
vecSize: 13
```

Vybral som najdôležitejšie nastavenia modelu a to `nStates` – počet emitujúcich stavov skrytého Markovovho modelu, `parmKind` – parametre kódovania na koeficienty, `vecSize` – počet prvkov vektoru stredných hodnôt a rozptylov. Je plne určený parametrami kódovania, tu konkrétne 12 prvkov dávajú koeficienty MFCC a jeden pridáva hodnota energie.

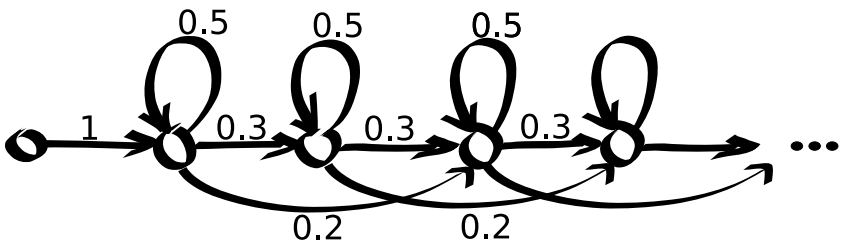
Definovanie povolených prechodov som rozšíril nasledovne:

```
<BEGINtransp>
  0.000e+0 1.000e+0
  5.000e-1 3.000e-1 2.000e-1
<ENDtransp>
```

Každý riadok odpovedá jednému stavu modelu. Prvá hodnota na riadku je pravdepodobnosť zotrvania modelu v danom stave, ďalšie hodnoty sú pravdepodobnosti prechodu do stavov nasledujúcich. Pravdepodobnosti inak nešpecifikované položíme rovné nule. Pre stavy 3 a viac platí riadok 2. Je možné nadefinovať aj viac riadkov (ich počet označme k, $k >= 2$), potom pre stav n ($n > 1$) platí riadok

$$(n \bmod (k-1)) + 1.$$

Pravdepodobnosti prechodov medzi stavmi nám v tejto fáze určujú výlučne fakt, či je prechod povolený (nenulová pravdepodobnosť), alebo nie je. Ilustruje to obrázok 3.5.



Obr. 3.5: Prechody v skrytom Markovovom modele

Zjednodušený náhľad na súbor `proto` by mohol vyzerať pre 2 emitujúce stavy nasledovne:

```
<BeginHMM>
 <NumStates> 4
 <State> 2
    <Mean> 13 0.0  0.0  0.0  ...
    <Variance> 13 1.0  1.0  1.0  ...
 <State> 3
    <Mean> 13 0.0  0.0  0.0  ...
    <Variance> 13 1.0  1.0  1.0  ...
 <TransP> 4
  0.0  1.0  0.0  0.0
  0.0  0.6  0.4  0.0
  0.0  0.0  0.6  0.4
  0.0  0.0  0.0  0.0
<EndHMM>
```

Formálnu definíciu modelu máme pripravenú, môžme ju zinicializovať. Použijeme na to utilitu `HCompV`

```
HCompV -f 0.01 -m -S train1.scp -M hmm0 proto
```

HTK skript `train1.scp` obsahuje zoznam súborov s koeficientami. Vznikne podobnou technikou, ktorú sme úspešne použili v predošlej sekcii:

```
find data_train -iname '*.mfc' > train1.scp
```

Utilita `HCompV` vzala formálnu definíciu zo súboru `proto`, načítala všetky vstupné vzorky, vypočítala globálnu strednú hodnotu a rozptyl a týmito nahradila hodnoty vo formálnom modele. Parameter `-m` inštruuje `HCompV`, aby vygeneroval aj vektor stredných hodnôt, `-f 0.01` vygeneruje súbor `vFloors` obsahujúci `0.01` násobok globálnych rozptylov, ktoré sa použijú, ak by hrozilo, že rozptyl v niektorom zo stavov klesne dôsledkom nedostatku dát pod túto prahovú hodnotu.

Napokon parametrom `-M` nastavíme výstupný adresár na `hmm0` – doň sa zapíše zinicializovaný model `proto` a vektor prahových hodnôt rozptylov rozptylov `vFloors`.

Takto máme vytvorený akýsi priemerný model fonémy naprieč všetkými trénovacími dátami. Vytvoríme jeho kópiu pre každú z foném a všetky kópie zapíšeme pre lepšiu kompaktnosť do jedného súboru. K tomu som napísal jednoduchý skript, ktorý prečíta priemerný model (`hmm0/proto`) a zoznam foném (`monophones1`) a vygeneruje v adresári `hmm0` súbor `hmmdefs`.

```
./copyProtoHmm.pl
```

Zjednodušene, súbor `hmm0/hmmdefs` vyzerá pre fonémy `d,e` nasledovne:

```
~h "d"
<BeginHMM>
...
<EndHMM>
~h "e"
<BeginHMM>
...
<EndHMM>
```

Hodnoty parametrov jednotlivých skrytých Markovových modelov (vektor stredných hodnôt, rozptylov a matica pravdepodobností prechodu) sa následne prepočítajú pomocou Baum-Welchovho iteračného algoritmu. Utilita `HERest` implementuje jeden iteračný krok tohto algoritmu.

```
HERest -t 250 150 1000 -I phones0.mlf -S train1.scp
 -M hmm1 -H hmm0/hmmdefs -H hmm0/vFloors monophones0
```

Týmto príkazom načítame modely (`hmmdefs`) a minimá rozptylov z adresára `hmm0` a pre každú vzorku z `train1.scp` sa podľa popisného súboru `phones0.mlf` určí, ktoré modely foném sa budú odhadovať danou vzorkou.

Pripomeňme, že z popisného súboru `phones0.mlf` sme vyradili modely krátkych páuz (pauzy medzi slovami) a taktiež tento model vyradíme zo zoznamu foném:

```
sed '/sp/d' monophones1 > monophones0
```

Účelom je, aby tréning tohto modelu príliš nerozhodil hranice medzi fonémami. Baum-Welchov algoritmus je iteratívny, pre lepšie odhady je vhodné použiť ho ešte dva krát, výstup postupne zapisujeme do adresárov `hmm2` a `hmm3`.

```
HERest -t 250 150 1000 -I phones0.mlf -S train1.scp
 -M hmm2 -H hmm1/hmmdefs -H hmm1/vFloors monophones0

HERest -t 250 150 1000 -I phones0.mlf -S train1.scp
 -M hmm3 -H hmm2/hmmdefs -H hmm2/vFloors monophones0
```

Posledným nevysvetleným parametrom zostáva `-t`. Ten určuje zamietacie prahy. Pri odhadoch parametrov modelu sa počíta, s akou pravdepodobnosťou by daný model generoval pozorovanie, z ktorého sa parametre odhadujú. Kvôli urýchleniu výpočtu, nastavenie nižšej hranice spôsobí zúženie manévrovacieho priestoru pre cestu modelom.

Model prechádza zo stavu do stavu a generuje postupne pozorovanie. Rozvíjame všetky možnosti, ako sa môžu meniť stavy v rámci modelu (povolené prechody definované maticou pravdepodobností prechodov). Priebežne počítame pravdepodobnosť, s akou trénovaný model generuje dané pozorovanie. Ak táto klesne pod nastavenú hodnotu, ďalej túto cestu nerozvíjame. To je v poriadku, pretože tým sa nepripravujeme o najlepšiu možnosť, len odstraňujeme neperspektívne cesty – čím urýchľujeme výpočet.

Stáva sa, že zamietneme všetky možné cesty s danou úrovňou prahu. Vtedy je dobré prah trošku zdvihnúť a zopakovať výpočet. No po istej hranici je už veľmi pravdepodobné, že vzorka je poškodená.

Parametrom `-t` teda postupne nastavujeme prvý prah, krok, ktorým sa prah zvyšuje a prah, ktorý už nechceme prekročiť a radšej vzorku vyradíme z trénovacej sady.

3.6.2 Pauza medzi slovami

V predošlej sekcii sme odstránili pauzu medzi slovami (sp), modely sme nechali odhadnúť Baum-Welchovym algoritmom, čím získali určitú robustnosť a už nebudú také citlivé na výskyt tejto pauzy.

Skriptom makesp načítame skryté Markovove modely, aby sme zistili ich štruktúru, vytvoríme nový model pre pauzu medzi slovami (sp) a tento pridáme k existujúcej sade modelov. Výsledné modely (hmmdefs) budú sídliť v adresári hmm4.

```
cp hmm3/* hmm4/
./makesp hmm4/hmmdefs >> hmm4/hmmdefs
```

Pôvodne sme model pauzy (sp) odstánili z fonémického popisného súboru, teraz tento súbor vytvoríme znovu, aj s pauzami:

```
HLEd -l '*' -d dict -i phones1.mlf
     phones1.led train.mlf
```

Konfiguračný súbor phones1.led vyzerá veľmi podobne ako predošlý (phones0.led) – chýba mu však príkaz na odstránenie modelu sp:

```
EX
IS sil sil
```

Po pridaní modelu pauzy vykonáme ďalšie dve reestimácie[11] modelov.

