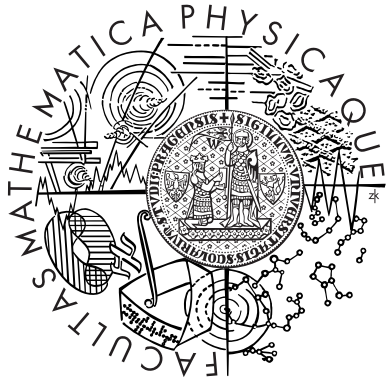
Univerzita Karlova v Praze
Matematicko–fyzikální fakulta

Diplomová práce



Miroslav Spousta

Automatické přiřazení tvaroslovných vzorů v češtině

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

Děkuji vedoucímu diplomové práce doc. Janu Hajičovi za trpělivé vedení práce, motivaci a velmi podnětné připomínky, jakož i za to, že mi byl vzorem, nikoli však morfologickým. Také děkuji RNDr. Jarce Hlaváčové za poskytnutí cenných rad a materiálů týkajících se formátu slovníku a dalších souborů, Mgr. Johance za morální podporu a teplé večeře a Mgr. PerMovi za připomínky nejen k obsahu práce.

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

V Praze dne 11. 8. 2005 Miroslav Spousta

Obsah

Název práce: Automatické přiřazení tvaroslovných vzorů v češtině

Autor: Miroslav Spousta

Katedra (ústav): Ústav formální a aplikované lingvistiky

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

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

Abstrakt: Cílem předložené práce je vytvořit metody pro automatické přiřazování morfologických vzorů českým slovům. Nejprve je provedena analýza problému, ve které jsou zdůrazněny některé podproblémy, se kterými se musíme vypořádat. Poté jsou navrženy čtyři různé algoritmy pro výběr z možných vzorů, pracující na základě analýzy slova a jeho kontextu. Dále jsme navrhli algoritmus pro rozdělení množiny slov na třídy ekvivalence podle společného lemmatu. Pro odhad optimálních parametrů jednotlivých metod jsme použili různé zdroje dat, na kterých jsme provedli přes 250 testů s různými hodnotami parametrů. Součástí práce je popis použitých algoritmů a jejich implementace v programovacích jazycích Perl a C++.

Klíčová slova: morfologická analýza, vzor, kořen, lemma

Title: Automatic paradigm assignment for czech words

Author: Miroslav Spousta

Department: Institute of Formal and Applied Linguistics

Supervisor: Doc. RNDr. Jan Hajič, Dr.

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

Abstract: Aim of the presented work is to explore possibility of automatic morphological paradigms assignment for the Czech words. Theoretical part of our work consists of the problem analysis with emphasized issues we have to deal with. We present four different algorithms for morphological paradigm assignment, using both word form analysis and contextual information processing. Word forms are partitioned into equivalence classes according to their lemma, using another algorithm. We performed more than 250 tests on the various corpus data with the purpose of estimating best method parameters. Presented algorithms are thoroughly described and implemented.

Keywords: morphological analysis, paradigm, root, lemma

Úvod

Tématem této práce je automatické přiřazování morfologických vzorů českým slovům. Naším cílem bude vyvinout metody, které se na základě zadaného slova a jeho kontextu pokusí najít správný vzor, podle něhož se slovo ohýbá, a najít k němu základní tvar (lemma). Práce je rozčleněna do čtyř kapitol.

První kapitola je motivační, zabývá se českou morfologií a zavádí pojmy a označení, které budeme dále používat. Je zde popsáno, jak funguje morfologický analyzátor, co jsou derivace slov a jakým způsobem je uložen slovník a další soubory. Závěrem kapitoly shrneme dosavadní přístupy k podobným problémům.

Druhá kapitola se zabývá řešením problému automatického přiřazení vzoru ke slovům, vysvětluje použité metody a algoritmy včetně jednoduché analýzy jejich složitosti. Najdeme zde také rozbor několika problémů, na které jsme narazili během naší práce a se kterými se musíme vypořádat.

Ve třetí kapitole vysvětlíme, jak dané metody v praxi použít, popíšeme procesy trénování, vyhlazování a testování. Dále ukážeme výsledky, kterých jsme dosáhli na testovacích datech, a pokusíme se najít optimální nastavení parametrů.

Závěrečná čtvrtá kapitola popisuje implementaci popsaných algoritmů. Zdůvodníme volbu programovacích prostředků a popíšeme ovládání jednotlivých programů a skriptů.

Na závěr naší práce zhodnotíme dosažené výsledky a ukážeme možnosti dalšího zlepšení.

Kapitola 1

Motivace

Už na základní škole jsme se naučili, že slovo kozel se skloňuje stejně jako slovo pán, ozdoba jako žena a slovo mečí se časuje podle prosí. Bylo nám vysvětleno, že slova českého jazyka tvoří skupiny, které se stejně ohýbají, a z každé skupiny jsme dostali *vzor*, podle něhož bylo snadné podobná slova skloňovat nebo časovat.

V oblasti zpracování přirozeného jazyka, počítačové lingvistice, před námi stojí problém podobný tomu ze základní školy: ke každému slovu jednoznačně určit jeho vzor. Chtěli bychom, aby počítač přiřazoval vzory pokud možno automaticky nebo jen s minimální pomocí uživatele.

Můžeme si položit otázku, proč je pro nás tento problém důležitý? Jedná se o úkol zajímavý teoreticky (jak moc dobře umíme automaticky přiřadit vzor), ale i prakticky. Na vzorech slov je založena automatická morfologická analýza českého jazyka a k jejímu rozšiřování a zdokonalování je potřeba doplňovat slovník o nová slova. K nim musí být přiřazen správný vzor, abychom věděli, které tvary daného slova jsou přípustné.

Morfologická analýza konkrétního jazyka tvoří úplný základ jeho lingvistického zpracování. Jejím úkolem je pro každé slovo určit morfologické kategorie (slovní druh, pád, číslo, čas, způsob...). Pro češtinu existuje několik implementací morfologické analýzy lišících se přístupem k zachycení slov, počtem vzorů a možnostmi, které použití vzorů nabízí. Kromě ,,pražské“ morfologie [1], kterou budeme při naší práci používat, se zde alespoň zmíníme o brněnské morfologii ajka [2]. V současné době na ní probíhají úpravy a není veřejně dostupná.

1.1 Morfologie češtiny

V této kapitole ve stručnosti popíšeme hlavní rysy české morfologie [1], soustředíme se především na ty, které v ostatních materiálech nenajdeme (popis interního formátu slovníku a derivační vzory).

Pro češtinu rozlišujeme tyto morfologické kategorie:

Slovní druh určuje základní rozdělení slov: podstatná jména (substantiva), přídavná jména (adjektiva), zájmena (pronomina), číslovky (numeralia), slovesa (verba), příslovce (adverbia), předložky (prepozice), spojky (konjunkce), částice (interjekce), citoslovce (partikule).

Rozšířený slovní druh jemněji člení některé slovní druhy do podkategorií (například předložky je možné dělit na obyčejné a vokalizované, zájmena na vztažná, neurčitá, osobní atd.).

Rod může ženský, mužský (životný a neživotný), střední.

Číslo singulár, plurál, duál.

Pád nominativ, genitiv, dativ, akuzativ, vokativ, lokál, instrumentál.

Přivlastňovací rod se určuje hlavně u přivlastňovacích zájmen a příslovcí.

Přivlastňovací číslo singulár nebo plurál.

Osoba může být první, druhá, nebo třetí.

Čas pro slovesa upřesňuje, kdy se děj odehrává (minulý, přítomný, budoucí)

Stupeň může být být první až třetí (např. úzký, užší, nejužší)

Negace říká, zda je slovo v negativní formě (s předponou ne-), nebo v afirmativu.

Slovesný rod určuje, zda je sloveso v aktivu, nebo v pasivu.

Varianta, styl slouží k rozlišení hovorových, knižních, archaických tvarů slov.

Pro různé slovní druhy se určují podmnožiny z těchto kategorií, např. pro podstatná jména určujeme slovní druh, rozšířený slovní druh, rod, číslo, pád a negaci (případně ještě variantu/styl).

Hodnoty morfologických kategorií se zapisují pomocí speciálních značek, *tagů*. Tag je skupina písmen a symbolů, které pro dané slovo jednoznačně určují hodnoty všech relevantních kategorií. V současnosti se používají dva systémy tagů: poziční a kompaktní.

1.1.1 Poziční systém tagů

Poziční tag se skládá z patnácti znaků, každá kategorie má určeno pevné číslo pozice, na kterém se vyskytují znaky reprezentující hodnoty dané kategorie. Přiřazení je následující:

1	slovní druh (POS)
2	detailní slovní druh (SUBPOS)
3	rod (GENDER)
4	číslo (NUMBER)
5	pád (CASE)
6	rod vlastníka (POSSGENDER)
7	číslo vlastníka (POSSNUMBER)
8	osoba (PERSON)
9	čas (TENSE)
10	stupeň (GRADE)
11	negace (NEGATION)
12	slovesný rod (VOICE)
13	rezervováno (RESERVE1)
14	rezervováno (RESERVE2)
15	varianta, styl (VAR)

V závorce jsou uvedeny zkratky používané pro dané kategorie. Poziční tag pro slovo `možnost` může vypadat takto:

```
NNFS4-----A----
```

nebo také:

```
NNFS1-----A----
```

V prvním případě je dané slovo substantivum (N) obyčejné (druhé N), ženského rodu (F), v singuláru (S), v akuzativu (4), afirmativní (A). Ve druhém je to stejné slovo, ale v nominativu. Kategorie, které pro daný slovní druh nemají smysl (např. osoba u substantiv), jsou ohodnoceny znakem `-`, neznámé hodnoty mají hodnotu `X`. Kompletní popis možných hodnot pro jednotlivé kategorie je možné nalézt v popisu morfologie [3].

1.1.2 Kompaktní systém tagů

Kompaktní systém zápisu tagů nemá pevně danou velikost (co do počtu znaků), je starší a v současné době se používá především v morfologickém slovníku. Jeho výhodou je menší velikost oproti pozičnímu systému. Kompaktní tag se skládá ze tří částí:

- **Prefix** určuje slovní druh a rozšířený slovní druh (např. VF pro slovesný infinitiv, DB pro příslovce bez možnosti negace a stupňování).
- **Morfologické kategorie** obsahují hodnoty (pole 3–12 pozičního systému), které jsou pro daný prefix relevantní. Například pro substantiva je to rod+číslo+pád+negace, tedy kompaktní tag pro slovo akuzativ slova `možnost` bude `NFS4A`.
- Pole **varianta, styl** je od předcházející části tagu odděleno pomlčkou.

Oba systémy jsou ekvivalentně silné a existuje mezi nimi vzájemně jednoznačné přiřazení. Pro převod z pozičního do kompaktního systému tagů a obráceně slouží jednoduchý skript, který je součástí PDT 1.0 [9]. Základ tvoří dva soubory b2800a.f2o a b2800a.o2f, které jsou v podstatě překladovými slovníky – pro každý tag z jednoho systému je uveden odpovídající tag druhého.

1.1.3 Morfologický slovník

Morfologická analýza ke své práci používá speciální slovník, který slouží k získání základních informací o daném slovu. Pro každé slovo je ve slovníku uložen *kořen* slova, *vzor* a *lemma*. Jednotlivé položky jsou odděleny mezerou, lemma je uvozeno znakem „=“. Například v záznamu pro sloveso `pohnout` je uvedena trojice `pohnou`, `t` a `pohnout`:

```
pohnou  t  =pohnout
```

Na tomto místě je nutné zdůraznit, že některé běžné termíny jsou používány nikoli v lingvistickém, ale v technickém smyslu, proto si uvedeme jejich definici (v některých ohledech poněkud vágní).

- **Předpona (prefix)** slova je volitelná část předcházející kořen slova. Pro potřebu morfologické analýzy se rozlišují pouze tři předpony: `nej`, `ne` a `nejne`. Ostatní jsou uvedeny přímo jako součást kořene u příslušných hesel ve slovníku. Tedy například slova `hrát` a `dohrát` jsou dvě různá slova:

  ```
  hrá    t  =hrát
  dohrá  t  =dohrát
  ```

- **Kořen (root)** je část slova, která se při ohýbání nemění. Případné změny v kořeni se řeší dalším záznamem ve slovníku nebo pomocí derivačních vzorů.

- **Vzor (paradigm)** určuje možné koncovky, které mohou být připojeny za kořen. Nepravidelná slova mají ve slovníku přiřazen vzor `0`.

- **Lemma** určuje základní tvar slova (nominativ singuláru pro substantiva, infinitiv pro slovesa atd.). Zároveň slouží k rozlišení záznamů ve slovníku – záznamy se stejným lemmatem patří k jednomu slovu. Speciálně nepravidelné tvary slov jsou tedy sloučeny právě pomocí lemmatu:

```
kozel  0   =kozel+NMS1@_N
kozla  0   =kozel+NMS2@/NMS4@_N
kozlu  0   =kozel+NMS3@/NMS6@_N
kozl   uv  =kozel
```

Některé vzory nepopisují chování celého slova, ale jen část jeho tvarů (jedná se o *částečné vzory*). Například u slovesa `nést` vzor `t` popisuje infinitiv, `i1` imperativ a `t1n` tvary přechodníku.

```
nés       t    =nést
nes       i1   =nést
nes       p1   =nést
nes       t1n  =nést
nesl      r    =nést
nesen     s2   =nést
pones     0    =nést+VMS2@-1/VMS3@-9
ponesme   0    =nést+VMP1@-1
poneste   0    =nést+VMP2@-1/VMP3@-9
ponesmež  0    =nést+VMP1@-3
ponestež  0    =nést+VMP2@-3/VMP3@-3
pones     p1f  =nést
```

Součástí lemmatu může být také:

- kompaktní tag (oddělený znakem `+`) – zejména pro nepravidelná slova je to jediná možnost, jak uvést tag.

- určení domény slova – umožňuje rozlišit zeměpisná jména od jmen osob, rozlišuje termíny ekonomické, technické, chemické, lékařské.

- záznam o odvození slova (v závorkách, na začátku se znakem `*`) – slova se odvozují derivací, viz kapitolu 1.1.4.

- komentář (v závorkách) – může upřesňovat význam nebo původ slova.

1.1.4 Derivační vzory

Každý přirozený jazyk vystihuje sada víceméně pravidelných postupů pro odvozování nových slov. Zatím jsme měli jedinou možnost, jak zachytit slovo: uvést ho ve slovníku. Pokud bychom to takto udělali, byl by slovník obrovský a bylo by v něm mnoho slov odvozených stejným způsobem. Dalším problémem uvedeného postupu je malá systematičnost slovníku — neměli bychom zachyceny některé souvislosti mezi slovy (např. že `důchodce` a `důchodcův` jsou odvozeny od stejného slova).

Z těchto důvodů se používají další speciální pravidla pro odvozování slov — *derivační vzory*. Derivační vzory se skládají z osmi položek a jsou uloženy v textovém souboru, na každém řádku jeden vzor:

```
vzor  r1,r2,r3,r4,r5,r6,r7
```

Jednotlivé položky popisuje následující tabulka:

vzor	vzor ze slovníku
r1	řetězec, který se přidá ke kořeni ze slovníku, aby tím vznikl nový kořen
r2	vzor, podle kterého se nové slovo ohýbá
r3	způsob odvození nového lemmatu (viz dále)
r4	řetězec, který je potřeba přidat k novému základu (ten se vytvoří podle r3), aby vzniklo nové lemma
r5, r6	týkají se zpětného (derivačního) odkazu na lemma, ze kterého bylo nové lemma utvořeno. Jejich významy jsou stejné jako významy položek r3 a r4, pouze se vytváří lemma, ze kterého bylo nové lemma odvozeno. Výsledné lemma se nezapisuje do slovníkového záznamu celé, ale jako pravidlo, které vypadá takto: xa, kde x je počet písmen, která se mají odtrhnout od nově vytvořeného lemmatu, a je řetězec, kterým se odtržená písmena nahradí, aby vznikl derivační odkaz.
r7	je-li r7 rovno `*`, znamená to, ze se derivační odkaz nevytváří. To platí především u *identických derivací*, kde r1=r3=r4=r5=r6=0 a r2=vzor.

Možné hodnoty r3:

0	lemma se nemění, tj. zůstává to, které je ve slovníku.
x	nové lemma se odvozuje od původního lemmatu, a to tak, že se od něj odtrhne x písmen a nahradí se řetězcem r4
rx	r znamená, že se nové lemma odvozuje od kořene, ne od lemmatu. Číslice x má stejný význam jako v předchozím případě, jen se neodtrhává od lemmatu, ale od kořene.

Odvozování nových slov pomocí derivačních vzorů (derivací) probíhá tak, že na slovníkové heslo se aplikují přípustné derivační vzory. Záznamy, které takto vzniknou, jsou naprosto rovnocenné s těmi, které jsou uvedeny jako hesla ve slovníku.

Ve skutečnosti se popsaný postup využívá vždy, pro slovníková hesla i slova odvozená. Na každé slovníkové heslo se aplikuje jedna nebo více derivací a teprve výsledné záznamy jsou používány v morfologické analýze. Poslední aplikovaný derivační vzor je vždy *identická derivace*[1], která má shodný původní a nový vzor a nemění kořen slova.

Postup derivování záznamů si ukážeme na příkladu slova `zájemce`. Ve slovníku je uvedeno heslo:

```
zájem  sc1  =zájemce
```

tedy vzor je `sc1` (jedna z variant vzoru soudce). V seznamu derivací jsou pro vzor `sc1` uvedeny čtyři řádky:

```
sc1  0,sc1,0,0,0,0,*
sc1  c,uv,r0,cův,0,0,-
sc1  ky,ns3n,r0,kyně,0,0,-
sc1  kynin,in,r0,kynin,r0,kyně,-
```

první derivace je identická, pro ostatní vzory v seznamu najdeme:

```
uv    0,uv,r0,ův,0,0,-
ns3n  0,ns3n,0,0,0,0,*
ns3n  nin,in,r0,nin,0,0,-
in    0,in,r0,0,0,0,-
```

Uvedeme nová hesla, která vznikla podle těchto derivačních pravidel:

První pravidlo pro vzor `sc1` vyrobí stejné slovníkové heslo, jako bylo původní (derivační odkaz se nevytváří, protože r7 je `*`).

```
zájem  sc1  =zájemce
```

Druhé pravidlo vede přes vzor `uv` k heslu:

```
zájemc  uv  =zájemcův_(*2e)
```

Derivační odkaz je `zájemc-e` (odtrhnout dvě písmena a přidat „e“). Třetí pravidlo vede přes `ns3n` a identickou derivaci:

```
zájemky  ns3n  =zájemkyně_(*4ce)
```

[1]Identická derivace nám vlastně říká, kdy máme během derivací generovat odvozený záznam.

Derivační odkaz je `zájem-ce`. Vzor `ns3n` má však ještě další derivaci se vzorem `in`:

```
zájemkynin  in  =zájemkynin_(*2e)
```

Podle posledního pravidla dostaneme:

```
zájemkynin  in  =zájemkynin_(*2ě)
```

Derivační odkaz je `zájemkyn-ě`. Dospěli jsme ke stejné derivaci jako v předchozím případě, tyto duplicitní řádky je třeba z výsledku vypustit.

