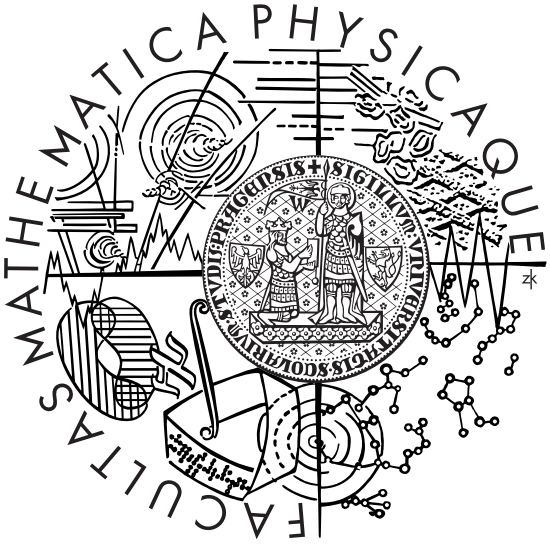
Univerzita Karlova v Praze
Matematicko-fyzikální fakulta

DIPLOMOVÁ PRÁCE



Petra Tobolíková

Detekce chyb v rozpoznávání mluvené řeči

Ústav formální a aplikované lingvistiky
Vedoucí diplomové práce: Doc. RNDr. Jan Hajič, Dr.
Studijní program: Informatika, obor Matematická lingvistika

Děkuji panu Doc. RNDr. Janu Hajičovi, Dr. za vedení diplomové práce. Děkuji také RNDr. Petru Podveskému, Ph.D. za uvedení do problematiky a poskytnutí dat z rozpoznávání řeči, Mgr. Pavlu Schlesingerovi za konzultace k použití metod strojového učení v R a Mgr. Nino Peterkovi, Ph.D. za data potřebná pro trigramový jazykový model.

Prohlašuji, že jsem svou diplomovou práci napsala samostatně a výhradně s použitím citovaných pramenů. Souhlasím se zapůjčováním práce.

V Praze dne 16. 4. 2008 Petra Tobolíková

Obsah

Název práce: Detekce chyb v rozpoznávání mluvené řeči

Autor: Petra Tobolíková

Katedra (ústav): Ústav formální a aplikované lingvistiky

Vedoucí diplomové práce: Doc. RNDr. Jan Hajič, Dr.

e-mail vedoucího: hajic@ufal.mff.cuni.cz

Abstrakt: Tématem této diplomové práce je detekce chyb v rozpoznávání mluvené řeči. Nejprve jsou stručně představeny principy současného rozpoznávání řeči. Jsou nastíněny problémy, se kterými se rozpoznávání řeči potýká a které způsobují, že stále nefunguje bezchybně. Dále jsou uvedeny stávající známé metody výpočtu tzv. skóre spolehlivosti. V následující části jsou popsány tři metody strojového učení, které byly využity pro implementovanou detekci chyb: logistická regrese, neuronové sítě a rozhodovací stromy. Poté jsou navrženy atributy slov v rozpoznaných větách, které jsou použity jako vstupní proměnné metod strojového učení. Výstupní proměnnou je odhad skóre spolehlivosti. Je zde předveden způsob, jakým byly využity implementace metod strojového učení v softwaru R. Metody byly testovány na nahrávkách českého rádia a televize. Výsledky jednotlivých metod jsou porovnány pomocí křivek ROC, směrodatné chyby detekce a možnosti redukce WER v rozpoznaných větách. Je připojen rovněž popis programu, který je součástí práce. Na závěr jsou shrnuty vlastnosti slova, které se osvědčily jako účinné atributy při detekci chyb.

Klíčová slova: rozpoznávání řeči, detekce chyb, skóre spolehlivosti, strojové učení

Title: Error detection in speech recognition

Author: Petra Tobolíková

Department: Institute of Formal and Applied Linguistics

Supervisor: Doc. RNDr. Jan Hajič, Dr.

Supervisor's e-mail address: hajic@ufal.mff.cuni.cz

Abstract: This thesis tackles the problem of error detection in speech recognition. First, principles of recent approaches to automatic speech recognition are introduced. Various deficiencies of speech recognition that cause imperfect recognition results are outlined. Current known methods of "confidence score" computation are then described. The next chapter introduces three machine learning algorithms which where employed in the error detection methods implemented in this thesis: logistic regression, artificial neural networks and decision trees. This machine learning methods use certain attributes of the recognized words as input variables and predict an estimated confidence score value. The open source software "R" has been used throughout, showing the usage of the aforementioned methods. These methods have been tested on Czech radio and TV broadcasts. The results obtained by those methods are compared using ROC curves, standard errors and possible (oracle) WER reduction. Programming documentation of the code used in the implementation is enclosed as well. Finally, efficient word attributes for error detection are summarized.

Keywords: speech recognition, error detection, confidence score, machine learning

Kapitola 1

Úvod

Řeč patří bezesporu mezi primární způsoby mezilidské komunikace. Její význam neklesá ani s rozvojem technologií a médií, spíše naopak. Telekomunikace, rozhlas, televizi a podobně si lze bez ní jen těžko představit. A podobně, jako člověk používá řeč při kontaktu s jiným člověkem, by ji rád uplatnil také při komunikaci s počítačem. Počítače, které naslouchají, rozumí lidské řeči a v lidské řeči také odpovídají, patří stále ještě do výjevů ze sci-fi filmů. Avšak i méně vyspělé použití tohoto komunikačního prostředku v počítačových technologiích může do značné míry ovlivnit způsob, jakým budeme počítače používat.

Prvním nezbytným krokem na cestě k využívání mluvené řeči je její automatické rozpoznávání. Úkolem automatického rozpoznávání mluvené řeči (v angličtině se často označuje zkratkou *ASR – automatic speech recognition*) je převod mluveného slova do podoby srozumitelné pro počítač, obvykle také do textové podoby čitelné pro člověka.

Automatické rozpoznávání mluvené řeči nachází již dnes uplatnění v široké škále aplikací. Použitelnost je často podmíněna omezením na úzkou oblast úkolů, tj. omezenou slovní zásobu pro specializovaný obor nebo předem stanovené příkazy, které má počítač rozpoznávat, apod. Na základě mluvených pokynů dnes funguje např. hlasové vytáčení v mobilních telefonech, vyhledávání v audio databázích, jednoduché vkládání dat (např. vyplňování formulářů), diktování textu textovým procesorům nebo diktování e-mailových zpráv. Ke složitějším aplikacím rozpoznávání řeči, které ještě čekají na své zdokonalení, patří mimo jiné ovládání počítače hlasem vhodné pro tělesně postižené osoby, automatický překlad mluveného slova, kontrola výslovnosti v programech pro výuku cizích jazyků, využití v robotice nebo např. pro automatizované ovládání různých přístrojů v domácnostech.

1.1 Problémy automatického rozpoznávání řeči

Úloha automatického rozpoznávání řeči se potýká s mnoha obtížemi, které vyplývají ze samotné povahy lidské řeči. Když člověk mluví, nikdy nevysloví stejné slovo dvakrát naprosto totožným způsobem. Je to ovlivněno mnoha faktory, z nichž jme-

nujme namátkou zdůrazňování slov, odlišné vyslovování slov na začátku a na konci věty, různé melodie různých typů vět (oznamovací vs. tázací), vliv předchozích a následujících slov ve větě (v češtině tzv. spodoba znělosti), emoce, náladu, případně zdravotní stav člověka.

Problémy mohou nastat též při určování hranic mezi slovy. Člověk nevyslovuje slova jako zřetelně oddělené jednotky, slova naopak úzce navazují a např. neslabičné předložky splývají s jiným slovem úplně. Akustická informace bývá nedostatečná a zpravidla je potřeba využít kontextu slov a dodat hlubší jazykové znalosti – syntaktické, případně sémantické.

Programy, které si kladou za cíl rozpoznávat spontánní mluvenou řeč, se musejí vypořádat se skutečností, že struktura spontánní řeči se značně liší od řeči připravené (diktování připraveného textu, čtení zpráv apod.), která má co do jazykové struktury spíše povahu psaného slova. Věty vyslovené v rámci spontánního projevu často neodpovídají syntaktickým pravidlům jazyka. Projevuje se v nich koktání, zadrhávání, opakování slov, slovní vycpávky jako „ehm“, „tak“, „prostě“ atd., věty mohou být neúplné.

Problémy se objeví také ve chvíli, kdy bychom chtěli do rozpoznávaného textu automaticky doplňovat interpunkční znaménka nebo dokonce rozdělovat text na odstavce. Při běžné komunikaci se k tomuto účelu používají pauzy, melodie hlasu apod., avšak zdaleka ne všem pauzám v řeči odpovídá čárka v textu a naopak. Člověk musí při psaní zohledňovat pravidla pravopisu příslušného jazyka, k rozdělení na větné celky používá další informace o struktuře a významu vět i o příslušném tématu. Při diktování počítači se tento problém řeší tak, že mluvčí explicitně diktuje všechna interpunkční znaménka a pokyny k formátování.

V praxi je často potřeba, aby rozpoznávače řeči zvládaly zapisovat řeč různých mluvčích. Mnohé programy se umí naučit rozpoznávat řeč více mluvčích, data každého z nich však ukládají zvlášť a před samotným rozpoznáváním vyžadují informaci o tom, kterého mluvčího mají očekávat. Programy zatím neumí spolehlivě identifikovat hlasy různých osob v průběhu rozpoznávání a automaticky přepínat např. během dialogu. Rozpoznávání řeči cizích mluvčích bez předchozího tréninku zatím nedosahuje dostatečně přesných výsledků. Potíže jsou způsobeny rozdílnými charakteristikami hlasů mluvčích, odlišnostmi ve výslovnosti, nestejnou rychlostí řeči a rozličnými přízvuky.

Tím však výčet aspektů, které komplikují automatické rozpoznávání řeči, zdaleka nekončí. Úlohu dále ztěžuje kupř. používání cizích slov, vlastních jmen, odborných či slangových termínů, okolní šum a hluk atd.

Je tedy zřejmé, že problematika detekce chyb v automatickém přepisu řeči je stále aktuální. Označování problematických úseků přepisu může být využito např. pro vylepšení dialogových systémů. Jedná se o počítačové systémy, které poskytují člověku informace na základě hlasové komunikace, zpravidla telefonicky. Mohou zprostředkovávat informace o odjezdech vlaků nebo čtení aktuálních zpráv... Pokud takový systém obdrží požadavek, při jehož rozpoznávání vznikají chyby, může si vyžádat jeho zopakování, případně přeformulování.

Kapitola 2

Automatické rozpoznávání řeči

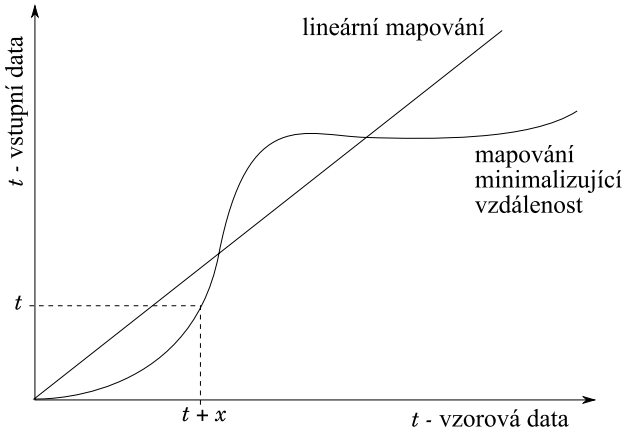
Prvotním krokem při rozpoznávání mluvené řeči je **analýza akustického signálu** řeči, kterou se zabývá samostatná vědní disciplína – digitální zpracování signálu. Obvykle se používá některá z metod tzv. krátkodobé analýzy, při nichž se úseky řečového signálu vydělují a zpracovávají tak, jako by to byly oddělené krátké zvuky. Tyto segmenty jsou reprezentovány většinou časovým úsekem 10 až 30 ms. Výsledkem analýzy je pak číslo nebo soubor čísel, které popisují daný segment. Protože segmenty na sebe navazují, eventuálně se překrývají, dostáváme časové posloupnosti čísel, které popisují daný promluvený celek. Metody krátkodobé analýzy většinou předpokládají, že jako vstup zpracovávají data získaná digitalizací, tj. kódováním tvaru zvukové vlny. Podrobnější informace o této problematice a o tématu rozpoznávání řeči vůbec uvádí např. Psutka [5], v anglickém jazyce pak Huang a kol. [2].

Výsledek zpracování akustického signálu řeči je dále předán algoritmu rozpoznávání řeči. Většina dnešních rozpoznávačů používá algoritmy založené na tzv. skrytých Markovových modelech, anglicky *hidden Markov models (HMMs)*. Starším přístupem je metoda *dynamic time warping (DTW)* (české označení „dynamické borcení času“ se příliš nepoužívá).

2.1 Dynamic time warping

Tato poměrně jednoduchá metoda je založena na hledání minimální vzdálenosti mezi vstupními a vzorovými daty. Vzorovými daty mohou být digitalizované nahrávky slov spolu s jejich přepisy, které tvoří slovník, nebo např. i vhodná reprezentace videa. Vstupními daty je zde nahrávka, která se má rozpoznat. Na výstup je předán přepis slova, u nějž je vzdálenost mezi vzorem a vstupem nejmenší ze všech dostupných vzorů. Může být stanovena hranice pro maximální přijatelnou vzdálenost a pokud ji vypočtené vzdálenosti u všech vzorů přesáhnou, výsledkem je chyba (neznámé slovo).

Algoritmus neporovnává pouze hodnotu vstupních dat v čase t s hodnotou vzorových dat v tomtéž čase, ale hledá takové mapování ze sekvence časových údajů ve vstupních datech do sekvence časových údajů ve vzorových datech, pro které je



Obrázek 2.1: Myšlenka DTW

součet vzdáleností odpovídajících úseků dat minimální. Mapovací funkce nemusí být vždy lineární, podmínkou je pouze neklesající charakter. To znamená, že pořadí událostí v porovnávaných datech musí být shodné. Příklad je naznačen na obrázku 2.1. Mapování může vypadat např. tak, že časy t_1, t_2 ve vstupních datech odpovídají po řadě časům $t_1 + 1$ s, $t_2 - 1$ s ve vzorových datech. Odtud pochází označení „borcení času“. Pro rozpoznávání řeči je tato vlastnost algoritmu výhodná, neboť umožňuje správně rozpoznávat i slova, která byla vyslovena jinou rychlostí než jejich uložený vzor. Rychlost se dokonce může průběžně měnit.

Přívlastek *dynamic* u jména metody naznačuje, že se pro počítání minimální vzdálenosti používá tzv. dynamické programování. Dynamické programování je jedním ze způsobů řešení optimalizačních úloh a je založeno na Bellmanově principu optimality: „Podstrategie optimální strategie je opět optimální.“

Nevýhodou metody DTW je její velká časová náročnost a také nutnost formulovat metriku pro výpočet vzdálenosti mezi daty, což může být někdy obtížné. Metoda se hodí pouze pro menší aplikace, jako je hlasové vytáčení v mobilních telefonech, rozpoznávání izolovaných číslic apod. Později byla nahrazena úspěšnějšími statistickými metodami, které používají skryté Markovovy modely.

2.2 Skryté Markovovy modely

Markovovy modely byly nejprve aplikovány na problematiku rozpoznávání izolovaných slov, tj. odděleně vyslovovaných slov z pevně daného slovníku omezené velikosti. Později bylo jejich použití úspěšně rozšířeno na rozpoznávání souvislé řeči.

2.2.1 Rozpoznávání izolovaných slov

Každé vyslovené slovo může být reprezentováno posloupností řečových vektorů (vektorů čísel, jež popisují jednotlivé segmenty řečového signálu) způsobem

$$\mathcal{P} = p_1, p_2, \ldots, p_T,$$

kde $\mathcal{P}$ je pozorování – vyslovené slovo – a p_t je řečový vektor pozorovaný v čase t. Problém rozpoznávání izolovaných slov pak může být popsán jako problém výpočtu

$$\arg\max_i \{P(w_i|\mathcal{P})\}, \tag{2.1}$$

kde w_i je i-té slovo ve slovníku a $P(w_i|\mathcal{P})$ označuje pravděpodobnost, že vysloveným slovem bylo slovo w_i za podmínky, že na vstupu rozpoznávače je pozorování $\mathcal{P}$. Tuto pravděpodobnost není reálně možné spočítat přímo, lze ji ale upravit s použitím Bayesova pravidla pro podmíněnou pravděpodobnost do tvaru

$$P(w_i|\mathcal{P}) = \frac{P(\mathcal{P}|w_i)P(w_i)}{P(\mathcal{P})}. \tag{2.2}$$

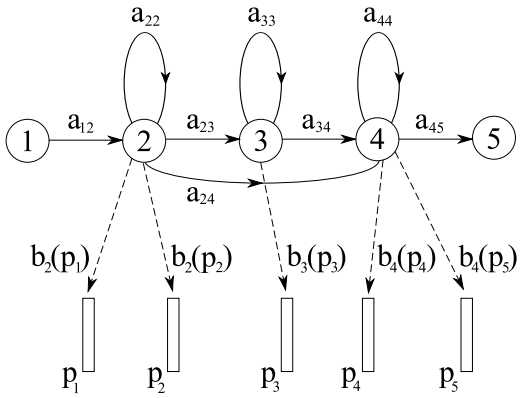
Pravděpodobnosti $P(w_i)$ vyslovení jednotlivých slov ze slovníku bývají předem dány. Z výše uvedeného vztahu tedy vyplývá, že pro určení slova, které bylo vysloveno s největší pravděpodobností, potřebujeme spočítat pouze pravděpodobnosti $P(\mathcal{P}|w_i)$. Pozorování $\mathcal{P}$ je však vektor a přímý výpočet sdružené podmíněné pravděpodobnosti $P(p_1, p_2, \ldots, p_T|w_i)$ není prakticky možný. Místo toho lze využít právě skryté Markovovy modely a problém převést na problém odhadu parametrů těchto modelů.

Vychází se z představy, že posloupnost řečových vektorů pro určité slovo je generována Markovovým modelem, jehož jednoduchý příklad je na obrázku 2.2. Základem Markova modelu je konečný stavový automat. Formálně jej lze popsat jako pětici $M = (Q, V, \mathbf{A}, \mathbf{B}, \vec{\pi})$, kde

- $Q = \{q_1, \ldots, q_N\}$ je množina N stavů Markovova modelu,
- $V = \{v_1, \ldots, v_M\}$ je abeceda M výstupních symbolů – akustických vektorů,
- $\mathbf{A} = (a_{ij})$ je matice přechodu, jejíž prvky určují, s jakou pravděpodobností přechází systém ze stavu i v kterémkoliv čase t do stavu j v čase $t + 1$,
- $\mathbf{B} = (b_j(s))$ je matice pravděpodobností generovaných symbolů, jejíž prvky určují, s jakou pravděpodobností je v kterémkoliv čase t generován symbol $s \in V$, je-li systém ve stavu j,
- $\vec{\pi}$ je sloupcový vektor pravděpodobností počátečního stavu, $\pi_i = P(q(1) = q_i)$. Pro účely modelování řeči se obvykle používají Markovovy modely s jedním pevně daným počátečním stavem a potom $\pi_1 = 1, \pi_i = 0\ \forall i \neq 1$.

Skrytý Markovův model pro každou časovou jednotku provede přechod mezi stavy a pokud v čase t vstoupí do stavu j, vygeneruje vektor p_t s pravděpodobností $b_j(p_t)$. Při modelování mluvené řeči se využívají zejména tzv. levo-pravé (*left-to-right*) Markovovy modely. Proces generování tímto modelem začíná v počátečním stavu a se vzrůstajícím časem dochází k přechodům ze stavů s nižšími indexy do stavů s vyššími indexy nebo setrvání ve stejném stavu. Na konci procesu se model nachází v koncovém stavu, který také obvykle bývá jeden (a má index N).

Obrázek 2.2 ukazuje příklad generování vektorů, v němž model postupně prochází sekvencí stavů $X = 1, 2, 2, 3, 4, 4, 5$ a generuje tak posloupnost vektorů $p_1, \ldots, p_5$. Počáteční a koncový stav zde negenerují žádný výstup. Takový model se používá zejména pro vhodnost rozšíření na rozpoznávání souvislé řeči.



Obrázek 2.2: Příklad HMM

Sdružená podmíněná pravděpodobnost, že pozorování $\mathcal{P}$ je generováno modelem M procházejícím sekvencí stavů X, se snadno spočítá jako součin pravděpodobností přechodů a výstupních pravděpodobností

$$P(\mathcal{P}, X \mid M) = a_{12}\, b_2(p_1)\, a_{22}\, b_2(p_2)\, a_{23}\, b_3(p_3)\, \ldots \tag{2.3}$$

V praxi bývá známa pouze posloupnost pozorování $\mathcal{P}$, kdežto posloupnost stavů X zůstává skryta. Odtud pochází označení „skryté“ Markovovy modely. Z tohoto důvodu nelze podmíněnou pravděpodobnost počítat přímo podle vzorce 2.3. Místo toho lze počítat $P(\mathcal{P}|M)$ jako součet pravděpodobností přes všechny možné sekvence stavů $X = x(1), x(2), \ldots, x(T)$. Častěji se však používá aproximace, při níž se bere v úvahu pouze jedna nejpravděpodobnější sekvence stavů, tj.

$$\hat{P}(\mathcal{P}|M) = \max_X \left\{ a_{x(0)x(1)} \prod_{t=1}^{T} b_{x(t)}(p_t)\, a_{x(t)x(t+1)} \right\},$$

kde $x(0)$ je počáteční stav a $x(T+1)$ je koncový stav modelu. K výpočtu této pravděpodobnosti se používá rekurzivní procedura využívající dynamického programování, tzv. **Viterbiho algoritmus**. Za předpokladu, že pro sadu modelů M_k odpovídajících slovům w_k platí $P(\mathcal{P}|M_k) = P(\mathcal{P}|w_k)\ \forall k$, je s použitím rovnice 2.2 vyřešena rovnice 2.1, tedy celá úloha rozpoznávání izolovaných slov. K tomu je však potřeba, aby pro všechny modely M_k byly známy všechny jejich parametry – pravděpodobnosti a_{ij} a $b_j(p_t)$, a ty v reálných situacích až na velmi speciální případy předem známy nejsou. Pro jejich odhad slouží reestimační proces (cyklicky opakovaný odhad) zvaný **Baumův-Welchův algoritmus**.

2.2.2 Viterbiho algoritmus

Tento algoritmus slouží k vyhledání maximálně pravděpodobné sekvence stavů modelu M, která generovala pozorování $\mathcal{P}$, a také k výpočtu její pravděpodobnosti. Označme $\phi_j(t)$ maximální pravděpodobnost cesty modelem, která vychází z počátečního stavu a končí ve stavu j v čase t a přitom generuje řečové vektory $p_1, \ldots, p_t$. Tuto pravděpodobnost lze efektivně spočítat díky rekurzivnímu vyjádření

$$\phi_j(t) = \max_i \{\phi_i(t-1)\, a_{ij}\}\, b_j(p_t), \tag{2.4}$$

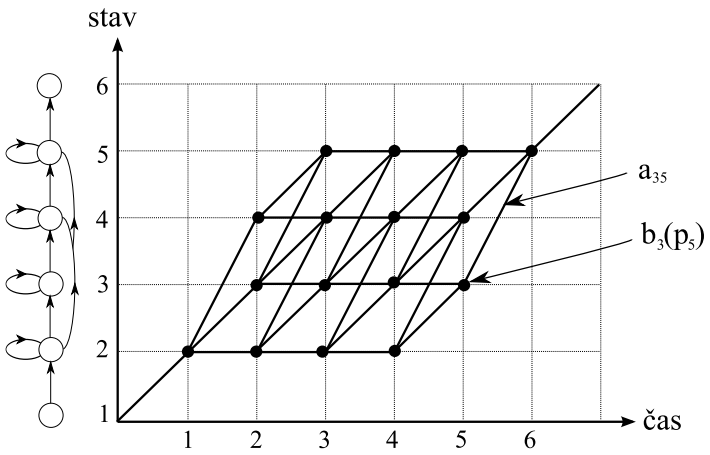
přičemž

$$\phi_1(1) = 1,\ \phi_j(2) = a_{1j}\, b_j(p_1)$$

pro $1 < i, j < N$. Pravděpodobnost $\hat{P}(\mathcal{P}|M)$ potom lze získat jako

$$\phi_N(T) = \max_i \{\phi_i(T)\, a_{iN}\}\,.$$

Hledání požadované sekvence stavů si lze představit jako hledání cesty v mřížce, která je znázorněna na obrázku 2.3. Na svislé ose jsou vyneseny jednotlivé stavy Markovova modelu a na vodorovné ose jsou časové úseky $1, \ldots, T$. Zvýrazněné body v mřížce naznačují generování výstupního symbolu, tučné čáry z bodu v čase t reprezentují všechny povolené přechody do stavů v čase $t+1$. Výpočet postupuje zleva doprava, přičemž v každém kroku jsou známy hodnoty $\phi_i(t-1)$ pro všechny stavy i a s použitím rovnice 2.4 se snadno spočítá $\phi_j(t)$. Algoritmus končí, laicky řečeno, v pravém horním rohu mřížky, kde je známa hodnota $\phi_N(T)$, a tedy je spočtena $\hat{P}(\mathcal{P}|M)$.



Obrázek 2.3: Schéma Viterbiho algoritmu

2.2.3 Baumův-Welchův algoritmus

Při trénování skrytého Markovova modelu je cílem nastavit jeho parametry tak, aby byla maximalizována pravděpodobnost generování pozorované posloupnosti $\mathcal{P}$ v trénovacích datech daným modelem. K tomuto účelu se používá Baumův-Welchův algoritmus, který představuje iterativní postup, jenž na základě trénovací množiny

a stávajícího modelu určí nový odhad parametrů. Nejdříve je tedy nutné parametry inicializovat nějakým prvotním odhadem, algoritmus pak zajistí dosažení alespoň lokálního maxima pravděpodobnosti $P(\mathcal{P}|M)$. Nejprve popíšeme způsob odvození parametrů modelu M, který odpovídá generování slova w, na základě jedné promluvy slova w – pozorování $\mathcal{P}$.

