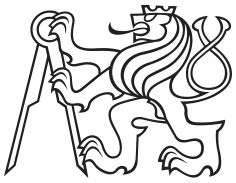
Bakalářská práce



České
vysoké
učení technické
v Praze

F3

Fakulta elektrotechnická
Katedra kybernetiky

Automatické vyhodnocování matematických slovních úloh

Jan Kadlec

Vedoucí: doc. RNDr. Daniel Průša, Ph.D.
Studijní program: Otevřená informatika
Obor: Informatika a počítačové vědy
Květen 2020

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

I. OSOBNÍ A STUDIJNÍ ÚDAJE

Příjmení: **Kadlec** Jméno: **Jan** Osobní číslo: **474727**

Fakulta/ústav: **Fakulta elektrotechnická**

Zadávající katedra/ústav: **Katedra kybernetiky**

Studijní program: **Otevřená informatika**

Studijní obor: **Informatika a počítačové vědy**

II. ÚDAJE K BAKALÁŘSKÉ PRÁCI

Název bakalářské práce:

Automatické vyhodnocování matematických slovních úloh

Název bakalářské práce anglicky:

Automatic Evaluation of Mathematical Word Problems

Pokyny pro vypracování:

Cílem práce je návrh a implementace metod pro automatické vyhodnocování jednoduchých matematických slovních úloh (1. až 3. třída ZŠ) zadaných v textové podobě, v českém jazyce.
1. Seznamte se s existujícími metodami pro řešení daného problému v případě jiných jazyků (angličtina).
2. Seznamte se s dostupnými lingvistickými nástroji pro větný rozbor českého jazyka, které mohou být pro řešení úlohy nápomocné.
3. Vytvořte trénovací a testovací množinu v rozsahu stovek slovních úloh.
4. Navrhněte vlastní metody pro automatické řešení slovních úloh. Metody založte na větném rozboru, vhodných heuristikách a klasifikátorech (strojové učení).
5. Navržené metody implementujte (Java, C/C++, Python - dle vlastního uvážení).
6. Proveďte evaluaci na testovací množině, analyzujte příčiny chyb.
7. Postup, výsledky a poznatky ze všech výše uvedených bodů zdokumentujte.

Seznam doporučené literatury:

[1] M. Hejný, J. Novotná, N. Vondrová: Dvacet pět kapitol z didaktiky matematiky, Pedagogická fakulta, UK, Praha, 2004.
[2] D. Zhang, L. Wang, L. Zhang, B. T. Dai, H. T. Shen: The Gap of Semantic Parsing: A Survey on Automatic Math Word Problem Solvers, CoRR, arXiv:1808.07290, 2018.
[3] M. Straka a J. Strakova: Tokenizing, POS Tagging, Lemmatizing and Parsing UD 2.0 with UDPipe, In proceedings of the CoNLL 2017, Association for Computational Linguistics, 2017.
[4] Ch. M. Bishop: Pattern Recognition and Machine Learning, Springer Science + Bussiness Media, 2006.

Jméno a pracoviště vedoucí(ho) bakalářské práce:

doc. RNDr. Daniel Průša, Ph.D., Strojové učení FEL

Jméno a pracoviště druhé(ho) vedoucí(ho) nebo konzultanta(ky) bakalářské práce:

Datum zadání bakalářské práce: **19.12.2019** Termín odevzdání bakalářské práce: **22.05.2020**

Platnost zadání bakalářské práce: **30.09.2021**

doc. RNDr. Daniel Průša, Ph.D.	doc. Ing. Tomáš Svoboda, Ph.D.	prof. Mgr. Petr Páta, Ph.D.
podpis vedoucí(ho) práce	podpis vedoucí(ho) ústavu/katedry	podpis děkana(ky)

III. PŘEVZETÍ ZADÁNÍ

Student bere na vědomí, že je povinen vypracovat bakalářskou práci samostatně, bez cizí pomoci, s výjimkou poskytnutých konzultací.
Seznam použité literatury, jiných pramenů a jmen konzultantů je třeba uvést v bakalářské práci.

. _____
Datum převzetí zadání

_____
Podpis studenta

Poděkování

Děkuji doc. RNDr. Danielovi Průšovi, Ph.D. za odborné vedení práce.

Prohlášení

Prohlašuji, že jsem předloženou práci vypracoval samostatně a že jsem uvedl veškeré použité informační zdroje v souladu s Metodickým pokynem o dodržování etických principů při přípravě vysokoškolských závěrečných prací.

V Praze dne .....

.....
Jan Kadlec

Abstrakt

Tento dokument shrnuje práci na téma automatické vyhodnocování jednoduchých slovních úloh. Jedná se o problém, jenž spadá do kategorie počítačového zpracování přirozeného jazyka, které je stále aktuálním tématem. Ve své práci se konkrétně věnuji automatickému vyhodnocování slovních úloh v českém jazyce. V textu popisuji implementaci dvou řešení. Prvním řešením je vyhodnocování podle syntaktické analýzy, slovníků a heuristik. Druhé řešení je primárně založeno na strojovém učení a kromě toho jiným způsobem využívá syntaktickou analýzu, slovníky a heuristiky.

Klíčová slova: automatické rozpoznávání, slovní úlohy, český jazyk, větný rozbor, strojové učení

Vedoucí: doc. RNDr. Daniel Průša, Ph.D.
Karlovo náměstí 13,
Praha 2

Abstract

This document summarizes work about automatic evaluation of simple word problems. This is a problem that belongs to the category of computer processing of a natural language, which is still a topical issue. In this work I focus on automatic evaluation of word problems in Czech language. Implementations of two solutions are presented. The first solution is evaluation by syntax analysis, dictionaries and heuristic. The second solution is primarily based on machine learning and besides that it uses syntax analysis, dictionaries and heuristic in other way.

Keywords: automatic evaluation, word problems, Czech language, sentence analysis, machine learning

Title translation: Automatic Evaluation of Mathematical Word Problems

Obsah

Tabulky

Část I

Seznámení s problematikou

Kapitola 1

Úvod

1.1 Cíl práce

Se slovními úlohami se setkal každý žák v hodinách matematiky. Můžeme si položit otázku, jakou mají roli v matematice. Odpovědí je, že staví žáka do pozice, kdy na základě teorie řeší praktické problémy. Dalo by se také říct, že žáka motivují a jsou prostředkem k dosažení zážitnější formy výuky. Příkladem může být úloha na výpočet plochy zahrady. Žáci se ve škole učí, jak vypočítat velikost plochy pomocí vzorečku. Bohužel k tomu, aby tuto znalost využili v praxi moc příležitostí nemají. Proto jsou tu slovní úlohy, které je postaví před praktický problém, kde svou znalost využijí.

Pokud si žák neví rady s řešením slovní úlohy, může pro něj být tento praktický způsob matematiky na druhou stranu demotivující, a tím pádem může zájem o matematiku naopak opadat. Proto je důležité zkoumat slovní úlohy a pomoci tak žákům odhalit postup, kterým by se slovní úloha dala řešit. V současné době již dokonce existují některé aplikace, které řeší slovní úlohy i s postupem. Příkladem je aplikace na matematickém webu Wolfram Alpha[1]. K této aplikaci se podrobněji dostaneme v kapitole 2.

Cílem této práce je pokus o nalezení obecného algoritmu, který by řešil slovní úlohy pro první tři třídy prvního stupně základních škol. Konkrétně se bude jednat o dva algoritmy implementované dvěma různými způsoby.

[1] wolframalpha.com

V rámci této práce vznikly dva datasety slovních úloh v českém jazyce. Ve slovních úlohách se mohou objevit základní matematické operace (+,−,∗,/) v různých kombinacích. Na základě tohoto algoritmu by mohl vzniknout výukový nástroj, který by žákům pomáhal při výuce matematiky. Jakmile by žák měl se slovní úlohou potíže, nástroj by mu mohl poskytnout nápovědu.

1.2 Didaktický pohled

Nejprve jsem se při svém projektu rozhodl prozkoumat to, jak na danou problematiku pohlíží ti, kterých se tato problematika týká, tedy dětí. Velkým zdrojem informací pro mě byla práce [MH04], konkrétně kapitola 22 - Zpracování informací při řešení slovních úloh. Pro vyřešení slovní úlohy je potřeba nejprve znát teorii - nějaký model, který následně po extrahování potřebných informací ze slovní úlohy použijeme k získaní výsledku.

Uvažujme slovní úlohu:
Pavel má 2 jablka. Petr má 3 jablka. Kolik jablek mají dohromady?

Jako teoretický model nám v této úloze poslouží sčítání dvou čísel. Slovní úloha by se nám nezměnila, kdyby místo jablek bylo v zadání cokoliv jiného. Slovní úloha by se také nezměnila, kdyby místo Pavla a Petra byla jiná jména.

Pro řešitele slovních úloh je v první řadě klíčové extrahovat správné informace (vztahy, vazby, čísla). Řešitel musí pochopit, o jaký problém se ve slovní úloze jedná a následně musí pochopit, na co se slovní úloha ptá. Podoba extrahovaných informací není pevně dána. Tvorba i uchování informací zůstává na řešiteli. Extrahování informací je klíčovým procesem v řešení slovních úloh. Při chybném extrahování nemusí existovat model, který by slovní úlohu řešil.

Po správném extrahování důležitých informací řešitel hledá model, který má být aplikován ke správnému vyřešení úlohy. Cílem modelu je správně řešit slovní úlohu. Nalezení modelu nemusí být jednoznačné. Model může být těžké nalézt, anebo je zapotřebí zkombinovat více modelů. Slovní úloha může obsahovat podúlohy, jejichž výsledky jsou potřeba ke správnému řešení. Tím se nám řešení slovních úloh stává složitější. Model je závislý na podobě extrahovaných informací. To znamená, že pro slovní úlohu nemusí existovat pouze jedno řešení a záleží na tom, jak data extrahujeme a následně jaký model použijeme.

Neexistuje žádný obecný algoritmus, jak extrahovat data a nalézt správný model. Jak se tedy děti učí řešit slovní úlohy? Existuje několik metod, kterými pedagogové mohou pomoci žákům. Jednou z metod je například zdramatizování dané slovní úlohy. To znamená, že děti se do slovní úlohy „vžijí“ a tím lépe pochopí, které informace jsou důležité, a které ne. Tato metoda se využívá u nejmenších žáků. Další metodou je například hledání potřebných informací a modelu s pomocí pedagoga. Pedagog v danou situaci funguje jako průvodce, který žákům udává směr v řešení slovních úloh.

1.3 Počátky automatického vyhodnocování matematických slovních úloh

Myšlenka automatického vyhodnocování matematických slovních úloh sahá až do roku 1963 a je naznačena v práci [FF63]. Jedná se o shrnutí dosavadních poznatků umělé inteligence známých k tomuto datu a diskuzi o tom, co by v budoucnu mohla umělá inteligence přinést.

V práci můžeme najít kapitolu *Machines That Play Games*, která shrnuje poznatky o teorii her nebo kapitolu *Pattern Recognition*, která se zaměřuje na rozpoznávání. Kapitola *Problem-solving*, jejíž podkapitolou je *GPS*[2]*, A PROGRAM THAT SIMULATES HUMAN THOUGHT*, shrnuje poznatky ohledně obecného řešení problémů. Základy počítačového zpracování přirozeného jazyka můžeme nalézt v kapitolách *Question-answering Machines* a *Verbal Learning and Concept Formation.*

Za zmínku stojí první program řešící matematické slovní úlohy, který nese název STUDENT. Program vznikl roku 1964 jako diplomová práce [Bob64], jejímž autorem je Daniel G. Bobrow. Program je napsaný v programovacím jazyce Lisp.

[2]GPS - General Problem Solver

Kapitola 2

Existující řešení

Mezi průkopníky automatického řešení slovních úloh patří Nate Kushman a spol., kteří se ve své práci [KAZB14] z roku 2014 věnují právě automatickému řešení slovních úloh. Jejich řešení je založeno na sémantické interpretaci, extrakci informací a samotném řešení slovních úloh. Řešení dosahuje téměř 70% úspěšnosti. V rámci zmiňované práce byl vytvořen dataset Alg514, čítající 514 anotovaných slovních úloh.

2.1 Principy

Automatické řešení slovních úloh může fungovat na 3 principech. Slovní úlohy můžeme vyhodnocovat:

- podle předem definovaných vzorů
- pomocí strojového učení (dnes zejména neuronové sítě)
- pomocí syntaktické analýzy, slovníků a heuristik

Je možné tyto principy kombinovat, jak to například dělá i Nate Kushman v práci [KAZB14], kde pomocí definovaných vzorů a strojového učení řeší slovní úlohy. Více informací k principům je možné nalézt v [ZWZ+19], kde je použití postupů popsáno i chronologicky.

