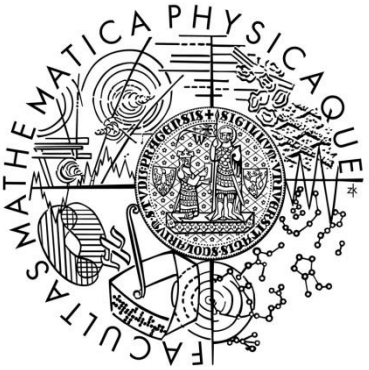
Univerzita Karlova v Praze

Matematicko-fyzikální fakulta

BAKALÁŘSKÁ PRÁCE



Jiří Dutkevič

Implementace metod analýzy struktury českých souvětí.

Ústav formální a aplikované lingvistiky

Vedoucí bakalářské práce: doc. RNDr. Vladislav Kuboň, Ph.D.

Studijní program: Informatika

Studijní obor: programování

Praha 2012

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně a výhradně s použitím citovaných pramenů, literatury a dalších odborných zdrojů.

Beru na vědomí, že se na moji práci vztahují práva a povinnosti vyplývající ze zákona č. 121/2000 Sb., autorského zákona v platném znění, zejména skutečnost, že Univerzita Karlova v Praze má právo na uzavření licenční smlouvy o užití této práce jako školního díla podle § 60 odst. 1 autorského zákona.

V Praze dne 16.5.2012

Název práce: Implementace metod analýzy struktury českých souvětí.

Autor: Jiří Dutkevič

Katedra / Ústav: Ústav formální a aplikované lingvistiky

Vedoucí bakalářské práce: doc. RNDr. Vladislav Kuboň, Ph.D., Ústav formální a aplikované lingvistiky

Abstrakt: Práce se zabývá automatickou analýzou struktur souvětí v češtině. Shrnuje výsledky předchozího výzkumu, užívá v něm navržené metody rozdělení souvětí do segmentů pomocí přesně definované množiny separátorů a na základě přednesených pravidel navrhuje tři metody automatického určování úrovní jednotlivých segmentů v souvětí a tím i jejich vzájemných vztahů. Jedna metoda přímo využívá pravidel navržených v odkazovaném výzkumu, druhá používá genetický algoritmus a třetí neuronovou síť. Práce obsahuje implementaci těchto metod a vyhodnocení výsledků na ručně anotovaných datech z Pražského závislostního korpusu.

Klíčová slova: počítačová lingvistika, syntaxe, struktura souvětí, genetické algoritmy, neuronové sítě

Title: An Implementation of Methods of Structural Analysis of Czech Complex Sentences

Author: Jiří Dutkevič

Department: Institute of Formal and Applied Linguistics

Supervisor: doc. RNDr. Vladislav Kuboň, Ph.D., Institute of Formal and Applied Linguistics

Abstract: This paper discusses automated analysis of complex sentences in Czech language. It summarizes the results of preceding research, uses therein described method for splitting complex sentences into segments using well defined set of separators and proposes three methods of automated assignment of levels to segments (which also describe relations between the segments) in sentences based on rules presented in the research. First method directly applies the rules presented in referenced research papers, the second method uses a genetic algorithm and the third makes use of a neural network. This paper includes an implementation of these methods and an analysis of the results using manually annotated data from the Prague Dependency Treebank.

Keywords: computational linguistics, syntax, complex sentence structure, genetic algorithms, neural networks

1 Úvod

1.1 Předmluva

Pro mnoho aplikací v počítačové lingvistice je třeba provést syntaktickou analýzu souvětí. To je náročný úkol, zejména v jazycích s volným slovosledem, mezi něž patří i čeština.

Z provedeného výzkumu vzešly metody provádějící automatickou syntaktickou analýzu. Úspěšnost těchto metod je relativně dobrá u kratších větných celků, se vzrůstající délkou se snižuje. Z toho vyplývá, že rozdělením souvětí na menší nezávislé celky může usnadnit syntaktickou analýzu a zvýšit tak její úspěšnost.

Řešením je mezi morfologickou a syntaktickou analýzu vložit analýzu segmentační, jejímž vstupem budou morfologická data o slovech ve větě a výstupem rozdělení souvětí do segmentů a klauzí (vět) a vztahy mezi nimi. Takovéto zmenšené celky pak budou vstupem pro syntaktickou analýzu.

Čeština má dostatečně striktní pravidla pro interpunkci, na základě definované množiny separátorů lze tedy souvětí spolehlivě rozdělit na segmenty, což jsou lingvisticky motivované jednotky, jejichž hlavní předností je jejich snadná rozpoznatelnost.

Cílem této práce je dle pravidel definovaných v předešlém výzkumu rozdělit souvětí na segmenty a následně navrhnout a implementovat metody, které segmentům určí úrovně zanoření v souvětí a umožní segmenty spojit do jednotlivých klauzí. Následně pak vyhodnotit úspěšnost těchto metod.

1.2 Česká gramatika

Tato kapitola popisuje gramatické jevy, jichž využívám při návrhu jednotlivých metod. Omezuje se téměř výhradně na větnou skladbu, protože ta je zásadní pro řešení problému. Strukturou vycházím z učebnice [1]. Detaily čerpám z [2] a [3].

Dále se věnuji morfologii, protože zdrojová data z korpusu jsou morfologicky anotována.

1.2.1 Morfologie

Morfologie nebo-li tvarosloví je nauka zabývající se strukturou slov.

Čeština je syntetický, flektivní jazyk, tedy jedno slovo je většinou tvořeno vícero morfémy, jeden morfém může nést více informací o gramatických kategoriích, toto vede ke kratším slovům než u aglutinačních jazyků. Mezi závislými členy ve větě často dochází ke shodě, čehož lze využít při identifikaci k sobě náležejících segmentů.

Každé slovo je charakterizováno několika gramatickými kategoriemi. Charakteristika je součástí vstupních dat (byla slovům přiřazena při morfologické anotaci) ve formě tagů. (Podrobněji popsáno v části 1.5.)

1.2.2 Větná skladba

Větná skladba je nauka zkoumající stavbu vět a souvětí.

Souvětí může být tvořeno jednou či více klauzemi (větami). Každá klauze se skládá z jednoho či více segmentů. Při určování jednotlivých klauzí je důležitá znalost vnitřní struktury vět.

Čeština je jazykem s volným slovosledem ve větě s převažujícím slovosledem Subject-Verb-Object (podmět-přísudek-předmět) ([3] 3.4). Častým jevem jsou neprojektivní konstrukce, kde mezi dva závislé členy je vložen třetí, který náleží do jiné větve závislostního stromu (tedy nezávisí na řídícím členu a to ani tranzitivně). Dle [5] téměř 25% vět obsahuje aspoň jednu neprojektivní konstrukci.

Př.:

a) - Soubor, jež byl zašifrovaný, jsem neuměl otevřít.

b) - Neuměl jsem otevřít soubor, jež byl zašifrovaný.

Obě věty mají tentýž význam. Neprojektivní konstrukci ve větě a) tvoří řídící člen „otevřít“ a závislý člen „soubor“. Neprojektivní konstrukce zde rozděluje jednu klauzi na dvě části, protože člen „soubor“ si při přesunu z b) do a) ponechá u sebe závislou vedlejší větu a tudíž v a) rozdělí hlavní větu na dva kusy.

Věta b) je příkladem slovosledu (SVO), kde podmět na začátku věty je nevyjádřený.

Odchylky od pravidelné větné stavby mohou způsobovat vsuvky, u nichž je problematické, že nemusí být ničím od související věty odděleny. Dále oslovení, která mohou stát i samostatně jako jednočlenné věty a samostatné větné členy, které lze zpravidla rozlišit tím, že související věta na ně odkazuje zájmenem ten.

Valence je schopnost slov na sebe syntakticky vázat další větné členy. Dělí se na tři typy: a) obligatorní, b) potenciální a c) fakultativní. Typy a) a b) se musí v gramaticky správných větách vyskytovat, typy b) však mohou být nevyjádřené (známé z kontextu) ([3] 2.2). Typu a) lze využít k identifikaci segmentů jedné věty. Pokud v jednom segmentu nejsou splněny všechny obligatorní vazby slovesa, pak musí daná věta obsahovat nejméně jeden další segment. Prakticky to ale použít nelze právě kvůli tomu, že i obligatorní vazby slova mohou být splněny nevyjádřeně (pouze v kontextu).

Základním stavebním prvkem jsou skladebné dvojice, které obsahují řídící a závislý člen. Ve skladebné dvojici dochází buď ke shodě, kde se závislý člen shoduje s řídícím v pádě / osobě, čísle, rodě a životnosti, nebo se závislý člen řídí pádem, nebo závislost vůbec není vyjádřena. Těchto jevů lze využít k hledání k sobě náležících členů (hlavně napříč segmenty). V základní skladebné dvojici se podmět shoduje s přísudkem, shody (zejména u několikanásobného podmětu) lze využít ke spojení segmentů náležících do jedné klauze.

Př.: Petr a Pavel jeli tramvají.

Kdyby druhý segment (Pavel jeli tramvají) tvořil samostatnou klauzi, neshodoval by se podmět s přísudkem v čísle.

Základní skladebnou dvojici obsahují dvojčlenné věty, věty jednočlenné obsahují pouze přísudek (slovesné věty), nebo podmět. Věty slovesné jsou svou stavbou podobné větám s nevyjádřeným podmětem (za nevyjádřený podmět můžeme považovat ono).

Zbytek věty (pokud není holá) je tvořen rozvíjejícími větnými členy. Předmět je závislý na slovese, nebo přídavném jméně. Přívlastek rozvíjí a vymezuje podstatné jméno. Přívlastek shodný se se jménem shoduje v pádě, čísle a rodě. Je-li shodný přívlastek vyjádřen podstatným jménem, nazývá se přístavek. Doplněk je člen závislý na dvou jiných členech – přísudku a jméně. Příslovečné určení rozvíjí sloveso, závislost není vyjádřena ani shodou, ani řízeností.

Souvětí je celek obsahující alespoň dvě věty. Vedlejší věty nahrazují větné členy řídících vět. Mohou to být podmět, přívlastek, přístavek, doplněk a příslovečné určení. Vedlejší věty lze vložit do řídící věty, tato věta je pak rozdělena na (nejméně dvě) části. Souvětí musí obsahovat aspoň jednu hlavní větu, která nezávisí na žádné jiné větě.

1.3 Pražský závislostní korpus

Tato kapitola pojednává o Pražském závislostním korpusu (PDT), z jehož dat tato práce vychází. Informace čerpám z dokumentace [7].

Pražský závislostní korpus je projektem Ústavu formální a aplikované lingvistiky Matematicko-fyzikální fakulty Univerzity Karlovy v Praze. Vychází z funkčního generativního popisu (FGD) vytvořeného Petrem Sgallem. Základem FGD je popis jazyka na několika rovinách.

Morfologická anotace přiřazuje jednotlivým slovům tag (charakteristiku gramatických vlastností slova) a lemma (základní formu slova). Analytická anotace má již stromovou strukturu a přidává závislosti mezi slovy ve větě – popisuje vztahy uzlů ve stromu k řídícím uzlům. Tektogramatická anotace je nejvyšší anotační úrovní dat v PDT, popisuje hloubkovou strukturu věty, uzlu stromy reprezentují pouze plnovýznamová slova a stromy navíc obsahují i nevyjádřené větné členy. V této práci budu používat morfologickou a analytickou anotaci dat.