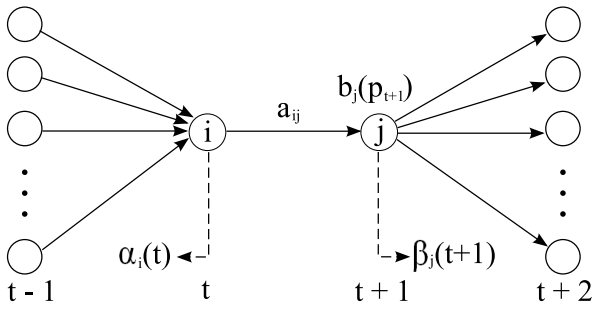
Využívá se tzv. *forward-backward algoritmus*, kdy se odpředu počítá pravděpodobnost $\alpha_i(t)$, definovaná jako pravděpodobnost generování částečné posloupnosti $p_1, \ldots, p_t$ a příchodu do stavu i při daném modelu M, a odzadu se počítá pravděpodobnost $\beta_i(t)$, definovaná jako pravděpodobnost generování částečné posloupnosti $p_{t+1}, \ldots, p_T$ ze stavu i při daném modelu M. Obě pravděpodobnosti lze definovat rekurzivně pomocí vztahů

$$\alpha_j(t+1) = \left(\sum_{i=1}^{N} \alpha_i(t)\, a_{ij}\right) b_j(p_{t+1}) \;\; \forall j, 1 < j \leq N, \quad \alpha_1(1) = 1$$

$$\beta_i(t) = \sum_{j=1}^{N} a_{ij}\, b_j(p_{t+1})\, \beta_j(t+1) \;\; \forall i, 1 \leq i < N, \quad \beta_N(T) = 1.$$

Označme $\varphi_{i,j}(t)$ pravděpodobnost, že model M je v čase t ve stavu i a v čase $t+1$ ve stavu j. Jak ilustruje obrázek 2.4, tuto pravděpodobnost lze vyjádřit vztahem

$$\varphi_{i,j}(t) = \frac{\alpha_i(t)\, a_{ij}\, b_j(p_{t+1})\, \beta_j(t+1)}{P(\mathcal{P}|M)} \quad \forall i, j \; 1 \leq i, j \leq N.$$



Obrázek 2.4: Schéma výpočtu $\varphi_{i,j}(t)$

Definujme dále proměnnou $\sigma_i(t)$ jako pravděpodobnost, že model M je v čase t ve stavu i. Platí

$$\sigma_i(t) = \sum_{j=1}^{N} \varphi_{i,j}(t).$$

Sečtením hodnoty $\sigma_i(t)$ pro všechny časové okamžiky (od $t = 1$ do $t = T - 1$) získáme hodnotu, kterou lze interpretovat jako očekávaný počet přechodů ze stavu i do dalšího stavu. Obdobná suma pro $\varphi_{i,j}(t)$ může být chápána jako očekávaný počet přechodů ze stavu i do stavu j. Z toho lze odvodit vztahy pro nové odhady parametrů

skrytého Markovova modelu. Odhad pravděpodobnosti a_{ij} lze vyjádřit jako poměr očekávaného počtu přechodů modelu ze stavu i do stavu j ku očekávanému počtu přechodů modelu ze stavu i do libovolného stavu, tedy

$$\hat{a}_{ij} = \frac{\sum_{t=1}^{T-1} \varphi_{i,j}(t)}{\sum_{t=1}^{T-1} \sigma_i(t)}.$$

Odhad pravděpodobnosti generování symbolu s stavem j $b_j(s)$ lze vyjádřit jako poměr očekávaného počtu případů, kdy je model ve stavu j a zároveň pozorovaný výstupní symbol $p_t = s$, ku očekávanému počtu případů, kdy je model ve stavu j, tedy

$$\hat{b}_j(s) = \frac{\sum_{t=1,p_t=s}^{T} \sigma_j(t)}{\sum_{t=1}^{T} \sigma_j(t)}.$$

Ve všech rovnicích $1 \leq i, j \leq N$.

Je dokázáno, že pro každý nový odhad parametrů tvořící model $\hat{M}$ vždy platí $P(\mathcal{P}|\hat{M}) > P(\mathcal{P}|M)$, s výjimkou stavu, kdy bylo dosaženo optimálního nastavení parametrů. Je tedy zaručeno, že algoritmus konverguje k (lokálnímu) maximu funkce $P(\mathcal{P}|M)$. V praxi se obvykle iterativní proces ukončuje ve chvíli, kdy rozdíl mezi novými parametry a parametry z předcházející iterace nepřesáhne určitou předem stanovenou malou hodnotu.

Pro kvalitní natrénování skrytého Markovova modelu potřebujeme obvykle několik různých promluv každého slova. To je důležité zejména při trénování parametrů modelu na hlasy více řečníků. Máme-li tedy pro určité slovo k dispozici soubor $s = 1, \ldots, S$ promluv, odpovídá těmto promluvám soubor S posloupností pozorování $\tilde{\mathcal{P}} = \left\{\mathcal{P}^{(1)}, \mathcal{P}^{(2)}, \ldots, \mathcal{P}^{(S)}\right\}$. Přitom platí, že $\mathcal{P}^{(s)} = \left\{p_1^{(s)}, p_2^{(s)}, \ldots, p_{T_s}^{(s)}\right\}$. Úkolem je nalézt takové parametry modelu M, aby byla maximalizována pravděpodobnost

$$P(\tilde{\mathcal{P}}|M) = P(\mathcal{P}^{(1)}, \mathcal{P}^{(2)}, \ldots, \mathcal{P}^{(S)}|M).$$

Považujeme-li jednotlivé pozorované posloupnosti za na sobě nezávislé, lze tuto pravděpodobnost přepsat do tvaru

$$P(\tilde{\mathcal{P}}|M) = \prod_{s=1}^{S} P(\mathcal{P}^{(s)}|M).$$

V procesu trénovaní pomocí Baumova-Welchova algoritmu nelze předložit rozpoznávači najednou celý spojený trénovací soubor. Naopak je třeba předkládat zvlášť každou posloupnost $\mathcal{P}^{(s)}$ a každá promluva reprezentovaná posloupností $\mathcal{P}^{(s)}$ musí začínat v čase $t = 1$ a stavu s číslem 1 a končit v čase $t = T_s$ ve stavu s číslem N.

Podrobnější popis Baumova-Welchova a Viterbiho algoritmu a odvození vztahů uvádí Psutka [5, str. 162–172]. Další informace lze rovněž nalézt v příručce The HTK Book [12], která byla napsána pro nástroj HTK – *Hidden Markov Model Toolkit*. Jedná se o nástroj sloužící k vytváření a zpracování skrytých Markovových modelů, který je primárně využíván při výzkumu automatického rozpoznávání řeči.

2.3 Rozpoznávání souvislé řeči

Aby byla člověku umožněna plynulá komunikace s počítačem mluvenou řečí, je třeba vyřešit několik dalších problémů, kterými se rozpoznávání izolovaných slov nezabývá. Patří sem např. otázka počtu slov v promluvě, určování hranic jednotlivých slov a problematika trénování a rozpoznávání na úrovni posloupnosti slov.

2.3.1 Markovovy modely fonémů

I při rozpoznávání souvislé řeči se uplatňují skryté Markovovy modely, přičemž jejich rozšíření na rozpoznávání souvislé řeči spočívá ve spojování více modelů za sebou. Každý model v posloupnosti pak odpovídá nějakému symbolu řeči. Těmito symboly obvykle bývají jednotky kratší než slova, zpravidla fonémy nebo některé jednotky od nich odvozené, jak je uvedeno dále. Motivací k tomuto kroku je snaha omezit zdlouhavé trénování při práci s rozměrnějšími slovníky. Počet různých fonémů, které využívá nějaký jazyk, je nesrovnatelně menší než počet různých slov, obzvláště v případě flektivních jazyků, jakým je např. čeština. Trénování potom spočívá ve hledání parametrů několika desítek modelů fonému namísto mnoha tisíců modelů slov.

Foném je fonetická jednotka, kterou lze popsat jako nejmenší součást zvukové stránky řeči, která má rozlišovací funkci. Záměna fonému vede ke změně významu slova, např. jako u dvojice slov **n**ovinka – **r**ovinka. Existují algoritmy, které automaticky provádějí tzv. *fonetickou transkripci*, tj. převádějí slova psaná obvyklým způsobem do fonetické abecedy, jejíž nejobecnější verzí je mezinárodní fonetická abeceda – International Phonetic Alphabet (IPA). Při fonetické transkripci se provádí např. přepis „oběd“ → [objet], „nikdy“ → [ňigdi].

Nevýhodou fonému je, že neobsahuje důležitou koartikulační informaci, tj. informaci o tom, jak se mění vyslovení dané jednotky v závislosti na jednotkách předcházejících a následujících. Lépe je na tom v tomto ohledu jednotka označovaná jako *difón*, jenž představuje segment začínající ve středu jedné hlásky a končící ve středu hlásky následující. Pro modelování mluvené řeči se používá ještě o něco obsáhlejší *trifón*, což je segment, který začíná stejně jako difón, ale pokračuje přes celou následující hlásku a končí ve středu třetí hlásky. Trifón tak modeluje každý foném v kontextu jeho levého a pravého souseda.

Rozpoznávač řeči s Markovovými modely trifónů uchovává jednak parametry jednotlivých modelů trifónů a jednak slovník, v němž však na rozdíl od rozpoznávače izolovaných slov ukládá slova klasickým způsobem jako posloupnosti písmen. Modely jednotlivých slov si pak vytváří automaticky tak, že aplikací fonetické transkripce získá základní fonetický tvar daného slova a podle něj utvoří zřetězení odpovídajících modelů trifónů. Při trénování a posléze rozpoznávání posloupností slov jsou tyto zřetězené modely dále propojovány v modely celých posloupností slov.

Markovovy modely fonémů a potažmo trifónů se obvykle liší od výše uvedených Markovových modelů slov tím způsobem, že namísto generování symbolu s ve stavu j s pravděpodobností $b_j(s)$ je zde generování symbolu spojeno s přechodem ze stavu i do stavu j a jeho pravděpodobnost se označuje $b_{ij}(s)$. Vedle přechodů mezi

stavy, které generují určitý výstup, jsou modely fonémů často vybaveny i přechody, které neprodukují žádný výstup. Tyto přechody bývají označovány jako přechody s nulovým výstupem. Zřetězení více po sobě jdoucích modelů trifónů lze jednoduše realizovat tak, že se počáteční stav Markovova modelu ztotožní s koncovým stavem bezprostředně předcházejícího modelu.

2.3.2 Struktura rozpoznávače

Nechť $W = \{w_1, w_2, \ldots, w_N\}$ je posloupnost N slov a nechť $\mathcal{P} = \{p_1, p_2, \ldots, p_T\}$ je akustická informace odvozená z řečového signálu (posloupnost výstupních symbolů Markovových modelů). Cílem rozpoznávače je nalézt posloupnost slov W', která maximalizuje podmíněnou pravděpodobnost $P(W'|\mathcal{P})$. Pomocí Bayesova pravidla se stejným způsobem jako v kapitole 2.2.1 o rozpoznávání izolovaných slov odvodí vztah analogický rovnici 2.2

$$P(W'|\mathcal{P}) = \max_W \frac{P(\mathcal{P}|W)P(W)}{P(\mathcal{P})} = \max_W P(\mathcal{P}|W)P(W). \tag{2.5}$$

Zde $P(\mathcal{P}|W)$ je pravděpodobnost, že při vyslovení posloupnosti slov W bude produkována posloupnost výstupních symbolů $\mathcal{P}$, $P(W)$ je pravděpodobnost posloupnosti slov W v daném jazyce a $P(\mathcal{P})$ je pravděpodobnost posloupnosti výstupních symbolů. $P(\mathcal{P})$ lze opět při hledání maxima ignorovat, neboť se nejedná o funkci proměnné W.

Úloha rozpoznávání může nyní být rozdělena na následující kroky:

1. akustické zpracování řečového signálu,
2. vytvoření **akustického modelu** nebo též modelu řečníka pro vypočtení podmíněné pravděpodobnosti $P(\mathcal{P}|W)$; zde se uplatní zřetězené skryté Markovovy modely trifónů, případně fonémů,
3. vytvoření **jazykového modelu** pro vypočtení pravděpodobnosti $P(W)$,
4. nalezení nejpravděpodobnější posloupnosti slov použitím prohledávací strategie.

Jazykový model

Pravděpodobnost posloupnosti $W = \{w_1, w_2, \ldots, w_N\}$, která je doménou jazykového modelu, může být přepsána na součin podmíněných pravděpodobností

$$P(W) = P(w_1)P(w_2|w_1)P(w_3|w_1w_2)\ldots P(w_N|w_1\ldots w_{N-1}) = \prod_{i=1}^{N} P(w_i|w_1\ldots w_{i-1}).$$

Pravděpodobnosti $P(w_i|w_1\ldots w_{i-1})$ je však prakticky nemožné vyčíslit jednak z důvodu vysoké výpočetní náročnosti, jednak proto, že v trénovacích datech konečné velikosti se nemohou nikdy vyskytovat všechny různě dlouhé posloupnosti slov v dostatečném počtu potřebném pro statistické výpočty.

Přijatelnou aproximaci lze získat tak, že všechny historie končící na stejná dvě slova $w_{i-2}\,w_{i-1}$ jsou považovány za ekvivalentní, tj.

$$P(w_i|\,w_1 \ldots w_{i-1}) \approx P(w_i|\,w_{i-2}\,w_{i-1}),$$
$$P(W) \approx P(w_1)\,P(w_2|\,w_1) \prod_{i=3}^{N} P(w_i|\,w_{i-2}\,w_{i-1}).$$

Tento model jazyka, který je založen na pravděpodobnostech výskytu trojic po sobě jdoucích slov – trigramů, se nazývá **trigramový model**. Obdobně pro n-tice slov lze obecně vytvořit n-gramové modely. Jako odhad pravděpodobností n-tic se obvykle používá poměr četnosti výskytu příslušné n-tice v trénovacích datech ku počtu všech n-tic v datech (jedná se o aplikaci metody *MLE – maximum likelihood estimation*). Trénovací text by přitom měl být vybírán ze stejné problémové oblasti, ve které bude pracovat navrhovaný rozpoznávač.

Ani tím však není sestavení modelu úplně vyřešeno. Uvážíme-li slovník o velikosti např. 10 000 slov, počet všech různých trigramů bude 10^{12}. Je zřejmé, že většina z tohoto množství se vůbec neobjeví v jakémkoli běžném trénovacím textu. Problém nastane v okamžiku, kdy se v testovacím textu objeví trigram, který nebyl zastoupen v textu trénovacím. Pokud by pravděpodobnosti trigramů byly odvozeny pouze z četností a ani při ohodnocování testovacích dat by výskyt nových trigramů nebyl nijak ošetřen, byla by těmto trigramům přiřazena nulová pravděpodobnost, což je samozřejmě pro funkci rozpoznávání nežádoucí. Tento problém se nazývá problém nedostatečných dat (*sparse data*) a k jeho eliminaci lze přistupovat různými způsoby. Pravděpodobně nejrozšířenějším způsobem jsou metody, které se souhrnně označují jako vyhlazování (*smoothing*).

Základní přístup k vyhlazování je založen na počítání váženého průměru relativních četností trigramů, bigramů a individuálních četností slov. Odhad pravděpodobností trigramů je prováděn podle vztahu

$$P_s(w_i|\,w_{i-2}\,w_{i-1}) = \lambda_3 \frac{c(w_{i-2}\,w_{i-1}\,w_i)}{c(w_{i-2}w_{i-1})} + \lambda_2 \frac{c(w_{i-1}\,w_i)}{c(w_{i-1})} + \lambda_1 \frac{c(w_i)}{|T|} + \lambda_0 \frac{1}{|V|},$$

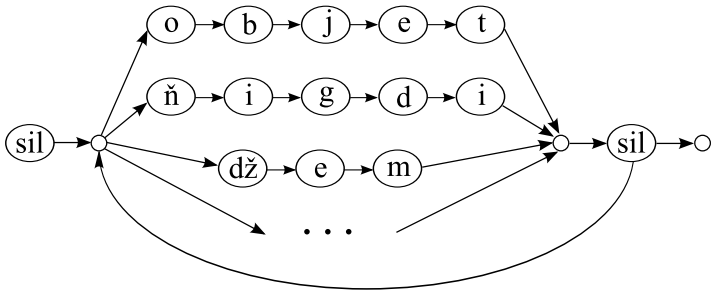
kde $c(w_{i-n} \ldots w_i)$ je četnost n-gramu $w_{i-n} \ldots w_i$, $|T|$ je počet všech slov v trénovacích datech a $|V|$ je počet všech různých slov v datech neboli velikost slovníku. Pro váhové koeficienty $\lambda_0, \ldots, \lambda_3$ platí

$$\lambda_0 + \lambda_1 + \lambda_2 + \lambda_3 = 1.$$

Pro spočtení jejich hodnot je potřeba oddělená část trénovacích dat, která nebyla použita při počítání četností n-gramů. Pro tato data se nejvíce vžil anglický termín *heldout data*. Hodnoty váhových koeficientů jsou určovány tak, aby výsledný vyhlazený model maximalizoval pravděpodobnost heldout dat. U jazykového modelu tohoto tvaru je pak zajištěno, že pravděpodobnost trigramu v testovacích datech nebude nikdy nulová. Pokaždé, když je četnost n-gramu vyššího řádu v trénovacích datech nulová, může být nenulová četnost n-gramu nižšího řádu. Pokud by se ani samostatné slovo (unigram) v trénovacích datech nevyskytovalo, bude výsledkem výpočtu pravděpodobnosti hodnota $\lambda_0 \frac{1}{|V|}$.

Dekódování

Ve chvíli, kdy máme definován akustický i jazykový model, je pro vyřešení rovnice 2.5 ještě nutné navrhnout jejich spojení a způsob vyhledání nejpravděpodobnější posloupnosti $W' = \{w_1, w_2, \ldots, w_N\}$. Obvykle se k tomuto problému přistupuje tak, že se sestaví síť všech slov ve slovníku, jejíž struktura může vypadat jako na obrázku 2.5. Stavy této sítě jsou ve skutečnosti skryté Markovovy modely, stav s označením „sil" je skrytý Markovův model pro ticho. Pravděpodobnosti přechodů uvnitř Markovových modelů jsou získány trénováním např. pomocí Baumova-Welchova algoritmu, přechody mezi modely v rámci slova jsou obvykle ohodnoceny konstantní pravděpodobností. Slova jsou spojena do smyčky a v rozpoznávané řeči mohou následovat libovolně za sebou. Přechody mezi slovy mohou být modifikovány pro doplnění pravděpodobností na základě jazykového modelu, pro složitější modely je možné síť slov uložit v podobě specifičtější gramatiky.



Obrázek 2.5: Síť slov

Pro nalezení výsledné posloupnosti lze opět použít Viterbiho algoritmus. Za účelem omezení příliš velkého počtu cest, které jsou brány v úvahu pro jeden časový okamžik t, se aplikuje prořezávání neboli *pruning*. Nejčastěji používaným kritériem prořezávání je požadovaný počet cest, které se mají udržovat, přičemž se vyberou cesty s doposud nejvyšší pravděpodobností a všechny ostatní jsou ignorovány. Dalšími možnostmi jsou minimální požadovaná pravděpodobnost cesty nebo hodnota udávající maximální výši součtu pravděpodobností cest (vyberou se opět cesty s nejvyšší pravděpodobností). Tato operace s sebou nese určité riziko, že optimální cesta bude při prořezávání „zahozena", avšak pro dosažení potřebné rychlosti je obvykle nezbytná.

Výstupem rozpoznávače může být jedna jediná posloupnost slov odpovídající nejlepší cestě, nebo je na výstup vydáno několik různých cest s nejvyššími pravděpodobnostmi (několik hypotéz), a to buď v podobě grafu slov – tzv. *word graph*, kdy jednotlivé hypotézy mohou sdílet části cest, nebo jako seznam samostatných cest – tzv. *N-best list*.

Kapitola 3

Metody detekce chyb

3.1 Chyby rozpoznávání

Podle Voll [10] lze identifikovat několik fází zpracování chyb v rozpoznávání řeči:

1. prevence chyb
2. předvídání chyb – zjišťování pravděpodobností chyb na základě známých nedostatků systému
3. detekce chyb – identifikace chyb, které již nastaly
4. odstranění chyb
   I. zjištění příčiny chyby
   II. zvolení a implementace nějaké strategie opravy chyby
5. udržování zpětné vazby o tom, jak úspěšná je detekce a oprava chyb

Při rozpoznávání řeči vznikají chyby vážnější i méně vážné. Menší chyby, jako například vynechání členu, obvykle mění význam (sémantiku) sdělení málo nebo vůbec a nehrozí, že by obsah byl nesprávně interpretován. Jiné chyby však mohou způsobit změnu významu, přičemž nebezpečnější jsou ty z nich, jež nejsou čtenářem na první pohled odhalitelné. Může se jednat o záměnu dvou odborných termínů, vynechání záporu ve větě apod. Tyto chyby přitom nemusí narušit syntaktickou správnost věty, ale změní její sémantiku. Chyby můžeme rozdělit podle typů záměny oproti správnému přepisu na šest druhů:

Chyby nevýznamových slov – chybně rozpoznaná slova, která nejsou nositeli významu, tedy předložky, členy atd.

Spojení slov – dvě nebo více slov chybně rozpoznáno jako jedno slovo, např. „pod nos“ → „podnos“

Rozdělení slov – jedno vyslovené slovo chybně rozpoznáno jako více slov, např. „podnos“ → „pod nos“

Záměna slova – nahrazení jednoho slova jiným slovem

Vložení slova – přidání slova, které není součástí promluvy

Vynechání slova – slovo, které patří do vyslovené promluvy, chybí ve výstupu rozpoznávače.

Přesnost rozpoznávání lze vyjádřit pomocí míry zvané *word error rate – WER*, která udává podíl počtu chybných slov ku počtu všech slov v mluvených datech.

3.2 Skóre spolehlivosti

Jednotlivé části rozpoznaného textu (zpravidla slova) lze ohodnocovat pomocí skóre spolehlivosti (*confidence score*). Tato hodnota vyjadřuje pravděpodobnost, že daná část textu je rozpoznána správně.

Skóre spolehlivosti je obecně výsledkem kombinace několika měr spolehlivosti (*confidence measure*). Tyto míry spolehlivosti jsou zpravidla odvozeny od statistických vlastností akustického a jazykového modelu na úrovni fonémů, slov nebo celé promluvy. Jsou vybírány takové charakteristiky, které jsou statisticky v nějakém vztahu ke správnosti hypotézy rozpoznávání, ať už se jedná o závislost přímou (vyšší hodnota charakteristiky naznačuje vyšší šanci na správnost) či nepřímou (vyšší hodnota souvisí s méně pravděpodobnou správností).

Skóre spolehlivosti je rozpoznanému textu přiřazováno buď již při procesu rozpoznávání, nebo je později doplněno samostatným algoritmem detekce chyb. Jak již bylo dříve zmíněno, výstupem rozpoznávače nemusí být jediná posloupnost slov, ale může to být více hypotéz (*N-best list*). Kritériem pro výběr N nejlepších hypotéz může být právě skóre spolehlivosti. Seznam N hypotéz s nejvyšším skóre pak může být vstupem pro další algoritmus detekce chyb, který vytvoří vlastní seznam nejlepších hypotéz. Metody detekce lze klasifikovat jako metody pravděpodobnostní nebo jako metody, které nejsou založeny na pravděpodobnostních principech, případně metody smíšené, které si kladou za cíl využít lepší vlastnosti od obou typů metod a potlačit jejich nedostatky.

3.3 Pravděpodobnostní metody

Jistě nejjednodušší pravděpodobnostní mírou spolehlivosti je dlouhodobý průměr úspěšnosti rozpoznávače, tj. u rozpoznávače, pro nějž byla z dosavadního rozpoznávání vypočtena úspěšnost 80 %, lze očekávat správné rozpoznání každé nové promluvy s pravděpodobností $0,8$. Tento naivní přístup pochopitelně nepřináší příliš informace pro vylepšení rozpoznávače a je třeba představit metody sofistikovanější.

3.3.1 Aposteriorní pravděpodobnost

Wessel a kol. [11] poukazují na skutečnost, že zanedbání pravděpodobnosti akustického pozorování $P(\mathcal{P})$ ve výpočtu rovnice 2.5 (str. 16) znemožňuje přímo v průběhu

rozpoznávání vypočítat pravděpodobnost správnosti slova, která by mohla být využita jako skóre spolehlivosti. Navrhují však způsob, jak spočítat aposteriorní[1] pravděpodobnost slova v případě, že výstup rozpoznávače je uložen v podobě grafu slov (*word graph*) nebo seznamu N nejlepších vět (*N-best list*).

Pravděpodobnost v grafu slov

Grafem slov se rozumí orientovaný acyklický graf s váženými hranami, jehož vrcholy reprezentují diskrétní časové okamžiky a hrany představují hypotézy pro slova trvající od jednoho časového okamžiku – počátečního vrcholu – do druhého časového okamžiku – koncového vrcholu hrany. Váhy na hranách mají hodnotu akustické pravděpodobnosti hypotézy. Libovolná cesta v grafu od vrcholu prvního časového okamžiku k vrcholu posledního časového okamžiku vytváří alternativní hypotézu vyslovené věty.

Předpokládá se, že jazykový model má podobu n-gramů jako v kapitole 2.3.2. Aposteriorní pravděpodobnost každé hypotézy slova (hrany) lze spočítat na principu forward-backward algoritmu tak, že se sčítá součin dopředné (forward) a zpětné (backward) pravděpodobnosti přes všechny možné cesty, které vedou od počátku grafu po danou hranu (případ forward) a od dané hrany k poslednímu vrcholu grafu (případ backward).

Takto spočtená pravděpodobnost ještě nemá velkou úspěšnost jakožto skóre spolehlivosti, neboť jedno slovo se v grafu slov obvykle objevuje vícekrát s nepatrně odlišnými počátečními a koncovými časy a pravděpodobnost slova je rozdělena mezi tyto instance (problém je označován jako segmentace grafu slov). Z tohoto důvodu navrhli Wessel a kol. [11] několik způsobů, jak zkombinovat aposteriorní pravděpodobnosti více hran reprezentujících totéž slovo. Nejlepších výsledků dosáhla metoda, která aposteriorní pravděpodobnost hypotézy, že slovo w začíná v čase t_0 a končí v čase t_1, počítá jako maximum z aposteriorních pravděpodobností všech hypotéz, které patří danému slovu w a jejich časový interval má s intervalem $\langle t_0, t_1 \rangle$ neprázdný průnik.

Pravděpodobnost v seznamu nejlepších vět

Seznam N nejlepších vět může mít různé podoby v různých aplikacích. Pro účely počítání aposteriorní pravděpodobnosti je definován jako seznam N různých posloupností slov $v_1^{M_1}, \ldots, v_1^{M_N}$, kde $v_1^{M_i} = v_1, \ldots, v_{M_i}$, přičemž u každého slova je znám pouze údaj o pozici v dané posloupnosti a nikoli informace o časovém intervalu. Mezi těmito posloupnostmi však nelze přímo porovnávat slova na stejné pozici. Posloupnosti mohou být různě dlouhé a mohou obsahovat chybně vložená nebo vynechaná slova. Posloupnosti se proto zarovnávají pomocí technik dynamického programování. Pro toto zarovnání musí platit, že pro každé slovo w_m na pozici

[1] Pojem *aposteriorní* pravděpodobnost se používá společně s výrazem *apriorní* pravděpodobnost; apriorní pravděpodobnost znamená pravděpodobnost jevu jako takového, např. pravděpodobnost výskytu slova $P(w)$, zatímco aposteriorní pravděpodobnost je pravděpodobnost podmíněná, např. pravděpodobnost správnosti slova za předpokladu pozorování $\mathcal{P}$ $P(w|\mathcal{P})$.

m v referenční větě w_1^M je možné zjistit odpovídající slovo v v libovolné další větě z $v_1^{M_1}, \ldots, v_1^{M_N}$, značeno

$$v = \mathcal{L}_m \left(w_1^M, v_1^{M_n} \right).$$

Aposteriorní pravděpodobnost pro každé slovo w_m v referenční větě w_1^M lze pak spočítat jako

$$P\left(w_m | x_1^T, w_1^M, \mathcal{L}\right) = \frac{\sum_{n=1}^{N} P\left(x_1^T | v_1^{M_n}\right)^{\alpha} P\left(v_1^{M_n}\right)^{\beta} \delta\left(w_m, \mathcal{L}_m\left(w_1^M, v_1^{M_n}\right)\right)}{\sum_{n=1}^{N} P\left(x_1^T | v_1^{M_n}\right)^{\alpha} P\left(v_1^{M_n}\right)^{\beta}},$$

kde x_1^T je posloupnost T akustických pozorování, α a β jsou normalizační faktory akustického a jazykového modelu. Kroneckerova funkce δ vrací 1, pokud jsou oba argumenty identické, a 0, pokud jsou různé. K navýšení aposteriorní pravděpodobnosti slova w_m tedy přispívají pouze ty z N posloupností, v nichž po zarovnání s posloupností w_1^M tomuto slovu odpovídá slovo identické.

3.3.2 Indexování latentní sémantiky

Metoda indexování latentní (skryté) sémantiky (*latent semantic indexing – LSI*) sleduje společné výskyty termínů ve smyslu výskytu jednoho termínu v kontextu druhého termínu a podle toho odvozuje, zda je mezi těmito termíny nějaký vztah.

Tato metoda původně vznikla v oblasti indexování dokumentů a vyhledávání informací. Je založena na základní úvaze, že všechny sémanticky související dokumenty nemusejí obsahovat konkrétní hledaná slova a fráze kvůli používání synonymních výrazů.

