VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INFORMAČNÍCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INFORMATION SYSTEMS

EXTRAKCE HLAVNÍHO TEXTU Z WEBOVÝCH DOKUMENTŮ

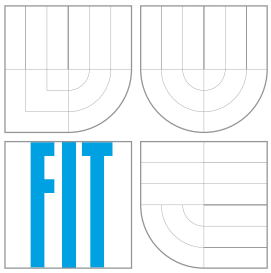
BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE
AUTHOR

DANIEL MRÓZEK

BRNO 2014

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INFORMAČNÍCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INFORMATION SYSTEMS

EXTRAKCE HLAVNÍHO TEXTU Z WEBOVÝCH DOKUMENTŮ

MAIN TEXT EXTRACTION FROM WEB DOCUMENTS

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE
AUTHOR

DANIEL MRÓZEK

VEDOUCÍ PRÁCE
SUPERVISOR

Ing. VLADIMÍR BARTÍK, Ph.D.,

BRNO 2014

Abstrakt

Tato práce se zabývá extrakcí hlavního textu z webových dokumentů ve formátu HTML. Jsou zde popsány již použité metody a jejich rozdělení. Praktická část se pak zabývá návrhem algoritmu pro detekci hlavního textu v HTML stránkách založeném na analýze především textových rysů stránky v kombinaci s vlastnostmi založených na pozici v dokumentu. Výsledná klasifikace je řešena pomocí vícevrstvé perceptonové sítě. Je zde rovněž popsána implementace navrhnutého algoritmu, postup při testování a prezentace zjištěných výsledků.

Abstract

This thesis deals with the main text extraction from the web documents in HTML format. It describes some methods that are already used and their separation. The goal of the practical part is to propose an algorithm for main text detection in HTML pages using primarily text features in combination with position features. Block classification is solved by multilayer perceptron. It also describes implementation of the proposed algorithm, the testing procedure and presentation of the obtained results.

Klíčová slova

extrakce, dolování, hlavní obsah, textové rysy, HTML, MLP, umělá neuronová síť

Keywords

extraction, mining, main text, text features, HTML, MLP, artificial neural network

Citace

Daniel Mrózek: Extrakce hlavního textu z webových
dokumentů, bakalářská práce, Brno, FIT VUT v Brně, 2014

Extrakce hlavního textu z webových dokumentů

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením pana ...

.....
Daniel Mrózek
31. července 2014

Poděkování

Děkuji vedoucímu práce Ing. Vladimírovi Bartíkovi, Ph.D. za trpělivost a vstřícnost. Děkuji také svým blízkým a přátelům za podporu.

Obsah

Kapitola 1

Úvod

Na internetu se nachází nepřeberné množství informací, většinou ve formě HTML stránek. Ty však neobsahují pouze užitečný obsah, ale rovněž různé navigační prvky, odkazy, reklamy apod. Pro efektivní analýzu takovýchto dokumentů je potřeba znát hlavní obsah. Člověk jej od neužitečných informací jednoduše rozliší, počítač ne. Proto je potřeba mít k dispozici nástroj, program, který je schopen plně samostatně rozlišit co je hlavní obsah a co je generický obsah. Získání pouze hlavního obsahu má pak nesporné výhody při analýze dokumentů, hledání duplicit a přináší menší datovou náročnost.

Cílem této práce je navrhnout program pro extrakci hlavního textu z webových dokumentů a implementovat jej. Současná řešení nejsou často plně vyhovující, zejména z důvodu jejich zaměření na konkrétní obor nebo jazyk. Proto se v této práci budeme snažit navrhnout program, který bude univerzální a nebude jazykově ani oborově specifický. Program bude využívat přístupy z již známých řešení a vzájemně je kombinovat, díky čemuž by měl dosahovat podobných, v ideálním případě lepších výsledků a neměl by být v důsledku zaměřen pouze na jednu oblast. Program může být pak použit jako součást větších systému především v oblasti dolování dat a analýzy dokumentů.

Práce je rozdělena následujícím způsobem. V kapitole 2 budou probrány různé postupy využívané při detekci hlavního obsahu, přičemž bude kladen důraz na metody založené na DOM (objektový model dokumentu). Rovněž zde bude uvedena nezbytná teorie pro pochopení následné implementace. Kapitola 3 obsahuje návrh systému pro extrakci hlavního obsahu a porovnává jej s dalšími, již implementovanými systémy. V 4. kapitole bude popsána již samotná implementace systému v jazyce Ruby. Kapitola 5 popisuje průběh testování a porovnání s některými již existujícími řešeními. V 6. kapitole, tedy závěru, se nalézá ohlédnutí za výsledky a jsou popsány další možnosti rozšíření.

Kapitola 2

Existující nástroje, přístupy a neuronové sítě

Tato kapitola pojednává o studiích souvisejících s touto prací. Zahrnuje rovněž rozdělení metod na základě toho na jaké úrovní přistupují k dokumentu.

2.1 Dolování textu

Extrakce hlavního textu spadá do oblasti tak zvaného dolování textu.

Dolování textu lze definovat jako intenzivní proces s využitím znalostí kdy uživatel interaguje s kolekcí dokumentů za použití sady nástrojů pro analýzu. Analogicky k dolování dat, při dolování textu se snažíme získat užitečné informace z datových zdrojů pomocí zkoumání a identifikace známých vzorů v nestrukturovaných datech.[5]

2.2 Rozdělení metod

Metody pro rozpoznávání hlavního textu lze rozdělit několika způsoby. Kohlschütter et al. [6] dělí metody na ty, které analyzují dokument na základě vizuálních vlastností dokumentu a metody, které provádějí analýzu na úrovni DOM.

Dále je možno rozdělit metody podle toho, jakým způsobem funguje daný algoritmus, a to buď algoritmy fungující na principu heuristik bez strojového učení a algoritmy, které využívají strojové učení.

2.2.1 Principy bez strojového učení

Tyto algoritmy pracují na základě většinou jednoduché heuristiky, popřípadě několika heuristik. Pro správnou funkci nevyžadují žádné vstupní znalosti ani trénovací množinu. Z valné většiny je výsledek zpracování pomocí heuristiky porovnán vůči určitému prahu a na základě výsledku je rozhodnuto o tom, do které množiny daný blok náleží.

Ačkoliv tyto metody patří k jednodušším metodám, při správném nastavení a po řádném otestování můžou dosahovat velmi dobrých výsledků. Díky jednoduchosti je lze řadit většinou také mezi velmi efektivní řešení a časově nenáročné. Jejich hlavní nevýhodou je zaměření na určitou oblast, nepřesné výsledky v případě nestandardních dokumentů a složité, často nemožné změny v případě, že se chceme zaměřit na jinou oblast. Metody, patřící do této skupiny budou popsány níže.

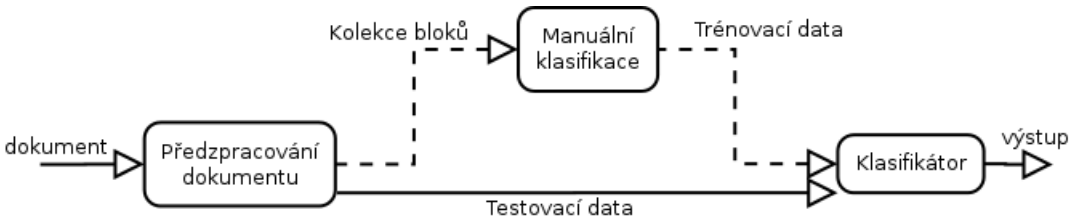
2.2.2 Principy využívající strojové učení

Metody spadající do této kategorie vyžadují určité vstupní znalosti resp. trénovací množinu pro správnou funkcionalitu. Na rozdíl od metod z předchozí kapitoly, které nevyžadují před použitím žádné další nastavení ani testování, u těchto metod je to nezbytné.

Pro hlavní analýzu je v tomto případě využito různých klasifikátorů. Takovýmto klasifikátorem může být Bayesovský klasifikátor, neuronová síť a další. Před použitím musí být klasifikátor správně nastaven, čehož je docíleno důkladným testováním a experimentováním s jednotlivými nastaveními. Tento postup je často velmi zdlouhavý a náročný na zdroje, což tyto metody řadí k metodám složitějším. Aby mohly být provedeny experimenty, je potřeba mít k dispozici relevantní a dostatečně obsáhlou množinu vstupních resp. trénovacích dat.

Výhodou těchto algoritmů je, že pouhou změnou nastavení klasifikátoru, popř. změnou trénovací množiny, můžeme pokrýt jinou oblast. Tyto metody si často dokáží velmi dobře poradit i s nestandardními a špatně strukturovanými dokumenty. Hlavní nevýhodou je jejich složitost a nutnost získaní trénovací množiny dat a následné trénování klasifikátoru.

Na obrázku níže (2.1) je zobrazen princip fungování metod založených na strojovém učení. Přerušované čáry charakterizují část, kdy je prováděno strojové učení, plné čáry pak zobrazují průběh, kdy je klasifikátor používán již v praxi pro analýzu dokumentu.



Obrázek 2.1: Metody se strojovým učením

2.3 Metody založené na vizuálních vlastnostech dokumentu

Metody založené na vizuálních vlastnostech dokumentu většinou dělí dokument do menších logických celků na základě různých mezer a oddělovačů a jednotlivé celky poté analyzují.

Výhodou těchto metod je, že při použití relativně jednoduchého algoritmu je možno získat kvalitní a dostatečně přesné výsledky. Hlavní nevýhoda spočívá v množství informací, které je potřeba znát. Kromě HTML musí být k dispozici rovněž informace definující vzhled stránek, například CSS (kaskádové styly). Tyto informace značně navyšují objem dat nutný k analýze. Aby stránka mohla být analyzována na základě vzhledu, je většinou nutné dokument, včetně připojených stylů, interpretovat. Tato operace je často dosti výpočetně náročná a pomalá.

Nejpalčivějším problémem u algoritmů tohoto typu je, jak danou stránku pohodlně a v rámci možností efektivně renderovat. K tomuto účelu existují různé nástroje. Jedním z nich je například CSSBox[1] vyvíjený na Fakultě informačních technologií VUT v Brně[2]. Program je napsán v jazyce Java a poskytuje přehledné a jednoduché aplikační rozhraní

[1] http://cssbox.sourceforge.net/
[2] http://www.fit.vutbr.cz/

pro vykreslování a práci s webovými dokumenty. Existují i další nástroje, např. Selenium[3] a další.

2.3.1 Metody bez strojového učení

Cai et al. [3] rozděluje webový dokument na jakési základní objekty. Jeden nebo více objektů pak tvoří logické celky - bloky. Uzly v modelu založeném na vizuálních vlastnostech nemusejí nutně odpovídat uzlům v objektovém modelu dokumentu. To je způsobeno tím, že sémantika dokumentu není popsána pouze pomocí jazyka HTML, ale rovněž pomocí kaskádových stylů. Cai et al. [3] popisuje webový dokument jako blok, skládající se z konečné množiny nepřekrývajících se podřízených bloků, které mají opět vlastnosti rodičovského bloku. Tyto bloky jsou odděleny konečnou množinou jak vertikálních tak horizontálních oddělovačů. Každý blok má navíc informaci o vztazích mezi každými dvěma bloky, ze kterých se skládá. Každému oddělovači je rovněž přiřazena váha. Algoritmus zkoumá oddělovače od nejmenší váhy po největší a bloky mezi těmito oddělovači spojuje a tvoří bloky nové. Takto pokračuje dokud nenarazí na oddělovač s nejvyšší váhou. Při vytvoření nového bloku je kontrolována správná zrnitost. Pokud blok neodpovídá požadavkům, vrací se algoritmus o krok zpět aby vytvořil vnitřní obsah bloku. Výstupem algoritmu je dokument rozdělený podle jeho vizuálních vlastností na jednotlivé bloky.