```
HERest -t 250 150 1000 -I phones1.mlf -S train1.scp
 -M hmm5 -H hmm4/hmmdefs -H hmm4/vFloors monophones1

HERest -t 250 150 1000 -I phones1.mlf -S train1.scp
 -M hmm6 -H hmm5/hmmdefs -H hmm5/vFloors monophones1
```

3.6.3 Prerovnanie vzoriek

Pod týmto tajomným nadpisom sa skrýva pokus o rozpoznanie trénovacích vzoriek. Ak máme v slovníku (dict) viac výslovností k jednému slovu, napríklad [nazhledano_u] verzus [nasxledano_u] (znak x je výslovnostná fonéma náležiaca ch), použije sa tá, ktorá lepšie vyhovuje doteraz natrénovaným fonémam.

[11] iteratívne znovu-odhady

Tento krok tiež odhalí rečové vzorky, ktoré sa podľa doteraz natrénovaných foném veľmi odkláňajú a napovedajú, že je niečo v neporiadku so vstupnými dátami. Napríklad zle vyslovené slovo či nadmerný šum na pozadí.

```
HVite -l '*' -o SWT -b SENT-START -b SENT-END -t 250
-H hmm6/vFloors -H hmm6/hmmdefs -i aligned.mlf -m -a
-y lab -I train.mlf -S train1.scp dict monophones1
```

Parametrov je pomerne veľa, vysvetlime ich. `-l` má podobnú funkciu ako pri utilite `HLEd`, to znamená, že určuje výstupný adresár pre popisky. Keďže ho nechceme meniť, použijeme `*`. Výstupné formátovanie je určené parametrom `-o`, popisy nechceme rozpoznaním meniť, preto potlačíme niektoré vlastnosti výstupu, a to `S` – odstráni výsledné skóre rozpoznania, `W` – odstráni rozpoznané slová (nepotrebujeme ich tam, lebo máme popis vo fonémickom tvare), `T` – nevypisuje časové známky pri rozpoznávaní.

`SENT-START` a `SENT-END` sú definície symbolov, ohraničujúcich vety[12]. Parametre `-a, -m, -i, -y` po rade určujú, že sa vzorky majú prerovnávať, že sa má generovať výstupný prerovnaný popisný súbor, `-i` definuje meno tochto súboru a napokon `-y` príponu pre popisky. Tá nehrá veľkú rolu, pretože všetky popisky koncentrujeme priamo v jednom súbore. Prah pre zamietnutie vzorky ako nekvalitnej na trénovanie nastavíme pomocou `-t`.

Vo výsledku budeme mať nový súbor s popisom vzoriek, kde sa prepíšu vzorky podľa viacerých možných výslovností tak, aby to lepšie zodpovedalo rečovým vzorkám. Nekvalitné vzorky tam nebudú vôbec, preto ich vyhodíme z trénovacej sady.

Napísal som skript, ktorý porovná prerovnané vzorky v `aligned.mlf` a pôvodné v `train1.scp`. Tie, ktoré sa prerovnaním odstránili sa odstránia aj z trénovacej sady a vytvorí sa nová sada `train2.scp`.

```
./fix-missing_labels.pl aligned.mlf
  train1.scp train2.scp
```

S touto lepšou sadou opäť preladíme modely Baum-Welchovym algoritmom a výstup očakávame v adresári `hmm8`:

```
HERest -t 250 150 1000 -I aligned.mlf -S train2.scp
 -M hmm7 -H hmm6/hmmdefs -H hmm6/vFloors monophones1

HERest -t 250 150 1000 -I aligned.mlf -S train2.scp
 -M hmm8 -H hmm7/hmmdefs -H hmm7/vFloors monophones1
```

[12] sentence boundary

3.6.4 Z foném trifóny

Povedzme najprv, čo je trifón. Trifón je obyčajná fonéma v kontexte. Napríklad fonéma `o` je v kontexte `f-o-n` trifónom. V inom kontexte vytvorí iný trifón.

Zoznam všetkých trifónov (`triphones`) vytvoríme editorom popisných vzoriek `HLEd` pomocou parametra `-n`.

```
HLEd -n triphones -l '*' -i wintri.mlf
     mktri.led aligned.mlf
```

Ďalší popisný súbor – `wintri.mlf`, vytvoríme pomocou parametra `-i`, tentokrát bude mať stavebnú jednotkou trifón. Činnosť utility `HLEd` je riadená konfiguračným skriptom `mktri.led`, ktorý vyzerá nasledovne:

```
WB sp
WB sil
TC
```

Prvé dva riadky hovoria o znakoch pre hranice slov[13] a posledný inštruuje `HLEd` k vytvoreniu trifónov z každého fonému.

Máme vytvorený zoznam trifónov. Lenže doteraz trénované modely boli len fonémy. Preto každý z fonémov rozkopírujeme potrebný počet krát (podľa toho, koľko trifonických kontextov je v trénovacej sade) a každú kópiu prehlásime za model príslušného trifónu.

```
HHEd -H hmm8/vFloors -H hmm8/hmmdefs
     -M hmm9 mktri.hed monophones1
```

Každá fonéma z množiny `monophones1` sa naklonuje. Aby sme zaistili podobnosť modelov všetkých trifónov s rovnakou strednou fonémou, budú tieto modely zdieľať spoločnú maticu prechodov medzi stavmi. Navzájom sa budú odlišovať strednými hodnotami a rozptylmi vektorov koeficientov. Toto zdieľanie definujeme konfiguračným súborom `mktri.hed`, ktorý automaticky vygenerujeme. Prvých pár riadkov tochto súboru:

```
CL triphones
TI T_p {(*-p+*,p+*,*-p).transP}
TI T_e {(*-e+*,e+*,*-e).transP}
...
```

[13] Word Boundary

Príkaz `CL` definuje zoznam trifónov, na ktoré sa majú príslušné fonémy naklonovať. Príkazy `TI` vytvoria makro pre množinu, ktorá je definovaná v `{...}`. Konkrétnu tu je definované, že prechodové matice trifónov so stredným fonémom `p` budú zdieľané. Zohľadníme aj okrajové javy, keď jeden z kontextov chýba. Rozsiahly konfiguračný súbor `mktri.hed` vytvoríme pre celú množinu fonémov pomocou skriptu:

```
./maketrihed monophones1 triphones > mktri.hed
```

Ako už viackrát uvedené, po naklonovaní fonémov na trifóny opäť reestimujeme modely dvomi prechodmi Baum-Welchovho algoritmu:

```
HERest -t 250 150 1000 -I wintri.mlf -S train2.scp
 -M hmm10 -H hmm9/hmmdefs -H hmm9/vFloors triphones

HERest -t 250 150 1000 -I wintri.mlf -S train2.scp
 -M hmm11 -H hmm10/hmmdefs -H hmm10/vFloors triphones
```

Oproti predošlým reestimáciám používame nový súbor s popismy vzoriek (`wintri.mlf`), ktorý je zložený z trifonických jednotiek. Naviac zoznam trénovaných modelov sa zmenil z fonémov (`monophones1`) na trifóny (`triphones`).

3.6.5 Spojenie podobných trifónov

Dôvod na spájanie niektorých trifónov je ten, že každá fonéma sa nám naklonuje v najhoršom prípade na $(pocet\_fonem)^2$ trifónov. To môže byť aj vyše 2000 trifónov ku každej fonéme. Tu by sme aj s veľkou trénovacou sadou trpeli nedostatkom výskytov niektorých trifónov. Preto sa snažíme nájsť trifonické reprezentácie, ktoré sú si v nejakom zmysle blízke a združiť ich do skupín.

Delenie do skupín vykonáme formou otázok. Položíme binárnu otázku na trénovaciu množinu, ktorá nám ju rozdelí na vzorky vyhovujúce a tie druhé. Ďalšou otázkou pokračujeme v delení oboch skupín, až do momentu, keď by po rozdelení mala zostať príliš malá skupina.

Po niekoľkých krokoch nám vznikne N skupín. Trifóny v každej zo skupín budeme trénovať vzorkami všetkých ostatných. Prakticky, stotožníme príslušné stavy trifónov v rovnakej skupine.

Konštrukcia otázok a prahov, podľa ktorých sa má rozdeľovať, vyžaduje pár zamyslení a experimentov. Akú formu majú otázky, dá náhľad do súboru `tree_questions`:

```
QS "R_Nasal"     {*+m,*+n,*+J,*+g}
QS "R_Fricative" {*+f,*+v,*+s,*+S,*+z,*+Z,*+j,*+x}
...
```

`QS` je riadiaci príkaz oznamujúci, že sa jedná o otázku, ďalej je táto pomenovaná a v {...} je zoznam trifónov, ktoré vyhovejú otázke.

Potom už zavolaním drobného skriptu, ktorý som pre zmenu napísal v bashi[14] docielime vygenerovanie konfiguračného súboru pre editor modelov `HHEd`.