2.2 Vyhodnocování podle vzorů

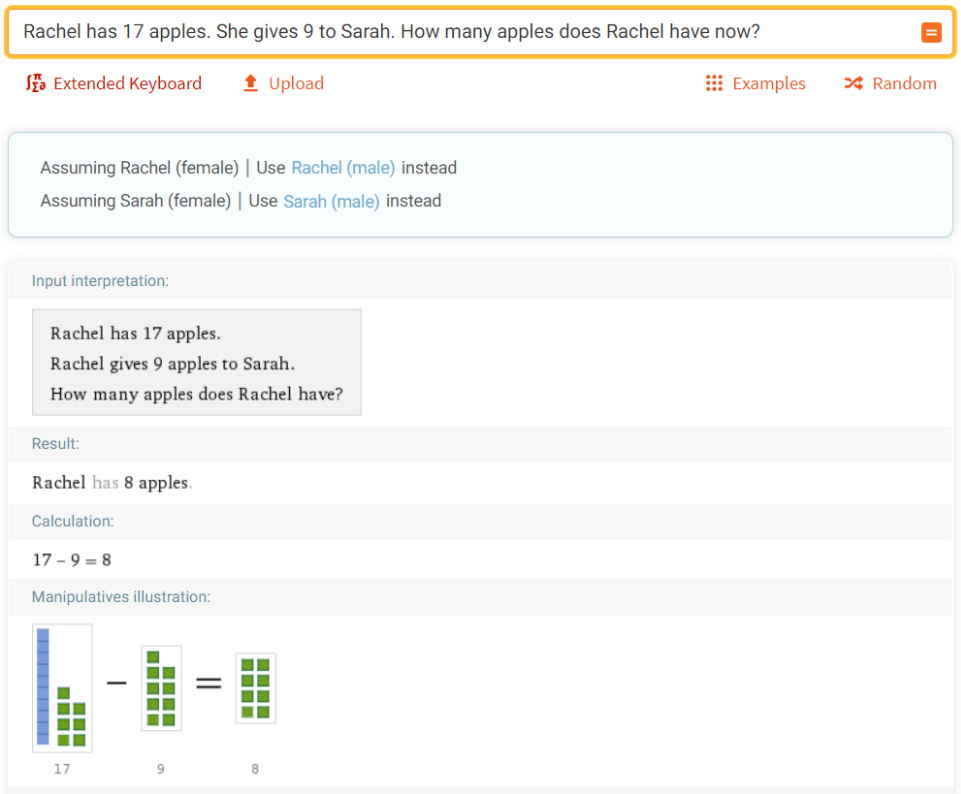
V kapitole 1 jsme měli příklad slovní úlohy, u které jsme si říkali, že model by se nezměnil, kdyby došlo ke změně jmen nebo čísel. Této vlastnosti využívá princip vyhodnocování podle vzorů. Pro slovní úlohu se hledá vzor, který by ji reprezentoval. Celé vyhodnocování si můžeme představit jako slovník, kde klíče jsou regulární výrazy a hodnota klíče jsou matematické operace s čísly ze slovní úlohy.

{NOUN1 má NUM1 NOUN3. NOUN2 má NUM2 NOUN3. Kolik NOUN3 mají dohromady? : NUM1 + NUM2}

NOUN1 a NOUN2 jsou podstatná jména - entity, NUM1 a NUM2 jsou čísla a NOUN3 je podstatné jméno - entita (předmět). Tento slovník o jednom klíči by řešil úlohu z kapitoly 1 pro libovolná jména, předměty i čísla. Výhodou tohoto principu je jednoduchost. Nevýhodou je potřeba velkého množství definovaných vzorů. Bohužel tento princip se nehodí pro slovní úlohy v českém jazyce. Český jazyk má volný slovosled, což znamená, že pořadí větných členů není pevně dáno. Proto, aby se tento princip vypořádal s volným slovosledem, by potřeboval ještě větší množství definovaných vzorů. Na tomto principu funguje zmiňovaná aplikace na webu Wolfram Alpha, o které jsme hovořili v kapitole 1. Pro větší generalizaci můžeme slova, která nejsou převedena do entit, čísel a předmětů, převést do jejich základního tvaru. Výsledný vzor by pak vypadal takto:

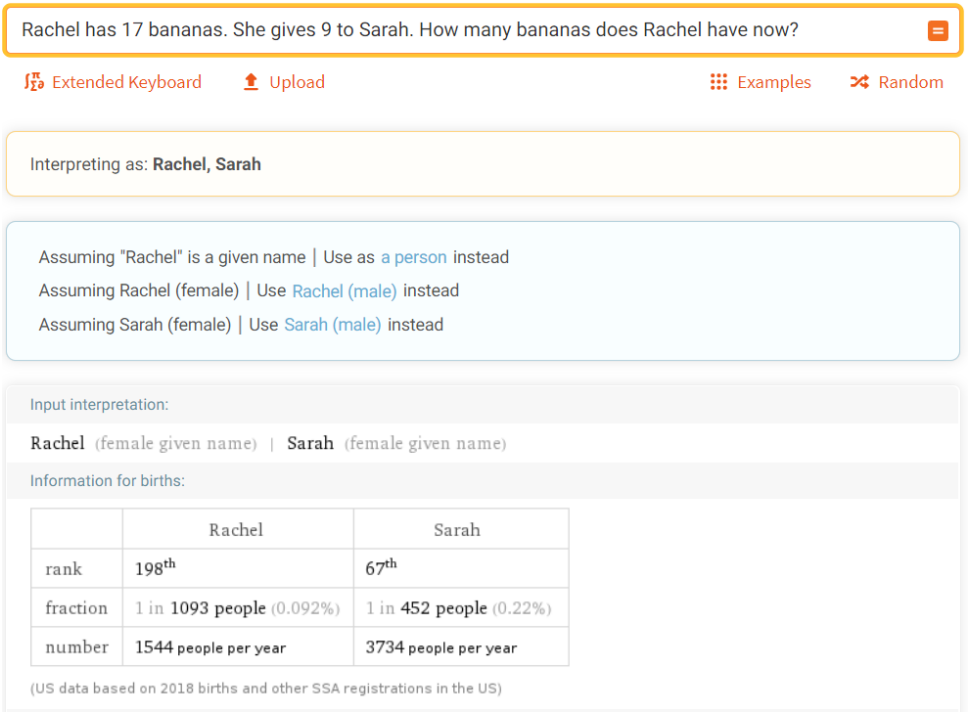
NOUN1 mít NUM1 NOUN2. NOUN3 mít NUM2 NOUN2. Kolik NOUN2 mít dohromady?

V takto převedeném tvaru můžeme eliminovat skloňování slov, které je pro český jazyk typické.



Obrázek 2.1: Příklad úspěšně vyřešené slovní úlohy aplikací WolframAlpha i s postupem.

Na Obrázku 2.1 si můžeme všimnout úspěšně řešené slovní úlohy již zmiňovanou aplikací na webu WolframAlpha. Kromě správného výsledku dostáváme i postup, který by v případě strojového učení nebyl součástí výstupu.



Obrázek 2.2: Příklad neúspěšně vyřešené slovní úlohy aplikací WolframAlpha.

Obrázek 2.2 je příkladem neúspěšně vyřešené slovní úlohy programem WolframAlpha. Neúspěšně řešená slovní úloha se liší od úspěšně řešené slovní úlohy pouze v jednom slově.

2.3 Strojové učení

Na rozdíl od předchozího principu je řešení pomocí strojového učení složitější, ale za to úspěšnější. Rád bych uvedl dvě práce, jejíž cílem bylo řešení automatického vyhodnocování slovních úloh právě za pomoci strojového učení.

Práce [HLLY18] z roku 2018 používá ve svém řešení model seq2seq a nepotřebuje žádnou extrakci informací. U tohoto řešení autoři objevili dvě nevýhody - generování špatných čísel, která nejsou ve slovní úloze a generování čísel na špatných pozicích. Generování špatných čísel, které nejsou ve slovní úloze, autoři popisují jako vlastnost, kdy se v rovnici řešící slovní úlohu vyskytují čísla, která se nenachází v zadání slovní úlohy. Autoři uvádí jednu klíčovou výhodu, kterou jejich řešení má a to, že jejich řešení je schopné tvořit rovnice, které nejsou v trénovací množině.

Práce [KAZB14] již byla zmíněna na začátku kapitoly. Jedná se o řešení, které na rozdíl od předchozí práce pracuje se sémantickou interpretací a extrakcí informací. Jde o zcela odlišný přístup k řešení pomocí strojového učení.

Úspěšnost obou prací můžeme porovnat, protože byly spuštěny nad datasetem Alg514. [HLLY18] dosáhla úspěšnosti 82.5%, zatímco [KAZB14] dosáhla úspěšnosti téměř 70%. Můžeme se domnívat, že lepších výsledků bylo v první práci dosaženo z důvodu časového rozdílu a použitím modelu seq2seq.

Ohledně velikosti a diverzifikovanosti datasetu přináší zajímavé poznatky [HSL+16]. V rámci této práce vznikl nový dataset Dolphin18K, čítající přes 18000 anotovaných slovních úloh, který je větší a více diverzifikovaný než dataset Alg514. V rámci experimentů se došlo k závěru, že všechny možná řešení spuštěná na tomto datasetu dosahují horších výsledků, než na původních, menších a méně diverzifikovaných datasetech (např. Alg514).

Models	Alg514	NumWordT6	NumWordT1	Dolphin18KT6	Dolphin18KT1
Seq2SeqAttn (MLE)	19.4%	19.7%	11.0%	13.0%	10.2%
+Copy (MLE)	41.4%	59.9%	23.0%	20.2%	12.9%
+Copy+Align (MLE)	41.8%	60.4%	26.8%	21.0%	13.1%
+Copy+Align (RL)	44.5%	64.0%	29.2%	23.3%	15.9%
Huang et al. (2017)	81.6%	42.0%	20.8%	30.6%	28.4%
+ CASS^{RL} (hybrid)	**82.5%**	**65.8%**	**29.7%**	**33.2%**	**29.0%**

Tabulka 2.1: Úspěšnost vyhodnocení podle [HLLY18].

Můžeme si všimnout, že jak je zmíněno v práci [HSL+16], tak dosahovaná úspěšnost na větších a více diverzifikovaných datasetech (Dolphin18KT6, Dolphin18KT1) je zpravidla menší, než na menších datasetech.

2.4 Syntaktická analýza, slovníky, heuristiky

Principem tohoto řešení je analýza množiny slovních úloh po syntaktické stránce. To znamená, že se zkoumají vztahy mezi slovními druhy, větnými členy a hledají se pravidla, která by za pomoci slovníků a heuristik řešila úlohy z trénovací množiny. Vyhodnocování pomocí syntaktické analýzy, slovníků a heuristik je možné použít na menší trénovací množinu. Nevýhodou tohoto principu je závislost na spolehlivosti lingvistických nástrojů a analyzátorů. Syntaktická analýza se v některých případech objevuje i ve strojovém učení jako předzpracování dat.

Příkladem použití tohoto principu může být [HHEK14] z roku 2014. Řešením je model ARIS, který nejprve analyzuje věty slovní úlohy. Následně extrahuje relativní proměnné a čísla. Tyto informace namapuje do rovnice. ARIS využívá také strojového učení, a to konkrétně algoritmu SVM, který kategorizuje slovesa ve spojení s kontejnery (slovo, související se slovesem) do 7 kategorií (pozorování, pozitivní, negativní, pozitivní přenos, negativní přenos, vytvoření, zánik).

Kapitola 3

Lingvistické nástroje, analyzátory

3.1 UDPipe

V jedné části navrženého řešení byl použit nástroj UDPipe [SS17], který byl doporučený odborníky na lingvistiku, konkrétně z Matematicko-fyzikální fakulty Univerzity Karlovy v Praze. Předností tohoto nástroje je univerzálnost - možnost použít k syntaktické analýze vět 90-ti jazyků. Nástroj zpracovává větu a vrací ji syntakticky analyzovanou ve formátu CONLL-U [1]. Příklad věty ve formátu CONLL-U je možné vidět na Obrázku 3.1 a syntaktický rozbor v podobě stromu na Obrázku 3.2.

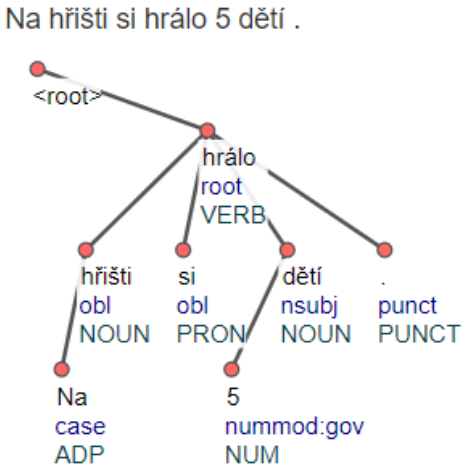
```
# newdoc
# newpar
# sent_id = 1
# text = Na hřišti si hrálo 5 dětí.
1   Na      na      ADP     2   case      _   _
2   hřišti  hřiště  NOUN    4   obl _     _
3   si      se      PRON    4   obl _     _
4   hrálo   hrát    VERB    0   root      _   _
5   5       5       NUM     6   nummod:gov  _   _
6   dětí    dítě    NOUN    4   nsubj     _   SpaceAfter=No
7   .       .       PUNCT   4   punct     _   SpaceAfter=No
```

Obrázek 3.1: Příklad výstupu nástroje UDPipe - CONLL-U formát

UDPipe funguje na principu neuronové sítě. V tomto případě musíme

[1] https://universaldependencies.org/format.html

bohužel počítat s chybovostí, protože věty jsou zpracovány na základě natrénovaného modelu, který i přes velikost použité trénovací množiny (1.5 milionů tokenů, 1.5 milionů slov a 88 tisíc vět) dělá chyby.



Obrázek 3.2: Syntaktický rozbor v podobě stromu.

3.2 Natural Language Toolkit

Za zmínku také stojí Natural Language Toolkit - NLTK [2], jedná se o sadu knihoven pro Python pro zpracování přirozeného jazyka. NLTK obsahuje spoustu aplikaci, mezi které patří například lemmatizace, tokenizace, určení slovního druhu a další. Bohužel, český jazyk nepatří mezi podporované jazyky, proto tento nástroj nemůže být použít v této práci.