PDT 2.0 obsahuje dva milióny českých slov. Zdrojem dat jsou původní články z Lidových novin, Mladé fronty Dnes, Českomoravského Profitu a Vesmíru z let 1991-1995. Dva milióny jsou anotovány na morfologické rovině, 1,5 miliónu na analytické rovině a 0,8 miliónu na tektogramatické rovině.

1.4 Provedený výzkum

Tato kapitola shrnuje obsah článků, na jejichž výsledcích zakládám své metody. Dle zadání vycházím z článků z konferencí FLAIRS07 [10], ITAT08 [11] a ITAT10 [12]. Protože je tato práce v češtině, místo článku [10] použiji článek z konference ITAT06 [9], protože se svým obsahem s článkem [10] téměř shoduje.

Jelikož jsou si články [9] a [11] obsahově velice blízké a tematicky se překrývají, věnuji se jejich obsahu najednou, pak zvlášť rozebírám článek [12].

1.4.1 Články:

O segmentaci českých vět [9]

A Linguistically-Based Segmentation of Complex Sentences [10]

Vztahy mezi segmenty– segmentační schémata českých vět [11]

Základní motivací pro segmentační analýzu souvětí je zjednodušení analýzy syntaktické. Syntaktická analýza je předpokladem pro mnoho dalších úkolů, jako příklad bych uvedl strojový překlad. Především pro jazyky s volným slovosledem, který má i čeština, se jedná o náročný úkol. V předchozím výzkumu se podařilo dosáhnout dobré spolehlivosti analýzy u krátkých vět, segmentační analýzou lze tedy ze souvětí získat několik kratších, relativně nezávislých, celků.

Metoda nalezení jednotlivých segmentů souvětí je založena na morfologické úrovni anotace. Existují nástroje s vysokou přesností anotace, na podkladová data se tedy lze spolehnout.

Definice: [9]

Nechť S, $S = w_1w_2...w_n$, je věta přirozeného jazyka. Jako segmentaci věty S označujeme každou posloupnost $D_0W_1D_1...W_kD_k$, kde jednotlivé D_i $(0 \leq i \leq k)$ reprezentují složené separátory a kde W_i $(1 \leq i \leq k)$ označují jednotlivé segmenty, tedy maximální posloupnosti lexikálních jednotek, které neobsahují separátory.

Protože na relevantních pozicích morfologického tagu je přesnost téměř stoprocentní, můžeme se na značky spolehnout a segmentaci souvětí považovat za jednoznačnou. Jednotlivé segmenty získáme identifikací separátorů.

Za elementární hranice považujeme:

- interpunkci – čárka, dvojtečka, středník, otazník, vykřičník, pomlčka, závorky, svislítko a uvozovky,
- interpunkci na konci věty,
- souřadné spojky – poznáme podle toho, že tag začíná J^.

Maximální souvislou posloupnost elementárních hranic nazýváme složenou hranicí.

Navíc požadujeme, aby každé souvětí začínalo a končilo hranicí. Není-li tomu tak, doplníme hranici prázdnou.

Tečku považujeme za hranici pouze na konci věty. Toto pravidlo jsem nedodržel, abych zachoval kompatibilitu s nástrojem SegView (více v části 1.6 popisující tento nástroj).

Dle obsahu segmentu se mu dále přidělují příznaky, které se následně používají při analýze. Příznak podřízenosti (dále jej budu, stejně jako v článku, označovat PP) se segmentu přiděluje tehdy, obsahuje-li slovo se začátkem (první dvě pozice) morfologického tagu J, (J následované čárkou), P4, PE, PJ, PK, PQ, PY, C?, Cu, Cz a D (zde je hodnota druhé pozice libovolná).

Články dále navrhují dvě metody automatické segmentace. První je založena na analytických stromech. Tuto metodu nepopisuji, protože stromy v této práci nevyužívám. Druhá metoda vychází pouze z morfologické úrovně anotace. Pro každý segment určujeme zleva doprava možný interval úrovní dle následujících pravidel:

- první segment je na úrovni 1 pokud má PP, jinak je na úrovni 0,
- má-li segment následující čárku PP, je o jednu úroveň níže než segment jemu předcházející, nemá-li PP, je nejníže na stejné úrovni, nejvýše na úrovni 0,
- u otevírací závorky je úroveň zvýšena o jednu, respektive o dvě je-li přiřazen PP,

- uzavírací závorka, jež má odpovídající otevírací závorku vrací úroveň zpět na úroveň před otevírací závorkou, jinak úroveň nemění; uzavírací závorka je problematická, protože je obtížné najít k sobě náležící otevírací a uzavírací závorky pokud je uzavírací závorka použita ve významu uzavření závorky i výčtu,
- koordinační spojka úroveň zachovává,
- dvojtečka nemění dolní mez úrovně, horní mez o jednu zvyšuje, má-li segment PP, pak navíc obě úrovně zvýší o jednu,
- otazník nebo vykřičník nastaví dolní mez na 0 a horní mez o jednu sníží,
- středník nemění horní mez úrovně, dolní nastavuje na 0,
- svislítko, pomlčka, uvozovky meze nemění.

Není-li žádný segment na úrovni 0, všechny meze je třeba náležitě posunout.

Autoři změřili recall (podíl segmentů na určité úrovni, které na ni byly správně zařazeny) této metody. Správně určeno bylo rozmezí úrovně u 64% segmentů, při toleranci dvě úrovně pak 76%.

1.4.2 Článek:

Od segmentů ke klauzím v češtině – analýza vybraných jevů [12]

Článek navazuje na předchozí dva. Syntaktická analýza jazyků s volným slovosledem je problematická. Správné určení jednotlivých klauzí a vztahů mezi nimi je obtížné. Náročné jsou hlavně vnořené větné konstrukce, u kterých lze sice často snadno určit začátek, ale konec a především úroveň následujícího segmentu je těžké najít. Nejzávažnějším problémem je pak rozlišení větně členské a větné koordinace, někdy je lze rozlišit pouze z kontextu a čistě syntakticky je to nejednoznačné.

Pro účely tohoto článku bylo za pomoci editoru SegView (popsán v následující kapitole) ručně anotován soubor vět z PDT. Byly doplněny informace o vztazích mezi většími větnými celky. Článek dále předkládá kvantitativní, lingvistickou a strukturní analýzu vybraných jevů.

Závěrem autoři před lingvistickou analýzou doporučují vyčlenit a zvlášť analyzovat věty s nestandartní strukturou. Za klíčové považují rozlišení větné a větně členské koordinace. U spojek vidí nutnost zkoumat i lexikální hodnotu, samotný morfologický tag nestačí pro správné určení úrovně segmentu. Dále navrhují využívat valence sloves pro identifikaci k sobě náležejících segmentů.

1.5 Zdrojová data

Data, se kterými pracuji pocházejí z PDT. Věty jsou v Prague Markup Language [13], což je jazyk založený na XML určený pro zachycení lingvistických informací. Data jsou tedy v přesně definovaném formátu, který respektuji, pracuji však pouze s částmi souborů (vynechávám data, která nepotřebuji).

Výstupní data jsou generována editorem SegView, který zadefinoval vlastní textový formát, kde oddělovačem je white space (mezery a konce řádků). Tento formát dodržím přesně, aby výstup šel následně otevřít v editoru SegView.

Základními datovými soubory PDT jsou ty, které obsahují slovní vrstvu korpusu. Tyto soubory nemám k dispozici, ale to nevadí, protože kompletní informace o souvětích je k dispozici i v dalších vrstvách.

1.5.1 Morfologická anotace

Morfologická anotace je uložená v XML souborech s příponou „m“, jejichž formát je definován ve schématu http://ufal.mff.cuni.cz/jazz/pml/schemas/mdata_schema.xml.

XML soubor obsahuje kořenový element „mdata“, jehož potomky jsou hlavička, metainformace a hlavně samotné věty a jejich morfologická anotace. Každá věta je reprezentována elementem „s“, která má atribut „id“, což je identifikátor věty. V těchto souborech má každé „id“ (jak vět, tak slov) předponu „m-“, kterou při zpracování souboru odstraňuji. Potomky elementu „s“ jsou elementy „m“, které reprezentují jednotlivá slova věty. Jsou rovněž identifikovány hodnotou atributu „id“. Důležitými potomky elementu „m“ jsou elementy „form“, „lemma“ a „tag“, které reprezentují původní slovo, lemma a morfologický tag (v daném pořadí).

Příklad formátu:

```
<s id="m-cmpr9406-002-p2s1A">
  <m id="m-cmpr9406-002-p2s1Aw1">
   <form>Bankroty</form>
   <lemma>bankrot</lemma>
   <tag>NNIP1-----A----</tag>
```

</m>

</s>

Formát morfologického tagu:

Tag je patnáct znaků dlouhý textový řetězec, každá pozice reprezentuje jednu gramatickou kategorii. Seznam kategorií je v následující tabulce.

Pozice	**Název**	**Popis**
1	POS	Slovní druh
2	SubPOS	Poddruh, bližší specifikace slovního druhu
3	Gender	Rod
4	Number	Číslo
5	Case	Pád
6	PossGender	Rod vlastníka
7	PossNumber	Číslo vlastníka
8	Person	Osoba
9	Tense	Čas
10	Grade	Stupeň
11	Negation	Negace
12	Voice	Aktivum / pasivum
13	Reserve1	Rezerva
14	Reserve2	Rezerva
15	Var	Forma

Tabulka 1 – Význam pozic v morfologickém tagu

Dokumentace PDT [14]

Možné hodnoty pro každou kategorii jsou v dokumentaci PDT [14].

1.5.2 Analytická anotace

Analytická anotace je uložena v XML souborech s příponou „a“, jejichž formát je definován ve schématu http://ufal.mff.cuni.cz/jazz/pml/schemas/adata_schema.xml.

XML soubor obsahuje kořenový element “adata”, jehož potomky jsou hlavička, metainformace a element “trees”, který obsahuje samotné analytické stromy. Uzly stromu jsou reprezentovány elementy LM s atributem “id” obsahujícím identifikátor (nyní s předponou “a-”). Jeden uzel reprezentuje kořen stromu a má “id” celé věty, jeho potomci pak jednotlivá slova ve větě. Podstatný je zde pouze element “afun”, který popisuje analytickou funkci slova.

Příklad formátu:

```
<LM id="a-cmpr9406-001-p2s1">
  <children>
   <LM id="a-cmpr9406-001-p2s1w2">
    <afun>ExD</afun>
   </LM>
  </children>
</LM>
```

Význam jednotlivých možných hodnot lze najít v dokumentaci [15].

V souborech, které jsem zpracoval pro použití v této práci, je odstraněna informace o stromové struktuře.

