VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
FACULTY OF INFORMATION TECHNOLOGY

ÚSTAV INFORMAČNÍCH SYSTÉMŮ
DEPARTMENT OF INFORMATION SYSTEMS

METODY STROJOVÉHO UČENÍ NAD WEBOVÝMI DOKUMENTY

MACHINE LEARNING METHODS FOR WEB DOCUMENTS

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE — **Bc. JOSEF KATRŇÁK**
AUTHOR

VEDOUCÍ PRÁCE — **doc. Ing. RADEK BURGET, Ph.D.**
SUPERVISOR

BRNO 2023

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
FAKULTA INFORMAČNÍCH TECHNOLOGIÍ

Zadání diplomové práce

144822

Ústav: Ústav informačních systémů (UIFS)
Student: **Katrňák Josef, Bc.**
Program: Informační technologie a umělá inteligence
Specializace: Informační systémy a databáze

Název: **Metody strojového učení nad webovými dokumenty**
Kategorie: Umělá inteligence
Akademický rok: 2022/23

Zadání:

1. Prostudujte současné metody klasifikace obsahu webových stránek s využitím metod strojového učení.
2. Seznamte se s experimentálním nástrojem FitLayout, jeho způsobem reprezentace dokumentů a možnostmi generování datových sad pro klasifikaci tímto nástrojem.
3. Navrhněte způsob aplikace vhodné metody nebo metod strojového učení pro klasifikaci konkrétních částí obsahu webových stránek.
4. Po konzultaci s vedoucím zvolte vhodné technologie a implementujte experimentální aplikaci pro klasifikaci obsahu s využitím navržených metod.
5. Proveďte testování implementovaných metod na vhodné datové sadě.
6. Zhodnoťte dosažené výsledky.

Literatura:

- Gogar, T., Hubáček, O., Šedivý, J.: Deep Neural Networks for Web Page Information Extraction. 12th IFIP International Conference on Artificial Intelligence Applications and Innovations (AIAI), Sep 2016, Thessaloniki, Greece. pp.154-163.
- Dále dle pokynů vedoucího.

Při obhajobě semestrální části projektu je požadováno:
Body 1 až 3

Podrobné závazné pokyny pro vypracování práce viz https://www.fit.vut.cz/study/theses/

Vedoucí práce: **Burget Radek, doc. Ing., Ph.D.**
Vedoucí ústavu: Kolář Dušan, doc. Dr. Ing.
Datum zadání: 1.11.2022
Termín pro odevzdání: 17.5.2023
Datum schválení: 24.10.2022

Abstrakt

Cílem práce je využití metod strojového učení pro klasifikaci specifických částí obsahu webových stránek. Nejprve jsou popsány současné metody reprezentace a klasifikace obsahu webových stránek s využitím metod strojového učení. Pro reprezentaci webové stránky se práce zaměřuje na experimentální nástroj FitLayout, jehož vizuální reprezentace webových stránek slouží jako vstup pro další zpracování a následné trénování modelů strojového učení. Výsledkem práce jsou natrénované modely, které klasifikují konkrétní části obsahu webových stránek. Architektura modelu je založena na grafových neuronových sítích. Pro experimenty je použita datová sada veřejně dostupných webových stránek, které obsahují stránky online prodávaných produktů. Výhodou navrženého a implementovaného přístupu je extrakce informací nezávislá na struktuře a jazyku webové stránky.

Abstract

This work aims to use machine learning techniques for the classification of specific parts of web page content. First, current methods for representing and classifying web page content using machine learning methods are described. For web page representation, the thesis focuses on the experimental tool FitLayout, whose visual representation of web pages serves as input for further processing and subsequent training of machine learning models. The work results in trained models that classify specific parts of the web page content. The model architecture is based on graph neural networks. For the experiments, a dataset of publicly available websites containing pages of products sold online is used. The advantage of the proposed and implemented approach is information extraction independent of the structure and language of a web page.

Klíčová slova

strojové učení, grafové neuronové sítě, klasifikace, webové dokumenty, extrakce informací z webu, dolování obsahu webu, reprezentace webových dokumentů, FitLayout

Keywords

machine learning, graph neural networks, classification, web documents, web information extraction, web content mining, web documents representation, FitLayout

Citace

KATRŇÁK, Josef. *Metody strojového učení nad webovými dokumenty.* Brno, 2023. Diplomová práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce doc. Ing. Radek Burget, Ph.D.

Metody strojového učení nad webovými dokumenty

Prohlášení

Prohlašuji, že jsem tuto diplomovou práci vypracoval samostatně pod vedením pana doc. Ing. Radka Burgeta Ph.D. Uvedl jsem všechny literární prameny, publikace a další zdroje, ze kterých jsem čerpal.

.....
Josef Katrňák
15. května 2023

Poděkování

Rád bych poděkoval svému vedoucímu práce doc. Ing. Radku Burgetovi Ph.D. za pravidelné konzultace, pomoc a cenné rady, které mi poskytl.

Obsah

Kapitola 1

Úvod

Webové dokumenty jsou každodenní součástí života většiny z nás. Avšak kontextuální informace, které jsou užitečné pro uživatele, mohou být ve webových stránkách skryté. Webové dokumenty převážně obsahují nestrukturované informace. I když existují automatické metody pro získání konkrétních částí obsahu webových stránek, stále se objevují problémy spojené se získáváním a korektní klasifikací těchto částí. Problémy pramení v neustále rostoucím množství webových stránek, které se kategoricky obsahem neliší, avšak samotné rozložení a vlastní obsah stránky mohou být značně odlišné. Vytvoření systému, který používá metody strojového učení s využitím vhodné reprezentace webové stránky, je způsob jak s těmito problémy bojovat. Výsledný nástroj umožní uživatelům rychlé získání klíčových informací z jakékoliv webové stránky z dané kategorie.

Následující práce se zaměřuje na automatickou extrakci strukturovaných dat z webových stránek. Pro experimenty je v práci zvolena úloha extrakce informací o produktu z e-shopových webových míst. Mezi takové informace patří například název produktu, jeho cena, hlavní obrázek atd. Automatické získání těchto informací z jakéhokoli e-shopu může vést k dalšímu užitečnému použití, jako třeba k automatickému porovnávání cen na různých e-shopech, analýze změn ceny a podobně. Cílem je extrahovat tyto informace nezávisle na konkrétním webovém místě, jeho struktuře a jazyce.