1.1.5 Seznam koncovek

Pro vzory je definována množina koncovek, které se připojují ke kmeni. U koncovky je uveden tag (v kompaktní podobě), který určuje morfologické kategorie slova vzniklého spojením kmene a dané koncovky.

Zatím jsme se nezmínili o předponách. Slova, která mohou obsahovat předponu (`nej-`, `ne-`, `nejne-`), mají v tagu na místě určeném pro stupeň (negaci) uveden speciální symbol `#` (`@`). Tyto symboly se při analýze slova nahradí podle toho, zda slovo obsahuje danou předponu, či nikoliv.

Například slova skloňovaná podle vzoru `sc1` mohou být použita i v negativním tvaru (tag má na místě pro negaci znak `@`):

```
sc1  ce[NMS1@,NMS2@,NMS4@,NMS5@,NMP4@],
     ci[NMS3@,NMS6@,NMP1@,NMP5@,NMP7@],
     covi[NMS3@-1,NMS6@-1],cem[NMS7@],
     ců[NMP2@],cům[NMP3@],cích[NMP6@],
     cum[NMP3@-6],cema[NMP7@-6],cové[NMP1@-6,NMP5@-6]
```

1.1.6 Morfologické vzory

Morfologické vzory se používají ke dvěma účelům: jednak pro generování jednotlivých tvarů, jednak v derivacích pro generování nových záznamů. Vzory přibližně odpovídají obecně známým vzorům pro češtinu, například pro substantiva to jsou vzory: pán, hrad, muž, stroj, žena, nůše, píseň, kost, město, moře, kuře, stavení, předseda, soudce.

Většina těchto vzorů se dále dělí podle různých výjimek ve skloňování na několik kategorií[2], naopak některá slova tvoří vzory pomocí několika částečných vzorů[3]. Podrobnější údaje o vzorech je možné najít v [1].

[2]Viz např. tabulku 2.4, kde jsou uvedeny různé varianty vzoru nůše.

[3]Celkem je v morfologii [1] vzorů 236.

1.1.7 Algoritmus morfologické analýzy

Zde uvedeme jen základní princip fungování algoritmu, který provádí morfologickou analýzu slova — bez řešení velikosti písmen, případných předpon atd. Kompletní verzi společně s algoritmem pro generování všech slov z lemmatu a tagu čtenář nalezne v [1].

Úkolem algoritmu je analyzovat slovo a určit jeho lemma a tag. Algoritmus používá dvě asociativní pole: `Ending2Paradigm`, které mapuje koncovku na všechny vzory+tagy, a pole `RootParadigm2Lemma`, které mapuje kořen a vzor na lemma+tag. Obě pole získáme předzpracováním slovníku, seznamu koncovek a seznamu derivací. Postup je formálně popsán jako algoritmus 1.

Algoritmus 1 Morfologická analýza

Vstup: slovo
Výstup: všechny dvojice lemma+tag

```
for all rozdělení slova na kořen root a koncovku ending do
  for all paradigm z pole Ending2Paradigm{ending} do
    for all lemma+tag z pole RootParadigm2Lemma{root}{paradigm}
    do
      result ← lemma + tag
    end for
  end for
end for
print(result)
```

1.2 Automatické doplňování slovníku

Z předchozích odstavců je zřejmé, že slovník tvoří jádro celé morfologické analýzy a jeho rozsah a úroveň velmi výrazně ovlivňuje její kvalitu. V současnosti má slovník přes 350 tisíc unikátních položek (v nerozvinuté podobě), přesto se však vyskytuje řada slov, která v něm chybí. Část tvoří slova vznikající neomezeným odvozováním z jiných (například skládáním číslovek: třimiliónyosmsetpadesátisedm), kterých je nekonečně mnoho, takže je není možné dosavadním způsobem zachytit. Ale existují i slova, která by ve slovníku být mohla (například antiautomobilový, předbalíčkový, některá vlastní jména). V této práci bychom se chtěli zaměřit právě na automatické určování vzorů neznámých slov, tedy těch, která nejsou uvedena ve slovníku.

V posledních letech se výzkum v oblasti zpracování přirozeného jazyka zaměřuje také na postupy, jak z rozsáhlých dat automaticky získat informace

o morfologické stavbě daného jazyka. Cílem těchto metod je s minimálním přispěním lidského faktoru najít rozdělení slov na kořen a koncovku, jak odpovídá gramatice daného jazyka. Jedná se především o metodu minimálního popisu[4] (Goldsmith [4], [5]) a kombinace různých jednoduchých přístupů (relativní četnost výskytu jednotlivých tvarů v korpusu, podobnost kontextu atd. (Yarowsky and Wincentowski [6])).

Nabízí se samozřejmě otázka, zda bychom podobný přístup nemohli použít i pro automatické zpracování těch českých slov, která zatím morfologická analýza nerozpozná, a tedy pro ně nezná vzor. Použité metody jsou výborné pro nalezení zákonitostí morfologické struktury jazyka, ale tu máme pro češtinu již velmi dobře zpracovanou. Těžiště významu těchto metod je tedy především ve využití pro jazyky, které nemají k dispozici dostatečně rozvinuté nástroje pro automatickou morfologickou analýzu.

Jak již bylo řečeno, úkolem morfologické analýzy je přiřadit slovům správná lemmata a tagy. Pro neznámé tvary slov existují postupy, jak přiřadit tag a lemma, aniž bychom znali vzor daného slova (Hlaváčová [7]). Ačkoli se jedná o problém podobný našemu, není úplně stejný – pro morfologickou analýzu slova nepotřebujeme znát přesný vzor. Pro demonstraci uvedeme dvě věty:

```
Na povrchu vyrostla jabloň.
Na povrchu vyrostla plíseň.
```

Slova `jabloň` a `plíseň` dostanou obě přiřazený tag `NNFS1-----A----`, ale mají rozdílné vzory (`ps1` a `ps1e`).

Existují i věty, kde některá slova mohou mít více různých vzorů a bez znalosti širšího kontextu nemůže čtenář rozhodnout, která varianta je správná. Příkladem může být slovo `Rozehnal` v následujících větách.

```
Rozehnal jí dobytek.
Rozehnal je dobytek.
```

Z uvedených příkladů je zřejmé, že přiřazení vzorů je úkol poměrně obtížný, protože i pro člověka někdy není snadné stanovit vzor pro daná slova, zejména pokud nám chybí kontext nebo pokud se jedná o slova neznámá. Našeho cíle — automaticky přiřadit každému slovu jeho správný vzor — tedy téměř jistě nedosáhneme, budeme se mu ale snažit maximálně přiblížit.

[4]Minimum Description Length (MDL).

Kapitola 2

Automatické určování vzorů

V této kapitole nejprve zavedeme dále používaná označení a upřesníme zadání našeho problému. Poté provedeme rozbor jednotlivých problémů a navrhneme jejich řešení. Následuje popis použitých algoritmů a metod pro přiřazení vzorů neznámým slovům. V poslední části kapitoly se zaměříme na přiřazení neznámých slov do skupin a určování vzorů pro skupiny slov.

2.1 Označení a definice

Zavedeme označení:[1]

- A je abeceda daného jazyka (češtiny),
- P je množina všech vzorů,
- $F = \{f; f \in A^*\}$ je množina všech řetězců abecedy tvořících slova,
- $L = \{l; l \in A^*\}$ je množina všech řetězců abecedy tvořících lemma,
- $R = \{l; l \in A^*\}$ je množina všech řetězců abecedy tvořících kořen slova,
- $p_1 \Rightarrow p_2$ je derivace vzoru p_1 na vzor p_2 (viz oddíl 1.1.4),
- P_{id} je množina vzorů s identickou derivací, které dále budeme nazývat vzory *základními*.

[1]Připomeňme, že následující text je úzce spjatý s morfologickou analýzou [1], která byla představena v kapitole 1. Termíny jako *vzor*, *lemma*, *kořen*, *koncovka*, *derivace* nebo *morfologický slovník* tedy budeme chápat v jejím kontextu. Dále budeme používat základní pojmy z oblasti pravděpodobnosti, jako jsou náhodná veličina, její rozdělení atd.

Dále definujme relaci $\Rightarrow^*$ a funkci[2] Cl:

- $p_1 \Rightarrow^* p_n \iff \exists_{p_2,\ldots,p_n}(\forall_{p_i} p_{i-1} \Rightarrow p_i)$
- $Cl : P_{id} \rightarrow 2^P : Cl(p_i) = \{p_j : p_j \Rightarrow^* p_i\}$

Přiřazení přípustných vzorů je taková funkce Pa:

$$Pa : F \rightarrow 2^{R \times P \times L} : Pa(w) = \{\langle r, p, l\rangle; r \in R \& p \in P \& l \in L\},$$

která každému slovu přiřazuje množinu trojic ⟨kořen, vzor, lemma⟩, jež jsou pro dané slovo přípustné. To znamená, že slovo w je tvar slova *lemma* s kořenem *kořen*[3] a má jednu z koncovek *vzoru*. Zobecníme-li tuto funkci na více slov, dostaneme:

$$Pa : 2^F \rightarrow 2^{R \times P \times L} : Pa(w_1, \ldots, w_n) = \cap Pa(w_i)$$

Dalším pojmem, který zde zavedeme, je *derivační strom*. Jedná se o množinu vzorů, které vzniknou derivací z jednoho vzoru:

$$Tr(p) : P \rightarrow 2^P : Tr(p) = \{q : p \Rightarrow^* q\}$$

2.2 Určování vzorů

Naším úkolem je ke každému slovu najít vzor, podle kterého se dané slovo ohýbá. Například pro slovo `pejsek` je to vzor `pn17`[4], pro slovo `ženou` je to vzor `zn1`[5]... nebo se jedná o tvar slovesa `hnát`? Je zřejmé, že z pouhé znalosti slova vzor jednoznačně neurčíme.

Dokonce i pro jeden vzor nemusí být vždy jasné, o který tvar daného slova se jedná. Představme si, že známe vzor `zn1` pro slovo `sambou`, ale nevíme, která část slova je kořen. Podíváme-li se na tabulku 2.1, kde jsou uvedeny tvary vzoru žena, vidíme, že kromě instrumentálu singuláru (kořen `samb`, zakončení `-ou`) by se mohlo jednat o akuzativ (kořen `sambo`, koncovka `-u`) nebo dokonce o genitiv plurálu (kořen `sambou`, koncovka prázdná). Proto budeme vždy uvažovat o dvojici ⟨kořen, vzor⟩, nikoli o vzoru samotném.

Naším cílem je vytvořit algoritmus, který na vstupu dostane text obsahující neznámá slova a pro každé z nich se pokusí určit kořen a vzor. K vytvoření úplného slovníkového záznamu nám zbývá najít lemma. Chceme tedy najít trojici ⟨kořen, vzor, lemma⟩, která tvoří nové heslo v morfologickém slovníku. Náš úkol zformulujeme v následujícím popisu.

[2] Označení 2^S reprezentuje množinu všech podmnožin S.
[3] *Kořen* zde, ani dále v textu není pasivum.
[4] Jedna z variant mužského životného vzoru pán.
[5] Vzor žena.

	1	2	3	4	5	6	7
sg.	žen-a	žen-y	žen-ě	žen-u	žen-o	žen-ě	žen-ou
pl.	žen-y	žen	žen-ám	žen-y	žen-y	žen-ách	žen-ami

Tabulka 2.1: Zakončení slov vzoru `zn1` žena

Vstup: text ve formátu CSTS[6] zpracovaný morfologickou analýzou. Tagem `X@----------` jsou označena slova, která morfologická analýza nerozpoznala, a nejsou tedy obsažena v jejím slovníku.

Výstup: seznam trojic ⟨kořen, vzor, lemma⟩ pro každé neznámé slovo ze vstupu. Trojice budou seřazeny podle pravděpodobnosti, s jakou příslušejí k danému slovu.

Následuje rozbor některých problémů a otázek, které vyplývají z podrobnější analýzy problému přiřazení vzorů ke slovům.

2.2.1 Nepravidelná slova

Morfologie, se kterou pracujeme, obsahuje 236 různých vzorů. Přestože je to podstatně více, než na kolik jsme zvyklí ze školy, nepopisují tyto vzory ani zdaleka každé české slovo. Slova, případně jen některé tvary slov, které nemají přiřazen vzor, jsou považována v tomto smyslu za *nepravidelná*. Jedná se především o slova nesklonná (předložky, spojky), cizí nebo přejatá slova a některé často používané tvary (především substantiv a sloves). Z celkového počtu 260 069 lemmat[7] v morfologickém slovníku má alespoň jeden nepravidelný tvar 25 069 lemmat (9,6 %).

Nepravidelná slova nám budou určování vzorů velmi znepříjemňovat. Snadno se nám například stane, že neznámé slovo má stejné zakončení jako některý z častých vzorů, a přesto se jedná o nepravidelný tvar.

2.2.2 Více vzorů pro jedno lemma ve slovníku

Navíc neplatí, že co lemma, to jeden vzor. Především častější slovesa, ale i jiná slova mají několik částečných vzorů, ze kterých se skládají všechny jejich možné tvary. Například lemma `slaboproudý` má ve slovníku záznamy:

`slaboproud yxx =slaboproudý` *slaboproudý, slaboproudost*
`slaboproud yxi =slaboproudý` *slaboproudí*

[6] „Czech Sentence Tree Structures“, primární formát dat uložených v PDT 1.0 [9]

[7] Počet unikátních záznamů ve slovníku, nepočítáme zde záznamy, které vzniknou derivacemi.

Jinou kategorií slov, která má také přiřazeno více vzorů, jsou obouvidá slovesa (mající dokonavý i nedokonavý vid). Například sloveso absolvovat má ve slovníku dva záznamy, pro každý z vidů jeden:

```
absolv  ovatd  =absolvovat  dokonavé (absolvovavší)
absolv  ovatn  =absolvovat  nedokonavé (absolvující)
```

Musíme se tedy zamyslet nad otázkou, zda se budeme snažit najít všechny vzory k danému lemmatu, nebo se spokojíme s nalezením alespoň jednoho z nich. Najít jeden ze vzorů je určitě snazší než najít všechny vzory. Pokud ovšem naším cílem bude najít jen některé vzory, je otázka, nakolik je vhodné přidat takový neúplný záznam do slovníku. Budeme tedy chtít najít — pokud možno — co nejúplnější množinu vzorů, která přísluší k danému slovu.

2.2.3 Hierarchie vzorů

Zaměřme se nyní na vzory, které můžeme daným slovům přiřadit. Jak bylo popsáno v kapitole 1, ve slovníku nejsou uvedeny ty záznamy, které lze pravidelně odvodit z jiných pomocí derivačních vzorů (derivací). Je otázka, zda budeme pro neznámé slovo chtít určit spíše záznam derivovaný[8] (*základní*), nebo se budeme snažit najít záznam, který by byl ve slovníku (tedy ten, ze kterého je odvozen záznam pro naše neznámé slovo).

Například u slova `zájemkynino` z oddílu 1.1.4 se jedná o vzory `sc1` (záznam ve slovníku), nebo `in` (základní vzor pro morfologickou analýzu), viz obrázek 2.1.

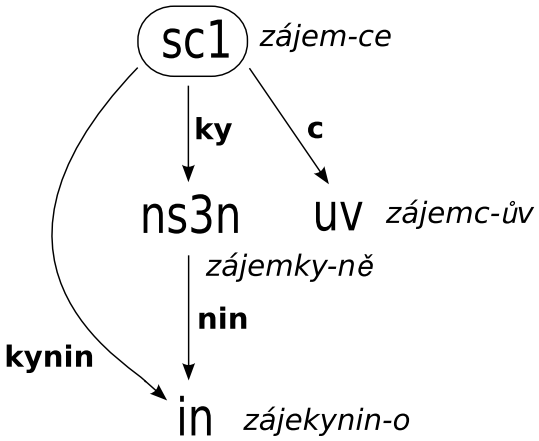
Jelikož naším cílem je rozšíření morfologického slovníku, zajímá nás především první možnost — záznam (vzor), který tvoří *kořen derivačního stromu*[9]. Jedná se o vzor, který je kořenem ve stromě derivací vedoucím od slovníkového záznamu k danému vzoru. Neboli formálně zapsáno

$$R(p) : P_{id} \rightarrow 2^P : R(p) = \{q : q \in Cl(p) \ \& \ \forall_{p_i \in Cl(p)} \ q \Rightarrow^* p_i\}$$

Jelikož záznam může vzniknout derivováním několika různých vzorů, může být (a většinou také je) takových kořenů více. Naší snahou bude hledat vzory, které jsou kořenem derivačního stromu.

[8] Tedy ten, který by se přímo použil v morfologické analýze pro dané slovo a vznikl nějakou derivací záznamu slovníkového.

[9] Dopouštíme se zde určité nepřesnosti: některé uzly derivačního grafu mohou být spojené, pokud více různých derivací vede ke stejnému vzoru (viz obr. 2.1). Jedná se pak o *orientovaný acyklický graf*, DAG. Nicméně můžeme si představit tyto uzly odděleně a zůstane nám strom i v matematickém smyslu. Dovolíme si tedy používat pojem strom i pro tento acyklický graf.



Obrázek 2.1: Kořen derivačního stromu.

2.2.4 Kombinované vzory

Jak už bylo uvedeno v oddílu 2.2.2, existují lemmata, která tvoří tvary pomocí více vzorů. Jedná se vlastně o určitý druh nepravidelnosti — kdyby tato slova byla pravidelná, pravděpodobně by měla zvláštní vzor. Mnoho těchto slov má více různých kořenů, například ke slovesu `poklást` patří vzorů šest, z nichž pouze dva mají stejný kořen

`poklás`	`t`	`=poklást`	*poklást, poklásti*
`poklad'`	`i1`	`=poklást`	*poklaďme, poklaďte, . . . , poklaďteže*
`poklad`	`p1`	`=poklást`	*pokladu, poklade, . . . , pokladou*
`pokladl`	`r`	`=poklást`	*poklad, poklada, poklado, . . . , pokladyť*
`poklad`	`md`	`=poklást`	*pokladši, pokladše, pokladšího, . . .*
`pokladen`	`s2`	`=poklást`	*pokladení, pokladeně, pokladenost, . . .*

Chtěli bychom nějakým způsobem formalizovat, že dané vzory mohou v některých případech tvořit tvary jednoho lemmatu. Prostředek, který se k tomu nabízí, jsou derivační vzory. Pro každou takovouto skupinu vzorů jednoho lemmatu vytvoříme nový morfologický vzor, který bude reprezentovat celou skupinu, a přidáme derivační vzory, které nám budou udávat, jak z nového vzoru získáme původní vzory.

Zbývá rozmyslet, jak bude vypadat nový společný kořen a jakými sufixy se z něj budou vytvářet kořeny původní — jako nový kořen vezmeme maximální společný prefix původních kořenů.