3.3 Český národní korpus

Český národní korpus [3] je akademický projekt Filozofické fakulty Univerzity Karlovy, jehož historie sahá až do roku 1994. Součástí toho projektu je spousta aplikací pro zkoumání českého jazyka. V rámci tohoto projektu vzniklo hned několik obsáhlých korpusů, z nichž největší - SYN (verze 8) čítá 4.5 mld. slov. Korpusy obsahují anotované texty z beletrie, oborové literatury a publicistiky.

[2] https://www.nltk.org/
[3] www.korpus.cz

Ke každému slovu nalezneme jeho lemma (základní tvar slova) a morfologické značky (informace o slovním druhu, pádu, číslu, čase,...).

Kapitola 4

Sesbírané slovní úlohy

V rámci této práce vznikly dva datasety - *WP150*, *WP500*. Slovní úlohy obsažené v těchto datasetech jsou z učebnice [Rei96]. Dataset *WP150* obsahuje 150 slovních úloh. Dataset *WP500* obsahuje 500 slovních úloh. Dataset *WP150* je podmnožinou trénovací množiny datasetu *WP500*.
Příklady sesbíraných úloh:

- *Jeden meloun váží 6 kg. Kolik váží 7 stejných melounů?*
- *Petr přečetl 9 knih. Eva přečetla 6 krát více knih. Kolik knih Eva přečetla?*
- *Závodu se zúčastnilo 63 cyklistů ze Slovenska a 7 krát méně cyklistů z Rakouska. Kolik cyklistů závodilo?*
- *Maminka koupila 20 tvarohových koláčků a 15 makových koláčků. Děti 8 koláčů snědly. Kolik koláčů zůstalo?*
- *Osobní vlak měl 17 vagonů. Nákladní vlak měl o 23 vagonů více. Kolik vagonů měl nákladní vlak?*

Tyto úlohy byly ručně přepisovány, nebo byl použit software OCR. Slovní úlohy jsou uložené v textovém souboru. Na každém řádku je jedna slovní úloha, za ní výsledek a výraz, který je řešením. Hodnoty jsou odděleny „|“.

```
Jeden meloun váží 6 kg. Kolik váží 7 stejných melounů? | 42 | NUM1 * NUM2
```

Datasety jsou uložené v adresáři *./dataset/WP150* a *./dataset/WP500*. V adresářích jsou dva textové soubory *traindata.txt* a *testdata.txt*.

Část II

Syntaktická analýza, slovníky a heuristiky

Kapitola 5

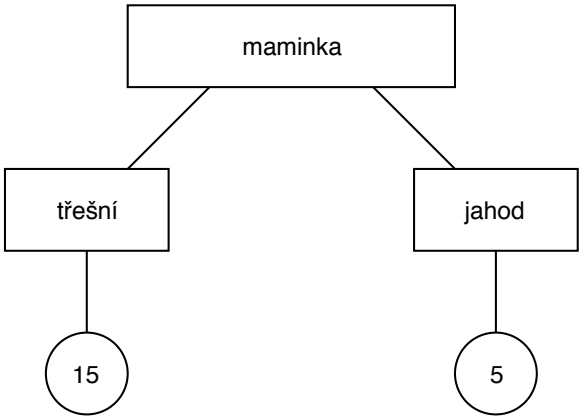
Zvolený postup

Ze zvolených postupů, které jsou uvedeny v kapitole 2, jsem se rozhodl pro řešení pomocí syntaktické analýzy, slovníků, heuristik a strojového učení. V této části bych se rád zaměřil na řešení, které k nalezení výsledku používá syntaktickou analýzu, slovníky a heuristiky. Motivací pro výběr tohoto řešení je jednoduchost implementace oproti strojovému učení a lepší úspěšnost oproti vyhodnocování podle vzorů.

5.1 Uchování dat

Po sběru prvních 20-ti úloh jsem začal s jejich analýzou. Analyzoval jsem, která data jsou užitečná a jak je uchovat. Na základě své analýzy jsem došel k závěru, že nejlepší způsob, jak extrahovaná data uchovávat, bude tvorbou struktur. Následující příklad ukazuje slovní úlohy a možnou podobu uchování dat.

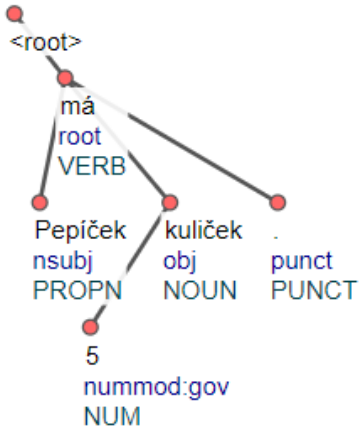
Příklad:
Maminka koupila na trhu 15 kg třešní a 5 kg jahod.



Obrázek 5.1: Příklad struktury z věty.

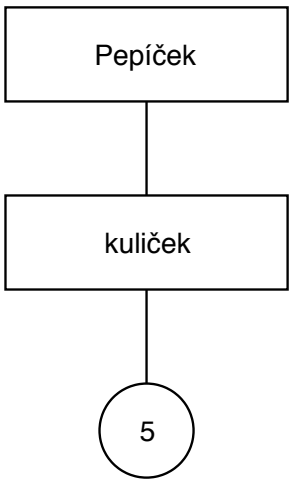
5.2 Tvorba struktur

Dalším cílem analýzy bylo zjistit, jak tato data extrahovat. Moji první myšlenkou bylo použít slovní druhy a větné členy, které nám vrátí program UDPipe ve formátu CONNL-U. Bohužel toto extrahování vedlo do slepé uličky. Syntaktická analýza pomocí programu UDPipe není tak spolehlivá, aby tento způsob extrahování mohl dobře fungovat. Vylepšování extrakce dat v trénovací množině vedlo k přefitování, tím pádem se úspěšnost v testovací množině zvyšovala minimálně. Další možností bylo využít syntaktický rozbor v podobě stromu. Tento způsob se osvědčil lépe než ten předchozí. Hlavní myšlenka využití stromu spočívá v tom, že pokud je věta smysluplně napsaná a program UDPipe ji spolehlivě ohodnotí, tak pohybem ve stromu od uzlu, který obsahuje důležité číslo, můžeme jednoduše vytvořit strukturu znázorněnou na Obrázku 5.1. Na následujících příkladech vysvětlím myšlenku extrakce dat ze stromové struktury.



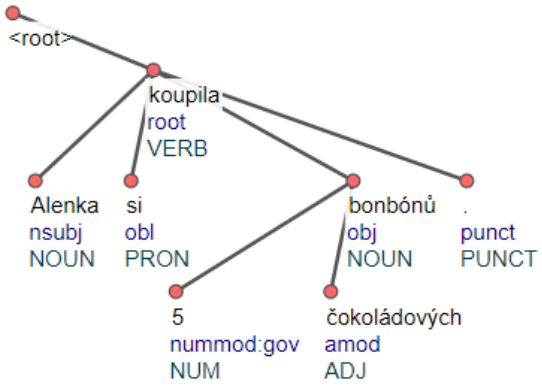
Obrázek 5.2: Syntaktický rozbor slovní úlohy v podobě stromu.

Na Obrázku 5.2 vidíme číslo „5“ v listu. Jeho rodič je slovo „kuliček“. Tuto vlastnost převezmou slova i do naší struktury. To, jestli je číslo „5“ kladné nebo záporné, určíme pomocí slovníku, který jsme získali analýzou. V takové struktuře nám chybí informace KDO má kuliček „5“. Tady využijeme toho, že nám program UDPipe dává informaci i o větných členech. Proto se ve větě pokusíme najít podmět, ten je „Pepíček“. Výsledná struktura je znázorněná na Obrázku 5.3:



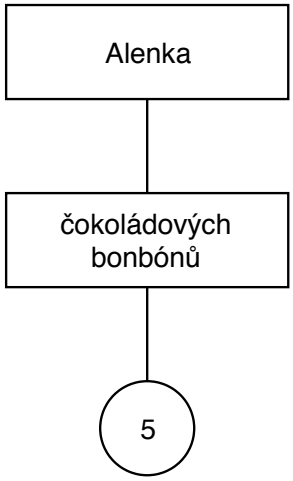
Obrázek 5.3: Diagram extrakce z Obrázku 5.2

Uvedeme si příklad ještě na jedné větě.



Obrázek 5.4: Syntaktický rozbor slovní úlohy v podobě stromu

Věta na Obrázku 5.4 se liší v tom, že obsahuje přívlastek „čokoládových“. Tím pádem musí mít struktura trochu jinou podobu. Rodič čísla 5 je slovo „bonbónů“. To nám v naší struktuře nestačí. Proto číslo 5 v naší struktuře bude mít za rodiče sousloví „čokoládových bonbónů“. Opět doplníme KDO má čokoládových bonbónů „5“.



Obrázek 5.5: Příklad struktury po extrakci dat.

Jak může být vidět na Obrázku 5.2 a Obrázku 5.4, obě věty jsou tvořeny pouze jednou větou hlavní a obsahují čísla. Při analýze jsem zjistil, že pro slovní úlohy z trénovací množiny není potřeba žádná analýza vět, které neobsahují čísla. Pro extrakci dat z vět, které neobsahují podmět, nebo se skládají z více vět, použijeme rozklad nebo substituci. Substitucí je myšleno,

že pokud věta neobsahuje podmět a chceme z té věty vytvořit strukturu, tak si převezmeme podmět z nejbližší předchozí věty, která podmět obsahuje. Rozklad věty znamená, že pokud věta obsahuje spojku „a“, tak můžeme větu rozdělit na dvě věty, aniž bychom narušili význam věty.

V některých slovních úlohách se extrakce dat dělat nemusí. Výsledná rovnice je patrná z počtu čísel a slov v otázce. Takovou slovní úlohu jsme již měli v podkapitole 1.2.

Slovní úloha:
Pavel má 2 jablka. Petr má 3 jablka. Kolik jablek mají dohromady?

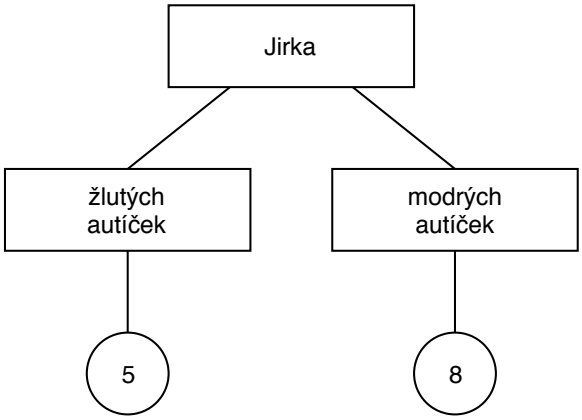
Konkrétně u této slovní úlohy si můžeme všimnout, že jsou v ní dvě čísla a otázka obsahuje slovo „dohromady“. Z toho můžeme usoudit, že existují nějaká slova, která nám mohou říct, jakou matematickou operaci provést, aniž bychom extrahovali data. Taková slova ukládám do slovníků a u každé věty budu nejprve nahlížet, jestli nějaké slovo ze slovníku není v otázce. Pokud žádné slovo v otázce není, pokračuji extrakcí dat. Pokud takové slovo v otázce je, zkontroluji podmínku počtu čísel a podle toho buď aplikuji matematickou operaci, nebo extrahuji data.

5.3 Zpracování struktury

V této podkapitole budu vysvětlovat, jakým způsobem zpracovávám vytvořenou strukturu popsanou výše. Zpracování uvedu na konkrétním příkladu.

Máme slovní úlohu:
Jirka má 5 žlutých autíček. Jirka má 8 modrých autíček. Kolik autíček má Jirka?

A máme vytvořenu strukturu:



Obrázek 5.6: Extrahovaná struktura.

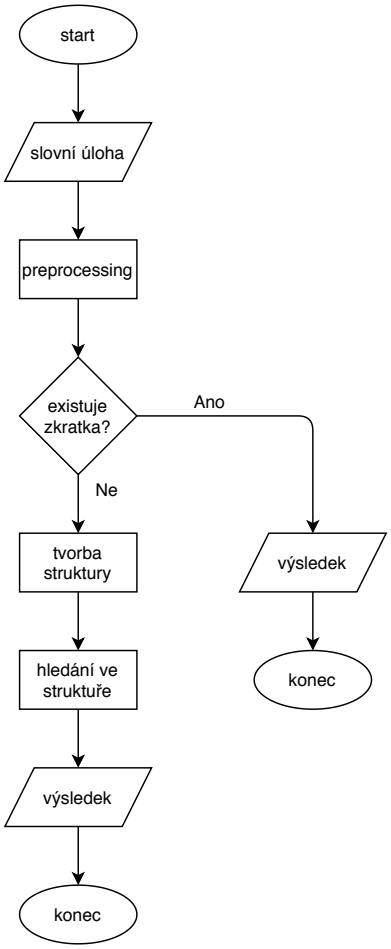
Algoritmus funguje následovně:
Pozn.: Každou strukturu si můžeme představit jako strom nebo více stromů.