```
./make_tree_hed.sh > tree.hed
```

V konfiguračnom súbore `tree.hed` sa okrem definície otázok nachádzajú generické príkazy na vytvorenie počiatočných skupín. Napríklad príkaz pre zoskupenie stavov č.4 trifónov so strednou fonémou „a“ vyzerá nasledovne:

```
TB 450.0 "ST_a_4_"{("*-a+*","a+*","*-a").state[4]}
```

Na každú takto definovanú počiatočnú skupinu sa potom postupne aplikujú všetky rozhodovacie otázky a zisťuje sa, ktorá otázka najviac zvýši pravdepodobnostné ohodnotenie rozdelených skupín. Táto je následne vybratá ako deliaca v danom kroku.

Podľa inštrukcií v konfiguračnom súbore `tree.hed` modely upravíme.

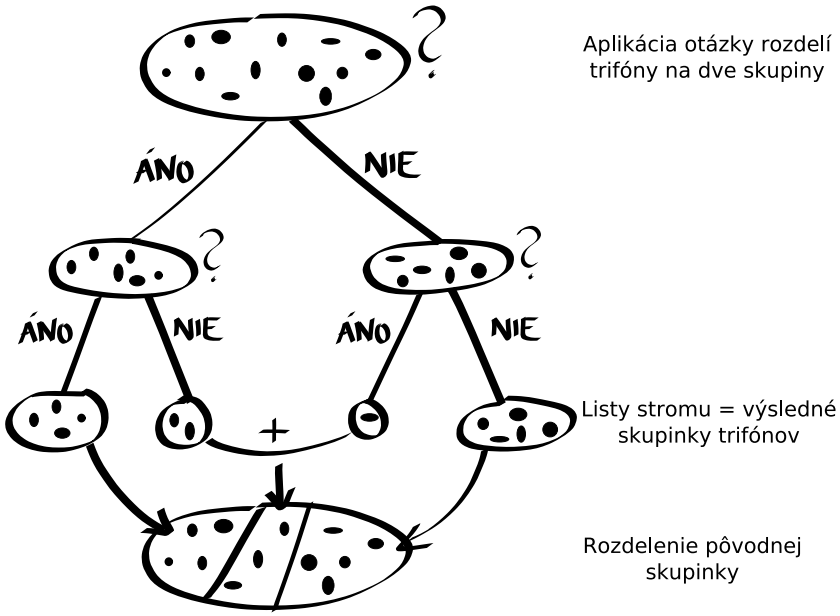
```
HHEd -H hmm11/vFloors -H hmm11/hmmdefs -M hmm12
     tree.hed triphones > log
```

Po vykonaní je užitočné nahliadnuť do protokolu `log` a zistiť, ako prebehelo viazanie trifónov. Obvyklý výstup vyzerá nasledovne:

```
TB 450.00 ST_a_4_ {}
 Tree based clustering
 Start   a[4] : 12  have LogL=-67.227 occ=1034.5
 Via     a[4] : 4  gives LogL=-63.596 occ=1034.5
 End     a[4] : 3  gives LogL=-63.858 occ=1034.5
 TB: Stats 12->3 [25.0%]{419->168 [40.1%] total}
...
```

Z užívateľského hľadiska nás zaujíma najmä informácia, že sme mali 12 trifónov, po rozdelení pomocou otázok z hraničnou hodnotou 450 nám

[14] bash – GNU Bourne-Again SHell [bash]



Obr. 3.6: Strom viazania trifónov

vznikli 4 skupiny, a dve z nich (s rôznymi najbližšími rodičmi) sa ešte oplatilo spojiť. Takže vo výsledku máme z 12 trifónov 3 skupiny. Čísla 419->168 informujú o celkovom znížení počtu stavov, je to len akumulácia hodnôt 12->3 cez všetky doposiaľ spracované stavy a fonémy.

Pre viac informácií sa oplatí zapnúť podrobnejší výstup (`-T 20400`). Tým docielime vypisovanie informácií o ohodnotení jednotlivých otázok, ktoré otázky boli vybraté na delenie množín a budeme vidieť, do akých množín sa trifóny rozdelili. Výstup by mohol vyzerať takto:

```
Start   a[4] : 12  have occ=1034.5
     Q   R_NonBoundary Imp =  1447.62 (691.2,343.3)
     Q          R_Stop Imp =  1521.17 (843.2,191.4)
     Q         L_Vowel Imp =     0.00 (1034.5,0.0)
               ...
  BestQ               R_Stop
...
Nodes for ST_a_4_
  0[Y]  == v-a+d v-a+t p-a+d p-a+t
  1[Y]  == k-a t-a v-a+n v-a+ts z-a+v
  2[N]  == l-a n-a v-a
```

Číslo 1034.5 ohodnocuje, ako tých 12 trifónov modeluje danú množinu – t.j. ako blízko celkovému rozptylu a strednej hodnote je každý z trifónov.

Pri rozdelení na dve množiny bude súčet týchto ohodnotení aspoň tak dobrý ako pre jednu množinu. Stred bude bližšie danej skupinke, tiež rozptyl nebude taký veľký. Otázka, ktorá docieli najvyšší nárast tohto ohodnotenia, bude vybratá ako deliaca.

Každým potencionálnym rozdelením dosiahneme zvýšenie ohodnotenia, to nám označuje pri prvej otázke číslo 1447.62. Ďalej v zátvorke je dvojica – príspevky členov jednotlivých množín v pôvodnej nerozdelenej skupinke.

V druhej časti výpisu je zoznam trifónov rozdelných otázkami na tri množiny.

Skúmanie výpisu procesu viazania trifónov má za účel ladenie nastavení, napríklad voľbu otázok a voľbu prahov. V predošlej ukážke bol príkazom `TB` nastavený prah na 450, delenie teda robíme až do momentu, keď nárast ohodnotenia (`Imp`) klesne pod zadaný prah.

V konfiguračnom súbore `tree.hed` sa nachádza aj príkaz `CO`, ktorý nastaví výstupný súbor pre skupiny trifónov. Tento budeme v ďalšom používať namiesto súboru so zoznamom trifónov (`triphones`).

```
CO "tiedlist"
```

Posledné dve reestimácie a sme pripravení rozpoznať vzorky z trénovacej sady.

```
HERest -t 250 150 1000 -I wintri.mlf -S train2.scp
 -M hmm13 -H hmm12/hmmdefs -H hmm12/vFloors tiedlist

HERest -t 250 150 1000 -I wintri.mlf -S train2.scp
 -M hmm14 -H hmm13/hmmdefs -H hmm13/vFloors tiedlist
```

3.7 Rozpoznávanie

Pri vývoji rozpoznávača bolo užitočné urobiť aj dávkové rozpoznávanie s vopred nahranými vzorkami. Samozrejme, efektnosť sa ukáže až pri priamom rozpoznaní z mikrofónu.

3.7.1 Dávkové rozpoznávanie

Nebudem už príliš zabiehať do detailov, pretože úvodné časti sú takmer totožné s tréningom rozpoznávača. Najprv vytvoríme súbor s testovacími

slovami a slovnými spojeniami, ten skriptom `number_prompts.pl` očíslujeme, aby si s tým program na nahrávanie poradil. Vypichnem len fakt, že pôvodne sme tento výstupný súbor nazývali `trainprompts`, takže analogicky ho teraz nazveme `testprompts`.

```
./number_prompts.pl subor_s_vyrazmi > testprompts
```

Takto očíslované vzorky si vezme za vstup program `record_prompts` a postupne bude vyzývať užívateľa k čítaniu, výsledok bude ukladať do adresára `data_test`.

```
./record_prompts -d data_test -p testprompts
```

Vytvoríme popisný súbor s testovacími vzorkami (`test.mlf`):

```
./prompts2mlf test.mlf testprompts data_test
```

Pomocou príkazu `find` zapíšeme dvojice súborov `mfc, wav` do súboru `test.scp`

```
find data_test -iname '*.wav' |
  sed 's/\(.*\)\.wav/\1.wav \1.mfc/' > test.scp
```

Následne vektorizujeme vzorky utilitou `HCopy`.

```
HCopy -C hcopy.cfg -S test.scp
```

Teraz príde zaujímavá vec. Keďže sme sa odhodlali rozpoznávať aj nenatrénované slová, potrebujeme urobiť drobné úpravy modelov. V prvom rade vytvoríme zoznam slov z testovacích vzoriek (`testprompts`) a tieto spojíme s pôvodným zoznamom slov.

```
./prompts2wlist testprompts > wlist_test

./add2wlist.pl wlist wlist_test
```

Ešte potrebujeme poznať výslovnosť nových slov, preto ju vytvoríme genericky (`dict.test.generic`) a pridáme do pôvodného výslovnostného slovníka `dict` utilitou `HDMan`.

```
cat wlist_test | ./wlist2dict_piped.pl >
  dict.test.generic

HDMan -i -b sp -n triphones.new dict.new
  dict dict.test.generic
```

Utilitu sme naviac nechali vygenerovať zoznam nových trifónov – `triphones.new`. Následne pomocou štandarných príkazov sady trifónov spojíme, utriedime (`sort`) a odstránime duplikáty (`-u`). Výsledná sada trifónov bude v súbore fulllist.