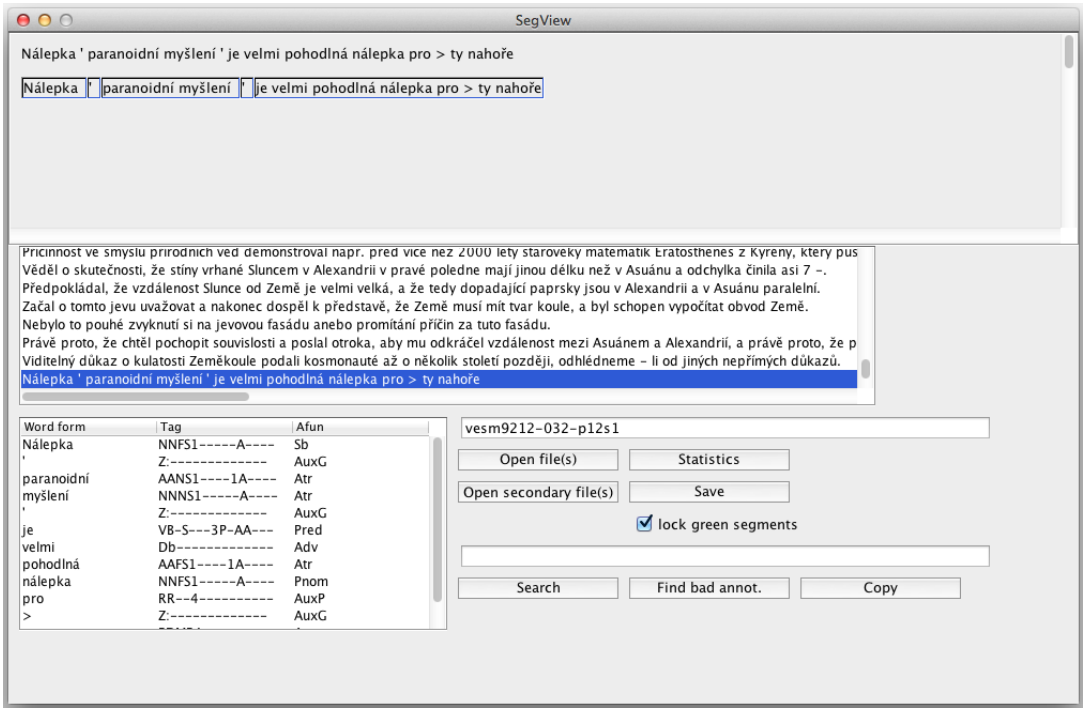
1.5.3 Segmentační struktury

Formát souborů byl zadefinován autorem editoru SegView. Jedná se o textový soubor. Dokumentaci k tomuto formátu nemám, takže jsem jej odvodil z jednotlivých instancí. Každý soubor obsahuje informaci o segmentačním schématu právě jednoho souvětí. Na prvním řádku je identifikátor souvětí, domnívám se však, že první řádek je nepodstatný, důležité je, aby “id” věty tvořilo název souboru před příponou “seg”.

Na zbylých řádcích jsou informace o segmentech, přičemž každý řádek odpovídá jednomu segmentu. Na řádku jsou dvě mezerou oddělená čísla, první vyjadřuje úroveň segmentu ve struktuře a druhé nabývá hodnoty 0 nebo 1 dle toho, zda jsou segmenty spolu spojené. Vzhledem k tomu, že tato informace je používána

v SegView pouze pro grafiké znázornění a usnadnění práce uživateli, všechny mnou generované soubory mají na tomto místě 0.

1.6 Editor SegView



Obrázek 1 – Uživatelské rozhraní editoru SegView

Editor SegView je program napsaný v Javě pro práci se segmentačními schématy souvětí v grafickém uživatelském rozhraní. Umožňuje rovněž vyhledávat mezi větami dle vlastností segmenů, klauzí, slov a jejich vzájemných vztahů.

Protože mým cílem bylo, aby výstup mého softwaru byl se SegView kompatibilní, musel jsem některé aspekty přizpůsobit SegView:

- Tečka je považována za separátor kdekoli ve větě.
- Kompletně jsem převzal sadu znaků použitých jako separátory.
- Rozdělení souvětí na segmenty je shodné se SegView.

Toto jsem učinil, abych zajistil, že souvětí bude mít stejný počet segmentů, a že to budou tytéž segmenty, protože formát výstupních „seg“ souborů editoru SegView neobsahuje mapování úrovní na segmenty, pouze „id“ souvětí. Tedy pro použití těchto souborů je nutno nejdříve provést totožnou segmentaci na souvětí a pak ve stejném pořadí přiřadit vzniklým segmentům úrovně.

Navíc SegView přiřazuje úrovně nejen segmentům, ale i samotným separátorům, vzhledem k tomu, že články umožnují vytvářet kopie separátorů na více úrovních [9], je to nejednoznačné. Tento fakt jsem zohlednil při vyhodnocování výsledků.

2 Implementce metod

Hlavním úkolem mé práce bylo naimplementovat aspoň dvě metody automatického určení úrovní segmentů v souvětí. Tato část se věnuje popisu použitých technologií, struktuře sofwarového projektu a pak popisu jednotlivých metod. U každé metody nejdříve popisuji teoretické pozadí použitého nástroje, pak způsob namapování problému na tento nástroj a nakonec detaily samotné implementace algoritmů.

2.1 Technologie

K implementaci jsem zvolil jazyk C# [16], protože se mi v něm dobře pracuje a je dostupný pro mnoho platforem. Pro spuštění je třeba mít nainstalován runtime Mono [18], nebo .NET Framework [17] Součástí zdrojového kódu je solution, soubor, který obsahuje popis struktury projektu a všechny soubory potřebné k jeho zkompilování. Solution byla vytvořena v programu MonoDevelop [19], měla by však být kompatibilní i s Visual Studiem [20]. Projekt vyžaduje .NET Framework verze 4 nebo novější, využívá vlastnosti, které v předchozích verzích nejsou k dispozici.

Mono, .NET Framework, MonoDevelop i Visual Studio (Express Edition) lze získat zdarma na webových stránkách výrobce. Kvůli tomu, že podporují více platforem, nejsou tyto produkty součástí práce.

Pro vývoj a testování jsem použil operační systém MacOS 10.7, MonoDevelop verze 2.8.8.4 a Mono verze 2.10.9.

2.2 Architektura

Projekt se skládá z několika dynamických knihoven, které obsahují veškerou logiku aplikace. Funkce těchto knihoven jsou uživateli zpřístupněny prostřednictvím konzolových aplikací. Výhodou této architektury je, že knihovny lze snadno využít v dalších projektech a konzolové aplikace lze spouštět pomocí skriptů. Projekt neobsahuje žádné vizualizace, neshledal jsem tedy nutným implementovat grafické uživatelské rozhraní.

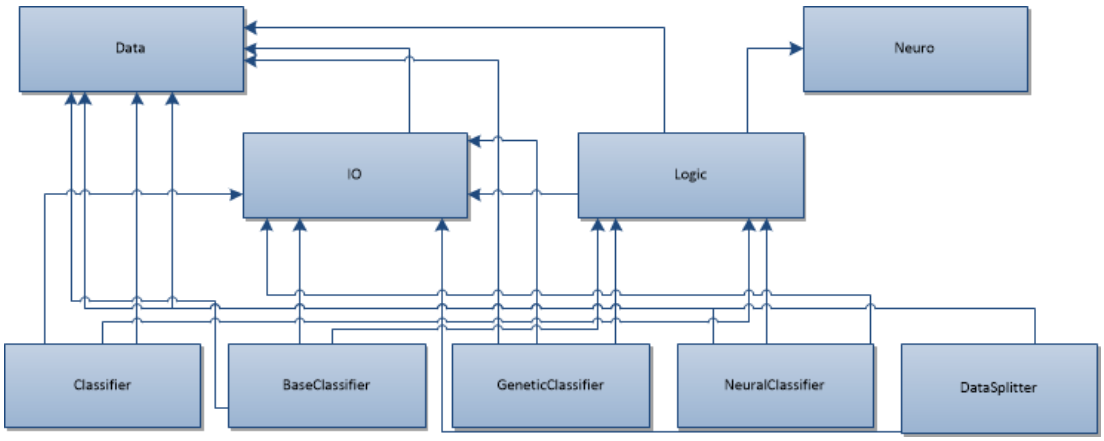
Konkrétně se jedná o následující knihovny:

- Data – obsahuje datové struktury, na které odkazují ostatní knihovny,
- IO – obsahuje logiku pro načítání dat ze vstupních souborů a výpis dat do výstupních souborů,
- Logic – obsahuje metody automatické segmentační analýzy,
- Neuro – obsahuje implementaci neuronové sítě (blíže popsáno v části 2.5 o metodě s neuronovou sítí),

a konzolové aplikace:

- Classifier – hlavní produkt, vstupem je soubor, jež popisuje úkoly, které má software vykonat (blíže popsáno dále v této kapitole),
- BaseClassifier – utilita na klasifikaci používaní při vývoji a testování,
- GeneticClassifier - utilita na klasifikaci používaní při vývoji a testování,
- NeuralClassifier - utilita na klasifikaci používaní při vývoji a testování,
- DataSplitter – utilita na rozdělení vstupních dat do tříd na učení, ověřování a testování.

Hlavním rozdílem mezi aplikací Classifier a ostatními konzolovými aplikacemi je, že Classifier se nastavuje pomocí textového souboru, kdežto ostatní mají nastavení ve zdrojovém kódu, pro úpravu funkcionality je třeba je vždy překompilovat.



Obrázek 2 – Diagram závislostí jednotlivých komponent

Formát vstupních a výstupních dat byl popsán v předchozí kapitole. Na vstupu můžou být data ve formátu PDT. Výstupní data jsou sice, ve stejném formátu, ale

některé informace chybí. Software je schopen použít svá výstupní data opět jako vstup. Výstup segmentačních schémat je kompatibilní s editorem SegView.

2.2.1 Module Data

Modul Data sdružuje třídy pro uchování dat a obsahuje základní logiku pro práci s nimi.

Třída Tag reprezentuje morfologická tag slova. Obsahuje proměnnou pro uchování informace o každé pozici v tagu, tyto proměnné jsou reprezentovány pomocí výčtového typu. Toto zajišťuje kontrolu platnosti tagu ve všech pozicích. Díky tomuto jsem zjistil, že v datech jsou obsaženy i informace v tagu, které nejsou v dokumentaci PDT, není tedy zřejmé, jakou gramatickou kategorii reprezentují.

Třída Word reprezentuje jednotlivá slova, uchovává informaci o identifikátoru, tvaru slova, lemmatu, tagu a analytické funkci (která je reprezentována výčtovým datovým typem). Navíc tato třída obsahuje informaci o separátorech segmentů. Tato informace je součástí datové vrstvy, protože není součástí výzkumu, ale je definována, v rámci práce je tedy neměnná (i kvůli kompatibilitě s editorem SegView).

Třída Segment reprezentuje segment v souvětí, a to jak separátory, tak významové části. Uchovává informaci o prvním a posledním slově segmentu (pozici ve větě), úroveň segmentu a rozsah úrovně segmentu. Segment neobsahuje informaci o větě kterou popisuje, což se ukázalo jako nevýhodné, protože kdykoli byl segment předáván jako parametr, musela s ním zároveň být předána i příslušná věta. Tento problém jsem si uvědomil až v průběhu implementace, rozhodl jsem se to již neměnit, protože by to nepomohlo úspěšnosti ani rychlosti metod.

Třída Sentence reprezentuje jednotlivá souvětí. Uchovává informace o identifikátoru věty a seznam slov ve větě (s informací o jejich pořadí). Implementuje rozhraní IEnumerable<Word>, lze tedy po jednom procházet jednotlivá slova ve větě. Dále obsahuje seznam segmentů a metodu na získání tohoto seznamu, protože závisí pouze na separátorech.