Technologie strojového učení lze efektivně využít i pro úkol automatické extrakce informací. Nástroj FitLayout k tomu poskytuje vizuální strukturovanou reprezentaci webového dokumentu, kterou je pro tyto technologie možné použít. Cílem práce je vytvoření datové sady a její následné použití pro trénování modelu hluboké neuronové sítě. Tento model má dvě části: embedder, který slouží k zachycení vlastností webové stránky a klasifikátor, pro určení zda jednotlivé části obsahují požadovanou informaci. Embedder se skládá z vícevrstvé grafové neuronové sítě, zatímco klasifikátor představuje klasickou lineární neuronovou vrstvu. Výhoda použití grafových neuronových sítí spočívá v zachycení vztahů mezi jednotlivými částmi webových stránek, které mohou být užitečné pro výslednou klasifikaci. Evaluace je provedena pomocí metrik jako je přesnost, skóre F1 a přesnost nominace.

Nejprve je v kapitole 2 popsán současný stav a pokrok v oblasti zpracování webových dokumentů. Následující kapitola 3 se zaměřuje na způsoby reprezentace webových stránek a navazující kapitola 4 na využití metod strojového učení v této oblasti. Kapitola 5 se věnuje návrhu systému, který využívá datovou sadu také popsanou v této kapitole. Popisem implementace navrženého systému se zabývá kapitola 6, experimenty a jejich výsledky kapitola 7. Kapitola 8 shrnuje získané poznatky a výstupy práce.

Kapitola 2

Zpracování webových dokumentů

World Wide Web (WWW) lze považovat za rozsáhlé a distribuované úložiště informací. Celkové množství webových stránek neustále roste. Průzkum firmy Netcraft vykazuje za listopad roku 2022 celkem 1 135 089 912 stránek rozmístěných mezi 271 689 143 jedinečnými webovými místy [45]. Oproti září stejného roku počet stránek vzrostl o 5 838 779 a počet webových míst o 63 883 [46]. Pokud srovnáme počty s dávnější minulostí zjistíme, že oproti listopadu 2012 vzrostl počet webových stránek o $81,5$ % a počet jedinečných webových míst o $45,4$ %.

Informace ve webových dokumentech jsou převážně nestrukturované. Kvůli velkému a stále rostoucímu množství stránek s různou a často se měnící strukturou a obsahem je potřeba systémů, které jsou schopny automaticky zpracovávat obsah těchto stránek pro získání klíčových informací. K tomu je možné použít inteligentní a flexibilní algoritmy pro získání vhodných bloků webového dokumentu (včetně všech užitečných dat a metadat), které je možno dále klasifikovat dle specifického typu obsahu.

2.1 Web mining

WWW nabízí bohaté zdroje informací pro dolování dat. Ať už se jedná o textové nebo jiné obsahové informace, informace o struktuře odkazů mezi jednotlivými stránkami nebo informace o přístupu na stránku a jejím použití.

Problémem získávání informací z webových stránek je jejich příliš velká rozsáhlost, která znemožňuje efektivní data warehousing. Dalším problémem je heterogenita a velká komplexita webových stránek. V kontextu data mining neexistují standardy a přesné struktury využitelné pro efektivní dolování.

Web mining (*dolování webu*) je proces automatického získávání informací a znalostí z webových dat [28]. Web mining se dále dělí na tři kategorie: web content mining, web structure mining a web usage mining [7].

- **Web content mining** je proces extrahování užitečných informací z obsahu webových dokumentů. Obsahem se v tomto kontextu myslí části webového dokumentu, které se prezentují uživatelům. Mezi tyto části patří například text, video nebo zvuk. Tato část dolování je hlavním cílem práce a na její realizaci pomocí metod strojového učení se zaměřuje kapitola 4.

- **Web structure mining** vidí web jako graf, kde uzly jsou dokumenty a hrany představují spojení mezi dokumenty. Tento typ dolování je proces extrahování strukturál-

ních informaci z webu. Součástí je také dolování struktury webu uvnitř jedné stránky [19]. Web structure mining pomáhá k pochopení webového obsahu a může nabízet transformaci do více strukturované reprezentace, která je použitelná k tvoření vhodných datových sad. Právě získáním vhodné strukturované reprezentace jedné webové stránky se zabývá následující kapitola 3.

- **Web usage mining** se věnuje aplikacím technik datového dolování pro objevení vzorů na webu. Prozkoumání těchto vzorů by mělo pomoci k pochopení a naplnění potřeb uživatele. Web usage mining cílí na data potřebné k použití webu pro uživatele. Informacemi použitými pro tento typ dolování jsou například sekvence akcí (kliknutí) uživatelů.

Web content mining (nebo také extrakce informací z webu) bere na vstupu webovou stránku a produkuje strukturované informace na výstupu. Webová stránka na vstupu může být reprezentována různými způsoby, které jsou podrobněji popsány v kapitole 3. Mezi typické úlohy patří identifikace obsahu stránky, vyhledávání cen produktů z různých webových míst nebo klasifikace webových stránek na základě jejich obsahu.

2.2 Terminologie

V následující sekci je vysvětlena terminologie, která se v práci dále používá.

- **Webové místo** (anglicky *website*) je unikátní adresa na internetu. Na jednom webovém místě se může nacházet více webových stránek.
- **Webová stránka** (webový dokument, anglicky *webpage*) je dokument, který je renderován prohlížečem a zobrazován uživateli.
- **URL** je adresa, která slouží k přesné specifikaci webových stránek na internetu.
- **Blok** je část obsahu webové stránky, která obsahuje informace a má šířku a výšku. Příkladem může být paragraf textu, navigační odkaz v menu, titulek článku atd.
- **Obsah** je hlavní informací webové stránky a je cílem extrakce/dolování.
- **HTML** je značkovací jazyk, který se používá pro tvorbu webových stránek.
- **Element** je část HTML dokumentu tvořená otevírací značkou, obsahem a odpovídající ukončovací značkou.
- **CSS** je jazyk pro popis způsobu zobrazení elementů na stránkách napsaných v jazyce HTML.
- **JavaScript** je skriptovací jazyk, který je možno vložit do webové stránky. Použití může být například pro ovládání interaktivních prvků grafického rozhraní nebo tvoření animací.
- **Webový prohlížeč** je počítačový program, který slouží k renderování (vykreslování) a prohlížení webových stránek.
- **Renderování** (vykreslování) je proces, který načte zdrojový kód webové stránky (HTML+CSS), případně další objekty (např. obrázky) a zobrazí naformátovaný obsah do okna webového prohlížeče.

Kapitola 3

Reprezentace webové stránky

Pro další analýzu a práci s webovými stránkami je nutná jejich správná reprezentace. V následující kapitole se seznámíme s nástroji a způsoby reprezentace webových stránek, které jsou vhodné pro další zpracování, například pro klasifikaci pomocí strojového učení, o které pojednává kapitola 4. Nejprve se seznámíme s klasickou reprezentací textového obsahu, poté se podíváme na základní model webové stránky a nakonec na další způsoby reprezentace, které berou v úvahu i vizuální vlastnosti webové stránky.