```
cat triphones triphones.new | sort -u > fulllist
```

Pôvodne som pre lepší prehľad nezmienil príkaz utility `HHEd`, ktorým definujeme súbor, do ktorého sa má uložiť strom s otázkami a zviazanými skupinkami trifónov. Týmto príkazom je `ST "trees"`[15] a analogický príkaz využijeme teraz na načítanie tohto stromu (`LT`[16]). Ďalším nový príkazom je `AU`, ktorý definuje sadu trifónov, rozsiahlejšiu ako trénovacie dáta. Nám už známym príkazom `CO` definujeme súbor, kam sa majú uložiť zviazané skupinky trifónov. Slovne som opísal, čo vložíme do konfiguračného súboru `tree_unseen.hed`, tak doň nahliadnime:

```
LT "trees"
AU "fulllist"
CO "tiedlist.full"
```

Editorom skrytých Markovovych modelov následne upravíme modely a zapíšeme ich do adresára `hmm16`:

```
HHEd -H hmm14/vFloors -H hmm14/hmmdefs
     -M hmm15 tree_unseen.hed tiedlist
```

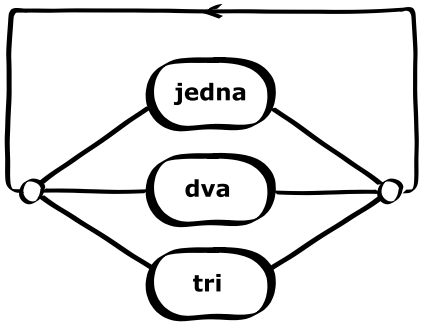
Tým sme vytvorili modely aj doteraz netrénovaných trifónov.

Vyvrcholením celého rozpoznávania je konečne aplikácia Viterbiho algoritmu na rozpoznanie neznámych vzoriek. Predtým však si ešte pripravíme vstupy. Aby sme nemuseli všetky vzorky vpísať do príkazového riadku, zapíšeme ich do súboru `test1.scp`:

```
find data_test -iname '*.mfc' > test1.scp
```

Poslednou kockou do skladačky je definícia gramatiky, podľa ktorej sa má Viterbiho algoritmus riadiť pri prechode rozpoznávacou sieťou. Gramatiku definujeme ako konečno-stavový automat, kde stavmi sú slová a prechody sú povolené náväznosti slov. V jednoduchom prípade povolíme prechody medzi každou dvojicou slov. To spravíme poľahky, ak vytvoríme nový stav (neznamenajúci slovo), naviažeme ho na každé zo slov spredu a podobný stav zozadu. Tieto dva následne spojíme.

[15] Store Trees
[16] Load Trees



Obr. 3.7: Gramatika

Súbor s intuitívnou definíciou gramatiky `gram` by mohol vyzerať nasledovne:

```
$word = jedna | dva | tri;
<$word>
```

Notácia používaná na zápis tohto súboru sa nazýva BNF[17], viac o nej na [Bnf].

Túto intuitívnu gramatiku preložíme do počítaču zrozumiteľnejšej podoby pomocou HTK utility `HParse`. Súbor `wdnet` predstavuje útočisko pre výslednú gramatiku.

```
HParse -A gram wdnet
```

Výpočtovo najnáročnejšou časťou je práve aplikácia Viterbiho algoritmu.

```
HVite -H hmm15/hmmdefs -H hmm15/vFloors -i out.mlf
  -S test1.scp -w wdnet dict.new tiedlist.full
```

Zo vstupných parametrov poznáme `hmmdefs`, `vFloors`, `dict.new`, `tiedlist.full` a teraz už aj `test1.scp` a `wdnet`. Parametrom `-i` definujeme výstupný popisný súbor s rozpoznanými vzorkami. Ten má štruktúru ako obyčajný popisný súbor, s výnimkou, že popisky sú vypočítane Viterbiho algoritmom na základe ostatných vstupov.

Teraz nás bude určite zaujímať, aké úspešné bolo rozpoznávanie. To vykonáme utilitou `HResults`, ktorá porovná pôvodný správny popisný súbor `test.mlf` so súborom vytvoreným rozpoznaním `out.mlf`.

[17] Backus Naur Form

```
HResults -t -I test.mlf tiedlist.full out.mlf
```

Typický výstup vyzerá nasledovne:

```
---------------- Overall Results -------------------
SENT: %Correct=99.20 [H=496, S=4, N=500]
WORD: %Corr=99.20,Acc=99.20 [H=496,D=0,S=4,I=0,N=500]
====================================================
```

Prvý riadok informuje o presnosti rozpoznania na úrovni slovných spojení a viet. Na druhom riadku je informácia o presnosti na úrovni jednotlivých slov. Z hranatých zátvoriek sa dá vyčítať, že správne bolo rozpoznaných 496 slov (`H`), chýb vymazaním bolo nula (`D`), chýb nahradením 4 kusy (`S`), vkladacích chýb taktiež nula (`I`), to všetko z celkového počtu 500 slov (`N`).

Hodnota `Corr` narozdiel od `Acc` informuje o presnosti bez započítania vkladacích chýb.

Tu ilustrovaný prípad obsahoval gramatiku, kde bolo povolené vždy len jedno slovo. Preto sú hodnoty na riadku `SENT` aj `WORD` rovnaké. Taktiež nevznikla žiadna vkladacia chyba (`I=0`) – v opačnom prípade by bola hodnota `Acc` nižšia.

3.7.2 Priamo z mikrofónu

Vieme rozpoznávať dávkovo, takže k priamemu rozpoznávaniu z mikrofónu je len krôčik. Pre tento účel som upravil (a premenoval na `live_input`) program na nahrávanie opísaný v sekcii 3.4, aby po nahratí neznámej vzorky z mikrofónu vykonal nejakú akciu. Touto akciou je zavolanie skriptu `recogniser_action.sh` s parametrom odkazujúcim na práve uložený wav súbor.

Skript len zavolá utility na vektorizáciu `HCopy` a `HVite` – utilitu na rozpoznávanie:

```
HCopy -T 0 -C hcopy.cfg vzorka.wav vzorka.mfc

HVite -T 1 -H hmm15/hmmdefs -H hmm15/vFloors -w wdnet
  dict tiedlist vzorka.mfc | tail -1 | ./showWord.pl
```

V prvom príkaze nutno spomenúť parameter `-T`, ktorý nastavuje mieru podrobnosti výpisu (nula znamená žiadny výpis). Naopak, v `HVite` je veľmi žiadané nastaviť `-T` na hodnotu 1, tak sa nám totiž rozpoznaný výraz vypíše (štandardne sa to nedeje).

Skúsené oko linuxového užívateľa iste neprehliadlo, že posledný riadok výstupu rozpoznania (linuxový príkaz `tail -1`) posielame na vstup skriptu `showWord.pl`. Tento má už jednoduchú, pritom populárnu úlohu, sformátuje podľa ľubovôle výstup rozpoznania, môže ho vypísať, na jeho základe spustiť program a podobne.

3.7.3 Čísla

Z prezentačných dôvodov, ako ukážku použiteľnosti som implementoval rozpoznávač jednoduchých matematických výrazov.

Trénovacia sada bola pomerne jednoduchá – zhruba 400 vyslovených čísel a matematických operácií. Rozhodol som sa rozpoznávať výrazy typu

```
53 + 119.5 - 0.12
1.33 * (590 - 518.2) + 26
```

To znamená, čísla od nula do tisíc, vrátane desatinných (tri desatinné miesta), operácie medzi číslami a zátvorky.

Gramatika, podľa ktorej sa rozpoznávač riadi začína nasledovne:

```
$jednotky = jedna|dva|dvě|tři|...;
$nast = deset|jedenáct|dvanáct|třináct|...;
$desiatky = dvacet|třicet|čtyřicet|...;
$stovky = sto|dvě stě|tři sta|čtyři sta|...;
```

Ďalej definujeme, ako sa vytvorí celé číslo do 1000:

```
$numInt1 = $stovky;
$numInt2 = [$stovky] $desiatky;
$numInt3 = [$stovky] [$desiatky] $jednotky;
$numInt4 = [$stovky] $nast;

$numberInt = $numInt1|$numInt2|$numInt3|$numInt4;
```

Celé čísla máme, desatinné vzniknú drobnou úpravou:

```
$decPlace = celá|celých|celé;

$numberFloat = (nula|$numberInt) ($decPlace)
               (nula|$numberInt);
```

Spojíme celé aj desatinné čísla a definujeme operácie, ktoré môžu byť medzi číslami:

```
$num = ($numberInt) | ($numberFloat);

$op = plus|mínus|krát|děleno|lomeno;
$brOpen = závorka;
$brClose = zavřít|zavřít závorku;
```

Výsledný rozpoznávaný výraz zapíšeme na záver:

```
[$brOpen] $num <[$op [$brOpen] $num [$brClose]]>
```

Uvedomujem si, že s tými zátvorkami to nie je vyriešené najlepšie[18], no nevymyslel som v BNF lepší zápis. Pridám len informáciu, že zátvorky `[,]` hovoria o nepovinnosti argumentu, zátvorky `<,>` tvrdia, že argument sa bude opakovať 1 až n krát. Znak `|` odďeľuje varianty.

Každopádne, s takto obmedzenou gramatikou a kvalitným tréningom sa ukazuje skutočná sila tohto prístupu. Za celkom dobrých podmienok – málo šumu na pozadí, cvičený rečník[19] sa dosahuje 100% presnosti rozpoznania.

Ako čerešničku som pridal vyhodnotenie rozpoznaných matematických výrazov. Na starosti to má skript `string_to_math.pl`, ktorý číta štandardný vstup, očakáva textový reťazec reprezentujúci matematický výraz, napríklad:

```
sto čtyřicet jedna plus dvanáct celá dvacet šest
```

Tento následne prevedie na číselný zápis:

```
141 + 12.26
```

Následným zavolaním interpretra PERL-u sa textový výraz v premennej `$output` vyhodnotí. Presnosť je nastavená na dve desatinné miesta operáciou krát sto, pretypovaním na celé čísla a podelením sto.