Třída Splitter rozděluje data do skupin na učení a testování. Celkem vytváří tři skupiny, vstupem jsou tedy tři celá čísla, která určují poměr velikostí skupin. Veškerá data jsou rozdělena do oddílů o velikosti součtu vstupních parametrů a následně je z každého oddílu vybrán příslušný (dle hodnoty parametru) počet vět do skupiny. Tento způsob jsem zvolil proto, aby jednotlivé skupiny pokud možno rovnoměrně pokrývaly všechny články z korpusu.

Třída Evaluation počítá výsledky tak, že zaznamenává poměry správně určených entit vůči celkovému počtu entit. Shromažďuje informace (podíl správně určených vůči celku) o následujících jevech:

- celkový počet vět,
- počet správně určených úrovní segmentů,
- počet správně určených úrovní segmentů s tolerancí jedné úrovně,
- počet správně určených směrů změn úrovně segmentu (tedy zda následující úroveň je nižší, stejná nebo vyšší),
- počet správně určených změn úrovně segmentu (zda přesně odpovídá rozdíl sousedních úrovní),
- počty segmentů na jednotlivých úrovních a úspěšnost na dané úrovni (podíl počtu segmentů na této úrovní vůči počtu segmentů na tuto úroveň zařazených),
- precision, recall a f-score (význam je vysvětlen ve třetí části) pro úrovně segmentů a změny úrovní segmentů.

U bodů 1, 2 a 4 jsou informace shromažďovány třikrát: pro všechny segmenty, pouze pro segmenty, které nejsou separátory a pro věty, které obsahují více segmentů, jež nejsou separátory (protože věty s pouze jedním segmentem mají pouze jedno možné segmentační schéma, vše na úrovní 0).

2.2.2 Module IO

Modul IO obsahuje dvě statické třídy pro načítání a zápis dat. Pro zpracování XML souborů používám LINQ to XML [21].

Třída DataReader obsahuje metody pro načítání dat do datových struktur. Umí z „m“, „a“ a „seg“ souborů načíst data a spojit data do struktur Sentence, Segment,

Word a Tag. Dále obsahuje pomocnou metodu na načtení souboru jako jednoho stringu, kterou používá třetí metoda pro načtení uložené neuronové sítě.

Třída DataWriter umí kromě reverzních operací k třídě DataReader navíc zapsat reprezentaci třídy Evaluation do textového souboru.

2.2.3 Module DataSplitter

Modul DataSplitter je konzolová aplikace, která poskytuje uživatelské rozhraní pro třídu Splitter.

2.2.4 Module Classifier

Modul Classifier je konzolová aplikace umožňující spouštění úkolů prostřednictvím skriptu ve formě XML souboru. Součástí přílohy této práce je ukázkový skript a pak všechny skripty, na jejichž základě byly vygenerovány výstupy použité v části Vyhodnocení, aby bylo možno výsledky co nejjednodušeji reprodukovat.

Skript je XML soubor, kde kořenovým elementem je element „tasks“ a jeho potomky je libovolné množství elementů „task“. Úkol má několik potomků, kteří jsou určeni typem úkolu. Jedním společným potomkem je tedy element „type“, který může nabývat pěti hodnot:

- read – Úkol má tři parametry (v podobě tří sourozeneckých elementů) „mdir“, „adir“ a „segdir“, které určují adresáře, kde se nacházejí jednotlivé typy souborů.
- split – Má šest parametrů; „train“, „crossvalidation“, „test“, které obsahují přirozená čísla, která určují poměr rozdělení dat a „traindir“, „crossvalidationdir“, „testdir“, které určují adresáře pro uložení rozdělených dat.
- classify – Tento typ má sourozenecký element „method“, který určuje metodu použitou při klasifikaci, a pak další parametry, které specifikují chování konkrétní metody. Ostatní parametry jsou podrobně popsány u každé metody dále v této kapitole.

- evaluate – Nemá žádné parametry, provede vyhodnocení výsledků. Pro funkčnost musí byt načteny věty a provedena na nich klasifikace nějakou metodou.
- write – Má pět parametrů; „mdir“, „adir“, „truesegdir“, „compsegdir“, které určují adresáře pro zápis výsledných souborů. U „segdir“ označuje „true“ zlatá data načtená z PDT a „comp“ data vypočítaná metodami. Parametr „evaldir“ označuje soubor, do kterého budou zapsány výsledky. Má-li libovolný z parametrů prázdnou hodnotu, nebudou daná data zapsána.
- movesep – Bez parametrů, nabízí alternativní metodu, jak určit úrovně separátorů z úrovní sousedních segmentů.

2.2.5 Module Logic

Tento modul obsahuje metody na určování úrovní segmentů, které jsou popsány později v této kapitole. Dále obsahuje statickou třídu Helpers, která poskytuje několik pomocných metod použitých napříč metodami. Jsou to:

- určení, zda segment má přiřazen příznak podřízenosti,
- posun úrovní tak, aby nebyla žádná úroveň vynechána (př.: jsou-li ve větě segmentu na úrovních 1, 2 a 4, posune všechny segmenty z úrovně 4 na úroveň 3),
- posun úrovní tak, aby nejnižší úroveň byla 0,
- úprava úrovní separátorů; jednoduchý separátor posune na nižší z úrovní sousedních segmentů, složený separátor tak, že krajní separátory jsou na stejné úrovni jako sousední segment, ostatní separátory jsou na nižší z úrovní sousedních segmentů.

Cílovou skupinou tohoto produktu nejsou „běžní“ uživatelé, ale technicky zdatní uživatelé. Z tohoto důvodu jsem nezahrnul do implementace ošetřování chyb, protože bych mohl akorát chyby zachytit a vypisovat chybové zprávy a to mi nepřipadalo užitečné. Ze začátku jsem datové objekty navrhl jako mutable (jednotlivé instance objektů lze upravovat i po vytvoření) a veškeré informace zapisoval přímo do objektů obdržených jako parametry metod. Toto se ukázalo jako

nevhodné, protože některé metody pracují lépe, pokud mohou mít více kopií s odlišnými informacemi. Problém jsem vyřešil přidáním metod na vytváření hlubokých kopií objektů, tedy jejich duplikaci.

2.3 Metoda 1

Tato metoda má dvě základní části, určování úrovně segmentu tedy provádí ve dvou krocích. Nejdříve omezí úrovně segmentů jen na určité rozsahy a pak z rozsahu vybere jednu konkrétní úroveň. Architektura metody je uzpůsobení tomu, aby do obou kroků šly zvenčí zapojit nové mechanismy aniž by uživatel měl zdrojové kódy ke knihovně. V této metodě to nepřináší zásadní výhody, ale je to užitečné pro další metody.

K omezení rozsahu úrovní segmentů jsou k dispozici tři způsoby:

- Základní metodou je nastavení minima na 0 a maxima na počet segmentů, což je teoretický rozsah úrovní segmentů v souvětí. Tento rozsah zaručuje stoprocentní správnost, bohužel není moc užitečný.
- Druhá metoda vychází z pravidel definovaných v článcích, na kterých tuto práci zakládám. Pravidla jsou naimplementována tak, jak byla popsána v úvodní části s výjimkou zpracování tečky jako separátoru uprostřed věty (toto bylo přidáno kvůli kompatibilitě s editorem SegView), kde úroveň segmentu zůstává beze změny (odpovídá ignorování separátoru).
- Třetí metoda vychází z druhé, zvyšuje možný rozsah úrovně segmentu o jednu na každou stranu, což zvyšuje úspěšnost určení rozsahu úrovní, ale komplikuje to práci v druhé části této metody.

Aby byl výsledek jednoznačný, je třeba z rozsahu úrovní jednu vybrat. O to se stará druhý krok této metody, k dispozici jsou čtyři způsoby:

- úroveň všech segmentů je nastavena na minimum rozsahu,
- úroveň všech segmentů je nastavena na maximum rozsahu,
- úroveň všech segmentů je nastavena směrem k minimu rozsahu, rozdíl úrovní je však pouze 1 (dovoluje-li to rozsah),
- úroveň všech segmentů je nastavena směrem k maximu rozsahu, rozdíl úrovní je však pouze 1 (dovoluje-li to rozsah).

Třetí a čtvrtý způsob preferují malé skoky v úrovních mezi sousedními segmenty.

BaseClassifier je utilita, která demonstruje použití této metody z kódu.

Pro tuto metodu má skript pro classifier dva parametry:

- „boundscomputation“ - možné hodnoty jsou „max“, „base“, „baseplusone“ (odpovídá pořadí, ve kterém jsou způsoby uvedeny výše),
- „disambiguation“ - možné hodnoty jsou „min“, „max“, „neighbormin“, „neighbormax“ (odpovídá pořadí, ve kterém jsou způsoby uvedeny výše).

2.4 Metoda 2

První metoda přistupovala k souvětí tak, že přesnými pravidly přiřadila segmentům úroveň. Druhá metoda k problému přistupuje z opačné strany. Uvažuje všechna možná segmentační schémata, hodnotí je a vybírá to nejlepší.
K prohledávání prostoru segmentačních schémat používám genetický algoritmus. Jednoduchý horní odhad počtu segmentačních schémat, kde vezmeme v úvahu, že každý segment může být na samostatné úrovni a sousední segmenty nemusí být na sousedních úrovních, vede k exponenciálnímu množství možných řešení.
Genetický algoritmus je heurestická metoda pro prohledávání velkých prostorů možných řešení. Vychází z Darwinovy teorie evoluce, tedy že přežívá lepší jedinec a dva lepší jedinci mají větší šanci mít lepšího potomka. Zde lepší používám vzhledem k nějaké funkci, která jedinci přiřadí jeho kvalitu (fitness).
Vzhledem k použití hodnotící (fitness) funkce je tento přístup vhodný hlavně na problémy, u kterých lze snadno takovou funkci navrhnout (třeba problémy hledání maxima či minima něčeho), což není případ této práce, přesto jsem se tuto metodu pokusil použít.

Teorii genetických algoritmů čerpám z knihy [22] a materiálů k části AI Search modulu Software Applications vyučovaného na University of Durham prof. Iainem Stewartem. Při implementaci jsem vycházel z úkolu, který jsem řešil v rámci tohoto modulu.

```
function GENETIC-ALGORITHM ( population, FITNESS-FN ) returns an individual
    inputs: population, a set of individuals
        FITNESS-FN, a function that measures the fitness of an individual
    repeat
        new_population ← empty set
        loop for i from 1 to SIZE ( population ) do
            x ← RANDOM-SELECTION ( population, FITNESS-FN )
            y ← RANDOM-SELECTION ( population, FITNESS-FN )
            child ← REPRODUCE ( x, y )
            if ( small random probability ) then child ← MUTATE (child)
            add child to new_population
        population ← new_population
    until some individual is fit enough, or enough time has elapased
    return the best individual in population, according to FITNESS-FN

function REPRODUCE ( x, y ) returns an individual
    inputs: x, y, parent individuals
    n ← LENGTH ( x )
    c ← random number from 1 to n
    return APPEND( SUBSTRING ( x, 1, c ) , SUBSTRING ( y, c + 1, n ) )
```

Schéma genetického algoritmu

Figure 4.17 v knize [22]

Pro věty s malým počtem segmentů není genetický algoritmus potřeba, implementace tedy obsahuje parametr počet segmentů, věty s menším počtem jsou řešeny bez genetického algoritmu tak, že jsou procházena všechna možná schémata a vybráno to nejlepší.