- začínáme v hloubce 0 stromu/stromů
- pro každé slovo v otázce (může se jednat pouze o vybrané slovní druhy z otázky) se podíváme, jestli se vyskytuje v aktuální úrovni stromu/stromů
  - pokud se slovo vyskytuje, inkrementujeme hloubku a pokračujeme v prohledávání v uzlech, kde jsme slovo našli
  - pokud se slovo nevyskytuje v žádném uzlu, tak inkrementujeme hloubku a pokračujeme prohledávání ve všech uzlech, které jsou v patřičné hloubce
- pokud jsem v listu, vrátím číslo, nebo součet

Ukázka algoritmu:
Prvek v hloubce 0: Jirka
Kolik autíček má **Jirka** ?
Inkrementujeme hloubku (hloubka = 1), prohledáváme potomky prvku „Jirka“ v hloubce 1.
Prvky v hloubce 1: žlutých autíček, modrých autíček
Kolik autíček má Jirka ?
V otázce je slovo „autíček“. Naše prvky sice obsahují toto slovo, ale před ním mají ještě přídavné jméno. Proto bereme, že jsem ani jeden z prvků v hloubce 1 nenalezli v otázce. Inkrementujeme hloubku (hloubka = 2) a posouváme se do potomků prvků „žlutých autíček“ a „modrých autíček“
Prvky v hloubce 2 jsou listy obsahující čísla 5 a 8. Sečteme je.
Výsledek je 13.

5.4 Algoritmus

V této podkapitole se budu věnovat samotnému algoritmu, který řeší slovní úlohu. Hlavní části algoritmu jsou zobrazeny na Obrázku 5.7.



Obrázek 5.7: Diagram znázorňující algoritmus.

Vysvětlení komponent algoritmu.

- vstupem je slovní úloha
- preprocessing
  - V rámci preprocessingu dojde k rozdělení vět, které obsahují sloveso

a spojku „a“. Dojde k nahrazení sekvencí typu „5 krát méně“, „o 7 více“, apod. za konkrétní číselnou hodnotu. Proběhne několikrát syntaktická analýza programem UDPipe.

- existuje zkratka?
  - Tato komponenta se využívá pro slovní úlohy, které obsahují pouze dvě čísla. Algoritmus hledá specifická slova ze slovníku (viz příloha C), která by určila, jestli se jedná o součet nebo rozdíl. Slovník byl vytvořený manuálně po ruční analýze úloh ve trénovací množině.
- tvorba struktury - popsána v podkapitole 5.2
- hledání ve struktuře - popsáno v podkapitole 5.3
- výsledek - řešení úlohy reprezentované celým nezáporným číslem, pokud algoritmus nenalezne žádné řešení, je jeho výstupem číslo -1

Příklad použití algoritmu (varianta s použitím zkratky):
vstup: *Pohádková kniha má 96 stran. Klárka již přečetla 35 stran. Kolik stran jí ještě zbývá přečíst?*
preprocessing: Syntaktická analýza.
existuje zkratka?: Slovní úloha obsahuje 2 čísla. V otázce je slovo zbýt. Zkratka existuje.
výsledek: 61

Příklad použití algoritmu (varianta bez použití zkratky):
vstup: *Petr přečetl 9 knih. Eva přečetla 6 krát více knih. Kolik knih Eva přečetla?*
preprocessing: *Petr přečetl 9 knih. Eva přečetla 54 knih. Kolik knih Eva přečetla?* Syntaktická analýza.
existuje zkratka?: Slovní úloha obsahuje 2 čísla, sloveso přečíst ale není ve slovníku. Zkratka neexistuje.
tvorba struktury: 'Petr': 'kniha': '9', 'Eva': 'kniha': '54'
hledání ve struktuře: Celý proces hledání ve struktuře je popsán v podkapitole 5.3. Zjednodušené znázornění hledání: Eva → kniha → 54
výsledek: 54

Kapitola 6

Evaluace

Algoritmus SA řešící slovní úlohy, pomocí syntaktické analýzy, slovníků a heuristik, je popsán v podkapitole 5.4. Zde uvádíme výsledky analýzy jeho přesnosti. Předzpracování a samotný algoritmus trvá několik desítek vteřin.

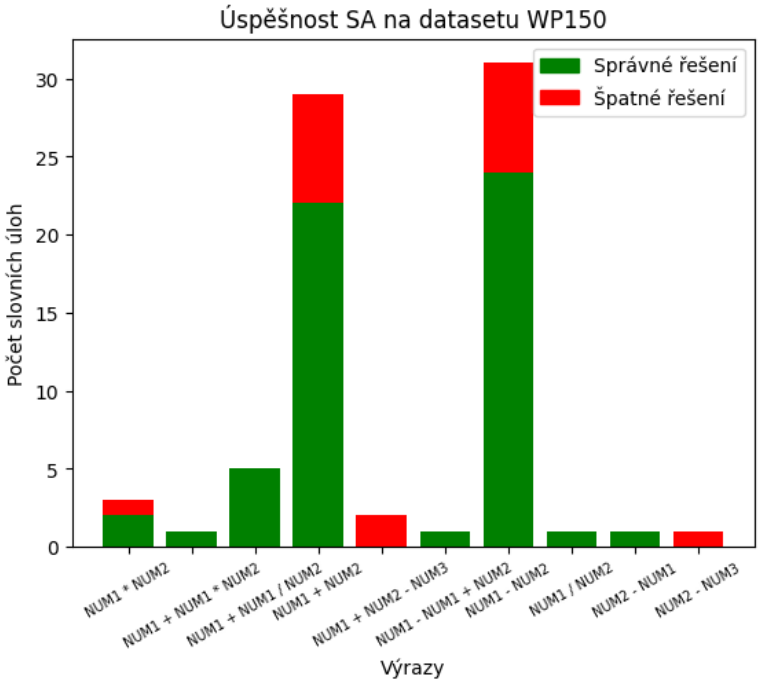
	čas [s]
předzpracování	19.6
algoritmus	2.9

Tabulka 6.1: Údaje o rychlosti řešení. Bylo měřeno předzpracování slovních úloh z testovací množiny datasetu WP500 a následně byla zvlášť měřena rychlost algoritmu na všech datech ze zmiňované množiny. Měření proběhlo na počítači s *Windows Subsystem for Linux* a procesorem *Intel(R) Core(TM) i7-8750H CPU*.

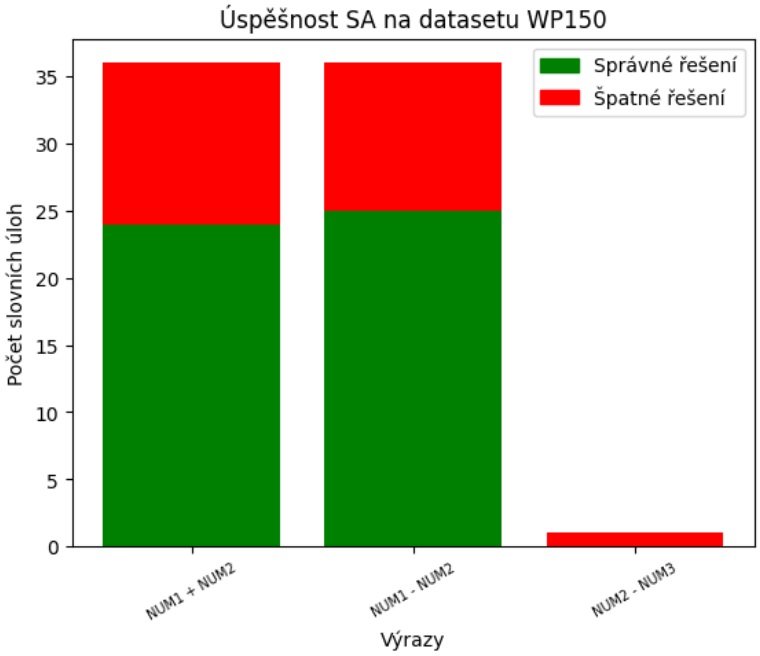
Řešení dosáhlo následujících výsledků:

	WP150 dataset	**WP500 dataset**
trénovací množina	76%	53.2%
testovací množina	67.12%	28.87%

Tabulka 6.2: Úspěšnost na datasetech pro testovací a trénovací množinu.

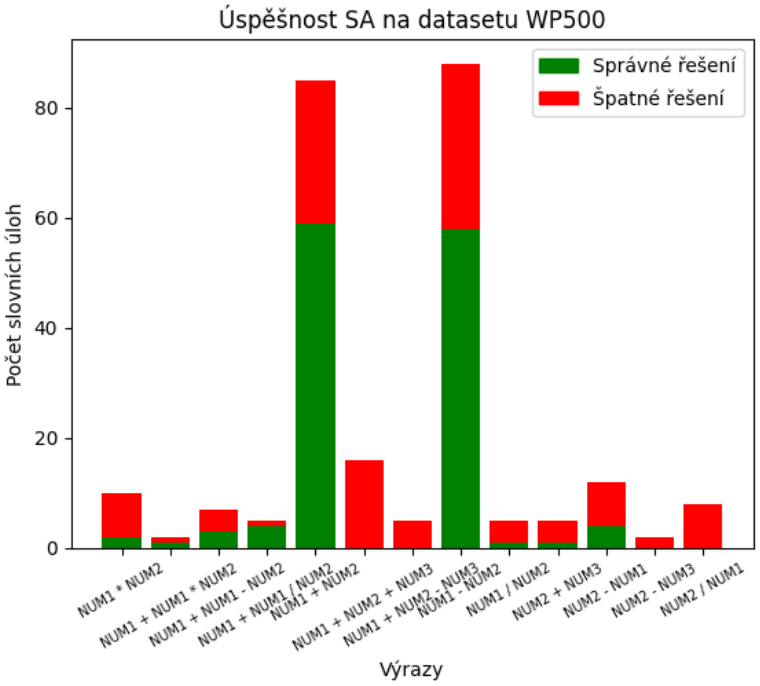


(a): Vyhodnocení na trénovací množině datasetu WP500.

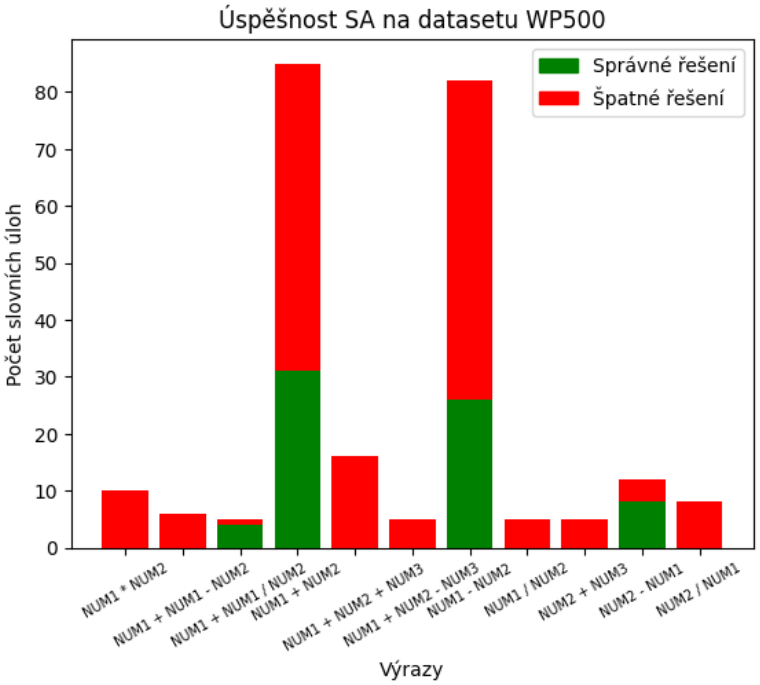


(b): Vyhodnocení na testovací množině datasetu WP150.

Obrázek 6.1: Úspěšnost na trénovací a testovací množině datasetu WP150



(a): Vyhodnocení na trénovací množině datasetu WP500.



(b): Vyhodnocení na testovací množině datasetu WP500.

Obrázek 6.2: Úspěšnost na trénovací a testovací množině datasetu WP500

Po evaluaci jsem se rozhodl zjistit příčinu neúspěchu. Příčiny špatného výsledku nebo neohodnocení slovní úlohy jsem rozdělil do následujících tří skupin:

I. chybné ohodnocení pomocí programu UDPipe

II. chybný preprocessing

III. chybně vytvořená struktura

IV. chybné zpracování struktury

V tabulce je možné vidět rozložení chyb podle typu chyby a množiny, ve které chyba nastala. V následujících podkapitolách budu rozebírat konkrétní chyby na datasetu WP150. Vždy je budu rozebírat v rámci trénovací a testovací množiny. Uvedené příklady chyb jsou z obou množin.

Množina	**I.**	**II.**	**III.**	**IV.**	**celkem**
trénovací	7 / 38.89%	3 / 16.67%	5 / 27.78%	3 / 16.67%	18
testovací	9 / 37.5%	5 / 20.83%	9 / 37.5%	1 / 4.17%	24
obě	16 / 38.1%	8 / 19.05%	14 / 33.33%	4 / 9.52%	42

Tabulka 6.3: Procentuální zastoupení typů chyb v trénovací a testovací množině pro datatest WP150.

Rád bych zmínil úspěšnost komponenty *existuje zkratka?*, kterou můžeme pozorovat v následnicích třech tabulkách. V tabulce 6.5a můžeme vidět úspěšnost s použitím komponenty, v tabulce 6.5b úspěšnost bez použití komponenty. Tabulka 6.6 nám dává informaci o tom, kolikrát byla komponenta použita, a v závorce je údaj, který říká, kolikrát použití zkratky vedlo k chybnému výsledku.

(a) : Komponenta je použita.

	WP150	WP500
train	76%	53.2%
test	67.12%	28.87%

(b) : Komponenta není použita.