Voll [10, str. 40] uvádí příklad v tabulce 3.1 (původně publikovaný v Manning, Schütze: *Foundations of Statistical Natural Language Processing*, 2002). Do databáze dokumentů je vložen dotaz na dokumenty obsahující klíčová slova „uživatel“ a „rozhraní“. Pokud se omezíme na vyhledání těchto slov, bude nalezen pouze dokument 1. Jelikož se však termíny „HCI“ a „interakce“ společně vyskytují jak v dokumentu 1, tak v dokumentu 2, je pravděpodobné, že dokument 2 také souvisí se zpracovávaným dotazem. Metoda LSI definuje měřítko pro sémantickou souvislost dvou termínů a potažmo dvou dokumentů na základě obsažených termínů. Umožňuje tedy exaktně vyjádřit podobnost mezi dokumentem 1 a dokumentem 2 a po vložení dotazu vrátit nejen dokument 1, který obsahuje klíčová slova, ale i další dokumenty, které mají s prvním dokumentem blízkou sémantickou souvislost.

	Termín 1	Termín 2	Termín 3	Termín 4
Dotaz	uživatel	rozhraní		
Dokument 1	uživatel	rozhraní	HCI	interakce
Dokument 2			HCI	interakce

Tabulka 3.1: Příklad využití společného výskytu termínů

K aplikaci pojmu sémantické souvislosti při detekci chyb vede úvaha, že chybně rozpoznaná slova mají nízkou sémantickou souvislost s okolními slovy ve svém kontextu. Sémantická míra souvislosti může doplnit pravděpodobnost akustického a jazykového modelu ve fázi dekódování a vylepšit tak výběr N nejlepších posloupností na výstupu rozpoznávače.

3.3.3 Vzájemná informace

Vzájemná informace (*pointwise mutual information*) $I(x, y)$ je statistická veličina, která měří stupeň závislosti mezi dvěma náhodnými jevy x a y. Je definována vztahem

$$I(x, y) = \log \frac{P(x, y)}{P(x)P(y)},$$

kde $P(x, y)$ je sdružená pravděpodobnost, která charakterizuje společný výskyt obou jevů x a y, a $P(x)$, $P(y)$ jsou pravděpodobnosti výskytu každého z jevů bez ohledu na jevy ostatní. Za jevy x a y můžeme dosadit přítomnost slov w_x, w_y v textu, jejich pravděpodobnost pak odhadnout pomocí relativní četnosti v textovém korpusu. Vyšší hodnota vzájemné informace poukazuje na větší tendenci těchto slov vyskytovat se společně, např. v ustálených slovních spojeních (typickým příkladem je výraz „křížem krážem“, kde slovo „krážem“ bude stěží mít vyšší četnost než toto slovní spojení).

Nevýhodou vzájemné informace je skutečnost, že je citlivá na problém nedostatečných dat. Pokud se dvě slova v textu vyskytují pouze společně, tedy $P(x, y) = P(x) = P(y)$, pak $I(x, y) = \log \frac{1}{P(x,y)}$. To znamená, že čím větší je počet výskytů dvojice slov, tím nižší hodnota vzájemné informace vychází, přestože jejich tendence sdružování je evidentně větší.

Pro účely detekce chyb lze míru závislosti použít po vyslovení úvahy, že slova, která vykazují nízkou hodnotu závislosti na okolních slovech – kontextu, jsou kandidáty na chyby rozpoznávání. Závislost na kontextu lze vypočítat např. tak, že se (po přesnější definici kontextu) postupně spočte hodnota vzájemné informace daného slova a všech slov v kontextu a tyto hodnoty se zkombinují do jednoho čísla. Toto číslo vyjadřuje jakousi „sémantickou koherenci“ tohoto slova a může být použito jako míra spolehlivosti rozpoznávání.

3.3.4 Detekce založená na kontextu

Sarma a Palmer [8] představili způsob detekce a opravy chyb založený na analýze kontextu slova (*context-based error detection*). Tato metoda byla primárně navržena pro úlohu vyhledávání v kolekci dokumentů, které byly z mluvené řeči automaticky přepsány systémem rozpoznávání řeči. Pokud uživatel chce nalézt např. dokumenty s výskytem slova „autorizace“ a toto slovo bylo při přepisu často chybně rozpoznáno, bude to mít negativní vliv na kvalitu výsledků vyhledávání. Metoda detekce popsaná v [8] má pomoci nalézt i ty dokumenty, v nichž bylo hledané slovo vysloveno, ale přepsáno chybně.

Metoda využívá dvou vlastností chyb při rozpoznávání řeči. Za prvé tvrdí, že specifická slova ve velkém korpusu mají tendenci objevovat se častěji v kontextu s určitými dalšími slovy a že chybné přepisy těchto specifických slov se budou také objevovat se stejnými slovy v kontextu. Druhou vlastností chybně přepsaných slov je obvykle jejich fonetická podobnost se slovy, která byla opravdu vyslovena. Pokud by rozpoznávač chybně přepsal slovo „autorizace“ např. jako podobně znějící slovo „atomizace“, objeví se zápis tohoto slova v kontextu typickém pro slovo „autorizace“, který se pravděpodobně bude lišit od kontextu typického pro slovo „atomizace“.

Aby bylo možno metodu použít k detekci chyb, je nejprve potřeba natrénovat parametry na trénovacích datech, která jsou tvořena dokumenty přepsanými automatickým rozpoznávačem včetně všech případných chyb. Pro každé slovo w ze slovníku rozpoznávače se provede analýza společného výskytu s dalšími slovy v kontextu. Za tímto účelem se stanoví velikost okna o (empiricky optimální pro konkrétní úlohu) a pro všechna slova w', která se v dokumentech objevila v rozmezí $\frac{o}{2}$ slov před slovem w nebo $\frac{o}{2}$ slov za slovem w se spočte vzájemná informace $I(w, w')$. Poté se pro každé slovo w sestaví typický kontext – seznam n slov s nejvyšší hodnotou vzájemné informace.

Druhou fází detekce je samotné vyhledávání dotazovaného slova w_d v nových (testovacích) datech. Algoritmus se snaží identifikovat části textu, které obsahují slova typická pro kontext vyhledávaného slova a budou tedy s vysokou pravděpodobností obsahovat toto slovo nebo chybný přepis tohoto slova. Vyhledávání těchto částí textu probíhá tak, že se okno velikosti o (stejné jako při trénování) postupně posunuje od začátku textu do konce a v něm se hledají výskyty slov z typického kontextu slova w_d (vytvořeného při trénování). Pokud se najde alespoň c slov z kontextu slova w_d, je slovo uprostřed okna označeno jako potenciální chybný přepis slova w_d (pokud jím není právě slovo w_d). Konstantu c je opět nutné vhodně zvolit.

Tímto způsobem vzniká velké množství kandidátů na chybný přepis slova w_d, je proto třeba následně ještě zpřesnit jejich výběr. První krok zpřesňování se provádí na základě Kullbackovy-Leiblerovy divergence $D(p \parallel q)$ definované jako

$$D(p \parallel q) = \sum_{x \in X} p(x) \log \frac{p(x)}{q(x)},$$

kde za $p(x)$ se dosadí podmíněná pravděpodobnost výskytu slova w_d společně s kontextovým slovem x z množiny c kontextových slov X a $q(x)$ bude podmíněná pravděpodobnost výskytu slova, které je kandidátem na chybný přepis, s kontextovým slovem x. Vyšší hodnota divergence nasvědčuje tomu, že kandidát na chybný přepis je opravdu chybným přepisem slova w_d.

Další zúžení výběru kandidátů se provede pomocí fonetické podobnosti kandidátů s hledaným slovem w_d. Sarma a Palmer [8] použili techniku váženého fonetického zarovnávání s minimální vzdáleností. Vysoká míra podobnosti vede k závěru, že kandidát na chybný přepis má být opraven na slovo w_d.

3.4 Metody nevyužívající pravděpodobnost

3.4.1 Akustická stabilita

K použití akustické stability jako skóre spolehlivosti vede následující idea: šance, že slovo je rozpoznáno správně, je tím větší, čím vícekrát se objevuje na stejné pozici v seznamu vět, které byly získány různým nastavením váhy mezi akustickým a jazykovým modelem při dekódování. Toho může být dosaženo např. postupným nastavováním různých hodnot normalizačního faktoru jazykového modelu β. Jako referenční posloupnost w_1^M se stanoví výstup rozpoznávače pro standardní hodnotu faktoru (jedna nejlepší posloupnost) a dále pro N různých hodnot faktoru vzniká seznam N posloupností $v_1^{M_1}, \ldots, v_1^{M_N}$. Tyto posloupnosti pak lze k referenční posloupnosti zarovnat stejným způsobem jako při výpočtu aposteriorní pravděpodobnosti na seznamu N nejlepších vět v části 3.3.1. Akustická stabilita slova z referenční posloupnosti se pak spočte jako relativní četnost výskytu tohoto slova v ostatních posloupnostech na stejné pozici neboli

$$acu(w_n) = \frac{1}{N} \sum_{n=1}^{N} \delta\left(w_n, \mathcal{L}_n\left(w_1^M, v_1^{M_n}\right)\right).$$

3.4.2 Hustota hypotéz

Dalším kritériem navrženým pro měření spolehlivosti je hustota hypotéz. Vychází z předpokladu, že pokud jedna hypotéza svou pravděpodobností významně převyšuje hypotézy ostatní, bude to opravdu správný přepis promluvy. Většina ostatních hypotéz s nízkou pravděpodobností bude odstraněna prořezáváním. Pokud však neexistuje taková výrazně nejlepší hypotéza a mnoho hypotéz má podobné hodnoty pravděpodobnosti, šance na správný výběr přepisu se zmenšuje. Hustota hypotéz v časovém okamžiku se určí jako počet všech různých slov v grafu slov, jejichž hypotézy překrývají tento časový okamžik. Vyšší hodnota hustoty pak poukazuje na nižší míru spolehlivosti.

3.4.3 Analýza rysů na vyšší úrovni

Vedle lexikálních a statistických informací mohou k vylepšení skóre spolehlivosti přispět informace z vyšších úrovní jazyka, jako jsou kupříkladu **prozodické rysy**. Prozódie v lingvistice popisuje zvukové vlastnosti jazyka, které se uplatňují na úrovni vyšší, než je úroveň fonémů. Souhrnně se hovoří o tzv. suprasegmentálních jevech, kterými jsou slabika, přízvuk, tón, intonace (melodie), rytmus aj.

Prozódie promluvy může být ovlivněna mnoha faktory, mimo jiné pohlavím, věkem, postavením nebo hlasovými indispozicemi mluvčího. Když např. v dialogových systémech lidé opakují svoji promluvu, často slova příliš zdůrazňují – mění se jejich prozódie – a to zhorší úspěšnost rozpoznávání. Na základě prozodických rysů, jako je doba trvání a tempo promluvy, vytvořili Diane J. Litman a kol. pomocí algoritmů

strojového učení sadu pravidel, která klasifikují výsledky rozpoznávání jako správné nebo chybné. Strojovému učení je věnována kapitola 4.

Kromě prozódie se mohou uplatnit také rysy na sémantické a syntaktické úrovni. Sémantická informace může upravit ohodnocení hypotéz rozpoznávače a zvýšit ohodnocení hypotéz, které mají sémantický vztah ke svému kontextu. K tomu je vhodné ukládat data do podoby **sémantických sítí**. Sémantická síť je jedním z formátů reprezentace znalostí. Má podobu orientovaného grafu, v němž vrcholy reprezentují koncepty a hrany vyjadřují sémantické vztahy mezi koncepty. V případě úlohy rozpoznávání řeči uvažme např. větu „Výr je největší sova.“ Mohlo by se stát, že rozpoznávač vydá jako pravděpodobnější hypotézu se slovem „vír“. V sémantické síti však budou koncepty „sova“ a „výr“ v sémantickém vztahu, a tak správná hypotéza dostane vyšší ohodnocení.

3.4.4 Paralelní rozpoznávače

Pokud je výstup rozpoznávání na úrovni slov a promluv porovnán s výstupem rozpoznávače, který pracuje pouze na úrovni fonémů, mohou vyjít najevo nesrovnalosti, které poukazují na možné chyby v rozpoznávání. Porovnáním posloupností fonémů z obou typů rozpoznávačů se vlastně oddělí funkce jazykového a akustického modelování v rozpoznávači promluv a eliminuje se výlučná závislost na algoritmu dekódování.

Tento postup navrhli Cox a Dasmahapatra v *High-level Approaches to Confidence Measure Estimation in Speech Recognition*, 2002. Ačkoli však paralelní rozpoznávač fonémů vydával statisticky významné výsledky, základní techniku akustické stability se nepodařilo překonat.

3.4.5 Rozpoznávání vzorů

Další z přístupů k detekci chyb, založený na pravidlech, využívá obecnou teorii rozpoznávání vzorů (*pattern matching*). Nejprve se sestaví databáze chybových vzorů pro určitý jazyk nebo častěji pro specifickou doménu a potom se pomocí pravidel porovnává výstup rozpoznávače řeči vůči této databázi. Tento způsob detekce bývá úspěšný pro omezenou oblast, která je pokryta databází chybových vzorů, nepatří však mezi příliš robustní metody. Chyby, které nemají příslušnou šablonu v databázi vzorů, jsou přehlíženy, neboť systém nedokáže informace v databázi zobecňovat. Je také náchylný k vyznačování chyb tam, kde nejsou, pokud se správně přepsaná slova objeví v kontextu, v němž obvykle bývají chyby.

3.4.6 Konceptuální podobnost

Ve specializovaných oborech, kde existují rozsáhlé konceptuální modely znalostí, jako např. v medicíně, lze při detekci chyb v rozpoznávání řeči využít pojem konceptuální podobnosti. Porovnávání konceptů umožňuje provádět analýzu na vyšší než lexikální a syntaktické úrovni a může doplnit informace rozpoznávače podobným způsobem,

jako sémantické vztahy v sémantických sítích. Důležitým měřítkem, které se zde odvozuje, je kvantitativní skóre podobnosti mezi dvěma koncepty. Obecné techniky pracující se sémantickou podobností využívají míry vektorového prostoru a množinové operace. Na databázi znalostí uloženou ve formě hierarchické reprezentace lze nahlížet z pohledu dvou základních způsobů porovnávání konceptů – jednak pouze na základě vlastností hran, jednak s přihlédnutím k vlastnostem uzlů.

Podobnost podle hran

Jednou z metrik, která byla navržena pro definici vzdálenosti konceptů, je délka nejkratší cesty mezi koncepty, která vede po hranách s významem „širší než“ nebo „předek“. Při práci se systémem UMLS – Unified Medical Language System, systémem pro přehled znalostí z biomedicíny, byly použity míry *hloubka* a *konceptuální vzdálenost*. Hloubka udává skutečnou hloubku konceptu v hierarchii konceptů, přičemž koncept na vrcholu hierarchie je nejobecnější, hlouběji uložené koncepty jsou specializovanější. Konceptuální vzdálenost je založena na počtu hran nejkratší cesty. Hloubku konceptu lze při počítání vzdálenosti využít jako váhu. Dvojice specifičtějších konceptů pak získává vyšší hodnotu konceptuální podobnosti než dvojice obecnější, která je od sebe vzdálena stejný počet hran.

Podobnost podle uzlů

Problémem při zjišťování podobnosti pouze na základě počtu hran je předpoklad, že hrany v síti reprezentují uniformní a symetrické vzdálenosti. To však nemusí být pravda, pokud různé části sítě mají odlišnou hustotu dat nebo pokud jsou použity hrany s negativním významem. Počítání podobnosti bylo tedy rozšířeno o váhy, které zohledňují obsah informace v uzlu (konceptu). Obsah informace (*informational content*) konceptu c může být definován vztahem

$$IC(c) = -\log P(c).$$

Intuitivně platí, že koncepty, které mají vysokou frekvenci výskytu, a tedy vyšší pravděpodobnost, přinášejí menší množství informace. S pomocí obsahu informace můžeme odvodit další měřítko pro podobnost konceptů. Čím více informace dva koncepty c_1, c_2 sdílí, tím vyšší je jejich míra podobnosti (*similarity*). Formálně zapsáno

$$sim(c_1, c_2) = \max_{c \in S(c_1, c_2)} IC(c),$$

kde $S(c_1, c_2)$ je množina všech konceptů, které subsumují (zahrnují) jak c_1, tak c_2.

Klasifikace chyb rozpoznávání řeči i metod detekce chyb byla volně převzata z práce Voll [10].

Kapitola 4

Metody strojového učení

V předchozí kapitole byly zmíněny metody strojového učení jakožto způsob, kterým lze z určitých rysů vyšší úrovně rozpoznávaných vět získat skóre spolehlivosti přepisu vět (sekce 3.4.3). V této diplomové práci bude navržen postup, jak spočítat skóre spolehlivosti na základě finálního textového výstupu z rozpoznávače řeči. Budou rovněž aplikovány metody strojového učení.

Teorie strojového učení se zabývá otázkou, jak naprogramovat počítače tak, aby se dokázaly sami učit, tedy automaticky vylepšovat svoje schopnosti na základě zkušeností. Algoritmy strojového učení se hojně využívají např. v oblasti dobývání dat z databází (*data mining*).

Cílem algoritmů strojového učení je odvozovat zobecněná pravidla – hypotézy (zde v jiném významu než v případě výstupu z rozpoznávače mluvené řeči) na základě informací v trénovacích datech – zkušeností. Aby algoritmy dosahovaly očekávané úspěšnosti, potřebují dostatečně velké množství reprezentativních trénovacích dat. Byla vybudována přesná teorie, která popisuje vztahy mezi počtem trénovacích příkladů, počtem hypotéz, které se berou v úvahu, a očekávanou chybou naučených hypotéz. Popis teorie uvádí Mitchell [3, str. 201–229].

Rozlišuje se několik typů algoritmů strojového učení:

Učení s učitelem (*supervised learning*) – vytváří funkci, která mapuje vstup na požadovaný výstup. Patří sem klasifikační úlohy, v nichž má program za úkol naučit se (aproximovat) chování funkce, která mapuje vektor na jednu z množiny tříd. K tomu účelu jsou mu poskytnuty příklady této funkce v podobě dvojic (vstup, výstup).

Učení bez učitele (*unsupervised learning*) – agent, který si sám modeluje množinu vstupních dat, příklady vstupu a výstupu „od učitele“ nejsou k dispozici.

Kombinace učení s učitelem a bez učitele (*semi-supervised learning*) – kombinuje jak příklady z vnějšího zdroje, tak vlastní vygenerovaná vstupní data.

Učení posilováním (*reinforcement learning*) – algoritmus se učí postupy pomocí pozorování okolního prostředí. Každá jeho akce má nějaký vliv na toto prostředí, které mu pak poskytuje zpětnou vazbu, jež usměrňuje proces učení.

Transdukce (*transduction*) – algoritmus podobný jako učení s učitelem, na základě trénovacích příkladu se však nesnaží přímo sestrojit funkci (tedy vyvozovat obecné pravidlo), ale pouze určuje hodnoty pro konkrétní nová vstupní data.

Problém detekce chyb v rozpoznávání řeči lze chápat jako **klasifikační úlohu**. Klasifikace je postup, kdy se na základě trénovacích dat $\{(\vec{x_1}, y_1), \ldots, (\vec{x_n}, y_n)\}$ hledá funkce $d : \mathcal{X} \to \mathcal{Y}$ nazývaná *klasifikátor*, která přiřazuje instanci $\vec{x_i} \in \mathcal{X}$ příslušnost do třídy $y_j \in \mathcal{Y}$, $1 \leq i \leq n, 1 \leq j \leq |\mathcal{Y}|$. Vektor $\vec{x_i}$ je p-rozměrný vektor znaků měřených na trénovacích příkladech, $\mathcal{X} = \mathcal{X}_1 \otimes \ldots \otimes \mathcal{X}_p$ je stavový prostor a $\mathcal{Y}$ je množina tříd. Klasifikátor d potom musí být schopen přiřadit jednu z tříd y_j nové instanci $\vec{x_t} \in \mathcal{X}$ (testovacímu příkladu), kdy $\vec{x_t} \neq \vec{x_i}\ \forall i, 1 \leq i \leq n$. Znaky $x_i^1, \ldots, x_i^p$ mohou být buď spojité, tedy $x_i^l \in \Re$, nebo kategoriální, tedy $x_i^l \in \mathcal{C}$, kde $\mathcal{C}$ je množina konečného počtu kategorií, $1 \leq l \leq p$.

Pro účely detekce chyb lze za instanci $x_i^1, \ldots, x_i^p$ dosadit znaky (rysy) jednoho slova ve větě zapsané automatickým rozpoznávačem a množina tříd $\mathcal{Y} = \{$správně rozpoznané slovo, chybně rozpoznané slovo$\}$. Následuje teoretický popis tří metod strojového učení, které byly v této práci využity pro řešení klasifikační úlohy detekce chyb.

4.1 Logistická regrese

Model logistické regrese lze chápat jako jeden z typů zobecněného lineárního modelu (*generalized linear model – GLM*), který popisují Venables a Ripley [9, str. 183]. Pracuje s vysvětlujícími proměnnými (*stimulus variables*) $x_1, \ldots, x_p$ a závislou proměnnou (*response variable*) y. Vztah vysvětlujících proměnných a závislé proměnné je dán lineární funkcí

$$l(\mu) = \beta_0 + \beta_1 x_1 + \ldots + \beta_p x_p, \tag{4.1}$$

kde μ je střední hodnota rozdělení proměnné y, l je tzv. spojovací funkce (*link function*) a $\beta_0, \ldots, \beta_p$ jsou regresní koeficienty. Hustota rozdělení proměnné y pro binomické rozdělení s pevně daným počtem pokusů a a parametrem p má tvar

$$\log f(y) = a\left(y \log \frac{p}{1-p} + \log(1-p)\right) + \log \binom{a}{ay}.$$

Na výběr máme několik spojovacích funkcí. Často používanou funkcí ve spojení s binomickým rozdělením závislé proměnné je funkce *logit*, která byla zvolena i pro tuto úlohu:

$$l(\mu) = \ln \frac{\mu}{1-\mu}. \tag{4.2}$$

Za předpokladu, že proměnná y nabývá hodnot 0 a 1, střední hodnota μ odpovídá pravděpodobnosti, že $y = 1$, a $(1-\mu)$ je poté pravděpodobnost, že $y = 0$. Poměr $\frac{\mu}{1-\mu}$ se někdy nazývá šance, že nastane jev $y = 1$, dále zapisováno jako $šance(y = 1)$ (tento

poměr se běžně používá i mimo statistiku, např. při sázkách). Pravděpodobnost lze z šance snadno spočítat pomocí vztahu

$$\mu = \frac{šance(y = 1)}{1 + šance(y = 1)}.$$

Regresní koeficienty β_i jsou odhadovány metodou iterativních vážených nejmenších čtverců (*iterative weighted least squares*), [9, str. 185].

Pokud jsou některé z vysvětlujících proměnných kategoriální, místo $\beta_i x_i$ se pracuje s tzv. *dummy variables*, [9, str. 144–147]. Pro neuspořádané faktory o k třídách se zpravidla používají indikační proměnné $\alpha_2, \ldots, \alpha_k$ nabývající hodnot 0 nebo 1, přičemž nulovost všech $\alpha_2, \ldots, \alpha_k$ vyjadřuje příslušnost instance do třídy s indexem 1, která nemá explicitní indikační proměnnou.

Jestliže proměnná y v trénovacích datech nabývá hodnot 0 a 1, pak hodnota proměnné y pro nový příklad $\vec{x}$ predikovaná modelem logistické regrese je reálné číslo mezi 0 a 1 a odpovídá pravděpodobnosti $P(y = 1|\vec{x})$. Tento výsledek lze přímo použít jako skóre spolehlivosti slova, jemuž odpovídá vektor znaků $\vec{x}$.

4.1.1 Interpretace modelu

Vítanou vlastností modelu logistické regrese je možnost poměrně intuitivní interpretace hodnot koeficientů $\beta_0, \ldots, \beta_p$ ve vztahu vysvětlujících a závislé proměnné, díky níž je možné získat představu, jakým způsobem každá proměnná x_i ovlivňuje hodnotu závislé proměnné y. V situaci, kdy bude tento model aplikován na úlohu detekce chyb, proměnnými $x_1, \ldots, x_p$ budou znaky rozpoznaného slova a hodnoty závislé proměnné budou vyjadřovat příznaky

0 ... chybně rozpoznané slovo,
1 ... správně rozpoznané slovo,

lze analýzou odhadnutých koeficientů $\beta_0, \ldots, \beta_p$ zjistit, zda a jakým způsobem jednotlivé znaky slova souvisí s úspěšností rozpoznávání řeči.

Kombinací rovnic 4.1 a 4.2 dostáváme

$$\ln \frac{\mu}{1 - \mu} = \beta_0 + \beta_1 x_1 + \ldots + \beta_p x_p.$$

Koeficient β_0 lze interpretovat jako hodnotu logitu v případě, že by všechny ostatní atributy nebo jejich koeficienty byly nulové. Potom

$$\frac{\mu}{1 - \mu} = šance(y = 1) = \exp(\beta_0).$$

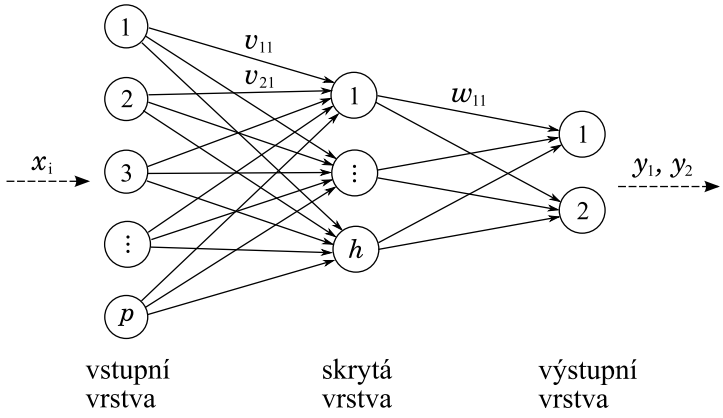
Predikovaná hodnota je za takových okolností shodná pro všechny testovací příklady a spočítá se jako

$$P(y = 1) = \frac{\exp(\beta_0)}{1 + \exp(\beta_0)}.$$

Tento údaj lze také chápat jako výchozí hodnotu pro predikovanou proměnnou y. Ta je dále upravena vlivem vysvětlujících proměnných. Koeficient β_i, $i = 1, \ldots, p$ lze interpretovat jako změnu logitu při změně hodnoty x_i o 1 a při nezměněných hodnotách ostatních vysvětlujících proměnných. Hodnota *šance*$(y = 1)$ se při takové změně násobí číslem $\exp(\beta_i)$. Je-li $\beta_i > 0$, pak $\exp(\beta_i) > 1$ a šance se zvětší, je-li $\beta_i < 0$, šance se zmenší.

4.2 Neuronové sítě

Obecný popis algoritmů strojového učení s použitím neuronových sítí uvádí Mitchell [3, str. 81–126]. Jednoduchým, ale přesto poměrně silným a flexibilním nástrojem jsou neuronové sítě s jednou skrytou vrstvou, práci s nimi popisují Venables a Ripley [9, str. 243]. Schematický příklad je na obrázku 4.1.



Obrázek 4.1: Příklad neuronové sítě s jednou skrytou vrstvou

Tyto sítě mají vstupní vrstvu, jejíž velikost závisí na počtu atributů p. Z ní se hodnoty přenášejí do jednotek skryté vrstvy, jejichž počet označíme h. V nich se vstupy vynásobí vahami a sečtou, přičte se konstanta a aplikuje se funkce ϕ_h. Výsledek se předá výstupní vrstvě, kde se přičte jiná konstanta a použije funkce ϕ_o. Výsledek na k-té výstupní jednotce lze tedy popsat jako

$$y_k = \phi_o \left(\alpha_k + \sum_{j=1}^{h} w_{jk}\, \phi_h \left(\alpha_j + \sum_{i=1}^{p} v_{ij}\, x_i \right) \right),$$

kde $x_1, \ldots, x_p$ jsou vysvětlující proměnné (v případě faktorů indikační proměnné), v_{ij} je váha na spojení mezi i-tou jednotkou vstupní vrstvy a j-tou jednotkou skryté vrstvy, w_{jk} je váha na spojení mezi j-tou jednotkou skryté vrstvy a k-tou jednotkou výstupní vrstvy. Jako „aktivizační funkce“ ϕ_h se obvykle volí logistická funkce

$$l(z) = \frac{\exp(z)}{1 + \exp(z)},$$

výstupní jednotky mohou být lineární, logistické nebo prahové. Síť může umožňovat přechody přímo mezi vstupní a výstupní vrstvou (*skip-layer connections*).