Dalším parametrem je funkce určující rozmezí úrovní jednotlivých segmentů (byly popsány u předchozí metody), které jsou na větu aplikovány před během genetického algoritmu, omezí se tím prostor možných řešení.

Na začátku je vygenerována náhodná populace (s omezením na rozmezí úrovní segmentů), velikost populace je dána parametrem metody. Výpočet algoritmu není ukončen nalezením dostatečně dobrého jedince, protože jsem nezadefinoval dostatečně dobré kritérium, jak to poznat. Běh skončí vždy po počtu generací daných vstupním parametrem.

Jedinec je reprezentován jako seznam úrovní segmentů (délka odpovídá počtu segmentů). Vytvoření potomka je prováděno generováním náhodného indexu do seznamu úrovní a spojením dvou části rodičů vymezených tímto indexem. Rodiče jsou vybíráni náhodně s pravděpodobností odpovídající jejich fitness. Toho dosahuji sečtením fitness celé populace, vygenerováním náhodného čísla z intervalu

[0,součet] s přidělením intervalu velikosti fitness každému jedinci. Jako rodič je vybrán ten jedinec, do jehož intervalu padne náhodně generované číslo.
Pro mutaci je náhodně vybrána pozice v jedinci a s pravděpodobností dle parametru je náhodně změněna hodnota na dané pozici. Mutace je provedena u každého potomka tolikrát, kolikrát určuje parametr.
Nejnáročnější u této metody je zvolit vhodnou fitness funkci. Navrhl jsem několik prvků, které lze ve větách ohodnotit a jejich kombinací jsem vytvořil několik fitness funkcí, jejichž úspěšnost budu v další části hodnotit. Hlavním problémem hodnocení je určit, jakou váhu vlastnosti přidělit. Tento problém se snažím adresovat v metodě 3.
Fitness funkci jsem tvořil ohodnocováním věty z hlediska počtu sloves, rozdílů mezi sousedními úrovněmi a toho, jak úrovně odpovídají pravidlům z předchozí metody.

GeneticClassifier je utilita, která demonstruje použití této metody z kódu.
Pro tuto metodu má skript pro classifier následující parametry:

- „boundscomputation“ - možné hodnoty jsou „max“, „base“, „baseplusone“, určuje metodu omezení rozsahů úrovní před spuštěním genetického algoritmu,
- „populationsize“ - přirozené číslo určující velikost populace (jedné generace),
- „generations“ - přirozené číslo určující počet generací,
- „mutationcount“ - počet pokusů o mutaci u každého potomka,
- „mutationchance“ - pravděpodobnost mutace,
- „threshold“ - hranice počtu segmentů, od které se bude aplikovat genetický algoritmus,
- „fitnessfunction“ - specifikuje použitou fitness funkci, jsou k dispozici čtyři - „fone“, „ftwo“, „fthree“ a „ffour“.

2.5 Metoda 3

Ve druhé metodě jsem narazil na problém, kdy jsem vlastnostem souvětí přiřazoval body, ale neuměl jsem ohodnotit význam jednotlivých vlastností. Tento problém se snažím adresovat ve třetí metodě použitím neuronové sítě, kde váhy vlastností neuronová síť odvodí z trénovacích dat. Nevýhodou použití neuronové sítě je, že má pevný počet výstupů, takže kvůli proměnlivé délce souvětí nemohu souvětí hodnotit

jako celek. Musím volit nějaké lokální omezení, což omezuje užitečnost této metody, protože dvě části jedné věty mohou být v češtině libovolně daleko od sebe (odpovídá počtu vložených vět).
Při studiu teorie a implementace neuronových sítí jsem čerpal z knihy [22] a online kurzu Machine Learning vedeného profesorem Andrew Ng ze Stanford University [23]. Při návrhu metody jsem použil implementaci neuronové sítě publikovanou v článku [24], jehož autorem je Andrew Kirillov. Ve finální verzi jsem použil vlastní implementaci neuronové sítě.

Neuronová síť patří do kategorie algoritmů strojového učení. Cílem neuronové sítě je simulovat fungování lidského mozku. Mozek je kolekce neuronů, každý neuron má na vstupu vzruchy z jiných neuronů a jednu výstupní hodnotu, kterou předává dalším neuronům.

Struktura neuronové sítě je oproti mozku zjednodušená. Síť se skládá z několika vrstev neuronů, kde neurony v jedné vrstvě přijímají jako vstup výstup z neuronů předchozí vrstvy a vytváří výstup, který slouží jako vstup do vrstvy následující. Vstupem pro síť je seznam hodnot, které popisují vlastnosti každého exempláře, na jehož klasifikaci chceme neuronovou síť použít. Vstup celé sítě je zároveň vstupem první vrstvy. Výstup každé vrstvy je vstupem pro následující vrstvu a výstup poslední vrstvy je výstupem celé sítě.
Každý neuron v síti přijímá vstup z předchozí vrstvy a pro každý vstup má váhu, se kterou jej zohledňuje. Výstup neuronů lze spočítat jako sumu hodnot na vstupu krát váha každého vstupu. Na výstup se dále aplikuje aktivační funkce, v mé implementaci používám sigmoid funkci $f(x) = 1/(1-e^{-x})$.
Inicializaci neuronové sítě (nastavení výchozích hodnot vah u všech neuronů) provádím náhodně z malého okolí nuly. Hodnoty nenastavuji přesně na nulu, aby nedošlo ke špatnému fungování sítě vinou symetrických vah na začátku.
Před použitím sítě pro klasifikaci je třeba ji naučit funkci, která vztah reprezentuje. K tomu je třeba množina exemplářů a k nim správná řešení, z nichž se dá síť vytrénovat. V této práci používám metodu zpětného učení, která nejdříve vždy klasifikuje exemplář za použití aktuální konfigurace sítě a následně v opačném směru (proto tento název) upraví hodnoty vah u neuronů, tak aby byla chyba v síti menší. Zpětné učení postupně odzadu spočítá chybu, jakou síť udělala při klasifikaci

(srovnáním vypočítané a správné hodnoty), podívá se, jakou měrou k chybě přispěly jednotlivé vstupy neuronu a náležitě upraví váhy vstupů. Tento proces je opakován v cyklech na každém exempláři dokud neuplyne předem určená doba, nebo chyba neklesne pod určitou úroveň, nebo síť dorazí do (lokálního) minima a již nedochází k dalšímu zlepšování.

Dostatečně komplexní neuronovou sítí lze reprezentovat spojité i nespojité funkce, je však obtížné říci, jakou funkci lze reprezentovat nějakou konkrétní strukturou sítě ([22] Chapter 20).

function BACK-PROP-LEARNING (*examples*, *network*) **returns** a neural network
inputs: *examples*, a set of examples, each with input vector **x** and output vector **y**
network, a multilayer network with *L* layers, weights Wj,i, activation function g
repeat
for each *e* **in** *examples* **do**
for each node *j* in the input layer **do** $a_j \leftarrow x_j[e]$
for l=2 **to** L **do**
$in_i \leftarrow \sum_j W_{j,i}\, a_j$
$a_i \leftarrow g(in_i)$
for each node *i* in the output layer **do**
$\Delta_i \leftarrow g'(in_i) * (y_i[e] - a_i)$
for l=L-1 **to** 1 **do**
for each node *j* in layer *l* **do**
$\Delta_j \leftarrow g'(in_j) * \sum_i W_{j,i}\, \Delta_i$
for each node *i* in layer l+1 **do**
$W_{j,i} \leftarrow W_{j,i} + \alpha * a_j * \Delta_i$
until some stopping critecion is satisfied
return NEURAL-NET-HYPOTHESIS (*network*)

Schéma algoritmu zpětného učení
Figure 20.25 v knize [22]

První for-cyklus prochází všechny exempláře, druhý. a třetí provádí výpočet sítě, čtvrtý počítá chybu na poslední vrstvě, další dva počítají chyby na předchozích vrstvách a poslední náležitě upravuje váhy u neuronů.

Kvůli omezení na konstantní počet výstupů sítě je na jednom souvětí spouštěna klasifikace sítí několikrát, pro každý segment jednou.

Vstupem a výstupem neuronové sítě je pole čísel, je tedy třeba jej ze souvětí vyrobit a následně výstup převést v segmentační schéma. Jedním z parametrů sítě je objekt,

který toto mapování provádí. Je specifikován rozhraním IFeatureMapper, které definuje pět metod, které musí objekt implementovat:

- počet vstupních parametrů,
- počet výstupních parametrů,
- mapování souvětí na vstup sítě,
- mapování souvětí na výstup sítě,
- mapování výstupu sítě na souvětí.

Údaje 1 a 2 specifikují velikost vstupní a výstupní vrstvy sítě.

U mapování vstupu (3) je definováno několik vlastností, každá vlastnost má ve vstupu svou pozici. Na dané pozici bude 1, pokud souvětí (vzhledem k aktuálně zpracovávanému segmentu) danou vlastnost má, jinak 0. Konkrétně je popsáno u mapovacích tříd.

Mapování souvětí na výstup (4) je prováděno změnou úrovně segmentu oproti předcházejícímu segmentu (nebo 0 u prvního segmentu). Kvůli konstantnímu počtu výstupů je však rozdíl úrovní shora omezen, což znemožňuje správně klasifikovat všechny věty.

Př.: Je-li změna úrovně mezi dvěma segmenty rovna +1 a rozsah sítě [-2,2], pak výstup bude [0,0,0,1,0].

Zpětné mapování (5) je definováno tak, že výstup s největší hodnotou dostane přiřazeno 1, zbytek 0. Pak se určí změna úrovně opačným postupem k (4).

Př.: Je-li výstup [0.1,0.2,0.7,0.3,0.05], pak je upraven na [0,0,1,0,0] a úroveň je tedy zachována (změna 0).

V současné implementací jsou k dispozici dvě mapovací třídy. Následuje seznam tříd a vlastnosti, které třída zaznamenává:

- „fmone“
    - zda je segment první v souvětí,
    - zda má segment přiřazen PP,
    - zda je předchozí segment na úrovní 0,
    - zda segmentu předchází konkrétní separátor (jedna vlastnost pro každý typ separátoru),
    - zda segmentu předchází více separátorů,
    - zda segment obsahuje sloveso,
    - zda předchozí segment obsahuje sloveso.

- „fmtwo“
    - všechny vlastnosti fmone a navíc:
        - zda je segment poslední v souvětí,
        - zda dva předchozí a dva následující segmenty mají přiřazen PP,
        - zda dva předchozí a dva následující segmenty obsahují sloveso.
- „fmthree“
    - oproti fmtwo navíc rozlišuje jednotlivé separátory, nejen jejich skupiny.

Vzhledem k tomu, že veškeré výpočty probíhají zleva doprava a úroveň segmentu nezávisí na vlastnostech segmentů následujících, neočekávám zde velkou úspěšnost. Úplně irelevantním vlastnostem však neuronová síť při učení může přiřadit váhu 0 (a tím vlastnost ignorovat).