```
perl -e 'print (int((\$output)*100)/100)'
```

3.8 Test parametrov trénovania

Po skonštruovaní rozpoznávača sa naskytala otázka, aké parametre kódovania a parametre modelov zvoliť. Napísal som krátke skripty, ktoré vykonávajú zmenu parametrov, znovu-trénovanie rozpoznávača a napokon vyhodnotenie presnosti.

[18] Vidí čitateľ kde je zrada so zátvorkami v gramatike?

[19] Cvičeným rečník vie, ako má robiť pauzy medzi slovami, vie, ako mikrofón držať a nefúkať doň. Cvičený rečník vznikne obvykle zácvikom za 10-15 minút.

Štruktúru som navrhol nasledovne. V danom adresári sa vytvoria číselné (zložené len z číslic 0-9) podadresáre (ktoré nie sú číselné, skript neuvažuje). V každom z nich je konfiguračný súbor `hcopy.cfg`, ktorý určuje parametre kódovania a ďalej súbor `models.pcf`, čo je formálna definícia modelu, popísaná v sekcii 3.6.1.

Skript `incremental_tests.sh` skopíruje súbory z prvého adresára do rozpoznávača, spustí tréning modelov a následne otestuje presnosť na testovacej sade. Toto vykoná pre každý z adresárov a výsledky ukladá do súboru `test_result` v každom z podadresárov.

Celá činnosť skriptov je priehľadná, odporučím teda čitateľa do adresára `incremental_tests`, kde môže skúmať detaily.

Výsledkami testov som nedospel k ničomu prevratnému, skôr som sa utvrdil v mojich predošlých predstavách. Testoval som vplyv počtu stavov modelu na presnosť, testoval som výhodnosť koeficientov MFCC verzus LPC a tiež som skúšal rôzne topológie skrytého Markovovho modelu. To všetko pre rozpoznávanie závislé a nezávislé na rečníkovi.

3.8.1 Topológie modelu

Topológiou modelu sa myslí štruktúra stavov a prechodov medzi nimi. Pre mňa to znamenalo úpravu matice pravdepodobností prechodu. Graficky znázornené, definujeme,z ktorého do ktorého stavu je možný prechod. Tým určíme, ako dokáže model vstrebať prípadné natiahnutia alebo skrátenia vo vyslovovaní.

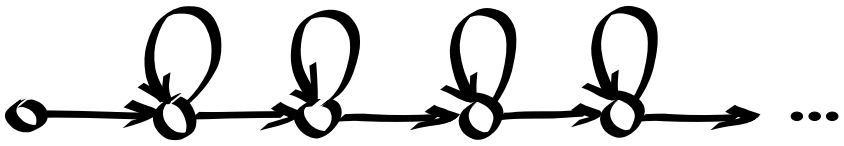
Skúšal som tri štruktúry modelu, nazval som ich topológia A, B, C. Ich znázornenie je na obrázkoch 3.8, 3.9 a 3.10.

Aby som nemusel pre každý počet stavov ručne písať maticu pravdepodobností prechodu, vytvoril som sekciu v súbore `models.pcf`, ktorá pre topológiu A vyzerá:

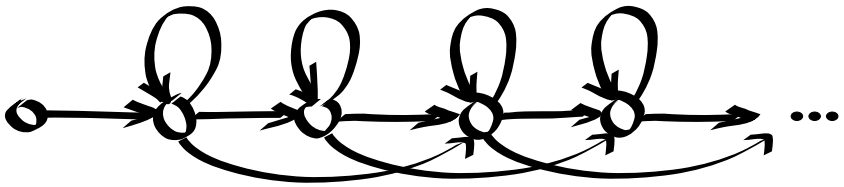
```
<BEGINtransp>
  0.000e+0 1.000e+0
  6.000e-1 4.000e-1
<ENDtransp>
```

Topológia B pridáva naviac prechod do stavu $n + 2$:

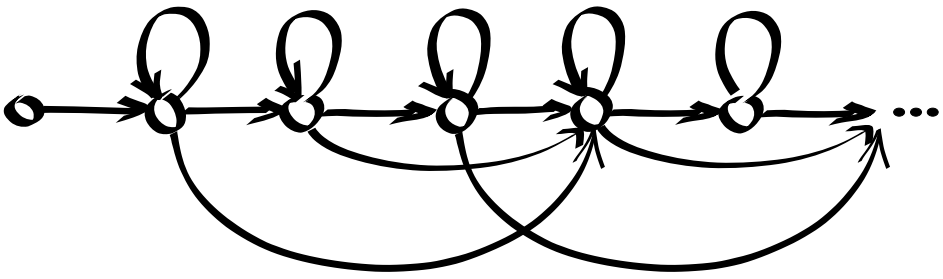
```
<BEGINtransp>
  0.000e+0 1.000e+0
  5.000e-1 3.000e-1 2.000e-1
<ENDtransp>
```



Obr. 3.8: Topológia A



Obr. 3.9: Topológia B



Obr. 3.10: Topológia C

V poslednom type sú z každého druhého stavu namiesto prechodov objedna prechody obdva.