Webová stránka neobsahuje pouze samotný obsah vztahující se k tématu stránky. Oproti klasickým úlohám z oblasti zpracování přirozeného jazyka je třeba pamatovat na to, že webová stránka s sebou nese jak jiné typy informací (odkazy na jiné stránky, multimédia, reklamu), tak i vizuální strukturu a metadata, které mohou být užitečná pro následné úlohy.

3.1 Reprezentace textového obsahu

Jelikož velké množství požadované informace z webových stránek tvoří textový obsah, zaměříme se nejprve na vhodnou reprezentaci textu. Obecně se tématem zpracování přirozeného jazyka zabývá obor *Natural Language Processing* (NLP), jehož techniky slouží pro analýzu a prezentaci přirozeně se vyskytujících textů za účelem dosažení vhodného zpracování jazyka pro řadu úloh a použití [32]. Přirozeně vyskytující se texty jsou takové, které jsou použity v komunikaci mezi lidmi a nejsou upraveny pro další analýzu. Dá se tedy říct, že přirozeně vyskytujícími se texty jsou i ty, které se nacházejí na webových stránkách prezentovaných uživatelům. Mezi úkoly, ke kterým se NLP používá, patří získávání informací, strojový překlad, odpovídání na otázky nebo právě získání vhodné reprezentace textu pro další zpracování.

Textová data lze charakterizovat jako semi-strukturovaná. Rysy textových dat jsou vysoce dimenzionální a řídké. Mezi nejčastější rysy texty patří znaky, slova a termy (slova nebo jejich spojení). Metody reprezentace textového obsahu často požadují různé druhy předzpracování vstupního textu. K předzpracování často patří čištění textu (např. odstranění nežádoucích znaků), tokenizace (segmentace textu na tokeny) nebo padding (doplnění vstupních textů pro zajištění fixní velikosti).

Jazykový model (anglicky Language Model) je sbírka statistických a pravděpodobnostních technik, které jsou použity pro reprezentaci slov. Jednoduché modely používají přímé indexy pro reprezentaci slov. Tento přístup ale nemusí být efektivní kvůli velkému množství slov, které může být potřeba reprezentovat. Obecnější přístup je založen na *word embeddingu*, kde je každé slovo reprezentováno v určité fixní dimenzi, která umožňuje zapouzdření

vztahů mezi slovy. *Word embedding* je tedy postup získání vektorové reprezentace textu, kde se jednotlivá slova vyjádří jako vektor s hodnotou z předem definovaného vektorového prostoru.

Základní jazykové modely

Bag of words je základní jazykový model pro reprezentaci textových dat. Text se reprezentuje jako multi-množina slov, nehledě na gramatiku a pořadí slov. Uchovává se pouze informace o četnosti slov v textu. Tuto reprezentaci je možné použít pro sumarizaci, klasifikaci, shlukování nebo asociaci. Nevýhodou přístupu je ztráta informace o pořadí slov a tedy i ztráta kontextu [1].

Dalším možným způsobem je reprezentace každého slova z předem definovaného slovníku nebo lexikální databáze. Příkladem může být anglická lexikální databáze WordNet [13], která obsahuje přes 130 000 slov. WordNet zachycuje různé sémantické vztahy mezi slovy jako jsou například synonyma, antonyma atd. Podobná databáze existuje pro český jazyk pod názvem Český WordNet [48]. Nevýhodou tohoto přístupu je závislost na konkrétním jazyce.

Statistické jazykové modely přiřazují pravděpodobnost sekvenci slov na základě podmíněné pravděpodobnosti slov sekvence vzhledem k jejich předchůdcům. Předchůdci představují kontext daného slova. *N-gram* model je specifický statistický jazykový model, který nevyužívá celý kontext ale předpokládá, že slovo záleží jen na daném počtu jeho předchůdců. Nevýhodou je, že tento model umí efektivně reprezentovat pouze slova, která byla použita pro trénování modelu. Zároveň jsou tyto modely omezeny při modelování na velkém množství dat [23].

Jazykové modely založené na neuronových sítích

Jazykové modely založené na neuronových sítích (anglicky Neural Network Language Models, NNLM) používají neuronové sítě k modelování jazyka ve spojitém prostoru pomocí distribuované reprezentaci slov [23].

Reprezentace mohou být bezkontextové nebo kontextové. Kontextové reprezentace generují reprezentaci slova, která je založena na ostatních slovech ve větě. K základním kontextovým modelům patří CBOW a Skip-gram, které pracují s pevně daným okolním kontextem pro predikování dalšího slova [41]. Kontextovou reprezentaci je možné získat i například pomocí enkodéru neuronové architektury *Transformer* [61]. Ta je založena na *attention* mechanismu, který se aplikuje pro shromáždění relevantního kontextu k reprezentaci slova.

Bezkontextové reprezentace při reprezentaci slova neberou v potaz další slova větě. Mezi bezkontextové modely patří například *word2vec* [42] nebo *GloVe* [51]. Tyto modely využívají neuronové sítě pro naučení se slovních asociací. Přístupy modelující text jako sekvenci znaků nebo slov jsou typicky založeny na rekurentních neuronových sítích. Dalším možným přístupem je text modelovat jako distribuci slov v daném prostoru pomocí konvolučních neuronových sítí.

BERT

Příkladem předtrénovaného jazykového modelu založeného na Transfrormer architektuře je BERT (Bidirectional Encoder Representations from Transformers), který se řadí mezi obousměrné kontextové modely [12]. Obousměrné kontextové reprezentace reprezentují slovo jak vzhledem k předchozímu kontextu, tak vzhledem ke kontextu následujícímu. Jednosměrné

kontextové reprezentace slovo reprezentují pouze k levému nebo pouze k pravému kontextu. Výsledkem jsou předtrénované modely, které byly trénovány na velkém korpusu prostého neanotovaného textu. Některé předtrénované modely je možno použít pro reprezentaci textu i bez dalšího dotrénování, jelikož generují reprezentace textu pro obecné použití. S poměrně menším množstvím dat je také možné tyto modely doladit pro specifičtější úlohy.

Na technologie generující *word embedding* navazují další technologie generující *sentece embedding*, který namísto jednoho slova reprezentuje celou větu. Příkladem může být *Sentence-BERT* [54], jenž je výsledkem přetrénování BERT modelu pro úlohu reprezentace celých vět. *Sentence-BERT* dokáže zpracovat větu jako celek a zachytit její sémantiku a kontext v rámci textu pomocí *sentece embeddingu* fixní velikosti. Mezi předtrénované modely patří i vícejazyčné modely, které generují podobné výstupy pro více než 50 jazyků [55].