Nový vzor pojmenujeme například tak, že součástí jména derivačního vzoru jsou jednak původní vzory, jednak sufixy, které doplňují nový kořen na původní kořeny:

`i1_ad'+md_ad+p1_ad+r_adl+s2_aden+t_ás`

Nové derivační vzory pak mají tvar (viz oddíl 1.1.4):

```
i1_ad'+md_ad+p1_ad+r_adl+s2_aden+t_ás  aden,s2,0,0,0,0,*
i1_ad'+md_ad+p1_ad+r_adl+s2_aden+t_ás  ad',i1,0,0,0,0,*
i1_ad'+md_ad+p1_ad+r_adl+s2_aden+t_ás  adl,r,0,0,0,0,*
i1_ad'+md_ad+p1_ad+r_adl+s2_aden+t_ás  ad,md,0,0,0,0,*
i1_ad'+md_ad+p1_ad+r_adl+s2_aden+t_ás  ad,p1,0,0,0,0,*
i1_ad'+md_ad+p1_ad+r_adl+s2_aden+t_ás  ás,t,0,0,0,0,*
```

Pokud popsanou operaci provedeme na všechna lemmata ve slovníku a vypustíme derivační vzory, které se vyskytují pouze jednou, dostaneme se z původních 236 vzorů na 1311 a z původních 651 derivačních vzorů na 4439.

Tímto postupem získáme možnost zlepšit přesnost výsledku, na druhou stranu se jedná o poměrně výrazné zvýšení počtu vzorů (a tím i zpomalení programů, které se vzory pracují).

2.2.5 Předpony

Použitá morfologie rozeznává pouze dvě předpony: `nej-` a `ne-`. Slova, která jsou odvozena jinou předponou (např. `do-`, `na-`, `u-`, `vy-`, `za-`), se pokládají za různá slova a vyžadují samostatný záznam ve slovníku.

Zda daný tvar slova může být uvozen jednou z předpon `nej-` a `ne-` (případně oběma), je možné zjistit z použitého vzoru. Bohužel to ale neznamená, že by ve slovníku byly uvedeny pouze pozitivní tvary slov:

```
pravd    zn1  =pravda
nepravd  zn1  =nepravda
```

Vzor `zn1` umožňuje u všech svých tvarů negaci (tedy `ne-pravda` i `ne-nepravda`). Máme-li určit kořen a vzor slova `nepravda`, nevíme, jestli je to negativní tvar slova `pravda`, nebo zvláštní slovo. Některé tyto záznamy ve slovníku mohou být hodnoceny jako chybné, ale existují například i slova s předponou, která nemají pozitivní tvar, ačkoli z hlediska morfologie by klidně existovat mohl. Typickým příkladem je slovo `negace`.

Pokud tedy máme zjistit kořen (a lemma) příslušející k neznámému slovu, které obsahuje jednu z předpon (`nej-`, `ne-`, `nejne-`), nemůžeme si být jistí, zda se jedná opravdu o předponu, nebo jen o součást slova.

2.3 Algoritmy

Nyní se zaměříme na to, kterými metodami lze s co největší přesností určit, jaký vzor náleží neznámým slovům. Jedná se převážně o jednoduché sta-

tistické metody založené na odhadu pravděpodobností pomocí sledování četnosti v trénovacích datech.

Na začátku uvedeme algoritmus pro generování všech možných vzorů pro zadaná slova, poté ho rozšíříme na několik slov. Následovat bude popis metod, které jsme použili a implementovali. Budeme používat různé pravděpodobnostní modely, před jejich použitím je vždy popíšeme.

2.3.1 Generování možných vzorů

Zcela základním úkolem pro nás bude vygenerovat pro každé slovo všechny jeho možné vzory. Budeme k tomu potřebovat tabulku `Ending2Paradigm`, ve které bude uveden seznam možných *základních* vzorů pro každou koncovku. Můžeme si ji předpočítat invertováním tabulky koncovek z morfologie (viz oddíl 1.1.5).

Algoritmus generování vzorů bude probíhat takto:

- Dané slovo nejprve rozdělíme všemi možnými způsoby na kořen a koncovku (ta může být i prázdná).
- Pro každou koncovku najdeme v tabulce `Ending2Paradigm` možné základní vzory. Dostaneme tak trojici ⟨kořen, vzor, koncovka⟩. Koncovku zapomeneme, a tím vzniknou dvojice ⟨kořen, vzor⟩. Získali jsme základní vzory příslušející danému slovu.
- Dále je potřeba najít zpětný „uzávěr“ základních vzorů. Ten získáme tak, že projdeme opačným směrem možné derivace (s tím, že zohledníme změnu v kořeni slova).

Uvedený postup si ukážeme na slově `zájemkynino`. Podle prvního bodu dostaneme jako jedno z rozdělení `zájemkynin-o`. Podíváme-li se do tabulky `Ending2Paradigm`, zjistíme, že seznam vzorů pro tuto koncovku má dvacet položek (`r`, `mt7`, `zn1x`, `in`, `aj`, `zn19`, `mts`, `mt1`, `s1`, `ajn`, `pd1`, `rx`, `zn20`, `mt1x`, `ccf`, `mt1e`, `zn1`, `pd2`, `zn7`, `mt1i` — pět různých variant vzoru žena, šest variant vzoru město, dvě varianty vzoru předseda atd). Je mezi nimi i vzor `in` (s kořenem `zájemkynin`. Podíváme se nyní, jak najdeme uzávěr pro tento vzor. V přehledu derivačních vzorů vyhledáme možné zpětné derivace, které přicházejí v úvahu. Zajímají nás pouze ty, které jako druhou položku mají konec aktuálního kořene.

```
ns3n  nin,in,r0,nin,0,0,-
zn1n  in,in,r0,in,0,0,-
sc1o  kynin,in,r0,kynin,r0,kyně,-
sc1   kynin,in,r0,kynin,r0,kyně,-
mz5   kynin,in,r0,kynin,r0,kyně,-
```

Do seznamu vzorů přidáme položky: ⟨zájemky, ns3n⟩, ⟨zájemkyn, zn1n⟩, ⟨zájem, sc1o⟩, ⟨zájem, sc1⟩ a ⟨zájem, mz5⟩.

Rekurzivní aplikací na vzory `ns3n` a `mz5` (ostatní mají jen identické zpětné derivace) dostaneme další derivační vzory:

```
sc1o  ky,ns3n,r0,kyně,0,0,-
sc1   ky,ns3n,r0,kyně,0,0,-
mz5   ky,ns3n,r0,kyně,0,0,-
mz5i  0,mz5,0,0,0,0,*
```

Jediný nový vzor je ⟨zájem, mz5i⟩, ostatní už v seznamu máme. Takto jsme získali uzávěr pro vzor `in`, pro zbývajících devatenáct vzorů bychom postupovali obdobně. Důležité je, že původní slovníkové heslo[10]

```
zájem  sc1  =zájemce
```

je mezi nalezenými páry.

Je zřejmé, že možných dvojic ⟨kořen, vzor⟩ bude mnoho. Navíc existují vzory, které mají pro některé tvary prázdné koncovky (například vzor hrad v nominativu singuláru). Prázdnou koncovku získáme při dělení slova na kořen a koncovku u libovolného slova, takže takové vzory budou v seznamu obsaženy vždy.

Uvedený postup je formálně popsán jako algoritmus 2.

Algoritmus 2 GetPossibleWordParadigms: Vrátí všechny možné dvojice ⟨kořen, vzor⟩, pro slovo w.

Vstup: slovo w
Výstup: všechny dvojice ⟨kořen, vzor⟩ odpovídající slovu w
  for all rozdělení slova w na kořen *root* a koncovku *ending*[11] **do**
    for all vzor $paradigm \in Ending2Paradigm[ending]$ **do**
      $basic_i \leftarrow \langle root, paradigm \rangle$
    end for
  end for
  {Najdeme „uzávěr“ zpětných derivací}
  for all $\langle root, paradigm \rangle \in basic$ **do**
    $result_i \leftarrow Cl(root, paradigm)$[12]
  end for
  return$(result)$

---

[10] Viz příklad v oddílu 1.1.4

[11] Algoritmus předpokládá, že vstupní slova již neobsahují předponu.

[12] Je zde použito rozšíření funkce Cl tak, že současně se změnou vzoru se patřičně změní také kořen slova.

Generování možných vzorů pro skupinu slov

Nyní algoritmus rozšíříme na skupinu slov. To bude užitečné především v případě, kdy najdeme více tvarů jednoho slova a budeme chtít zjistit, které vzory jsou přípustné pro všechny tvary současně.

Využijeme algoritmus 2, který nám vrátí seznam přípustných dvojic ⟨kořen, vzor⟩ pro každé slovo. Jelikož chceme vybrat ty položky, které odpovídají všem slovům, provedeme operaci průniku na všechny seznamy. Popsaný postup formalizujeme zápisem jako algoritmus 3.

Algoritmus 3 GetPossibleWordGroupParadigms: Generuje možné dvojice ⟨kořen, vzor⟩ pro skupinu slov

Vstup: slova $w = \{w_1, \ldots, w_n\}$
Výstup: všechny dvojice ⟨kořen, vzor⟩ odpovídající slovům $w_1, \ldots, w_n$

```
for i = 0 to n do
    ⟨root_i, paradigm_i⟩ ← GetPossibleWordParadigms(w_i)
end for
result ← ⋂_{i=1,...,n} ⟨root_i, paradigm_i⟩
return(result)
```

2.3.2 Přiřazení nejpravděpodobnějších vzorů

Nyní umíme najít možné vzory (páry ⟨kořen, vzor⟩) k jednomu i více neznámým slovům. Můžeme se tedy začít zabývat metodami, které nám umožní s co největší přesností určit, který z možných vzorů je ten správný.

K tomu můžeme využít nejrůznější postupy, od zkoumání předpon, kořenů a koncovek slov až po zkoumání okolí neznámých slov. Přestože jsou to metody rozličné, můžeme mezi nimi najít dvě významné skupiny:

Metody nekontextové zkoumají pouze samotné slovo, jeho možné koncovky, kořen, předpony, případně další vlastnosti.

Metody kontextové se zaměřují na okolí neznámého slova a snaží se v něm najít nápovědu k určení tvaru, případně vzoru slova[13], nebo alespoň některé tvary vyloučit.

Dále uvedeme tři metody nekontextové a jednu kontextovou. Nejprve však zavedeme model, ve kterém se budeme pohybovat.

[13] Je zřejmé, že pokud se těsně před neznámým slovem `kostliveček`, `broučíček` nebo `ševeček` vyskytuje slovo `malý`, bude to pravděpodobně tvar některého mužského vzoru.

2.4 Pravděpodobnostní model

V oblasti zpracování přirozeného jazyka často potřebujeme odhadnout, s jakou pravděpodobností nastane určitý jev. Například nás může zajímat, jak je pravděpodobné, že po slově `vysoký` bude následovat slovo `strom`, `dům`, případně slovo `kaluž`. Je zřejmé, že první dva případy budou pravděpodobnější než případ třetí. Chceme-li takové zákonitosti nějak zachytit či formalizovat, budeme potřebovat *pravděpodobnostní model*, který každému možnému výsledku přiřadí pravděpodobnost, s jakou jev nastane. Protože přesné pravděpodobnosti neznáme, budeme se k nim chtít alespoň přiblížit.

K odhadu pravděpodobnosti $P(A)$ jevu A použijeme jeho relativní četnost, což je jeho absolutní četnost[14] $C(A)$ dělená počtem pokusů N:

$$P(A) = \frac{C(A)}{N}$$

Podobně pro podmíněnou pravděpodobnost $P(A|B)$ jevu A, nastal-li jev B, odhadneme počet případů, kdy nastal jev A, nastal-li jev B, a vydělíme počtem případů, kdy nastal jev B:

$$P(A|B) = \frac{C(A|B)}{C(B)}$$

Pro použitý postup se používá termín Metoda maximální věrohodnosti (MLE)[15]. Je to jedna z nejjednodušších metod, jak určit neznámé rozdělení náhodné veličiny z předcházejících výsledků pokusů.

Použití této metody je rozděleno do dvou fází: nejdříve se stanoví pravděpodobnosti jednotlivých jevů pomocí četnosti v předem označených datech (fáze trénování) a poté se zjištěné pravděpodobnosti aplikují na vstupní data (fáze vybavování, v našem případě se jedná o přiřazování vzorů).

2.4.1 Odhad pravděpodobností

Tato fáze se někdy také nazývá trénování[16] — určujeme pravděpodobnosti jednotlivých podmíněných jevů na základě trénovacích dat, která máme k dispozici. Trénujeme vlastně náš model tak, aby co nejlépe odpovídal trénovacím datům, neboli se snažíme maximalizovat pravděpodobnost těchto dat v našem modelu. V případě metody maximální věrohodnosti počítáme relativní frekvence jednotlivých jevů.

[14] Neboli počet příznivých výsledků pokusu.

[15] Maximum Likelihood Estimation

[16] Především u složitějších metod (neuronové sítě, odhad parametrů skrytých Markovových modelů apod.).

Podívejme se na metodu maximální věrohodnosti v souvislosti s pravděpodobností výskytu koncovek pro daný vzor, například vzor píseň.

V předem označených (*trénovacích*) datech se vyskytuje celkem sedm různých tvarů slov, která mají vzor píseň: `da-ní`, `da-ni`, `dla-ň`, `hole-ně`, `kampa-ni`, `kampa-ň`, `skří-ň`. Počet koncovek, jejich četnosti a odhadnuté pravděpodobnosti jsou shrnuty v tabulce 2.2.

koncovka	absolutní četnost	relativní četnost
-ň	3	0.42857
-ni	2	0.28571
-ně	1	0.14286
-ní	1	0.14286

Tabulka 2.2: Přehled koncovek vzoru píseň v trénovacích datech

Narazíme-li při hodnocení na slovo, které má vzor píseň, můžeme podle našeho modelu říct, že s největší pravděpodobností (42,857 %) bude mít koncovku `-ň`.

2.4.2 Vyhlazování

Uvažme, co se stane, pokud máme odhadnuté pravděpodobnosti pro vzor píseň (viz tabulka 2.2) a narazíme v textu na slovo `náplněmi`. Zakončení `-emi` v tabulce nemáme, a jelikož součet pravděpodobností možných jevů je vždy 1, musíme této koncovce přiřadit nulovou pravděpodobnost. Vidíme, že náš model není příliš dobrý — slovu, které se vyskytuje v textu, přiřazuje nulovou pravděpodobnost (jev nemožný).

Narazili jsme na problém, který se při aplikaci statistických metod v počítačové lingvistice vyskytuje velmi často — problém řídkých dat. Ačkoli jsou pro češtinu dostupné poměrně rozsáhlé zdroje dat (milióny až stovky miliónů slov), stejně nastanou jevy, které se v těchto datech nevyskytují, a tudíž ve fázi trénování nebyly zachyceny (a nebyla patřičně upravena pravděpodobnosti jejich výskytů).

Vyhlazování je metoda, jejímž úkolem je přiřadit nenulové pravděpodobnosti i jevům, které se v trénovacích datech vůbec nevyskytovaly z důvodu řídkosti dat. Stále ovšem musí platit, že součet pravděpodobností všech možných jevů je 1. Abychom mohli přiřadit nenulovou pravděpodobnost jevům neviděným, musíme naopak nějakým způsobem snížit pravděpodobnost jevům, které jsme v trénovacích datech viděli.

Laplaceův zákon

Nejjednodušší způsob, jak dosáhnout přiřazení nenulových pravděpodobností, je každému možnému výsledku přiřadit absolutní četnost alespoň 1 (tedy předpokládáme, že každý výsledek jsme viděli alespoň jednou):

$$P_{Lap}(A) = \frac{C(A) + 1}{N + B}$$

B je zde počet možných výsledků. Problém této metody je, že nikdy neviděným jevům přiděluje příliš velkou pravděpodobnost oproti jevům, které byly v trénovacích datech pozorovány s malou četností. Špatně rozlišuje jevy nemožné a jevy nastávající málokdy. V literatuře je tento postup označován jako Laplaceův zákon.

Lidstonův zákon

Používanější je zákon Lidstonův, který umožňuje nastavit, jaká četnost bude přidělena jevům, které se nevyskytovaly v trénovacích datech. Parametr λ ovlivňuje váhu, kterou se tyto jevy budou projevovat v celkovém rozdělení. Může nabývat hodnoty od 0, kdy se vzorec mění v původní odhad pravděpodobností pomocí metody maximální věrohodnosti, až po 1, což je vlastně Laplaceův zákon.

$$P_{Lid}(A) = \frac{C(A) + \lambda}{N + B\lambda}$$

Je zajímavé, že na Lidstonův zákon můžeme pohlížet také jako na lineární kombinaci metody maximální věrohodnosti a uniformního rozdělení:[17]

$$P_{Lid}(A) = \mu\frac{C(A)}{N} + (1 - \mu)\frac{1}{B}$$

V literatuře [8] se uvádí jako nejpoužívanější hodnota parametru $\lambda = \frac{1}{2}$. Jedná se zřejmě o lepší odhad než v případu Laplaceova zákona, ale stále to nemusí být hodnota optimální. Jak ale takovou optimální hodnotu najít?

Jak již bylo zmíněno výše, snažíme se maximalizovat pravděpodobnost trénovacích dat. To nám ovšem v praxi příliš nevyhovuje, protože algoritmy nebudeme aplikovat na trénovací data, ale data reálná, která jsme ještě nikdy neviděli. Chtěli bychom tedy maximalizovat pravděpodobnost na reálných datech, ta ovšem nemáme k dispozici. Můžeme zvolit kompromis, nejdříve budeme maximalizovat pravděpodobnost trénovacích dat a poté se na jiných

[17] Dosazením $\mu = \frac{N}{N+B\lambda}$ se dostaneme k původnímu vzorci.

datech budeme snažit odhadnout pravděpodobnosti jevů, které jsme nezaznamenali v trénovacích datech.

Je zřejmé, že k poslednímu kroku potřebujeme data jiná než trénovací[18], používá se pro ně označení *odložená data*[19].

Budeme tedy maximalizovat pravděpodobnost odložených dat, v případě Lidstonova zákona pomocí parametru λ. Pravděpodobnost dat umíme spočítat pro jednu konkrétní hodnotu parametru; budeme potřebovat metodu, jak získat maximum funkce $P(\lambda)$ pro různé hodnoty λ.

Metoda zlatého řezu

Maximalizace (minimalizace) metodou zlatého řezu[20] je jednoduchý numerický algoritmus, který pro danou funkci určí extrém pomocí jejích hodnot. Funkce může být poměrně obecná: stačí, aby byla spojitá a aby v daném intervalu nabývala alespoň jednoho maxima (minima). Pro každou iteraci je potřeba spočítat hodnotu pouze v jednom dalším bodě.

Předpokládejme, že funkce $f(x)$ je na intervalu $\langle a, c\rangle$ spojitá a nabývá na intervalu (a, c) maxima. Existuje tedy bod $b \in (a, c)$ takový, že $f(b) > f(a)$ a současně $f(b) > f(c)$.