NeuralClassifier je utilita, která demonstruje použití této metody z kódu.
Pro tuto metodu má skript pro classifier parametr „action“, který určuje činnost, kterou bude neuronová síť provádět:

- „train“ - Vytvoří a vytrénuje neuronovou síť, k učení použije aktuálně načtené věty. Vyžaduje přítomnost následujících parametrů:
    - „featuremapper“ - zabudovaná třída, která provádí mapování souvětí na vstup sítě a výstupu sítě na segmentační schéma, jsou k dispozici dvě „fmone“ a „fmtwo“,
    - „learningrate“ - parametr určující rychlost učení sítě, čím menší, tím přesnější je učení a tím menší pravděpodobnost, že síť bude divergovat, čím větší, tím rychleji se síť učí,
    - „requiredimprovement“ - vyžadované zlepšení chyby v každé iteraci učení sítě, je-li zlepšení menší, učení je ukončeno (odpovídá nalezení lokálního minima),
    - „learningiterations“ - maximální počet iterací při učení sítě,
    - „hiddenlayercount“ - počet skrytých vrstev v neuronové síti,
    - „hiddenlayersize“ - počet neuronů v každé skryté vrstvě sítě.

- „save“ - Uloží vytrénovanou neuronovou síť do souboru, který je určen parametrem „path“.
- „load“ - Načte vytrénovanou neuronovou síť ze souboru, který je určen parametrem „path“.
- „compute“ - Nemá žádné parametry, provede klasifikaci na aktuálně načtených větách.

3 Výsledky

V této části hodnotím výsledky metod na testovacích datech. Nejdříve srovnávám jednotlivé varianty metod mezi sebou a následně metody navzájem.

V příloze práce je více souborů s výsledky běhu metod, zde popisuji pouze nejlepší z nich pro danou metodu. Pro každou metody jsem vybral čtyři varianty a formou tabulky prezentuji výsledky každé z těchto variant na trénovacích a testovacích datech. Testovací data uvádím proto, že jsou přímo určena k vyhodnocení. Trénovací proto, že je jich více a proto mají jejich výsledky větší vypovídající hodnotu (u neuronové sítě nikoli, protože na těchto datech byla síť naučena).

V následující tabule jsou základní informace o datech, na kterých bylo vyhodnocení provedeno. V řádcích tabulky jsou nejdříve testovací, pak trénovací data. Ve sloupcích je celkový počet souvětí, počet segmentů, počet vztahů mezi segmenty (odpovídá změně úrovně mezi dvěma sousedními segmenty) a pak počet segmentů na jednotlivých úrovních ve zlatých datech z PDT. Úrovně jsou uvedeny pouze do úrovně 5, na vyšších úrovních již žádná data nebyla.

Data	**Vět**	**Segmentů**	**Vztahů**	**0**	**1**	**2**	**3**	**4**	**5**
Test	250	945	695	616	261	65	3	0	0
Train	2005	7808	5803	4979	2274	467	73	11	4

Tabulka 2 – Data

V následujících oddílech jsou čtyři tabulky se společným formátem. Ve sloupcích jsou jednotlivé testovací případy, v řádcích jsou data, vše je zaokrouhleno na dvě desetinná místa.

Význam jednotlivých řádků:

- úspěšnost vět – podíl vět, kde jsou všechny segmenty určeny správně,
- úspěšnost segmentů – podíl správně určených úrovní segment,
- úspěšnost změn úrovní – podíl správně určených změn úrovně segmentu oproti segmentu jemu předcházejícímu,
- úspěšnost vztahů – podíl správně určených vztahů dvou sousedních segmentů (změna úrovně je kladná, nula, nebo záporná),

- úspěšnost rozsahů – podíl správně určených rozsahů segmentů při předzpracování dat (nebylo prováděno u neuronové sítě),
- úspěšnost segmentů (tolerance jedna úroveň) – podíl správně určených úrovní segmentů s tolerancí ±1,
- data pro jednotlivé úrovně:
    - precision,
    - recall,
    - f-score.

Následující tabulka označuje možné výsledky klasifikace segmentu vůči úrovní x (př.: 1).

	Správná úroveň je x	**Správná úroveň není x**
Určená úroveň je x	True positive (tp)	False positive (fp)
Určená úroveň není x	False negative (fn)	True negative (fn)

Tabulka 3 – Rozdělení chyb pro úroveň x

Precision (p) je pravděpodobnost, že určená úroveň x je správně. Lze ji spočítat jako tp / (tp + fp). Recall (r) je pravděpodobnost, že segment na úrovní x dostane úroveň x přidělenu. Lze ji spočítat jako tp / (tp + fn). F-Score je harmonický průměr precision a recall. Lze je spočítat jako 2 * p * r / (p + r).

3.1 Metoda 1

První řádek tabulky popisuje variantu metody, jako první je použitá sada dat, pak metoda určení rozsahů úrovní a nakonec metoda určení konkrétní úrovně. Metody pro určení rozsahu “max” a “baseplusone” měly velice malou úspěšnost, v tabulce proto prezentuji výsledky metody “base” se všemi variantami určení konkrétní úrovně.

Metoda 1	**Test base max**	**Test base min**	**Test base neighbormax**	**Test base neighbormin**	**Train base max**	**Train base min**	**Train base neighbormax**	**Train base neighbormin**
Úspěšnost vět	43,2%	48,4%	43,2%	48,4%	42,93%	48,38%	42,93%	47,88%
Úspěšnost segmentů	72,06%	77,35 %	72,06 %	76,6%	68,84%	74,82%	68,87%	74,36%
Úspěšnost změn úrovní	67,48%	74,96 %	67,63 %	47,24 %	68,6%	75,1%	68,53%	73,41%
Úspěšnost vztahů	70,5%	77,7%	70,5%	76,4%	70,22%	76,65%	70,15%	74,78%
Úspěšnost rozsahů	86,56%	86,56 %	86,56 %	86,56 %	86,67%	86,67%	86,67%	86,67%
Úspěšnost segmentů (tolerance 1 úroveň)	92,91%	94,7%	92,91 %	94,71 %	92,11%	95,15%	92,12%	95,17%
0	0,81 0,87 0,84	0,79 0,96 0,87	0,81 0,86 0,84	0,8 0,95 0,87	0,79 0,84 0,81	0,76 0,96 0,85	0,79 0,84 0,81	0,77 0,95 0,85
1	0,64 0,48 0,55	0,8 0,46 0,58	0,64 0,48 0,55	0,77 0,46 0,57	0,62 0,45 0,52	0,78 0,41 0,53	0,62 0,45 0,52	0,74 0,42 0,53
2	0,42 0,37 0,39	0,48 0,29 0,36	0,42 0,37 0,39	0,48 0,29 0,36	0,35 0,36 0,35	0,53 0,29 0,37	0,35 0,36 0,35	0,51 0,29 0,37
3	0 0 0	0 0 0	0 0 0	0 0 0	0,03 0,18 0,05	0,04 0,09 0,06	0,16 0,44 0,23	0,22 0,18 0,2
4	N/A	N/A	N/A	N/A	0 0 0	0 0 0	0,03 0,18 0,05	0,03 0,09 0,05
5	N/A	N/A	N/A	N/A	0 0 0	0 0 0	0 0 0	0 0 0

Tabulka 4 – Výsledky metody 1

Úspěšnost této metody řádově odpovídá výsledkům z odkazovaných článků.

3.2 Metoda 2

První řádek tabulky popisuje variantu metody, jako první je použitá sada dat, pak metoda určení rozsahů úrovní a nakonec fitness funkce.

Metoda 2	Test base fone	Test base ftwo	Test baseplu sone fthree	Test basepl usone ffour	Train base fone	Train base ftwo	Train baseplu sone fthree	Train basepl usone ffour
Úspěšnost vět	67,2%	66,8%	62,4%	63,6%	65,34%	64,89 %	57,76%	60,45 %
Úspěšnost segmentů	79,68 %	79,89 %	76,4%	77,25 %	77,52%	77,19 %	73,17%	73,66 %
Úspěšnost změn úrovní	81,15 %	80,72 %	73,23%	72,52 %	79,32%	79,25 %	70,53%	71,77 %
Úspěšnost vztahů	83,59 %	83,45 %	76,55%	75,54 %	79,32%	80,94 %	73,08%	74,07 %
Úspěšnost rozsahů	86,56 %	86,56 %	96,08%	96,08 %	86,67%	86,67 %	97,25%	97,25 %
Úspěšnost segmentů (tolerance 1 úroveň)	92,91 %	93,12 %	92,91%	93,12 %	94,63%	94,48 %	93,6%	93,9%
0	0,87 0,91 0,89	0,87 0,91 0,89	0,89 0,84 0,87	0,89 0,85 0,87	0,84 0,91 0,88	0,84 0,9 0,87	0,86 0,83 0,85	0,86 0,84 0,85
1	0,8 0,62 0,7	0,8 0,62 0,7	0,69 0,63 0,66	0,7 0,63 0,66	0,75 0,57 0,65	0,74 0,57 0,64	0,65 0,58 0,61	0,65 0,59 0,62
2	0,44 0,48 0,46	0,46 0,48 0,47	0,41 0,57 0,48	0,47 0,58 0,52	0,43 0,44 0,43	0,43 0,44 0,43	0,35 0,43 0,39	0,34 0,45 0,39
3	0,06 0,33 0,1	0,05 0,33 0,09	0,05 0,33 0,09	0,05 0,33 0,08	0,14 0,27 0,19	0,14 0,26 0,18	0,13 0,36 0,19	0,13 0,36 0,2
4	N/A	N/A	N/A	N/A	0,04 0,18 0,06	0,03 0,18 0,05	0,02 0,09 0,03	0,03 0,18 0,05
5	N/A	N/A	N/A	N/A	0 0 0	0 0 0	0 0 0	0 0 0

Tabulka 5 – Výsledky metody 2

Úspěšnost jednotlivých variant algoritmu je velmi podobná. U testovacích dat se dokonce podařilo dosáhnout nenulového f-score na všech úrovních.

3.3 Metoda 3

První řádek tabulky popisuje variantu metody, jako první je použitá sada dat, pak metoda mapování souvětí na vstup a výstup neuronové sítě. Zkoušel jsem trénovat různé struktury neuronových sítí a na „cross validation“ datech jsem zkoumal, které

vedou k nejlepším výsledkům. U složitých sítí se mi nepodařilo dosáhnout dobré úspěšnosti, většinou docházelo k určení všech úrovní segmentů na nulu. Všechny sítě v následující tabulce měly pouze jednu skrytou vrstvu obsahující 30 až 40 neuronů.

Metoda 3	**Test fmone**	**Test fmone**	**Test fmtwo**	**Test fmthree**	**Train fmone**	**Train fmone**	**Train fmtwo**	**Train fmthree**
Úspěšnost vět	48,4%	46,8%	39,2%	38,8%	48,63 %	46,43 %	41,9%	40,45%
Úspěšnost segmentů	74,29 %	72,91%	64,97 %	64,76 %	75,52 %	73,39 %	64,74%	63,92%
Úspěšnost změn úrovní	73,81 %	72,95%	58,41 %	54,53 %	75,71 %	74,25 %	62,36%	56,21%
Úspěšnost vztahů	76,11 %	75,4%	58,7%	55,82 %	77,08 %	75,6%	62,73%	57%
Úspěšnost segmentů (tolerance 1 úroveň)	93,54 %	94,07%	92,38 %	94,39 %	96,11 %	95,63 %	94,12%	94,61%
0	0,8 0,9 0,85	0,82 0,86 0,84	0,67 0,98 0,79	0,7 0,93 0,8	0,79 0,92 0,85	0,8 0,87 0,83	0,66 0,99 0,79	0,69 0,91 0,79
1	0,68 0,49 0,57	0,62 0,52 0,57	0,5 0,05 0,08	0,4 0,14 0,21	0,72 0,49 0,58	0,64 0,54 0,58	0,44 0,04 0,08	0,44 0,15 0,22
2	0,46 0,29 0,36	0,42 0,34 0,37	0,09 0,02 0,03	0,17 0,08 0,1	0,5 0,36 0,41	0,42 0,36 0,39	0,34 0,08 0,13	0,27 0,18 0,22
3	0 0 0	0 0 0	0 0 0	0 0 0	0,27 0,36 0,31	0,24 0,34 0,28	0,13 0,04 0,06	0,12 0,15 0,13
4	N/A	N/A	N/A	N/A	0,04 0,09 0,06	0,04 0,09 0,06	0,2 0,018 0,19	0,04 0,09 0,05
5	N/A	N/A	N/A	N/A	0 0 0	0 0 0	0 0 0	0 0 0

Tabulka 6 – Výsledky metody 3

Chyby v neuronové síti mohou být způsobeny dvěma okolnostmi. Buď jsem nenavrhl síť s dostatečně komplexní strukturou, která by mohla správně reprezentovat vztah vstupu a výstupu, nebo vstup nedostatečně charakterizoval vlastnosti věty, takže z něj nebylo možné správný vztah vyvodit.