Pro kategoriální výstupní proměnné se obvykle používá tolik jednotek ve výstupní vrstvě, kolik je kategorií, a $y_k = -\log(P(y \in \mathcal{Y} \mid \vec{x}))$. Pro jednu spojitou výstupní proměnnou je ve výstupní vrstvě jediná jednotka.

Pro určení správných vah se používá mezi jinými metoda nejmenších čtverců, kdy je cílem minimalizovat chybu

$$E = \sum_{i=1}^{n} (t_i - y_i)^2,$$

kde t_i je požadovaná hodnota a y_i je aktuální výstupní hodnota sítě pro každý z n trénovacích příkladů. Dalším z možných kritérií pro minimalizaci je entropie. Iniciální hodnoty vah mohou být udány, nebo se vygenerují náhodné.

Pro dosažení hladké výstupní funkce se používá „váhový rozklad“ (*weight decay*), kdy trénovací kritérium má tvar $E + \lambda C(f)$. $C(f)$ je funkce, která penalizuje příliš velké váhy, λ je nastavitelný parametr. Tato úprava má rovněž předcházet přetrénování neuronové sítě. Podobná penalizace se používá i v jiných typech modelování. Funkce $C(f)$ obecně znevýhodňuje příliš složité funkce, v praxi se používá např. druhá derivace modelované funkce.

4.2.1 Interpretace modelu

V případě neuronových sítí bohužel není srozumitelná interpretace prakticky možná. Neuronová síť funguje jako „černá skříňka“ a natrénované váhy neposkytují užitečné informace o tom, jak se který znak x_i uplatňuje při predikci výstupní hodnoty.

4.3 Rozhodovací stromy

Rozhodovací stromy jsou rozšířeným nástrojem používaným pro modelování funkce, kterou lze zapsat ve formě pravidel typu „jestliže hodnota znaku A je B, pokračuj krokem C“. Jejich schéma má tvar zakořeněného stromu ve smyslu teorie grafů. Ohodnocení instance probíhá průchodem od kořene k listu, každý uzel testuje hodnotu atributu (znaku) a každá větev, která z uzlu vychází, odpovídá jedné hodnotě atributu, případně intervalu hodnot. Každý list pak přiřazuje instanci hodnotu predikované proměnné. Teorii rozhodovacích stromů uvádí Mitchell [3, str. 52–78].

Instancí dat, která vstupuje do rozhodovacího stromu, je opět vektor proměnných $x_1, \ldots, x_p$, požaduje se predikce proměnné y. Mezi rozhodovací stromy se řadí jednak klasifikační stromy, jejichž listy rozřazují instance do tříd, jednak regresní stromy, které přiřazují instancím odhad numerické proměnné. Regresní stromy rozdělují prostor $\mathcal{X}$ možných pozorování na „přihrádky“ odpovídající jednotlivým listům a v každé přihrádce je konstantní hodnota predikované proměnné.

Konstrukce klasifikačního stromu Při vytváření klasifikačního stromu od kořene směrem k listům bude kritériem pro dělení trénovacích dat $D = \vec{x_1}, \ldots, \vec{x_n}$ v jednotlivých uzlech stromu *entropie* těchto dat vztažená k příslušnosti instancí do jednotlivých tříd. Spočítá se jako

$$E(D) = -\sum_{i=1}^{|\mathcal{Y}|} p_i \log_2 p_i,$$

kde p_i je pravděpodobnost, že instance dat patří do třídy $y_i \in \mathcal{Y}$, odhadnutá pomocí relativní četnosti třídy y_i v datech. Redukce entropie po rozdělení dat podle kategoriálního atributu $x_A \in x_1, \ldots, x_p$ je pak

$$Rd(D, x_A) = E(D) - \sum_{v \in val(x_A)} \frac{|D_v|}{|D|} E(D_v), \tag{4.3}$$

kde $val(x_A)$ je množina všech možných hodnot atributu x_A, D_v je množina všech instancí, u nichž atribut x_A nabývá hodnoty v.

Pokud je některý z atributů spojitý, je třeba nejprve najít prahovou hodnotu c, která umožní rozdělit všechny instance na dvě části podle toho, jestli hodnota atributu je menší nebo větší než c (je možno určit i více než jednu prahovou hodnotu pro dělení na více částí, ale binární dělení se používá nejčastěji). Postupuje se tak, že se instance seřadí podle hodnoty tohoto atributu a potenciální dělící hodnoty jsou tam, kde se u dvou sousedních instancí liší příslušnost do tříd. Prahové hodnoty se spočtou jako průměr hodnot atributu pro tyto sousední instance, pro všechny prahové hodnoty se určí redukce entropie v případě jimi určeného rozdělení dat analogicky jako v rovnici 4.3 a tyto výsledky se pak mohou porovnat s redukcemi při rozdělení podle ostatních (spojitých i kategoriálních) atributů.

Konstrukce regresního stromu Při vytváření regresního stromu bude kritériem pro dělení dat D minimalizace součtu čtverců $S(T, D)$ ve stromě T, kdy

$$S(T, D) = \sum_{i=1}^{n} \left(y_i - \mu_{T[i]}\right)^2,$$

kde $T[i]$ označuje uzel stromu T přiřazený instanci i, $\mu_{T[i]}$ je predikovaná hodnota cílové funkce v uzlu $T[i]$ a určuje se jako průměrná hodnota cílové funkce pro všechny instance, které se při cestě od kořene stromu dostanou do tohoto uzlu.

Prořezávání Aby se zabránilo přetrénování stromu na daná data, provádí se prořezávání, konkrétně tzv. *cost-complexity pruning*. Při něm je cílem minimalizovat hodnotu

$$Measure(T, D) = S(T, D) + \alpha|T|,$$

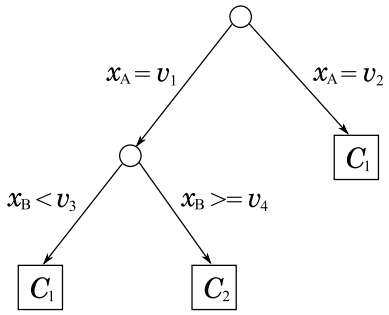
kde $|T|$ je počet listů stromu T a α je parametr, který určuje míru penalizace velikosti stromu. Vhodná hodnota parametru α se určuje *cross-validací* (nebo na heldout datech, jsou-li k dispozici).

Cross-validace znamená, že se trénovací data rozdělí na N stejných částí, jedna část se označí jako testovací a na ní se otestuje model, který byl vytvořen na základě zbylých $N-1$ částí dat. To se zopakuje N-krát vždy s jinou testovací částí a výsledky se zprůměrují.

Na počátku je celý strom před prořezáváním a postupně se odřezávají listy a zaznamenávají hodnoty α_k, které jejich odřezání způsobily. Cross-validací se spočítají kvadratické chyby ϵ_k jednotlivých prořezaných stromů. Jako výsledná hodnota α se může vybrat ta, která minimalizuje ϵ_k. Další možností je použít tzv. pravidlo 1-SE (zkratka zřejmě pochází od spojení *within one Standard Error of the minimum*), kdy se spočítá interval spolehlivosti pro minimální ϵ_k a vybere se největší α_k, jehož ϵ_k nepřesahuje horní mez tohoto intervalu spolehlivosti. O použití tohoto pravidla v praxi píší Venables a Ripley [9, str. 260].

4.3.1 Interpretace modelu

Interpretace rozhodovacího stromu je naprosto přirozená. Sestavená pravidla dělení dat podle hodnoty určitého atributu jsou čitelná i pro člověka a pokud je nějaká závislost mezi atributem a predikovanou hodnotou, lze ji obvykle z „rozumně“ velkého stromu vyčíst.



Obrázek 4.2: Jednoduchý klasifikační strom

Klasifikační strom lze chápat jako disjunkci konjunkcí jednotlivých dělících pravidel v uzlech stromu. Má podobu disjunkcí přes všechny cesty, které vedou z kořene do listu, jenž přiřazuje instancím příslušnou třídu. Pro třídy predikované stromem na obrázku 4.2 to bude

$$\begin{aligned} C_1 &\leftarrow ((x_A = v_1) \wedge (x_B < v_3)) \vee (x_A = v_2), \\ C_2 &\leftarrow (x_B \geq v_4). \end{aligned}$$

V případě regresního stromu nejspíše nebude disjunkce užitečná, neboť každý list bude pravděpodobně predikovat jinou hodnotu cílové numerické proměnné, avšak konjunkci tvořenou cestou od kořene k listu lze sestavit stejným způsobem.

Kapitola 5

Implementovaná detekce

V rámci této diplomové práce byla navržena detekce chyb v rozpoznávání mluvené řeči, která se opírá o tři metody strojového učení popsané v kapitole 4. Vstupními daty detektoru je prostý text tvořený větami zapsanými automatickým rozpoznávačem, jednotlivá slova mohou být ještě doplněna informací o ceně hrany v lattice (viz níže kapitola 5.1.1), pokud je k dispozici. Další informace nebyly dostupné ani v datech, která byla využita při vývoji. To je také důvod, proč nebyl implementován např. výpočet aposteriorní pravděpodobnosti slova, pro nějž je nutno znát pravděpodobnosti akustického a jazykového modelu použitého v rozpoznávači řeči. Výstupem detekce je skóre spolehlivosti pro každé rozpoznané slovo.

5.1 Data

Data, která byla použita jako trénovací a testovací, pocházejí z nahrávek českého rádia a televize (Czech Broadcast News Corpus, bližší popis uvádí Psutka a kol. [6], byl vydán na CD – Radová a kol. [7]).

5.1.1 Příprava dat

Získaná data jsou tvořena binárními soubory s výstupem rozpoznávače v podobě *lattice* uloženými ve formátu souborů FSM, v každém souboru je jedna rozpoznaná věta. Ke každému souboru byl k dispozici ruční přepis promluvy.

Lattice je orientovaný acyklický graf s jedním počátečním a jedním koncovým vrcholem, jeho hrany jsou označeny slovy a mohou obsahovat různé další informace podle toho, k jakému účelu je lattice určena. Používá se mj. pro ukládání sítě slov (kapitola 2.3.2) a pro výstup rozpoznávače v podobě grafu slov (kapitoly 2.3.2, 3.3.1). Příklad lattice pro graf slov vykreslený pomocí knihovny FSM je na obrázku A.1 v příloze A (po transformacích knihovnou FSM neobsahuje jediný koncový vrchol, to lze však snadno napravit přidáním nového vrcholu a spojením se všemi stávajícími koncovými vrcholy). Implementaci lattice pro účely rozpoznávání řeči obsahuje knihovna a sada nástrojů **HTK** – *Hidden Markov Model Toolkit* pod názvem *Standard Lattice Format – SLF*. HTK má svoje webové stránky na adrese

http://htk.eng.cam.ac.uk, odkud je možno jej zdarma stáhnout. Doprovází jej obsáhlý manuál The HTK Book [12].

FSM je knihovna nástrojů pro práci s konečnými automaty (*finite-state machines*) od společnosti AT&T Labs. Software je zdarma pro nekomerční účely a spolu s dokumentací je k dispozici na webu www.research.att.com/~fsmtools/fsm. Knihovna FSM je využívána též při rozpoznávání řeči, jak uvádí Mohri a kol. [4]. Umožňuje např. minimalizaci automatů, hledání průniku a sjednocení, zobrazení grafické reprezentace nebo také kompilaci do binárního formátu, v němž byla uložena získaná data.

V těchto datech jsou hrany lattice ohodnoceny cenou (*cost*), přičemž

$$cost = -\log P_A(w) - \beta \log P_L(w),$$

kde w je slovo patřící k dané hraně, P_A je pravděpodobnost v akustickém modelu, P_L je pravděpodobnost v jazykovém modelu a β je normalizační factor (*scaling factor*) jazykového modelu. Knihovna FSM nabízí program `fsmbestpath` s funkcí hledání nejlepší cesty automatu z počátečního do koncového stavu. Nejlepší cestou se myslí cesta s nejnižším součtem cen přes všechny hrany cesty (může se lišit podle definice ceny, místo součtu může FSM počítat součin nebo také logické &).

Aby bylo možno data použít pro zamýšlený detektor chyb, bylo potřeba převést je do podoby jednotlivých rozpoznaných vět v textovém formátu. Toho bylo dosaženo pomocí nástrojů knihovny FSM včetně `fsmbestpath` a z každé lattice byla vypsána jedna nejlepší hypotéza rozpoznané věty.

Dále bylo nutno pro každé slovo hypotézy zjistit, zda je správným přepisem či nikoli. Kvůli chybně vynechaným a vloženým slovům však nelze jednoduše porovnat rozpoznanou větu s ručním přepisem slovo po slovu, nýbrž je potřeba tyto dvě věty nejprve zarovnat. K tomu byl využit program `HResults` z knihovny HTK, který hledá optimální zarovnání rozpoznané a referenční (ručně přepsané) věty a počítá statistiky správnosti rozpoznávání. Optimální zarovnání zde znamená zarovnání s nejmenším počtem „trestných bodů“, přičemž identický řetězec nemá žádné trestné body, vložené a vynechané slovo mají po 7 bodech a zaměněné slovo přidává 10 bodů. Ze zarovnaných vět lze pak snadno odvodit správnost jednotlivých rozpoznaných slov.

5.1.2 Vlastnosti dat

Data zahrnují výstup z rozpoznávače mluvené řeči a ruční transkripci celkem 2 469 vět (33 565 slov). Celkový podíl chybně rozpoznaných slov (*word error rate – WER*) se spočítá jako

$$WER = \frac{S + I + D}{N},$$

kde S je počet zaměněných slov (*substitution*), I je počet vložených slov (*insertion*), D je počet vynechaných slov (*deletion*) a N je celkový počet slov v ruční transkripci. Pro tato data platí

$$S = 5\ 727, I = 708, D = 2\ 333, N = 33\ 565$$

a vychází $WER = 26,12\%$. Detekce chyb, která je založena na klasifikaci slov ve výstupu rozpoznávače, bohužel nedokáže odhalit chyby vzniklé vynecháním slova, neboť ve výstupu rozpoznávače se žádná známka vynechání nenachází a neexistuje tedy slovo, které by detektor chyb mohl na základě jeho atributů klasifikovat jako chybně vynechané. Proto se na datech počítá navíc s modifikovanou hodnotou WER^* jakožto podílem chybných slov v rozpoznaných větách, tedy chyb, které máme šanci odhalit:

$$WER^* = \frac{S + I}{N - D + I}.$$

Tato míra je zde rovna $20,15\%$.

5.1.3 Atributy slova pro strojové učení

Instancí pro strojové učení je vždy jedno slovo. Ke slovům byly dopočítány atributy, které lze získat z čistého textu a nejsou závislé na implementaci rozpoznávače řeči (s výjimkou atributu `cost`). Tabulka 5.1 vyjmenovává všechny použité atributy, druhý název atributu odpovídá označení ve zdrojových kódech.

Atribut	**Typ**	**Popis**
správnost `correctness`	kategoriální	absolutní správnost rozpoznaného slova podle ruční transkripce
pozice `pos`	celočíselný	pořadí slova ve větě
cena `cost`	spojitý	údaj z výstupu rozpoznávače řeči – cena na té hraně lattice, jejímž výstupním symbolem je dané slovo
délka slova `word.length`	celočíselný	délka slova ve znacích
1 poslední znak `last.char.merged`	kategoriální	poslední znak slova, znaky s malou četností sloučeny do speciálních tříd [č+]: č, ď, ň, ř, š, ť [b+]: b, c, p, j [f+]: f, g, ö, ú, w, x

1 poslední znak*
`last.char.`
`.samohl.souhl`
	kategoriální	jedna z následujících tříd:
		[samohlásky krátké]: a, e, ě, i, o, ö, u, y
		[samohlásky dlouhé]: á, é, í, ó, ú, ů, ý
		[souhlásky měkké]: c, č, ď, j, ň, ř, š, ť, ž (vč. d, t, n v di, ti, ni, dě, tě, ně)
		[souhlásky tvrdé]: d, h, k, n, r, t
		[souhlásky obojetné]: b, f, g, l, m, p, s, v, w, x, z

2 poslední znaky*
`last.2chars.`
`.samohl.souhl`
	kategoriální	třída určující příslušnost posledních dvou znaků mezi samohlásky nebo souhlásky stejně jako u posledního znaku, tedy např.

Atribut	Typ	Popis
		[souhlásky měkké, samohlásky dlouhé] pro kombinace, které se v datech vyskytují řídce nebo vůbec, je vyhrazena samostatná třída

3 poslední znaky*
`last.3chars.`
`.samohl.souhl`
	kategoriální	podobně jako u předchozího atributu, řídké a chybějící kombinace opět sloučeny, tentokrát do dvou tříd podle toho, jestli posledním znakem je samohláska nebo souhláska

P(slovo) `word_prob`	spojitý	pravděpodobnost slova v trigramovém jazykovém modelu
P(poslední znak) `last_1_prob`	spojitý	pravděpodobnost slova v trigramovém modelu posledního znaku slova
P(2 poslední znaky) `last_2_prob`	spojitý	pravděpodobnost slova v trigramovém modelu posledních dvou znaků slov
P(3 poslední znaky) `last_3_prob`	spojitý	pravděpodobnost slova v trigramovém modelu posledních tří znaků slov

Tabulka 5.1: Atributy slov pro účely strojového učení

Omezením při řešení této úlohy je velikost dostupných dat. Při navrhování atributů je nutné vzít v úvahu dostatečné zastoupení všech kategorií v trénovacích datech a také pokrytí celého prostoru možností, aby se v testovacích datech nevyskytly neznámé kategorie. Proto také nejsou jako atributy použity přesné „koncovky“ (jeden, dva a tři znaky), ale jejich sloučení podle příslušnosti mezi samohlásky a souhlásky a u vícepísmenných skupin ještě samostatné třídy pro řídce zastoupené kombinace.

Jazykový model a vyhlazování

Několik atributů slova v předchozí tabulce potřebuje ke svému spočítání jazykový model. Motivace pro použití trigramového modelu a odůvodnění nutnosti vyhlazování bylo uvedeno již v kapitole 2.3.2 o jazykovém modelu používaném při procesu rozpoznávání řeči. Protože kromě klasického modelu posloupnosti slov v jazyce bylo cílem vyzkoušet rovněž model posloupnosti posledního, dvou posledních a tří posledních znaků slov v jazyce, byl implementován vlastní mechanismus výpočtu pravděpodobností v modelu s použitím informací o četnostech n-gramů.

Jelikož nahrávky, na nichž bylo rozpoznávání prováděno, obsahují čtené – předem připravené – zprávy z rozhlasu a televize, jejich jazyk odpovídá mnohem více psaným textům než spontánní mluvené řeči. Jazykový model byl proto vytvořen na základě dat z vybraných časopisů z Českého národního korpusu, především z Lidových Novin, Mladé fronty Dnes a Profitu. Data mají podobu unigramových, bigramových a trigramových četností slov a obsahují celkem 614 708 různých slovních tvarů. Podle těchto četností byly spočítány četnosti posledních znaků slov a skupin dvou a tří posledních znaků.

Jako vyhlazování byla zvolena technika známá pod názvem *Modified Kneser-Ney Smoothing*, kterou popisují Chen a Goodman [1, str. 16–21] a uvádějí její vynikající

výsledky. Pravděpodobnost slova v n-gramovém modelu s použitím tohoto vyhlazování je definována jako

$$P_{KN}(w_i|w_{i-n+1}^{i-1}) = \frac{c(w_{i-n+1}^{i}) - D(c(w_{i-n+1}^{i}))}{\sum_{w_i} c(w_{i-n+1}^{i})} + \gamma(w_{i-n+1}^{i-1}) P_{KN}(w_i|w_{i-n+2}^{i-1}).$$

Zde $c(w_{i-n+1}^{i}) = c(w_{i-n+1}, \ldots, w_i)$ je četnost výskytu posloupnosti n slov w_{i-n+1}^{i}, $P_{KN}(w_i|w_{i-n+2}^{i-1})$ na konci řádku je stejným způsobem spočtená pravděpodobnost pro $(n-1)$-gram. Při výpočtu trigramové pravděpodobnosti se tak vždy projeví jak četnost trigramu, tak i četnosti bigramu a unigramu (pouze u prvního slova ve větě je použita jen unigramová četnost a u druhého slova kombinace unigramové a bigramové četnosti). Funkce $D(c)$ vrací pro různé četnosti posloupnosti různé konstanty odhadnuté předem z dat.

$$\begin{aligned} D(0) &= 0 \\ D(1) &= D_1 \\ D(2) &= D_2 \\ D(c) &= D_{3+} \ \forall c \geq 3 \end{aligned}$$

Funkce γ je definována tak, aby všechny pravděpodobnosti v pravděpodobnostním rozdělení P_{KN} dávali součet 1.

$$\gamma(w_{i-n+1}^{i-1}) = \frac{D_1 N_1(w_{i-n+1}^{i-1}\bullet) + D_2 N_2(w_{i-n+1}^{i-1}\bullet) + D_{3+} N_{3+}(w_{i-n+1}^{i-1}\bullet)}{\sum_{w_i} c(w_{i-n+1}^{i})},$$

kde $N_1(w_{i-n+1}^{i-1}\bullet)$ je počet různých slov, která spolu s historií w_{i-n+1}^{i-1} tvoří n-gram s četností 1, formálně

$$N_1(w_{i-n+1}^{i-1}\bullet) = |\{w_i : c(w_{i-n+1}^{i-1} w_i) = 1\}|,$$

analogicky $N_2(w_{i-n+1}^{i-1}\bullet)$ je počet slov, která s historií w_{i-n+1}^{i-1} tvoří n-gram s četností 2 a $N_{3+}(w_{i-n+1}^{i-1}\bullet)$ totéž pro n-gramy s četností 3 a více.

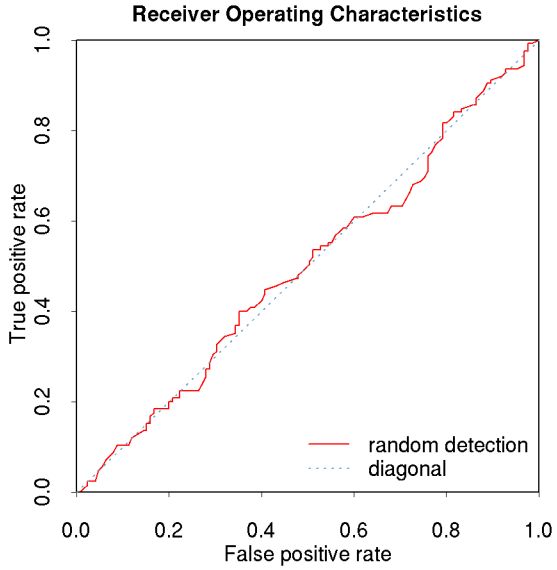
Optimální hodnoty parametrů D_1, D_2, D_{3+} byly určeny podle vzorců

$$\begin{aligned} Y &= \frac{n_1}{n_1 + 2n_2} \\ D_1 &= 1 - 2Y\frac{n_2}{n_1} \\ D_2 &= 2 - 3Y\frac{n_3}{n_2} \\ D_{3+} &= 3 - 4Y\frac{n_4}{n_3}, \end{aligned}$$

kde n_i je počet n-gramů, jejichž četnost je i. Jak navrhují Chen a Goodman [1], sada těchto tří parametrů byla spočtena zvlášť pro každý řád n-gramů, tj. $D_{1,1}, D_{1,2}, D_{1,3+}$ pro unigramy (n_1 pak bude počet všech unigramů s četností 1), $D_{2,1}, D_{2,2}, D_{2,3+}$ pro bigramy a $D_{3,1}, D_{3,2}, D_{3,3+}$ pro trigramy.

5.2 Vyhodnocování výsledků

Pokud detektor vydává na svém výstupu hodnoty v rozsahu $\langle 0, 1\rangle$, jako je tomu zde v případě skóre spolehlivosti, můžeme úspěšnost jeho detekce znázornit pomocí křivky ROC – **receiver operating characteristics**. Obrázek 5.1 znázorňuje, jak by určitě křivka pro fungující detektor neměla vypadat. Při počítání této křivky se postupně nastavuje práh (*threshold*) na reálná čísla mezi 0 a 1 a testovací příklady, jejichž skóre spolehlivosti přesáhne tento práh, se zařadí mezi příklady detektorem označené jako správné, ostatní mezi příklady označené jako chybné.



Obrázek 5.1: Náhodná detekce

Na osu x se pak vynáší tzv. *false positive rate*, tj. podíl případů, které jsou ve skutečnosti chybné, ale detektor (pro tuto hodnotu prahu) je označil za správné, ku všem chybným případům v datech. Jinými slovy je to podíl neodhalených chyb. Na osu y připadne tzv. *true positive rate*, neboli podíl správných případů, které i detektor označil za správné.

Použijeme-li místo detektoru generátor náhodných čísel mezi 0 a 1, bude se křivka ROC pohybovat kolem diagonály mezi body [0,0; 0,0], [1,0; 1,0] jako na obrázku 5.1. Nezáleží přitom na poměru skutečných chyb a správných slov v datech. Čím je detektor úspěšnější, tím více se křivka blíží levému hornímu rohu.

Při počítání bodů ROC lze rovněž určit hodnotu prahu, která by byla nejvýhodnější pro danou úlohu, pokud bychom od detektoru chyb vyžadovali přímé označování slov jako správně nebo chybně rozpoznaná namísto číselného skóre spolehlivosti. Při postupném nastavování prahu se nejspíše bude hledat taková hodnota, pro niž vychází co nejlepší *true positive rate (TPR)* a přijatelný *false positive rate (FPR)*.

Dalším možným měřítkem pro adekvátnost skóre spolehlivosti je statistická veli-

čina **směrodatná chyba** (*standard error*). Spočítá se jako

$$SE = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(\hat{y}_i - y_i)^2},$$

kde za $\hat{y}_i$ dosadíme skóre spolehlivosti přiřazené detektorem instanci s pořadím i a za y_i dosadíme správnost rozpoznání této instance (0 – chyba, 1 – správně rozpoznáno).

5.3 Metody strojového učení

Z metod strojového učení se pro úlohu detekce chyb nabízejí zejména rozhodovací stromy a neuronové sítě, neboť dokáží reprezentovat velmi obecné predikční funkce a nestanovují si požadavky na pravděpodobnostní rozdělení atributů instancí, jejich nezávislost apod. V této diplomové práci byly využity již hotové implementace metod strojového učení v knihovnách statistického softwaru R. Bližší informace o jazyku a výpočetním prostředí R i samotný software, který je šířen pod licencí GPL, lze nalézt na stránkách www.r-project.org.

Pro účely trénování a testování metod strojového učení byla data rozdělena na trénovací a testovací nejprve v poměru 9:1, poté ještě v poměru 2:1. Testovací data jsou tvořena $\frac{1}{10}$ a $\frac{1}{3}$ náhodně vybraných vět z celých dat.

5.3.1 Logistická regrese

Teoretický popis modelu logistické regrese byl uveden v kapitole 4.1. V programu R slouží k vytvoření zobecněného lineárního modelu funkce `glm`. Při vývoji detektoru se postupovalo od jednodušších modelů ke složitějším, ve kterých se přidávaly další vysvětlující proměnné a srovnávalo se, jak se zlepšují výsledky predikce. V prvním modelu byla analyzována závislost správnosti na ceně hrany z rozpoznávače:

```
> glm.costOnly <- glm(correctness ~ cost, data = train.data, family = binomial)
> confidence <- predict.glm(glm.costOnly, newdata = test.data, type = "response")
```

Prvním parametrem funkce `glm` je formule, která udává, jakou závislost má hledaný model vyjadřovat (stejně je tomu u většiny funkcí v R, které vytvářejí nějaký statistický model). Parametrem `family` se zadává pravděpodobnostní rozdělení závislé proměnné. Lze zde specifikovat i spojovací funkci, pro `family = binomial` je však defaultním nastavením právě `link = logit`, který použijeme.