Metoda zlatého řezu hledá maximum na intervalu (a, c) následujícím způsobem:

- V delším z intervalů (a, b) a (b, c) vybereme bod x. Bez újmy na obecnosti můžeme předpokládat, že $x \in (a, b)$, neboli $a < x < b < c$. Bod je vybrán tak, že dělí delší z intervalů v poměru zlatého řezu[21].
- Spočítáme $f(x)$ a podle jejího výsledku změníme jeden z krajních bodů intervalu (a, c).
  - Je-li $f(x) > f(b)$, budeme maximum hledat v intervalu (a, b) $(a < x < b)$,
  - je-li $f(x) < f(b)$, budeme maximum hledat v intervalu (b, c) $(x < b < c)$.
- Maximum, které jsme dosud našli, je prostřední bod z aktuálního intervalu. Postup opakujeme, dokud interval, ve kterém extrém hledáme, není dostatečně malý (menší než předem daná hodnota).

[18] Zajímá nás totiž pravděpodobnost jevů, které se v trénovacích datech vůbec nevyskytují, tedy z principu musí jít o jiná data.

[19] Často jsou označovaná jako heldout data. Název poměrně dobře vystihuje, že data jsou odložená před trénováním za účelem vyhlazování.

[20] V literatuře je často používán termín Golden Section Search.

[21] $(1 - g) : g$, kde $g = \frac{3-\sqrt{5}}{2}$

Jiný druh vyhlazování si představíme v kapitole 2.4.4, kde se bude jednat o vyhlazování n-gramů.

2.4.3 Nekontextové metody

Nyní konečně můžeme přikročit k metodám, které použijeme pro nalezení nejpravděpodobnějších vzorů pro neznámá slova. V tomto oddílu se zaměříme na metody, které se zabývají pouze daným slovem, bez ohledu na slova okolní. Naším cílem je tedy odhadnout pravděpodobnost, s jakou je určitá dvojice ⟨kořen, vzor⟩ vzorem slova w, neboli:

$$P(\langle\text{kořen, vzor}\rangle|w)$$

První metodou, kterou si představíme, je výběr nejpravděpodobnějšího z možných vzorů, další metody se budou zabývat koncovkami slov a zakončením kořene slova.

Nejpravděpodobnější ze vzorů

Tato metoda je naprosto základní a asi jedna z prvních, která člověka při pohledu na problém napadne, je však až překvapivě účinná. Představme si, že pro neznámé slovo známe seznam dvojic ⟨kořen, vzor⟩, které jsou pro něj přípustné (výstup algoritmu 2). Tento seznam se liší slovo od slova, záleží především na možných koncovkách slova. Není přesně určeno, jak dlouhé koncovky se berou v úvahu — to záleží na tom, zda konec slova tvoří koncovku některého ze vzorů, či nikoli.

Setříděný seznam možných vzorů tvoří vlastně něco jako otisk daného slova, který ho svým způsobem reprezentuje. Slovo, které je stejného vzoru (a stejného tvaru), bude mít pravděpodobně stejný nebo velmi podobný seznam možných vzorů.

Úplně stejná úvaha platí i pro skupinu slov — opět dostaneme seznam možných vzorů, ale tentokrát bude pro všechna slova společný. Seznam není nikdy delší než nejkratší ze seznamů pro jednotlivá slova[22].

Například pro slova: `demonstracích` a `demonstracemi` je společný otisk[23] vzorů následující[24]:

```
ns07.ns1.ns10.ns3x.ns7.ps18.ps5
```

[22] Jak plyne z vlastností průniku.

[23] Otisk pro jedno slovo má vzorů mnohonásobně více, je to způsobeno mnoha vzory s nulovou koncovkou.

[24] Vzory jsou lexikograficky uspořádané a oddělené tečkou

Přesně stejný otisk dostaneme i od slov `filtracích` a `filtracemi`. Otisk nám takto vlastně umožňuje uchopit slova, která mají shodné nebo velmi podobné koncovky.

Na tomto místě je dobré připomenout, že dosud jsme vždy uvažovali nejen o vzoru samotném, ale současně i o kořeni daného slova. To se nám v tomto případě příliš nehodí — v otisku nemůžeme mít zakomponován kořen, protože otisk by pak platil jen pro jedno konkrétní slovo. Zároveň ale potřebujeme nějak zachytit, že dané slovo může patřit ke vzoru s použitím několika různých koncovek (např. tvar `ženou` z oddílu 2.2). Nabízí se řešení takové, že si výskyty koncovek pro různé vzory očíslujeme a pro otisk budeme používat vzory současně s pořadím. Například pro tvar `ženou` by součástí otisku bylo:

```
.zn.zn-2.zn-3.
```

Fáze trénování probíhá tak, že pro každý otisk, který se vyskytuje v trénovacích datech, počítáme rozdělení jednotlivých možných vzorů. Z něj odvodíme pravděpodobnosti metodou MLE a hodnoty vyhladíme pomocí metody Lindstonova zákona a metody zlatého řezu na odložených datech. Parametr λ se určuje pro všechny otisky dohromady.

Výsledkem trénovacího procesu je tabulka $\mathtt{P_{PS}}[\mathtt{otisk}, \mathtt{vzor}]$, kde je pro každý otisk uvedena pravděpodobnost jednotlivých vzorů (ať už získaná z trénovacích dat, nebo ve fázi vyhlazování). Její použití je přímočaré, nejprve získáme otisk pro zadané vstupní slovo (případně slova) a poté pro každý z možných vzorů najdeme v tabulce jeho pravděpodobnost.

Postup je popsán jako algoritmus 4.

Koncovky vzorů

Další z metod s poměrně zřejmou motivací je zaměření se na četnosti výskytů jednotlivých koncovek vzorů v běžných textech. Čeština je flexivní jazyk a velká část slov tvoří mnoho různých koncovek[25]. Je zřejmé, že ne všechny se budou používat stejně často, a toho se pokusíme využít v metodě sledující četnost koncovek pro různé vzory.

Popsaná metoda je formalizovaná zápisem algoritmu 5. Používá se v ní tabulka $\mathtt{P_{PE}}[\mathtt{vzor}, \mathtt{koncovka}]$, která vznikne v trénovací fázi a vyjadřuje pravděpodobnost, že pro `vzor` bude použita `koncovka`. Stejně jako u předchozí metody dojde při trénování ke stanovení rozdělení metodou MLE a následně jsou rozdělení vyhlazena na odložených datech.

[25] Existuje mnoho slov, která mají přes dvě desítky různých koncovek.

Algoritmus 4 ScorePS: Každou dvojici ⟨kořen, vzor⟩ ohodnotí pravděpodobností, s jakou se může vyskytnout pro danou skupinu slov. Používá tabulku $\mathtt{P_{PS}}$, která přiřadí množině možných vzorů pravděpodobnost výskytu každého z nich.

Vstup: seznam slov $w = \{w_1, \ldots, w_n\}$, seznam $gp = \{gp_1, \ldots, gp_m\}$ možných dvojic ⟨kořen, vzor⟩ pro slova z w

Výstup: vektor $prob = \{prob_1, \ldots, prob_m\}$ – pravděpodobnostní ohodnocení každé dvojice.

```
for all ⟨kořen, vzor⟩ ∈ gp do
  otisk ← otisk ⊕ vzor {konkatenace}
end for
for all ⟨kořen, vzor⟩ ∈ gp do
  prob_i ← P_PS[otisk, vzor]
end for
return(prob)
```

Algoritmus 5 ScorePE: Každou dvojici ⟨kořen, vzor⟩ ohodnotí pravděpodobností, s jakou se může vyskytnout pro danou skupinu slov. Používá tabulku $\mathtt{P_{PE}}[\mathtt{vzor}, \mathtt{koncovka}]$ pravděpodobnosti výskytu koncovek pro jednotlivé vzory.

Vstup: seznam slov $w = \{w_1, \ldots, w_n\}$, seznam $gp = \{gp_1, \ldots, gp_m\}$ možných dvojic ⟨kořen, vzor⟩ pro slova z w

Výstup: vektor $prob = \{prob_1, \ldots, prob_m\}$ – pravděpodobnostní ohodnocení každé dvojice.

```
for all ⟨root, paradigm⟩ ∈ gp do
  for i = 0 to n do
    ending ← w_i ⊖ root {ze slova w_i odřízneme koncovku}
    prob_i ← P_PE[paradigm, ending]
  end for
end for
return(prob)
```

Zakončení kořene slova

Při testování předcházejících dvou metod jsme zjistili, že algoritmus poměrně často chybuje při rozlišování dokonavých a nedokonavých sloves. Aby také ne, podle otisků ani podle koncovek se většinou tyto dvě třídy sloves nerozliší[26].

Například pro slova `šít`, `přišít` a `zašít` je konec slova identický, a přesto je slovo `šít` nedokonavé a `přišít` a `zašít` jsou slova dokonavá.

Metoda, která se přímo nabízí, je sledovat četnosti výskytu různých předpon. Její výsledky nicméně nebyly uspokojivé — většina předpon nevytváří jednoznačnou vazbu s dokonavými slovesy, jak naznačuje tabulka 2.3. Navíc tato metoda by byla vázaná pouze na několik vzorů pravidelných dokonavých a nedokonavých sloves. Při analýze problému ale zaujme jiná vlast-

dokonavá	nedokonavá
při-ší-t	při-šív-at
za-ší-t	za-šív-at
do-ší-t	do-šív-at

Tabulka 2.3: Předpony dokonavých a nedokonavých sloves

nost těchto slov, zřejmá i z uvedené tabulky. Ukázalo se, že posledních několik písmen kořene slova je poměrně úspěšný indikátor toho, zda je sloveso dokonavé, či nikoli. Proto jsme zavedli metodu, která přiřazuje pravděpodobnost na základě výskytu posledních dvou písmen kořene daného slova.

Opět je pro trénování použita metoda MLE, pro vyhlazování Laplaceův zákon a metoda zlatého řezu. Vznikne tabulka pravděpodobnosti pro daný vzor a koncovku kořene $\mathtt{P}_{\mathtt{RE}}[\mathtt{vzor}, \mathtt{koncovka}]$. Postup vyhodnocování s pomocí této tabulky je popsán jako algoritmus 6.

2.4.4 Metody využívající kontext

Kromě metod založených na zkoumání slova můžeme využít i informace, které nám dává okolní text. Podobně postupuje i člověk, pokud čte text, ve kterém jsou pro něj neznámá slova — všímá si kontextu a podle něj se snaží domýšlet význam neznámých slov. Nás sice bude zajímat pouze vzor, nikoliv význam slova, přesto nám informace z okolí slova může významně pomoci.

Hezkým příkladem je shoda adjektiva se substantivem v rodě a pádu: vyskytují-li se ve větě bezprostředně za sebou slova `viděl vousatého` a za nimi substantivum, bude to velmi pravděpodobně substantivum rodu

[26]Dokonavé vzory se liší od nedokonavých pouze v existenci koncovek pro přechodník minulý resp. přítomný.

Algoritmus 6 ScoreRE: Každou dvojici ⟨kořen, vzor⟩ ohodnotí pravděpodobností, s jakou se dvojice může vyskytnout pro danou skupinu slov. Používá tabulku $\mathtt{P_{RE}}[\mathtt{vzor}, \mathtt{koncovka}]$ pravděpodobností výskytu vzorů pro zakončení kořene.

Vstup: seznam slov $w = \{w_1, \ldots, w_n\}$, seznam $gp = \{gp_1, \ldots, gp_m\}$ možných dvojic ⟨kořen, vzor⟩ pro slova z w

Výstup: vektor $prob = \{prob_1, \ldots, prob_m\}$ – pravděpodobnostní ohodnocení každé dvojice.

for all ⟨root, paradigm⟩ $\in gp$ **do**
  $root\_ending \leftarrow substring(root, length(root) - 2, 2)$
  $prob_i \leftarrow \mathtt{P_{RE}}[paradigm, root\_ending]$
end for

mužského v akuzativu, tedy se shodnými kategoriemi pádu a rodu, jako má předcházející adjektivum.

n-gramový model

Na zachycení okolí (kontextu) slova použijeme n-gramový model. Ačkoli není pro češtinu, jazyk s poměrně volnou stavbou věty, příliš vhodný, je jednoduchý, srozumitelný a pro náš účel postačující.

Cílem modelu je odhadnout, jak bude vypadat aktuální slovo (w_n) na základě historie, neboli slov jemu předcházejících ($w_1, \ldots, w_{n-1}$):

$$P(w_n|w_1 \ldots w_{n-1})$$

Principem n-gramového modelu je předpoklad lokální závislosti (Markovův předpoklad): slovo je ovlivněno pouze $n-1$ předchozími slovy. V takovém modelu tedy záleží vždy jen na $n-1$ posledních slovech, starší slova nemají na aktuální žádný vliv.

Tento předpoklad nám umožní vytvořit na skupinách slov třídy ekvivalence, kdy v každé třídě jsou slova, která mají stejnou historii. V trénovací části počítáme pro každou třídu ekvivalence (historii) četnosti výskytu slov a z nich pak odvozujeme pravděpodobnosti metodou maximální věrohodnosti (MLE):

$$P_{MLE}(w_n|w_1 \ldots w_{n-1}) = \frac{C(w_1 \ldots w_n)}{C(w_1 \ldots w_{n-1})}$$

Pro náš účel budeme používat n-gramy složené nikoliv ze slov, ale z morfologických tagů. Budeme se tedy z historie snažit předpovědět, jaký tag

bude na místě neznámého slova[27]. Zde je dobré podotknout, že tag samotný nám k určení vzoru přímo nestačí, ale pomůže nám (společně s koncovkou) některé vzory vyloučit a jiné zvýhodnit.

Kromě snadné aplikovatelnosti je výhodou použití tagů oproti slovům také to, že máme k dispozici podstatně „hustší" data, protože stejné tvary různých slov jsou reprezentovány jedním tagem. Některá slova může být přesto vhodné zahrnout mezi n-gramy, protože se vyskytují srovnatelně často jako některé tagy a mohou významně ovlivňovat historii[28].

Poslední otázka, kterou si musíme rozmyslet, je, jak velkou historii budeme potřebovat[29]. Zatím jsme vždy uvažovali jen o předcházejících slovech (historii), ale nic nám v principu nebrání podívat se i do „budoucnosti" na následující slova[30]. V kapitole 3 provedeme experimenty s různým nastavením n-gramů pro předchozí i pro následující tagy.

Vyvažování: lineární interpolace

Stejně jako v předchozích metodách se i zde setkáváme s problémem řídkých dat. Některé n-gramy jsme v trénovacích datech neviděli, přestože jsou možné. Metoda MLE přiřazuje takovým n-gramům pravděpodobnost 0, stejně jako n-gramům nemožným.

Čím větší n, tím větší detaily nám model umožňuje zachytit, ale tím méně spolehlivé informace dostaneme (z důvodu řídkých dat). Proto bychom chtěli nějakým způsobem kombinovat informace z modelů vyšších a nižších řádů. Například pro trigramový model můžeme použít trigramy (s historií velikosti 2), bigramy, unigramy (s historií velikosti 0) a rovnoměrné rozdělení. Jednou z možností kombinování různých modelů je *lineární interpolace*.

Jak název napovídá, výsledná pravděpodobnost vznikne jako lineární kombinace pravděpodobností, které jsou výsledkem jednotlivých metod. Váhy metod jsou pak zohledněny pomocí koeficientů λ_i:

$$\lambda = (\lambda_0, \lambda_1, \ldots, \lambda_n), \sum_{i=0}^{n} \lambda_i = 1$$

$$p'_\lambda(w_i|w_{i-n+1}, \ldots, w_{i-1}) = \frac{\lambda_0}{|V|} + \sum_{k=1}^{n} \lambda_k p_k(w_i|w_{i-k+1}, w_{i-1})$$

[27] Motivace je jasná z uvedeného příkladu. Navíc jsme v rané verzi programu vyzkoušeli poměrně účinný filtr na tagy, který vždy vyžadoval shodu po sobě následujících adjektiv a substantiv. n-gramy jsou vlastně jeho zobecněním.

[28] Jedná se především o tvary slovesa být, předložky, spojky atd.

[29] Neboli určit hodnotu n ve slově n-gram.

[30] To, že se často používají pouze předcházející slova, vychází především z aplikací, kde se podobné metody používají. Pokud bychom například uvažovali o rozpoznávání řeči, můžeme se pouze dívat do historie na to, co řečník pronesl, nikoli na to, co pronese.

Pro nejčastěji používané trigramy je vzoreček následující:

$$p'_\lambda(w_i|w_{i-2},w_{i-1}) = \\ = \lambda_3 p_3(w_i|w_{i-2},w_{i-1}) + \lambda_2 p_2(w_i|w_{i-1}) + \lambda_1 p_1(w_i) + \frac{\lambda_0}{|V|}$$

K nalezení optimálních koeficientů použijeme opět optimalizační algoritmus na odložených datech, tentokrát EM algoritmus. Z trénovací fáze máme pro každou historii určeny koeficienty $p_1,\ldots,p_n$. Naším cílem je maximalizovat pravděpodobnost odložených dat H neboli minimalizovat entropii E (h_i je historie délky $i-1$, H jsou odložená data):

$$E = -(1/|H|) \sum_{i=1,\ldots,|H|} log_2(p'_\lambda(w_i|h_i)$$

EM algoritmus probíhá v několika krocích:

- Zvolíme vektor λ libovolně tak, že $\lambda_i > 0, \sum_{i=0}^{n} \lambda_i = 1$.
- Průchodem přes odložená data spočítáme očekávané četnosti pro λ_i:

$$c(\lambda_j) = \sum_{i=1,\ldots,|H|} \frac{\lambda_j p_j(w_i|h_i)}{p'_\lambda(w_i|h_i)}$$

- Spočítáme nový vektor λ' založený na očekávaných četnostech:

$$\lambda'_j = \frac{c(\lambda_j)}{\sum_{k=0}^{n} c(\lambda_k)}$$

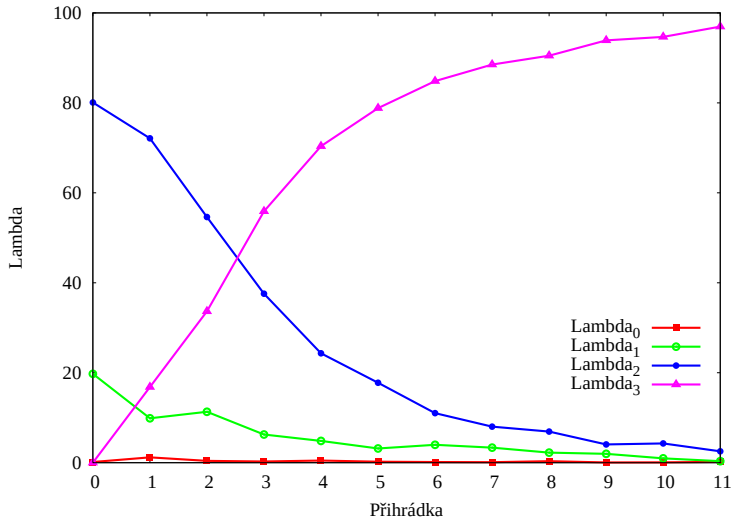
- Poslední dva kroky provádíme iterativně, dokud není $|\lambda_j - \lambda'_j| < \epsilon$ pro předem zvolenou konstantu ϵ.