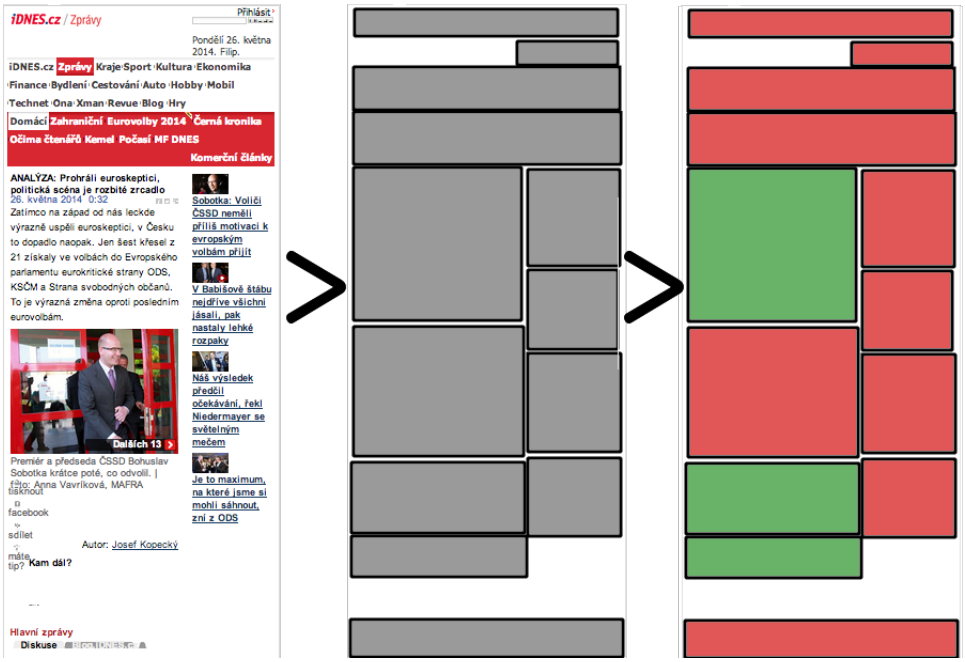
Burget [2] ve své práci popisuje algoritmus, který pracuje ve 4 krocích. Nejprve vykreslí dokument, který je poté rozdělen do bloků. V této klíčové fázi jsou určeny logické segmenty, které mohou být uvažovány jako samostatné. Následně je určeno finální pořadí jednotlivých celků a je generován jednoduchý HTML kód. Pro vykreslení stránky je využíván již zmiňovaný nástroj CSSBox [**?**]. Výstupem první fáze je strom bloků, kdy je blokem rozuměn obdélníkový útvar, který není skrytý. Ve fázi segmentace jsou nalézány logické celky. Samotný proces sestává ze 3 fází. Detekce jednotvárných oblastí, kdy oblast tvoří sousedící bloky s konzistentním stylem. Další fázi je detekce nadpisů na základě vlastnosti fontu. Poslední fázi tohoto kroku je detekce logických celků, které se skládají z více bloků. Jako výchozí bod se využívají detekované nadpisy. Výsledkem tohoto kroku je strom vizuálních oblastí a informací o nich. Na rozdíl od HTML kódu je tato reprezentace mnohem bližší lidskému vnímaní dokumentu. V kroku následujícím se algoritmus snaží přesunout důležité oblasti k začátků dokumentu. Pořadí zbývajících části se snaží zachovat, aby nebyla narušena konzistence obsahu. Na konec je vytvořen jednoduchý HTML kód skládající se v podstatě pouze z elementů `<div>` pro každý logický celek a pro textové bloky, kterým je rovněž přiřazen odpovídající styl. Experimentální výsledky ukazují, že algoritmus funguje spolehlivě pro většinu testovaných stránek.

2.3.2 Metody se strojovým učením

Kunc a Burget [7] ve svém příspěvku popisují způsob klasifikace dokumentů na základě jejich vizuálních rysů. V prvním kroku provádí rekonstrukci výsledné podoby dokumentu. Tento krok vyžaduje kompletní renderování stránky, které může být dosti náročné. Poté jsou pro každou oblast ze získané hierarchické struktury určeny jejich významné vizuální rysy a na základě těch jsou oblasti spojovány do větších celků. Nakonec jsou pro každý celek vypočítány různé vizuální atributy a na základě těch je provedena analýza. Zmiňovanými vlastnostmi jsou průměrná velikost písma, převažující váha písma, převažující styl písma (skloněné nebo normální), počet oblastí v okolí, počet znaků textu, počet číslic, malých

[3] http://docs.seleniumhq.org/

a velkých písmen abecedy a mezer v textu, světelnost textu a pozadí, a nakonec kontrast. Na základě vypočtených atributů je pak možné detekovat například text článku, protože jak je v práci popsáno, tento text je většinou větší než je průměrná velikost textu v dokumentu a obsahuje delší, souvislý text. Pro klasifikaci byl využit volně dostupný nástroj Weka. i když tato práce nesouvisí přímo s extrakcí hlavního textu z dokumentu, lze většinu poznatků a popisovaný postup využít i při řešení tohoto problému.



Obrázek 2.2: Postup při detekci hlavního obsahu na základě vizuálních vlastností

2.4 Metody pracující na úrovní DOM

Tyto metody vyžadují pro analýzu pouze zdrojový kód dokumentu ve formátu HTML nebo XML resp. jejich objektový model dokumentu. Další informace o stránce, jako jsou kaskádové styly nebo soubory se skripty, nejsou vyžadovány. Dokument je na základě různých vlastností textu, struktury a dalších ukazatelů analyzován a na základě těchto ukazatelů je detekován hlavní obsah. Existuje více způsobů jak tyto metody využít v praxi a jak je implementovat.

Před několika lety, byla extrakce textu poměrně jednoduchá. Webové dokumenty nebyly příliš komplexní a struktura stránky byla tvořena většinou pomocí tabulky. Například McKeown et al. [8] popisuje způsob jakým získávají hlavní obsah pár slovy. Pokud vybraná buňka tabulky obsahuje po odstranění značek a odkazů více znaků než je nastavený práh, je toto považováno za hlavní obsah.

V dnešní době je však situace jiná. Struktura HTML je složitější, tabulkové rozvržení je zřídkakdy používáno a nežádoucí prvky, jako je například reklama, se často vyskytují i mezi textem.

2.4.1 Metody bez strojového učení

Kohlschütter et al. [6] představuje postup pro detekci generického obsahu, tudíž rozdělení dokumentu na generický obsah a hlavní text, založený na základních vlastnostech textu. Nejprve je dokument rozdělen do atomických bloků a poté jsou pro každý blok zjištěny jeho vlastnosti. Během dělení dokumentu na atomické bloky, je obsah některých značek přímo odstraňován, jelikož je dopředu jasné, že tyto značky neobsahují hlavní obsah. Jediné značky, které jsou v blocích ponechány jsou odkazy. Poté jsou bloky analyzovány především podle dvou vlastností, počtu slov a hustoty odkazů. Výsledky ukazují, že i pomocí takto jednoduchého řešení lze dosahovat velmi přesných výsledků. Algoritmus implementovaný na základě této studie nese název boilerpipe. V základním extraktoru je pro analýzu využit rozhodovací strom, založený ve výsledku pouze na hustotě odkazů a počtu slov, jak pro aktuální blok, tak pro předchozí a následující. i přesto, že při implementaci bylo využito nástroje Weka, řadím tento algoritmus mezi metody bez strojového učení, jelikož ve finále je použit hotový rozhodovací strom. Nástroj Weka byl použit pouze pro zjištění ideálních prahů.

Vieira et al. [13] popisuje postup lišící se od předchozích. Detekci a odstraňování generického obsahu provádí na základě znalosti více podobných stránek. i když se na první pohled může zdát, že toto řešení je neefektivní, kvůli nutnosti stahování více stránek, není tomu tak. Alespoň ne v případě, kdy je analyzováno více dokumentů z jedné domény. V tomto případě je nevýhoda nutnosti více stránek eliminována, jelikož bychom je museli tak či onak získat. z výsledku je vidět, že tato metoda dosahuje velmi uspokojivých výsledků a je aplikovatelná na téměř jakýkoliv dokument, nezávisle na oboru. Nutností je však mít alespoň dva dokumenty se stejnou nebo alespoň podobnou strukturou.

Pomikálek a jeho jusText [10] využívá jednoduché heuristiky pro detekci. Vychází z předpokladů, že krátké bloky obsahující odkaz jsou téměř vždy generickým obsahem a totéž tvrdí o blocích obsahujících velké množství odkazů. Dále tvrdí, že dlouhé bloky obsahující gramatický text jsou většinou hlavním obsahem, a naopak jiné dlouhé bloky jsou téměř vždy generickým obsahem. Nakonec ještě zavádí tvrzení, které říká, že bloky obsahující hlavní text většinou tvoří shluky, což znamená, že blok s generickým obsahem je často obklopen jinými bloky s generickým obsahem a analogicky pro hlavní obsah. Jeho algoritmus se skládá ze 3 hlavních části. Postupně jsou to preprocessing, bezkontextová analýza a kontextová analýza. V první části zbavuje dokument zbytečných značek. Během bezkontextové analýzy rozděluje jednotlivé bloky do následujících tříd:

- špatné - generický obsah
- dobré - hlavní obsah
- krátké - nelze rozhodnout, příliš krátký blok
- téměř dobré - na pomezí mezi dobrými a krátkymi

Toto rozdělení je prováděno pomocí vyhodnocení tří vlastností, a to hustoty odkazů, počtu stopslov a počtu slov v bloku. V poslední fázi je za pomocí kontextové analýzy rozhodnuto zda jednotlivé krátké a téměř dobré bloky jsou dobré popř. špatné. Téměř dobré bloky sousedící s alespoň jedné strany s dobrým blokem, je vyhodnocen taktéž jako dobrý. Krátké bloky sousedící z obou stran s dobrými nebo téměř dobrými jsou označeny jako dobré, zbývající jako špatné. Je důležité uvést, že jako sousedící bloky v této analýze ignorují bloky krátké. Zároveň pokud se posloupnost krátkých nebo téměř dobrých bloků objevuje na začátku nebo na konci dokumentu, jsou tyto bloky ihned označeny jako špatné.

Evert [4] popisuje svůj algoritmus pod jménem NCleaner. NCleaner má dvě základní úrovně zpracování. První fáze má za úkol připravit dokument na další analýzu. Jsou během ní pomocí regulárních výrazů odstraňovány nežádoucí značky včetně jejich obsahu (obrázky, komentáře a skripty), poté je dokument zkonvertován na čistě textový. Dále jsou odmazány lehce detekovatelné prvky generického obsahu, např. ty, které obsahují mnoho znaků `|`. Druhou fází je samotné jádro algoritmu, sestávající z dvou samostatných znakových n-gram [14] jazykových modelů, jeden pro "správný"a druhý pro "špatný"text. Tyto modely jsou použity na každou část dokumentu a pokud model pro špatný text vypočítá větší pravděpodobnost, je tato část považována za generický obsah a tudíž odstraněna.

Z existujících nástrojů ještě zmíním Readability[4]. Tento nástroj byl ze začátků pouze jednoduchým algoritmem naprogramovaným v Javascriptu, postupně byl však přetvořen na kompletní platformu pro extrakci hlavního textu článku a usnadnění čtení. K extrakci využívá jednoduchých heuristik, kdy ohodnocuje bloky pozitivně či negativně na základě různých textových vlastností a vlastností na úrovní značek či jejich atributů. Výsledné skóre jednotlivých bloků jsou porovnány vůči určitému prahu a na základě výsledku je rozhodnuto zda blok vyhovuje hlavnímu textu či ne.

2.4.2 Metody se strojovým učením

Pasternack et al. [9] popisuje metodu zaměřenou na dokumenty se články, která je schopna se vypořádat i s dokumenty, které nejsou tvořené pomocí tabulek. Problém rozděluje do dvou části. Detekce zda stránka obsahuje článek a zjištění kde přesně článek začíná a kde končí. V části druhé navrhuje odstranit ze článku reklamy a jiné nežádoucí prvky. Pro tento úkon popisuje jednoduchou heuristiku, která nejdříve odstraní obsah uvnitř všech značek `<iframe>` a `<table>` a poté odstraní obsah všech značek `<div>`, obsahujících značky odkaz (`<a>`), `<iframe>`, `<table>`, `<img>`, `<embed>`, `<applet>` nebo `<object>`. Při implementaci byl použit naivní bayesovský klasifikátor [15] pracující na základě pouze dvou jevů: trigramy a nejbližší neuzavřená značka. Vstupem pro klasifikátor jsou pouze dvě základní vlastnosti, a to trigramy a poslední neuzavřený element. Byly zkoušeny rovněž další vlastnosti jako bigramy, unigramy a poslední dva neuzavřené elementy, nicméně experimentálně bylo zjištěno, že na přesnost algoritmu to nemá žádný vliv. Pro strojové učení byl zvoleno učení s učitelem a kombinované učení, přičemž druhé zmiňované podávalo lepší výsledky. Kombinované učení spočívalo v naučení na trénovací množině, predikce extrakce z neanotovaných dokumentů a iterace, kdy byly nalezeny důležité váhy v množině vybraných dokumentů, natrénovaní nového Bayessovského klasifikátoru pomocí trigramů a predikce nové extrakce pro další dokumenty. Výsledky experimentů jasně dokazují velice dobrou přesnost algoritmu.

Schäfer [11] ve své práci popisuje algoritmus využívající MLP (Multilayer Perceptrons) což je vícevrstvá umělá neuronová síť. Pro klasifikaci využívá mnoho různých vlastností. Vlastnosti spojené se značkami jako jsou např. poměr značek v bloku a typ obalující značky (`<article>`, `<div>` apod.). Vlastnosti související s odkazy (počet emailových adres v bloku), délka textu a jeho pozice v rámci dokumentu. Dále využívá vlastnosti na úrovní jednotlivých znaků. Hodnotí poměr interpunkčních znamének, velkých písmen, čísel, znaků abecedy a konečně výskyt copyright znaku. Poslední dvě vlastnosti, které využívá ke klasifikaci, jsou lingvistické (průměrná délka věty, jejich počet a poslední znak věty) a vlastnosti týkající se celého dokumentu (typ dokumentu - `doctype` a proporce značek). Všechny tyto vlastnosti jsou vyjádřeny jako normalizované skóre v rozmezí od -1 resp. 0 po 1. Toto je

[4] https://www.readability.com/

rozmezí hodnot nutné pro vstup MLP. Jako trénovací algoritmus byl v tomto případě použit algoritmus RPROP. Trénovací množina sestávala z jednotlivých bloků, kdy špatné bloky byly označeny hodnotou $-0,1$ a správné hodnotou 1. Experimentální výsledky ukazují, že algoritmus funguje velmi dobře.