```
<BEGINtransp>
  0.000e+0 1.000e+0
  5.000e-1 3.000e-1 0.000e+0 2.000e-1
  5.000e-1 3.000e-1 2.000e-1
<ENDtransp>
```

3.8.2 Závislé na rečníkovi

Najprv som chcel otestovať parametre na závislom rozpoznávači. Nechal som chvíľu sťahovať stránky z internetu, potom som vyseparoval slová a 1340 z nich vybral ako trénovaciu množinu. Testovacia sada vznikla ako 5 krát 100 slov, každá stovka bola náhodne vybraná z trénovacej sady. Z 500 slov bolo 487 unikátnych. Všetky vzorky nahovoril jeden rečník – moja maličkosť.

Cieľom testov bolo zistiť, či sa skutočne koeficienty LPC hodia viac na rozpoznávanie nezávislé na rečníkovi. Ďalej som skúmal, aký počet stavov modelu je najlepší pre fonémy.

Graf[20] 3.11 ukazuje presnosť rozpoznania, keď skryté Markovove modely mali topológiu A, v závislosti na počte stavov modelu pre MFCC a LPC vektorizáciu. Podľa predpokladov, MFCC sa správa výrazne lepšie.

Rovnaká úloha, zmena topológie modelov. Výsledky ukazuje graf 3.12. Aj tu sa javí vektorizácia MFCC ako lepšia v porovnaní s LPC. Ďalším pozorovaním je zhoršenie presnosti oboch vektorizácií oproti topológii A.

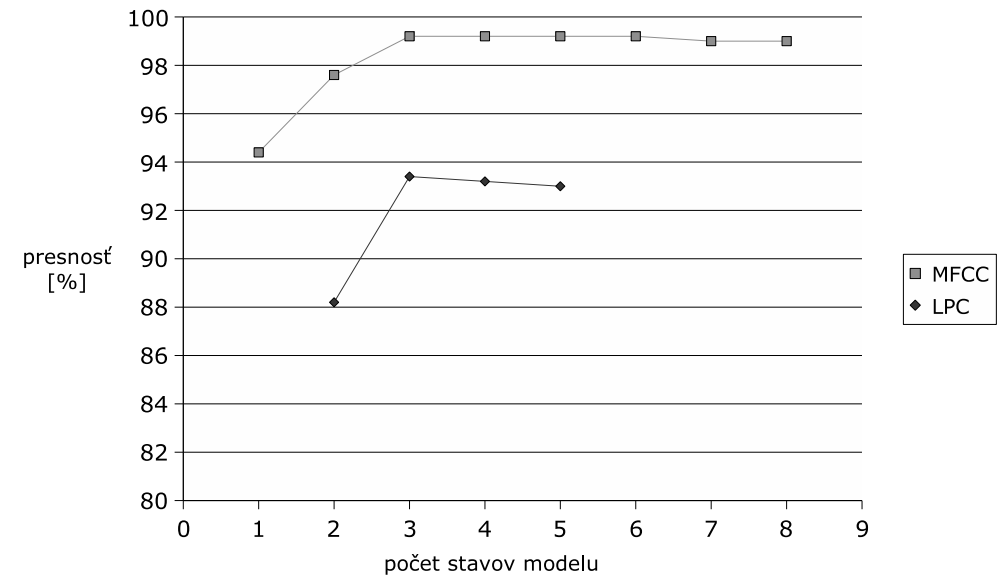
Posledná topológia (graf 3.13) dáva veľmi podobné výsledky ako predchádzajúca štruktúra.

3.8.3 Nezávislé na rečníkovi

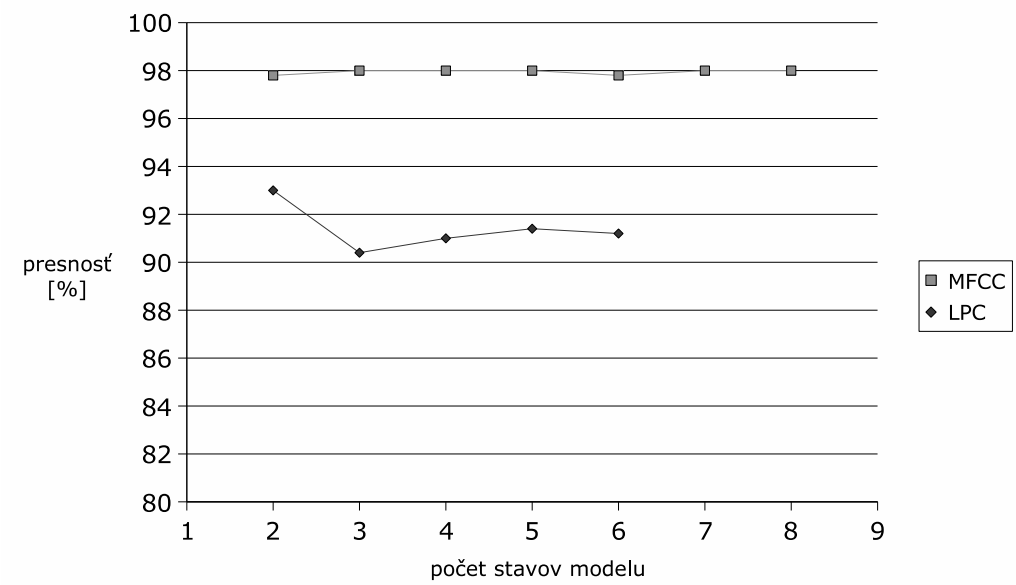
Táto časť práce vyčnievala nad ostatné najmä svojou časovou náročnosťou, pretože som sa rozhodol trénovať na troch rôznych rečníkoch a testovať na ďalších dvoch. Trénovacích vzoriek bolo dokopy 4420, testovacích, vybraných rovnakou metódou ako v predchádzajúcej úlohe, bolo 2000.

Časovo náročné bolo nie len nahrávanie vzoriek, ale aj tréning modelov a následné testovanie presnosti.

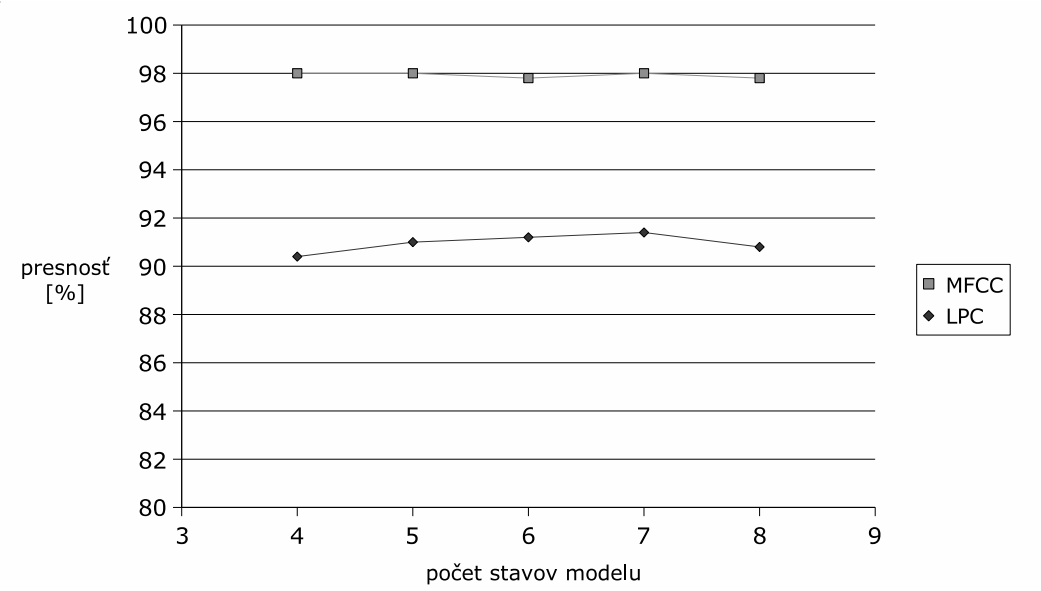
[20] Čiary spájajúce jednotlivé merania slúžia výlučne pre lepšiu orientáciu v grafe. Hodnoty, ktoré sa v grafe nenachádzajú neboli namerané alebo nemajú v úlohe zmysel. Z optimalizačných dôvodov neboli počítané hodnoty vo všetkých stavoch.



Obr. 3.11: Závislosť na rečníkovi, topológia A



Obr. 3.12: Závislosť na rečníkovi, topológia B



Obr. 3.13: Závislosť na rečníkovi, topológia C

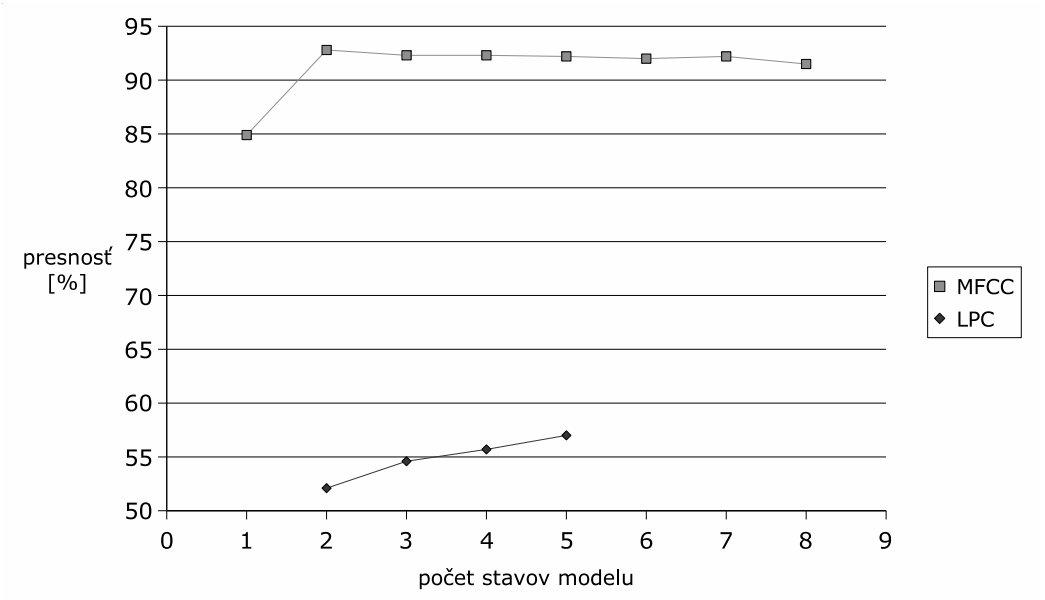
Tento experiment mal ukázať, či sú koeficienty LPC lepšou voľbou ako MFCC. Zaujímal ma aj pokles presnosti oproti závislému rozpoznávaniu.

Prvý graf (3.14) ukazuje topológiu A a presnosť rozpoznávania v závislosti na počte stavov modelu. Opäť robíme výpočet pre koeficienty LPC aj MFCC. Povšimnime, že y-ová os grafu je preškálovaná oproti predchádzajúcej úlohe.

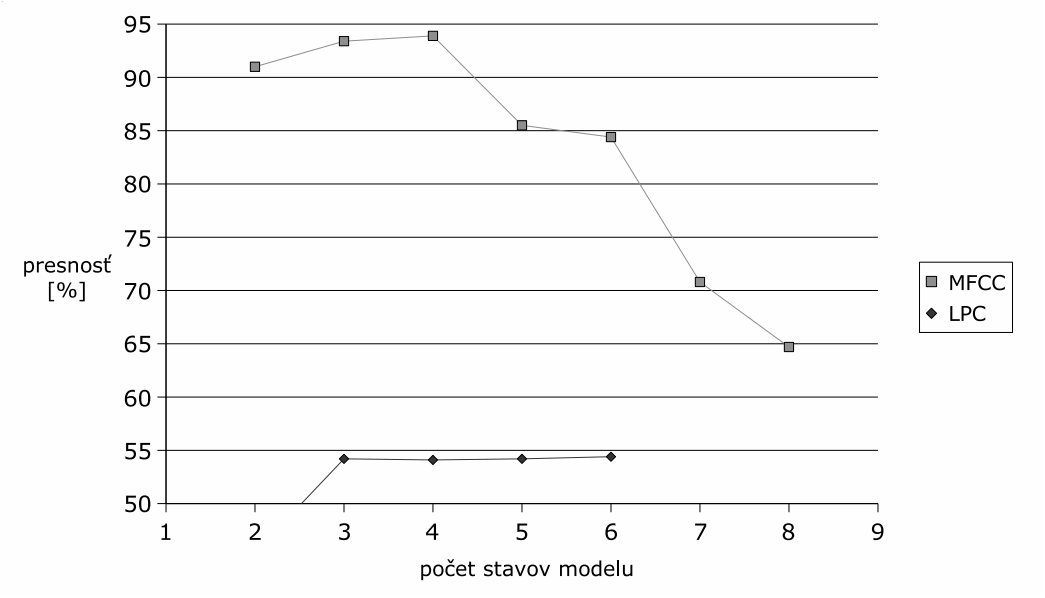
Prekvapením tohto pokusu bolo výrazné zaostávanie vektorizácie LPC. Snažil som sa hľadať chybu v niečom inom, menil som počet koeficientov na jedno okienko, počet kanálov, no výsledok sa nezlepšil. Zdá sa mi totiž dosť nepravdepodobné, že by presnosť rozpoznania vystúpala len tesne nad 50 percent.

Topológie B (3.15) a C (3.16) sa stali pri vyššom počte stavov zatratenia hodné.

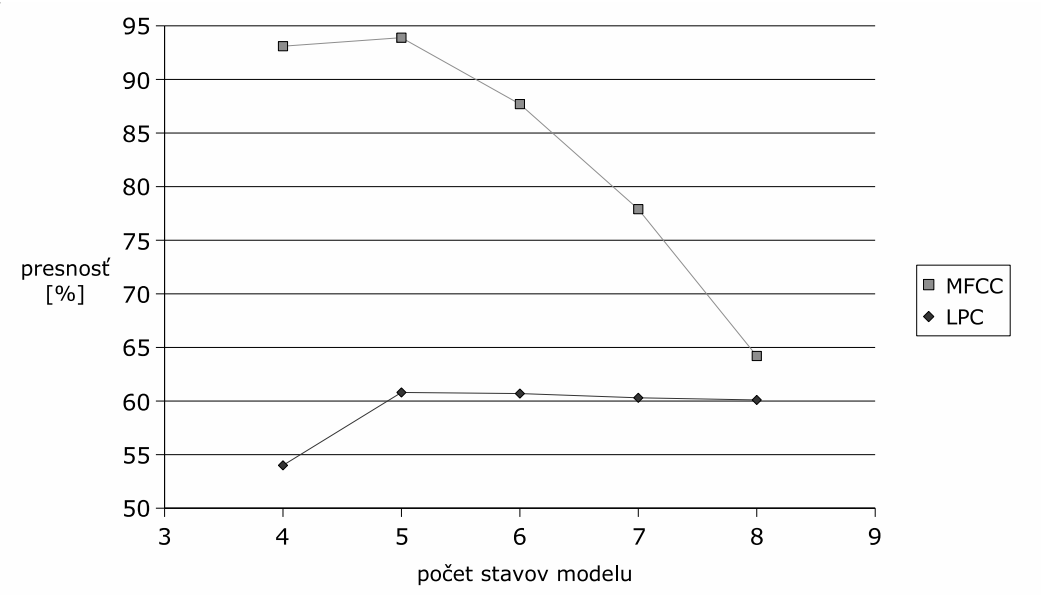
Na záver som porovnal najlepšie výsledky jednotlivých topológií. Na rečníkovi závislé rozpoznávanie (3.17) ukazuje prevahu topológie A. No pri nezávislom rozpoznávaní (3.18) to už nie je také jasné, topológie B a C predčili chvíľami A, ale táto topológia sa javila veľmi dobre vo všetkých experimentoch, preto som si na tréning rozpoznávača vybral práve ju.



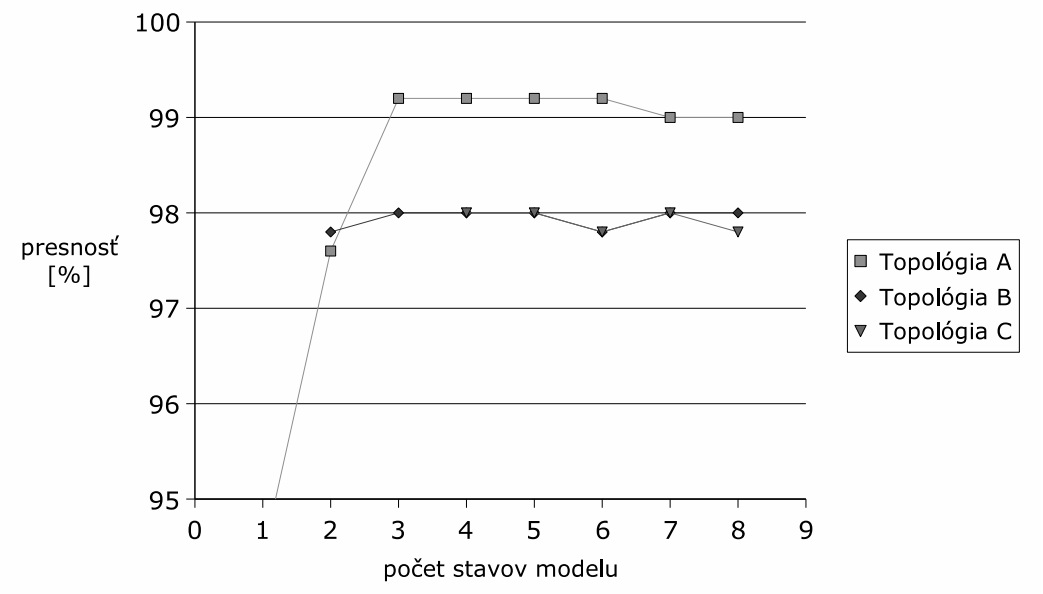
Obr. 3.14: Nezávislosť na rečníkovi, topológia A



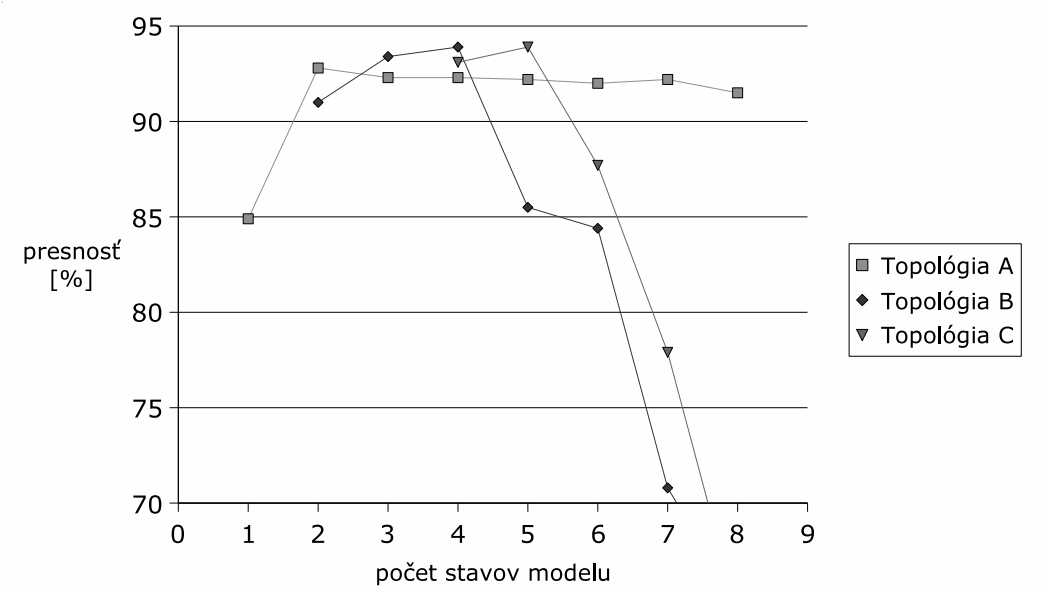
Obr. 3.15: Nezávislosť na rečníkovi, topológia B



Obr. 3.16: Nezávislosť na rečníkovi, topológia C



Obr. 3.17: Závislosť na rečníkovi, topológie modelu



Obr. 3.18: Nezávislosť na rečníkovi, topológie modelu

3.8.4 Výsledky

Na týchto pár experimentoch som otestoval vhodnosť jednotlivých topológií. Výsledky hovoria, že topológia A je obzvlášť vhodná pri závislom rozpoznávaní. 3-5 stavov modelu sa zdá byť vhodným počtom na tréning foném. Z experimentov ďalej vyplýva, že koeficienty MFCC sa vo všeobecnosti hodia viac.

3.9 Rozpoznávací framework