3.2 Document Object Model

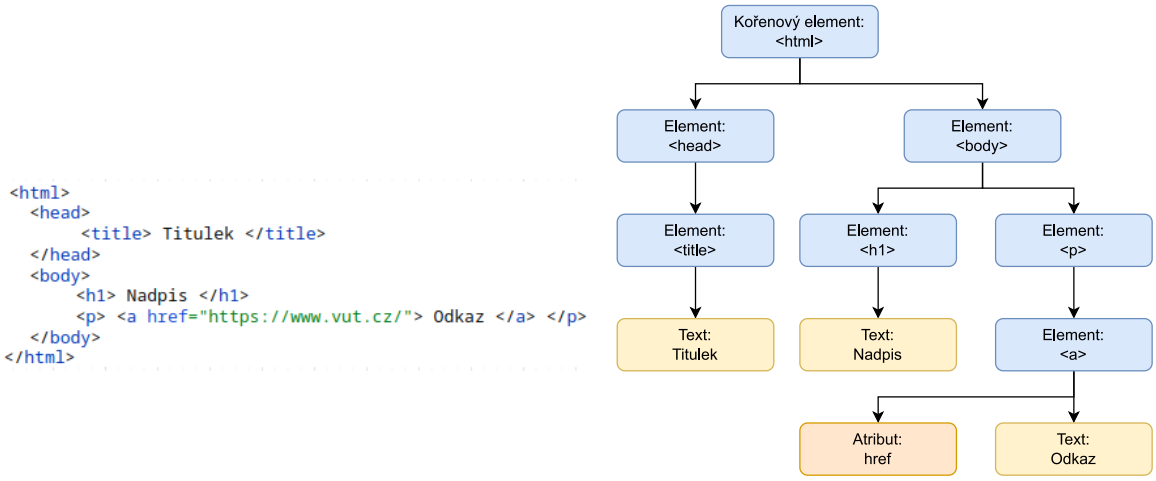
DOM (Document Object Model) je standardní způsob reprezentace webové stránky. Jedná se o stromovou strukturu získanou z HTML kódu webové stránky [47]. DOM strom je možné dynamicky transformovat pomocí JavaScript kódu a rozšířit z CSS. Webové prohlížeče z HTML kódu vytvoří DOM strom, který poté slouží jako reprezentace pro manipulaci a renderování.

Základním prvkem DOM stromu je **uzel** (anglicky *node*). Každý uzel (kromě kořenového *<html>* uzlu) má svého rodiče a může mít seřazenou sekvenci potomků. Pokud uzel žádné potomky nemá říká se mu **list** (anglicky *leaf*). Speciálním případem listu je **textový fragment**, který obsahuje prostý a dále nestrukturovaný text. **Textový uzel** je takový, který obsahuje textový fragment. Textový fragment je možné dále dělit na **tokeny**. Existuje více způsobů rozdělení textového fragmentu na tokeny, například dělení pomocí mezer a interpunkčních znamének. Všechny ne-textové uzly mají **HTML značku** (anglicky *tag*), která představuje sémantický význam uzlu. Značka se zapisuje do úhlové závorky, např. *<htlm>* pro kořenový uzel. Každý ne-textový uzel také může mít sadu **HTML atributů**. Některé atributy ovlivňují renderování uzlů (např. *style* nebo *class*). V rámci kontextu extrakce informací z webových stránek budeme uzel obsahující požadovanou informaci nazývat cílový uzel. Element je specifickým druhem uzlu, který je reprezentován HTML značkou. Na obrázku 3.1 je vidět příklad jednoduché HTML stránky a z ní vytvořeného DOM stromu.

3.3 Vizuální reprezentace

Při renderování stránky prohlížečem není použitý pouze HTML kód. Rozložení a další vizuální vlastnosti lze určovat pomocí CSS. V neposlední řadě je možné pomocí JavaScript kódu modifikovat DOM strom před jeho samotným vykreslením uživateli. Existují i webové aplikace, které používají pouze JavaScript pro vykreslení obsahu (např. SPA – Single-Page Application). Z těchto důvodů je zpracování finálního zobrazení stránky uživateli složité.

Vizuální reprezentace se získá stažením a vyrenderováním stránky ve virtuálním prostředí. Díky tomuto přístupu se získá přesné rozložení stránky tak, jak bylo míněno pro uživatele. Poté se identifikují bloky a pro každý blok se extrahuje sada vlastností včetně velikosti a pozice bloku, obsaženého textu, použitého fontu, velikosti řádku, barvy, zarovnání atd. Pro každý blok je možné získat i přes 300 různých CSS vlastností [66]. Velikost a




```
<html>
  <head>
       <title> Titulek </title>
  </head>
  <body>
       <h1> Nadpis </h1>
       <p> <a href="https://www.vut.cz/"> Odkaz </a> </p>
  </body>
</html>
```

Obrázek 3.1: Příklad DOM stromu.

pozice bloku může být popsána například čtveřicí (x, y, w, h), kde x a y jsou souřadnice, w šířka a h je výška bloku.

Rozdílem mezi vizuální reprezentací a textovou reprezentací je to, že mezitím co textová reprezentace pracuje pouze se samotným textem zobrazeným na webu, vizuální reprezentace bere v úvahu DOM strom, další typy obsahu, meta-informace neprezentované uživateli a rozložení stránky.

Oproti použití reprezentace pomocí DOM stromu je výhodou získání nových informací o pozici bloku ve vyrenderované stránce. Získají se také informace o blízkosti jednotlivých bloků. Teoreticky je možné, aby v DOM reprezentaci byly dva uzly na zcela nesouvisejících místech ve stromu, ale na výslednou obrazovku se pomocí CSS vykreslily vedle sebe. Vizuální reprezentace by měla být oproti té interní konzistentnější.

Výhodou je získání vizuálních informací, které jsou užitečné pro další úlohy nad webovou stránkou. Nevýhodou je vyšší výpočetní složitost způsobená nutností lokálního renderování webové stránky ve virtuálním prohlížeči.