Funkce `predict.glm` predikuje hodnoty závislé proměnné na základě modelu získaného funkcí `glm`. Pokud se v `predict.glm` uvede `type = "response"`, výsledkem predikce jsou odhady hodnoty závislé proměnné, jinak jsou predikovány hodnoty závislé proměnné modifikované spojovací funkcí.

Další modely:

```
> glm.wordsFeats <- glm(correctness ~ pos + word.length + cost + word_prob,
                        data = train.data, family = binomial)
```

```
> glm.noMerged <- glm(correctness ~ pos + word.length + cost + word_prob
                      + last_1_prob + last_2_prob + last_3_prob,
                      data = train.data, family = binomial)
> glm.full <- glm(correctness ~ pos + word.length + cost + word_prob + last_1_prob
                  + last_2_prob + last_3_prob + last.char.merged
                  + last.char.samohl.souhl + last.2chars.samohl.souhl +
                  last.3chars.samohl.souhl, data = train.data, family = binomial)
```

Výsledky

Výpis předposledního modelu dává nahlédnout, jak si model logistické regrese poradil se stanovením závislostí. (Nejsložitější model zde není uveden kvůli velikosti výpisu. Ten narostl zejména v důsledku použití kategoriálních regresorů, které byly převedeny na poměrně velký počet indikačních proměnných. Všechny výpisy jsou k dispozici v přiložených souborech na CD v adresáři vyvoj/R).

```
> summary(glm.noMerged)

Call:
glm(formula = correctness ~ pos + word.length + cost + word_prob +
      last_1_prob + last_2_prob + last_3_prob, family = binomial,
      data = train.data)

Deviance Residuals:
      Min         1Q    Median          3Q        Max
-4.8424       0.1621    0.6143      0.7333     2.9072

Coefficients:
                  Estimate Std. Error z value Pr(>|z|)
(Intercept)      8.415e-01  4.676e-02  17.997  < 2e-16 ***
pos             -1.567e-02  2.318e-03  -6.760 1.38e-11 ***
word.length      1.207e-01  6.513e-03  18.537  < 2e-16 ***
cost            -1.508e-04  9.368e-06 -16.102  < 2e-16 ***
word_prob        5.708e+00  4.234e-01  13.479  < 2e-16 ***
last_1_prob      2.728e+00  3.305e-01   8.257  < 2e-16 ***
last_2_prob      4.493e+00  6.142e-01   7.316 2.56e-13 ***
last_3_prob      1.851e+00  6.715e-01   2.756  0.00584 **
---
Signif. codes: 0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
...
```

Z výpisu je vidět, že od hladiny $\alpha = 0,01$ (**) jsou všechny koeficienty modelu signifikantní (zamítáme jejich nulovost). Všechny navrhované proměnné tedy mají významný vliv na predikci závislé proměnné. Jak tento vliv popsat přesněji?

Pro lepší názornost uveďme již známou rovnici

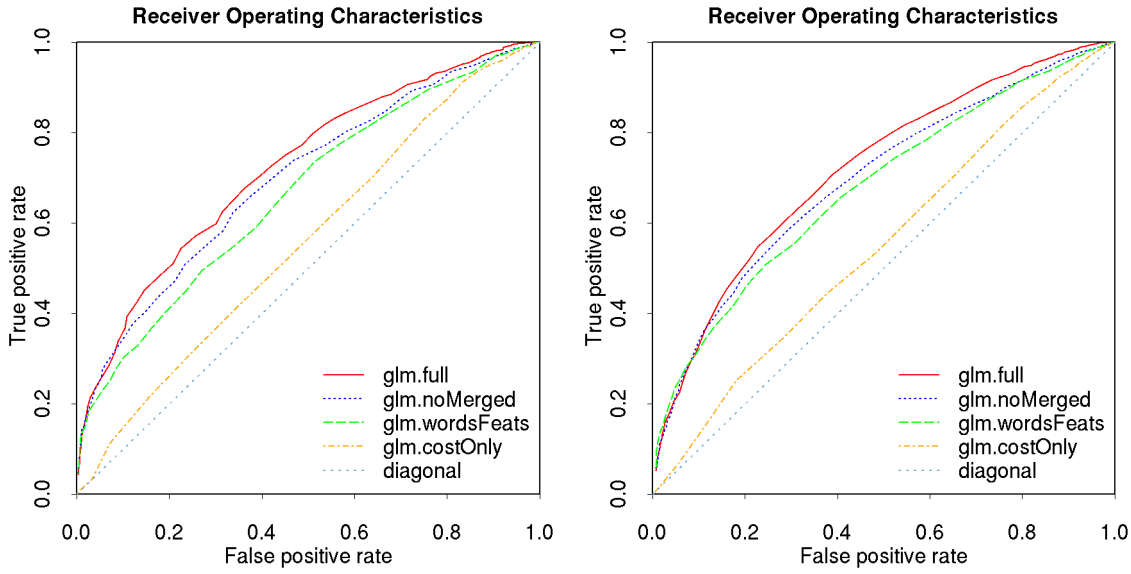
$$\ln \frac{\mu}{1-\mu} = \beta_0 + \beta_1 x_1 + \ldots + \beta_p x_p.$$

Hodnoty regresních koeficientů $\beta_0, \ldots, \beta_p$ jsou uvedeny ve výpisu ve sloupečku nadepsaném `Estimate`. První řádek s označením `Intercept` odpovídá koeficientu β_0.

Jak bylo řečeno v kapitole 4.1.1 pojednávající o interpretaci modelu logistické regrese, lze na koeficient β_0 pohlížet jako na hodnotu logitu v případě, že by všechny ostatní proměnné nebo jejich koeficienty byly nulové. Zde dostáváme $šance(y = 1) = \exp(0,8415) \approx 2,32$, což si lze představit tak, že $P(y = 1)$ je přibližně 2,32krát větší než $P(y = 0)$. Pro tyto hodnoty vychází $P(y = 1) \approx 0.699$. Toto je výchozí hodnota pro predikovanou proměnnou `correctness` modelem `glm.noMerged`.

Vlivem vysvětlujících proměnných se hodnota `correctness` dále mění. Uvažujme např. vysvětlující proměnnou `pos` v našem modelu. Její regresní koeficient má hodnotu $-0,01567$. Tvrdíme, že hodnota koeficientu odpovídá změně logitu při změně proměnné o 1. Při zvýšení pozice slova ve větě o 1 se tedy původní $šance(y = 1)$ násobí číslem $\exp(-0,01567) \approx 0,984$, tedy mírně zmenšuje $(2,32 \times 0,984 = 2,28288)$. Predikovaná hodnota se pak mění na $0,695$. Analogicky, ovšem v opačném směru, bude fungovat jednotková změna proměnné `word.length`. Šance se vynásobí hodnotou $\exp(0,1207) \approx 1,128$; $2,32 \times 1,128 = 2,61696$; dostáváme výsledek $0,724$.

V uvedeném modelu mají záporné znaménko pouze koeficienty pozice slova ve větě a ceny z rozpoznávače, tyto dva atributy tedy ovlivňují správnost slova negativně, pozice silněji než cena. Tento model tak nasvědčuje tomu, že ve slovech na začátku věty se chybuje méně. Nejvýraznější pozitivní vliv vykazuje pravděpodobnost slova v jazykovém modelu a pravděpodobnost posledních dvou znaků slova. Pravděpodobnost posledních tří znaků přináší ještě další informaci, ale je jí méně nejspíše proto, že velká část je již vysvětlena pravděpodobností dvou znaků. Délka slova mírně zvyšuje šanci na správné rozpoznání.



Obrázek 5.2: Srovnání modelů s různými množinami atributů

Na obrázku 5.2 je srovnání modelů logistické regrese různé složitosti. Přidání dalších atributů do modelu křivku ROC vždy zlepšilo. Vlevo jsou křivky pro rozdělení dat 9:1 a vpravo pro rozdělení 2:1. Z porovnání lze usoudit, že větší objem trénovacích dat

již regresní model nevylepšil (ROC pro 9:1 není lepší než ROC pro 2:1). Křivky pro rozdělení 2:1 jsou obecně méně kostrbaté díky důkladnějšímu otestování na větším počtu příkladů.

5.3.2 Neuronové sítě

Pro účely detekce chyb byla použita implementace neuronových sítí v knihovně `nnet` systému R. Trénování se provádí pomocí stejnojmenné funkce. Pro získání nejvýhodnějších hodnot parametrů `size` (velikost skryté vrstvy) a `decay` (parametr λ váhového rozkladu) byla implementována cross-validace, při které se postupně trénovaly neuronové sítě pro všechny hodnoty parametru `size`: 1–7, `decay`: 0; 0,003; 0,01; 0,1. K tomu účelu byla vždy trénovací data náhodně rozdělena na 10 částí, pro každou z kombinací parametrů `size` a `decay` byla 5krát (kvůli náhodným iniciálním vahám) na 9 částech natrénována síť a na 10. části otestována (predikční funkce se zde jmenuje `predict.nnet`).

```
...
learn <- nnet(formula, data = data[ri != i,], size = size, decay = decay,
              maxit = maxit, trace = F)
...
```

Celá procedura byla provedena nejprve pouze s atributy vztahujícími se k celým slovům, poté s úplnou množinou atributů, tedy

```
formula1 = correctness ~ pos + word.length + cost + word_prob
formula2 = correctness ~ pos + word.length + cost + word_prob + last_1_prob
                         + last_2_prob + last_3_prob + last.char.merged
                         + last.2chars.samohl.souhl + last.3chars.samohl.souhl
```

Srovnávacím kritériem byla směrodatná chyba.

Aplikace funkce tune.nnet

Dalším pokusem o získání vhodných parametrů neuronové sítě bylo využití funkce `tune.nnet` v knihovně R s názvem `e1071`. Ta má rozhraní

```
tune.nnet(x, y = NULL, data = NULL, size = NULL, decay = NULL, trace = FALSE,
          tunecontrol = tune.control(nrepeat = 5), ...)
```

Jak se ale ukázalo, tato funkce nezjišťuje hodnoty parametrů `size` a `decay`, nýbrž používá napevno zadané hodnoty, tudíž nenahradí ručně prováděnou cross-validaci. Defaultně není použit žádný váhový rozklad (`decay = NULL` odpovídá `decay = 0`), parametr `size` musí být zadán.

Pomocí funkce `tune.nnet` byly natrénovány neuronové sítě pro všechny kombinace `formula1`, `formula2`, `size = 6`, `size = 7`, `decay = 0`, `decay = 0,1`, což jsou parametry, které prokázaly dobrou úspěšnost při přímém použití funkce `nnet`.

Výsledky

Při vlastní cross-validaci byly vytvářeny sítě s velikostí vnitřní vrstvy $1, 2, \ldots, 7$. Omezení velikosti na nejvýše 7 jednotek bylo zvoleno podle implicitního omezení funkce `nnet` na nejvýše 1000 vah, které je nutno natrénovat. Toto omezení lze změnit pomocí parametru `MaxNWts`, ale větší počet vah se nedoporučuje z důvodu příliš dlouhého trénování a zpravidla nepřináší očekávané vylepšení sítě. Odpovídá tomu výsledek pokusu pro `formula2` s parametry `size = 15, decay = 0,1`, jehož ROC je pod označením `nnet.big` zahrnuta v obrázku 5.5 (porovnání směrodatné chyby spolu s ostatními pokusy je uvedeno v příloze v tabulce B.1).

Pro jednodušší formuli `formula1` nedosahuje počet vah ani při větší velikosti vnitřní vrstvy vysokých hodnot, avšak úspěšnost neuronové sítě se již nelepší (cross-validace byla provedena pro formuli 1 až do hodnoty `size = 20` a přestože některé směrodatné chyby byly nepatrně menší, výsledná síť vytvořená na celých trénovacích datech vykazovala menší úspěšnost detekce a větší rozptyl – opakovaně trénované sítě dávaly velmi odlišné výsledky).

V příloze v tabulce B.2 jsou porovnány směrodatné chyby ve všech krocích cross-validace (označení sloupečku „rozdělení dat“ je prvotní rozdělení na trénovací a testovací data, nikoli rozdělení pro cross-validaci, to je vždy 9:1). V tabulce jsou tučně zvýrazněny nejmenší směrodatné chyby pro danou formuli a rozdělení dat, které vycházejí pro tyto kombinace parametrů `size` a `decay`:

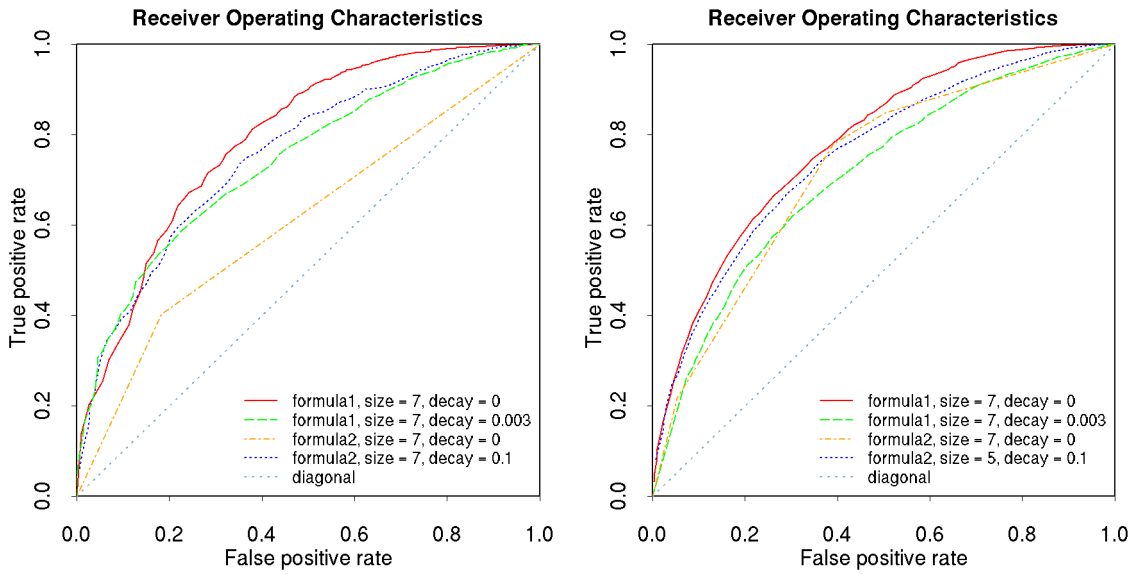
rozdělení dat 9:1 + formula1: size = 7, decay = 0,0 (SE = 0,3572550),
rozdělení dat 9:1 + formula2: size = 7, decay = 0,1 (SE = 0,3640548),
rozdělení dat 2:1 + formula1: size = 7, decay = 0,0 (SE = 0,3608342),
rozdělení dat 2:1 + formula2: size = 7, decay = 0,1 (SE = 0,3660084).

Pro tyto kombinace parametrů byly na celých trénovacích datech natrénovány finální neuronové sítě, které pak byly otestovány na testovacích datech a zároveň byly vytvořeny křivky ROC.

```
> nnet.CVBest <- nnet(correctness ~ pos + word.length + cost + word_prob,
                      data = train.data, size = 7, decay = 0, maxit = 1000)
> confidence <- predict(nnet.CVBest, newdata = test.data, type = "raw")
```

Výsledky jsou na obrázku 5.3, vlevo opět pro rozdělení dat 9:1 a vpravo pro rozdělení 2:1. Tentokrát přidání atributů do formule (`formula1` → `formula2`) detekci chyb nevylepšilo, spíše naopak. Vidíme, že při použití nenulového `decay` je křivka souměrnější. Při použití jednodušší formule 1 však váhový rozklad nebyl potřeba a k viditelnému přetrénování nedošlo, naopak výsledky jsou lepší než při aplikaci váhového rozkladu s `decay = 0,003` (což je nejlepší nenulová hodnota `decay` pro danou formuli a velikost skryté vrstvy).

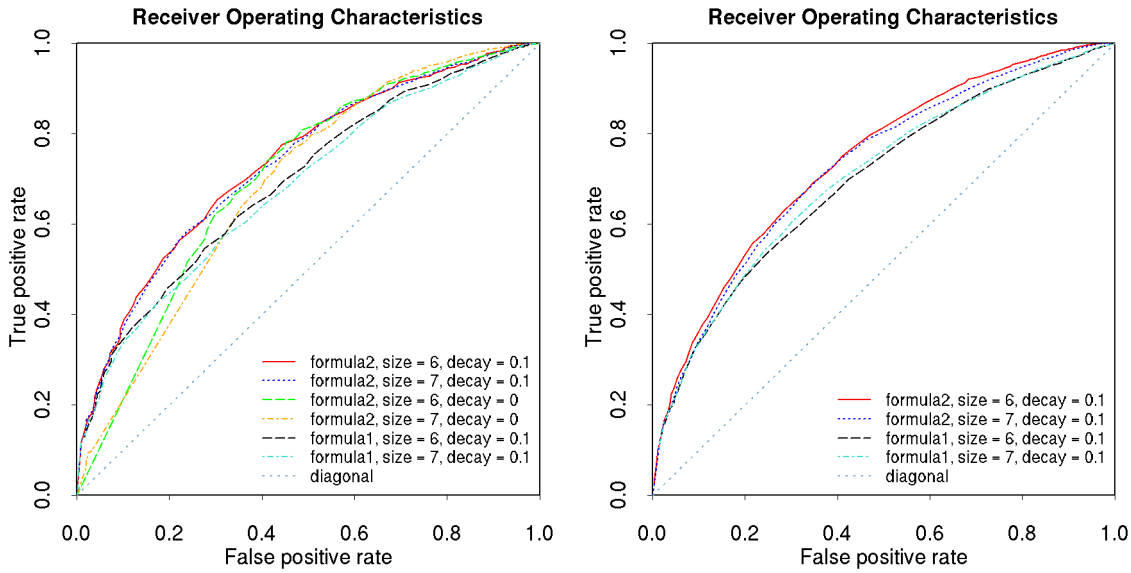
Jak je ovšem z obrázku patrné, při zahrnutí všech atributů (`formula2`) bez použití rozkladu k přetrénování došlo, obzvláště u větší množiny trénovacích dat. Zavedení rozkladu s `decay = 0,1` přibližuje výsledek pro `formula2` nejlepšímu výsledku (`formula1`, `size = 7`, `decay = 0`).



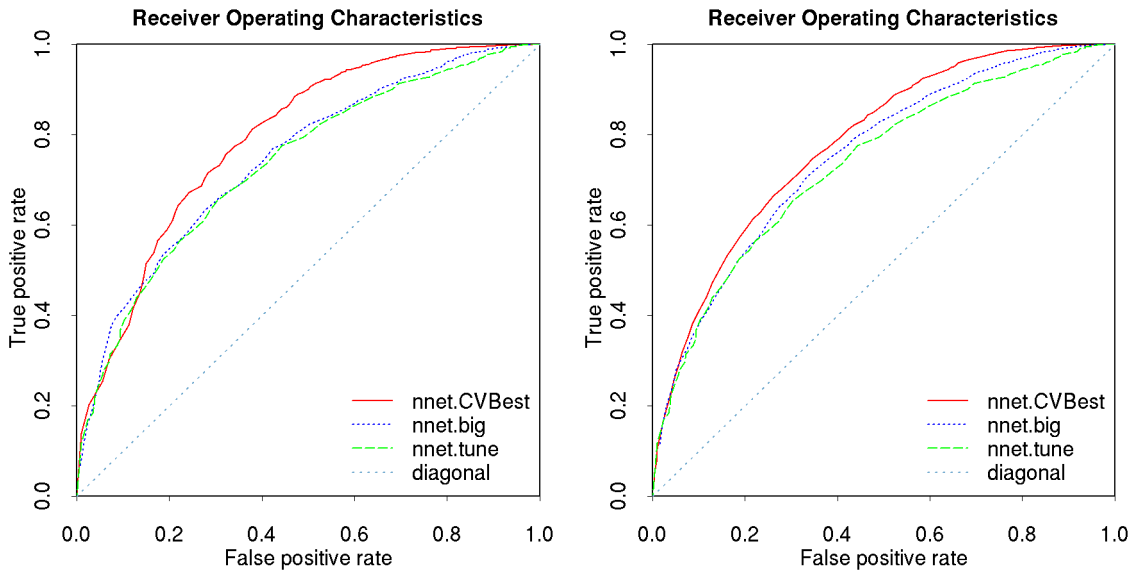
Obrázek 5.3: Výsledky neuronových sítí s nejlepšími parametry z cross-validace

Na obrázku 5.4 je srovnání pokusů vycházejících z funkce `nnet.tune`. S kombinacemi parametrů, pro něž zde není vyobrazena ROC křivka, vznikaly sítě, které všem testovacím instancím přiřadily predikovanou hodnotu `correctness = 1`.

Nakonec obrázek 5.5 ukazuje ROC výsledků nejlepší sítě vytvořené na základě cross-validace (`nnet.CVBest`: `formula1`, `size = 7`, `decay = 0`) vedle výsledků nejlepší sítě z `nnet.tune` (`formula2`, `size = 6`, `decay = 0,1`). Současně je zde výsledek sítě s větším počtem jednotek ve skryté vrstvě (`nnet.big` zmiňovaná v prvním odstavci: `formula2`, `size = 15`, `decay = 0,1`).



Obrázek 5.4: Výsledky `nnet.tune`



Obrázek 5.5: Srovnání nejlepších ROC neuronových sítí

V případě sítě `nnet.CVBest` více trénovacích dat úspěšnosti mírně pomohlo (směrodatná chyba 0.355 pro 9:1 oproti 0.36 pro 2:1, více v příloze B.1).

V opozici proti mnoha dobrým vlastnostem neuronových sítí stojí jedna jejich nezanedbatelná nevýhoda, totiž že neposkytují vysvětlení vztahu mezi vstupními atributy a výstupní proměnnou. Natrénované váhy obvykle nemají pro člověka srozumitelnou interpretaci, o čemž se můžeme přesvědčit v následujícím výpisu `> summary(nnet.CVBest)`. Další slabinou je náhodné nastavení iniciálních vah, kvůli němuž opakované trénování sítě se stejnými parametry vede k různým výsledkům.

```
> summary(nnet.CVBest)
a 4-7-1 network with 43 weights; options were - entropy fitting
      b->h1     i1->h1      i2->h1     i3->h1     i4->h1
       0.19      -1.32       -0.97      -1.58       0.03
      b->h2     i1->h2      i2->h2     i3->h2     i4->h2
      -0.73      -0.54        0.51       2.08       0.53
      b->h3     i1->h3      i2->h3     i3->h3     i4->h3
     -21.17    -268.30      -44.36    -192.71     -14.42
      b->h4     i1->h4      i2->h4     i3->h4     i4->h4
       5.13      -1.80        1.34      -0.10       0.94
      b->h5     i1->h5      i2->h5     i3->h5     i4->h5
      32.79     230.15      194.61     322.17      16.73
      b->h6     i1->h6      i2->h6     i3->h6     i4->h6
      -5.92       0.00       -0.15       0.00 -352047.46
      b->h7     i1->h7      i2->h7     i3->h7     i4->h7
      10.67      72.54       62.31      94.51       5.61
       b->o      h1->o       h2->o      h3->o      h4->o     h5->o     h6->o
      75.24       0.30       76.21      31.44       7.65     -1.15 -3001.60
      h7->o
    -148.33
(formula1, size = 7, decay = 0, data 9:1)
```

5.3.3 Rozhodovací stromy

Funkce `rpart` ve stejnojmenné knihovně R vytváří rozhodovací strom a zároveň počítá optimální prořezávání pro všechny možné hodnoty α_k, ϵ_k (kapitola 4.3) cross-validací při rozdělení trénovacích dat na 10 částí.

```
> rpart0001 <- rpart(correctness ~ pos + word.length + cost + word_prob
                     + last_1_prob + last_2_prob + last_3_prob + last.char.merged
                     + last.char.samohl.souhl + last.2chars.samohl.souhl
                     + last.3chars.samohl.souhl,
                     data = train.data, cp = 0.0001, method = "anova")
```

Parametrem `method = "anova"` se zadává požadavek na vytváření stromu, který predikuje číselné hodnoty (oproti `method = "class"` pro strom predikující zařazení do třídy). Parametr `cp` (*complexity parameter*) je $\alpha/S(T_0, D)$, kde T_0 označuje strom, který má pouze kořenový uzel. Nastavením `cp` se udává nejmenší hodnota α, kterou chceme brát v úvahu. Do formule zapíšeme všechny dostupné atributy slov a necháme na proceduře trénování a prořezávání, aby vybrala ty nejužitečnější.

Výše uvedené volání `rpart` vytváří velký strom až po velmi malé α způsobující jen malé prořezání. Funkce `printcp` vypíše výsledky cross-validace:

```
> printcp(rpart0001)

Regression tree:
rpart(formula = correctness ~ pos + word.length + cost + word_prob + last_1_prob
      + last_2_prob + last_3_prob + last.char.merged + last.char.samohl.souhl
      + last.2chars.samohl.souhl + last.3chars.samohl.souhl,
      data = train.data, method = "anova", cp = 1e-04)

Variables actually used in tree construction:
 [1] cost                 last.2chars.samohl.souhl    last.3chars.samohl.souhl
 [4] last.char.merged     last.char.samohl.souhl      last_1_prob
 [7] last_2_prob          last_3_prob                 pos
[10] word.length          word_prob

Root node error: 4614.2/28760 = 0.16044
n = 28760
            CP  nsplit   rel error    xerror       xstd
1    0.20633973       0     1.00000   1.00013  0.0088125
2    0.01874956       1     0.79366   0.79520  0.0090185
3    0.01213608       2     0.77491   0.77644  0.0089578
4    0.00901033       3     0.76277   0.76544  0.0087901
5    0.00877857       4     0.75376   0.75391  0.0087342
6    0.00834366       6     0.73621   0.75160  0.0087640
7    0.00655452       7     0.72786   0.74032  0.0086108
8    0.00512187       8     0.72131   0.73527  0.0085926
9    0.00417687       9     0.71619   0.73343  0.0086243
10   0.00323752      10     0.71201   0.72877  0.0085385
11   0.00323466      11     0.70877   0.72877  0.0085712
12   0.00294013      12     0.70554   0.72815  0.0085859
13   0.00271998      13     0.70260   0.72822  0.0085962
```

```
14   0.00259529      14     0.69988   0.72626  0.0085838
15   0.00248125      15     0.69728   0.72521  0.0085743
16   0.00247605      16     0.69480   0.72541  0.0085814
17   0.00235301      18     0.68985   0.72527  0.0085981
18   0.00230663      19     0.68750   0.72498  0.0086031
19   0.00219982      20     0.68519   0.72469  0.0086087
20   0.00208820      21     0.68299   0.72035  0.0086013
21   0.00207599      22     0.68090   0.71949  0.0086015
22   0.00205010      23     0.67883   0.71985  0.0086072
23   0.00199335      24     0.67678   0.71950  0.0086051
24   0.00174336      25     0.67478   0.71831  0.0086098
25   0.00173818      26     0.67304   0.71740  0.0086065
26   0.00157993      27     0.67130   0.71581  0.0086019  <- pravidlo1-SE
27   0.00128022      28     0.66972   0.71307  0.0086056
28   0.00127094      30     0.66716   0.71193  0.0086136
29   0.00126252      31     0.66589   0.71176  0.0086138
30   0.00123704      33     0.66336   0.71176  0.0086138
31   0.00119690      34     0.66213   0.71187  0.0086182
32   0.00106350      35     0.66093   0.71045  0.0086335
33   0.00098215      40     0.65561   0.70837  0.0086464
34   0.00097049      41     0.65463   0.70779  0.0086531  <- minimální xerror
35   0.00094841      42     0.65366   0.70831  0.0086595
36   0.00092640      43     0.65271   0.70898  0.0086732
37   0.00090449      44     0.65179   0.70947  0.0086830
38   0.00088906      46     0.64998   0.70932  0.0086866
39   0.00086867      47     0.64909   0.70917  0.0086955
40   0.00085704      48     0.64822   0.70925  0.0087025
...  (zbytek není zajímavý)
```

Chyby `rel error`, `xerror` a `xstd` udávají relativní chyby ku $S(T_0, D)$ – chybě stromu, který má pouze kořenový uzel. Sloupečky `xerror` a `xstd` závisí na náhodném rozdělení dat při cross-validaci.