	WP150	WP500
train	44%	30.8%
test	32.87%	19.24%

Tabulka 6.4: Úspěšnost řešení s použítím komponenty *existuje zkratka?* a bez použití.

Z tabulek úspěšnosti můžeme vidět, že použití komponenty ve všech případech zvyšuje úspěšnost.

	WP150	WP500
train	32 (0)	82 (2)
test	35 (0)	31 (1)

Tabulka 6.6: Informace o tom, kolikrát byla komponenta *existuje zkratka?* v řešení použita. Údaj v závorce udává, kolikrát použití komponenty vedlo k chybnému výsledku.

Analýza chyb proběhla na 10 vzorcích z trénovacího datasetu WP500. Výsledkem bylo, že v 30% došlo ke špatnému vytvoření struktury, ve 20% ke špatnému preprocessingu, ve 20% ke špatnému zpracování struktury, ve 20% byla chyba v ohodnocení UDPipe a 10% byla chyba v komponentě *existuje zkratka?*. Některé chyby lze opravit, v případě, že by chyba souvisela s ohodnocením programem UDPipe, pak by oprava nemusela být možná.

6.1 Chybné ohodnocení UDPipe

Chyb vzniklých špatným ohodnocením UDPipe bylo nejvíce, a to 38.1%.
Příklad slovní úlohy:
Rodiče koupili Toníkovi do školy 8 sešitů a Blance jen 5 sešitů. O kolik sešitů má Blanka méně?
První větu této slovní úlohy byla v předzpracování rozdělena.
Vznikly tyto 2 věty:
Rodiče koupili Toníkovi do školy 8 sešitů.
Rodiče koupili Blance jen 5 sešitů.
Víme, že by slovo rodiče mělo být v obou větách podmět.
Program UDPipe mi tyto dvě věty ohodnotil následovně:

```
# sent_id = 1
# text = Rodiče koupili Toníkovi do školy 8 sešitů.
1   Rodiče  rodič   NOUN    2   obj _   _
2   koupili koupit  VERB    0   root    _   _
3   Toníkovi    Toník   PROPN   2   nsubj   _   _
4   do  do  ADP 5   case    _   _
5   školy   škola   NOUN    2   obl _   _
6   8   8   NUM 7   nummod:gov  _   _
7   sešitů  sešit   NOUN    2   obj _   SpaceAfter=No
8   .   .   PUNCT   2   punct   _   _
```

Obrázek 6.3: Ohodnocení části slovní úlohy: *Rodiče koupili Toníkovi do školy 8 sešitů.* programem UDPipe.

```
# sent_id = 2
# text = Rodiče koupili Blance jen 5 sešitů.
1   Rodiče  rodič   NOUN    3   xcomp   _   _
2   koupili koupit  VERB    0   root    _   _
3   Blance  Blanka  PROPN   2   nsubj   _   _
4   jen jen PART    5   advmod:emph _   _
5   5   5   NUM 6   nummod:gov  _   _
6   sešitů  sešit   NOUN    2   obj _   SpaceAfter=No
7   .   .   PUNCT   2   punct   _   SpaceAfter=No
```

Obrázek 6.4: Ohodnocení části slovní úlohy: *Rodiče koupili Blance jen 5 sešitů.* programem UDPipe.

Přestože se věty od sebe liší, tak v obou případech by slovo *Rodiče* mělo být podmětem. Jak můžeme vidět na Obrázku 6.3 a Obrázku 6.4, tak ani v jednom případě není slovo *Rodiče* určeno jako podmět. Toto chybné určení má za následek špatnou tvorbu struktury a tím pádem i nesprávný výsledek.

Při pokusu o opravu chyb v kódu, docházelo k přefitování trénovací množiny. Tím pádem jsou chyby vzniklé chybným ohodnocením UDPipe neřešitelné. Jediný progress může nastat vydáním nového natrénovaného modelu.

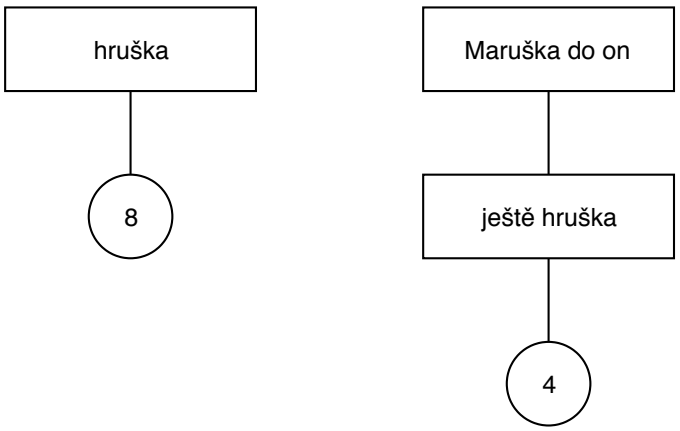
6.2 Chybný preprocessing

Chyba v preprocessingu nastala v 19.05% případů chybně řešených slovních úloh. Tato chyba by se v určitých případech dala eliminovat tím, že by se upravila implementace preprocessingu. V ostatních případech by bylo potřeba velmi spolehlivého nástroje pro syntaktickou analýzu a možnost skloňování slov.
Uvažujme slovní úlohu:
V dědečkově zahradě kvetou 3 třešně a 10 jabloní. Kolik stromů kvete v dědečkově zahradě?
Preprocessing rozdělil první větu na dvě věty následovně:
V dědečkově zahradě kvetou 3 třešně.
V dědečkově zahradě kvetou 3 třešně 10 jabloní.
Na první pohled je vidět, že rozdělení podle spojky „a“ nebylo validně provedeno, zároveň by v druhé větě mělo být slovo „kvetou“ ve správném tvaru.
Správný tvar druhé věty by vypadal následovaně:
V dědečkově zahradě kvete 10 jabloní.

6.3 Chybná struktura

Chybně vytvořená struktura se vyskytovala v 33.33% chybně řešených slovních úloh. Tato chyba vznikla, protože implementace tvorby struktur si nedokázala poradit s konkrétní slovní úlohou. Chyba se dá jednoduše eliminovat a to tak, že danou slovní úlohu zanalyzujeme a pokusíme se řešení implementovat do stávající tvorby struktur.

Příklad slovní úlohy:
V košíčku je 8 hrušek. Maruška do něho ještě 4 hrušky přidala. Kolik hrušek je už v košíčku?
Vytvořená struktura:



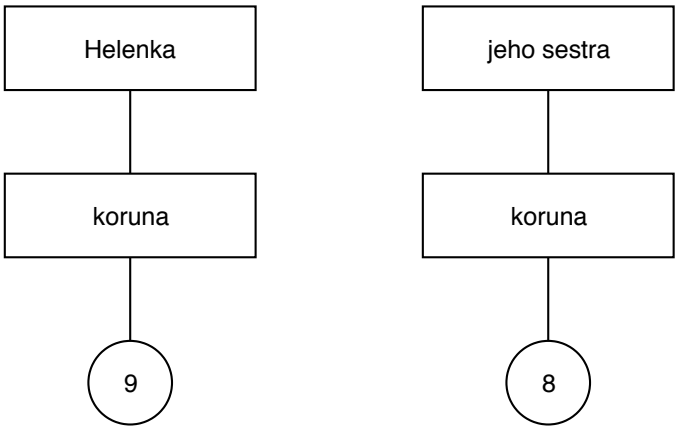
Obrázek 6.5: Chybně vytvořená struktura slovní úlohy: *V košíčku je 8 hrušek. Maruška do něho ještě 4 hrušky přidala. Kolik hrušek je už v košíčku?*

Hned na první pohled je vidět, že struktura na Obrázku 6.5 je chybně vytvořena. Moje implementace tvorby struktur nepředpokládá, že entitou je jiný větný člen, než podmět.

6.4 Chybné zpracování

Nejméně častou chybou je chybné zpracování struktury a to 9.52%. Slovní úlohy, ve kterých vznikla tato chyba je velmi těžké určit, vždy totiž souvisí s předešlými chybami. Chyby, které jsem klasifikoval jako chybné zpracování by se daly rozdělit do předešlých dvou skupin.
Příklad slovní úlohy:

Helenka má v kasičce 9 korun. Její sestra Věra má ve své kasičce o 1 korunu méně. Kolik korun má v kasičce Věra?



Obrázek 6.6: Chybné vyhodnocení struktury pro slovní úlohu: *Helenka má v kasičce 9 korun. Její sestra Věra má ve své kasičce o 1 korunu méně. Kolik korun má v kasičce Věra?*

Na struktuře v Obrázku 6.6 můžeme vidět, že struktura je fakticky správně. Jedinou malou chybou je chybný základní tvar „jeho sestra“. Slovní úloha by se nám správně vyhodnotila, kdyby místo „jeho sestra“ bylo slovo „Věra“.

Část III

Kombinace strojového učení, syntaktické analýzy, slovníků a heuristik

Kapitola 7

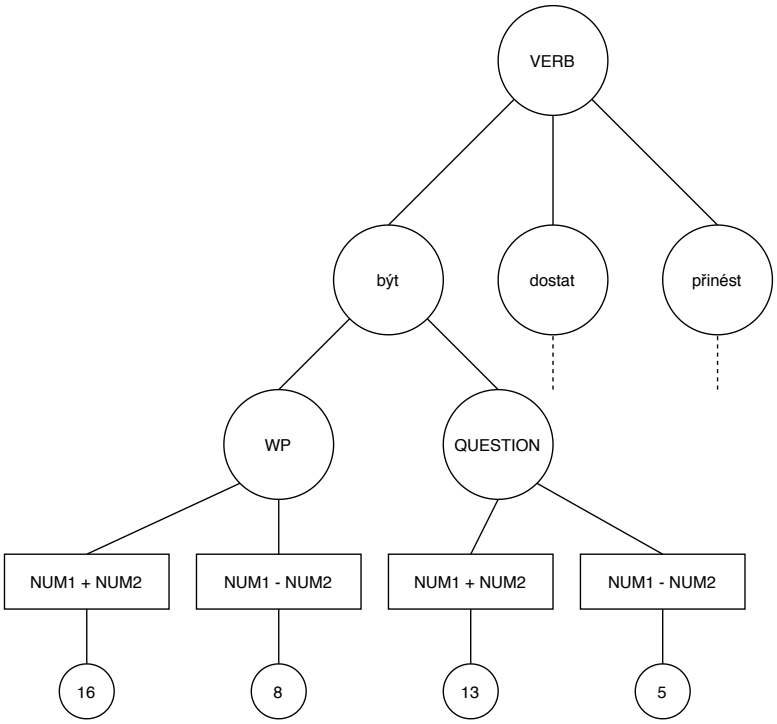
Zvolený postup

Tato část se věnuje kombinaci strojového učení, syntaktické analýzy, slovníků a heuristik. Kvůli nedostatkům programu UDPipe, které byly objeveny v prvním řešení, není v tomto druhém řešení program UDPipe použit. Místo programu UDPipe extrahujeme informace z korpusu SYN2015 [KCČ+15]. Extrahovaná data z korpusu jsou v podobě slovníku, kde klíčem je slovo a hodnotou je dvojice slovní druh a lemma. Slovní druh ve slovníku je vybrán podle absolutní četnosti výskytu slovního druhu ke slovu v korpusu. Extrahovaný slovník má velikost téměř 1 milion klíčů. Ani tento způsob nám nezaručuje bezchybnost, s chybovostí se potýkáme u určení slovního druhu. Slovní druh není vždy jednoznačný pro slovo, ale pro použití daného slova ve větě. Příkladem je slovo „růst“.
Slovo „růst“ jako podstatné jméno:
„Jeho růst je obrovský.“
Slovo „růst“ jako sloveso:
„Přestaň už růst!“

7.1 Slovní úloha jako bod v $\mathbb{R}^n$

Rozhodl jsem se ke slovním úlohám přistupovat jako k bodům v prostoru $\mathbb{R}^n$, kde n je počet unikátních výrazů, řešící některou ze slovních úloh v trénovací množině. Převod slovních úloh na bod je nezbytný pro aplikaci klasifikátoru. U tohoto řešení využijeme skutečnost, která je zmíněna v podkapitole 2.2 - ne všechny slovní druhy jsou k řešení slovní úlohy potřebné. Pro připomenutí

se jedná o podstatná a přídavná jména. Pro ostatní slovní druhy vytvoříme číselné vektory, pomocí kterých budeme slovní úlohy převádět na čísla.



Obrázek 7.1: Příklad vytvořené struktury pro sloveso, pomocí které převádíme slovní úlohy na bod v $\mathbb{R}^n$

Na Obrázku 7.1 můžeme vidět vytvořenou strukturu z trénovací množiny, se kterou budeme pracovat. Ze slovních úloh extrahujeme všechny slovní druhy kromě podstatného a přídavného jména. Všechna slova roztřídíme podle slovního druhu a k nim uložíme informaci, kde se ta slova nacházela, odlišíme tedy otázku slovní úlohy (QUESTION) od části slovní úlohy se zadanými údaji (WP) a dále uložíme informaci o výrazech příslušných úloh (NUM1 + NUM2, ...). Pro každý slovní druh nám vznikne stromová struktura, kde v listech jsou čísla, reprezentující absolutní četnost slov ve výrazech a částech slovních úloh. Díky této struktuře bude možné každou slovní úlohu reprezentovat jako bod $\mathbb{R}^n$.