Pri každom novom pokuse som bol postavený pred úlohu, znovu a znovu kopírovať konfiguračné súbory, vytvárať adresáre a podobne. Vytvoril som preto skript (`construct_framework.sh`), ktorého spustením sa nám vytvorí adresár `framework_output`. V tomto už bude všetko pripravené na dodanie rečových vzoriek a popisných súborov, prípadne špeciálnej gramatiky.

Predpokladám, že to výrazne uľahší úvodné pokusy začínajúcim experimentátorom. Všetky tu popisované príkazy boli pre lepší prehľad v texte zjednodušené, ale nebola porušená (aspoň dúfam) korektnosť. Ich plné znenie je zapísané v súboroch `step1.sh` až `step6.sh`. Napokon zhrňujúci skript (`all_steps.sh`) volá jednotlivé kroky.

Pre dávkové testovanie je poskytnutý skript `test1.sh`.

3.9.1 Ako rýchlo vybudovať rozpoznávač?

Najprv nutno premyslieť, čo by sa malo rozpoznávať. Obvykle to budú slová, krátke spojenia alebo vety. Vytvoríme súbor, ktorý bude obsahovať stovky riadkov s výrazmi podobnými v budúcnosti rozpoznávaným. Skriptom `number_prompts.pl` sa riadky očíslujú a vznikne tak súbor s výzvami (`trainprompts`). Ten bude vstupom programu `record_prompts`. Spustením s príslušnými parametrami sa začne nahrávanie dát, v ktorom program je sprievodcom. Obvykle je dobré vypočuť si nahrané vzorky a extrémne rušené zmazať – následné spustenie `record_prompts` to spozná a nechá ich prečítať znovu.

Po úspešnom nahrávaní nutno skopírovať wav súbory do adresára `data_train`, súbor `trainprompts` do adresára `prompts`. Spustiť skript `all_steps.sh`.

Za pár chvíľ – v závislosti na počte dát, pre tisíc slov je to 5-10 minút, dostaví sa výsledok v podobe natrénovaných modelov. Tieto môžeme skopírovať do živého rozpoznávača `live_input`, prípadne doladiť gramatiku, lebo štandardne je až príliš voľná.

Hotovo, rozpoznávanie beží, s trochou snahy, odvážim sa tvrdiť, nezasvätený človek zvládne to za hodinku-dve.

3.9.2 Problémy

Skrytý Markovov model, je modelom štatistickým. Skutočne, treba mu poskytnúť veľké množstvo dát, aby bol tréning kvalitný. Potýkal som sa často s problémom segmentácie jednotiek. Slovo sa skladá z foném. Ale ako určiť, kde sa fonéma začína a kde končí v rečovom signále? Pri veľa výskytoch fonémy v rôznych kontextoch, sa to napokon naladí správne, no častokrát je obtiažne získať tak veľa dát.

Ďalším východiskom by mohla byť ručná segmentácia. Vyžaduje si to veľa času – každú rečovú vzorku skontrolovať a z vizualizovaného signálu určiť, kde majú fonémy hranice. Pravda, to vyžaduje skúsenosti so vzhľadom rečového signálu.

Tak často spomínaný problém výpočtovej náročnosti sa postupne so stále rýchlejšími počítačmi uťahuje do pozadia.

Záver

Očakávaným záverom je vyhodnotenie naplnenia cieľa práce. Budem sa toho držať a tvrdím, že cieľ bol naplnený. Rozpoznávač funguje, nie stopercentne, ale kritérium na presnosť nebolo dané. Odhadom, rozpoznávanie nezávislé na rečníkovi funguje v okolí 90% pre gramatiku tisíc slov. Rozpoznávanie závislé na rečníkovi, pri gramatike rozpoznania matematických výrazov ide blízko ku 100%.

Zistil som, že skryté Markovove modely sú skutočne opodstatnené pri rozpoznávaní reči. Naviac, existujú knižnice, ktoré prácu s nimi uľahčujú a programovanie sa stáva skutočnou zábavou.

Nutno na záver konštatovať, že podľa mojich dojmov, samotné rozpoznávanie je dosť presné. Veď aj človek má častokrát problém rozumieť, ak nepozná kontext, nie je zvyknutý na hlas rečníka a pod. Myslím, že so sofistikovaným jazykovým modelom, ktorý by dokázal odhadovať, čo mohlo byť v danej chvíli najpravdepodobnejšie povedané by rozpoznávanie reči vstúpilo medzi bežných ľudí.

Literatúra

[bash] GNU Bourne-Again Shell
http://www.gnu.org/software/bash/

[Bnf] About BNF notation
http://en.wikipedia.org/wiki/Backus-Naur_form

[Com04] Comparaison of several acoustic modeling techniques and decoding algorithms for embedded speech recognition systems
http://www.lia.univ-avignon.fr/fich_art/419-article.pdf

[Htk] The Hidden Markov Model Toolkit
http://htk.eng.cam.ac.uk/

[Htk05] The HTK Book (for HTK Version 3.3)
http://htk.eng.cam.ac.uk/prot-docs/htkbook.pdf

[perl] Perl documentation
http://perldoc.perl.org/

[phoCz] Česká výslovnosť
http://home.unilang.org/main/wiki2/index.php/Czech_pronunciation