3.4 Srovnání metod

První je dvakrát metoda 1, pak dvakrát metoda 2 a dvakrát metoda 3.

Srovnání metod	**Test base min**	**Train base min**	**Test base fone**	**Train base fone**	**Test fmone**	**Train fmone**
Úspěšnost vět	48,4%	48,38%	67,2%	65,34%	48,4 %	48,63 %
Úspěšnost segmentů	77,35 %	74,82%	79,68 %	77,52%	74,29 %	75,52 %
Úspěšnost změn úrovní	74,96 %	75,1%	81,15 %	79,32%	73,81 %	75,71 %
Úspěšnost vztahů	77,7%	76,65%	83,59 %	79,32%	76,11 %	77,08 %
Úspěšnost rozsahů	86,56 %	86,67%	86,56 %	86,67%	N/A	N/A
Úspěšnost segmentů (tolerance 1 úroveň)	94,7%	95,15%	92,91 %	94,63%	93,54 %	96,11 %
0	0,79 0,96 0,87	0,76 0,96 0,85	0,87 0,91 0,89	0,84 0,91 0,88	0,8 0,9 0,85	0,79 0,92 0,85
1	0,8 0,46 0,58	0,78 0,41 0,53	0,8 0,62 0,7	0,75 0,57 0,65	0,68 0,49 0,57	0,72 0,49 0,58
2	0,48 0,29 0,36	0,53 0,29 0,37	0,44 0,48 0,46	0,43 0,44 0,43	0,46 0,29 0,36	0,5 0,36 0,41
3	0 0 0	0,04 0,09 0,06	0,06 0,33 0,1	0,14 0,27 0,19	0 0 0	0,27 0,36 0,31
4	N/A	0 0 0	N/A	0,04 0,18 0,06	N/A	0,04 0,09 0,06
5	N/A	0 0 0	N/A	0 0 0	N/A	0 0 0

Tabulka 7 – Srovnání metod

Nejvíce segmentů se nachází na nižších úrovních, všechny metody měly na nižších úrovních nejlepší úspěšnost, na dolních úrovních byla úspěšnost velice malá. Vzhledem k tomu, že všechny metody používaly k určení úrovně aspoň částečně postup zleva doprava a chyba určení úrovně mohla ovlivnit správnost úrovně následující, nelze sledovat pouze absolutní úroveň segmentu, je třeba přihlížet i

k úspěšnosti ve změnách úrovně (proto jsou tyto statistiky zahrnuty). V datech v příloze (a součástí programu) je ještě vyhodnocení jednotlivých změn úrovní sousedních segmentů prostředníctvím precision / recall / f-score.

3.5 Diskuze výsledků

Úspěšnosti na jednotlivých úrovních mohly být ovlivněny tím, kolik segmentů se na jednotlivých úrovních vyskytovalo v trénovacích datech. V řádcích následující tabulky jsou precision, recall a f-score pro triviální metody definované tak, že přiřadí každému segmentu úroveň odpovídající číslu v prvním sloupci.

	Trénovací data			Testovací data		
Úroveň	Počet	Precision	F-Score	Počet	Precision	F-Score
0	4976	0,86	0,92	616	0,89	0,94
1	2274	0,39	0,56	261	0,38	0,55
2	467	0,08	0,14	65	0,09	0,15
3	73	0,01	0,02	3	0,00	0,00
4	11	0,00	0,00	0	N/A	N/A
5	4	0,00	0,00	0	N/A	N/A

Tabulka 8 – Triviální metody

Metoda s úrovní x přiřazuje všem segmentům úroveň x. Recall na úrovni x je 1. Recall na ostatních úrovních je 0, precision a f-score na ostatních úrovních nejsou definovány.

Výsledky metod jsou s těmito triviálními řádově stejné, na úrovní 0 horší, jinak lepší. Je třeba si však uvědomit, že metody dosahují těchto úspěšností na všech úrovních najednou, kdežto předchozí tabulka popisuje šest různých triviálních metod. Celkově tedy metody navržené v této práci dosahují lepších výsledku než triviálně navržené metody.

Úspěšnost metod však zdaleka není stoprocentní. Následující příklad ilustruje, proč bez analýzy významu souvětí nelze navrhnout metody, které by dosáhly stoprocentní úspěšnosti.

Souvětí 1: Chlapec vstoupil do domu, který měl pěkný vchod, který byl lemován prahem, který byl zelený, a bydlel v něm jeho kamarád.
Souvětí 2: Chlapec vstoupil do domu, který měl pěkný vchod, který byl lemován prahem, který byl zelený, a měl kovové výztuhy.

Souvětí nejsou stylisticky zajímavá a jejich obsah není klíčový. Zásadní je jejich struktura. Obě souvětí mají pět vět (oddělovače jsou čárky a souřadící spojka „a“), stejný počet segmentů, tytéž separátory a stejně přiřazené příznaky podřízenosti. První věta je hlavní na úrovni 0. Druhá věta je vedlejší na úrovní 1, rozšiřuje člen „domu“. Třetí věta je vedlejší na úrovní 2, rozšiřuje člen „vchod“. Čtvrtá věta je vedlejší na úrovni 3, rozšiřuje člen „prahem“. Poslední (pátá) věta je v obou souvětích vedlejší. V prvním souvětí je na úrovni 2 a rozšiřuje člen „domu“. Ve druhém souvětí je na úrovní 3 a rozšiřuje člen „vchod“. Jsem si vědom toho, že toto přidělení úrovně a závislosti není jednoznačné, závisí na interpetaci souvětí a odráží to, jakou myšlenku jsem těmito souvětími chtěl sdělit. Tedy v případě těchto souvětí nelze úrovně určit jednoznačně ani z jejich významu, závisí na kontextu. Druhou možností je, že význam souvětí by jednoznačně určoval závislosti a úrovně, struktura však ne. Pátá věta by dokonce (při zachování struktury) mohla být hlavní na úrovni 0 (třeba „a po chvíli opět vyšel ven.“).
Tento příklad ukazuje, že pro rozlišení některých jevů je nutné studovat i význam souvětí. Ani to však nemusí stačit, protože samotná podstata sdělení vyjádřeného souvětím v přirozeném jazyce nemusí být jednoznačná. Proto tedy nelze navrhout stoprocentně úspěšné metody.

Metoda 1 má dvě fáze. První fáze určuje možné rozsahy úrovní segmentů. Úspěšnost této fáze je asi 86%, tedy již tato fáze zavádí chyby, dokud nebude stoprocentní, nelze uspět v druhé fázi. Odkazovaný výzkum popisoval pouze určení rozsahů, pro druhou fázi jsem tedy navrhl metody vybrání konkrétní úrovně z rosahu. Nepodařilo se mi však najít vhodnou metodu, protože nejlepší se ukázala metoda přiřazení minima z rozsahu úrovně segmentu, což není založeno na vlastnostech souvětí. Úspěch minima je tedy spíše náhodný.

Výhodou druhé metody je její globální pohled na věc, hodnotí souvětí jako celek, úskalím je fitness funkce, jíž je obtížné zadefinovat. Ideální fitness funkce má jedno

globální maximum, které je přiřazeno všem správným segmentačním schématům (s přihlédnutím pouze na strukturu, nikoli obsah souvětí). Fitnes funkci jsem zadefinoval jako jakési skóre věty, kde za každý pozitivní jev je skóre zvýšeno o nějakou hodnotu. Bohužel tato hodnota by měla být jiná pro každou vlastnost dle její váhy. V této metodě se mi nepodařilo najít vhodný mechanismus nalezení vah pro jednotlivé vlastnosti. Ve všech sledovaných statistikách je tato metoda nejlepší, ačkoli si nemyslím, že vlastnosti ve fitness funkci vystihují souvětí přesně. Lepší výsledek přisuzuji zahrnutí globálního pohledu na souvětí (hlavně sledování sloves na jednotlivých úrovních).

Zásadní problém třetí metody spočívá v tom, že neuronová síť má konstantní počet vstupů a tedy nemůže dobře zachytit celkový pohled na souvětí, protože to má proměnlivý počet segmentů. Proto jsem neuronovu sítí klasifikoval jednotlivá rozhraní segmentů a nikoli celé souvětí. Dalším problémem je popis segmentů pomocí vlastností na vstupu sítě. Je třeba zajistit, aby hranice dvou segmentů s odlišnou změnou úrovně nebyly popsány stejnými hodnotami parametrů na vstupu. Zjednodušeně rečeno neuronová síť může reprezentovat pouze funkce, a proto je třeba zajistit, aby opravdu ke každým dvěma různým výstupům vedly dva různé vstupy.

Výběr vlastností, kterými je popsáno souvětí je velice důležitý pro metody 2 i 3. Jedině pokud vlastnosti zachytí jevy, které jsou původcem konkrétní struktury souvětí, může metoda uspět.

Klíčovými problémy metody 2 a 3 jsou chybějící váhy vlastností (bezvýznamná vlastnost může přebít významnější) a lokální pohled na souvětí (v češtině lze očekávat aspoň jedno sloveso na každé úrovni, což však lokálně u neuronové sítě zachytit nelze). Proto jsem se na základě tohoto vyhodnocení rozhodl zkusit naimplementovat čtvrtou metodu, která vhodně zkombinuje metodu 2 a 3 tak, aby byly eliminovány oba nedostatky.

Problémem genetické metody byla vhodná definice fitness funkce. Proto jsem fitness souvětí zadefinoval jako 1 – konstanta * $\Sigma_{segmenty}$ velikost chyby v úrovni segmentu.

Př.: Souvětí, kde dva segmenty jsou umístěny o dvě úrovně výše budou mít fitness 0,96 při hodnotě konstanty 0,01.

Hodnota fitness funkce bohužel nelze spočítat bez znalosti správného segmentačního schématu souvětí, vzniká zde kruhová závislost. Rozhodl jsem se tedy fitness funkci naučit pomocí neuronové sítě. Díky použití sigmoid funkce v síti má síť výstup z intervalu [0,1], což dobře koresponduje s definicí fitness funkce. Data pro učení lze jednoduše vygenerovat ze zlatých dat PDT.