Spousta et al. [12] prezentuje ve své práci přístup založený na CRF (Conditional random field). Je to statistická metoda často využívaná při rozpoznávaní vzorů a strojovém učení, a výstupem je strukturovaná predikce. Samotný proces extrakce sestává z několika kroků. Nejdříve jsou dokumenty filtrovány na základě jednoduchého ngram klasifikátoru, a nevyhovující jsou zahozeny. Poté probíhá standardizace dokumentů a výstupem jsou validní dokumenty, které jsou následně vyčištěny od nepotřebných značek (skripty, styly, objekty atd.). Následně probíhá identifikace textových bloků, pro které jsou v následujícím kroku získány jejich vlastnosti. Předposledním krokem je učení klasifikátoru s učitelem, kdy trénovací množina obsahuje bloky manuálně označené jako hlavička, text a ostatní. Po naučení klasifikátoru je již možno extrahovat hlavní text, kdy jsou v dokumentu ponechány pouze bloky označené jako hlavička nebo text a ostatní jsou zahozeny. Vlastnostmi, na kterých je klasifikace založena, jsou vlastnosti založené na značkách (typ bloku), vlastnosti založené na obsahu (absolutní a relativní počty slov, znaků, duplikátů apod.) a vlastnosti založené na celém dokumentu (pozice, počet slov, vět apod.).

2.5 Umělá neuronová síť

Umělá neuronová síť je systém, založený na fungování biologických neuronových sítí. [1] Síť tvoří vzájemně propojené neurony tak, že výstup neuronu je vstupem neuronů jiných. Každá síť musí obsahovat vstupní vrstvu, výstupní a libovolný počet vrstev skrytých.

Pomocí neuronových sítí lze řešit mnoho různých problémů, zejména pak ty, které by byly řešitelné velmi obtížně pomocí klasického lineárního programu, popřípadě by nebyly řešitelné vůbec. Mezi takovéto problémy patří například rozpoznávaní obrazu nebo řeči. Další výhodou je, že v případě selhání neuronu, síť může bez problému pokračovat díky její paralelní povaze. Umělá neuronová síť je poměrně jednoduše implementovatelná a může být využita ve většině aplikací. Mezi hlavní nevýhody patří nutnost neuronovou síť naučit. Jednak je třeba mít k dispozici vhodnou a poměrně velkou množinu trénovacích dat a jednak je učení výpočetně náročné.

Existují dva základní modely pro učení umělé neuronové sítě: učení s učitelem a učení bez učitele. V prvním případě je síti předložen vstup i požadovaný výstup. Síť vždy vyhodnotí vstup a svůj výstup porovná s požadovaným. Pokud se výstup liší, je provedena korekce vah popř. prahů neuronů a proces je opakován, dokud nedosáhneme námi stanovené minimální chyby. V případě učení bez učitele není předem znám výstup. Rozdělení vzorů do skupin je čistě v režii neuronové sítě.

Každý jednotlivý neuron v síti obsahuje aktivační funkci. Tyto funkce mohou být binární nebo spojité, v závislosti na povaze neuronové sítě. Aktivační funkce určují zda neuron bude na základě vstupu aktivován nebo zůstane v pasivním módu. Nastavení správných aktivačních funkcí je stěžejní pro získání přesných výsledků.

2.5.1 Vícevrstvá dopředná neuronová síť

Vícevrstvé dopředné sítě odstraňují oproti jednovrstvým dopředným neuronovým sítím (perceptronům) některá omezení. Pro učení takovýchto síti se většinou používá metoda

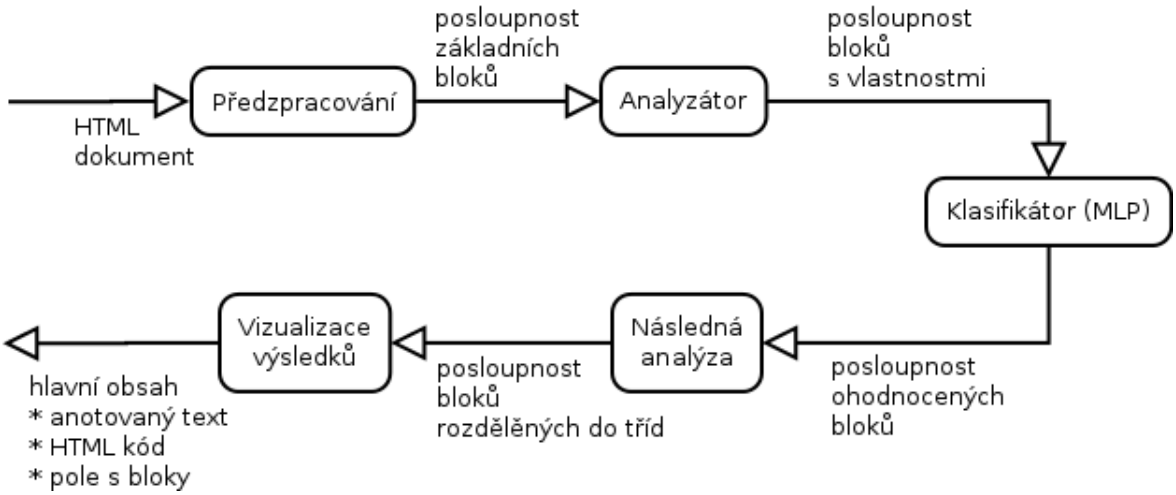
zpětného šíření (backpropagation) a její variace nebo také algoritmus Rprop (resilient backpropagation). U obou těchto algoritmů se jedná o učení s učitelem. Metoda zpětného šíření funguje ve třech fázích, šíření signálu směrem dopředu, zpětné šíření chyby a v poslední fázi jsou aktualizovány váhy neuronů.

Vícevrstvý perceptron je možné použít v mnoha oblastech a dosahuje velmi dobrých výsledků, v případě, že je použita kvalitní množina trénovacích dat. z předchozího tedy vyplývá, že je vhodné jej použít i při extrakci hlavního textu z webových dokumentů.

Kapitola 3

Návrh algoritmu

V této práci se zabývám extrakci hlavního textu z webových dokumentů založené na textových rysech. Pro tento přístup jsem se rozhodl především proto, že algoritmy založené na této metodě dosahují velmi dobrých výsledků při zachování efektivity i při malé náročnosti na výpočetní výkon. Na následujících řádcích bude postupně popsán návrh tohoto algoritmu, od předzpracování původního dokumentu, přes následnou analýzu až po samotnou extrakci hlavního textu a vizualizaci výsledků. Jádro programu, tedy samotná analýza, bude využívat umělou vícevrstvou neuronovou síť. Na obrázku 3.1 je zobrazen návrh programu a rozdělení na jednotlivé části a vstupy resp. výstupy jednotlivých částí.



Obrázek 3.1: Návrh programu - jednotlivé části

3.1 Předzpracování dokumentu

Aby mohl být webový dokument potažmo HTML stránka dále analyzován a mohlo dojít k extrakci hlavního textu, je třeba jej nejdříve dostat do podoby, se kterou bude moci algoritmus dále pracovat. Pro tento účel bude sloužit jakýsi preprocesor, jehož vstupem bude webový dokument ve formátu HTML a výstupem bude posloupnost objektů, které budeme nazývat základními bloky.

V první fázi preprocesor dostane na vstupu HTML stránku, kterou pomocí některého z dostupných syntaktických analyzátorů jazyka HTML převede na DOM. Poté z dokumentu odstraní značky, o kterých víme, že určitě nemohou obsahovat hlavní obsah. Nejdříve bude tedy odstraněna celá značka `<head>`, přičemž budou uložené některé užitečné informace, které tato značka může obsahovat. Takovouto informaci může obsahovat například `<title>`, kterou můžeme využít při následné analýze. Následně se odstraní značky jako jsou `<script>, <style>, <object>` a další s podobným sémantickým významem.

Posledním úkonem v této části je již poměrně zredukovaný DOM převést do vnitřní reprezentace, tedy posloupnosti základních bloků, ve stejném pořadí, jako se nalézají v DOM. Během tohoto převodu budou ještě z DOM odstraněny veškeré řádkové značky (např. `<span>`, `<strong>`) a budou nahrazeny pouze jejich obsahem. Určité značky, jako například odkazy, budou počítány a přiřazovány jako vlastnosti základních bloků a později využity pro analýzu. Základním blokem tedy budou ve výsledku blokové značky obsahující text. Základním blokem je vždy blokový HTML element, který neobsahuje žádné další blokové elementy. Výjimku tvoří elementy, které obsahují jak samotný text, tak blokové značky. V tomto případě bude text obalen do nové značky, stejné jako obalující, a bude z něj vytvořen základní blok v závislosti, kde se text nachází. Blokové elementy neobsahující text budou automaticky ignorovány. Taktéž budou ignorovány elementy obsahující pouze bílé znaky, a posloupnosti více než jednoho bílého znaku budou nahrazeny bílým znakem jedním.

3.2 Analýza dokumentu

Táto část algoritmu bude mít za úkol analýzu jednotlivých bloků a později na základě této analýzy detekovat a extrahovat hlavní obsah dokumentu. Při detekci jsem se rozhodl využít mnoho různých vlastností založených jak na textových rysech, tak i na základě vlastností samotných bloků, jako je například jejich pozice v posloupnosti a vlastnosti předchozího a následujícího bloku.

3.2.1 Textové rysy

Stěžejní roli při rozhodování, zda blok obsahuje či neobsahuje hlavní text, jsou textové rysy. V této fázi analýzy budeme zkoumat některé z nich pro každý základní blok samostatně. Vypočítány budou parametry jako průměrná délka slov, průměrná délka věty, počty vět v bloku, počet slov, počet písmen, počet slov začínajících velkým písmenem a počet slov vysázených pouze verzálkami.

Všechny hodnoty musí být před vstupem do klasifikátoru, tedy neuronové sítě normalizovány a upraveny tak, aby byly v intervalu $< 0, 1 >$. Všechny počty jsou upraveny podle vzorce $vstup = \frac{1}{pocet}$. Tedy podíl počtu dané vlastnosti v bloku vůči maximálnímu nalezenému počtu v dokumentu. U průměrných délek bude situace podobná a bude se počítat analogicky.

3.2.2 Speciální znaky

Jako další ukazatel při klasifikaci bude počet výskytu různých speciálních znaků a slov v bloku. Speciální znaky budou v programu implicitně nastaveny. Například blok obsahující znak `'©'` není většinou hlavním obsahem ale patičkou. Navigační prvky stránky zase často obsahují znak `'|'`. Tyto znaky a řada jiných budou v této fázi spočítány a počty uloženy jako atributy bloků.

Vstupem do neuronové sítě je pak pouze příznak tedy hodnota 0 resp. 1 což odpovídá významu obsahuje resp. neobsahuje daný speciální znak.

3.2.3 Hustotní rysy

V této fázi jsou již známy základní ukazatelé a počty a z nich budou vypočítány další. Jednou z důležitých vlastností je hustota odkazů vůči počtu slov popřípadě hustota vět obsahujících odkaz oproti větám bez odkazu. Další ukazatele založené na hustotě, které budou počítány je poměr slov vysázených verzálkami oproti slovům obsahující malé písmena, hustota speciálních znaků, hustota slov obsahující více znaků nežli je průměrná délka slov v bloku. Konečně bude vypočítána i hustota některých značek v bloku vůči textu.

Jelikož hustota je počítána podle vzorce $vstup = \frac{pocet\_vyskytu\_vlastnosti}{celkovy\_pocet}$, tedy podíl počtu počítané vlastnosti (např. odkazů) vůči celkovému počtu (např. slov), není nutno hodnoty dále upravovat, protože výsledkem je vždy hodnota z požadovaného intervalu. V případě, že se ve jmenovateli objeví 0, je výsledkem 1.

3.2.4 Vnořené řádkové elementy

Při analýze budeme brát ohled také na to, jaké řádkové elementy základní blok obsahoval. Například pozitivním způsobem budeme hodnotit bloky se značkami jako jsou `<cite>`, `<code>` a další.

Hodnoty pro vstup do klasifikátoru jsou v tomto případě normalizovány stejně jako počty u textových rysů.

3.2.5 Pozice v dokumentu

Při zkoumání webových dokumentů jsem zjistil, že ve většině případů jsou webové dokumenty strukturované podobným způsobem. Na začátku se nalézá hlavička následována hlavním obsahem a různými dalšími bloky a na konci se nalézá patička. Proto budou základní bloky na okrajích posloupnosti hodnoceny negativně, kdežto naopak bloky blíže středu pozitivně.

Na vstup neuronové sítě je v tomto případě přivedena relativní pozice v dokumentu, která je vypočítána podle vzorce $pozice_{relativni} = \frac{index\_bloku}{celkovy\_pocet\_bloku}$

3.2.6 Atributy značek

Při důkladnějším pohledu na HTML stránky nalézající se na internetu jsem zjistil, že často lze části stránky rozpoznat na základě hodnot atributů značek. Například velice oblíbeným pravidlem je označovat hlavičku, obsah a patičku pomocí atributů `id` nebo `class` s hodnotami `header`, `content`, `footer`. Tyto a další ukazatele založené na podobném principu budou v navrhovaném algoritmu využity.

V tomto případě jsou opět na vstup klasifikátoru přivedeno pouze příznaky, tedy hodnoty 0 nebo 1, zda blok nebo některý z jeho rodičů obsahuje pozitivně hodnocenou třídu.

3.3 Klasifikace bloků

V předchozích odstavcích byla popsána většina vlastností, na základě kterých budeme provádět v této části klasifikaci jednotlivých bloků.

Pro samotnou klasifikaci jednotlivých bloků bude využita umělá dopředná vícevrstvá neuronová síť. Síť bude mít vstupní vrstvu, výstupní vrstvu s jedním neuronem a tři skryté vrstvy. Na vstup budou přivedeny hodnoty vlastností vypočítaných v předchozích krocích, nejdříve však budou tyto hodnoty normalizovány. Na vstupu budou rovněž vlastnosti předchozího a následujícího bloku. Počet neuronů v jednotlivých skrytých vrstvách nelze dopředu určit, a proto budou správné počty určeny na základě důkladného testování. Rovněž aktivační funkce neuronů ve vrstvách a strmost těchto funkcí bude třeba určit až při implementaci. Jako nejlepší učící algoritmus se jeví v tomto případě metoda pružného zpětného šíření chyby (resilient backpropagation, Rprop). Budou však odzkoušeny i alternativní učící algoritmy.