Zjednodušené vytvoření číselného vektoru (za předpokladu, že uvažujeme pouze dva výrazy):
Uvažujme slovní úlohu:
Adamovi je 8 let. Tomášovi je 10 let. Kolik let je oběma klukům dohromady?
Počáteční číselný vektor:
{'NUM1 + NUM2': 0, 'NUM1 - NUM2': 0}
Číselný vektor na ukázku vytvořme pouze na základě sloves, proto použijeme

strukturu z Obrázku 7.1. Začneme iterováním přes věty a hledáním slovního druhu.
Věta *Adamovi je 8 let.* obsahuje sloveso „být“ a jedná se o část WP. Proto počáteční číselný vektor inkrementujeme následovně:
{'NUM1 + NUM2': 0 + 16, 'NUM1 - NUM2': 0 + 8}
Věta *Tomášovi je 10 let.* obsahuje sloveso „být“ a jedná se o část WP (část slovní úlohy bez otázky). Proto číselný vektor inkrementujeme následovně:
{'NUM1 + NUM2': 16 + 16, 'NUM1 - NUM2': 8 + 8}
Věta *Kolik let je oběma klukům dohromady?* obsahuje sloveso „být“ a jedná se o část QUESTION (otázka slovní úlohy). Proto číselný vektor inkrementujeme následovně:
{'NUM1 + NUM2': 32 + 13, 'NUM1 - NUM2': 16 + 5}
Výsledný číselný vektor je:
{'NUM1 + NUM2': 45, 'NUM1 - NUM2': 21}
(45, 21)

7.2 Použití SVM

V předchozí podkapitole je popsáno, jakým způsobem převedeme slovní úlohu na číslo. V této podkapitole se budeme věnovat použití metody *Support vector machines* pro naučení klasifikátoru. Motivací pro výběr této metody bylo několik. *Support vector machines* je metoda, která nám umožňuje práci s daty ve velkých dimenzích a na rozdíl od perceptronového algoritmu dochází k menšímu přefitování. Předností metody *Support vector machines* je její flexibilita docílená pomocí volby jádra (kernel) a parametru *C*. *Kernel* nám dává možnost v určitých případech lineárně separovat nelineárně separovatelná data tím, že data zobrazí do vyšší dimenze, kde lineárně separovat jdou. Parametr *C* nám udává penalizaci za špatné klasifikování. Pro více informací k *Support vector machines* doporučuji [Bis06]. Pro SVM je použita knihovna *scikit-learn* [1].

Pro připomenutí nejzákladnější varianta SVM (binární klasifikace, bez kernelu, bez parametru C) [VF09]:
Cílem SVM je najít nadrovinu $\langle w, x \rangle + b \geq 0$, která maximalizuje vzdálenost bodů od nadroviny. Trénovací množina má podobu $\{(x_1, y_1), ..., (x_m, y_m)\}$, kde $x_i \in \mathbb{R}^n$ jsou body a $y_i \in \{-1, 1\}$ značí příslušnost třídy. Řešení lze vyjádřit jako optimalizační úlohu:

$$w, b = \arg\min_{w,b} \frac{1}{2} \|w\|^2$$

[1] https://scikit-learn.org/stable/

Za podmínek:

$$\langle w, x_i \rangle + b \geq 1, \quad y_i = +1$$

$$\langle w, x_i \rangle + b \leq -1, \quad y_i = -1$$

Výrazy v trénovací množině nejsou rovnoměrně zastoupeny, proto za pomoci genetického algoritmu najdeme optimální váhu pro každou složku bodu. Nové váhy jsou uloženy v podobě slovníku, kde klíčem je část slovní úlohy, tzn. WP nebo QUESTION, a hodnotou je další slovník, kde klíčem je slovní druh a hodnotou váha (celé číslo). Tyto váhy využijeme při převodu slovní úlohy na bod.

Pro výběr *kernelu* a parametru C byl také použit genetický algoritmus. Parametr C by se správně měl zjišťovat crossvalidací, kterou bohužel v případě datasetů nebylo možné použít, kvůli nerovnoměrnému zastoupení výrazů a malé velikosti trénovací a testovací množiny. Nejlépe vyšel *Radial basis function kernel*, který by v případě většího počtu výrazů a velké trénovací množiny mohl mít delší výpočetní čas. Dále byly testovány *Linear kernel* a *Polynomial kernel*, kde oba dosahovaly horších výsledků, než *Radial basis function kernel*. Původní myšlenka hledání parametru C byla pro hodnoty v rozmezí od 10^{-3}, až po 10^3. Tato myšlenka se ukázala jako velmi nepraktická, z důvodu výběru nejvyšší hodnoty, která měla za následek velkou úspěšnost na trénovacích datech, za to menší úspěšnost na testovacích datech. Proto byla zvolena hodnota parametru C rovna 10, tato hodnota dosahovala dobrého poměru úspěšnosti na trénovací a testovací množině.

Genetický algoritmus je spuštěn nad daty z trénovací množiny. Hodnoty v nulté generaci jsou voleny náhodně a velikost nulté generace je větší než 10. Maximální počet generací je možné předat parametrem, defaultně je nastaven počet 100. Po vyhodnocení nulté generace se vybere 10 nejúspěšnějších jedinců. Úspěšností jedince je myšlena úspěšnost klasifikátoru naučeném na trénovacích datech a validovaným na trénovacích datech. Z nejúspěšnějších jedinců se následně vytvoří nová generace reprodukcí (kopíruje jedince do nové generace beze změny), křížením (jedince rozpůlím a doplním o nové hodnoty) a mutací (z jedince vytvořím 10 nových jedinců tím, že ke každé složce přičtu náhodně celé číslo v intervalu <-10, 10>). Zastavovací podmínka pro genetický algoritmus je úspěšnost 1.

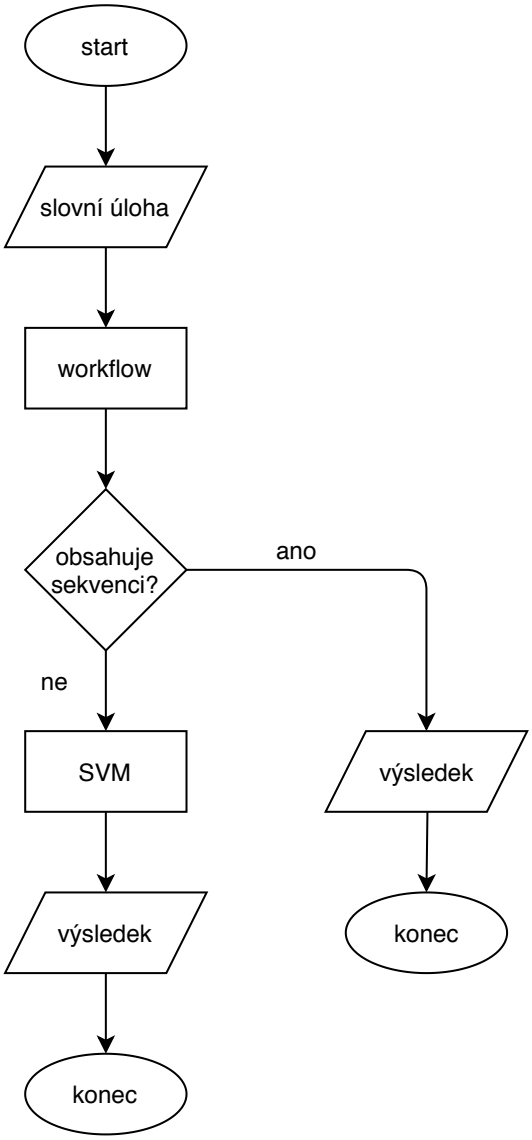
7.3 Sanity check

Tuto kapitolu jsem pojmenoval „sanity check“, podle terminologie stanovené v článku [Har14] (viz Kapitola 2) pro techniku představenou v [KAZB14]. Nate Kushman ve své práci využívá předpoklad, který také využiji ve svém řešení, že výsledkem slovní úlohy je nezáporné číslo. Proto při tvorbě bodu, který je popsán v podkapitole 7.1, nastavím na hodnotu 0 všechny položky, které reprezentují výrazy, pro které by vycházel záporný výsledek.

Další předpoklad, který je v tomto řešení použit, je hledání sekvencí. Jde o podobný princip, na kterém funguje komponenta *existuje zkratka?* z prvního řešení (popsáno v podkapitole 5.4), která je více popsána v podkapitole 5.4. Hledání sekvencí se nyní liší tím, že sekvence byly sbírány poloautomaticky a sekvence jsou komplexnější a jsou hledány v obou částech slovní úlohy - QUESTION, WP.
Příklad vyextrahované sekvence:
NOUN a NUM krát málo NOUN | NUM1 / NUM2
Příklad slovní úlohy, ve které by byla sekvence nalezena:
Maminka koupila na trhu 15 kg třešní a 3 krát méně jahod. Kolik kg ovoce koupila dohromady?
Sekvence je řetězec, ve kterém jsou podstatná jména, přídavná jména a čísla nahrazena identifikátory NOUN, ADJ, NUM, ostatní slova jsou ve svém základním tvaru. Sekvence jsou oddělená od výrazu pomocí symbolu „|“.

7.4 Algoritmus

V této podkapitole se budeme blíže věnovat samotnému algoritmu. Sled komponent algoritmu je na Obrázku 7.2.



Obrázek 7.2: Diagram znázorňující algoritmus.

Vysvětlení komponent algoritmu.

- vstupem je slovní úloha
- workflow
  - Dochází k převodu slovní úlohy na číslo v $\mathbb{R}^n$ postupem, který je popsán v podkapitole 7.1.
- SVM

- Před samotným algoritmem dochází k normalizaci dat. Data jsou normalizována tak, aby střední hodnota byla rovna 0 a směrodatná odchylka byla rovna 1. V algoritmu SVM je použit *Radial basis function kernel* a parametr C je roven 10. Výběr těchto konkrétních parametrů je popsán v podkapitole 7.2.

- Kód pro normalizaci:

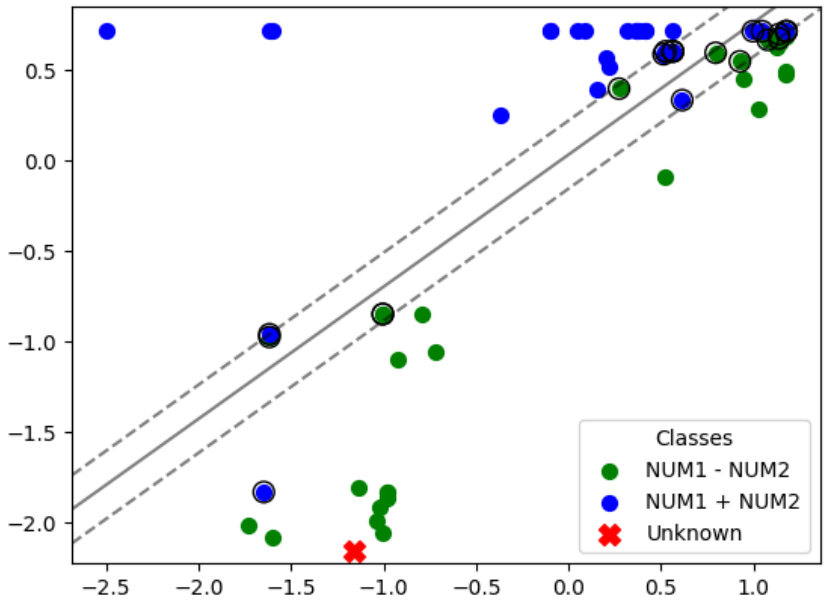
```
import numpy as np

def scale(x_points):
        x_mean = np.mean(x_points, axis=0)
        x_std = np.std(x_points, axis=0)
        x_scaled = (x_points - x_mean) / x_std
        return x_scaled
```

- obsahuje sekvenci?

  - Projde sekvenci výrazů (viz příloha C), pokud existuje shoda, aplikuje výraz nebo kombinaci výrazů. Hledání sekvence probíhá v QUESTION (otázka slovní úlohy), WP (část slovní úlohy bez otázky). V tomto případě byly sekvence sbírány poloautomaticky.

- výsledek - na rozdíl od předchozího řešení toto řešení vždy vrací nezáporné číslo

Zjednodušený příklad použití algoritmu:
vstup: *Kája si nechal dát na talíř 5 jahodových knedlíků. Má je sice moc rád, ale snědl jenom 4 knedlíky. Kolik knedlíků zbylo na talíři?*
workflow: V tomto zjednodušeném příkladu použití bereme, že n=2, proto převedený normalizovaný bod vypadá následovně: (-1.16131931, -2.15062701)
SVM:



Obrázek 7.3: Graf na kterém jsou vidět body z trénovací množiny a neznámý bod.

Na grafu, který je na Obrázku 7.3 můžeme vidět, že klasifikace pomocí metody SVM byla úspěšná a danému bodu by byl přiřazen výraz NUM1 - NUM2. Body, které jsou v kroužku jsou *support vektory* a přerušované přímky značí *margin*.

obsahuje sekvenci?: Komponenta *obsahuje sekvenci?* by v tomto případě podle počtu čísel ve slovní úloze, který je 2, a slovese „zbylo“ v otázce určila, že se jedná o výraz NUM1 - NUM2.

výsledek: 1

Kapitola 8

Evaluace

V tomto řešení není zapotřebí převod do CONLLU, díky čemuž nám celkové řešení spolu s předzpracováním dat trvá kratší dobu.