Počet listů tohoto stromu je extrémní číslo 902 a bez prořezání se neobejdeme. Jako první byl zvolen prořezaný strom s hodnotou α pro minimální `xerror`, jež je na řádku 34. Zde `nsplit` = 41, strom tedy bude mít 42 listů. Pro další dělení uzlů již `xerror` roste, neboť rozhodovací strom se příliš adaptuje na konkrétní trénovací data a na testovacích datech pak dosahuje horších výsledků. Bylo tedy provedeno ořezání funkcí `prune` na základě odvozené hodnoty `cp`. Automatický výpočet vhodné hodnoty `cp` podle některé z metod R nenabízí.

```
> rpart.pruned <- prune(rpart0001, cp = 0.0009705)
```

Po tomto ořezání zbyly ve stromě jen některé atributy:

```
...
Variables actually used in tree construction:
[1] cost                 last.3chars.samohl.souhl     last_3_prob
[4] last.char.merged     word.length                  word_prob
...
```

V dalším pokusu se provedlo ořezání na základě pravidla 1-SE. Minimální `xerror` a `xstd` určují horní mez intervalu spolehlivosti jako 0,70779 + 0,0086531 = 0,7164431 a největší α, u něhož je `xerror` menší než tato hodnota, je na řádku 26. Jemu odpovídající strom bude mít 28 listů.

```
> rpart.pruned1SE <- prune(rpart0001, cp = 0.0016)
...
Variables actually used in tree construction:
[1] last.3chars.samohl.souhl    last_3_prob        last.char.merged
[4] word.length                 word_prob
...
```

Vidíme, že se dále „odřezalo“ použití atributu `cost`.

Následuje výpis některých počátečních dělení stromu, celý strom je k nahlédnutí v přiložených souborech (vyvoj/R/log_rpart.txt).

Pro každý uzel jsou na řádku uvedeny informace „`node), split, n, deviance, yval`“. Číslo `node` znamená číslo uzlu v pořadí prohledávání do šířky (*breadth-first search*). Část řádku `split` vyjadřuje podmínku, kterou musí splňovat instance, aby se při predikci dostala cestou z kořene do tohoto uzlu. Pro spojité znaky je zapsána jako nerovnost, pro znaky kategoriální jako výčet kategorií.

Číslo `n` je počet trénovacích instancí, které na základě podmínky `split` patří do podstromu, jehož kořenem je daný uzel. Tučně vyznačené `yval` je hodnota predikované proměnné pro instance spadající do tohoto podstromu a počítá se jako průměr této proměnné u trénovacích příkladů. Např. u kořene stromu, ve výpisu označeného „`1) root`“, má `yval` hodnotu 0,7992698, což je průměrná hodnota atributu `correctness` ze všech příkladů v trénovacích datech. Předposlední číslo na řádku, `deviance`, udává součet čtverců odchylek skutečné hodnoty `correctness` od hodnoty `yval` pro všechny trénovací příklady v podstromu.

Listy stromu jsou označeny hvězdičkou na konci řádku. Pokud instance po průchodu stromem skončí např. v uzlu 9), bude mu přiřazena predikovaná hodnota `correctness` = 0,2117647. Podle toho, jak je nastaven práh pro detekci, může být slovo považováno za správné i za chybné. Např. práh o hodnotě 0,2 zařadí slovo mezi správně rozpoznaná, při hodnotě 0,3 to již bude slovo chybné.

```
> print(rpart.pruned1SE)
n = 28760
node), split, n, deviance, yval
      * denotes terminal node
1) root  28760  4614.18500 0.79926980
  2) word_prob < 5.4255e-07  3283  679.28110 0.29241550
    4) word.length < 6.5  1348  119.07420 0.09792285
      8) last.3chars.samohl.souhl = 0.1,0.2,124,131,132,134,142,145,152,155,232,
         234,241,242,251,252,254,3,311,313,314,315,32,323,324,325,331,332,341,41,
         411,413,414,415,423,425,431,432,441,442,444,511,513,515,52,523,524,525,
         531,54,541,542,544,551,552  923  40.08884 0.04550379 *
```

```
    9) last.3chars.samohl.souhl = 114,115,141,144,15,151,154,231,342,351,424,
       451,452,514,532  425  70.94118 0.21176470 *
 5) word.length >= 6.5  1935  473.69300  0.42790700
  10) last_3_prob < 0.002039896  1241  269.27480  0.31829170
   20) word_prob < 1.7205e-07  783  143.37930  0.24137930
     40) last.3chars.samohl.souhl = 114,115,131,132,134,141,142,145,151,154,
         155,231,232,234,241,252,311,324,332,342,415,424,425,441,442,444,451,
         452,512,513,514,515,524,531,532,541,542,551 550  76.61091
          0.16727270 *
     41) last.3chars.samohl.souhl = 144,152,242,251,313,314,315,325,331,341,
         411,414,423,431,432,511,525,552 233  56.61803  0.41630900
   ...
3) word_prob >= 5.4255e-07  25477  2982.81400  0.86458370
  6) last_3_prob < 0.0002035813  3067  593.23510  0.73785460
   12) last.3chars.samohl.souhl = 1,115,13,3,31,312,32,323,324,35,4,41,5,511,
       52,54  123  23.18699  0.25203250 *
   13) last.3chars.samohl.souhl = 0.1,0.2,114,124,131,132,134,14,141,142,144,
       145,15,151,152,154,155,231,232,234,241,242,245,251,252,254,311,313,314,
       315,325,331,332,341,342,351,411,413,414,415,423,424,425,431,432,441,
       442,444,451,452,51,512,513,514,515,523,524,525,531,532,541,542,544,551,
       552  2944  539.80430  0.75815220
       ...
   7) last_3_prob >= 0.0002035813  22410  2333.58100  0.88192770
    14) word.length < 4.5  8510  1214.11100  0.82761460
        ...
    15) word.length >= 4.5  13900  1078.99700  0.91517990
        ...
```

Číselné značení kategorií pro posloupnosti znaků vychází ze značení skupin hlásek:

1 . . . samohlásky krátké (a, e, ě, i, o, ö, u, y)
2 . . . samohlásky dlouhé (á, é, í, ó, ú, ů, ý)
3 . . . souhlásky měkké (c, č, ď, j, ň, ř, š, ť, ž (vč. d, t, n v di, ti, ni, dě, tě, ně))
4 . . . souhlásky tvrdé (d, h, k, n, r, t)
5 . . . souhlásky obojetné (b, f, g, l, m, p, s, v, w, x, z)

Pro atribut `last.3chars.samohl.souhl` je potom slovo s koncovou trojicí hlásek např.„ení“ zařazeno do kategorie s označením 132. Speciální kategorie 0.1 a 0.2 jsou vyhrazeny pro kombinace hlásek, které se v trénovacích datech vyskytují málo nebo vůbec, přičemž do kategorie 0.1 padnou ty kombinace, v nichž poslední hláskou z trojice je samohláska, v 0.2 je poslední souhláska.

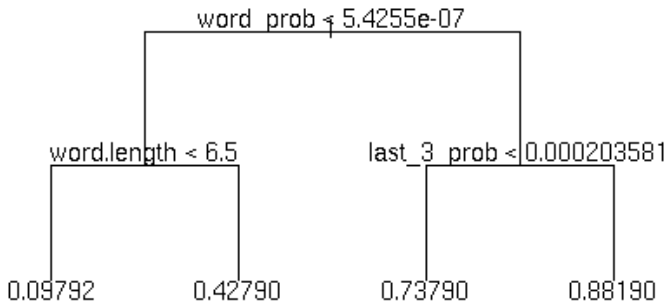
Z výpisu stromu je patrné, že největší schopnost rozdělit instance na správné a chybné má pravděpodobnost slova v jazykovém modelu (podmínka u uzlů 2 a 3), pro méně pravděpodobná slova se dále použije délka slova, pro více pravděpodobná pravděpodobnost posledních tří znaků.

Knihovna `e1071` nabízí „ladící“ funkci `tune.rpart`:

```
> rpart.tuned <- tune.rpart(correctness ~ pos + word.length + cost + word_prob
                            + last_1_prob + last_2_prob + last_3_prob
                            + last.char.merged + last.char.samohl.souhl +
                            last.2chars.samohl.souhl + last.3chars.samohl.souhl,
                            data = train.data)
> tree.tuned <- rpart.tuned$best.model
> printcp(tree.tuned)

Regression tree:
best.rpart(formula = correctness ~ pos + word.length + cost + word_prob
     + last_1_prob + last_2_prob + last_3_prob + last.char.merged
     + last.char.samohl.souhl + last.2chars.samohl.souhl
     + last.3chars.samohl.souhl, data = data)

Variables actually used in tree construction:
[1] last_3_prob   word.length   word_prob

Root node error: 4614.2/28760 = 0.16044
n = 28760
         CP nsplit  rel error    xerror       xstd
1  0.206340      0    1.00000   1.00008  0.0088121
2  0.018750      1    0.79366   0.79551  0.0090205
3  0.012136      2    0.77491   0.77759  0.0089761
4  0.010000      3    0.76277   0.76680  0.0088154
```

Tato funkce používá defaultní hodnotu `cp` = 0,01. Ve výsledném stromě jsou vidět stejná dělení jako u kořene výše uvedeného složitějšího stromu.

```
> print(tree.tuned)
n = 28760
node), split, n, deviance, yval
      * denotes terminal node
1) root  28760  4614.1850 0.79926980
  2) word_prob < 5.4255e-07  3283  679.2811 0.29241550
    4) word.length <  6.5  1348  119.0742 0.09792285 *
    5) word.length >= 6.5  1935  473.6930 0.42790700 *
  3) word_prob >= 5.4255e-07  25477  2982.8140 0.86458370
    6) last_3_prob <  0.0002035813  3067  593.2351 0.73785460 *
    7) last_3_prob >= 0.0002035813  22410  2333.5810 0.88192770 *

> plot(tree.tuned, uniform=T, compress=T)
> text(tree.tuned, digits=3)
```



Obrázek 5.6: Rozhodovací strom z `tune.rpart` pro data 9:1

Starší knihovna `tree` nabízí další implementaci klasifikačních a regresních stromů. Zde je vytváření a prořezávání stromů prováděno zvlášť. Výstupem z procedury `tree` byl v tomto případě strom totožný se stromem získaným z `tune.rpart` (a to jak pro data rozdělená 9:1, tak pro data 2:1).

```
> tree1 <- tree(correctness ~ pos + word.length + cost + word_prob + last_1_prob
                + last_2_prob + last_3_prob + last.char.merged
                + last.char.samohl.souhl + last.2chars.samohl.souhl
                + last.3chars.samohl.souhl, data = train.data)
> summary(tree1)

Regression tree:
tree(formula = correctness ~ pos + word.length + cost + word_prob
     + last_1_prob + last_2_prob + last_3_prob + last.char.merged
     + last.char.samohl.souhl + last.2chars.samohl.souhl
     + last.3chars.samohl.souhl, data = data)

Variables actually used in tree construction:
[1] "word_prob"  "word.length"  "last_3_prob"

Number of terminal nodes: 4
Residual mean deviance: 0.1224 = 3520 / 28760
Distribution of residuals:
      Min.    1st Qu.     Median       Mean    3rd Qu.       Max.
-8.819e-01  1.181e-01  1.181e-01  3.842e-16  1.181e-01  9.021e-01
```

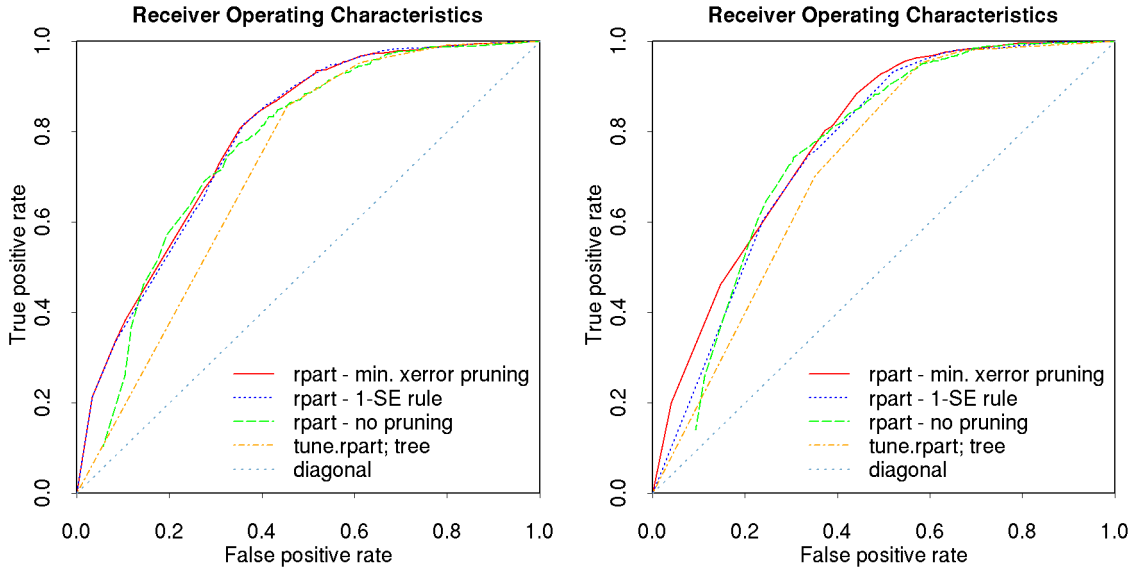
`> print(tree1)` – stejný výsledek jako „`> print(tree.tuned)`“

Výsledky

Následují ROC křivky rozhodovacích stromů před a po prořezání, jak je zvykem vlevo při rozdělení dat 9:1, vpravo při rozdělení 2:1.

Prořezání velmi snížilo komplexnost stromu a jak je z obrázku 5.7 patrné, nezhoršilo jeho detekční schopnost. Naopak ji ještě zlepšilo – jak klesá FPR přibližně k hodnotě 0,5 (tedy roste počet odhalených chyb), TPR klesá pomaleji (tedy vzniká méně chyb na správných slovech).

Další prořezání podle pravidla 1-SE snížilo počet listů z 42 na 28 (u dat 2:1 dokonce z 33 na 16) prakticky bez negativního vlivu na úspěšnost detekce (v případě dat 2:1 pouze s malým zhoršením). Směrodatná chyba stromu dokonce klesla z 0,3501208 na 0,3489094 (u dat 2:1 mírně vzrostla z 0,3430131 na 0,3472199). Složitější strom tedy nepřináší výrazné výhody.

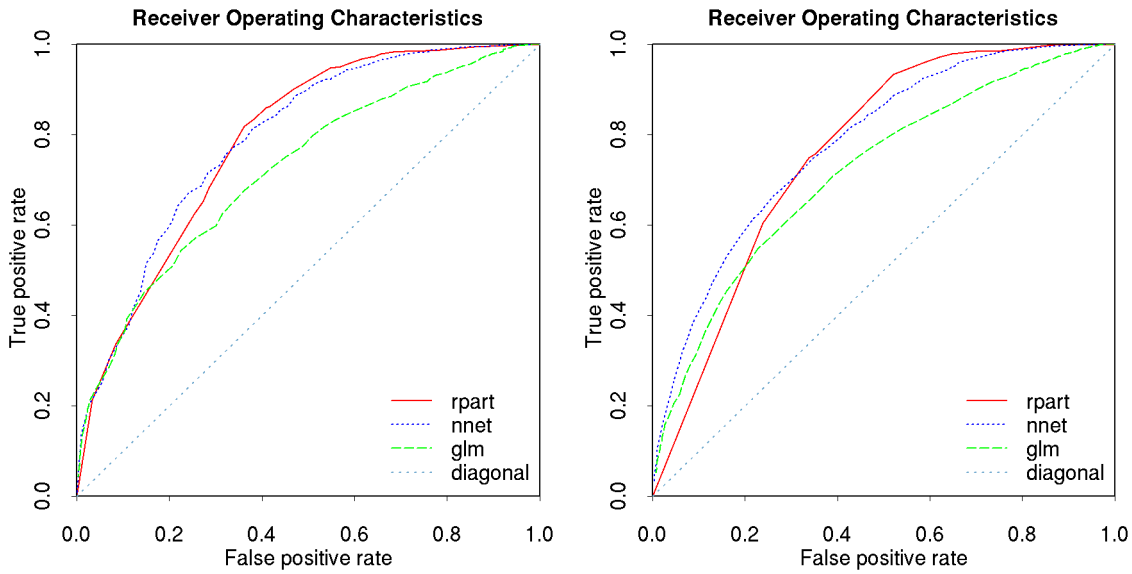


Obrázek 5.7: Srovnání ROC rozhodovacích stromů

I jednoduchý strom z knihovny `tree` a zároveň výsledek funkce `tune.rpart`, jehož výsledky jsou na obrázku pod označením „tune.rpart; tree“, vykazuje určitou schopnost detekce. Atributy `word_prob`, `word.length`, `last_3_prob` by se tedy mohly i v budoucnu uplatnit při řešení úlohy detekce chyb v rozpoznávání mluvené řeči.

5.4 Srovnání výsledků

Po porovnávání výsledků v rámci metod strojového učení následuje grafické srovnání nejlepších výsledků, jakých bylo s použitím jednotlivých metod dosaženo. Za logistickou regresi (glm) je zde vyobrazen výsledek modelu, jenž využívá všech dostupných atributů – „glm.full“ na obrázku 5.2 na str. 43. Neuronové sítě (nnet) reprezentuje síť s parametry `formula1`, `size = 7`, `decay = 0` neboli „nnet.CVBest“ z obrázku 5.5, str. 47. A konečně zástupcem rozhodovacích stromů (rpart) je strom s prořezáním podle pravidla 1-SE, na obrázku 5.7 pod popisem „rpart – 1-SE rule“.



Obrázek 5.8: Srovnání nejlepších ROC pro jednotlivé metody

5.4.1 Redukce WER

Při srovnání křivek ROC zjišťujeme, že neuronové sítě a rozhodovací stromy jsou přibližně na stejné úrovni a viditelně překonávají model logistické regrese. Křivky modelů rpart a nnet mají poněkud odlišný průběh, v levé části je křivka nnet nad křivkou rpart, v pravé část však rpart převyšuje nnet. Z porovnání směrodatných chyb vychází lépe rozhodovací strom (rpart: 0,3489094 nnet: 0,3546269 na datech 9:1). Je to proto, že body v křivce ROC nejsou rozmístěny rovnoměrně a více jich je tam, kde je rpart lepší než nnet.

Jak by mohly být tyto modely prakticky použity pro detekci chyb? Mohou pomoci zmenšit celkový WER v rozpoznaném textu? Pro konkrétní aplikaci je nutné brát v úvahu aktuální WER použitých dat a na základě této charakteristiky zvolit nejvýhodnější hodnotu prahu pro tato data. Volba prahu pro značkování dat nulami a jedničkami ovlivňuje detektor tak, že čím větší je hodnota prahu, tím více chybných slov je označováno jako chyba, ovšem také tím více správných slov je mylně označováno za chybu (slovo musí dosáhnout vyšší hodnoty skóre spolehlivosti, aby bylo považováno za správné). V grafu ROC roste práh směrem z pravého horního rohu do levého dolního rohu.

Logistická regrese

Mezi logistickými modely dosahuje nejlepších výsledků model využívající všech dostupných atributů (glm.full). Uvažujme v ROC křivce tohoto modelu (obrázek 5.8, glm) pro rozdělení dat 9:1 např. bod o souřadnicích

$$FPR = 0,5,$$
$$TPR = 0,78726,$$

který byl vypočten při nastavení hodnoty prahu na 0,749. *False positive rate* (*FPR*) je podíl chyb, které detektor nepoznal. Zde jich tedy nepoznal polovinu a zbylou polovinu se jako chybu označit podařilo. Chybných slov v datech (WER^* představený v kapitole 5.1.2) je přibližně 20%. Celkově tedy může být na základě detekce úspěšně opraveno

$$0,5 \times 20\% = 10\%$$

slov v datech. *True positive rate* (*TPR*) je podíl správných slov, která také byla označena jako správná. Zbylá část správných slov, tedy

$$1 - TPR = 1 - 0,78726 = 0,21274,$$

byla mylně považována za chybu. Protože správných slov je v datech přibližně 80%, bylo takto mylně označeno

$$0,21274 \times 80\% = 17,0192\%$$

slov v datech. Když od původního WER^* odečteme opravené chyby a přičteme nově vytvořené chyby, získáme $20\% - 10\% + 17\% = 27\%$.

Takto nastavený detektor chyb k očekávanému zlepšení WER^* nevede. Tento postup výpočtu výsledného WER_D však lze aplikovat postupně na všechny hodnoty (*FPR, TPR*) v ROC křivce a nalézt tak práh s nejlepšími potenciálními výsledky pro původní WER^*

$$WER_D = WER^* - (1 - FPR)\ WER^* + (1 - TPR)\ (100\% - WER^*).$$

Hodnota WER^* v části dat nemusí být přesně rovna WER^* v celých datech. WER^* v těchto konkrétních testovacích datech vychází o něco větši: 20,8%. Pro ROC modelu glm.full dostáváme nejlepší hodnoty výsledného WER_D pro práh 0,441, jenž na testovacích datech dává poměry

$$FPR = 0,944109,$$
$$TPR = 0,995242.$$

$WER_D \approx 20,8 - 1,163 + 0,377 \approx 20,01\%$. Zlepšení oproti původním 20,8% je nevýrazné, nicméně rozdíl při srovnání s první volbou prahu je značný. Je tedy vidět, že posuzování detektoru pouze na základě křivky ROC není dostačující a je třeba „promítnout“ poměry *FPR* a *TPR* do poměru chyb a správných slov v použitých datech.

Neuronové sítě

„Nejvystouplejší“ ROC křivku a nízkou směrodatnou chybu vykazuje síť s parametry `formula1`, `size = 7`, `decay = 0`. V této křivce se nalézá bod o souřadnicích

$$FPR = 0,676737,$$
$$TPR = 0,970262,$$

který odpovídá hodnotě prahu 0,49. Podíl chyb, které detektor identifikoval, je zde $1 - 0,676737 \approx 0,323$ ze všech chyb. Podíl správných slov označených jako chyba je $1 - 0,970262 \approx 0,0297$. Skvěle vypadající výsledek se trochu zhorší, když uvážíme, že chybných slov v textu je 20,8%. Potom nalezených chybných slov je $0,323 \times 20,8\% \approx 6,718\%$ slov z celého textu, zatímco správných slov označených jako chyba je $0,0297 \times 79,2\% \approx 2,352\%$. Výsledný WER_D v procentech vychází $\mathbf{20,8 - 6,718 + 2,352 = 16,43}$.

Rozhodovací stromy

Z vytvořených rozhodovacích stromů byl zvolen strom s 1-SE prořezáváním kvůli přijatelné velikosti a efektivitě detekce srovnatelné s větším stromem. Vidíme, že ROC křivka rozhodovacího stromu (obrázek 5.8, rpart) není souměrná, více se přibližuje hornímu okraji, kdežto levému je vzdálenější. Lze z toho soudit, že vhodná hodnota prahu bude jedna z těch, jejichž $FPR > 0,5$. Jak se při změně prahu zmenšuje FPR (tedy roste počet odhalených chyb), rychle roste počet chyb na správných slovech.

Skutečně na datech s 20,8% WER^* vychází jako nejlepší práh hodnota 0,42, která vygenerovala v ROC bod o souřadnicích

$$FPR = 0,614804,$$
$$TPR = 0,967486.$$

Z něj můžeme odhadnout WER_D v procentech na $\mathbf{20,8 - 8,012 + 2,575 = 15,36}$. Tento výsledek ještě o procento překonává výsledek neuronových sítí.

5.5 Kombinace metod

Další vývoj byl směřován k zavedení detekce, která by kombinovala více různých metod strojového učení. K vylepšení by mohlo dojít díky tomu, že jedna metoda dobře funguje při odhalení určitého druhu chyb, zatímco druhá metoda umí lépe zachytit jiné typy chyb.

Za tímto účelem byla dosavadní trénovací data rozdělena na první $\frac{3}{5}$ (data A) a druhé $\frac{2}{5}$ (data B). Na datech A byl postupně natrénován model logistické regrese, neuronová síť a rozhodovací strom, vše s parametry zvolenými jako nejlepší v předchozím textu. Poté byla získána predikce skóre spolehlivosti těchto tří modelů pro instance v datech B. Na datech B pak byly natrénovány tři kombinující modely, jejichž vstupními atributy byla skóre spolehlivosti ze tří různých modelů a údaj o správnosti rozpoznání slova, predikovanou hodnotou bylo nové skóre spolehlivosti.

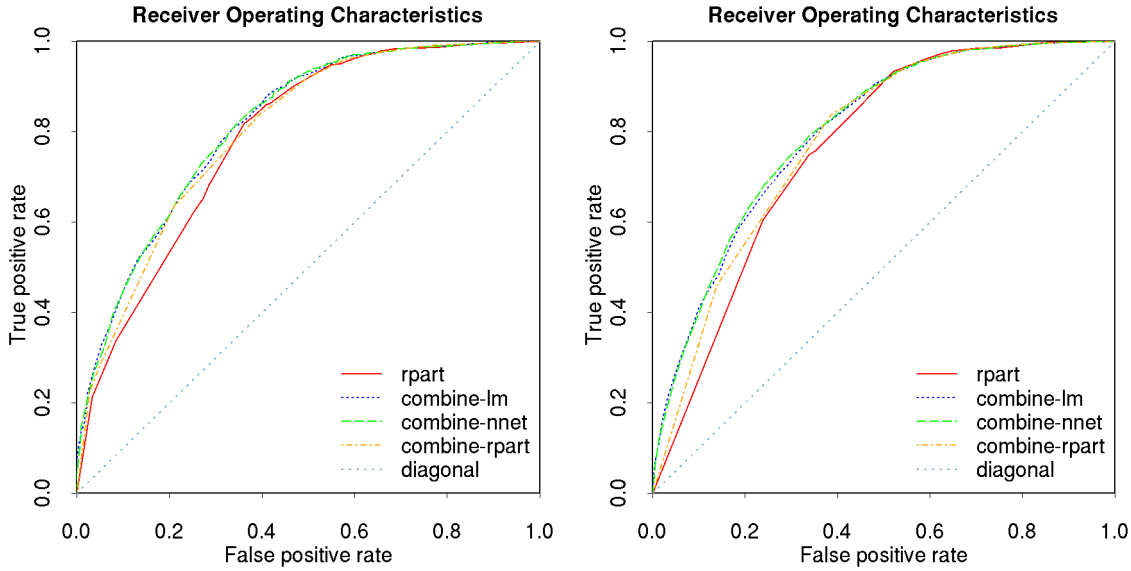
```
> combine.lm <- lm(correctness ~ glmA + nnetA + rpartA, data = dataB)
> combine.nnet0 <- nnet(correctness ~ glmA + nnetA + rpartA, data = dataB,
                        size = 3, decay = 0)
> combine.nnet <- nnet(correctness ~ glmA + nnetA + rpartA, data = dataB,
                       size = 3, decay = 0.1)
```

```
> combine.rpart <- rpart(correctness ~ glmA + nnetA + rpartA, data = dataB,
                         cp = 0.0015, method = "anova")
```

Velikost skryté vrstvy neuronových sítí zde byla stanovena na tři jednotky z důvodu podstatně menšího počtu vstupních atributů (tři reálná čísla), než tomu bylo u trénování na všech atributech slova.

Výsledky

Kombinující modely byly otestovány na testovacích datech, která zůstala stejná jako doposud. Nejprve bylo třeba na těchto datech určit skóre spolehlivosti tří základních modelů, poté byly postupně aplikovány modely `combine.lm`, `combine.nnet0`, `combine.nnet`, `combine.rpart`. Jejich ROC křivky jsou na obrázku 5.9 spolu s křivkou rozhodovacícho stromu rpart jakožto nejúspěšnější samostatné metody. Neuronová síť bez váhového rozkladu `combine.nnet0` zde chybí, všem testovacím instancím přiřazovala predikovanou hodnotu `correctness = 1`.