Kritickou úlohou bude správné nastavení hraničních hodnot. Výstupem neuronové sítě bude číslo v rozmezí $< -1, 1 >$ a je třeba nastavit práh, kdy se má s blokem zacházet jako s blokem obsahujícím hlavní text, blokem obsahujícím šablonu a kdy bude blok považován za téměř správný. Tyto prahy budou v programu nastaveny implicitně, pravděpodobně ale poskytnu možnost je explicitně přenastavit.

3.4 Následná analýza

Tato část programu bude mít za úkol částečně eliminovat chybovost neuronové sítě. Jelikož se webové dokumenty většinou dosti liší, může dojít při klasifikaci i k značné chybovosti. Většinou se jedná o špatné vyhodnocení hlavního nadpisu, blok z dlouhým textem v patičce může být označen jako hlavní text popř. krátký text v rámci hlavního obsahu může být označen za nevyhovující. Většinu těchto nedostatků se budu snažit v této fázi eliminovat pomocí různých heuristik. Tuto část bude možno explicitně zakázat.

3.4.1 Společné nadřazené značky

Při klasifikaci bude rovněž brán ohled na nadřazené elementy. Dá se totiž předpokládat, že hlavní obsah bude oddělen od generického obsahu, tzn. bude obalen stejnou značkou. Tato značka však může být několik úrovní nad základním blokem. Proto bude nutné porovnávat u základních bloků cesty k nadřazeným elementům a tyto porovnávat. Každý základní blok bude mít ve výsledku uloženou informaci o tom, se kterými jinými bloky má společné nadřazené elementy. Zjistil jsem, že tento ukazatel ve značné míře přispívá k zpřesnění klasifikace u těžko rozhodnutelných bloků.

3.4.2 Vlastnosti sousedních bloků

Za předpokladu, že hlavní obsah tvoří souvislý text, lze u těžko rozhodnutelných bloků hodnotit i vlastnosti jejich sousedů, a to následovným způsobem. Blok je považován za hlavní obsah pokud se v jeho blízkém okolí nalézají z obou stran bloky s hlavním obsahem nebo se blok nalézá blízko začátku a následující blok je hlavním obsah nebo analogicky pro blok nalézající se ke konci dokumentu. V opačném případě je blok považován spíše za generický obsah. Zároveň bude zacházeno poněkud rozdílným způsobem s různými typy bloků. Například pro nadpis budou pravidla jiná než je tomu například u položky seznamu.

3.4.3 Detekce začátku hlavního obsahu

Důležitý ukazatel, který může značně přispět ke značnému zvýšení přesnosti algoritmu, je detekce, kde hlavní obsah začíná. K tomu nám může pomoci obsah elementu `<title>` v

<head>, protože bývá dobrým zvykem, že tato značka obsahuje alespoň jako podřetězec nadpis hlavního obsahu. Pokud předpokládáme, že nadpis je obsažen ve značce <h1> popř. <h2>, pak lze poměrně jednoduchým postupem nalézt začátek hlavního textu. V případě, že takovýto nadpis byl nalezen, budou bloky nad nadpisem označeny za generický obsah.

3.5 Rozhraní programu

Program bude vytvořen jako knihovna, s rozhraním pro příkazový řádek a bude rovněž obsahovat jednoduché webové rozhraní.

3.5.1 Výstup programu

Výstupem programu bude kolekce jednotlivých základních bloků. Pomocí metod, které bude mít blok implementován, bude možno přistoupit k vypočítaným vlastnostem, zjistit zda je blok hlavním obsahem či generickým obsahem atd.

3.5.2 Knihovna

Aby byla zaručena univerzálnost, jednoduchá instalace a možná integrace do jiných systému, bude program distribuován jako knihovna. Knihovnu bude pak možno použít v jakémkoliv systému napsaném ve stejném jazyce a dále pracovat s výstupem.

3.5.3 Webové rozhraní a vizualizace výsledků

Webové rozhraní bude jednoduché a uživatelsky přívětivé. Bude obsahovat pouze základní funkčnost, zadání URL stránky, ze které se má extrahovat text, výpis všech bloků, výpis bloků pouze z hlavním obsahem a možnost zakázat či povolit následnou analýzu. Ke každému bloku bude kromě textu zobrazeno jeho výsledné hodnocení a typ elementu.

3.5.4 Příkazový řádek

Program bude možno spustit rovněž přímo z příkazové řádky, kdy na základě URL popř. cesty k souboru bude extrahován hlavní text z dokumentu a následně zapsán do zadaného souboru nebo na standardní výstup.

Kapitola 4

Implementace

V této kapitole popisuji samotnou implementaci systému. V samotném úvodu kapitoly jsou shrnuty ve zkratce technologie používané pro tvorbu webových dokumentů, implementační jazyk a použité gemy (knihovny).

4.1 Značkovací jazyky pro tvorbu webových dokumentů

Pro tvorbu webových dokumentů se ve většině případů využívá značkovacího jazyka HTML (HyperText Markup Language). V současné době převažují dokumenty vytvořené za pomocí HTML ve verzi 4.01[1] nebo v HTML5[2]. Vytváření standardů pro HTML má na starosti konsorcium W3C[3]. i když je HTML5 v současnosti stále ve stádiu návrhu a schválen by měl být až příštím rokem, je již naprosto běžně využíván. Oproti předchozím verzím již není závislý na SGML (Standard Generalized Markup Language), přidává nové značky a některé zastaralé odstraňuje.

Pro vytváření webových dokumentů lze kromě jazyka HTML využít také XML (Extensible Markup Language), což je obecný značkovací jazyk a vychází z jazyka SGML. Stejně jako HTML je standardizován a spravován konsorciem W3C[4].

Jelikož se zdrojovým kódem ve značkovacích jazycích není možné pracovat přímo, je třeba je nejdříve interpretovat a tím získat reprezentaci dokumentu v DOM (Document Object Model, objektový model dokumentu). Dokument v této struktuře nám pak umožňuje jeho modifikaci, čtení a prohledávání.

4.2 Programovací jazyk Ruby

Ruby je interpretovaný skriptovací programovací jazyk. Díky své jednoduché syntaxi je poměrně snadný k naučení, přesto však dostatečně výkonný, aby dokázal konkurovat známějším jazykům jako je Python a Perl [16].

Ruby[5] je plně objektovým jazykem, což znamená, že vše je objekt. Existuje několik různých interpretů Ruby, lišících se zejména v jazyce, ve kterém jsou implementovány a zda obsahují nebo neobsahují GIL (Global Interpreter Lock, globální zámek interpretu).

[1] http://www.w3.org/TR/REC-html40/
[2] http://www.w3.org/TR/html5/
[3] http://www.w3.org/
[4] http://www.w3.org/
[5] https://www.ruby-lang.org/

GIL zaručuje, že současně může vykonávat práci pouze jedno vlákno. Původní, a dnes pravděpodobně nejrozšířenější interpret jazyka Ruby pod názvem MRI (Matz's Ruby Interpreter) GIL obsahuje.

Tento jazyk jsem zvolil pro implementaci systému v rámci této práce zejména pro pohodlný a rychlý vývoj. Rovněž komunita kolem tohoto jazyka je poměrně rozsáhlá a dostupnost různých knihoven je obšírná. Zároveň v ruby podobný systém není zatím implementován nebo není rozšířen. Ruby je stále aktivně vyvíjeno a díky velkému zájmu ze strany vývojářů nic nenaznačuje tomu, že by v blízké budoucnosti tomu mělo být jinak. Tyto důvody předurčují Ruby jako vhodný jazyk pro implementaci tohoto systému.

Knihovny jsou v Ruby distribuovány pomocí tzv. gemů. Instalace probíhá jednoduše pomocí příkazu `gem install NAME`, vše ostatní, od stažení až po instalaci včetně dokumentaci již proběhne automaticky.

Ruby je současně ve verzi `2`, nicméně stále hojně využívaná je rovněž verze `1.9.3`, popř. starší `1.9.2`. Já se zaměřím primárně na vývoj pro verzi `>= 2.0.0`, nebudu však používat konstrukce možné pouze v této verzi, tudíž program by neměl mít problém i s nižší verzí Ruby.

Ruby jako takové neposkytuje jmenné prostory, lze je však nahradit modulem, který neimplementuje žádnou funkcionalitu, pouze obaluje dané třídy nebo moduly. Toto se v praxi hojně využívá a bude rovněž použito v této práci. Výhodou tohoto řešení je eliminace konfliktů mezi názvy tříd při použití gemu v systému.

4.3 Použité gemy

Níže popíšu stěžejní gemy při vývoji programu. Podpůrné gemy zde neuvádím, jelikož se jedná většinou pouze o různá vylepšení nebo závislosti gemu jiných.

4.3.1 Syntaktický analyzátor Nokogiri

Nokogiri[6] je nejpoužívanějším syntaktickým analyzátorem pro XML a HTML v Ruby. Obsahuje přehledné rozhraní, přístup k elementům pomocí různých zápisu jako je XPath nebo CSS notace, je flexibilní a poradí si i s nevalidními či poškozenými dokumenty. Umožňuje veškerou manipulací s DOM, čtení, zápis nebo modifikaci již existujících elementů.

4.3.2 Umělá neuronová síť RubyFann

RubyFann[7] je gem, který umožňuje použít přímo v Ruby volně šířenou knihovnu FANN (Fast Artificial Neural Network)[8], implementovanou v jazyce C. Tato knihovna zvládá jak plně propojené tak pouze částečně propojené neuronové sítě. Knihovna dovoluje poměrně rozsáhlé a podrobné nastavení. Pro učení lze využít čtyř různých variací metody zpětného šíření chyby, mezi jinými podporuje i algoritmus RProp. Aktivační funkci i její strmost lze jednoduše nastavit pro každou vrstvu zvlášť, pro všechny skryté vrstvy najednou a v případě potřeby lze nastavit aktivační funkci pro každý neuron zvlášť. Při učení je potřeba zvolit maximální počet epoch, po kolika epochách se má vypsat na výstup aktuální stav učení a požadovanou chybu. Trénovací data mohou být předané přímo funkci nebo je lze načíst

[6] http://nokogiri.org/
[7] http://ruby-fann.rubyforge.org/RubyFann.html
[8] http://leenissen.dk/fann/wp/

z připraveného souboru. Natrénovanou síť lze jednoduše uložit do souboru pro pozdější použití, kdy je síť načtena přímo ze souboru a není třeba nového učení.

4.3.3 Doménově specifický jazyk Sinatra

Sinatra[9] je interní doménově specifický jazyk napsaný v Ruby, určen pro tvoření malých webových aplikací v Ruby. Umožňuje jednoduchým způsobem definovat cesty URL, a definovat jejich chování. Sinatra je navržená tak, aby zápis cesty a chování byl co nejjednodušší a nejkratší, viz. Zdrojový kód 4.1

```
# app.rb
require "sinatra"

get "/" do
  "Ahoj svete!"
end
```

Zdrojový kód 4.1: Příklad aplikace "Ahoj svete" v Sinatře

4.3.4 Verzovací systém Git

Git[10] je jeden z nejznámějších a nejrozšířenějších distribuovaných systému správy verzí. Při vývoji jej používám pro udržování verzí kódu. Git byl původně vytvořen Linusem Torvaldsem a byl určen pro vývoj jádra Linuxu. Později se však díky své univerzálnosti a dobré použitelnosti masivně rozšířil a dnes jej používá mnoho dalších velkých i malých projektů. Jedná se o svobodný software.

4.4 Kostra programu

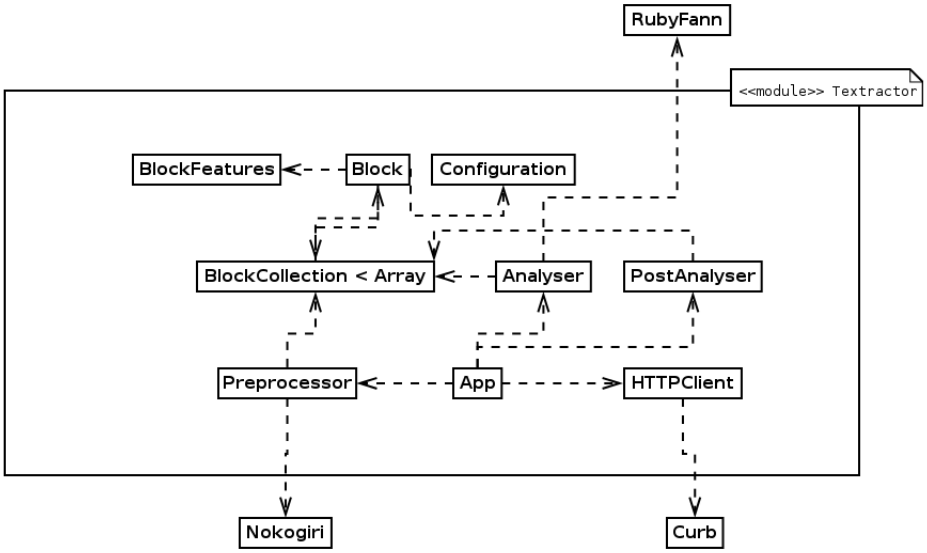
Před samotnou implementací programu jsem si vytvořil základní kostru programu. Kostra obsahuje pouze nezbytné adresáře a soubory. Základem každé moderní aplikace v Ruby je soubor `Gemfile`, ve kterém definujeme gemy, které bude aplikace využívat. Standardní gemy, které obsahuje jazyk Ruby, není třeba uvádět, uvádí se pouze gemy třetích stran. Definice pomocných úloh, které jsou spouštěny přímo z příkazové řádky jsou obsaženy v souboru `Rakefile`. Jelikož jsem již od začátku počítal se šířením programu v podobě gemu, veškerý kód potřebný k fungování aplikace je umístěn v adresáři `lib`. Program jsem nazval `Textractor`, tudíž jsem vytvořil ještě v adresáři `lib` soubor `textractor.rb`, který bude obsahovat potřebné příkazy pro připojení potřebných gemů a také základní modul `Textractor`, který je v tomto kontextu použit jako jmenný prostor, protože klasické jmenné prostory v Ruby neexistují. Po vytvoření kostry jsem udělal inicializační commit do Gitu. Commity jsem prováděl dále vždy, když byla přidána nová funkčnost, popř. opravená chyba.