Segmentace stránek

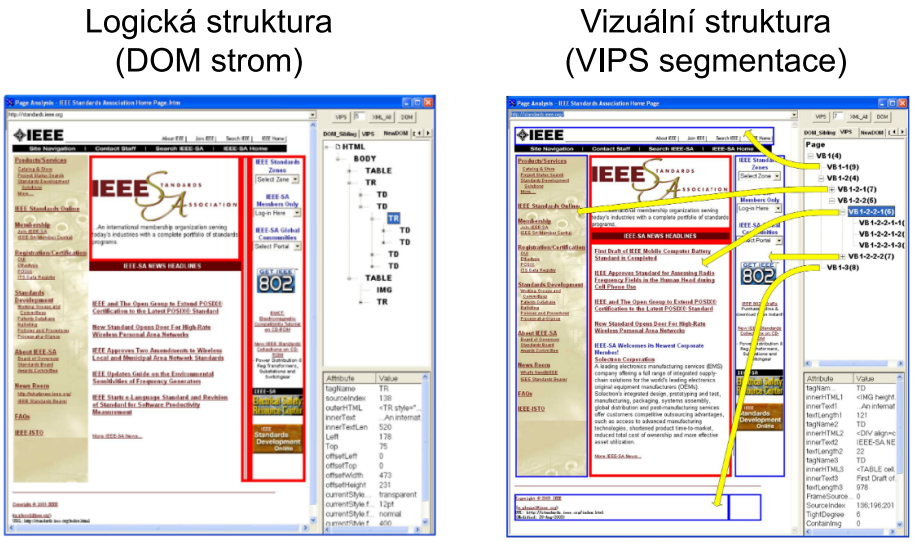
Webová stránka zpravidla neobsahuje jen samotné důležité informace. Často se na stránce nachází bloky určené pro navigaci, reklamu nebo interakci s uživatelem. Pro účely dolování informací jsou tyto části méně důležité.

Logická struktura webové stránky je založena na blocích stánky a jejich hierarchii (jeden blok logicky následuje druhý, který v sobě obsahuje další bloky a podobně). Tato struktura lze získat z DOM stromu (viz kapitola 3.2). Problémem je, že DOM strom je založen spíše na obsahu a nemusí vždy odrážet sémantickou strukturu webového dokumentu.

Vizuální struktura je důležitá informace webové stránky, která se skládá z vizuálních efektů vytvořených různými technikami, formáty, oddělovači, barvami, obrázky, prázdnými oblastmi atd. Je důležitá především pro detekci sémantických oblastí stránky. Ukázka srovnávající vizuální strukturu získanou pomocí VIPS algoritmu a logickou strukturu z DOM stromu je zobrazena na obrázku 3.2.

Příkladem algoritmu, který bere v potaz vizuální strukturu, je právě zmíněný VIPS (*a Vision-based Page Segmentation Algorithm*) [4]. Algoritmus je založen na faktu, že v mnoha případech jsou bloky webových stránek odlišeny pomocí vizuálních vlastností jako je pozice,

vzdálenost, font nebo barva. Výstupem algoritmu je sémantická struktura stránky založená na vizuální prezentaci ve formě stromu, kde každý uzel stromu odpovídá jednomu bloku stránky. Každý uzel má hodnotu stupně koherence, která ukazuje složitost struktury bloku. U algoritmu je možné nastavit požadovaný stupeň koherence pro dosažení požadované granularity.



Obrázek 3.2: Srovnání logické struktury webové stránky získané pomocí DOM stromu a vizuální struktury získané pomocí VIPS algoritmu [4].

Výsledek segmentace lze využít mimo jiné i pro klasifikaci bloků. Této úloze se věnuje následující kapitola 4. Pro získání bloků se webová stránka nejprve vyrenderuje a segmentuje. Poté probíhá klasifikace vizuálních bloků stránky na základě vizuálních a jiných vlastností. Jelikož je výsledkem segmentace stromová struktura, je také možné využít informace o vztazích mezi bloky, například pomocí grafových neuronových sítí.

3.4 Další způsoby reprezentace

V následující sekci si ukážeme některé další způsoby reprezentace webových stránek, které je možné použít jako vstup pro strojové učení. Mezi takové patří kontextové vlastnosti webové stránky, textové mapy a screenshoty.

Kontextové vlastnosti

Kontextové vlastnosti vznikají kombinací individuálních vlastností na úrovni uzlů nebo bloků. Pro každý uzel nebo blok se vezmou vlastnosti jeho sousedů a pomocí funkce se agregují a spojí s individuálními vlastnostmi uzlu/bloku pro vytvoření výsledného vektoru kontextuálních vlastností. Různé reprezentace pomocí kontextových vlastností se liší výběrem sousedů a agregační funkcí.

Výběr sousedů většinou probíhá pomocí předem vytvořených pravidel. Například je možné definovat sadu uzlů o určitém počtu, které se vyberou podle jejich blízkosti k cílovému uzlu v DOM stromu [65]. Vlastnosti sousedních uzlů se pak spojí dohromady k získání kontextuálních vlastností. Další možný způsob spočívá v tom, že se pro každý blok stránky nalezne jeho předcházející text na základě vizuální blízkosti [20].

Příkladem může být použití textových uzlů a rekurentních neuronových sítí k získání kontextuálních informací každého uzlu z jeho sousedů [36]. K získání této informace je použit *attention* mechanismus, který se aplikuje na všechny textové fragmenty z webové stránky. Obecně je možné k získání kontextových vlastností použí grafové neuronové sítě, které jsou detailněji popsány v sekci 4.3.

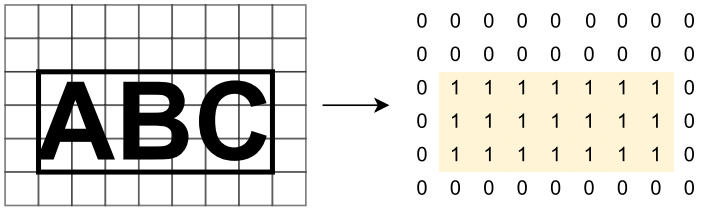
Textové mapy

Textové mapy (prostorové bag of wods) jsou způsob zakódování textového obsahu webové stránky. Oproti klasickému *bag of words* modelu pro reprezentaci textového obsahu (viz 3.1) bere v potaz přesnou pozici slov na webové stránce.

K tvorbě textových map se nejprve pro textové uzly DOM stromu vypočítá jejich ohraničující pole. Stránka se rozdělí pomocí mřížky, díky které je možné uložit text jako řídký tenzor[1]. Tento tenzor má tři dimenze: velikost slovníku (tato dimenze reprezentuje index konkrétního slova), poměr velikosti výšky stránky a buňky mřížky, a poměr velikosti šířky stránky a buňky mřížky. Pro každé slovo textu se přidá 1 do tenzoru stránky, která indikuje buňky zaplněné textem.

Nevýhodou textových map je vysoká komprese textové informace. Ztrácí se tak informace o uspořádání slov, přesné pozici jednotlivých písmen slova a jejich přesné identitě. Na druhou stranu poskytuje kompaktní reprezentaci textu, která je stejná jako reprezentace obrazových dat a dovoluje tak kombinovat textové i obrazové informace. Tato reprezentace je tedy vhodná pro použití konvolučních sítí [16].

Příklad převodu textu do textových map je ukázán na obrázku 3.3.



Obrázek 3.3: Ukázka převodu textu do reprezentace textových map [16].