Přihrádkové vyhlazování

EM algoritmus nám umožňuje kombinovat různě spolehlivé metody tak, aby co nejlépe odpovídaly odloženým datům. Jednotlivým i-gramům (0-gramům (rovnoměrné rozdělení), unigramům, bigramům atd.) přiřadí koeficient odpovídající jejich váze pro daná data. Například pro trigramový model může vypadat výsledek EM algoritmu takto:

$$\lambda = (\lambda_0, \lambda_1, \lambda_2, \lambda_3) = (0.00394, 0.10546, 0.52202, 0.36859)$$

Vidíme, že největší váhu získaly bigramy, následované trigramy, unigramy a rovnoměrným rozdělením. Chtěli bychom, aby trigramy měly



Obrázek 2.2: Koeficienty λ_i pro trigramový model s přihrádkovým vyhlazováním. Čím vyšší číslo přihrádky, tím větší je četnost historie trigramu.

váhu co nejvyšší (umožňují pojmout více detailů), ale EM algoritmus je znevýhodnil oproti bigramům. Důvodem je (jak jinak) řídkost dat pro trigramový model: na mnoha místech odložených dat trigramy dávaly nulovou pravděpodobnost[31]. Chtěli bychom náš model vylepšit tak, aby ty trigramy, které jsou spolehlivé, byly ve výpočtu vah zvýhodněny oproti méně spolehlivým.

Jak ale rozhodnout, které jsou spolehlivé a které nikoli? Nabízí se jednoduchá hypotéza: čím četnější historie (čím častěji jsme ji viděli), s tím větší spolehlivostí můžeme určit, jaké slovo bude následovat. Rozdělíme tedy n-gramy podle četnosti historie na třídy a spočítáme koeficienty lineární interpolace zvlášť pro každou třídu. Tomuto postupu se říká *přihrádkové vyhlazování*[32].

Rozdělíme-li n-gramy podle historie na m přihrádek, bude výsledkem EM algoritmu $m + 1$ vektorů λ, pro každou přihrádku jeden a jeden navíc pro neviděné historie. Možný výsledek je graficky znázorněn na obrázku 2.2. Je zřejmé, že hypotéza o spolehlivějších n-gramech pro častější historie je pravdivá.

[31] A tudíž získaly nízký součet očekávaných četností v EM algoritmu.

[32] V literatuře se často setkáváme s pojmem Bucketed Smoothing.

2.4.5 Kombinace metod

Jednotlivé metody, ačkoli jsou v některých ohledech velmi podobné, využívají různých způsobů určení nejpravděpodobnějšího vzoru. Žádná z nich však není tak dokonalá, aby zaručovala dostatečnou úspěšnost sama o sobě[33], ale můžeme se jistě pokusit výsledky jednotlivých metod nějakým způsobem zkombinovat.

Každá z metod nám dává pravděpodobnost přiřazení jednoho z možných vzorů k danému neznámému slovu. Nabízí se tedy možnost jednoduše kombinovat jednotlivá pravděpodobnostní rozdělení. Pokud budeme předpokládat, že rozdělení jsou vzájemně nezávislá, můžeme použít vzorec:

$$\begin{aligned} &Score(\langle\text{kořen, vzor}\rangle|w,h) = \\ &\quad = P_{PS}(\langle\text{kořen, vzor}\rangle|w) * P_{PE}(\langle\text{kořen, vzor}\rangle|w) \\ &\quad * P_{RE}(\langle\text{kořen, vzor}\rangle|w) * P_{NG}(\langle\text{kořen, vzor}\rangle|h) \end{aligned}$$

kde $P_{PS}, P_{PE}, P_{RE}, P_{NG}$ jsou pravděpodobnosti přiřazené nekontextovými metodami ,,nejpravděpodobnější ze vzorů“, ,,koncovky vzorů“, ,,zakončení kořene“ a kontextovou metodou ,,ngramů“.

Ke kombinaci metod se ještě vrátíme v oddílu 3.5.1, kde otestujeme účinnost jednotlivých metod a jejich kombinací.

2.4.6 Přiřazení slov do skupin

V předchozích odstavcích jsme se pokusili představit několik jednoduchých metod, jak na základě analýzy neznámého slova a bezprostředního okolí co nejlépe určit jeho vzor a kořen. Všechny popsané metody ale pracovaly s každým slovem zvlášť, což jistě nemusí být ideální. Máme-li ve vstupních datech různé tvary jednoho slova, můžeme lépe usoudit, o který vzor se jedná. Dokonce i stejné tvary slov nám mohou pomoci, protože se vyskytují v různých kontextech.

V ideálním případě nám takové sloučení slov výrazně pomůže, jako v případě tvarů `vášeň` a `vášní`, které sdílí jediný vzor `ps1e`.

Přiřadit správně do skupin slova se stejným lemmatem je poměrně obtížný úkol. Hledáme vlastně rozklad množiny neznámých slov na třídy ekvivalence podle příslušnosti k lemmatu. Takových rozdělení může být velmi mnoho — stejně jako všech podmnožin dané množiny — 2^n, tedy algoritmus, který by každý takový systém podmnožin prověřil bude mít složitost[34] alespoň 2^n. Už

[33] K účinnosti jednotlivých metod je ještě vrátíme v kapitole 3.

[34] Používáme notaci $O(f(n))$, která vyjadřuje, že algoritmus má asymptotickou složitost jako funkce $f(n)$.

pro poměrně malá n přestává být tato úloha v praxi řešitelná. Musíme najít nějaký postup, jak počet možností zredukovat.

Podíváme-li se na slova: `kořenáček, kalhoty, postrkovaný, výrovi, poslouchal, postrkoval, prchat, poslední`, je každému čtenáři zřejmé, že kromě slov `postrkoval` a `postrkovaný` má každé slovo jiné lemma. Jak to člověk pozná? Určitě nezkouší vytvořit všechny podmnožiny daných slov, vidí hned, že `kořenáček` a `výrovi` jsou různá slova, protože začínají úplně jiným písmenem (nemají společný prefix). Na druhou stranu u slov `poslední` a `poslouchal` už se možná někdo na okamžik zamyslí.

Tím, že budeme požadovat stejný prefix u tříd ekvivalence, můžeme několikanásobně zredukovat složitost problému. Je otázka, jak dlouhý by prefix měl být. Jako dobrý kompromis se jeví velikost dvou písmen[35]

Přestože jsme výrazně snížili počet možných podmnožin, stále se může stát, že slov se stejným prefixem bude mnoho (v praxi desítky) a počet podmnožin bude stále ještě příliš velký. Navíc potřebujeme každé rozdělení ohodnotit a zjistit, zda se jedná opravdu o rozklad podle příslušnosti k lemmatům, či nikoli, což však zdaleka není jednoduchý ani rychlý úkol.

Abychom se z této zapeklité situace dostali, nabízí se možnost použít hladový algoritmus, který sice nemusí vždy najít optimální rozdělení na podmnožiny, zato bude pracovat v polynomiálním čase. Algoritmus používá funkci $Score(w_1, \ldots, w_n)$, která přiřadí každé skupině slov číslo, jak moc je pravděpodobné, že všechna slova ve skupině sdílí jeden vzor a lemma.

Algoritmus nejprve vytvoří pro každé slovo zvláštní třídu a pro každé dvě třídy vypočítá hodnotu funkce *Score*. Pokud je tato hodnota pro některé dvě třídy vyšší než předem daná prahová hodnota, stávají se třídy kandidátem na sloučení. Z takovýchto kandidátských dvojic vybereme tu, která dosahuje maximální hodnoty *Score*, a třídy sloučíme. Dále je nutné přepočítat funkci *Score* nově vzniklé třídy s ostatními. Algoritmus končí ve chvíli, kdy neexistuje dvojice tříd, která by byla tvořena kandidáty na sloučení. Postup je formálně popsán jako algoritmus 7, prahová hodnota je označena jako *Cutoff*.

Funkce Score

Zatím jsme předpokládali existenci funkce $Score(w_1, \ldots, w_n)$, ale neukázali jsme, jak takovou funkci zkonstruovat. Zamysleme se nad tím, jaké vlastnosti po ní požadujeme. Chceme, aby funkce zhodnotila skupinu slov, zda může mít společné lemma, neboli zda jsou to tvary odvozené od jednoho slova. Jak víme, tvary se odvozují pomocí vzorů a koncovek, takže vlastně chceme, aby funkce odpovídala tomu, zda slova mohou mít společný kořen a vzor.

[35] Existují slova, jejichž tvary mají různá první dvě písmena (např. slova `úzkých`, `užších`, což jsou tvary slova `úzký`), ale to jsou spíše výjimky.

Algoritmus 7 CreateWordGroups: Rozdělí slova na skupiny, v každé skupině jsou slova odvozená od stejného lemmatu.

Vstup: slova $w = \{w_1, \ldots, w_n\}$
Výstup: skupiny slov $g = \{g_1, \ldots, g_m\}$
  {Vytvoříme n skupin, každé slovo má vlastní skupinu}
  for all $w_i \in \{w_1, \ldots, w_n\}$ **do**
    $gw_i \leftarrow w_i$
    $gp_i \leftarrow GetPossibleWordParadigms(w_i)$
  end for
  {Spočítáme skóre pro průniky vzorů}
  for all $i \in \{1, \ldots, n\}, j \in \{1, \ldots, n\}, i < j$ **do**
    $S_{i,j} \leftarrow max_{prob}(Score(gw_i \cup gw_j, gp_i \cap gp_j))$
  end for
  while $\exists_{i,j}(S_{i,j} > Cutoff)$ **do**
    {Najdeme kandidáta na sloučení}
    $(i, j) \leftarrow argmax(S_{i,j})$
    {Sloučíme skupiny i a j}
    $gp_i \leftarrow gp_i \cap gp_j$
    $gp_j \leftarrow \emptyset$
    $gw_i \leftarrow gw_i \cup gw_j$
    $gw_j \leftarrow \emptyset$
    {Přepočítáme skóre pro novou skupinu g_i}
    for $k = 1$ **to** n **do**
      if $k < i$ **then**
        $S_{k,i} \leftarrow max_{prob}(Score(gp_i \cap gp_k))$
      else if $k > i$ **then**
        $S_{i,k} \leftarrow max_{prob}(Score(gp_i \cap gp_k))$
      end if
    end for
  end while
  {Vypustíme prázdné skupiny}
  for $i = 1$ **to** n **do**
    if $gw_i \neq \emptyset$ **then**
      $g_i \leftarrow gw_i$
    end if
  end for
  return(g)

Nedal by se pro tento účel použít předpis, který jsme vytvořili v oddílu 2.4.5? Funkce ohodnotí každý ze společných vzorů pravděpodobností, s jakou je vzor pro dané slovo relevantní. Pro každou hodnocenou skupinu slov vybereme vzor s maximální pravděpodobností a budeme mít za to, že s touto pravděpodobností patří skupině slov daný vzor a kořen, a tudíž slova sdílí společné lemma.

Stačí nám pouze rozšířit existující algoritmy pro více slov. Budeme-li předpokládat, že přiřazení vzorů jednotlivým slovům jsou jevy nezávislé, pak výslednou pravděpodobnost dvojice $P(\langle\text{kořen, vzor}\rangle)$ pro skupinu slov získáme vynásobením pravděpodobností daného vzoru všech slov $w_1, \ldots, w_n$ ze skupiny. Maximum z těchto pravděpodobností budeme pokládat za hodnotu funkce $Score_{max}$, kterou použijeme jako ohodnocující funkci.

$$\begin{aligned} &Score_{max}(w_1, h_1, \ldots, w_n, h_n) = \\ &\quad = \; max_{\langle\text{kořen, vzor}\rangle} \prod_{i=1}^{n} Score(\langle\text{kořen, vzor}\rangle | w_i, h_i) \end{aligned}$$

Problém popsaného postupu je, že s narůstajícím počtem slov ve skupině pravděpodobnost připojení dalšího slova klesá[36]. Řešením je upravit funkci $Score_{max}$ tak, aby byla pokud možno co nejméně ovlivněna počtem slov, která jsou ve skupině. Úprava, která se nabízí, je použít odmocninu n-tého řádu[37] pro n slov.

$$Score'_{max}(w_1, h_1, \ldots, w_n, h_n) = \sqrt[n]{Score_{max}(w_1, h_1, \ldots, w_n, h_n)}$$

Stále se ovšem může stát, že slova nejsou tvary jednoho lemmatu, a přesto existuje vzor, který má nenulovou pravděpodobnost (zvláště po aplikaci vyhlazování pro jednotlivé metody). S tím se pokusíme alespoň částečně vypořádat nastavením parametru *Cutoff*.

Parametr Cutoff

Tento parametr vyjadřuje, jaká hodnota funkce $Score'_{max}$ je pro nás ještě přijatelná, abychom pokládali skupinu slov za tvary jednoho lemmatu. Hodnotu parametru určíme maximalizací metodou zlatého řezu, postup je následující:

- Stanovíme počáteční hodnotu: $0 < Cutoff < 1$.

[36] Jedná se o podobný problém jako při aplikaci n-gramů: delší věty mají menší pravděpodobnost výskytu než věty kratší.

[37] Vlastně se jedná o geometrický průměr.

- Spustíme ohodnocovací algoritmus, vyhodnotíme úspěšnost.
- Metodou zlatého řezu rozhodneme, jak nastavit hodnotu *Cutoff* pro další iteraci.
- Předchozí dva body provádíme, dokud není interval s hledaným maximem dostatečně malý (případně algoritmus ukončíme po předem daném maximálním počtu iterací).

Časová složitost algoritmu

V tomto oddílu budeme analyzovat složitost algoritmu 7 v závislosti na počtu slov n. Rozebereme časové a prostorové požadavky jednotlivých částí algoritmu a odhadneme celkovou asymptotickou složitost.

Pro efektivní implementaci budeme potřebovat datovou strukturu halda (heap), která umožňuje provádět operace $insert()$, $delete()$ v čase $O(\log_2 n)$ a operaci zjištění maxima $max()$ v čase $O(1)$. Bude se nám hodit pro rychlé nalezení maxima ohodnocovací funkce, abychom věděli, které dvě skupiny máme v daném kroku sloučit.

Algoritmus nejprve přidělí každému slovu jednu skupinu. Poté spočítá hodnotu funkce $Score'_{max}$ pro každou dvojici skupin. Budeme předpokládat, že tato funkce závisí lineárně na počtu slov ve skupině. Výsledné hodnoty si uložíme do tabulky velikosti $n \times n$ a současně je budeme udržovat v haldě kvůli rychlému vyhledávání maxima. Celkově tedy provedeme $2*n^2$ kroků (výpočet ohodnocovací funkce) a na uložení do haldy (operaci $insert()$) potřebujeme $O(\log_2 n)$ kroků. Dohromady tedy pro inicializační fázi dostaneme složitost $(2 * n^2 * \log_2 n) = O(n^2 * log_2 n)$.

V iterační fázi nejprve sloučíme dvě skupiny slov (konstantní složitost) a poté musíme přepočítat ohodnocovací funkci nově vzniklé skupiny se všemi ostatními (maximálně n) skupinami. Jelikož složitost výpočtu funkce $Score'_{max}$ je úměrná počtu slov ve skupině n, dostáváme maximálně $n * n$ kroků. Práce s haldou zahrnuje smazání $2 * n$ záznamů o sloučených skupinách a přidání n nových záznamů, dohromady tedy $3 * n * \log_2 n$. Celková složitost jedné iterace je tedy $(n^2 + 3 * n * log_2 n) = O(n^2)$. Těchto iterací se provede maximálně n (při každé dojde ke sloučení dvou skupin, jejich počet tedy poklesne o 1). Dostáváme se tedy k celkové složitosti[38] algoritmu, která je $\mathbf{O(n^3)}$.

[38] Na první pohled se může zdát, že pro určení maxima není potřeba halda, ale stačil by libovolný třídicí algoritmus, který pracuje s kvadratickou složitostí (neboť v každé iteraci algoritmu 7 strávíme čas $O(n^2)$). Je ovšem nutné si uvědomit, že vybíráme maximum z n^2 hodnot, takže třídicí algoritmus by potřeboval $O(n^2 * \log_2 n^2)$ kroků, což je více, než kolik má uvedený algoritmus s použitím haldy.

Paměťová náročnost algoritmu je poměrně zřejmá, potřebujeme uschovat $O(n^2)$ hodnot funkce $Score'_{max}$, halda bude mít také velikost $O(n^2)$. Celková paměťová náročnost je tedy $\mathbf{O(n^2)}$.

2.4.7 Předpony nej- a ne-

Zatím jsme taktně mlčeli o předponách. Je zřejmé, že při použitém algoritmu, který umí přiřadit ke stejnému lemmatu pouze slova začínající stejnými dvěma písmeny, narazíme při slučování slov s některou z předpon do skupin na problém: existují-li neznámá slova, z nichž některá obsahují předponu a jiná ne, nebudou nikdy sloučena do jedné skupiny. To sice v principu nevadí (prostě by se dala tato slova vyšetřovat odděleně), ale pokud bychom uměli taková slova sloučit, mohlo by nám to pomoci v určení vzorů. Proto se pokusíme najít cestu, jak sloučení docílit.

Nemůžeme prostě jen odstranit od slov případnou předponu, protože u některých slov se jedná o část kořene. Postupujeme proto následujícím způsobem:

- Od slov zkusíme odstranit každou ze tří možných předpon `nej-`, `ne-` a `nejne-`. Pokud se nám to povede, přidáme všechny tvary slova bez předpon mezi ostatní slova na vstupu (a poznamenáme si, že dané slovo bylo změněno).
- Po vytvoření skupin pro danou předponu se podíváme, zda některá skupina není tvořena jen slovy, ze kterých byla odtržena stejná předpona, je-li tomu tak, skupinu zahodíme. Zabráníme tak vzniku skupin typu: `bozízek`, `bozízkem`, `bozízku`.
- Pokud je naopak celá skupina tvořena slovy, která obsahují všechna stejnou původní předponu, skupinu odstraníme, ale zapamatujeme si ji. Stejně tak si pamatujeme, zda bylo některé ze slov v zapamatované skupině použito i v jiné skupině slov.
- Po zpracování všech ostatních skupin se podíváme na ty, které jsme si zapamatovali. Vyřadíme z nich slova, která byla mezitím použita v jiné skupině, a tyto zapamatované skupiny vyhodnotíme jako libovolné ostatní.

Tímto postupem umožníme každému slovu „přispět“ svým tvarem bez prefixu k jiné skupině slov (pokud slovo opravdu mělo prefix `ne-`) a zároveň pro slova, která jsou vždy použita s předponou, nebudeme vytvářet neexistující tvary[39].