4.5 Základní modul

Základním modulem aplikace, jak už bylo zmíněno výše je modul `Textractor`. Tento modul obsahuje pouze konstanty, díky kterým budu později rozlišovat zda se jedná o element

[9] http://www.sinatrarb.com
[10] http://git-scm.com

blokový nebo řádkový. Některé ve skutečnosti blokové elementy jsem přeřadil k řádkovým a opačně, zejména kvůli tomu, že se většinou ve skutečnosti používají v jiném kontextu než byly původně navrhnuty. Dále modul obsahuje třídy `HTTPClient`, `Configuration`, `Preprocessor`, `Analyser`, `PostAnalyser`, `Block`, `BlockFeatures`, `BlockCollection` a `App`. Jak jednotlivé třídy mezi sebou interagují je možné vidět na Obrázku 4.1.



Obrázek 4.1: Diagram tříd a vztahů mezi nimi

4.6 Konfigurace a HTTP klient

Třída `Configuration` implementuje načítání konfiguračního souboru z adresáře `config` a poskytuje přístup k položkám konfigurace. Přesto, že v jazyce Ruby neexistují statické třídy jak je známe třeba z C++, tato třída je takto myšlena. Pro přístup ke konfiguraci obsahuje pouze třídní metody, tudíž není potřeba její instanciace. Načtený konfigurační soubor je uložen do třídní proměnné, je tedy načten pouze jednou.

Pro stažení dokumentu z internetu slouží objekt třídy `HTTPClient`, který obaluje rozhraní gemu `curb`, což je gem poskytující vazbu na knihovnu `libcurl`.

4.7 Vstup programu

Vstupem aplikace může být buď soubor nebo URL adresa webového dokumentu, který má být analyzován. Pokud je jako vstup uveden soubor pak je otevřen pro čtení a dále zpracováván, pokud je na vstupu URL, je stránka nejdříve stažena s využitím instance třídy `HTTPClient`. V této chvíli jsou data připravená k analýze a jsou předána třídě `Preprocess` k předzpracování. Vstup programu je předáván během vytváření nové instance třídy `App`.

4.8 Předzpracování dokumentu

Tento proces zastřešuje třída `Preprocess`, která na vstupu obdrží webový dokument ve formátu HTML popř. XML. Výstupem je pak kolekce neboli pole základních bloků připravených k další analýze.

Třída má pouze dvě veřejné metody. Metoda `new` jako parametr vyžaduje webový dokument ve formě řetězce. Metoda druhá, `perform`, již spouští samotné předzpracování a její návratovou hodnotou je požadována kolekce bloků.

Předzpracování probíhá následujícím způsobem. Již při vytvoření instance je webový dokument zpracován pomocí `Nokogiri` a je uložen pouze objektový model dokumentu. Při zavolání metody `perform` je nejdříve získán z dokumentu obsah elementu `title` pomocí metody `get_title` a jeho obsah je uložen pro další použití. Následně metoda `remove_head` z dokumentu odstraní celý element `head` resp. nahradí proměnnou s dokumentem obsahem elementu `body`. Nyní jsou odstraněny zbytečné elementy, konkrétně: komentáře, `script`, `noscript`, `object`, `video`, `style`, `canvas`, `iframe`, `svg`, `embed`. Odstraňování elementů probíhá pomocí metody `remove_tags`, které pomocí parametru předám název elementu, který je třeba odstranit. Následně je na základě takto zredukovaného dokumentu vytvořena kolekce bloků. Rozdělení na bloky má plně ve své režii objekt třídy `BlockCollection`, jejíž chování bude popsáno níže. Poté jsou pomocí metod `remove_blank_blocks!` a `remove_whitespaces!` z kolekce odstraněny prázdné bloky resp. bloky obsahující pouze bílé znaky a posloupnosti dvou a více bílých znaků v blocích jsou nahrazeny bílým znakem jedním. Nakonec každému bloku nastavím jeho pozici v rámci kolekce. Tímto předzpracování končí.

4.9 Kolekce bloků

Kolekce bloků je instancí třídy `BlockCollection`, která dědí od třídy standardní třídy `Array`, a její chování rozšiřuje o další metody a některé z metod redefinuje. Konstruktor vyžaduje na vstupu jeden povinný a jeden volitelný parametr, jmenovitě: předzpracovaný objektový model dokumentu a název (`title`). V rámci instanciace třídy je první z parametrů zkonvertován na pole základních bloků čili jednotlivé instance třídy `Block`.

Konvertování provádí metoda `convert`. Metoda rekurzivně prochází DOMem. Pro každý je pomocí metody `_terminal_node?` zjištěno, zda aktuální element obsahuje nějaké další blokové elementy. V případě negativního vyhodnocení metoda volá sama sebe a zanořuje se dále. V opačném případě je na základě elementu vytvořen nový základní blok, vložen do kolekce a metoda se vynořuje zpět.

Objekt si navíc při vytvoření uloží pro další použití hodnotu parametru `title`.

4.10 Základní blok

Nejrozsáhlejší a stěžejní třídou celého programu je `Block`. Každá instance této třídy obsahuje instanční proměnné pro originální element z objektového modelu dokumentu, vyextrahovaný a upravený text pro potřeby programu, název rodiče, cestu k elementu v rámci dokumentu v XPath reprezentaci, atributy `class` a `id` všech elementů nadřazených, odkaz na celou kolekci bloku aby mohl získat informace o svých sousedech, pozici v rámci kolekce.

Každý objekt obsahuje rovněž podpůrné proměnné a metody pro další zpracování. Do instanční proměnné `nn_score` bude uložen výsledek z klasifikace pomocí neuronové

sítě a proměnná `score` bude mít finální hodnotu po následné analýze. Pro všechny výše zmiňované proměnné jsou dostupné metody pro čtení hodnot a pro některé rovněž zapisovače. Pro analýzu a následnou analýzu jsou rovněž připraveny metody `good?` a `bad?` a analogicky stejné metody s předponou `nn_`. Metody jako `headline?`, `paragraph?` apod. jsou určeny pro zjišťování typu bloku. Metoda `features` slouží pro získaní vlastností bloku a její návratovou hodnotou je instance třídy `BlockFeatures`.

Metody `features_data` a `data_for_neural` slouží k získaní vstupů pro neuronovou síť. První jmenovaná vrací normalizované vlastnosti aktuálního bloku, čili všechny hodnoty jsou v rozmezí $< 0, 1 >$, kdežto druhá metoda vrací i hodnoty pro předchozí a následující blok. Výstupy těchto metod jsou uloženy do instančních proměnných, aby při dalším voláni nemusely být počítány znovu.

4.11 Vlastnosti bloku

Pro získání a kalkulaci vlastností bloku využívám třídu `BlockFeatures`. Samotná třída sestává z několika základních metod pro získaní slov, vět a odkazů. Do třídy jsou však připojeny jednotlivé moduly neboli mixiny, které implementují pomocí zmíněných základních metod další propočty. Moduly se postupně jmenují následovně `Counts`, `Averages`, `Densities`, `NestedElements` a `SpecialChars`. Dle mého názoru je již z jejich názvu jasné jejich určení a není potřeba je dále popisovat.

Tato třída obsahuje navíc instanční metodu `good_parent_class?`, která zkoumá třídy a identifikátory nadřazených elementů, a v případě nalezení vhodné hodnoty vrací `true` jinak `false`. Tato metoda je důležitá během následné analýzy, která bude teprve popsána.

4.12 Implementace analýzy dokumentu

Postup pro hlavní analýzu dokumentu je implementován ve třídě `Analyser`. Konstruktor akceptuje pouze jeden parametr `block_collection`, ve kterém předáme objektu kolekci bloků. Další a poslední veřejnou metodou je `perform`, podobně jako u třídy `Preprocess`. Tato metoda pomocí neuronové sítě vyhodnotí každý jeden blok a nastaví mu `nn_score`.

Před samotným vyhodnocením se kontroluje zda neuronová síť již byla inicializována. V kladném případě se přistupuje ihned ke klasifikaci, naopak pokud síť ještě neexistuje je volána metoda `train_network`, která má za úkol síť vytvořit a uložit do třídní proměnné pro pozdější využití. Po vytvoření sítě je síť uložena do souboru, aby mohla být příště načtena bez učení. Pokud však žádná síť zatím vytvořena nebyla, je třeba vytvořit zcela novou síť a provést učení. Síť je vytvořena i učena prostřednictvím gemu `RubyFann`.

Vstupní vrstva neuronové sítě obsahuje 78 neuronů a výstupní vrstva neuron pouze jeden. Na vstup jsou přivedeny následující hodnoty:

- Počet písmen
- Počet slov
- Počet slov začínajících velkým písmenem
- Počet slov vysázených verzálkami
- Počet vět
- Hustota čísel

- Hustota mezer
- Počet odkazů
- Počet svislých čar
- Počet obrázků
- Počet "pozitivně hodnocených"řádkových elementů (např. `cite`)
- Zda obsahuje speciální znak `'©'`
- Zda obsahuje znaky `'<'` nebo `'>'`
- Průměrná délka slov
- Průměrná délka vět
- Hustota obrázků vůči textu
- Hustota svislých čar
- Hustota odkazů
- Hustota slov vysázených verzálkami
- Hustota dlouhých slov
- Hustota krátkých slov
- Zda některý z nadřazených elementů obsahuje "pozitivně hodnocenou"třídu či identifikátor
- Zda je to nadpis
- Zda je to element type `<p>`
- Zda je to prvek seznamu
- Relativní pozice v dokumentu

Všechny tyto hodnoty jsou navíc pomocí metody `normalize_for_neural` normalizovány do intervalu $< 0, 1 >$. Pro každý element jsou tyto hodnoty posílány třikrát. Pro předchozí blok, aktuální blok a následující.

Aby síť podávala korektní výsledky a byla co nejpřesnější, musel jsem nejdříve najít vhodné nastavení aktivačních funkcí a jejich strmosti, vhodný učící algoritmus, vhodný počet skrytých vrstev a počty neuronů v jednotlivých vrstvách. Mnou nalezeny optimální parametry sítě jsou uvedeny v Tabulce 4.1. Informace o tom, jak jsem našel vhodné nastavení budou uvedeny v některé z následujících kapitol.

Výstupem této analýzy je kolekce bloků, kdy každý blok již obsahuje svoje hodnocení na základě výstupní hodnoty z neuronové sítě. Tyto hodnoty jsou následně porovnány vůči implicitně nastaveným prahům a na základě výsledku jsou bloky rozděleny do tří tříd: správné bloky, téměř správné a špatné.

Vlastnost	Typ/Hodnota
Počet neuronů ve vstupní vrstvě	78
Počet skrytých vrstev	3
Počet neuronů v 1. skryté vrstvě	39
Aktivační funkce v 1. skryté vrstvě	Sigmoid stepwise
Strmost aktivační funkce v 1. skryté vrstvě	0.65
Počet neuronů v 2. skryté vrstvě	19
Aktivační funkce v 2. skryté vrstvě	Gaussian symmetric
Strmost aktivační funkce v 2. skryté vrstvě	0.75
Počet neuronů v 3. skryté vrstvě	9
Aktivační funkce v 3. skryté vrstvě	Gaussian symmetric
Strmost aktivační funkce v 3. skryté vrstvě	0.75
Počet neuronů ve výstupní vrstvě	1
Aktivační funkce ve výstupní vrstvě	Linear piece symmetric
Strmost aktivační funkce ve výstupní vrstvě	0.8

Tabulka 4.1: Nastavení neuronové sítě

4.13 Následná analýza

V této části se snažím eliminovat chybovost neuronové sítě. Určité bloky v rámci hlavního textu totiž můžou mít vlastnosti generického obsahu. Takovými bloky jsou zpravidla krátké bloky obsahující velké množství odkazů, nadpisy nižší úrovně apod.

Následná analýza je implementována ve třídě `PostAnalyser` a stejně jako většina implementovaných tříd obsahuje pouze dvě veřejné metody `new` a `perform`.

První algoritmus následné analýzy je implementován v metodě `main_text_start` a snaží se nalézt začátek hlavního textu, čili startovací hlavní nadpis na základě podobnosti s názvem stránky. Pokud je takový nadpis nalezen, algoritmus mu nastaví skóre 1.0 a všechny elementy před ním ve vzdálenosti alespoň tří bloků jsou označeny jako špatné, tedy je jim nastaveno skóre 0.