Screenshoty

Podobně jako vizuální reprezentace cílí na výslednou podobu webové stránky prezentovanou uživateli. Stránka se nejprve vykreslí (podobně jako u vizuální reprezentace), a pak se zachytí snímek vykreslené webové stránky (screenshot), který slouží jako reprezentace pro další analýzu a zpracování.

Nevýhodou je velká ztráta informace. Textové a další strukturované informace jsou nedílnou součástí webové stránky a mohou být užitečné pro analýzu. Zároveň se ztrácí všechna data a metada, která nejsou přímo zobrazována uživateli. Screenshoty ale mohou být užitečnou doplňkovou informací a lze je použít pro některé druhy úloh, například používající konvoluční neuronové sítě [16].

[1] Tenzor je zobecněný vektor, který může mít více indexů.

3.5 FitLayout

FitLayout/2 [43] je rozšiřitelný framework pro renderování, modelovaní a analýzu webových dokumentů. Poskytuje Java API pro CLI (*rozhraní pomocí příkazového řádku)* nezávislé na platformě a REST API, které dovoluje implementovat kompletní postup zpracování webové stránky. Pomocí frameworku je možné popsat webové stránky tak, jak byly vyrenderovanány webovým prohlížečem (screenshot), ale i detailním popisem přesné pozice a vizuálních vlastností každého textového nebo jiného elementu.

Proces zpracování dokumentů

V následující sekci jsou popsány jednotlivé části nástroje. Dohromady jsou tyto části použity pro automatickou analýzu webových dokumentů.

Nástroj poskytuje jednotný model pro renderování reprezentace webové stránky v Javě. Model popisuje webovou stránku na úrovni jednotlivých bloků generovaných renderovacím enginem. Model je nezávislý na formátu zdrojové stránky a je tedy vhodný pro další analýzu (např. pro vytvoření datové sady pro strojového učení).

Renderovací enginy jsou součástí nástroje a je možné si vybrat ze dvou variant. První z variant je založena na webovém prohlížeči Chromium a dovoluje renderovaní jakékoliv stránky včetně komplexních a dynamických stránek obsahujících JavaScript. Nevýhodou je potřeba dodatečných závislostí obsahujících právě Chromium prohlížeč. Druhá varianta další závislosti nevyžaduje jelikož se jedná o čisté Java renderování HTML+CSS a PDF dokumentů. Pro kratší dokumenty může být tato varianta rychlejší ale nepodporuje dynamické stránky s JavaScriptem a komplexní CSS rozvržení stránek.

Nástroj také poskytuje segmentační algoritmy nad renderovanými stránkami, které umožňují stránku segmentovat a popsat prostorovým stromovým modelem. Dostupné jsou následující algoritmy: VIPS (*Vision-based page segmentation*), BCS (*Block clustering segmentation*) a vlastní algoritmus pro seskupování vizuálních oblastí, který poskytuje základní, ale konfigurovatelnou metodu segmentace.

Pro uložení renderovaných stránek, výsledků segmentací a dalších artefaktů je použito RDF úložiště. Renderované stránky a výsledky segmentací jsou popsány pomocí ontologií, a je tedy možné je ukládat a sdílet. Díky tomu je možné stránky znovu analyzovat bez nutnosti opakovaného zpracování. Další výhodou je jednoduchá anotace částí stránek a vyhledávání obsahu stránky pomocí dotazovacího jazyka SPARQL.

Artefakty

Výsledky jednotlivých kroků zpracování je možné uložit do centrálního RDF repozitáře, který obsahuje jeden RDF graf. Tento graf se dělí do pojmenovaných podgrafů, kterým se říká artefakty. Každý artefakt reprezentuje konkrétní webovou stránku na určité úrovni abstrakce. Artefakt může nabývat různých typů, a podle toho se odráží i obsahové elementy, které zahrnuje.

Prvním krokem zpracování webové stránky je typicky její renderování. Výsledkem tohoto kroku je artefakt *Page*, který reprezentuje stránku tak jak byla vykreslena webovým prohlížečem. V rámci úrovně abstrakce se jedná o fyzickou vrstvu. Page se skládá ze stromu obsahových elementů, kterým se říká *boxy*. Box reprezentuje obdélníkové oblasti generované pomocí zdrojových DOM elementů. Pro každý box jsou uloženy detailní informace o jeho obsahu a vizuálních vlastnostech. Mezi tyto informace patří například přesná pozice ve

stránce, použitý font, barva, text atd. Podobné informace jsou uloženy i pro celkový Page artefakt, včetně bitmapového snímku.

Další kroky zpracování, mezi které patří například segmentace, vytváří nové pohledy na zpracovávanou stránku. Jelikož nové pohledy souvisí s určitou logickou interpretací částí obsahu, jsou artefakty produkované těmito kroky z hlediska abstrakce na logické úrovni. *AreaTree* artefakt modeluje důležité vizuální oblasti stránky včetně jejich hierarchie. Každá část je reprezentována obsahovým elementem *Area*, který má podobné vlastnosti jako box z Page artefaktu. Význam Area elementů je ale více obecný. Tyto části mohou být získány prostým filtrováním vizuálně odlišitelných bloků nebo aplikováním segmentačních algoritmů. V případě použití segmentačního algoritmu jsou hranice jednotlivých částí a jejich organizace ve stromu závislé na konkrétní použité segmentační metodě a jejím nastavení. Každá vizuální oblast může v tomto případě obsahovat jakýkoliv počet boxů. Výsledné AreaTree je možné získat z Page artefaktu nebo zpracováním jiného AreaTree.

Artefakt *ChunkSet* reprezentuje stránku jako sadu *text chunků*. Text chunk je souvislá část textu dokumentu, která je v určitém smyslu specifická. Typicky se jedná o rozpoznání dat, použití regulárních výrazů nebo rozpoznání pojmenovaných entit. Hranice text chunků není limitována hranicemi boxů nebo vizuálních částí. Text chunk může být částí boxu, ale také může překrývat více boxů.

LogicalAreaTree je artefakt, který poskytuje nejabstraktnější pohled na stránku. Skládá se z logických částí, kde každá část obsahuje libovolný počet vizuálních částí nebo text chunků, které spolu tvoří logickou entitu. Příkladem může být tělo článku, který se skládá z více paragrafů, které představují oddělené vizuální části. Hrany stromu reprezentují vztahy mezi logickými částmi, které mají určitou sémantickou interpretaci.

Reprezentace

Reprezentace vyrenderované stránky je založena na modelu definovaného CSS specifikací. Tato specifikace definuje box jako základní jednotku prezentace obsahu dokumentu a dále způsob jakým jsou boxy získány ze vstupních DOM elementů. Boxy jsou organizovány ve stromu, který vždy obsahuje jeden kořenový box představující celou stránku. Listové uzly jsou nejmenšími atomickými částmi obsahu stránky. Strukturou je tento strom podobný DOM stromu, ale jeden HTML element může generovat žádný nebo více boxů a vztahy mezi boxy nemusí nutně odpovídat vztahům DOM elementů.