[Psu95] Psutka Josef: Komunikace s počítačem mluvenou řečí, Academia, Praha, 1995

[smpSk] SAMPA for Slovak
http://www.ui.sav.sk/speech/sampa_sk.htm

[smpCz] SAMPA for Czech
http://noel.feld.cvut.cz/sampa/

[] ...helping dyslexic people make the most of their abilities
http://www.dyslexic.com/

[] Elektronický lexikón slovenského jazyka
http://www.forma.sk/onlines/slex/index.asp

[] Open Source Scalable Vector Graphics Editor
http://www.inkscape.org/

[] The International Phonetic Association
http://www2.arts.gla.ac.uk/IPA/

[] C library for reading and writing files containing sampled sound
http://www.mega-nerd.com/libsndfile/

[] Objektově orientované programování v C++
http://www.builder.cz/serial24.html

[] Getting Started With POSIX Threads
http://dis.cs.umass.edu/~wagner/threads_html/tutorial.html

[] Praat – Speech analysis and synthesis program
http://www.praat.org

[] Seman Ondrej: Ovládanie počítača hlasom, diplomová práca MFF UK, Bratislava, 2004
http://delo.dcs.fmph.uniba.sk/~seman/dipl/diplomka.pdf

[] The Carnegie Mellon Sphinx Project
http://cmusphinx.sourceforge.net/html/cmusphinx.php

[] Standard Template Library Programmer's Guide
http://www.sgi.com/tech/stl/

[] A C++ implementation of posix threads
http://threads.sourceforge.net/

[] Návrh aplikací v jazyce UML
http://interval.cz/clanek.asp?article=2783

[] Speech recognition – Wikipedia
http://en.wikipedia.org/wiki/Speech_recognition

Dodatok A

Fonémy a pravidlá fonetického prevodu

Pre úplnosť prikladám zoznam foném, ktoré používam vo všetkých skriptoch. Je to fonetický zápis hlások a hlások v kontexte pomocou znakov ASCII[1].

Ďalej popisujem výslovnostné pravidlá, podľa ktorých prepisujem text na fonetický tvar. Všetku prácu na fonetickom preklade som implementoval PERL-ovským skriptom `sampa_translate_cz.pl`.

[1] American Standard Code for Information Interchange

A.1 Špeciálne fonémy

Aby sme odbúrali problémy so zápisom písmenok v národných jazykoch a aby sme mohli lepšie modelovať reč, zavedieme prepis pravopisných symbolov na fonémy. V tabuľke sú vyznačené iba fonémy, ktoré sa líšia od pravopisného zápisu. Inšpiráciou mi bola fonetická abeceda SAMPA [smpCz].

Fonéma	Pravopis	Iba v kontexte	Príklad
i:	í,ý	–	pít
e:	é	–	lék
a:	á	–	rád
o:	ó	–	móda
u:	ú	–	údolí
u:	ů	–	půl
o_u	ou	–	mouka
a_u	au	–	auto
e_u	au	–	euforie
i_^e	au	–	teorie
c	t	zmäkčené t	tito
J	n	zmäkčené n	nic
J/	d	zmäkčené d	děd
ts	c	–	cíl
tS	č	–	čas
dz	c	ak nasleduje znelo vyslovená spoluhláska	leckdy
dZ	č	viď dz	léčba
Q/	ř	ak pred je t,v,p,b,d,k	tři
P/	ř	zvyšné prípady	řád
Z	ž	–	žal
S	š	–	šaty
x	ch	–	chata
N	n	ak nasleduje k	banka
N/	n	ak nasleduje s	Slovensko
mF	mf	–	amfiteáter
w	v	na začiatku slova	vlak
l=	l	ak nasleduje k	vlk
m=	m	pred je s	osm
r=	r	pred je k	krk

A.2 Pravidlá českej výslovnosti

V češtine, na rozdiel od angličtiny, sa výslovnosť prevažne riadi pravopisom. Preto sa dajú automatizovane generovať výslovnostné slovníky, treba však dodržať pár pravidiel. Ako zdroj pre konštrukciu pravidiel mi slúžil [phoCz].

Zmäkčovanie

- **d, t, n** – sa vyslovujú mäkko, ak za nimi nasleduje `ě, i, í`.
- **ě** – vkladá sa predeň `j` (bje, pje, vje) alebo `ň` (mňe).

Spodobovanie

Najprv uveďme zoznam znelých a neznelých spoluhlások.

Znelé: `b, v, d, ď, z, ž, g, h, dz, dž`

Neznelé: `p, f, t, ť, s, š, k, ch, c, č`

Spodobovaním potom myslíme zmenu výslovnosti v kontexte.

- znelé spoluhlásky sú vyslovované ako neznelé na konci slov (lev [lef])
- ak je viac spoluhlások za sebou, výslovnosť predošlých sa zmení podľa poslednej (kto [gdo], tužka [tuška])
- výnimky:
  - ak znelé `v` nasleduje po neznelej spoluhláske, nechá ju na pokoji (svět [svjet])
  - znelé `h` písane po neznelom `s` môže ponechať výslovnosť alebo sa samo zmeniť na neznelý protějšok `ch` ([naschledanou] verzus [nazhledanou])

Ďalšie javy

Určite existuje niekoľko výnimiek, na ktoré sa nehodia žiadne pravidlá. Z českých slov sú to napríklad `sedm, osm`, ktoré je možné vysloviť aj ako `sedum, osum`.

Špeciálnou skupinou sú slová cudzieho pôvodu, výslovnosť týchto nutno definovať oddelene. Napríklad `botanik, ministr, justice`.