Další metoda, `common_path`, zkoumá společné nadřazené elementy. Pokud alespoň tři bloky vyhodnoceny jako správné mají nadřazený prvek se stejnou cestou, pak všechny bloky v kolekci se stejnou vlastností jsou označeny jako správné.

Předposlední metodou je metoda `alone_blocks`, která prochází bloky odzadu a v případě, že nalezne blok, před kterým nejdále 5 bloků se nenachází žádný správný blok, je označen za špatný. Tento algoritmus končí ve chvíli kdy nalezne správný blok, který nelze označit na základě předchozího tvrzení za špatný.

Ve finále je využita metoda `block_neighbours`, která zkoumá okolí aktuálního bloku a na základě okolních bloků rozhoduje o správnosti téměř správných bloků. Například nadpis je označen za správný v případě, že v okolí 3 bloků oběma směry se nalézá správný nebo téměř správný blok, nebo pokud se maximálně 3 bloky pod vyskytuje správný blok.

4.14 Hlavní třída

Třída `App` propojuje všechny výše uvedené třídy dohromady. Postupně tvoří instance každé z nich a provádí extrakci od předzpracování až po samotnou extrakci. Konstruktor vyžaduje

jako parametr cestu k souboru s dokumentem nebo adresu URL. Metody `preprocess`, `analyse` a `post_analyse` provádí jednotlivé úkony. Tyto metody jsou veřejné a všechny vrací kolekci bloků ve stavu odpovídajícímu danému úkonu. Metoda `perform` provede všechny operace najednou a vrací již klasifikovanou kolekci bloků.

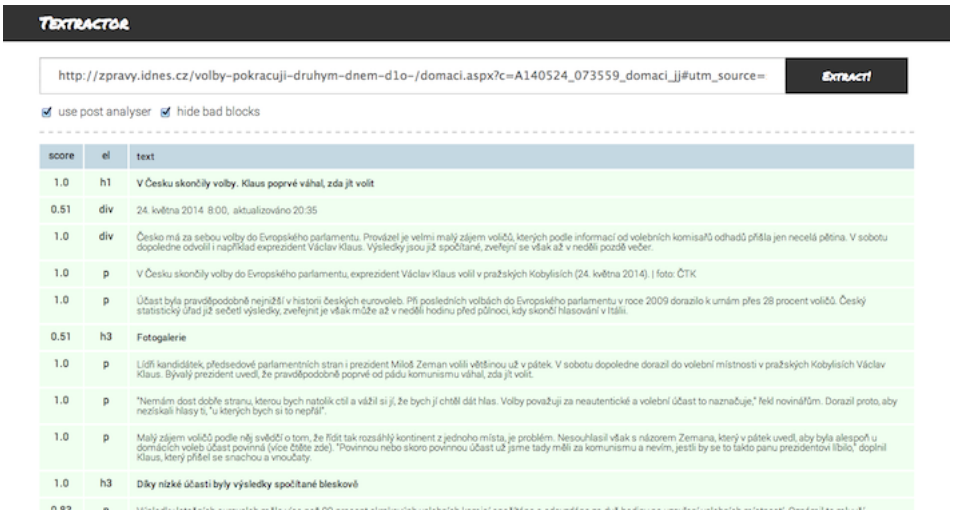
Pokud je program připojen do jiného systému jako gem, lze extrakce docílit například takto: `Textractor::App.new('/path/to/doc.html').perform`. Výstupem pak bude instance třídy `BlockCollection`, s vlastnostmi pole a pro každý blok obsažen v kolekci budou dostupné všechny jeho vlastnosti, skóre atd.

4.15 Příkazový řádek

Rozhraní pro ovládaní z příkazové řádky je implementováno v adresáři `bin` v souboru `textractor` ve třídě `TextractorCLI` pomocí gemu `Thor`. Příklady použití lze najít ve Zdrojovém kódu 4.2.

```
textractor go! URL_OR_FILE [-P bool]   # Extrakce textu ze stranky
textractor help [COMMAND]              # Napoveda
textractor server                      # Spusti webove rozhrani
```

Zdrojový kód 4.2: Příklady práce z programem v příkazové řádce



Obrázek 4.2: Webové rozhraní programu

4.16 Webové rozhraní

V rámci programu jsem implementoval jednoduché webové rozhraní za použití gemu `sinatra`. Toto rozhraní je implementováno v rámci modulu `Server`. Ukázka webového rozhraní je na na Obrázku 4.2, kde můžeme vidět výsledek zpracování stránky. Jednotlivá zaškrtávací tlačítka nám dovolují zapnout resp. vypnout následnou analýzu a zobrazit či skrýt nevyhovující bloky. V tabulce jsou pak informace o jednotlivých blocích, kdy první sloupec určuje výsledné ohodnocení bloku, v následujícím sloupci je informace o typu bloku a v posledním je již samotný obsah bloku. V případě, že zaškrtneme nezobrazování nevyhovujících bloků, vidíme v podstatě pouze hlavní obsah.

Kapitola 5

Testování a vyhodnocení

V této části popisuji průběh testování a porovnávám výsledky oproti jiným, již existujícím řešením. Hlavním úkolem během testování bylo najít odpovídající nastavení neuronové sítě.

5.1 Trénovací množina

Abych byl schopen dostatečně otestovat a najít správné nastavení neuronové sítě, musel jsem nejdříve získat dostatečně obsáhlou množinu dat pro učení.

Do trénovací množiny jsem zahrnul nejznámější české zpravodajské servery, náhodně vybrané české blogy a stránky z wikipedie. Všechny webové dokumenty byly získávány ručně pomocí webového prohlížeče a následně uloženy pro další zpracování. Trénovací množina obsahuje celkem 4519 bloků z 34 stránek. Podrobný přehled použitých stránek lze vidět v tabulce 5.1. Ve všech případech byly použity články z uvedených webů. Stránky použité pro vytvoření trénovací množiny lze rovněž najít na přiloženém CD.

Abych nemusel soubor s trénovací množinou generovat manuálně, vytvořil jsem třídu `TrainUtils` v rámci modulu `Train`. Instanční metoda třídy přijímá jako argument cestu k adresáři s dokumenty pro trénování. Poté stránky dělí na bloky a ke každému bloku se dotáže zda je blok správný. Takto pokračuje dokud nenarazí na konec adresáře. Trénovací data ukládá do souboru, tyto pak budou využity při trénování sítě. Metoda zároveň ukládá do vedlejšího souboru otisky jednotlivých bloků a zda je blok správný. Při dalším učení tedy není nutné hodnotit již známé bloky.

Pro každý dokument jsou tedy ručně anotovány všechny bloky a zároveň je generován soubor ve formátu, který je vyžadován knihovnou FANN. Tento soubor je pak načten a na základě informací obsažených v něm je síť trénována.

5.2 Experimenty s nastavením neuronové sítě

V této fázi testování jsou již k dispozici trénovací data a je tedy možné experimentovat s nastavením sítě a najít tak vhodné hodnoty jednotlivých parametrů neuronové sítě. Zároveň jsem na základě výsledku a experimentů opravoval případné chyby a nepřesnosti v implementaci, zejména pak optimalizoval následnou analýzu.

V první iteraci testování jsem použil pouze 26 vstupů, nebyly tedy použity data ze sousedních bloků a neuronová síť byla ponechána ve výchozím nastavení, tedy na hodnoty, které jsou nastaveny knihovnou FANN. Tato knihovna má k dispozici na výběr ze čtyř různých učících algoritmů:

Zdroj	Stránky		Zdroj	Stránky
idnes.cz	2		blog.idnes.cz	1
blesk.cz	1		blog.filosof.biz	1
blog.komart.cz	1		blog.nic.cz	1
blogs.technet.com	1		ct24.cz	3
ctsport.cz	1		denikreferendum.cz	1
digitalniekonomika.cz	2		g.cz	1
itbiz.cz	1		jenpromuze.cz	1
lidovky.cz	1		mozkyinternetu.cz	1
moravskoslezsky.denik.cz	1		mywindows.cz	1
novinky.cz	1		sip.denik.cz	1
sport.cz	1		super.cz	2
trail-busters.com	1		tyinternety.cz	1
wikipedie.cz	2		zive.cz	2

Tabulka 5.1: Trénovací množina

- `Incremental` - klasická implementace algoritmu zpětného šíření chyby. Váhy jsou aktualizovány mnohokrát během jedné epochy. Určité problémy se tímto algoritmem trénují velmi rychle, na druhou stranu u určitých problému je trénování velmi pomalé až nemožné.
- `Batch` - klasický algoritmus zpětného šíření chyby. V tomto případě jsou váhy aktualizovány na základě střední kvadratické odchylky po vyčerpání celé trénovací množiny. Jelikož, na rozdíl od předchozího algoritmu, dochází k přesnějšímu výpočtu chyby, určité problémy mohou dosahovat lepších výsledků. Trénovaní je pomalejší než v případě předchozího.
- `Rprop` - pokročilejší forma předchozího algoritmu. Je velmi obecný, použitelný pro většinu řešených problému. Je adaptivní, a proto není možné explicitně určit tempo učení.
- `Quickprop` - podobně jako `Rprop`, je to pokročilejší forma `Batch` algoritmu. V tomto případě lze explicitně nastavit tempo učení. Velmi efektivní pro mnoho problémů, nicméně u některých problému může selhávat.

Při prvním testu jsem použil síť s jednou skrytou vrstvou s 26 neurony. Již na první pohled se však zdálo toto nastavení nevyhovující a po několika testech jsem skončil s 13 neurony ve skryté vrstvě. Hlavním cílem v této fázi bylo zjištění možnosti nastavení sítě, jak jednotlivá nastavení ovlivňují výsledky a zejména vyzkoušení všech trénovacích algoritmů. Výhodou takto jednoduché neuronové sítě je rychlost trénování, lze tedy bez problému testovat velké množství různých kombinací nastavení. Nejdříve jsem zkoušel trénovací algoritmy. Algoritmy `Batch` a `Quickprop` se ukázaly jako naprosto nepoužitelné pro testování. Trénování velmi pomalé, i po několika desítkách tisíc epoch byla chyba příliš vysoká. Nejlepší výsledky jsem získával pomocí algoritmu `Rprop`, přičemž `Incremental` podával také zajímavé výsledky. Vhodnost trénovacího algoritmu jsem hodnotil zejména podle výsledné chyby po 15000 epochách, kdy trénování jsem prováděl vždy v 5 cyklech a do úvahy jsem

bral nejlepší výsledek. Srovnání nejmenší dosažené chyby lze vidět v tabulce 5.2. Pro testování jsem tedy zvolil jako výchozí algoritmus `Rprop`. Vhodnost tohoto algoritmu bude pak ještě přehodnocena po nalezení ideálního nastavení neuronové sítě. Jak se dalo již na začátku předpokládat, takto jednoduchá síť podávala velmi zmatené výsledky. V několika málo případech detekovala bloky správně, většinou však naopak. Největším problémem v této konfiguraci bylo příliš jednoznačné ohodnocení bloků, kdy se hodnoty téměř vždy vyskytovaly v podobě 0 nebo 1. Jak již jsem zmiňoval, síť detekovala bloky zmateně a jednoznačné ohodnocení je naprosto nevyhovující pro další zpracování.

Algoritmus	**Počet epoch**	**Nejmenší dosažená chyba**
`Incremental`	15000	$0,01279$
`Batch`	15000	$0,03651$
`Rprop`	15000	$0,01025$
`Quickprop`	15000	$0,03429$

Tabulka 5.2: Dosažené chyby pro různé učící algoritmy

V další fázi jsem přidával postupně další skryté vrstvy a experimentoval jsem s různým počtem neuronů v jednotlivých vrstvách. Již při dvou skrytých vrstvách bylo viditelné zlepšení, zejména se zmenšovala výsledná chyba během učení sítě. Po několika desítkách testů jsem zvolil nastavení se třemi skrytými vrstvami s počtem neuronů postupně 13, 7, 4. Je důležité zmínit, že ostatní nastavení bylo stále ponecháno na výchozích hodnotách, tedy sigmoidální skoková aktivační funkce se strmostí $0,5$. Takto nastavena neuronová síť podávala již podstatně uspokojivější výsledky, zejména pak nebylo ohodnocení již tak jednoznačné, a bloky bylo možno dělit do více tříd, tedy na vyhovující, téměř vyhovující a nevyhovující, což je nezbytné pro další zpracování, zejména pro následnou analýzu.

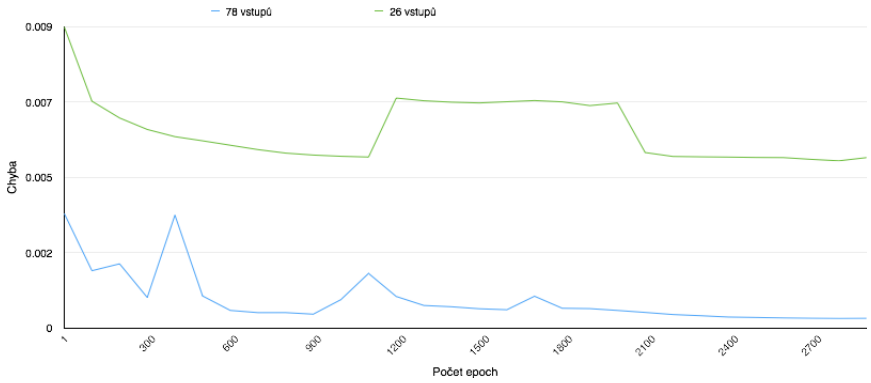
Dále jsem se snažil najít vhodnější aktivační funkce jednotlivých vrstev nežli je výchozí, tedy sigmoidální skoková. Knihovna FANN poskytuje mnoho různých aktivačních funkcí. Již na začátku se však ukázalo, že pro řešení mého problému jsou použitelné pouze 3, a to sigmoidální symetrická skoková aktivační funkce, symetrická Gaussova aktivační funkce a symetrická saturovaná lineární aktivační funkce. Vhodnost jednotlivých funkcí byla opět vyhodnocena na základě chyby při učení dosažené po 3000 epochách, při použití algoritmu `Rprop`. Nejlepší výsledky byly dosaženy při následujícím nastavení: sigmoidální symetrická skoková aktivační funkce pro 1. skrytou vrstvu, symetrická Gaussova aktivační funkce pro 2. a 3. skrytou vrstvu a symetrická lineární aktivační funkce pro výstupní vrstvu.