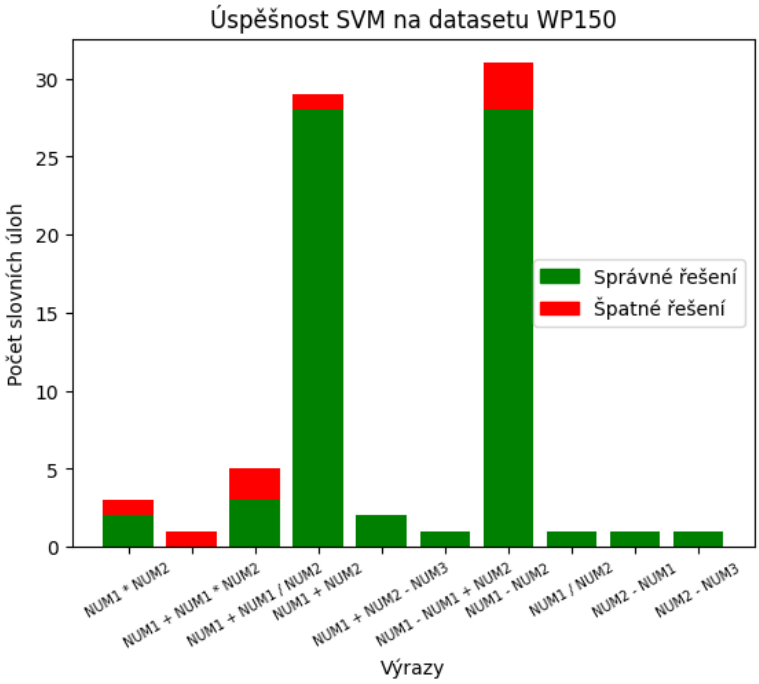
OS	**procesor**	**čas [s]**
WSL	Intel(R) Core(TM) i7-8750H CPU	2.626

Tabulka 8.1: Údaje o rychlosti řešení. V rámci měřeného času byla načtena trénovací množina, na které byl natrénovaný klasifikátor a následně proběhlo vyhodnocení nad testovací množinou datasetu WP500.

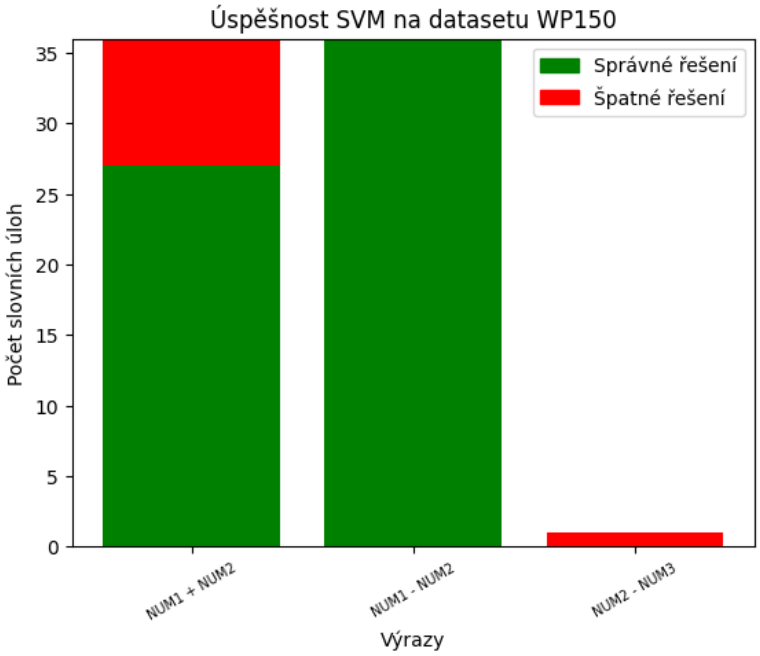
V Tabulce 8.2 je možné vidět úspěšnosti na datasetech.

	WP150 dataset	**WP500 dataset**
trénovací množina	89.33%	86.8%
testovací množina	86.30%	74.89%

Tabulka 8.2: Úspěšnost na datasetech pro testovací a trénovací množinu.

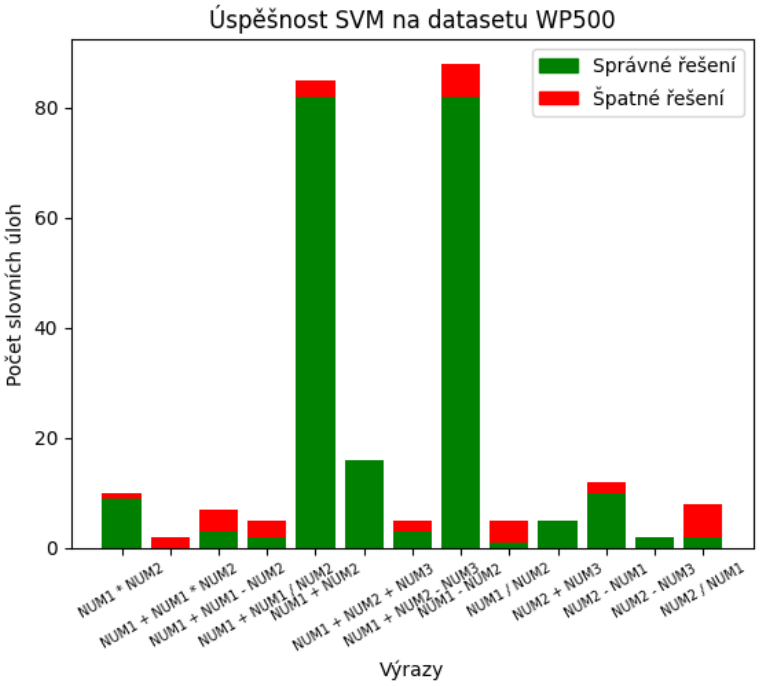


(a): Vyhodnocení na trénovací množině datasetu WP500.

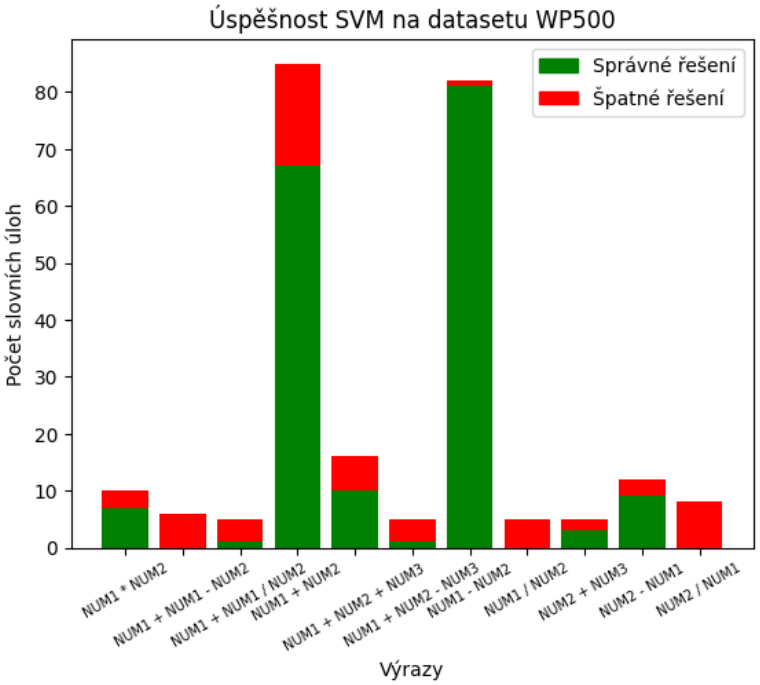


(b): Vyhodnocení na testovací množině datasetu WP150.

Obrázek 8.1: Úspěšnost trénovací a testovací množiny datasetu WP150



(a): Vyhodnocení na trénovací množině datasetu WP500.



(b): Vyhodnocení na testovací množině datasetu WP500.

Obrázek 8.2: Úspěšnost trénovací a testovací množiny datasetu WP500

V rámci evaluace došlo k porovnání s heuristikou, kde řešením je výraz, který

reprezentuje v číselném vektoru složka s největší hodnotou. Toto porovnání proběhlo na jednotkových vahách a číselných vektor nebyl normalizován. Příklad slovní úlohy, která byla heuristikou špatně klasifikována, ale správně klasifikována navrhovaným řešením:
Na louce poletuje 35 žlutých motýlů a 5 krát méně modrých motýlů. Kolik motýlů poletuje na louce?

	úspěšnost MAX [%]	**úspěšnost SVM** [%]
trénovací WP150	0.76	92
testovací WP150	89.04	87.67
trénovací WP500	69.2	85.2
testovací WP500	66.1	76.15

Tabulka 8.3: Přehled úspěšnosti heuristiky MAX a řešení SVM. Heuristiku MAX je nutné spouštět s jednotkovými váhami, proto v tomto případě bylo řešení SVM spouštěno také s jednotkovými vahami.

Rád bych zmínil úspěšnost komponenty *obsahuje sekvenci?*, kterou můžeme pozorovat v následnicích třech tabulkách. V tabulce 8.5a můžeme vidět úspěšnost s použitím komponenty, v tabulce 8.5b úspěšnost bez použití komponenty. Tabulka 8.6 nám dává informaci o tom, kolikrát byla komponenta použita, a v závorce je údaj, který nám udává, kolikrát použití zkratky vedlo k chybnému výsledku.

(a) : Komponenta je použita.

	WP150	WP500
train	89.33%	86.8%
test	86.30%	74.89%

(b) : Komponenta není použita.

	WP150	WP500
train	90.66%	87.6%
test	86.3%	61.08%

Tabulka 8.4: Úspěšnost řešení s použitím komponenty *obsahuje sekvenci?* a bez použití.

Z tabulek vidíme, že bez použití komponenty došlo k mírnému nárůstu úspěšnosti na trénovací množině, ale k poklesu úspěšnosti o 13% na testovací množině datasetu WP500.

	WP150	WP500
train	32 (3)	95 (10)
test	29 (0)	115 (8)

Tabulka 8.6: Informace o tom, kolikrát byla komponenta *obsahuje sekvenci?* v řešení použita. Údaj v závorce udává, kolikrát použití komponenty vedlo k chybnému výsledku.

8.1 Příčina chyb

Možnou příčinou chyb u tohoto řešení je chybné reprezentování slovních úloh jako bodů. Chyby mohou být způsobeny i volbou nevhodných parametrů v metodě SVM. Komponenta *obsahuje sekvenci?* funguje spolehlivě pro slovní úlohy, jejichž řešením jsou jednoduché výrazy, obsahující pouze jednu matematickou operaci. V případě složitějších výrazů nemusí slovní úloha obsahovat všechny sekvence, díky kterým by *obsahuje sekvenci?* objevila hledaný výraz.

Analyzoval jsem 10 náhodně vybraných chybně klasifikovaných slovních úloh z trénovací množiny datasetu WP500. Mezi chybnými slovními úlohami se objevila slovní úloha, jejíž číselný vektor byl nulový. Tento stav zapříčinil vznik struktury, který je popsán v podkapitole 7.1, kde se do struktury nedostanou jednotkové výskyty slov. Zbylé chybně klasifikované úlohy se těžko analyzují, což demonstruje následující příklad:

Martinka měla 6 panenek. K svátku dostala zase 1 panenku. Kolik panenek už Martinka má? Pro tuto slovní úlohu vyšel číselný vektor:
(0, 0, 0, 0, -240, 0, 0, -129, 0, 0, 0, 0, 0, 0)
Nenulové položky reprezentují výrazy 'NUM1 + NUM2' a 'NUM1 - NUM2'.
Normalizovaný vektor má podobu:
(0.45, 0.70, 0.45, 0.87, 0.77, 0.30, 0.32, 0.56, 1, 0.29, 0.36, 0.16, 0.38)
Z obou číselných vektorů je možné vidět, že nedostaneme informace, proč daná slovní úloha byla špatně klasifikována.

Část IV

Shrnutí

Kapitola 9

Porovnání obou metod

Obě evaluace řešení, které jsou popsány v kapitolách 6 a 8, proběhly podle stejných pravidel. Pravidla byla taková, že se vyhodnocovaly výrazy, které jsou průnikem trénovací a testovací množiny. Druhé řešení je schopné vyhodnocovat výrazy, které jsou pouze zastoupeny v trénovací množině. První řešení na rozdíl od druhého toto omezení nemá.
Příkladem je slovní úloha:
Adam má 5 kuliček. Tomáš má 2 kuličky. Anna má 3 kuličky. Pepa má 1 kuličku. Kolik kuliček mají Anna a Pepa?
Výraz řešící tuto slovní úlohu není zastoupen ani v jedné trénovací množině, přesto první řešení vyhodnotí slovní úlohu správně. Můžeme tedy říct, že první řešení je při vyhodnocování nezávislé na trénovací množině.

	úspěšnost SA	**úspěšnost SVM**
trénovací WP150	76%	89.33%
testovací WP150	67.12%	86.30%
trénovací WP500	53.2%	86.8%
testovací WP500	28.87%	74.89%

Tabulka 9.1: Přehled úspěšnosti obou řešení.

I přesto, že první řešení mělo nižší úspěšnost než druhé řešení, tak se domnívám, že pro vyhodnocování slovních úloh je koncept prvního řešení nezbytný. V některých případech může mít i klíčový přínos. Pokud bychom chtěli řešit slovní úlohy a spolu s výsledkem znát i postup, pak by postup byl lépe získatelný z prvního řešení.

Velice mě překvapila úspěšnost použití zkratek v prvním řešení a její nadstavby komponenty *obsahuje sekvenci?* ve druhém řešení. V prvním řešení tvořily zkratky pouze 2 databáze slov, které rozlišovaly 2 možné výrazy. Ve druhém řešení již šlo o hledání posloupnosti více slov. Hledání klíčových slov popřípadě jejich posloupností je úspěšnější pro řešení jednoduchých výrazů, pokud výsledný výraz obsahuje více matematických operací, pak takové hledání nemusí validně zafungovat a může vrátit pouze část výsledku.