Reprezentace na logické úrovni je strukturována také pomocí stromu. Pouze text chunky náležící do jednoho ChunkSet artefaktu se vyjádří jako jednoduchá kolekce. Každý logický element může mít určitý počet *tagů*, které jsou použity pro kategorizaci obsahových elementů.

Architektura

Architektura FitLayout frameworku je rozdělena do více softwarových balíčků. Základní funkcionalitu, včetně datových modelů, úložiště artefaktů, implementací algoritmu a CLI se nachází v Java knihovně *FitLayout core*. *FitLayout server* poskytuje serverovou aplikaci s REST rozhraním pro přístup k uloženým artefaktům a volání implementovaných algoritmů. *Page View* je klientská webová aplikace s grafickým uživatelským rozhraním pro práci pomocí webového prohlížeče. Aplikace komunikuje se serverem pomocí REST API. Oddělená JavaScript aplikace *fitlayout-puppeteer* obsahuje Chromium prohlížeč a dovoluje renderování webových stránek a získání požadovaných dat.

FitLayout core knihovnu je možno použít přímo pro implementaci Java aplikací pro zpracování webových stránek, ale poskytuje také rozhraní pomocí příkazového řádku. K využití Chromium prohlížeče pro renderování je třeba doplnit knihovnu o balíček fitlayout-puppeteer. FitLayout server a PageView spolu tvoří klient-server webovou aplikaci, kterou je možné ovládat interaktivním způsobem pomocí grafického uživatelského rozhraní.

Kapitola 4

Metody strojového učení pro extrakci informací z webu

V následující kapitole se seznámíme s technologiemi v oblasti extrakce obsahu webových dokumentů. Jedná se tedy o kategorii web content mining. Detailněji se zaměříme na využití strojového učení, jehož největší výhodou je flexibilita řešení a nezávislost na konkrétním webovém místě.

Web content mining se dělí na dolování ze strukturovaných a nestrukturovaných dat. Při strukturovaných datech může být vstupem například seznam produktů nebo služeb. U tohoto přístupu je možným řešením napsání wrapperu založeného na formátovacích vzorech stránky. Nevýhodou je omezená funkčnost na různých webových místech, protože pro plně automatický přístup je třeba počítat s pevnou šablonou webové stránky. U dolování nestrukturovaných dat je možné na webové stránky pohlížet například jako na textové dokumenty nebo jako na snímky obrazovky. U tohoto přístupu se používají metody strojového učení. Většina dostupných údajů v reálném světě má nestrukturovanou podobu. Průzkumy ukazují, že až 90 % informací ve většině organizací jsou nestrukturovaná data [52]. Proto je vhodné s nestrukturovanými daty, mezi které patří i webové stránky z různých webových míst, umět pracovat.

Cílem automatizované extrakce je pomoci uživatelům zjednodušit zdlouhavé úlohy jako je vyplňování formulářů a hledání informací on-line. Automatizovaná extrakce dovoluje přeskočit manuální navigaci webem.

4.1 Strojové učení

Strojové učení (anglicky *machine learning*) je podoblast umělé inteligence, která se zabývá algoritmy, metodami a technologiemi, které umožňují takovou změnu interního stavu počítačového systému, která dovoluje přizpůsobit se systému změnám okolního prostředí. Učením se v kontextu strojového učení myslí využití dat k zlepšení výkonu pro určité úkoly. Obecně řečeno se počítačový program učí ze zkušenosti E vzhledem k nějaké třídě úloh T a metrice výkonnosti P, pokud se jeho výkon na úkolech z T zlepšuje se zkušeností E měřeno metrikou P [44]. Pro extrakci informací z webu může být úkolem klasifikace bloků webového dokumentu, metrikou přesnost klasifikace a zkušeností datová sada anotovaných webových dokumentů.

Základním dělení algoritmů strojového učení je podle formátu vstupních dat. Učení s učitelem (*supervised learning*) používá jako vstup data, u kterých je správně určen jejich

požadovaný výstup. U klasifikace je to většinou anotovaná třída. Naopak u učení bez učitele (*unsupervised learning*) není znám požadovaný výstup ke vstupním datům [59]. V oblasti extrakce informací z webu se zpravidla využívá metody strojového učení s učitelem.

Mezi aplikace strojového učení patří například hodnocení webových stránek pro vyhledávání relevantních stránek, automatický překlad, rozpoznání tváře, rozpoznání hlasu nebo rozpoznání pojmenovaných entit. Všechny uvedené příklady je možné shrnout jako problém klasifikace, který je i předmětem této práce. Mezi další úlohy strojového učení patří regrese, shlukování a detekce odlehlých hodnot.

Klasifikace je proces řazení dat do jistých tříd na základě jejich vlastností. U klasifikace je obecně konečný počet výstupních tříd. Klasifikace se dělí na binární a vícetřídní. Binární klasifikace rozřazuje vstupní data do dvou možných tříd. Vícetřídní klasifikace je rozšíření binární, kde počet možných výstupních tříd je větší než dva. Obvykle se proces klasifikace dělí na dvě fáze. V první fázi se učí model na trénovací datové sadě. Trénovací datová sada musí být anotována, musí tedy být známa požadovaná třída pro každý vzorek této datové sady. Vzorky trénovací datové sady jsou vstupem klasifikačního modelu, jehož úkolem je naučit se klasifikační pravidla, pomocí kterých může vzorek klasifikovat do správné třídy. V druhé fázi se testuje vytvořený model z první fáze. K tomuto úkolu se používá testovací datová sada. Tato datová sada je také anotována, ale neměla by se používat při učení. Průnik trénovací a testovací datové sady by tedy měl být prázdný. Model se tedy snaží klasifikovat vzorek z testovací datové sady do třídy podle pravidel naučených v první fázi. Na základě predikce modelu a skutečné správné třídy vzorku se určí přesnost modelu. Výsledné metriky určují schopnost modelu klasifikovat nová neviděná data.

Rozpoznání pojmenovaných entit (anglicky *named entity recognition*) je úkol klasifikace, který se zabývá identifikací entit jako jsou místa, názvy, jména, akce atd. z dokumentů. V případě extrakce informací o produktech z webových stránek by byly hledané entity např. název, popis, cena a obrázek produktu. Použití metod strojového učení oproti ručně psaných pravidel může vést k zlepšení výsledků, hlavně při úlohách požadujících jazykovou nezávislost.