V další fázi jsem upravoval ještě strmost daných aktivačních funkcí. Důvodem proč bylo potřeba upravit strmost je zejména metoda, pomocí které normalizuji hodnoty do intervalu $< 0, 1 >$. Jelikož tato normalizace většinou vrací hodnoty z intervalu $< 0; 0, 3 >$, je třeba strmost aktivačních funkcí zvýšit. Různým posouváním strmostí ve skrytých vrstvách a výstupní vrstvě jsem došel k následujícímu nastavení, postupně pro první skrytou vrstvu až vrstvu výstupní, $0,65$, $0,75$, $0,75$, $0,8$.

V této konfiguraci neuronová síť dosahovala již velmi dobrých výsledků, ale stále nebyly výsledky dostačující. Napadlo mě tedy přidat na vstup i vlastnosti okolních bloků a otestovat zda se detekce zlepší. Počet vstupů byl tedy navýšen na 78 a na vstup se začaly přivádět i vlastnosti předchozího i následujícího bloku. Jelikož v předchozích experimentech jsem již určil ideální počet skrytých vrstev, ponechal jsem tedy toto nastavení, s podobnými parametry, čili v následující vrstvě vždy poloviční počet neuronů než ve vrstvě předchozí, což ve výsledků dává 39, 19 a 9 neuronů v jednotlivých skrytých vrstvách. S počtem neuronů

jsem experimentoval, ukázalo se však, že původní nastavení je opravdu vhodné. Strmost aktivačních funkcí a jejich typ byl rovněž ponechán z předchozí fáze testování. Protože rapidně vzrostl počet vstupů, odrazilo se to i na době trénování, které bylo nyní o mnoho pomalejší. Výsledná chyba při trénování však byla o jeden řád nižší než v případě 26 vstupů. Rovněž reálné výsledky začaly být již velmi uspokojivé a v mnoha případech byly bloky detekovány správně anebo bylo ohodnocení na hranici. Problém činí zejména nadpisy v rámci článku, které jsou často vyhodnocovány, stejně jako v případech předchozích, nesprávně. Tento neduh nicméně ve většině případů odstraňuje následná analýza, která ohodnocení koriguje na základě okolních bloků, které jsou vyhodnoceny správně.

V grafu 5.1 je možno vidět průběh učení neuronové sítě s 26 vstupy a 78 vstupy ve finálním nastavení. Jak je vidět, síť s 78 vstupy, kdy jsou brány do úvahy i okolní bloky, se učí podstatně rychleji a výsledná chyba je mnohem menší. Výsledné nastavení neuronové sítě, která je použita při následném vyhodnocení, je znázorněno v tabulce 4.1.



Obrázek 5.1: Průběh učení neuronové sítě pro 26 vstupů a 78 vstupů

5.3 Výsledky

V této části budou popsány výsledky algoritmu. Bude zde vyhodnocena úspěšnost detekování správných resp. špatných bloků a úspěšnost detekování začátku a konce hlavního obsahu. Zaměřím se hlavně na přední české zpravodajské servery, zahrnu však i některé zájmové weby a zahraniční zpravodajské portály a jiné.

Porovnání jsem prováděl manuálně. z každého webu resp. portálu jsem získal většinou alespoň dvě stránky s článkem, extrahoval hlavní text pomocí mnou vytvořeného programu a manuálně porovnal s očekávaným výstupem. Výsledky jsem zanesl do tabulek.

Tabulka 5.3 ukazuje výsledky pro 33 webových dokumentů. V prvním sloupci je vždy uveden zdroj, ze kterého byl dokument s článkem získán, v následující sloupec určuje celkový počet bloků na stránce. Dále je uveden celkový počet bloků, které neuronová síť detekovala jako bloky hlavního obsahu. Další dva sloupce postupně určují, kolik z bloků označených za správné je detekováno nesprávně a kolik bloků nebylo nalezeno. V některých případech jsou v závorce uvedeny detaily ohledně nesprávně určených bloků resp. nenalezených bloků. Jak je vidět, většinou jsou chybně označeny bloky jako správné v případě, že se jedná o popisky k fotografiím v rámci hlavního obsahu, popřípadě určité doplňující texty, nepatřící k hlavnímu obsahu nicméně často související. Výsledky rovněž ukazují, že se často chybně interpretují bloky, které jsou položkami seznamu a neobsahují mnoho textu, a jsou často

Portál	CB	DB	NDB	NB	Z	K
ct24.cz	155	20	1	5 (4LI)	ok	ok
ct24.cz	130	8	0	0	ok	ok
ct24.cz	147	12	2	5 (3LI)	ok	-1
ctsport.cz	121	6	1	1	ok	-1
ctsport.cz	130	10	3	1	ok	-2
idnes.cz	262	8	3	1	-1	-2
idnes.cz	277	11	2 (1H, 1F)	2	-1	ok
lidovky.cz	452	20	8 (2H, 1F, 1IN)	0	ok	-6
lidovky.cz	437	17	3 (2IN)	3	ok	-2
sport.cz	125	11	6 (6F)	0	ok	ok
sport.cz	109	7	2 (2F)	0	ok	ok
ihned.cz	159	9	5 (2H)	8	-1	-4
ihned.cz	159	12	6 (2H,1IN)	7	-1	-4
novinky.cz	104	17	4 (1F,2IN)	1	ok	-1
novinky.cz	93	11	5 (2F)	0	-1	-2
denik.cz	173	24	0	0	ok	ok
denik.cz	151	6	0	1	+1	ok
ceskenoviny.cz	125	6	2 (1F)	4	+1	+3
ceskenoviny.cz	121	11	6 (1F)	2	+1	-5
super.cz	44	5	1 (1F)	0	-1	ok
super.cz	37	6	2 (1F)	0	-1	-1
extra.cz	71	11	6 (3F, 2H)	1	-1	-5
extra.cz	68	6	3 (3F)	2	ok	-2
digitalniekonomika.cz	26	7	1	0	ok	-1
blog.respekt.ihned.cz	86	12	0	1	+1	ok
blog.respekt.ihned.cz	89	7	1	1	+1	-1
blog.scuk.cz	27	12	2	7	-2	+6
blog.tomashajzler.com	109	12	8	1	-1	-8
cnn.com	297	10	2	13	ok	+13
cnn.com	342	10	2	36	ok	+20
reuters.com	268	10	1	9	-1	+2
blogs.afp.com	147	26	11 (9F)	1	ok	-3

Tabulka 5.3: Výsledky dosažené bez použití následné analýzy

CB	Celkem bloků na stránce
DB	Počet bloků označených programem za správné
NDB	Počet špatně označených bloků za správné
NB	Počet správných bloků, které nebyly nalezeny
z	Správně detekován začátek obsahu
K	Správně detekován konec obsahu
F	Popisek fotografie v rámci obsahu
H	Nadpis
IN	Vedlejší text v rámci obsahu
LI	Položka seznamu

chybně označeny za bloky nevyhovující i když k hlavnímu obsahu patří. Toto je způsobeno tím, že obdobné položky jsou většinou používány pro vytváření menu na stránce. Poslední dva sloupce určují zda byl správně detekován začátek resp. konec hlavního obsahu. Záporné číslo určuje, kolik bloků bylo označených nesprávně jako vyhovující před začátkem resp. koncem hlavního obsahu. Kladné číslo naopak určuje, kolik bloků před koncem hlavního obashu chybělo, to znamená, že byly nesprávně označeny za bloky nevyhovující. Nutno podotknout, že výsledky, popsány výše, jsou výsledky získané pouze pomocí neuronové sítě, bez použití následné analýzy.

Tabulka 5.4 zobrazuje výsledky, kterých bylo dosaženo při použití neuronové sítě spolu s následnou analýzou. Tabulka je ve stejném formátu jako předchozí. z tabulky je na první pohled jisté, že následná analýza velmi úspěšně eliminuje chybovost neuronové sítě. Ve většině případů úspěšně odstraňuje nesprávně označené bloky za hlavním obsahem. Problém nastává však ve chvíli, kdy neuronová síť nesprávně určí celou posloupnost bloků jako správné, v tom případě se stává, že následná analýza výsledek ještě zhorší. Naštěstí k tomuto jevu příliš často nedochází. z výsledku je rovněž patrné, že díky následné analýze je často dodatečně určen začátek obsahu a také jsou označeny za správné bloky v rámci hlavního obsahu. Většinou se jedná o bloky obsahující velmi krátký text, příliš mnoho odkazů apod., které jsou často vyhodnoceny neuronovou sítí jako bloky nevyhovující.

5.4 Porovnání s existujícími algoritmy

5.4.1 jusText

V porovnání s algoritmem jusText program dosahuje podobných výsledků. Dá se říci, že program, který jsem implementoval si lépe poradí s dokumenty se složitější strukturou. Na druhou stranu jusText dosahuje přesnějších výsledků pokud je hlavní text při sobě. Pomikálkův algoritmus spíše ohodnotí správný blok za špatný, opačně málokdy, což je přesně opačný případ nežli v případě mého programu. Otázkou zůstává, který z případů je vhodnější. Dle mého názoru lze chybně označené dobré bloky pomocí dalších heuristik dodatečně odstranit, opačně je to již složitější, protože špatných bloků je v dokumentu zpravidla několikanásobně více než–li správných. V tabulce 5.5 můžeme vidět porovnání jusText s mou implementací. Sloupce označené jako "úspěšné"udávají poměr bloků, které byly detekovány správně jako vyhovující vůči všem vyhovujícím blokům v dokumentu. Sloupec "neúspěšné"naopak udává poměr chybně označených bloků za vyhovující vůči všem vyhovujícím blokům na stránce. Z tabulky je patrné, že co se úspěšností týče, jsou algoritmy srovnatelné. Často se stává, že tam kde jusText mírně selhává, můj program dosahuje lepších výsledků a opačně. Hodnoty ve sloupci udávající chybné určení bloků můžou být však matoucí. Jak již bylo zmíněno výše, často se v tomto případě jedná o popisky k fotografiím či určitý doplňující text a je v tomto případě sporné, zda tyto bloky do hlavního obsahu patří nebo ne. V tomto porovnání jsem je za hlavní text nepovažoval a takovéto bloky jsem bral jako chybně vyhodnocené.

5.4.2 boilerpipe

Jelikož jsem se v rámci hodnocení výsledků zaměřil zejména na české webové stránky a nikoliv anglické, bylo porovnání s boilerpipe dosti složité. U tohoto algoritmu je na první pohled vidět, že je optimalizován zejména na právě anglicky psané weby. Při extrakci z českých stránek se často stávalo že byl chybně detekován konec hlavního obsahu a do výsledku se

Stránka	CB	DB	NDB	NB	Z	K
ct24.cz	155	21	1	4 (4LI)	ok	ok
ct24.cz	130	8	0	0	ok	ok
ct24.cz	147	13	0	3 (3LI)	ok	ok
ctsport.cz	121	5	0	1	ok	ok
ctsport.cz	130	8	0	0	ok	ok
idnes.cz	262	11	5 (1F)	0	ok	-3
idnes.cz	277	12	1 (1F)	0	ok	ok
lidovky.cz	452	24	12 (3H, 2F, 1IN)	0	ok	-9
lidovky.cz	437	18	1 (1IN)	0	ok	ok
sport.cz	125	14	9 (8F)	0	ok	ok
sport.cz	109	7	2 (2F)	0	ok	ok
ihned.cz	159	5	1 (1H)	8	ok	-1
ihned.cz	159	15	3 (1H, 2IN)	1	ok	-1
novinky.cz	104	17	3 (1F, 2IN)	0	ok	ok
novinky.cz	93	9	3 (2F)	0	-1	ok
denik.cz	173	25	1 (1F)	0	ok	ok
denik.cz	151	7	0	0	ok	ok
ceskenoviny.cz	125	10	2 (2F)	0	ok	ok
ceskenoviny.cz	121	15	8 (1F, 1H)	0	ok	-7
super.cz	44	5	1 (1F)	0	-1	ok
super.cz	37	5	1 (1F)	0	ok	-1
extra.cz	71	9	3 (3F)	0	ok	-3
extra.cz	68	5	2 (2F)	2	-1	-2
digitalniekonomika.cz	26	8	2	0	ok	-2
blog.respekt.ihned.cz	86	13	0	0	ok	ok
blog.respekt.ihned.cz	89	7	0	0	ok	ok
blog.scuk.cz	27	16	0	2	ok	ok
blog.tomashajzler.com	109	10	5	0	ok	-5
cnn.com	297	25	4 (4IN)	0	ok	-4
cnn.com	342	51	7 (4F, 3IN)	0	ok	-4
reuters.com	268	20	1	0	ok	ok
blogs.afp.com	147	28	12 (9F)	0	ok	-3