[39]Tvary typu `bozízek` přece jen často vyrobíme, protože vzor `hd1ek` umožňuje použití

2.4.8 Přiřazení lemmat

K určení kompletního slovníkového záznamu pro neznámé slovo nám ještě zbývá k danému vzoru a kořeni získat lemma. Lemma je definováno jako základní tvar, který je společný pro všechny záznamy jednoho slova ve slovníku. Většinou platí, že od jednoho vzoru se lemma tvoří připojením jediné koncovky ke kořeni slova. Bohužel tomu tak není vždy, u vzoru `0` (pro nepravidelné tvary) existuje několik desítek používaných koncovek, ale existují i další vzory, které používají více různých koncovek, například částečný vzor `s2`:

```
uctěn     s2  =uctít
znevážen  s2  =znevážit
sněden    s2  =sníst
```

Nakonec jsme zvolili způsob určování lemmat pomocí seznamu koncovek pro každý vzor[40]. Umožnili jsme také před připojením koncovky odstranit několik znaků z konce kořene slova v závislosti na použitém vzoru. Seznam koncovek jsme získali průchodem morfologického slovníku tak, že jsme pro každý vzor zaznamenali koncovku kořene (s případným odstraněním znaků z konce kořene slova). V seznamu jsme uvedli vždy pouze nejčetnější koncovku. V tabulce 2.4 jsou uvedeny příklady koncovek lemmat pro vzory ,,nůše“.

vzor	koncovka	příklad
`ns1`	`e,0`	*evaluac-e*
`ns2`	`ě,0`	*sukn-ě*
`ns3n`	`ně,0`	*řeky-ně*
`ns3x`	`e,0`	*inkvizic-e*
`ns7`	`e,0`	*housl-e*
`ns07`	`e,0`	*Strašnic-e*
`ns10`	`e,0`	*hubic-e*

Tabulka 2.4: Přehled koncovek lemmat pro vzory ,,nůše“.

předpony `ne-`, přesně v duchu oddílu 2.2.5.

[40] V prvních verzích programu jsme používali algoritmus, který hledal minimální koncovku pro daný vzor. Vybrala se ta, která byla největší podle uspořádání tagů. Nicméně algoritmus nefungoval příliš dobře.

2.4.9 Algoritmus pro určení vzorů

Na závěr kapitoly ještě uvedeme algoritmy, které dovršují celý proces hledání vzorů neznámých slov. Algoritmus 8 najde pro skupinu slov ohodnocení vzorů pravděpodobnostmi výskytu, algoritmus 9 zastřešuje celý problém řešený v této kapitole: jeho úkolem je pro neznámá slova na vstupu navrhnout nejpravděpodobnější vzory. Jedná se vlastně o popis cíle, který jsme si vytyčili na začátku této kapitoly.

Algoritmus 8 EvalWordGroup: najde pro skupinu slov možné dvojice ⟨kořen, vzor⟩ a pro každou z nich určí pravděpodobnost výskytu.

Vstup: skupina slov $w = \{w_1, \ldots, w_n\}$
Výstup: seznam trojic ⟨pravděpodobnost, kořen, vzor⟩
{Zjistíme možné dvojice ⟨kořen, vzor⟩ pro celou skupinu}
$gp \leftarrow GetPossibleWordGroupParadigms(w)$
{Aplikujeme metody pro zjištění pravděpodobnosti jednotlivých možností}
for all $gp_i \in gp$ **do**
  $prob_i \leftarrow Score(w, gp_i)$
  $result_i \leftarrow \langle prob_i, gp_i \rangle$
end for
return$(result)$

Algoritmus 9 AssignParadigms: Každému neznámému slovu ve vstupním textu přiřadí trojici ⟨skóre, kořen, vzor⟩.

Vstup: text ve formátu CSTS obsahující neznámá slova w_i (slova s tagem `X@----------`).

Výstup: pro každé slovo w_i vektor návrhů trojic ⟨skóre, kořen, vzor⟩, které jsou setříděny podle pravděpodobnosti.

{Přečteme trénovací data, vyfiltrujeme neznámá slova}
ReadStatistics()
$i \leftarrow 1$
for all $x \in ReadInputData()$ **do**
  if $Tag(x) =$ "`X@----------`" **then**
    $w_i \leftarrow x$
    $i \leftarrow i + 1$
  end if
end for
{Vytvoříme skupiny}
$g \leftarrow CreateWordGroups(w)$
{Vypočítáme ohodnocení pro jednotlivé skupiny}
for all $g_i \in g$ **do**
  $prob_{g_i} \leftarrow EvalWordGroup(g_i)$
end for
{Vypíšeme ohodnocení pro každé slovo}
for all $g_i \in g$ **do**
  for all $w_i \in g_i$ **do**
    print $w_i, prob_{g_i}$
  end for
end for

Kapitola 3

Experimenty a výsledky

V této kapitole nejprve popíšeme data, která budeme používat pro trénování a testování modelů, vysvětlíme, jak budeme výsledky ověřovat a zavedeme pojmy přesnost a úplnost tak, jak se používají v oblasti information retrieval. Následovat bude popis experimentů a jejich výsledků.

3.1 Trénovací a testovací data

K určení parametrů metod, které byly popsány v kapitole 2, budeme potřebovat trénovací a odložená data. Stejně tak budeme potřebovat data k testování výsledků nastavení jednotlivých metod a parametrů – testovací data. Máme k dispozici několik kolekcí textů, které se liší svým rozsahem a zpracováním a pro naše účely jsou různě vhodné.

Pražský závislostní korpus

Pražský závislostní korpus [9] je soubor textů, které jsou morfologicky zpracované a ručně (anotátorem) disambiguované[1]. Jedná se o nejkvalitněji zpracované texty, které jsou k dispozici. Jejich nevýhodou je poměrně malá velikost[2] daná nutností ruční anotace. Data pocházejí z deníků Lidové noviny, Mladá fronta Dnes, Českomoravský profit a z časopisu Vesmír.

[1] Data jsou anotovaná i na vyšších úrovních: analytické a tektogramatické.

[2] PDT 1.0 obsahuje 1 673 554 slov ve 111 175 větách.

Český národní korpus

SYN2000[3] [10] je velmi rozsáhlý[4] korpus synchronní češtiny, který je tvořen texty novin a časopisů, knih, poezie atd. Texty jsou zpracovány automatickými nástroji (morfologickou analýzou a disambiguací).

Ostatní data

Další data je možné získat například z webových stránek některých časopisů (Vesmír, ScienceWorld atd.). Jedná se většinou o menší objemy dat, které následně předzpracujeme automatickou morfologií a disambiguací (stejně jako je předzpracován ČNK).

3.2 Fáze trénování

Jak již bylo zmíněno, pro trénování, vyhlazování a testování metod potřebujeme předem upravená data. Různé metody mají rozdílné požadavky na kvalitu a zpracování trénovacích a testovacích dat. K trénování parametrů modelů nekontextových budeme potřebovat texty, které jsou morfologicky označkované a disambiguované, n-gramový model můžeme trénovat buď opět na disambiguovaných textech, nebo na textech pouze morfologicky označkovaných[5].

3.2.1 Nekontextové metody

Nekontextové metody, které přímo určují pravděpodobnost možných vzorů, vyžadují v trénovací fázi, aby pro každé slovo bylo možné jednoznačně určit jeho vzor. Tato podmínka je splněna, pokud je každé slovo v textu opatřeno právě jedním lemmatem a morfologickým tagem, neboli text je morfologicky dismbiguovaný.

V kvalitě přiřazení tagů se uvedené zdroje dat liší: zatímco v PDT jsou značky přiřazeny dvěma nezávislými anotátory a kontrolovány (takže tato data by měla být s minimálními chybami), v ostatních textech (především v ČNK) jsou určeny automaticky tzv. taggerem. Tagger má v současné době úspěšnost kolem 95 %, což znamená, že v průměru pro každé dvacáté slovo

[3] V dalším textu na něj budeme odkazovat zkratkou „ČNK“.

[4] Korpus obsahuje asi 100 miliónů slov.

[5] Morfologicky označkovaná data jsou ta, která mají pro každé slovo přiřazenou množinu lemmat a k nim příslušející množinu možných tagů, disambiguovaná data mají pak pro každé slovo jediné (správné) lemma a tag.

je tag určen nesprávně. To může způsobovat problémy při trénování metod, které se „naučí“ na nesprávné vzory.

Další důležitou otázkou týkající se trénování je, která slova z trénovacích dat vybereme pro učení nekontextových metod. Jelikož naším cílovým objektem jsou slova, která nejsou uvedena v morfologickém slovníku, dá se předpokládat, že se budou vyskytovat poměrně zřídka. Proto zaměříme svoji pozornost především na slova, která mají v trénovacích datech četnost v řádu jednotek. Různé hodnoty četnosti výskytu slov použitých k trénování budou předmětem experimentů v oddílu 3.5.3.

3.2.2 n-gramy

Zajímavější je situace u metody, která využívá kontext – n-gramů. Jako trénovací data můžeme použít libovolný disambiguovaný text, který budeme zpracovávat jako posloupnost tagů.

No tomto místě je dobré si uvědomit, že pro reálná data není jiná možnost než použít tagger, takže musíme počítat s určitým procentem špatně přiřazených tagů. Navíc většina systémů pro disambiguaci umí dobře pracovat pouze se slovy, která jsou morfologicky zpracovaná – jsou jim přiřazeny možné tagy. Jak víme, nás budou v reálném nasazení zajímat právě slova, která naopak ve slovníku nejsou[6]. Jelikož většina taggerů používá pro rozhodování kontext daného slova[7], lze očekávat, že v okolí neznámých slov budou přiřazeny tagy hůře než v jiných částech textu.

Z tohoto důvodu jsme pro srovnání upravili metodu učení n-gramového modelu tak, aby nevyžadoval disambiguovaná data. Místo jednoho tagu použijeme spojené všechny tagy (lexikograficky uspořádané), které morfologická analýza přiřadila danému slovu. Jsou-li například pro slovo `židli` (tvar slova `židle`) přípustné tři různé tagy `NNFS3-----A----`, `NNFS4-----A----` a `NNFS6-----A----`, složíme je dohromady a použijeme je místo jediného tagu, který by vznikl disambiguací:

`NNFS3-----A----NNFS4-----A----NNFS6-----A----`

Pokud má slovo více lemmat, pokládáme je za různá slova a při trénování i vyhlazování počítáme se všemi možnými kombinacemi spojených tagů těchto slov. Máme-li v textu za sebou slova `mám vpředu`, budeme uvažovat následující bigramy[8].

[6] Budou mít přiřazený tag `X@----------`.

[7] Například lze dobře využít shodu v rodě, pádu a čísle pro po sobě následují adjektivum a substantivum.

[8] V tabulce 3.1 jsou uvedena lemmata a k nim příslušné tagy.

slovo	lemma_1	tag_1	lemma_2	tag_2
mám	mít	VB-S---1P-AA---	máma	NNFP2-----A----
vpředu	vpříst	VB-S---1P-AA---	vpředu	Db----------1

Tabulka 3.1: Lemmata a tagy slov `mít` a `vpředu`

```
VB-S---1P-AA---  VB-S---1P-AA---
VB-S---1P-AA---  Db----------1
NNFP2-----A----  VB-S---1P-AA---
NNFP2-----A----  Db----------1
```

Na odložená data, která použijeme k vyhlazení pravděpodobností, máme stejné požadavky jako na data trénovací. Z dat, která jsou určena pro trénování, vybereme asi 10 % za účelem vyhlazování pravděpodobností. Opět pro nekontextové metody je nutné mít data disambiguovaná, pro n-gramy mohou být pouze morfologicky analyzovaná.

3.3 Evaluace

Je nutné nějakým dobře definovaným a dostatečně obecným způsobem vyhodnotit, který algoritmus má lepší výsledky než jiný, případně které nastavení parametrů je optimální. Potřebujeme definovat míru, která nám umožní vyhodnocovat výsledky algoritmů.

Vzniká otázka, který výsledek pro dané slovo považovat za korektní a který už nikoliv. Pokud nás bude zajímat pouze aplikace automatického rozšiřování morfologického slovníku, budeme chtít úspěšnost hodnotit jako poměr počtu slov, kterým algoritmus přiřadí vzor a lemma správně (tedy odpovídají přesně záznamu ve slovníku), ku celkovému počtu slov.

Pokud se však budeme na náš program dívat jako na pomůcku pro anotátora, který přidává záznamy do slovníku, budeme moci připustit jako úspěšné přiřazení i takové, kdy správný vzor je například mezi prvními třemi nabídnutými vzory.

Další problém, který řešíme, je ten, zda daný vzor přesně odpovídá slovníkovému záznamu, nebo se jedná o vzor derivovaný (viz oddíl 2.2.3). Stejně tak se naskýtá otázka, zda budeme za neúspěšné přiřazení považovat pouhé špatné určení předpony (zejména `ne-`, viz oddíl 2.2.5).

Pro zjednodušení budeme dále používat pro jednotlivé druhy výsledků evaluace zkratky: *1* bude znamenat úspěšnost, počítáme-li pouze úplně korektní záznamy, *1–3* je označení pro úspěšnost „jeden z prvních tří“ a *1–3+* znamená jeden z prvních tří s tím, že jako správné počítáme i vzory, které

nejsou kořenem derivačního stromu a ignorujeme případné chyby v přiřazení předpon.

Dále je potřeba zdůraznit, že jako úspěšné přiřazení vzoru pokládáme přiřazení alespoň jednoho správného vzoru, může to být i vzor částečný. To je jeden z důvodů, proč evaluace při použití původních vzorů vychází lépe, než při použití vzorů rozšířených o sloučené částečné vzory (viz také oddíl 2.2.4).

Průměrné počty vzorů, ze kterých vybíráme ten správný, jsou pro původní i kombinované vzory uvedeny v tabulce 3.2.

	vzory původní	vzory kombinované
před rozdělením	106	360
po rozdělení	72	292

Tabulka 3.2: Průměrný počet vzorů před rozdělením na třídy ekvivalence podle lemmat a po něm.

Pro přiřazení lemmatu používáme algoritmus z oddílu 2.4.8 a při evaluaci toto přiřazení nehodnotíme (lemmata slouží především k identifikaci slova ve slovníku, je to jakýsi klíč pro zjištění, které záznamy k sobě patří a které ne).

3.4 Evaluace přiřazení do skupin

Kromě ověření celkové úspěšnosti se můžeme podívat také na úspěšnost jedné z podúloh: přiřazení slov se stejným lemmatem do skupin. Je jistě zajímavé sledovat, jak se úspěšnost rozdělení na skupiny projeví v celkovém výkonu algoritmu. K zachycení, jak dobré je toto rozdělení, nám velmi dobře poslouží veličiny, které se běžně používají pro hodnocení výběru. Jedná se o *přesnost* a *úplnost*.

3.4.1 Přesnost a úplnost

Přesnost a úplnost jsou dvě veličiny, pomocí nichž se běžně hodnotí výkon aplikací, které zpracovávají texty (např. vyhledávání v textu, třídění textů do kategorií, rozdělení textu na části podle různých kritérií apod.). Vyjadřují, jak dobře daný algoritmus pracuje, a umožňují porovnávat jednotlivé algoritmy mezi sebou.

Přesnost (precision, P) je poměr

$$P = \frac{N_{vr}}{N_r}$$

Úplnost (recall, R) je poměr

$$R = \frac{N_{vr}}{N_v}$$

- N_{vr} je počet vrácených relevantních objektů,
- N_r je počet relevantních objektů,
- N_v je počet vrácených objektů.

Většinou platí mezi přesností a úplností vztah nepřímé úměrnosti: čím dosáhneme lepší přesnosti (najdeme více relevantních objektů), tím se nám zhorší úplnost (vrátíme více objektů, které nejsou relevantní).

Pro naše účely bude *přesnost* znamenat poměr správně určených[9] skupin ku počtu korektních skupin. *Úplnost* je potom poměr správně určených skupin ku počtu navržených skupin.

3.5 Experimenty

V této části popíšeme experimenty, které jsme provedli s cílem zjistit možnosti jednotlivých metod a určit nejlepší možnou kombinaci jejich parametrů, případně se pokusit odhadnout, kam by mělo směřovat další úsilí při zlepšování dosavadních výsledků.

Parametrů metod, které mohou mít vliv na celkový výkon algoritmů, je mnoho, proto jsme k testům zvolili jen některé kombinace hodnot. Ukazuje se, že většina parametrů je do jisté míry nezávislá na ostatních, takže můžeme optimalizovat jejich hodnoty bez ohledu na ostatní.

Tabulka 3.3 udává nastavení parametrů[10] pro provedené testy, nebude-li explicitně uvedeno jinak. Dodejme ještě, že pracujeme s novými, složenými vzory slov.

ngramy	buckets	hifreq	metody	min/max četnost
1:1	10	0	všechny	1/5

Tabulka 3.3: Implicitní nastavení parametrů.

[9]Správně určené jsou ty skupiny, které obsahují všechna slova se stejným lemmatem a neobsahují žádné slovo navíc.

[10]Počet přihrádek je uveden jako buckets, počet nejfrekventovanějších slov je uveden jako hifreq.

Většinu experimentů jsme prováděli na textech z PDT, některé také na části ČNK. Počty slov, která se vyskytovala v jednotlivých částech dat, jsou zachyceny v tabulce 3.4.

data	trénovací	odložená	testovací$_1$	testovací$_2$
PDT	1 129 254	100 594	108 816	105 691
ČNK	12 720 281	2 199 169	1 473 010	1 452 034

Tabulka 3.4: Počty slov v PDT a ČNK pro jednotlivé fáze. Testovací data jsou rozdělena na dvě části, první slouží k optimalizaci parametru *Cutoff*, druhou pak budeme používat na celkové otestování výsledku daného nastavení parametrů.

Při použití textů PDT jsme nejprve označili vybraná slova jako neznámá (tagem `X@----------`) a poté automaticky disambiguovali pomocí taggeru [1]. Tato data budeme dále značit $\texttt{data}_{\texttt{TG}}$. Některé testy jsme prováděli i bez disambiguace a pro srovnání jsme vykonali také testy na ručně disambiguovaných datech (PDT). Níže v textu je budeme značit $\texttt{data}_{\texttt{MO}}$, nebo $\texttt{data}_{\texttt{PDT}}$. Pokud neuvedeme jinak, budeme používat $\texttt{data}_{\texttt{TG}}$.

Parametr Cutoff

Parametr *Cutoff* slouží k nastavení prahu (citlivosti) pro přiřazení slov do skupin se stejným lemmatem. Přesný algoritmus je popsán v sekci 2.4.6, zde se pouze zmíníme, jak optimální hodnotu hledáme.

Optimální nastavení tohoto parametru je velmi důležité pro správné přiřazení skupin a tudíž i pro celkovou úspěšnost algoritmu. Přiřadíme-li totiž dvě slova, která mají odlišné lemma do stejné skupiny, téměř jistě vybereme špatný vzor. Opačný případ, kdy dvě slova se stejným lemmatem zůstanou každé ve svojí skupině, nám tolik nevadí, pouze se připravujeme o hodnotné informace k nalezení správného vzoru. Jak ale určit přesnou hodnotu?

K nalezení optimální hodnoty nám poslouží část testovacích dat. Nejdříve na první části určíme hodnotu parametru *Cutoff* pomocí optimalizace metodou zlatého řezu a teprve poté provedeme evaluaci na druhé části testovacích dat. Jelikož optimální hodnoty nastavení parametru se mohou lišit pro jednotlivá nastavení ostatních parametrů (ovlivňující pravděpodobnosti vzorů), musíme optimalizační algoritmus spustit pro každé jejich nastavení samostatně. Počet iterací jsme omezili na 30.