Obrázek 5.9: Výsledky modelů kombinujících různé metody

Směrodatné chyby jsou porovnány s chybami ostatních modelů v tabulce B.1. Došlo k mírnému zlepšení na chybu 0,3430705 u `combine.nnet` (ve srovnání s modelem rpart s chybou 0,3489094). Nejlepšího potenciálního snížení WER^* dosáhl kombinující model `combine.lm` a to zlepšení na 14,8% (doposud nejlepší hodnota 15,36% patřila rozhodovacímu stromu rpart).

Tohoto drobného zlepšení bylo dosaženo za cenu poměrně velkého zvýšení složitosti a náročnosti trénování, při němž se vlastně vytváří čtyři modely (tři základní a jeden kombinující). Pro výraznější vylepšení by pravděpodobně bylo zapotřebí dodat učícím algoritmům větší množství trénovacích dat, kterých takto měly méně než v případě samostatných metod, nebo použít lepší algoritmus kombinování.

5.6 Srovnání s jinými metodami

Přibližnou představu o tom, jak úspěšná obvykle bývá detekce chyb, lze získat nahlédnutím do článku [11], ve kterém Wessel a kol. prezentují výsledky svojí metody aposteriorní pravděpodobnosti (kap. 3.3.1) a srovnávají je s vlastní implementací metod akustické stability (kap. 3.4.1) a hustoty hypotéz (kap. 3.4.2). Ke srovnání používají kritérium *confidence error rate* – CER, které definují jako poměr nesprávně přiřazených příznaků detekce (správné/chybné slovo) ku celkovému počtu rozpoznaných slov. Základní CER (*baseline* CER) je pak dán počtem chyb vložení a záměn slov ku celkovému počtu rozpoznaných slov (jako kdyby detektor označil všechna slova za správná), což je přesně WER^*.

Ale CER není nic jiného než $\mathrm{WER_D}$. Do CER se počítají chybná slova označená za správná a naopak. A přesně tato slova tvoří výsledný $\mathrm{WER_D}$, neboť v datech buď zůstala chyba, protože ji detektor neoznačil, nebo vznikla nová chyba, protože detektor mylně označil správné slovo za chybu. Ostatní slova jsou buď správná a zůstávají, nebo jsou to detekované chyby, které jsou opraveny.

skutečná správnost:	1 1 **0** 1 0 **1** 1 1 1 1
detekce:	1 1 **1** 1 0 **0** 1 1 1 1
výsledek:	1 1 **0** 1 1 **0** 1 1 1 1

Wessel a kol. prováděli své pokusy na testovacích datech z pěti různých zdrojů. Výsledky jsou uvedeny v tabulce 5.2. Označení $\mathrm{CER_{HH}}$ znamená CER při použití hustoty hypotéz, $\mathrm{CER_{AS}}$ se vztahuje k metodě akustické stability a $\mathrm{CER_{AP}}$ k metodě aposteriorní pravděpodobnosti. Rel označuje relativní zlepšení chybovosti v procentech: $(\mathrm{WER}^* - \mathrm{CER})/\mathrm{WER}^*$.

Zdroj dat	WER^*	$\mathrm{CER_{HH}}$	$\mathrm{Rel_{HH}}$	$\mathrm{CER_{AS}}$	$\mathrm{Rel_{AS}}$	$\mathrm{CER_{AP}}$	$\mathrm{Rel_{AP}}$
ARISE	13,6	11,7	13,97	8,2	39,71	8,9	34,56
Verbmobil	27,3	26,0	4,76	22,1	19,05	18,9	30,77
NAB 20k	11,3	11,3	0,00	9,9	12,39	9,2	18,58
NAB 64k	9,2	9,1	1,09	8,0	13,04	7,2	21,74
Broadcast News	27,7	27,7	0,00	25,8	6,86	20,4	26,35

Tabulka 5.2: CER několika metod detekce chyb

ARISE je soubor nahrávek dialogů v holandštině mezi člověkem a počítačem, konkrétně automatickým informačním systémem pro vlakové jízdní řády. Německý korpus Verbmobil'98 je tvořen spontánními rozhovory nahranými ve vysoké kvalitě. Anglický korpus NAB'94 20k sestává ze čtených novinových článků opět ve vysoké kvalitě. V případě NAB 64k byl použit stejný korpus, lišila se však velikost slovníku rozpoznávače řeči. Korpus Broadcast News'96 obsahuje nahrávky rádia a televize, má tedy svou povahou nejblíže k datům používaným v této práci.

Stojí za povšimnutí, jak velmi se liší úspěšnost metod v závislosti na použitých datech. Metoda využívající hustotu hypotéz dokonce ve dvou případech nepřináší vů-

bec žádné vylepšení. Výsledky metody akustické stability jsou velmi rozdílné (39,71% vs. 6,86%). Nejmenší úspěšnosti dosáhla na korpusu Broadcast News, kde ani první metoda nezafungovala.

Tabulka 5.3 shrnuje výsledky detekce chyb implementované v této práci. Zde figuruje pouze jedna množina testovacích dat a ta má WER* = 20,798%. Výsledky pro kombinace metod v pravé části náleží vždy tomu pokusu, kde byla daná metoda strojového učení použita jako kombinující.

	Samostatná metoda		Kombinace metod	
Metoda	CER	Rel. zlepšení	CER	Rel. zlepšení
Logistická regrese	20,01	3,78	14,80	28,86
Neuronová síť	16,43	21,00	14,86	28,56
Rozhodovací strom	15,36	26,13	15,36	26,14

Tabulka 5.3: CER metod strojového učení

Je třeba si uvědomit, že srovnání výsledků z obou tabulek je pouze přibližné. Výsledky vznikly za jiných podmínek. Byla použita jiná vstupní data, což může hrát velkou roli, jak je vidět z první tabulky. Liší se rovněž rozpoznávané jazyky.

5.7 Závěr

Na navržených atributech prokázaly nejlepší schopnost detekce chyb rozhodovací stromy. Jejich velkou výhodou je také jejich intuitivnost a poměrně snadná čitelnost. Rozhodovací strom s 1-SE prořezáváním využívá atributy

- pravděpodobnost slova v trigramovém jazykovém modelu,
- délka slova,
- pravděpodobnost slova v trigramovém modelu posledních tří znaků,
- poslední tři znaky slova (třída podle samohlásek a souhlásek),
- poslední znak slova.

Podle ROC a vypočtené redukce WER následují v úspěšnosti neuronové sítě, ačkoli bychom možná očekávali – na základě jejich univerzálnosti a velké oblíbenosti v posledních letech – že právě neuronové sítě přinesou výsledky nejlepší. Není vyloučeno, že při použití jiných kombinací atributů by dosáhly ještě lepších výsledků, avšak kvůli jejich značné neprůhlednosti by k tomu bylo zapotřebí aplikovat metodu „pokus–omyl“. Na rozdíl od zbylých dvou použitých metod neuronové sítě nevydávají užitečné informace o tom, jak se jednotlivé atributy podílejí na predikci výsledné hodnoty.

Podle rozhodovacího stromu i metody logistické regrese je nejúčinnějším predikčním atributem pravděpodobnost slova v jazykovém modelu. Každý z předložených atributů byl v některé z metod do určité míry využit, žádný se nezdá být naprosto nevhodný.

Kapitola 6

Popis programu

Součástí práce je program, který obstarává přípravu dat, počítá atributy pro metody strojového učení, trénuje a aplikuje metody strojového učení a vyhodnocuje výsledky detekce včetně vytvoření křivky ROC. Program slouží zejména pro vývoj a demonstraci navrhovaných metod detekce. Byl napsán pro operační systém Linux v programovacím jazyku C++ s využitím prostředků FSM, HTK a softwaru R.

6.1 Uživatelské rozhraní

Na doprovodném CD v adresáři zdrojove-kody se nacházejí všechny zdrojové kódy potřebné pro kompilaci a běh programu. Program se zkompiluje příkazem `make` spuštěným v tomto adresáři. Spustitelný soubor bude vytvořen ve stejném adresáři, je tedy nutné mít v něm právo zápisu (adresář zdrojove-kody z CD tedy nejprve zkopírovat na disk). V adresáři program je již zkompilovaný spustitelný soubor, preferovaným postupem je však vlastní kompilace.

Aby program správně fungoval, musí být ve stejném adresáři, jako je spustitelný soubor, adresáře data_files, perl_code a R_code. Tak tomu je po kompilaci ze zdrojových kódů, je pouze třeba zajistit tyto podmínky v případě, že by byl spustitelný soubor přemísťován.

Program se spouští z příkazové řádky příkazem, který má tvar

```
detekce [--volba-faze-1 ... --volba-faze-N] --parametry SOUBOR-S-PARAMETRY
```

Chod programu se dělí na několik fází, přičemž volbami příkazové řádky se zadává, zda chceme spustit jednu konkrétní fázi, postupně několik fází, nebo postupně všechny fáze od začátku do konce (k tomu dojde, když při volání programu není uvedeno jméno žádné fáze). Program zahrnuje tyto fáze:

1. **extrakce rozpoznaných vět:** volba příkazové řádky `--rozpozn-vety`
   – získání rozpoznaných vět z lattice uložených v binárních souborech typu *.fsm (*finite state machine*)

2. **výpočet *n*-gramových četností znaků:** volba `--ngramy-znaky`
   – výpočet unigramových, bigramových a trigramových četností pro jeden, dva a tři poslední znaky slov

3. **analýza rozpoznaných vět:** volba `--analyza-dat`
   – výpočet atributů slov pro algoritmy strojového učení

4. **rozdělení dat na trénovací a testovací:** volba `--rozdeleni-dat`

5. **spuštění strojového učení:** volba `--stroj-uceni`
   – natrénování jedné ze tří metod strojového učení (logistické regrese, neuronové sítě nebo rozhodovacího stromu) a spočtení skóre spolehlivosti

6. **vyhodnocení výsledků:** volba `--vysledky`
   – spočtení úspěšnosti skóre spolehlivosti, vykreslení křivky ROC

Pro spuštění první fáze je třeba mít nainstalovány knihovny FSM a HTK. Obě je možno nalézt v aktuální verzi na CD v adresáři potrebne-knihovny. FSM je distribuováno v binární podobě, je pouze třeba rozbalit soubor fsm-4_0.linux.i386.tar.gz a přidat cestu ke spustitelným souborům do proměnné `$PATH`. Nástroj HTK se instaluje způsobem obvyklým v Linuxu pomocí příkazů `configure`, `make`, `make install`.

Třetí fáze, analýza rozpoznaných dat, využívá ke zpracování dat skript napsaný v jazyce Perl. Je proto třeba mít Perl nainstalovaný, což ve většině distribucí Linuxu bývá standardně splněno.

Fáze 5 a 6 vyžadují, aby byl k dispozici software R, pro fázi 5 navíc musí obsahovat knihovny nnet a rpart. Program by měl správně fungovat s R ve verzi 2.5.1 a vyšší. R lze nainstalovat buď ze zdrojových kódů na CD v adresáři software-R, nebo ze stránek www.r-project.org stáhnout binární instalaci pro příslušnou distribuci Linuxu.

6.1.1 Soubor s parametry

Kromě výběru spouštěné fáze se všechny potřebné parametry předávají programu prostřednictvím textového souboru, jehož umístění se zadává volbou

```
--parametry SOUBOR-S-PARAMETRY
```

V tomto souboru jsou uvedeny všechny parametry ve formátu „jméno=hodnota“. Většina parametrů je jednoho z následujících typů:

Jméno souboru – absolutní nebo relativní cesta k souboru (relativní vůči adresáři, z něhož je spouštěn program)

Jméno adresáře – jméno adresáře opět s absolutní nebo relativní cestou, lomítko na konci se může uvést, ale není nutné

Příznak ano/ne – hodnotou parametru může být řetězec „ano“, zkráceně „a“, nebo řetězec „ne“, zkráceně „n“; pokud parametr tohoto typu není při spuštění fáze v souboru uveden, chápe se stejně, jako kdyby bylo uvedeno „ne“.

Výjimku z tohoto seznamu tvoří dva parametry. Prvním je `pomer-tren-test`, který se zadává jako poměr dvou čísel `X:Y`, druhým je parametr `metoda`, který nabývá jedné ze tří hodnot – názvů metod strojového učení (přesněji uvedeno v popisu páté fáze, spuštění strojového učení).

Každá fáze má přiřazenu množinu parametrů, které potřebuje znát, některé mohou být nepovinné. Některé parametry jsou využívány ve více fázích. Typickým případem je jméno souboru, do kterého se v jedné fázi ukládají výstupní data a další fáze z něj čte svá vstupní data. Je proto výhodné ukládat parametry všech fázi dohromady do jednoho souboru. Parametry, které spuštěná fáze nevyužívá, jsou v tu chvíli ignorovány.

6.1.2 Popis jednotlivých fází a jejich parametrů

Extrakce rozpoznaných vět

Soubor parametrů musí obsahovat tyto údaje:

- `fsm-adresar` – adresář, ve kterém jsou uloženy soubory *.fsm s výstupem rozpoznávače řeči v podobě lattice
  data/01-rozpozn-vety/recog na CD
- `slovnik` – soubor, který mapuje slova ze slovníku rozpoznávače na čísla, která jsou použita jako výstupní symboly ve FSM souborech, bez tohoto souboru by byly rozpoznané věty pouze posloupností čísel
  data/01-rozpozn-vety/wordmap na CD, soubor je v kódování ISO 8859-2
- `master-label-file` – soubor v HTK formátu *Master Label File*, obsahuje ruční přepisy vět, jejichž automatické přepisy jsou v souborech FSM, a slouží k určení správného a chybného rozpoznání slov
  data/01-rozpozn-vety/labels.mlf na CD
- `vety-adresar` – adresář pro uložení výsledných vět, kterým odpovídá nejlepší cesta v příslušném FSM souboru; pokud neexistuje, bude vytvořen
  data/01-rozpozn-vety/recog-bestpath na CD
- `spravnost-slov` – soubor pro uložení seznamu, v němž vedle jména FSM souboru je zapsána posloupnost 1, 0 a D vyjadřující správně rozpoznané slovo (1), chybně rozpoznané slovo (0) a vynechané slovo (D)
  data/01-rozpozn-vety/correctness.map na CD

Program pomocí nástrojů knihovny FSM získá zápis nejlepší věty z každého souboru jmeno-fsm.fsm v adresáři `fsm-adresar` a uloží jej do adresáře `vety-adresar` do souboru jmeno-fsm.sntnc. Do podadresáře costs umístí soubor jmeno-fsm.cst s cenami na hranách lattice. Věty v souborech *.sntnc jsou pak porovnány s větami v `master-label-file` za použití HTK programu `HResults`, výsledek je uložen do souboru `spravnost-slov`.

Na přiloženém CD jsou všechny vstupní i vygenerované soubory umístěny v adresáři data a rozděleny do podadresářů podle fáze, v níž byly poprvé použity.

Výpočet *n*-gramových četností znaků

V této fázi se počítají n-gramové četnosti posledních znaků slov, které se použijí ve fázi analýzy dat pro výpočet pravděpodobností slov v n-gramových modelech. Máme-li v souboru s trigramy slov např. trojici „vypuknutí březnových nepokojů“ s četností c, přispěje tato trojice při výpočtu trigramu „utí ých ojů“ ke zvýšení četnosti o c. Parametry pro tuto fázi:

- `unigramy-slova` – soubor s četnostmi samostatných slov (přesněji slovních tvarů)
  data/02-ngramy-znaky/unigrams
- `bigramy-slova` – soubor s četnostmi dvojic slov
  data/02-ngramy-znaky/bigrams
- `trigramy-slova` – soubor s četnostmi trojic slov
  data/02-ngramy-znaky/trigrams
- `ngramy-znaky-adresar` – adresář, do nějž se mají uložit soubory s vypočtenými četnostmi znaků
  data/02-ngramy-znaky
- `pocitat-unigramy` – příznak, zda se mají počítat unigramové četnosti znaků
- `pocitat-bigramy` – příznak, zda se mají počítat bigramové četnosti znaků
- `pocitat-trigramy` – příznak, zda se mají počítat trigramové četnosti znaků

Program umožňuje počítat v jednom běhu četnosti pouze některého řádu (některých řádů) pro případ, že četnosti jiného řádu jsou již známy. Zejména počítání trigramových četností je značně náročné na výpočetní prostředky a dosti zdlouhavé, proto můžeme např. nechat spočítat unigramy bez nutnosti přepočítávání trigramů. Pokud je nastaven kladný příznak pro počítání některých n-gramů, musí být zadána cesta k souboru příslušných n-gramů slov (první tři parametry), jinak jsou tyto parametry nepovinné. Výsledek je uložen do adresáře `ngramy-znaky-adresar` do souborů s pevně danými názvy

- last_1_unigrams, last_1_bigrams, last_1_trigrams – četnosti jednoho posledního znaku po sobě následujících slov,
- last_2_unigrams, last_2_bigrams, last_2_trigrams – četnosti dvou posledních znaků po sobě následujících slov,
- last_3_unigrams, last_3_bigrams, last_3_trigrams – četnosti tří posledních znaků po sobě následujících slov.

Tyto názvy souborů musí zůstat nezměněny, aby správně fungovala následující fáze programu.

Analýza rozpoznaných vět

Tato fáze pracuje s parametry:

- `vety-adresar` – adresář, z nějž se mají číst rozpoznané věty, tedy pravděpodobně stejný adresář, do kterého byly v první fázi uloženy věty vybrané z FSM souborů
  data/01-rozpozn-vety/recog-bestpath
- `spravnost-slov` – soubor, do nějž byly v první fázi uloženy informace o správnostech rozpoznání všech slov ve větách z FSM souborů; využijí se pouze značky 0 a 1, ke značce D neexistuje slovo v rozpoznané větě
  data/01-rozpozn-vety/correctness.map
- `unigramy-slova` – soubor s četnostmi samostatných slov
  data/02-ngramy-znaky/unigrams
- `bigramy-slova` – soubor s četnostmi dvojic slov
  data/02-ngramy-znaky/bigrams
- `trigramy-slova` – soubor s četnostmi trojic slov
  data/02-ngramy-znaky/trigrams
- `ngramy-znaky-adresar` – adresář, v němž jsou z předchozí fáze uloženy soubory s četnostmi posledních znaků slov
  data/02-ngramy-znaky
- `analyzovana-data` – jméno výstupního souboru, do nějž se mají uložit věty s atributy slov
  data/03-analyza-dat/data-with-features

Program zde počítá atributy slov uvedené v tabulce 5.1. K výpočtu pravděpodobností v jazykovém modelu slov využívá soubory `unigramy-slova`, `bigramy-slova` a `trigramy-slova`, podle dat v `ngramy-znaky-adresar` počítá pravděpodobnost slova v modelech posledních znaků slov. Rozpoznané věty se chápou jako oddělené celky, u prvního slova věty se tedy nepočítá s trigramem, který by zasahoval do předchozí věty, ale pouze s četností unigramu.

Výsledný soubor s vypočtenými atributy má podobu CSV – *comma-separated values*, údaje každého slova jsou na samostatném řádku oddělené čárkami.

Rozdělení dat na trénovací a testovací

V této fázi jsou data rozdělena na trénovací a testovací v poměru `pomer-tren-test`. Data se dělí náhodně po celých větách. Program požaduje atributy:

- `analyzovana-data` – datový soubor s atributy slov, který se má rozdělit na trénovací a testovací data
  data/03-analyza-dat/data-with-features

- `pomer-tren-test` – požadovaný poměr velikostí rozdělených dat, uvádí se `X:Y` a trénovací data potom budou tvořena částí o velikosti $\frac{X}{X+Y}$ vět a testovací data $\frac{Y}{X+Y}$ vět
- `trenovaci-data` – soubor pro uložení oddělených trénovacích dat, bude obsahovat všechny atributy ze souboru `analyzovana-data`
  data/04-rozdeleni-dat/train-data
- `testovaci-data` – soubor pro uložení testovacích dat, bude obsahovat všechny atributy kromě příznaku správnosti `correctness`
  data/04-rozdeleni-dat/test-data
- `test-data-kontrolni` – soubor pro uložení informace o správnosti slov v testovacích datech, tato informace se použije při vyhodnocování úspěšnosti detektoru
  data/04-rozdeleni-dat/test-data-corr

Spuštění strojového učení

V této části programu se spouští trénování detektoru chyb založeného na jedné z metod strojového učení a poté se rovnou provede predikce skóre spolehlivosti na testovacích datech.

Pokud je zvolena metoda logistické regrese, vytváří se model, který využívá všech dostupných atributů slov a v kapitole 5.3.1 je uveden pod názvem `glm.full`. Při volbě neuronové sítě bude natrénována síť `nnet.CVBest` s parametry zvolenými v kapitole 5.3.2 pomocí cross-validace (`formula1`, `size = 7`, `decay = 0`). A konečně z rozhodovacích stromů je zde použit strom s 1-SE prořezáváním – `rpart.pruned1SE` z kapitoly 5.3.3.

- `trenovaci-data` – soubor s trénovacími daty pro metodu strojového učení
  data/04-rozdeleni-dat/train-data
- `testovaci-data` – soubor s daty určenými k otestování natrénovaného detektoru
  data/04-rozdeleni-dat/test-data
- `test-data-kontrolni` – soubor se správností slov v testovacích datech, tato informace se přidá do souboru s predikovaným skóre spolehlivosti a ten se pak může předat fázi vyhodnocení výsledků
  data/04-rozdeleni-dat/test-data-corr
- `metoda` – jeden z následujících řetězců: `logisticka-regrese`, `neuronova-sit`, `rozhodovaci-strom`
- `detekce-vystup` – soubor pro uložení skóre spolehlivosti a skutečné správnosti testovacích dat
  data/05-stroj-uceni/confidence-glm, confidence-nnet, confidence-rpart

- `ulozit-rdata` – příznak `ano/ne`, zda má R uložit na konci relace všechny proměnné (včetně množin dat a natrénovaných modelů) do souboru .Rdata v aktuálním adresáři. Pokud je pak R spuštěno z tohoto adresáře, data se ze souboru automaticky načtou a je možno s nimi standardně pracovat, např. si prohlédnout natrénované modely. Pokud však uložení dat není požadováno, doporučuje se tento příznak nenastavovat, neboť ukládání je poměrně časově náročné.

Ve výsledném souboru `detekce-vystup` jsou na každém řádku tyto údaje oddělené mezerou: pořadí věty, pořadí slova ve větě, slovo (v kódování ISO 8859-2), správnost rozpoznání slova (0 nebo 1) a skóre spolehlivosti.

Vyhodnocení výsledků

Posledním krokem testování detekce je vyhodnocení její úspěšnosti. Příkaz, kterým se spouští, má tvar

```
detekce --vysledky [--nahodne] [--parametry SOUBOR-S-PARAMETRY]
```

Uvede-li se volba `--nahodne`, program pouze vygeneruje náhodné hodnoty „skóre spolehlivosti“ a vyhodnotí jeho úspěšnost. Pokud je uveden soubor s parametry a v něm parametr `graf-roc`, uloží se graf ROC do tohoto souboru, jinak se uloží do souboru ROC-nahoda.png. Pokud není zadáno `--nahodne`, je volba `--parametry` povinná.

- `detekce1`, `detekce2`, `detekce3` – soubory s výstupem detekce chyb z předchozí fáze (`detekce-vystup`), všechny jsou nepovinné a pokud se zadávají méně než tři soubory, musí to být vždy v pořadí `detekce1`, `detekce2`, `detekce3`
- `graf-roc` – název souboru pro uložení obrázku s grafem ROC ve formátu PNG data/06-vysledky/ROC-plot.png
- `pocitat-roc-cen` – příznak `ano/ne`, zda se má do grafu ROC přikreslit též křivka účinnosti cen na hranách v lattice jakožto detekce chyb. Pokud je tento příznak nastaven na `ano`, musí být v souboru parametrů uvedeny rovněž parametry `vety-adresar` a `spravnost-slov`, které jsou potřeba pro nalezení příslušných dat – cen v lattice a správnosti rozpoznání slov.
- `vety-adresar` – stejný význam jako v první fázi data/01-rozpozn-vety/recog-bestpath
- `spravnost-slov` – stejný význam jako v první fázi data/01-rozpozn-vety/correctness.map

Pokud není nastavena volba `--nahodne`, je nutno zadat buď parametr `detekce1` (a příp. `detekce2`, `detekce3`), nebo parametr `pocitat-roc-cen=ano` spolu s parametry `vety-adresar` a `spravnost-slov`. Mohou být nastaveny všechny tyto parametry a v grafu ROC tak mohou být až čtyři křivky.

Graf je kreslen pomocí grafických prostředků programu R. Neboť u obrázku tohoto typu se velice špatně mění velikost, vytváří se najednou alespoň dva obrázky různé velikosti. Větší obrázek v rozlišení 465×480 bodů se ukládá do souboru `graf-roc` a menší obrázek v rozlišení 315×325 bodů do souboru podobného jména se sufixem „-mensi“.

Kromě uložení grafu ROC vypíše program na obrazovku směrodatnou chybu detekce, možnosti zmenšení WER a hodnotu prahu, při které je toto zmenšení nejvýraznější. Dosažené výsledky se budou mírně lišit při každém novém dělení trénovacích a testovacích dat a mohou se tedy nepatrně lišit od výsledků prezentovaných v této práci.

Příklady

Soubor parametrů obsahující všechny potřebné informace pro všechny fáze programu může vypadat například takto:

```
fsm-adresar=data/01-rozpozn-vety/recog
slovnik=data/01-rozpozn-vety/wordmap
master-label-file=data/01-rozpozn-vety/labels.mlf
vety-adresar=data/01-rozpozn-vety/recog-bestpath
spravnost-slov=data/01-rozpozn-vety/correctness.map
unigramy-slova=data/02-ngramy-znaky/unigrams
bigramy-slova=data/02-ngramy-znaky/bigrams
trigramy-slova=data/02-ngramy-znaky/trigrams
ngramy-znaky-adresar=data/02-ngramy-znaky/
pocitat-unigramy=a
pocitat-bigramy=a
pocitat-trigramy=a
analyzovana-data=data/03-analyza-dat/data-with-features
pomer-tren-test=9:1
trenovaci-data=data/04-rozdeleni-dat/train-data
testovaci-data=data/04-rozdeleni-dat/test-data
test-data-kontrolni=data/04-rozdeleni-dat/test-data-corr
metoda=rozhodovaci-strom
detekce-vystup=data/05-stroj-uceni/confidence-rpart
ulozit-rdata=a
detekce1=data/05-stroj-uceni/confidence-glm
detekce2=data/05-stroj-uceni/confidence-nnet
detekce3=data/05-stroj-uceni/confidence-rpart
graf-roc=data/06-vysledky/ROC-plot.png
pocitat-roc-cen=a
```

Příklady spuštění programu:

```
detekce --rozpozn-vety --parametry params.txt
detekce --rozpozn-vety --analyza-dat --parametry params.txt
detekce --parametry params.txt
```

6.2 Dokumentace

Program byl vyvíjen v prostředí NetBeans IDE 6.0.1. Projekt, který je možné otevřít v tomto IDE, je umístěn v adresáři zdroje-NetBeans.

6.2.1 Zdrojové kódy v C++

Soubory se zdrojovým kódem v C++ jsou uloženy v adresáři zdrojove-kody/src a podle účelu rozděleny do čtyř adresářů:

- MainProgram
  - common.cpp, common.h
  - main.cpp
- DataHandling
  - latticeExtract.cpp, latticeExtract.h – třída `LatticeExtract`
  - dataFeatures.cpp, dataFeatures.h – třída `DataFeatures`
  - resultsEval.cpp, resultsEval.h – třída `ResultsEval`
- LanguageModel
  - charNgrams.cpp, charNgrams.h – třída `CharNgrams`
  - ngramsStruct.cpp, ngramsStruct.h – třída `NgramsStruct`
  - orderCheck.cpp, orderCheck.h – třída `OrderCheck`
  - process.cpp, process.h – třída `Process`
  - model.cpp, model.h – třída `Model`
  - searchCount.cpp, searchCount.h – třída `SearchCount`
- MachineLearn
  - rMachineLearn.cpp, rMachineLearn.h – třída `RMachineLearn`

Hlavní program

Kód v souboru main.cpp zajišťuje spuštění programu a volání funkčností podle toho, jaká fáze byla určena volbou na příkazové řádce. Na začátku každé fáze je nejprve nutno načíst údaje ze souboru parametrů. Pro každou fázi je dán seznam parametrů s jejich vlastnostmi, který se předá obecné funkci pro získávání parametrů ze souboru a ta načte požadované hodnoty.