Kapitola 10

Závěr

V této kapitole je obecné shrnutí automatického vyhodnocování matematických slovních úloh pomocí dvou navržených řešení. Každé řešení k dané problematice přistupuje rozdílně. První řešení vyhodnocuje pouze za pomoci syntaktické analýzy, slovníků a heuristik a na základě pravidel vytváří stromové struktury, které obsahují důležité informace extrahované ze slovní úlohy. Druhé řešení funguje primárně na strojovém učení, kdy slovní úlohu reprezentujeme jako bod a následně pomocí metody SVM natrénujeme klasifikátor.

V každém řešení je ještě prostor ke zlepšení. V případě prvního řešení se jedná o zlepšení tvorby struktur, která je závislá na syntaktickém analyzátoru. Proto je také potřeba i spolehlivější syntaktický analyzátor. Zároveň by mělo velký přínos zaměřit se na tvorbu struktur pro složitá souvětí, kde vystupuje více vět hlavních, více vět vedlejších, nebo jejich kombinace. V případě druhého řešení je prostor pro zlepšení v reprezentaci slovní úlohy jako bodu, kde by při tvorbě bodu mohl mít vliv i slovní druh slov nebo pořadí slov.

Dalším možným rozšířením navrhovaných řešení by mohlo být například vyhodnocování slovních úloh o více otázkách, nebo slovní úlohy, jejíž řešení vede na soustavu rovnic (oba případy jsou nad rámec této práce). Pro takový typ problémů by navrhovaná řešení potřebovala modifikovat, nebo by bylo potřeba použít jiný přístup. V případě slovních úloh o více otázkách se domnívám, že by lépe zafungovalo první navrhované řešení, a to z důvodu rozpadu slovní úlohy na strukturu, kde by se následně struktura vyhodnocovala pro každou otázku zvlášť. Slovní úlohy, jejichž řešením je soustava rovnic, bych řešil jinou metodou, a to mapováním slovních úloh na předem definované

vzory.

Lingvistické nástroje pro český jazyk nelze porovnávat s těmi pro anglický jazyk, ale i tak mě velice pozitivně překvapila úroveň lingvistických nástrojů pro češtinu. Kvalitní a spolehlivé lingvistické nástroje mohou velmi pomoci pro vyhodnocování slovních úloh, ale i pro jiná pole výzkumu ve zpracování přirozeného jazyka.

Navržené metody by se do budoucna mohly stát součástí komplexnější aplikace, která by žákům pomáhala se slovními úlohami. Mohla by fungovat tak, že by si uživatel vybral složitost slovních úloh, popřípadě typ slovní úlohy. Aplikace by v rámci svých možností uživateli vrátila sadu slovních úloh, které by řešil. V případě správného výsledku by uživatel byl pochválen a dostal by možnost prohlédnout si postup řešení. V případě špatného výsledku by mu byla odkryta část postupu a uživatel by dostal možnost si výsledek opravit. Uživatel by měl možnost zadat svou vlastní slovní úlohu, popřípadě by mohl modifikovat slovní úlohu, kterou mu vrátil program. Mohlo by se jednat o webovou aplikaci, kam by se žáci přihlásili a učitel i rodiče by mohli sledovat žákův pokrok.

Automatické vyhodnocování matematických slovních úloh je výzvou pro bádání už několik let. Jedná se o dlouholetý cíl zpracování přirozeného jazyka. Budoucnost pro automatické vyhodnocování matematických slovních úloh vidím v kvalitních lingvistických nástrojích v kombinaci se strojovým učením.

Přílohy

Příloha A

Literatura

[Bis06] Christopher M. Bishop, *Pattern recognition and machine learning (information science and statistics)*, Springer-Verlag, Berlin, Heidelberg, 2006.

[Bob64] Daniel G. Bobrow, *Natural language input for a computer problem solving system*, Tech. report, USA, 1964.

[FF63] Edward A. Feigenbaum and Julian Feldman, *Computers and thought*, McGraw-Hill, Inc., USA, 1963.

[Har14] Larry Hardesty, *Computer system automatically solves word problems*, `http://news.mit.edu/2014/computer-system-automatically-solves-word-problems-0502`, May 2014.

[HHEK14] Mohammad Javad Hosseini, Hannaneh Hajishirzi, Oren Etzioni, and Nate Kushman, *Learning to solve arithmetic word problems with verb categorization*, Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP) (Doha, Qatar), Association for Computational Linguistics, October 2014, pp. 523–533.

[HLLY18] Danqing Huang, Jing Liu, Chin-Yew Lin, and Jian Yin, *Neural math word problem solver with reinforcement learning*, Proceedings of the 27th International Conference on Computational Linguistics (Santa Fe, New Mexico, USA), Association for Computational Linguistics, August 2018, pp. 213–223.

[HSL+16] Danqing Huang, Shuming Shi, Chin-Yew Lin, Jian Yin, and Wei-Ying Ma, *How well do computers solve math word problems?*

large-scale dataset construction and evaluation, Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers) (Berlin, Germany), Association for Computational Linguistics, August 2016, pp. 887–896.

[KAZB14] Nate Kushman, Yoav Artzi, Luke Zettlemoyer, and Regina Barzilay, *Learning to automatically solve algebra word problems*, Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers) (Baltimore, Maryland), Association for Computational Linguistics, June 2014, pp. 271–281.

[KCČ+15] Michal Křen, Václav Cvrček, Tomáš Čapka, Anna Čermáková, Milena Hnátková, Lucie Chlumská, Dominika Kováříková, Tomáš Jelínek, Vladimír Petkevič, Pavel Procházka, Hana Skoumalová, Michal Škrabal, Petr Truneček, Pavel Vondřička, and Adrian Zasina, *SYN2015: representative corpus of written czech*, 2015, LINDAT/CLARIAH-CZ digital library at the Institute of Formal and Applied Linguistics (ÚFAL), Faculty of Mathematics and Physics, Charles University.

[MH04] Jarmila NOVOTNÁ a Naďa VONDROVÁ Milan HEJNÝ, *Dvadcet pět kapitol z didaktiky matematiky*, Univerzita Karlova, Pedagogická fakulta, 2004.

[Rei96] Marie Reischigová, *Matematika na základní a obecné škole ve slovních úlohách*, Pansofia, 1996.

[SS17] Milan Straka and Jana Straková, *Tokenizing, pos tagging, lemmatizing and parsing ud 2.0 with udpipe*, Proceedings of the CoNLL 2017 Shared Task: Multilingual Parsing from Raw Text to Universal Dependencies (Vancouver, Canada), Association for Computational Linguistics, August 2017, pp. 88–99.

[VF09] Martin Švec Vojtěch Franc, *Exercise in rpz support vector machines*, `https://cw.fel.cvut.cz/b191/_media/courses/be5b33rpz/labs/08_svm/rpzcv-svm-eng.pdf`, November 2009, Accessed: 2020-05-16.

[ZWZ+19] Dongxiang Zhang, Lei Wang, Luming Zhang, Bing Dai, and Heng Shen, *The gap of semantic parsing: A survey on automatic math word problem solvers*, IEEE Transactions on Pattern Analysis and Machine Intelligence **PP** (2019), 1–1.

Příloha B

Dokumentace k přiloženému programu

Následující text je dokumentace k programu, který je přiložený k práci a obsahuje implementaci dvou řešení, popsaných v kapitolách 5 a 7.

Automatickcé vyhodnocování slovních úloh

Tento projekt vznikl v rámci bakalářské práce. Cílem tohoto projektu bylo vytvořit program, který by řešil jednoduché matematické slovní úlohy v českém jazyce pro žáky 1. - 3. třídy základních škol.

Tento program používá extrahovaná data z korpusu SYN2015.
https://wiki.korpus.cz/doku.php/:cnk:syn2015

Potřebné knihovny pro spuštění jsou specifikovány v souboru requirements.txt.

Spuštění

Spustitelné soubory:

- solver.py
- testing.py
- training.py

solver.py

Slouží uživateli k testování slovních úloh. Uživatel má možnost zadat jednu slovní úlohu, nebo zadat cestu k souboru s více slovními úlohami. Slovní úlohy předávány programu musí být v následujícím formátu:

- jeden řádek, jedna slovní úloha
- za každým interpunkčním znaménkem musí být mezera
- všechna důležitá čísla musí být napsaná číslicemi
- slovní úlohy nesmí obsahovat přímou řeč (tu bohužel program nepodporuje)

Příklad uložení slovních úloh ve správném formátu je v souboru:

- ./data/test.txt

Pro spuštění zadejte následující příkaz:

```
python3 solver.py
```

Po spuštění se zobrazí základní informace k programu a uživatel je vyzván k vybrání nějaké akce.

```
INFO:
Tento program vznikl v rámci bakalářské práce.
Program automaticky vyhodnocuje matematické slovní úlohy pro český jazyk.
Funguje na principu strojového učení a syntaktické analýzy, slovníků a heuristik.
Více informací je možné nalézt v samotné bakalářské práci, nebo v dokumentaci
programu.
_____
```

```
Obsluha programu:
Klasifikátor je natrénovaný na množině slovních úloh, kterou je možné nalézt v
adresáři ./data/traindata.
Program není schopen vyhodnotit slovní úlohy, jejichž řešením je jiný výraz než v
trénovací množině.
[1]: vstup slovní úloha
[2]: vstup soubor se slovními úlohami
[3]: konec
Jakou akci chcete vykonat?
```

Program není schopen vyhodnotit slovní úlohy, jejichž řešením je jiný výraz než v trénovací množině. Po vybrání nějaké akce dojde k jejímu vykonání a následně má uživatel možnost zadat další akci:

```
[1]: vstup slovní úloha
[2]: vstup soubor se slovními úlohami
[3]: konec
Jakou akci chcete vykonat?
1
Zadejte slovní úlohu:
Maminka koupila na trhu 15 kg třešní a 3 krát méně jahod. Kolik kg ovoce koupila
dohromady?
**********
Programu vyšel výsledek: 20

**********
[1]: vstup slovní úloha
[2]: vstup soubor se slovními úlohami
[3]: konec
Jakou akci chcete vykonat?
```

Rád bych uživatele upozornil, že program slouží k řešení slovních úloh pro 1. - 3. třídy základních škol.

testing.py

Tento soubor pouští evaluaci na slovních úlohách v adresaři testdata pro datasety WP150 a WP500. Výsledkem je tabulka s procentuální úspěšností jednotlivých výrazů a graf úspěšnosti výrazů.

training.py

Tento soubor pouští evaluaci na slovních úlohách v adresaři traindata pro datasety WP150 a WP500. Výsledkem je tabulka s procentuální úspěšností jednotlivých výrazů a graf úspěšnosti výrazů.

Příloha C

Databáze slov a sekvencí

Příklad uložení databáze slov pro komponentu *existuje zkratka?* v prvním řešení. Slova, která jsou v databázích, jsou slova, která program UDPipe vrátil jako lemma (v některých případech nejsou tato slova v jazykově správném tvaru).

./database/database_sa/plus.txt

```
dohromady
celkem
```

./database/database_sa/minus.txt

```
zůstat
zbýt
chybět
starý
zbývat
utřít
snědnout
```

Příklad uložení sekvencí slov pro komponentu *obsahuje sekvenci?* v druhém řešení.

./database/sequences_svm/question/sequences.txt

```
Kolik NOUN NOUN zbýt? | NUM1 - NUM2
Kolik NOUN NOUN zůstat? | NUM1 - NUM2
Kolik NOUN zbýt NOUN v NOUN? | NUM1 - NUM2
o kolik _ | max(NUM1,NUM2) - min(NUM1,NUM2)
celkem | NUM1 + NUM2
dohromady | NUM1 + NUM2
zbýt | NUM1 - NUM2
zůstat | NUM1 - NUM2
```

./database/sequences_svm/wp/sequences.txt

```
NOUN a NUM krát málo NOUN | NUM1 / NUM2
mít tak o NUM NOUN hodně | NUM1 - NUM2
mít tak o NUM NOUN málo | NUM1 + NUM2
což být o NUM NOUN hodně | NUM1 - NUM2
což být o NUM NOUN málo | NUM1 + NUM2
být tedy o NUM NOUN starý | NUM1 - NUM2
být tedy o NUM NOUN mladý | NUM1 + NUM2
což být o NUM hodně | NUM1 - NUM2
což být o NUM málo | NUM1 + NUM2
mít tak o NUM hodně | NUM1 - NUM2
mít tak o NUM málo | NUM1 + NUM2
být o NUM NOUN vysoký | NUM1 + NUM2
být o NUM NOUN malý | NUM1 - NUM2
být o NUM NOUN zlevnit | NUM1 - NUM2
být o NUM NOUN zdražit | NUM1 + NUM2
být o NUM NOUN levný | NUM1 - NUM2
být o NUM NOUN drahý | NUM1 + NUM2
o NUM NOUN hodně | NUM1 + NUM2
o NUM NOUN málo | NUM1 - NUM2
o NUM NOUN dále | NUM1 + NUM2
o NUM NOUN mladý | NUM1 - NUM2
o NUM NOUN starý | NUM1 + NUM2
o NUM NOUN dlouho | NUM1 + NUM2
o NUM málo | NUM1 - NUM2
o NUM hodně | NUM1 + NUM2
```