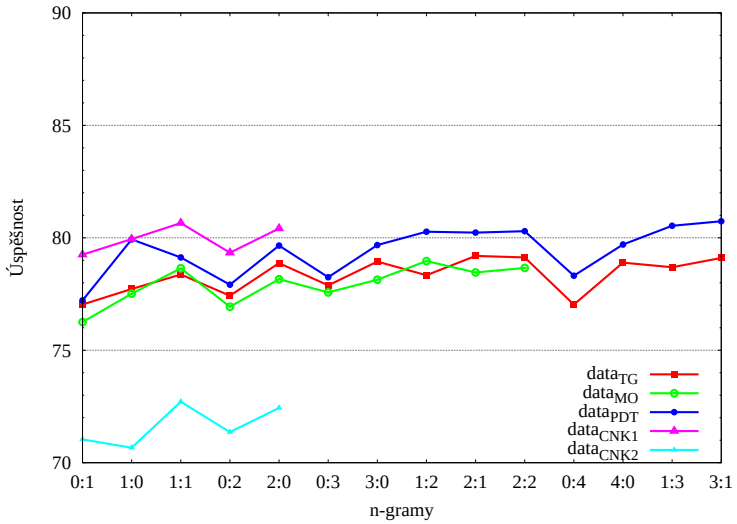
3.5.1 Úspěšnost jednotlivých metod

Nyní můžeme otestovat úspěšnost jednotlivých metod a splnit tak slib, který jsme dali v oddílu 2.4.5. Jednotlivé metody a jejich kombinace jsme použili vždy pro přiřazení slov do skupin i pro finální hodnocení vzorů. V tabulce 3.5 jsou uvedeny výsledky pro nové (složené) vzory, i pro vzory původní. Pro zajímavost uvádíme i přesnost a úplnost přiřazení slov do skupin.

metoda	skupiny slov		výsledek		
	přesnost	úplnost	1	1–3	1–3 +
nové (složené) vzory					
PE	95.800	96.954	14.575	26.943	30.207
RE	95.800	96.954	34.897	62.483	69.517
PS	95.590	96.673	46.046	66.989	72.782
NG	95.870	96.990	20.506	39.724	43.448
PS+NG	75.009	57.101	44.736	66.207	71.494
PE+PS+NG	95.625	94.566	45.287	66.345	71.793
RE+PS+NG	93.875	89.699	**48.782**	**72.713**	**79.333**
PE+RE+PS	95.240	93.185	47.379	69.195	76.184
PE+RE+PS+NG	94.925	91.653	47.310	71.195	78.368
původní vzory					
PE	96.570	96.637	16.116	31.113	31.672
RE	96.640	96.708	37.657	67.816	69.422
PS	94.435	92.082	51.863	74.034	75.827
NG	96.570	96.637	30.904	54.891	56.125
PS+NG	96.640	96.606	54.308	77.550	78.784
PE+PS+NG	96.745	96.006	50.140	75.361	76.735
RE+PS+NG	96.045	93.429	**55.310**	**82.091**	**83.768**
PE+RE+PS	96.605	95.502	51.816	78.295	80.298
PE+RE+PS+NG	94.400	89.721	49.837	78.878	80.531

Tabulka 3.5: Úspěšnost přiřazení vzoru pro jednotlivé metody.

Zajímavé je, že nejlépe dopadla v obou případech kombinace metod RE+PS+NG. Problém metody PE ,,koncovky vzorů“ je pravděpodobně v přílišném zvýhodnění vzorů s prázdnou koncovkou. Je zřejmé, že nejlepší samostatnou metodou je PS ,,nejpravděpodobnější ze vzorů“.



Obrázek 3.1: Úspěšnost přiřazení vzoru v závislosti na velikosti historie pro různá trénovací a testovací data.

3.5.2 *n*-gramy

Pro *n*-gramy existuje největší množství parametrů, které mohou mít vliv na výslednou úspěšnost. Rozebereme postupně jednotlivé možnosti v jejich nastavení.

Velikost historie

Počet tagů, které bereme jako historii, ovlivňuje nejen výslednou úspěšnost algoritmu, ale také jeho paměťovou a časovou náročnost. Se vzrůstajícím n roste počet různých kombinací tagů exponenciálně. Přestože všechny kombinace se běžně nevyskytují, jejich počet přesto výrazně stoupá.

Vyzkoušeli jsme několik nastavení parametru, a to jak pro předcházející tagy, tak pro následující. U všech testů jsme provedli vyvážení metodou lineární interpolace pomocí EM algoritmu s maximálním počtem kroků omezeným na 100. Testy jsme provedli na datech $\texttt{data}_{\texttt{TG}}$, $\texttt{data}_{\texttt{MO}}$, $\texttt{data}_{\texttt{PDT}}$ a navíc na ČNK. Tato data jsou označena jako $\texttt{data}_{\texttt{CNK1}}$ a $\texttt{data}_{\texttt{CNK2}}$, druhý případ se liší od prvního tím, že pro testy jsou použita data z PDT.

Výsledek testů na $\texttt{data}_{\texttt{TG}}$ je uveden v tabulce 3.6, ostatní výsledky jsou pro porovnání zobrazeny v grafu na obrázku 3.1[11].

Je zřejmé, že úspěšnost je pro různé nastavení historie poměrně vyrov-

[11]Některé hodnoty nejsou v grafu uvedeny (např. historie větší než 2 pro $\texttt{data}_{\texttt{CNK}}$) z důvodu příliš velké paměťové nebo časové náročnosti.

historie		skupiny slov		výsledek		
před	za	přesnost	úplnost	1	1–3	1–3 +
0	1	95.940	96.958	46.828	70.460	77.218
1	0	96.185	97.102	46.644	70.943	77.724
1	1	94.925	91.653	47.310	71.195	78.368
0	2	93.945	93.781	46.992	70.662	77.425
2	0	96.255	94.599	47.264	71.770	78.874
0	3	80.049	85.848	46.881	70.738	77.885
3	0	96.080	94.492	46.851	71.724	78.943
1	2	92.825	90.389	47.668	71.059	78.335
2	1	96.150	94.626	47.563	71.977	**79.195**
2	2	94.085	93.366	47.481	71.922	79.128
0	4	72.209	80.807	46.246	70.025	77.030
4	0	96.080	94.590	46.874	71.885	78.897
1	3	79.524	83.869	**47.817**	71.232	78.690
3	1	96.185	94.890	47.425	**72.161**	79.103

Tabulka 3.6: Úspěšnost přiřazení vzoru v závislosti na počtu tagů v historii n-gramů ($\texttt{data}_{\texttt{TG}}$).

naná, s výjimkou metod, které nepočítají vůbec s minulostí (0:1, 0:2, 0:3, 0:4). S přihlédnutím k paměťové složitosti jako nejlepší nastavení vychází trigramy 2:0.

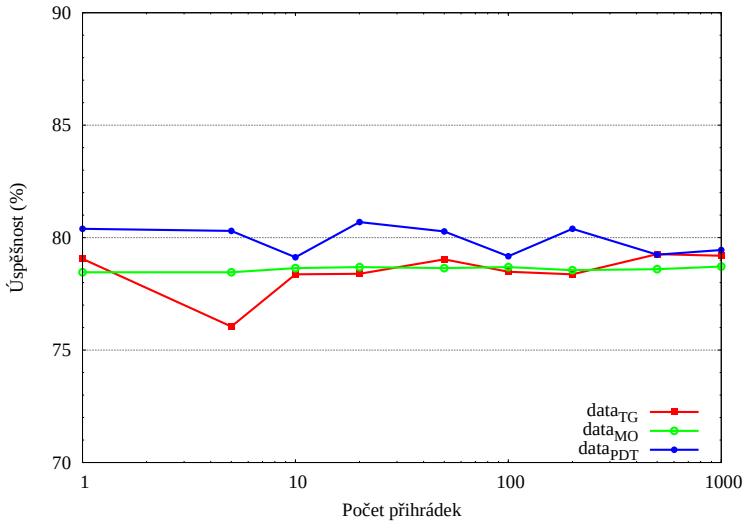
Počet přihrádek a často se vyskytující slova

Další parametry, kterými můžeme ovlivnit úspěšnost metody je počet přihrádek při vyhlazování a počet slov s vysokou četností, se kterými se nakládá jako s tagy. Opět jsme vyzkoušeli několik možných hodnot obou parametrů, výsledky jsou patrné z obrázků 3.2 a 3.3.

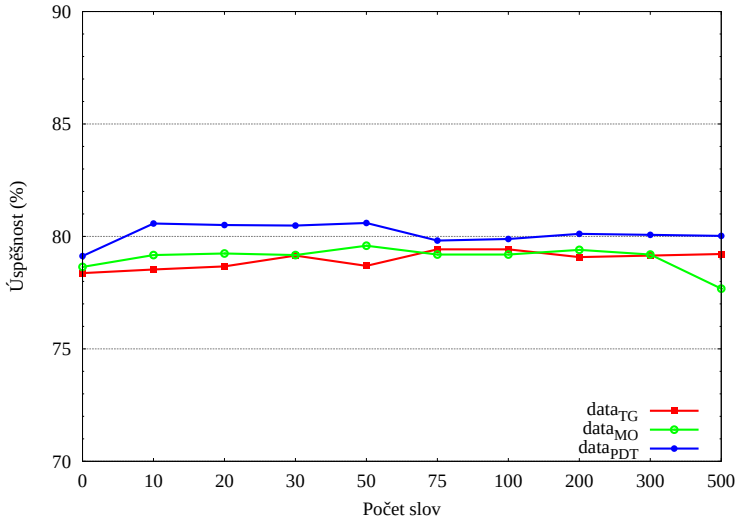
Je zřejmé, že hodnoty těchto parametrů nemají na celkovou úspěšnost příliš velký vliv, snad kromě nastavení příliš malého počtu přihrádek při vyhlazování. Výsledky na disambiguovaných datech jsou v tomto případě téměř stejné, jako na datech pouze morfologicky označkovaných.

3.5.3 Četnost slov

Pokusme se nyní zaměřit na četnosti výskytu slov, kterým chceme přiřadit vzor. Jak jsme viděli v předchozích odstavcích, úspěšnost přiřazení do skupin se pohybuje většinou nad 90 % (přesnost i úplnost), takže můžeme očekávat,



Obrázek 3.2: Úspěšnost přiřazení vzoru pro různý počet přihrádek použitých při vyhlazování.



Obrázek 3.3: Úspěšnost přiřazení vzoru pro různé hodnoty počtu slov s vysokou četností.

že slova, která se vyskytují v textu v několika exemplářích umíme poměrně dobře přiřadit do skupin. Pro skupinu slov se potom určuje vzor snáze, než pro slovo samostatné.

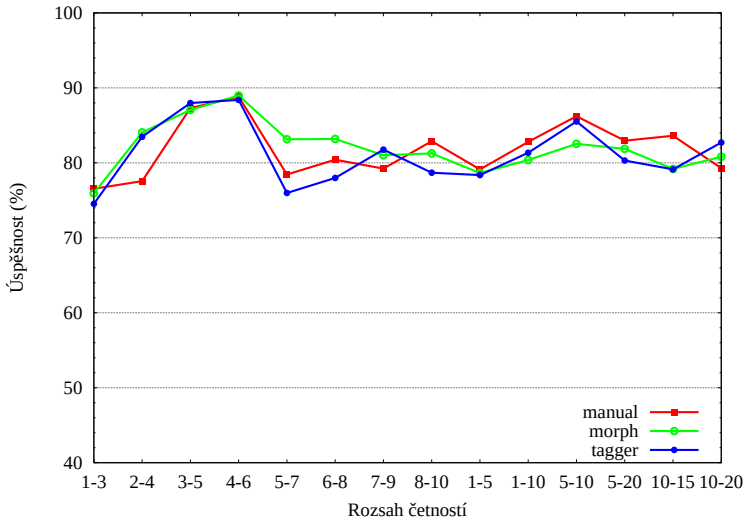
Běžně vybíráme slova s četností v textu od jedné do pěti v očekávání, že takto se co nejlépe přiblížíme četnosti slov, která jsou neznámá a pro která je třeba určit vzor. Můžeme se však zaměřit jen na slova, která se v textu vyskytují s určitou minimální četností. Bohužel, pokud neznáme vzor, neumíme ani rozhodnout, která slova jsou tvary jednoho lemmatu, a tudíž nemůžeme zavést omezení, že od každého slova budeme chtít před určením vzoru vidět alespoň n tvarů.

V dále uvedeném experimentu jsme se pokusili ukázat, jak nám více výskytů slov pomůže k celkové úspěšnosti algoritmu. V trénovací fázi používáme stále rozsah četností 1–5, ale v testovací fázi požadujeme četnosti slov v jiném rozsahu. Výsledky experimentů shrnuje tabulka 3.7, graficky jsou výsledky znázorněny na obrázku 3.4.

četnost		skupiny slov		výsledek		
od	do	přesnost	úplnost	1	1–3	1–3 +
1	3	95.516	92.352	39.177	66.486	74.531
2	4	96.707	94.629	53.007	76.381	83.472
3	5	95.749	92.203	**60.195**	**81.868**	87.973
4	6	90.491	81.944	58.540	81.606	**88.394**
5	7	61.250	35.167	47.652	69.053	75.982
6	8	62.632	34.000	48.578	71.405	77.985
7	9	77.241	56.000	51.504	75.752	81.767
8	10	73.600	54.438	49.809	71.224	78.681
1	5	94.925	91.653	47.310	71.195	78.368
1	10	95.453	93.692	50.197	74.309	81.348
5	10	93.002	87.473	58.021	79.340	85.521
5	20	73.219	48.720	54.882	72.510	80.309
10	15	59.884	29.178	53.690	70.280	79.135
10	20	88.846	78.840	55.750	74.643	82.707

Tabulka 3.7: Úspěšnost přiřazení vzoru v závislosti na četnosti určovaných slov ($\texttt{data}_{\texttt{TG}}$).

Jak je vidět, pokud požadujeme četnost dat alespoň 3, dostáváme se k šedesáti procentům absolutně správně určených slov. Naše strategie může být i taková, že budeme požadovat minimální počet výskytů jednoho tvaru slova (a předpokládat, že přitom se vyskytne i některý z dalších jeho tvarů).



Obrázek 3.4: Úspěšnost přiřazení vzoru v závislosti na četnosti určovaných slov.

Pro vyšší četnosti výskytu už kýžený efekt nepřichází, protože potom slova přestávají odpovídat těm, které jsme viděli v trénovacích datech.

3.5.4 Pravidelná slova

Spíše pro zajímavost se můžeme podívat, jak nám pomůže, zaměříme-li se pouze na pravidelná slova (ta, co mají vzor `0`). Intuice napovídá, že tím bychom si o něco usnadnili práci, neboť odpadají problémy zmíněné v oddílu 2.2.1. Výsledky experimentu nám dávají zapravdu, jak ukazuje tabulka 3.8.

skupiny slov		výsledek		
přesnost	úplnost	1	1–3	1–3 +
96.960	96.210	54.841	75.678	82.907

Tabulka 3.8: Úspěšnost přiřazení vzoru pro pravidelná slova ($\texttt{data}_{\texttt{TG}}$).

3.6 Výsledky

Během ověřování metod jsme provedli přes 250 experimentů, které měly za úkol zjistit vliv různých nastavení na celkový výkon algoritmů. Ověřili jsme si,

že jednotlivé metody pracují poměrně dobře a jejich kombinace dává poměrně dobré výsledky.

Pro metodu n-gramů má na celkové výsledky největší vliv délka historie, ideální pro trigramy je formát 2:0, pro 4-gramy potom 2:1. Naopak počet přihrádek ve vyhlazování a počet slov s velkou četností mají na výsledek přiřazování velmi malý vliv.

Zajímavé jsou také výsledky použití nedisambiguovaných dat v metodě n-gramů, kdy dostáváme výsledky jen o málo horší, než pro automaticky disambiguovaná data. Výraznější je rozdíl pro data z PDT, která jsou disambiguovaná ručně.

Přesvědčili jsme se o možnostech dalšího zlepšování úspěšnosti tím, že bychom nějakým způsobem vyžadovali dostatečný počet (3 a více) výskytů slova (případně tvaru slov) před definitivním rozhodnutím o výsledném vzoru.

Pokud použijeme optimální hodnoty (ngram=2:1, buckets=20, hifreq=20) a podstatně si pomůžeme pomocí požadavku minimální četnosti výskytu mezi 4 a 7, dostaneme výsledek naznačený v tabulce 3.9. Pro srovnání[12] uvádíme také výsledek pro četnosti testovaných slov 1 – 5.

četnost		skupiny slov		výsledek		
od	do	přesnost	úplnost	1	1–3	1–3 +
4	7	87.615	75.794	57.734	81.034	86.207
1	5	96.430	95.197	47.954	72.368	79.563

Tabulka 3.9: Úspěšnost přiřazení vzoru pro $buckets = 20$, $hifreq = 20$, $ngram = 2:1$ ($\texttt{data}_{\texttt{TG}}$).

[12]Zdá se paradoxní, že bylo dosaženo lepšího celkového výsledku při nižší přesnosti a úplnosti přiřazení do skupin. Je to dáno tím, že v případě četností 1 – 5 existuje mnoho skupin pro jedno slovo, které jsou korektně přiřazeny, kdežto u četností 4 – 7, je každé slovo ve skupině s více slovy.

Kapitola 4

Implementace

Součástí práce je kompletní implementace systému pro určování vzorů a lemmat, kterou popíšeme v této kapitole. Nejdříve vysvětlíme, jaké jsme zvolili programovací prostředky, poté popíšeme jednotlivé části implementace.

4.1 Použité programovací prostředky

Každý typ zpracovávané úlohy vyžaduje výběr jiných programovacích prostředků, a to jak z hlediska programátora (programovací jazyk, vývojové prostředí), tak z hlediska uživatele (použité grafické nebo textové prostředí, operační systém atd.). Pokusíme se nastínit, jaké prostředí jsme zvolili a proč.

4.1.1 Zvolený jazyk

První verze sady programů vznikla v programovacím jazyce Perl [11], který se pro implementaci podobných úloh používá poměrně často. Důvodem jsou jeho velmi silné, leč snadno použitelné nástroje pro práci s textem, především asociativní pole a velmi efektivní a přitom jednoduchá práce s regulárními výrazy. Navíc je to jazyk poměrně rozšířený a multiplatformní, takže se pro náš účel výborně hodí. Perl doplňuje obrovské množství knihoven CPAN, které lze s výhodou využít např. pro zpracování vstupních dat nebo pro formátování výstupu.

Protože je Perl jazyk interpretovaný (překládá se do mezikódu, který se poté vykonává), nedosahuje rychlosti programovacích jazyků nízké úrovně[1]. Bylo proto už od začátku vývoje zřejmé, že části kódu, které jsou z hlediska rychlosti důležité, bude nutné napsat v jazyce C/C++.

[1]Máme na mysli především C, C++ a ostatních jazyky, které se překládají přímo do strojového kódu počítače.