Tabulka 5.4: Výsledky dosažené s použitím následné analýzy

CB	Celkem bloků na stránce
DB	Počet bloků označených programem za správné
NDB	Počet špatně označených bloků za správné
NB	Počet správných bloků, které nebyly nalezeny
z	Správně detekován začátek obsahu
K	Správně detekován konec obsahu
F	Popisek fotografie v rámci obsahu
H	Nadpis
IN	Vedlejší text v rámci obsahu
LI	Položka seznamu

Zdroj	Textractor	Textractor	JusText	JusText
	Úspěšné	Neúspěšné	Úspěšné	Neúspěšné
ct24.cz	83.33%	4.17%	100.0%	16.67%
ct24.cz	100.0%	0.0%	100.0%	75.0%
ct24.cz	81.25%	0.0%	75.0%	31.25%
ctsport.cz	83.33%	0.0%	100.0%	100.0%
ctsport.cz	100.0%	0.0%	100.0%	87.5%
idnes.cz	100.0%	83.33%	100.0%	16.67%
idnes.cz	100.0%	9.09%	100.0%	18.18%
lidovky.cz	100.0%	100.0%	100.0%	41.67%
lidovky.cz	100.0%	5.88%	88.24%	17.65%
sport.cz	100.0%	180.0%	100.0%	80.0%
sport.cz	100.0%	40.0%	100.0%	40.0%
ihned.cz	33.33%	8.33%	91.67%	8.33%
ihned.cz	92.31%	23.08%	92.31%	7.69%
novinky.cz	100.0%	21.43%	100.0%	14.29%
novinky.cz	100.0%	50.0%	66.67%	0.0%
denik.cz	100.0%	4.17%	100.0%	4.17%
denik.cz	100.0%	0.0%	100.0%	0.0%
ceskenoviny.cz	100.0%	25.0%	100.0%	12.5%
ceskenoviny.cz	100.0%	114.29%	100.0%	14.29%
super.cz	100.0%	25.0%	100.0%	0.0%
super.cz	100.0%	25.0%	75.0%	0.0%
extra.cz	100.0%	50.0%	66.67%	16.67%
extra.cz	60.0%	40.0%	40.0%	0.0%
digitalniekonomika.cz	100.0%	33.33%	100.0%	0.0%
blog.respekt.ihned.cz	100.0%	0.0%	92.31%	0.0%
blog.respekt.ihned.cz	100.0%	0.0%	71.43%	0.0%
blog.scuk.cz	88.89%	0.0%	88.89%	5.56%
blog.tomashajzler.com	100.0%	100.0%	100.0%	0.0%
cnn.com	100.0%	19.05%	90.48%	285.71%
cnn.com	100.0%	15.91%	90.91%	136.36%
reuters.com	100.0%	5.26%	94.74%	0.0%
blogs.afp.com	100.0%	75.0%	100.0%	68.75%

Tabulka 5.5: Porovnání s JusText

tak dostávaly naprosto nerelevantní bloky. V těchto případech boilerpipe za mým řešením značně zaostával a podával dosti neuspokojivé výsledky. Vyzkoušel jsem rovněž několik cizojazyčných stránek, kdy byla situace už jiná a algoritmus dosahoval velmi dobrých výsledků, kdy se mu dařilo extrahovat většinou přesně celý hlavní obsah a hlavně bez žádných chybně vyhodnocených bloků. Boilerpipe poskytuje veřejně přístupné webové API, které je bohužel často přetížené a nepoužitelné, testování bylo proto velmi obtížné a zdlouhavé, soustředil jsem se tedy proto hlavně na porovnání s jusText a Readability.

5.4.3 Readability

Readability je kompletní platforma pro jednoduché čtení článků, předpokládal jsem tedy již předem, že bude dosahovat velmi dobrých výsledků. Toto tvrzení se i během testování potvrdilo jak je vidět v tabulce 5.6. Do tabulky jsem nezahrnul neúspěšnou detekci bloků, a to z jednoduchého důvodu. Zatímco já v mém řešení považuji popisky k fotografiím a vedlejší text jako nevyhovující, Readability se naopak snaží tyto data záměrně získávat. Porovnání by bylo tedy neprůkazné. Nicméně například v případě stránek `ct24.cz` nástroj Readability chybně detekoval konec obsahu a do obsahu se tak dostal i text formulářů. Dále se u některých dokumentů dostával do výsledku komentář. Ve výsledku se však dá říci, že je tento algoritmus v mnoha případech přesnější nežli můj, zejména co se chybného detekování správných bloků týče. Rozdíly nejsou však nijak markantní, a pomocí přidání dalších heuristik do následné analýzy by se daly tyto nedostatky z mnou implementovaného programu odstranit.

Zdroj	**Textractor**	**Readability**
	Úspěšné	**Úspěšné**
ct24.cz	83.33%	100.0%
ct24.cz	100.0%	100.0%
ct24.cz	81.25%	100.0%
ctsport.cz	83.33%	100.0%
ctsport.cz	100.0%	100.0%
idnes.cz	100.0%	100.0%
idnes.cz	109.09%	100.0%
lidovky.cz	100.0%	100.0%
lidovky.cz	100.0%	100.0%
sport.cz	100.0%	100.0%
sport.cz	100.0%	100.0%
ihned.cz	33.33%	91.67%
ihned.cz	92.31%	92.31%
novinky.cz	100.0%	92.86%
novinky.cz	100.0%	83.33%
denik.cz	100.0%	104.17%
denik.cz	100.0%	100.0%
ceskenoviny.cz	100.0%	87.5%
ceskenoviny.cz	100.0%	85.71%
super.cz	100.0%	75.0%
super.cz	100.0%	75.0%
extra.cz	100.0%	100.0%
extra.cz	60.0%	0.0%
digitalniekonomika.cz	100.0%	100.0%
blog.respekt.ihned.cz	100.0%	100.0%
blog.respekt.ihned.cz	100.0%	85.71%
blog.scuk.cz	88.89%	100.0%
blog.tomashajzler.com	100.0%	100.0%
cnn.com	100.0%	95.24%
cnn.com	100.0%	95.45%
reuters.com	100.0%	94.74%
blogs.afp.com	100.0%	87.5%

Tabulka 5.6: Porovnání s Readability

Kapitola 6

Závěr

Cílem této práce bylo vytvořit univerzální program pro extrakci hlavního textu z webových dokumentů. Na začátku byl čtenář seznámen s existujícími řešeními a možnými metodami pro extrakci textu. V závěru první kapitoly byly stručně popsány neuronové sítě. V dalších kapitolách již byl popsán návrh samotná implementace, které předcházel stručný úvod do použitých technologií a knihoven.

Program byl vyvíjen primárně pro Ruby `2.0.0`, neměl by však být problém ani s nižšími verzemi. Vývoj probíhal primárně na systému OS X, díky multiplatformní povaze jazyka Ruby bude však možno využívat program i na alternativních operačních systémech.

V práci je názorně ukázáno, že dopředné vícevrstvé umělé neuronové sítě jsou vhodným nástrojem při extrakci hlavního textu a podávají velice dobré výsledky. V průběhu realizace jsem si uvědomil, že jsem mohl určité vlastnosti vynechat a naopak některé přidat. Hlavním důvodem pro vynechání vlastností je pak veliká podobnost s vlastnostmi jinými, jako příklad uvedu počet dlouhých slov v bloku a hustotu těchto slov. i přes tyto nedostatky se mi podařilo vytvořit dobře použitelný nástroj pro extrakci hlavního textu, který v jazyce Ruby zatím chyběl.

Důležitým poznatkem je, že i přesto, že trénovací množina sestávala z pouze českých webových dokumentů, program dosahoval velice uspokojivých výsledků i v případě cizojazyčných stránek. Na základě této skutečnosti lze vysledovat, že většina stránek obsahujících článek, má podobnou strukturu a mnou vytvořený program je tedy jazykově nezávislý.

I když výsledky klasifikace neuronové sítě byly uspokojivé, díky této práce jsem si ověřil, že je vhodné provést po klasifikaci určitou korekci výsledků, kterou v tomto případě zajišťuje následná analýza.

Do budoucna bych chtěl zkusit naučit neuronovou síť na rozdílných datech, aby bylo dosaženo dobrých výsledků i v případech extrakce produktů z internetových obchodů nebo analýze hlavních stránek webů. V tomto případě by však bylo nutné upravit značným způsobem i následnou analýzu. V rámci další práce na programu bych se rovněž snažil dále experimentovat se vstupy sítě. V případě další práce bych v každém případě rozšířil i množství trénovacích dat na více různých jazyků. Zajímavým rozšířením by mohlo být i poskytnutí webového API.

Dle mého názoru byly cíle práce z většiny naplněny, což dokládají i předchozí tvrzení. V určitých směrech se nepodařilo dosáhnout vytyčených cílů, na druhou stranu byl vytvořen použitelný program, lehce rozšiřitelný s možností jednoduché integrace do jiných systému. Ačkoliv jsem v rané fázi nepočítal s použitím neuronové sítě pro klasifikaci, ve výsledků se toto rozhodnutí ukázalo jako velmi správné.

Literatura

[1] Introduction to Neural Networks [online]. 2014, [cit. 2014-05-23].
URL http://www.learnartificialneuralnetworks.com/introduction-to-neural-networks.html

[2] Burget, R.: Automatic Web Document Restructuring Based on Visual Information Analysis. In *Advances in Intelligent Web Mastering - 2, Proceedings of the 6th Atlantic Web Intelligence Conference - AWIC'2009*, Advances in Intelligent and Soft Computing , Vol. 67, Springer Verlag, 2010, ISBN 978-3-642-10686-6, s. 61–70.
URL http://www.fit.vutbr.cz/research/view_pub.php?id=9027

[3] Cai, D.; Yu, S.; Wen, J.-R.; aj.: VIPS: a Vision-based Page Segmentation Algorithm. Technická zpráva, Microsoft (MSR-TR-2003-79), November 2003.

[4] Evert, S.: A Lightweight and Efficient Tool for Cleaning Web Pages. In *Proceedings of the Sixth International Conference on Language Resources and Evaluation (LREC'08)*, Marrakech, Morocco: European Language Resources Association (ELRA), may 2008, ISBN 2-9517408-4-0, http://www.lrec-conf.org/proceedings/lrec2008/.

[5] Feldman, R.; Sanger, J.: *The Text Mining Handbook: Advanced Approaches in Analyzing Unstructured Data*. ITPro collection, Cambridge University Press, 2006, ISBN 9781139457835.
URL http://books.google.cz/books?id=3PcEoz48RBcC

[6] Kohlschütter, C.; Fankhauser, P.; Nejdl, W.: Boilerplate detection using shallow text features. In *WSDM*, editace B. D. Davison; T. Suel; N. Craswell; B. Liu, ACM, 2010, ISBN 978-1-60558-889-6, s. 441–450.

[7] Kunc, M.; Burget, R.: Klasifikace prvků dokumentu na základě vizuálních rysů. In *Znalosti 2008*, Vydavateľstvo STU, 2008, ISBN 978-80-227-2827-0, s. 347–350.
URL http://www.fit.vutbr.cz/research/view_pub.php.cs?id=8564

[8] McKeown, K. R.; Barzilay, R.; Evans, D.; aj.: Tracking and Summarizing News on a Daily Basis with Columbia's Newsblaster. 2002.

[9] Pasternack, J.; Roth, D.: Extracting Article Text from the Web with Maximum Subsequence Segmentation. In *Proceedings of the 18th International Conference on World Wide Web*, WWW '09, New York, NY, USA: ACM, 2009, ISBN 978-1-60558-487-4, s. 971–980, doi:10.1145/1526709.1526840.
URL http://doi.acm.org/10.1145/1526709.1526840

[10] Pomikálek, J.: *Removing Boilerplate and Duplicate Content from Web Corpora [online]*. Disertační práce, Masarykova univerzita, Fakulta informatiky, 2011 [cit. 2014-01-05].
URL http://is.muni.cz/th/45523/fi_d/

[11] Schfer, R.: Efficient High-precision Boilerplate Detection Using Multi-Layer Perceptrons. 2014, [cit. 2014-01-05] Submitted to LREC 2014.
URL https://hpsg.fu-berlin.de/{~}rsling/downloads/pubs/Schaefer_LREC2014-submitted_BoilerplateDetectionMLP.pdf

[12] Spousta, M.; Marek, M.; Pecina, P.: Victor: the Web-Page Cleaning Tool. In *Proceedings of the 4th Web as Corpus Workshop, LREC*, 2008.

[13] Vieira, K.; da Silva, A. S.; Pinto, N.; aj.: A Fast and Robust Method for Web Page Template Detection and Removal. In *Proceedings of the 15th ACM International Conference on Information and Knowledge Management*, CIKM '06, New York, NY, USA: ACM, 2006, ISBN 1-59593-433-2, s. 258–267, doi:10.1145/1183614.1183654.
URL http://doi.acm.org/10.1145/1183614.1183654

[14] Wikipedia: N-gram [online]. 2013, [cit. 2014-01-05].
URL http://en.wikipedia.org/w/index.php?title=N-gram&oldid=583934400

[15] Wikipedia: Naive Bayes classifier [online]. 2013, [cit. 2014-01-05].
URL http://en.wikipedia.org/w/index.php?title=Naive_Bayes_classifier&oldid=584046958

[16] Wikipedia: Ruby [online]. 2013, [cit. 2014-07-31].
URL http://cs.wikipedia.org/w/index.php?title=Ruby_(programovac%C3%AD_jazyk)&oldid=11580983

Příloha A

Obsah CD

- adresář `textractor` obsahující zdrojový text programu
- soubor `thesis.pdf` obsahující text práce