V souborech common.h a common.cpp jsou soustředěny deklarace a definice společných konstant a jednoduchých funkcí používaných na více místech. Jsou zde mimo jiné uloženy řetězcové konstanty pro názvy parametrů načítaných ze souboru.

Extrakce rozpoznaných vět

Třída `LatticeExtract` slouží ke zpracování lattice v binárních souborech *.fsm. Program nejprve projde zadaný adresář a vytvoří seznam jmen všech souborů s pří-

ponou fsm nebo FSM. Potom postupně na každý soubor spustí posloupnost příkazů knihovny FSM

```
fsmbestpath fsm-adresar/soubor.fsm | fsmproject -o | fsmrmepsilon | \
  fsmprint -o slovnik > vety-adresar/temp/soubor.txt
```

pomocí funkce `system()` a zkontroluje, zda byl vytvořen soubor soubor.txt nenulové velikosti. Zajistí se tak, že když dojde při provádění příkazu k chybě (např. není nainstalována knihovna FSM), program skončí, místo aby se pokoušel zpracovávat další soubory.

Příkaz `fsmbestpath` vyhledá v lattice cestu s nejnižším součtem cen. Výsledek, který je stále v binárním formátu FSM, se předá jako vstup příkazu `fsmproject`. Ten udělá z původního automatu, který byl typu převodník (*transducer*), klasický akceptor (*acceptor*). Uvedením volby `-o` se určuje, že se z převodníku mají převzít výstupní symboly přechodů, alternativní možností je `-i` pro ponechání vstupních symbolů. Příkaz `fsmrmepsilon` vrací ekvivalentní automat bez přechodů s nulovým výstupem a konečně `fsmprint` převede výsledný automat do textové podoby. Díky volbě `-o slovnik` budou výstupními symboly slova, a ne čísla. Výsledek vypadá např. takto:

```
0 1 DŮVODEM 4409.45215
1 2 BYLY  1614.42188
2 3 ZMATKY  3120.54688
3 4 KOLEM 1380.28809
4 5 HLASOVÁNÍ 2384.79492
5 3525.68555
```

Každý řádek vyjadřuje jednu hranu nejlepší cesty. V prvním sloupečku je číslo počátečního stavu, v druhém číslo koncového stavu, třetí údaj je výstupní symbol na hraně a poslední je cena hrany. Výjimkou je pouze poslední řádek, na kterém je uvedeno číslo koncového stavu a cena toho, že automat skončí v tomto stavu.

Program tedy vybere třetí sloupeček, který bude tvořit zápis rozpoznané věty, a uloží jej do souboru `vety-adresar/soubor.sntnc`. Poslední sloupeček s cenami hran uloží do `vety-adresar/costs/soubor.cst`. Následně se provede zarovnání věty s referenční větou v souboru `labels.mlf` (*master label file - MLF*) příkazem

```
HResults -t -I labels.mlf list.txt vety-adresar/soubor.sntnc \
  > vety-adresar/soubor.align
```

Volba `-t` říká, že `HResults` má vypsat zarovnání vět, pokud se věty liší (bez `-t` se zarovnání nevypisuje vůbec). Volba `-I labels.mlf` zajišťuje načtení souboru MLF. Výstup v souboru `vety-adresar/soubor.align` pak vypadá podobně jako tento:

```
Aligned transcription: 040712rn0338882.lab vs 040712rn0338882.sntnc
 LAB: OHLASY SHRNUJE NAŠE BRATISLAVSKÁ ZPRAVODAJKA RENATA HAVRANOVÁ
 REC: OHLASY SHRNUJE NAŠE BRATISLAVSKÁ ZPRAVODAJKA        VRÁNOVÁ
====================== HTK Results Analysis =======================
```

```
  Date: Wed Feb 20 17:51:32 2008
  Ref : data/01-rozpozn-vety/labels.mlf
  Rec : ./recog-bestpath/040712rn0338882.sntnc
------------------------ Overall Results --------------------------
SENT: \%Correct=0.00 [H=0, S=1, N=1]
WORD: \%Corr=71.43, Acc=71.43 [H=5, D=1, S=1, I=0, N=7]
================================================================
```

Program poté projde `soubor.align` a zjistí správnost jednotlivých slov. Pro každý FSM soubor zapíše jeden řádek na konec souboru, jehož jméno bylo zadáno parametrem `spravnost-slov`:

```
id sentencefile correctness
1 040712rn0329725 00111111111
2 040712rn0335854 1111111
3 040712rn0338882 11111D0
...
```

Jazykový model

Jako první přichází ke slovu třída `CharNgrams`, která musí spočítat četnosti n-gramů posledních znaků slov dříve, než je spuštěna fáze analýzy rozpoznaných vět. Používá k tomu asociativní kontejner `map<string, int>` ze Standard Template Library. Nejprve vytváří unigramy posledních znaků slov. Postupně načítá jednotlivé řádky ze souboru unigramových četností slov, které mají tvar

`slovo␣c`,

kde `c` je četnost slova, a pro každý unigram určí tři různé klíče – jeden poslední znak, dva a tři poslední znaky (pokud je slovo kratší, jsou kratší i tyto klíče). Toto budou klíče do tří různých map a do každé se přidá prvek s hodnotou četnosti `c`. Pokud v mapě již prvek s tímto klíčem existuje, pouze se k dosavadní hodnotě přičte `c`. Výsledné mapy se uloží do souborů last_1_unigrams, last_2_unigrams, last_3_unigrams. Analogicky se spočítají četnosti bigramů a trigramů znaků pouze s tím rozdílem, že klíče jsou tvořeny zřetězením posledních znaků slov (oddělených mezerou).

Když se mají počítat trigramové pravděpodobnosti slov ve větách z rozpoznávače, začíná se načtením četností ze souborů do datových struktur v paměti. Z prvotní představy, že počítání pravděpodobností bude napsáno v Perlu (implementace by byla snadná), sešlo v okamžiku prvního pokusu o spuštění – program brzy zabral celou paměť a dokončení operace bylo v nedohlednu. Byly proto navrženy datové struktury v C, které alespoň částečně optimalizují paměťové nároky a vyhledávání. Tyto struktury jsou uloženy v souborech ngramsStruct.cpp, ngramsStruct.h, kde je též třída `NgramsStruct` s funkcemi zajišťujícími jejich naplnění a závěrečné uvolnění.

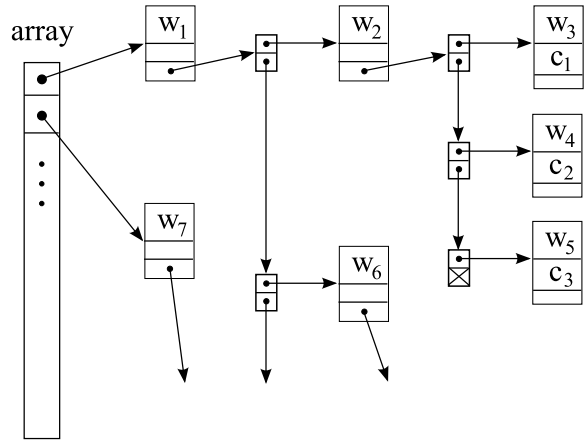
Způsob uložení unigramů se odlišuje od uložení bigramů a trigramů. Unigramy využívají obyčejné pole o velikosti stejné, jako je velikost slovníku. Každý prvek je ukazatelem na strukturu (`struct`), která obsahuje řetězec znaků a jeho unigramovou

četnost. Řetězce jsou lexikograficky setříděny a v poli se hledá binárním vyhledáváním (metodou půlení intervalu). Toto pole slouží rovněž jako slovník pro struktury bigramů a trigramů, které již nikde neukládají slovo jako řetězec, ale pouze jeho číselné id, kterým je pořadí v tomto poli unigramů. Každé slovo je tedy v paměti uloženo pouze jednou.

Trigramy jsou ukládány ve struktuře vyobrazené na obrázku 6.1. Struktura bigramů se liší pouze v tom, že je jednodušší („užší“). Ze struktury na obrázku lze vyčíst, že

$$\begin{aligned} c(w_1, w_2, w_3) &= c_1 \\ c(w_1, w_2, w_4) &= c_2 \\ c(w_1, w_2, w_5) &= c_3 \end{aligned}$$

a neznáme žádný další trigram začínající slovy w_1, w_2. Výhodou tohoto uložení je ušetřené místo za opakující se slova na prvních místech trigramů a navíc se omezí zbytečné porovnávání slov při vyhledávání. O hledání četnosti pro zadaný n-gram se stará třída `SearchCount`. Před načtením souboru s četnostmi n-gramů je nejprve vhodné zkontrolovat lexikografické uspořádání slov pomocí funkce `checkOrdering()` třídy `OrderCheck`.



Obrázek 6.1: Datová struktura pro trigramy

Třída `Model` obsahuje implementaci jazykového modelu, který používá vyhlazování *Modified Kneser-Ney Smoothing*, jak bylo popsáno v kapitole 5.1.3. Četnosti získává z právě popsaných struktur. Parametry vyhlazování $D_{1,1}, \ldots, D_{3,3+}$ jsou spočteny při načítání četností ze souborů do datových struktur třídou `NgramsStruct`. Jakmile máme sestaven mechanismus vyhledávání četností a počítání „vyhlazené“ pravděpodobnosti, lze naprosto totožným způsobem pracovat s n-gramy celých slov i s n-gramy posledních znaků.

Třída `Process` má za úkol přidat pravděpodobnost ke každému slovu v datovém souboru při spuštění analýzy dat. Dostane datovou strukturu n-gramů a nezáleží jí na tom, o který ze čtyř modelů se jedná.

Analýza rozpoznaných vět

Získávání atributů rozpoznaných slov obstarává třída `DataFeatures`. Nejprve volá perlovský skript, který počítá atributy slov mimo pravděpodobností v trigramových modelech. Ten vytvoří první verzi CSV souboru dat. Následně program využije funkčnost jazykového modelu, popsaného výše, a do datového souboru přidá pravděpodobnost slova a pravděpodobnosti jednoho, dvou a tří posledních znaků slova. Třída `DataFeatures` též spouští dělení dat na trénovací a testovací, které zajišťuje kód v divide_data.R.

Strojové učení

Třída `RMachineLearn` spouští trénování a testování metody strojového učení, které je podle typu metody v souboru glm.R, nnet.R nebo rpart.R. `RMachineLearn` zajistí předání trénovacích, testovacích a kontrolních testovacích dat programu v R a zkontroluje, zda byl vytvořen výstupní soubor s predikovaným skóre spolehlivosti.

Použité funkce R již byly popsány v kapitole 5.3. Vytváří se buď model `glm.full`, nebo neuronová síť `nnet.CVBest`, nebo rozhodovací strom `rpart.pruned1SE`.

Vyhodnocení výsledků

Třída `ResultsEval` podle zadání uživatele načítá skóre spolehlivosti buď z výstupu detektoru, nebo ze souborů s cenami (převede je na $1 -$ `cost/maxCost`, aby hodnoty byly mezi 0 a 1), nebo generuje náhodné skóre. Spočítá body křivky ROC, uloží je do souboru a zavolá vykreslení grafu v roc_plotting.R. Kromě toho ještě spočítá směrodatnou chybu detekce a možnost redukce WER.

6.2.2 Ostatní zdrojové kódy

Další zdrojové kódy, které již nejsou v jazyce C++, jsou obsahem adresářů zdrojove-kody/R_code a zdrojove-kody/perl_code.

- R_code
  - load_data.R – načítání trénovacích a testovacích dat ve tvaru CSV ze souborů do datových struktur R
  - chars_merging.R – vytváření kategorií znaků (samohlásky, souhlásky), skupin znaků a slévání kategorií, které jsou málo zastoupeny v datech
  - divide_data.R – rozdělení dat na trénovací a testovací
  - glm.R – trénování a testování modelu logistické regrese
  - nnet.R – trénování a testování neuronové sítě
  - rpart.R – trénování a testování rozhodovacího stromu
  - roc_plotting.R – vytváření grafu s křivkami ROC
- perl_code
  - prepare_data_forR.pl – počítání atributů slov kromě pravděpodobností v trigramových modelech

Kapitola 7

Závěr

Cílem této práce bylo navrhnout metodu detekce chyb v rozpoznávání řeči. Při úvahách nad tím, jaké informace máme v této úloze k dispozici, vznikl návrh zabývat se především těmi informacemi, které jsou dostupné prakticky vždy – samotnými rozpoznanými slovy. Pokud některé z vlastností těchto slov mají nějaký vztah k úspěšnosti rozpoznávání, mohou posloužit jako míra spolehlivosti rozpoznání nebo jako doplňující informace k jiné metodě detekce chyb.

K odhalování vztahů tohoto typu se dobře hodí metody strojového učení. V tomto případě dosáhly nejlepších výsledků rozhodovací stromy. Bylo zjištěno, že nejsilněji s úspěšností rozpoznávání souvisí:

Pravděpodobnost slova v jazykovém modelu – rozhodovací strom ukazuje, že pokud je tato pravděpodobnost rovna alespoň $5{,}4255 \cdot 10^{-7}$, je pravděpodobnost správného rozpoznání rovna přibližně 0,86, jinak je to pouze 0,29 (průměrná hodnota příznaků 0 a 1 na konkrétních trénovacích datech). Přestože rozpoznávač řeči využívá svůj jazykový model, nevydává vždy na výstup věty s vysokou pravděpodobností všech slov v tomto modelu. Může se stát, že skóre hypotézy je výrazně zvýšeno pravděpodobností v akustickém modelu, a tak „přebije“ ostatní hypotézy, nebo ostatní slova ve větě mají dostatečně vysokou pravděpodobnost v jazykovém modelu, nebo jednoduše není k dispozici hypotéza s vyšší pravděpodobností v jazykovém modelu.

Délka slova – pokud je pravděpodobnost slova v jazykovém modelu menší než $5{,}4255 \cdot 10^{-7}$ a zároveň je slovo kratší než 7 znaků, je pravděpodobnost správného rozpoznání dokonce jen 0,098. Pro delší slova vychází 0,43. Na horší úspěšnosti u kratších slov se nejspíše podílejí zejména předložky a spojky, jejichž rozpoznávání je problematické.

Pravděpodobnost slova v modelu posledních tří znaků – model posledních znaků slov byl navržen jako doplňkový k jazykovému modelu celých slov. Model celých slov trpí problémem nedostatečných dat obzvláště v češtině, která díky skloňování a časování disponuje obrovským množstvím různých slovních tvarů. Slovní koncovky obsahují nezanedbatelnou informaci díky pravidlům ohýbání

slov a jejich počet je mnohem menší. Skupiny posledních znaků pevné velikosti jsou určitou aproximací a přitom je lze snadno odvodit bez nutnosti jazykového rozboru.

Poslední tři znaky slova (třída podle typu hlásek) – vedle pravděpodobnostního modelu koncovek lze využít koncovky jako takové. Jejich počet je však přeci jen příliš velký pro strojové učení na datech omezené velikosti, proto byly sloučeny do tříd podle toho, jestli jednotlivé hlásky patří mezi samohlásky krátké, dlouhé, souhlásky měkké atd.

Poslední znak slova – nejkratší možná koncovka. V jejím případě mnoho různých možností není, a tak mohla být použita přímo. Byly pouze sloučeny znaky, které se v datech vyskytují opravdu málo, nejčastěji kvůli použití cizího slova (w, x), nebo omylem (ú).

Výhodou detekce chyb, která využívá tyto vlastnosti slov, je nezávislost na konkrétním rozpoznávači mluvené řeči. Všechny atributy lze spočítat z textu, který je výstupem rozpoznávače. Metoda nevyžaduje ani žádnou znalost nahrávky řeči, jako je tomu např. v případě analýzy prozodických rysů.

Pro praktické využití je důležitá také časová náročnost detekce. Trénování metod strojového učení je poměrně zdlouhavé, provede se však pouze jednou. Počítání skóre spolehlivosti je pak hotové takřka okamžitě (na testovacích datech s 3000 příkladů).

Metoda detekce byla vyvíjena pro češtinu a testována na nahrávkách z českého rozhlasu a televize, ale podobně může fungovat i při rozpoznávání jiných jazyků. Mohou být navrženy další vlastnosti, které lépe vystihují slova daného jazyka, a metody strojového učení určí nové vztahy k úspěšnosti rozpoznávání. U inflektivních jazyků pravděpodobně nebudou mít stejný význam skupiny posledních znaků slov.

Další práce

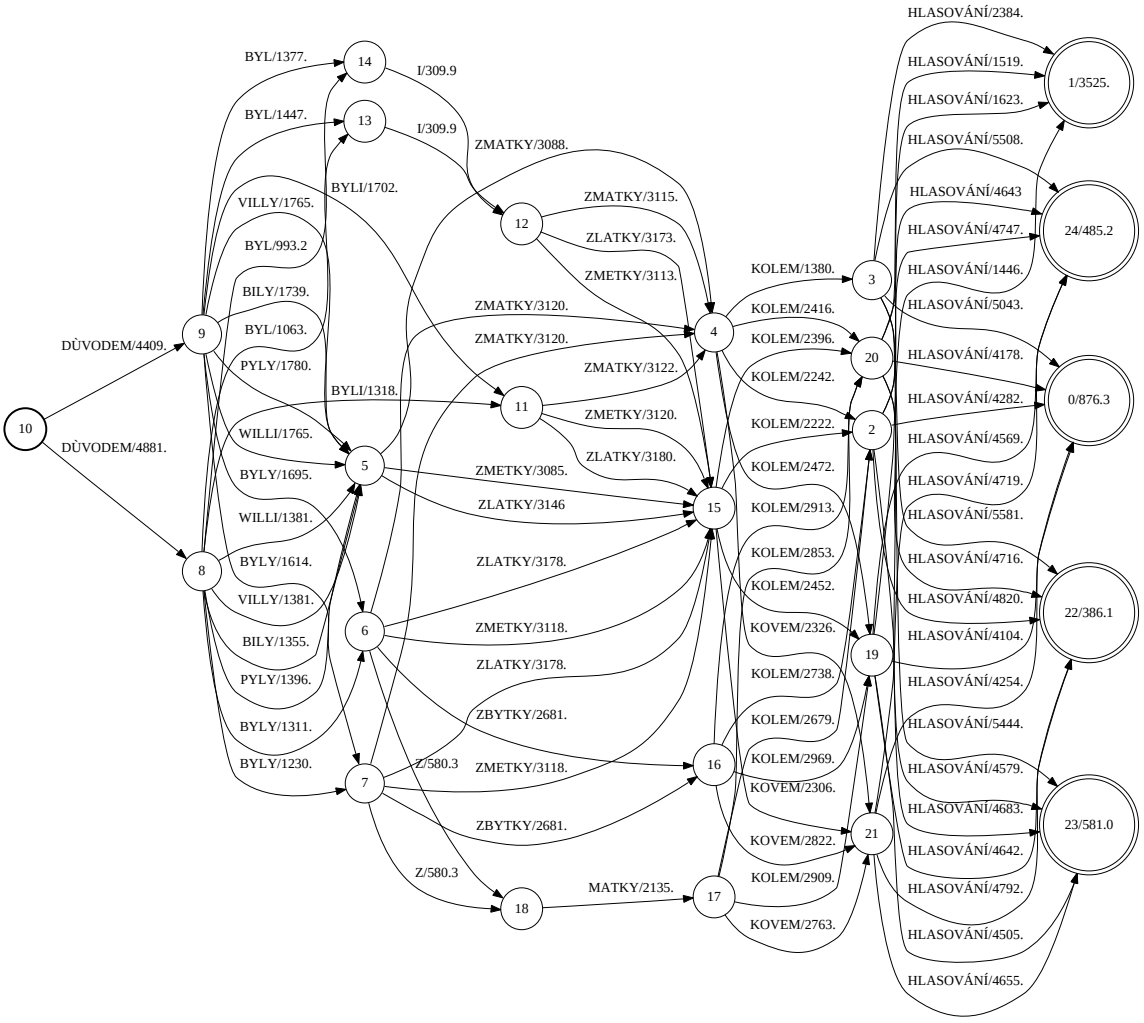
V dalším vývoji navržené detekce by bylo vhodné provést důkladnější porovnání úspěšnosti s jinými metodami. V případě výpočtu skóre spolehlivosti však není nutné použít vždy jen jedinou metodu a místo toho, aby metody mezi sebou soutěžily, mohou se doplňovat. U této metody je to navíc poměrně žádoucí vzhledem k tomu, že není schopna odhalit chyby vzniklé vynecháním slova.

Několikrát zde bylo zmíněno omezení způsobené velikostí trénovacích dat. Větší množství dat by bylo možné získat tak, že by se ze struktury ve výstupu rozpoznávače nevybírala pouze jedna nejlepší věta, ale vzalo by se vět více. Je však otázkou, jestli by při trénování vykazovaly statisticky stejné vlastnosti jako jedna nejlepší věta při následné detekci.

Další prostor pro práci na této úloze představuje množina atributů slova. Jistě bude možné navrhnout další atributy, případně modifikovat atributy stávající (vyzkoušet jiný způsob počítání pravděpodobností slov, jiné kategorie posledních znaků slov apod.). Existují rovněž různé další metody strojového učení, které by se navíc mohly vzájemně kombinovat.

Příloha A

Příklad lattice



Obrázek A.1: Lattice pro větu „Důvodem byly zmatky kolem hlasování.“

Příloha B

Směrodatné chyby

	Směrodatná chyba	
Jméno modelu	Rozdělení dat 9:1	Rozdělení dat 2:1
Model logistické regrese		
glm.costOnly	0,4038162	0,4022905
glm.wordsFeats	0,3945414	0,3917811
glm.noMerged	0,3906355	0,3893381
glm.full	0,3842639	0,3826044
Neuronové sítě		
nnet.CVBest, formula1, size=7, decay=0	0,3546269	0,3602966
nnet.CVBest, formula1, size=7, decay=0,003	0,3797321	0,3844034
nnet.CVBest, formula2, size=7, decay=0	0,3992210	0,3774055
nnet.CVBest, formula2, size=7, decay=0,1	0,3750960	0,3726260
nnet.big	0,3792433	0,3737545
nnet.tune, formula1, size=6, decay=0,1	0,3906002	0,3886119
nnet.tune, formula1, size=7, decay=0,1	0,3919231	0,3878779
nnet.tune, formula2, size=6, decay=0	0,3847366	0,4524723
nnet.tune, formula2, size=6, decay=0,1	0,3814463	0,3762669
nnet.tune, formula2, size=7, decay=0	0,3849066	0,4524723
nnet.tune, formula2, size=7, decay=0,1	0,3817693	0,3801789
Rozhodovací stromy		
rpart bez prořezání	0,3829523	0,3712800
rpart – prořezání podle min. xerror	0,3501208	0,3430131
rpart – prořezání podle pravidla 1-SE	0,3489094	0,3472199
tune.rpart; tree	0,3596220	0,3537761
Kombinace metod		
combine.lm	0,3431458	0,3460264
combine.nnet	0,3430705	0,3461423
combine.rpart	0,3464880	0,3456243

Tabulka B.1: Směrodatné chyby všech modelů

		Směrodatná chyba			
		Rozdělení dat 9:1		Rozdělení dat 2:1	
`size`	`decay`	`formula1`	`formula2`	`formula1`	`formula2`
1	0	0,3992591	0,3989510	0,3986795	0,3972188
1	0,003	0,3952270	0,3861626	0,3972557	0,3863087
1	0,01	0,3930331	0,3858713	0,3910094	0,3855318
1	0,1	0,3946311	0,3847179	0,3900924	0,3853996
2	0	0,3942897	0,3957776	0,3940694	0,3932557
2	0,003	0,3888725	0,3800491	0,3887293	0,3824246
2	0,01	0,3861374	0,3785330	0,3894614	0,3804438
2	0,1	0,3867332	0,3750189	0,3873181	0,3742448
3	0	0,3866555	0,3891117	0,3771240	0,3753019
3	0,003	0,3840883	0,3724540	0,3848383	0,3845447
3	0,01	0,3831843	0,3743533	0,3856321	0,3722882
3	0,1	0,3855697	0,3722042	0,3852010	0,3714369
4	0	0,3771355	0,3830515	0,3756375	0,3902209
4	0,003	0,3838432	0,3729747	0,3836509	0,3741729
4	0,01	0,3830632	0,3689693	0,3825034	0,3693108
4	0,1	0,3845809	0,3687757	0,3841202	0,3793452
5	0	0,3775959	0,3853864	0,3723405	0,3687393
5	0,003	0,3823323	0,3665884	0,3815191	0,3730920
5	0,01	0,3829694	0,3735032	0,3821443	0,3703437
5	0,1	0,3846293	0,3667745	0,3841612	0,3672072
6	0	0,3807787	0,3799610	0,3688091	0,3791957
6	0,003	0,3695825	0,3715130	0,3804491	0,3724366
6	0,01	0,3816193	0,3672694	0,3816414	0,3673207
6	0,1	0,3836046	0,3662517	0,3836907	0,3673579
7	0	**0,3572550**	0,3811716	**0,3608342**	0,3776583
7	0,003	0,3803602	0,3702935	0,3799813	0,3721130
7	0,01	0,3815065	0,3659722	0,3811268	0,3674796
7	0,1	0,3834478	**0,3640548**	0,3835684	**0,3660084**

Tabulka B.2: Neuronové sítě – směrodatná chyba pro různé kombinace formule, `size` a `decay` při cross-validaci

Literatura

[1] Chen, S. F., Goodman, J. T.: *An Empirical Study of Smoothing Techniques for Language Modeling*, Technical Report TR-10-98, Computer Science Group, Harvard University, 1998.

[2] Huang, Xuedong, Acero, A., Hon, Hsiao-Wuen: *Spoken Language Processing: A Guide to Theory, Algorithm, and System Development*, Prentice Hall, 2001.

[3] Mitchell, T. M.: *Machine Learning*, McGraw-Hill, 1997.

[4] Mohri, M., Pereira, F. C. N., Riley, M.: *Weighted Finite-State Transducers in Speech Recognition*, Computer Speech and Language, 16(1):69–88, 2002.

[5] Psutka, J.: *Komunikace s počítačem mluvenou řečí*, Academia, Praha, 1995.

[6] Psutka, J., Radová, V., Müller, L., Matoušek, J., Ircing, P., Graff, D.: *Large Broadcast News and Read Speech Corpora of Spoken Czech*, in Proceedings of EUROSPEECH, Aalborg, Denmark, 2001. pp. 2067–2070.

[7] Radová, V., Psutka, J., Müller, L., Psutka, J., Ircing, P., Matoušek, J., Byrne, W.: *Czech Broadcast News Speech and Transcripts*, Linguistic Data Consortium, CD-ROM LDC2004S01, LDC2004T01, Philadelphia, PA, USA, 2004.

[8] Sarma, A., Palmer, D., D.: *Context-based Speech Recognition Error Detection and Correction*, in Proceedings of the HLT-NAACL 2004, pp. 85–88, 2004.

[9] Venables, W. N., Ripley, B. D.: *Modern Applied Statistics with S*, Fourth edition, Springer-Verlag New York, Inc, 2002.

[10] Voll, K. D.: *A Methodology of Error Detection: Improving Speech Recognition in Radiology*, PhD thesis, Simon Fraser University, 2006.

[11] Wessel, F., Schlüter, R., Macherey K., Ney, H.: *Confidence Measures for Large Vocabulary Continuous Speech Recognition*, IEEE Transactions on Speech and Audio Processing, Vol. 9, No. 3, March 2001.

[12] Young, S. a kol.: *The HTK Book* (for HTK Version 3.3) Manual, Cambridge University Engineering Department, Cambridge, 2005.