Zvolili jsme jazyk C++, zejména kvůli existenci mnoha knihoven a objektovému přístupu k datovým typům (především řetězcům). Verze programů v Perlu je brána jako referenční (je lépe srozumitelná a dokumentovaná), verze v C++ je rychlejší, ale zdrojový kód je méně přehledný.

4.1.2 Porovnání rychlosti programů v Perlu a C++

Pro zajímavost jsme provedli porovnání rychlosti implementace v Perlu a C++, test probíhal na počítači Sun V20z se dvěma procesory AMD Opteron 248 (2200 MHz) a 16 GB RAM. Výsledek je uveden v tabulce 4.1, čas je v minutách a sekundách.

	trénování	vyhlazování	přiřazování
Perl	8m16s	53m48s	10m22s
C++	1m20s	8m28s	2m51s

Tabulka 4.1: Porovnání rychlosti programů v Perlu a C++.

Přepvapením pro nás bylo, že implementace v C++ je rychlejší pouze pěti až osminásobně, ale je do zřejmě dáno také tím, že nejvíce času stráví program při používání asociativních polí, které jsou implementovány obdobně.

Protože ve všech programech často používáme asociativní pole, záleží rychlost programu na kvalitě implementace tohoto datového typu. Otestovali jsme rychlost různých implementací asociativních polí v C++ a jako nejrychlejší se pro naše účely ukázala knihovna STL (resp. její rozšíření).

Programy napsané v Perlu byly odladěny ve verzi 5.8.5, pro programy v C++ byl použit kompilátor GCC 3.3.5 (a knihovna STL). Obě varianty programů jsou psány multiplatformně, vývoj probíhal na OS Linux (Fedora Core release 3).

Pokud v dalším textu uvádíme jména programů končící příponou `.pl`, znamená to, že program je psaný v jazyce Perl. Pokud neuvádíme příponu, jedná se o program, který má svoji „perlivou“ i C++ variantu.

4.1.3 Uživatelské prostředí

Z uživatelského pohledu se jedná o sadu programů a skriptů s ovládáním běžným v prostředí UNIXu. Jednotlivé parametry a zpracovávané soubory se zadávají na příkazové řádce, výstupy jsou vypisovány do souboru nebo na terminál. Cílem implementace nebylo vytvoření uživatelsky přívětivého

prostředí, ale maximální efektivita. Výhodou použití sady jednodušších programů je jejich možné propojení a automatické spouštění, například pro testování nebo hledání vhodného nastavení parametrů.

4.2 Knihovna Morph

Knihovna Morph slouží pro práci se soubory morfologie [1] a tvoří základ pro většinu používaných programů. Jejím účelem je vytvořit jednotné rozhraní pro práci s daty slovníku a s pomocnými soubory a vytvořit sadu metod pro práci s nimi. Uvedeme zde ve stručnosti nejdůležitější součásti knihovny.

Morfologický slovník zpřístupněný v rozvinuté formě[2]. Slovník obsahuje všechna lemmata (ale nerozlišuje jednotlivé významy lemmat ani velikost písmen), je dostupný ve formě trojrozměrného asociativního pole s indexy `[lemma][vzor][kořen]`. Pro rychlejší načítání může být rozvinutý slovník uchováván na disku v cachovacím souboru.

Seznam derivačních vzorů reprezentovaný dvojrozměrným asociativním polem s indexy `[vzor1][vzor2]`, ze kterého je možné získat jednotlivé položky záznamů derivačních vzorů. Na disku je uložen v souboru `hf_dist.il2`.

Seznam koncovek a tagů pro každý vzor[3] a jeho invertovaná forma – seznam vzorů pro danou koncovku. Jedná se o soubor `hf_end.il2`, který může být doplněný o koncovky nepravidelných slov.

Převod mezi kompaktními a pozičními tagy najde pro každý kompaktní tag jeho poziční variantu. Je implementován pomocí tabulky `Compact2Positional`, která odpovídá souboru `b2800a.f2o`.

Seznam koncovek pro lemmata je tabulka umožňující získat lemma z kořene slova a jeho vzoru. Obsahuje pro každý vzor koncovku, která se připojuje ke kořeni, aby vzniklo lemma. Pro některé vzory obsahuje také počet znaků, které je nutné odtrhnout od původního kořene, než se koncovka připojí.

Metoda pro získání možných vzorů vrací pro dané slovo nebo slova všechny přípustné dvojice ⟨kořen, vzor⟩. Funkce nepoužívá slovník, takže je možné ji použít i pro slova, která v něm nejsou obsažena.

[2] Jsou v něm obsaženy všechny položky, i ty vzniklé derivací.

[3] Pro vzory, které nejsou základní, obsahuje koncovky složené. Vzniknou spojením těch částí kořene, které se připojují během derivování, a koncovky výsledného základního vzoru.

Metoda pro získání společných vzorů vrací pro skupinu slov jejich možné vzory. Pracuje s páry ⟨kořen, vzor⟩ získanými předchozí metodou. Ani tato metoda nepoužívá morfologický slovník.

Nástroj pro spuštění morfologie a taggeru umožňuje spustit nezávisle morfologickou analýzu a tagger [1]. Tato funkce je implementovaná pouze ve verzi knihovny pro jazyk Perl, využívá ji skript `run-morph.pl`.

Knihovna je poměrně univerzální, je možné ji s malým úsilím využít i mimo náš systém pro určování vzorů. Užitečná může být především při práci s morfologickým slovníkem — byla úspěšně použita například pro vyhledání adjektiv pravidelně odvozených od sloves [12], slovních druhů určitého typu s jistou koncovkou atd.

4.3 Fáze přípravy dat

Pro trénování, vyhlazování a testování je potřeba nejprve připravit vhodně upravená data. Ve všech fázích potřebujeme označit slova, která jsou určena k nastavení parametrů nebo pro testování. Nejdřív však musíme tato vhodná slova nalézt, k tomu nám slouží program `prepare-lowfreq.pl`.

prepare-lowfreq.pl

Tento program připraví soubor se slovy splňujícími omezení, která je možné zadat jako parametry při spuštění:

- minimální a maximální četnost slov, které budou zahrnuty do souboru (`min, max`),
- z vhodných slov se náhodně vybere pouze část, která se zahrne do souboru (`percent`),
- do souboru se zahrnou i slova, která jsou nepravidelná[4] (`irregular`).

Kromě souboru vybraných slov může tento program generovat také seznam slovních tvarů s nejvyšší frekvencí, které je možné použít v navazujících fázích. Dalším úkolem programu je vytvořit seznam správného přiřazení slov do skupin, který se později použije ve fázi evaluace.

[4]Mají vzor `0` nebo více než pět různých vzorů.

prepare.pl

Úkolem tohoto programu je označit slova, která připravil předchozí program, což provede přepsáním původního tagu v souboru tagem pro nerozpoznané slovo `X@----------`. Původní tagy (potřebné pro trénování) jsou uchovány v souboru se seznamem vybraných slov.

4.4 Trénování

Na připravených datech můžeme spustit program `train` pro zjištění pravděpodobností jevů jednotlivých metod. Jako jeden z parametrů dostane odkaz na soubory s trénovacími daty. Kromě toho je možné zadat další parametry, které ovlivňují proces trénování:

- nastavení, kolik tagů před slovem a kolik tagů za slovem se bude používat pro n-gramy (parametr `ngram`),
- zda se místo tagů budou pro slova s vysokou četností používat přímo tvary slov (`use-hifreq`),
- kam se budou ukládat natrénované pravděpodobnosti (parametry `stat-xxx`).

Program pracuje inkrementálně, takže je možné ho spustit postupně na různých datech se stejným výsledkem, jako kdyby byl spuštěn na všech datech najednou.

4.5 Vyhlazování

Vyhlazování pravděpodobností má na starosti program `smooth`. Přečte z disku soubory s četnostmi (výsledek předchozí fáze) a upraví pravděpodobnosti pro jevy, které se vyskytují pouze ve vyhlazovacích datech. Při spuštění dostane jako parametr soubory, které mají sloužit pro vyhlazování (stejně jako trénovací data jsou předzpracované, jak je uvedeno v oddílu 4.3). Další možné parametry jsou:

- nastavení, kolik přihrádek se má použít pro vyhlazování n-gramů (`buckets`),
- jestli se místo tagů budou pro slova s vysokou četností používat přímo tvary slov (`use-hifreq`),

- zda má dojít k přepsání souborů novými verzemi s vyhlazenými parametry, nebo se vytvoří soubory nové, s příponou `.sm` (`overwrite`).

Tvar n-gramů (kolik tagů předchází a následuje) program zjistí ze souborů, které vznikly ve fázi trénování.

4.6 Přiřazení vzorů

Samotné přiřazení vzorů zajišťuje program s lakonickým názvem `work`. Jako parametr jsou uvedeny soubory, které obsahují označená neznámá slova, pro něž hledáme vzor. Soubory vzniknou buď v přípravné fázi za účelem testování, nebo jako výstup morfologické analýzy (a případně disambiguace), pokud se jedná o „ostré“ spuštění. Opět je možné zadat některé parametry ovlivňující práci programu:

- nastavení prahu, kdy ještě slova sdílejí lemma a kdy už nikoli (viz oddíl 2.4.6, parametr `cutoff`),
- jestli se místo tagů budou pro slova s vysokou četností používat přímo tvary slov (`use-hifreq`),
- které metody se mají použít pro rozhodnutí o přiřazení slov do skupin a pro přiřazení vzorů (`group-select`, `paradigm-select`),
- vstupní a výstupní soubory (`stat-xxx`, `output-xxx`).

4.7 Evaluace

Ověření úspěšnosti se skládá ze dvou částí: evaluace přiřazení do skupin a evaluace celkového výkonu algoritmu. Jelikož to jsou poměrně oddělené úkoly, provádí se evaluace pomocí dvou nezávislých programů.

eval-groups.pl

K evaluaci skupin se používá program `eval-groups.pl`, který porovná výstup vyprodukovaný programem `work` se souborem správných skupin, který vznikl ve fázi přípravy dat. Program vypíše hodnoty dvou veličin, které určují úspěšnost přiřazení do skupin: přesnosti a úplnosti. Tyto hodnoty udávají, jak dobře jsou slova přiřazena do skupin (viz oddíl 3.4.1).

eval.pl

Evaluace přiřazení vzorů probíhá tak, že program `eval.pl` porovná obsah souboru vybraných slov se souborem, který vyprodukoval program `work`. Stejně jako u ostatních programů může být i u evaluace zadáno několik různých parametrů:

- maximální pozice správného vzoru, aby bylo přiřazení ještě pokládáno za správné (`max`),
- zda se kořeny a lemmata, kterým chybí některá z předpon (`nej-`, `ne-`, `nejne-`), počítají jako úspěšné přiřazení, mají-li správný vzor (`nejne`),
- zda se za úspěšné přiřazení počítá i přiřazení vzoru, který není kořenem v derivačním stromu (viz oddíl 2.2.3, parametr `derivative`).

Výstupem programu je číslo, které odpovídá procentuálnímu zastoupení správně přiřazených vzorů (co se považuje za správně přiřazený vzor, určují parametry). V upovídaném (`verbose`) módu je pro každé slovo vypsána pozice, na které se vyskytuje správný vzor.

optimize-cutoff.pl

Tento program slouží pro optimalizaci parametru `cutoff` programu `work`. Jeho úkolem je spustit trénování, vyhlazování a testování na určených datech. Testování se provádí v cyklu s různým nastavením parametrů tak, aby bylo v evaluaci dosaženo co možná nejlepších výsledků. Umožňuje nastavovat parametry pro ostatní spouštěné programy (n-gramy, přihrádky pro vyhlazování atd.) a některé další vlastnosti:

- maximální počet iterací, které se provedou (`iter`),
- zda se budou používat data původní, pouze s morfologickými značkami, nebo po označení taggerem (`morph`, `tagger`),
- zda budou ukládány dočasné soubory a do jakých souborů (`prefix`, `keep`),
- která verze programů se použije (Perl/C++) (`perl`).

4.8 Pomocné programy

Kromě zmíněných programů, které tvoří základ implementace, vzniklo během vývoje mnoho dalších prográmků. Některé jsou vázány přímo na náš úkol a

pomáhají například vytvořit soubory používané v přiřazování vzorů, jiné vznikly spíše jen jako doplněk funkcí ostatních programů.

irregular-ending.pl

Jak název napovídá, cílem tohoto skriptu je vytvořit seznam možných tagů pro prázdné koncovky nepravidelných vzorů[5] Tento seznam vznikne na základě záznamů v morfologickém slovníku.

Protože některé vzory mohou reprezentovat mnoho různých tvarů, a tudíž by téměř o každém tagu platilo, že může být součástí vzoru, vybereme pouze ty tagy, které mají četnost větší než 10. Získané záznamy jsou přímo použity v souboru `hf_end.il2`.

lemma-ending.pl

Skript projde morfologický slovník a vygeneruje pro každý vzor koncovku, která se připojuje ke kořeni, aby vzniklo lemma. Výsledný soubor `lemma_end.il2` se používá přímo při získávání lemmat z kořene a vzoru v programu `work`.

random-filter.pl

Tento jednoduchý skript slouží ke zvolení určitého procenta náhodných řádek ze vstupního souboru. Použili jsme ho pro výběr trénovacích dat z většího souboru (ČNK).

run-morph.pl

Tento skript slouží ke spuštění morfologické analýzy a taggeru. Pro nás je důležitá především možnost spustit tagger na data, v nichž jsme některé tagy nahradili tagem `X@----------`, čímž jsme je připodobnili datům, ve kterých se vyskytují morfologii neznámá slova.

html2csts.pl

Součástí implementace je také program `html2csts.pl`, který převádí text z formátu HTML[6] do formátu CSTS, takže je možné zpracovat téměř libovolná textová data (podařilo se například úspěšně převést obsah časopisu Vesmír nebo webové stránky on-line časopisu ScienceWorld.cz).

[5]Na tomto místě máme na mysli vzory, které mohou mít uveden tag přímo ve slovníku u svého lemmatu (viz oddíl 1.1.3): `0`, `0n`, `0ns`, `0abbr` a `zkr`.

[6]HyperText Markup Language, v současnosti formát většiny webových stránek.

Ze stránek ve formátu HTML bychom chtěli získat texty, které jsou vhodné pro další zpracování. Jelikož součástí stránek bývá mnoho formátovacích informací (menu, lišty, seznamy odkazů, obrázky...), ve kterých se mohou užitečné texty téměř ztratit, je vhodné text stránek nějakým způsobem filtrovat. Naším cílem je získat z webových stránek pouze užitečné a pokud možno souvislé texty a nikoli věty typu ,,Pro zavření okna klikněte na obrázek.“ apod.

Program `html2csts.pl` používá k filtrování obsahu stránek metodu posuvného okénka. Pro každou pozici na stránce zjistí, zda text v okolí není přerušen horizontální čárou, obrázkem, příliš mnoha odkazy apod. Průchodem přes celý soubor potom vybere souvislé části textu, které přerušeny nejsou.

Program využívá knihovny `HTML::Parser` a `Text::Iconv`.

Závěr

V této práci jsme prozkoumali problém přiřazení morfologických vzorů českým slovům. Kromě rozboru několika problémů, které se při analýze úkolu vyskytly, jsme navrhli čtyři metody, které na základě různých vlastností slova nebo jeho okolí pomáhají určit ke slovu správný vzor. Implementovali jsme poměrně úspěšný algoritmus, který vytváří skupiny slov odvozených od stejného lemmatu. Dále jsme navrhli a implementovali postup, jak co možná nejlépe sloučit slova, která sdílejí stejné lemma a obsahují některou z přípustných předpon.

Jednotlivé metody a nastavení jejich parametrů jsme vyzkoušeli na textech PDT a části ČNK. Dosažené výsledky odpovídají předpokládané složitosti úlohy, ale nedosahují přesnosti, jaká by byla potřeba pro plně automatické určování vzorů neznámých slov. Navržené vzory však mohou výrazně pomoci lidskému anotátorovi v rozhodování, a urychlit tak jeho práci.

Uvedené metody by jistě šlo ještě modifikovat či rozšířit, myslíme však, že základní možnosti určování vzorů jsme vyčerpali a dosažené výsledky jsou poměrně dobré. Na závěr uveďme ještě několik oblastí, o kterých se domníváme, že skýtají možnosti dalšího vylepšování práce programu.

Možná zlepšení

První oblastí, na kterou by bylo dobré se zaměřit, jsou slova s nepravidelnými tvary (vzorem). Problém se týká zejména metod n-gramů a kocovek slov, které zvýhodňují „nepravidelné“ vzory před ostatními.

Dalším bodem v seznamu možných zlepšení je systém pro vytváření lemmatu k danému kořeni a vzoru. Jak bylo popsáno v oddílu 2.4.8, řešení, které jsme zvolili, není stoprocentně spolehlivé, nicméně ve velké většině případů přiřazuje lemmata správně, a je tudíž v praxi dobře použitelné.

Na základě experimetů uvedených v kapitole 4 se domníváme, že by bylo dobré se zaměřit především na slova, která se vyskytují častěji, než jednou, nejlépe ve více svých tvarech.

Přidání další metody pro hodnocení možných vzorů by bylo další možností, jak zlepšit výsledek práce algoritmu. Máme na mysli například

sledování četnosti předpon slov, změn v kořeni pro různé vzory atd.

Také je třeba vzpomenout, že implementace algoritmů není ideální zejména ohledně využití operační paměti, neboť důraz byl kladen spíše na srozumitelnost a rychlost algoritmů než na omezení paměťové náročnosti. Předpokládáme, že použitím speciálních datových struktur místo obecných asociativních polí by se využití paměti zlepšilo.

Literatura

[1] Hajič J., Disambiguation of Rich Inflection, Karolinum – Charles University Press, Prague, 2004

[2] Sedláček, R., Morfologický analyzátor češtiny, Diplomová práce, FI MU, Brno 1999

[3] Hajič J., Positional Tags: Quick Reference (Czech Morphology), 2000, <http://ufal.mff.cuni.cz/pdt/Morphology_and_Tagging/...
...Doc/hmptagqr.htm>

[4] Goldsmith J., Automorph, 1997-2000

[5] Goldsmith J., Unsupervised Learning of the Morphology of a Natural Language, Computational Linguistics 27(2)

[6] Yarowsky D., Wicentowski R., Minimally supervised morphology induction, NAACL'01, Pittsburgh, USA

[7] Hlaváčová J., Morphological Guesser of Czech Words, TSD, Železná Ruda 2001

[8] Manning Ch., Schütze, H., Foundations of statistical natural language processing, MIT Press 2002

[9] Hajič J., Hajičová E., Pajas P., Panevová J., Sgall P., Vidová-Hladká, B., Pražský závislostní korpus, Praha 2001
<http://ufal.ms.mff.cuni.cz/pdt/>

[10] Český národní korpus – SYN2000. Ústav Českého národního korpusu FF UK, Praha 2000. <http://ucnk.ff.cuni.cz/>.

[11] Programovací jazyk Perl <http://www.perl.com/>

[12] Doležalová, D. J., Automatic Construction of a Valency Lexicon of Czech Adjectives, TSD, Karlovy Vary 2005

Rejstřík