Toto řeší chybějící váhy u genetické metody, ale stále zde zůstává problém konstantního počtu vstupů neuronové sítě. Jako vstupy sítě jsem zvolil parametry fitness funkcí z metody 2 a výstup neuronové sítě z metody 3. Výstup neuronové sítě bylo pole čísel, kde každé reprezentovalo „pravděpodobnost“, že mezi segmenty došlo ke změně úrovní odpovídající pozici v poli. Výsledkem pak byla ta změna, která měla na odpovídající pozici pole největší hodnotu (správná řešení pro učení sítě měla u správné změny 1 a ve zbytku pozic 0). Protože fitness funkce hodnotí navržené úrovně pro segmenty, beru jako jeden parametr součet výstupů z opakovaného běhu neuronové sítě na každé rozhraní segmentů. Tedy součet pravděpodobností, že změna daná vstupní větou je správná. Tento jeden parametr tedy zachycuje informace o všech rozhraních segmentů a tak se pokouší řešit problém konstantního počtu vstupů. Hodnota odpovídá tomu, jak dobré jsou celkově změny úrovní dle neuronové sítě. Pro výpočet je možné použít libovolnou vytrénovanou síť z metody 3 (použil jsem tu s nejlepšími výsledky).

Vyzkoušel jsem několik variant této metody. Podařilo se mi překonat úspěšnost metod 1 a 3 a přiblížit se k úspěšnosti metody 2, ale ne ji překonat. Domnívám se, že neúspěch spočívá v součtu ohodnocení změn úrovní u segmentů (tento přístup jsem zvolil, abych dosáhl konstantního počtu vlastností na vstupu neuronové sítě), toto číslo nevystihuje dobře vlastnosti jednotlivých hranic. Problém konstantního počtu vstupů neuronové sítě zůstává nevyřešen.

Zvýšení míry použitelnosti metod lze dosáhnout dvěma způsoby. Prvně zlepšením vnitřních mechanismů (přesnější pravidla u metody 1 a lepší vlastnosti u metod 2 a

3) a pak navržením metody, která rozhodne, zda je souvětí vhodné pro konkrétní metodu a používat metodu pouze na ta souvětí, na kterých spíše uspějí.

V této práci jsem naimplementoval několik metod, software umožňuje zaznamenat výsledky metody a srovnat je. Další experimenty bych založil na zkoumání, jak se metody chovají na jednotlivých separátorech. Tedy rozdělit souvětí do skupin podle obsaženého typu separátoru (jedno souvětí může být ve více skupinách), v rámci skupiny pak rozdělit na správně a špatně určené úrovně vzhledem k separátoru s zkusit najít pravidla či vlastnosti (dle metody), které povedou k správnému výsledku při zachování výsledků na předtím správně určených souvětích. Jak je vidět i na příkladu v této kapitole, najít začátek vložené klauze a určit její úrovneň je celkem snadné, problém je ve změně úrovně po konci klauze, protože je obtížné určit pouze ze struktury, do jaké klauze následující segment patří a někdy to nelze určit ani z významu věty. Pro určení změny úrovně po konci klauze zjevně nestačí lokální informace o sousedních segmentech, metody 1 a 3 tento problém tedy nemohou obecně úspěšně řešit.

4 Závěr

4.1 Shrnutí

V první části jsem stručně popsal vybrané jevy české gramatiky, zdrojová data a PDT a výzkum, na jehož výsledcích jsem tuto práci založil.

Navrhl jsem tři metody automatického určení úrovně segmentů v souvětích. Metodu, která vycházela přímo z pravidel předložených v odkazovaném výzkumu, dále metodu využívající genetický algoritmus a metodu založenou na použití neuronové sítě.

Metody jsem naimplementoval na platformě Mono v jazyce C#. V práci jsem popsal návrh struktury programu a jeho implementaci.

Metody, každou v několika variantách, jsem spustil na dostupných datech a shromáždil informace o výstupních datech. Tyto informace jsem předložil a popsal v třetí části práce.

Co se týče úspěšnosti, byly na tom jednotlivé metody přibližně stejně. Na celých souvětích fungovala nejlépe metoda genetická. Na úrovni jednotlivých segmentů se úspěšnost metod výrazně nelišila. Všechny metody měly větší úspěšnost při určování segmentů na nižších úrovních, proto jsem do výsledků zařadil pouze ty varianty metod, které neurčovaly úroveň segmentu jako konstantu, ačkoli úspěšnost takové metody by byla přes 60%. Kvůli postupu určování úrovní zleva doprava byly úrovně segmentů ovlivněny chybou u segmentů předchozích, proto jsem zařadil rovněž statistiky změny úrovně segmentu vůči segmentu předcházejícímu, kde se tato chyba neprojevila.

Já osobně jsem nejvíce spokojen s třetí metodou, protože se mi zde podařilo naimplementovat neuronovou síť a učící algoritmus a i namapovat souvětí na vstup a výstup sítě tak, že jsem dospěl k nenulové úspěšnosti celé metody.

Dle mého názoru výsledky nejsou špatné, ale na praktické použití metod bez kontroly anotátorem to zatím nestačí. Dále bych se zaměřil na věty a segmenty, na kterých metody selhaly a zkusil bych najít důvody ve struktuře, které tyto chyby způsobují. Nalezením důvodů špatné klasifikace a jejich aplikováním do metod by mohlo dojít k zvýšení celkové úspěšnosti. Nelze však dosáhnout úplné správnosti, protože přirozený jazyk je sám o sobě nejednoznačný, a proto řešení nemusí existovat.

4.2 Odkazy, použitá literatury

[1] Melichar, Jiří a Styblík, Vlastimil. *Český jazyk.* 3. doplněné vydání. Praha: Státní pedagogické nakladatelství, 1970. Publikace číslo 01-28-03.

[2] Ústav pro jazyk český Akademie věd ČR. *Internetová jazyková příručka.* http://prirucka.ujc.cas.cz/

[3] doc. PhDr. František Daneš, DrSc., a PhDr. Zdeněk Hlavsa, CSc., (vedoucí autorského kolektivu), doc. PhDr. Miroslav Grepl, CSc., a kolektiv. *Mluvnice češtiny (3) Skladba.* 1. vydání. Praha: Academia, 1987.

[4] doc. RNDr. Vladislav Kuboň, Ph.D. materiály k přednášce Úvod do počítačové lingvistiky

[5] RNDr. Daniel Zeman, Ph.D. *Neprojektivity v PDT.* http://ufal.mff.cuni.cz/~zeman/projekty/neproj/index.html

[6] Markéta Lopatková, Zdeněk Žabokrtský, Václava Kettnerová. *VALLEX 2.5: Valency Lexicon of Czech Verbs.* http://ufal.mff.cuni.cz/vallex/2.5/doc/home.html

[7] Ústav formální a aplikované lingvistiky, kolektiv autorů. *Pražský závislostní korpus 2.0.* http://ufal.mff.cuni.cz/pdt2.0/index-cz.html

[8] Petr Sgall, E. Hajičová, J. Panevová. *The meaning of the sentence in its semantic and pragmatic aspects*. Dordrecht: Reidel and Prague: Academia 1986.

[9] Vladislav Kuboň, Markéta Lopatková, Martin Plátek, Patrice Pognan: *O segmentaci českých vět.* In: Proceedings of ITAT 2006 (Information Technologies - Application and Theory), Copyright © Univerzita Pavla Jozefa Šafárika, Košice, Slovakia, Košice, Slovakia, ISBN 80-969184-4-3, pp. 81-86, 2006

[10] Kuboň, V., Lopatková, M., Plátek, M., Pognan, P.: *A Linguistically-Based Segmentation of Complex Sentences.* In Wilson, D.C., Sutcli e, G.C., eds.: Proceedings of FLAIRS 2007 Conference, Menlo Park, AAAI Press (2007) 368-374

[11] Lopatková, M., Holan, T.: *Vztahy mezi segmenty - segmentační schémata českých vět.* In Vojtáš, P., ed.: Proceedings of ITAT 2008, Košice, University of P. J.Šafárika (2007) 15-22;

[12] Kuboň, V., Lopatková, M.: *Od segmentů ke klauzím v češtině - analýza vybraných jevů*, In Vojtáš, P., ed.: Proceedings of ITAT 2010, Košice, University of P. J.Šafárika.

[13] Petr Pajas. *Dokumentace Prague Markup Language.* http://ufal.mff.cuni.cz/jazz/pml/doc/

[14] *Dokumentace k morfologické anotaci v PDT.* http://ufal.mff.cuni.cz/pdt2.0/doc/manuals/en/m-layer/html/ch02s02s01.html

[15] *Dokumentace k analytické anotaci v PDT.* http://ufal.mff.cuni.cz/pdt2.0/doc/manuals/cz/a-layer/html/ch03s02.html

[16] Microsoft Visual C#. http://msdn.microsoft.com/en-us/vstudio/hh388566

[17] .NET Framework Developer Center. http://msdn.microsoft.com/en-us/netframework

[18] mono. http://mono-project.com/

[19] MonoDevelop. http://monodevelop.com/

[20] Microsoft Visual Studio. https://www.microsoft.com/visualstudio/en-us

[21] LINQ to XML. http://msdn.microsoft.com/en-us/library/bb387098.aspx

[22] Russell, Stuart a Norvig, Peter. *Artificial Intelligence A Modern Approach.* Second edition. Upper Saddle River, New Jersey 07458: Pearson Education, Inc., 2003. ISBN 0-13-790395-2

[23] Andrew Ng. *Machine Learning.* https://www.coursera.org/course/ml

[24] Andrew Kirillov. Neural Networks on C#. In: The Code Project. http://www.codeproject.com/Articles/16447/Neural-Networks-on-C

[25] Daniel Jurafsky a James H. Martin. *Speech and Language Processing: An Introduction to Natural Language Processing, Computational Linguistics, and Speech Recognition.* Second Edition. Upper Saddle River, New Jersey 07458: Pearson Education, Inc., 2009. ISBN-13 978-0-13-504-196-3. ISBN-10 0-13-504196-1.

[26] Ralph Grishman. *Computational Lingustics: An Introduction.* First Edition (Reprint). Cambridge University Press, 1986. ISBN 0-521-31038-5

[27] Jeffrey Richter. *CLR via C#.* Third Edition. Microsoft Press, 2010. ISBN 978-0-7356-2704-8

4.3 Seznam tabulek

4.4 Seznam použitých zkratek

SVO – Subject-Verb-Object (podmět-přísudek-předmět)

FGD – funkční generativní popis (Functional Generative Description)

PDT – Pražský závislostní korpus (Prague Dependency Treebank)

PP - příznak podřízenosti

AI – Artificial Intelligence

tp – true positive

fp – false positive

tn – true negative

fn – false negative

p – precision

r – recall

N/A – not available / applicable; není k dispozici či nelze uplatnit v daném kontextu

4.5 Přílohy

Přílohy práce jsou rozděleny do pěti adresářů dle typu.

4.5.1 Binary

Obsahuje zkompilovanou aplikaci a skripty použité při analýze dat prezentované v části 3. Pro použití skriptů je třeba v nich upravit cesty k souborům.

4.5.2 Data

Obsahuje zlatá data z PDT ve formátu popsaném v části 1.5.

4.5.3 Ref

Obsahuje články odkazované v části 1.4.

4.5.4 Results

Obsahuje výsledky prezentované v části 3 rozdělené do adresářů podle použité metody.

4.5.5 Source

Obsahuje zdrojový kód aplikace.