4.2 Existující technologie

V následující sekci se seznámíme s technologiemi pro extrakci informací z webu. Nejprve si představíme klasické nástroje pro získání strukturovaných informací z webu, poté technologie využívající textové informace webové stránky. Následující sekce 4.3 se důkladně věnuje využitím technologie neuronových sítí.

Web wrappery

Nástrojům pro extrakci strukturovaných informací z webových stránek se říká *web wrappery* [30]. Cílem wrapperu je ze semi-strukturovaného vstupu (obvykle HTML) získat výstup v předem definované a strukturované formě.

Wrappery se obvykle definují pomocí sad deklarativních pravidel. Například se může jednat o CSS selektory nebo XPath pravidla. Pravidlo by mělo jasně identifikovat cílové DOM uzly obsahující požadovanou informaci a to pro všechny dokumenty se stejnou strukturou.

Wrappery lze vytvářet manuálně ale nevýhodou tohoto přístupu je potřeba prozkoumat zdrojový kód stránky, a kvůli tomu může být tento způsob velmi pracný. Wrapper induction generuje wrappery automaticky z anotované datové sady [30]. Takové wrappery ale stále

pracují pouze na jednom webovém místě, a je třeba vytvářet novou datovou sadu a znovu učit wrapper při změně rozložení webových stránek na webovém místě.

Nevýhodou wrapperů je potřeba adaptace na konkrétní šablonu stránky a tedy úzké navázání na specifickou sadu webových stránek. Navíc se rozložení webových stránek může v průběhu času měnit, což způsobuje potřebu pravidelných úprav a aktualizací wrapperů. Použitím metod strojového učení (například neuronových sítí) vede k možnosti naučení modelu extrahovat informace i z předem neviděných šablon a tedy nezávisle na konkrétním webovém místě. Další nevýhodou web wrapperů je velké množství lidské práce potřebné pro vytvoření a případnou aktualizaci wrapperu.

TF-IDF

Následující metoda lze použít k analýze webových dokumentů pomocí vektoru termínů dokumentu a jejich vah. Váhy určují vzdálenost mezi termíny. Tato se používá pro extrakci informací založenou pouze na textovém obsahu webové stránky.

Váhy jsou vypočítány pomocí mechanismu TF-IDF (*Term Frequency-Inverse Document*) [57]. TF-IDF přiřazuje každému termínu z dokumentu váhu na základě počtu výskytů v dokumentu. Avšak ne všechny termíny mají vliv na určení relevance dokumentu. Proto se váha snižuje pomocí takzvané inverzní dokumentové frekvence, která bere v potaz počet výskytů daného termínu v kolekci všech dokumentů.

Další možností je použití Longest Common Sub-sequence (LCS) algoritmu. Algoritmus také pracuje s podobností pomocí společné subsekvence mezi termíny. Čím větší je *skóre* dané tímto algoritmem, tím víc jsou si termíny podobné. Tento přístup může pomoci k jazykové nezávislosti, jelikož segmentace slov se liší jazyk od jazyka. LCS je možné použít i pro shlukování bloků v rámci jedné webové stránky [66].

Sémantické grafy

Sémantické grafy nabízí kompaktní a strukturovanou reprezentaci konceptů z webových dokumentů. V potaz se berou sémantické informace, díky čemuž je možné vytvořit mapu sémantických částí dokumentu a jejich vztahů vzhledem ke konkrétnímu konceptu nebo termínu, který se nazývá *cílové slovo* [66]. Tato metoda stejně jako TF-IDF také pracuje pouze s textovým obsahem webové stránky.

Sémantické váhy ukazují jak moc je dokument relevantní vzhledem k cílovému slovu. Sémantický graf je tedy neorientovaný, plně propojený graf, skládající se z termínů dokumentu propojených vztahem ukazujícím podobnost k cílovému slovu.

Výpočet sémantického vztahu začíná prvním termínem (cílovým slovem). Poté se vypočítá podobnost mezi termíny pomocí lexikální databáze (například WordNet[13]). Poté se termíny více podobné cílovému slovu seřadí pomocí vhodně zvolené metriky. Předem vybraný počet nejlepších termínů se použije jako vrcholy grafu. Pro každý pár vrcholů se vytvoří hrana. Váha hrany je dána podle sémantické podobnosti termínů reprezentující vrcholy. Váha je obvykle reprezentována lineárně v intervalu <0,1> [34].

Informaci obsaženou v sémantickém grafu je možné reprezentovat jednou hodnotou, které se říká *sémantická váha*. Tato hodnota se vypočítá součtem rekonstruovaných vah hran, které patří do Minimum Spanning Tree grafu. Sémantická váha slouží k určení zda je dokument (reprezentovaný sémantickým grafem) relevantní vzhledem k cílovému slovu. Čím vyšší je sémantická váha, tím blíže je dokument tématu cílového slova.

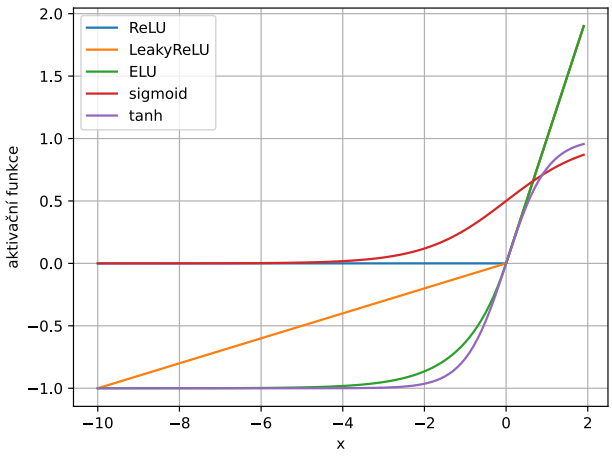
Sémantické grafy i TF-IDF nalézají vhodné použití v oblasti kategorizace webových dokumentů, kde se neextrahují informace z jedné webové stránky, ale webové stránky se porovnávají mezi sebou s cílem kategoricky odlišit skupiny webových stránek [66].

4.3 Neuronové sítě

Neuronovými sítěmi se zabývá disciplína hlubokého učení (anglicky *deep learning*), která spadá pod strojové učení. Neuronové sítě představují model, který se trénuje pro klasifikaci. V kontextu extrakce z webových dokumentů se často jedná o klasifikaci elementů DOM stromu do předem určených tříd, jako je například název nebo cena produktu z webové stránky. Předtrénované jazykové nebo obrazové modely je dále možné využít jako komponenty pro chatboty, samořídící auta, automatický překlad, převod textu na řeč nebo zpracování přirozeného jazyka. V dnešní době popularita a efektivita neuronových sítí roste díky velkému množství dat, a také výkonu grafických karet, na kterých je možné neuronové sítě na velkých datech trénovat [58].

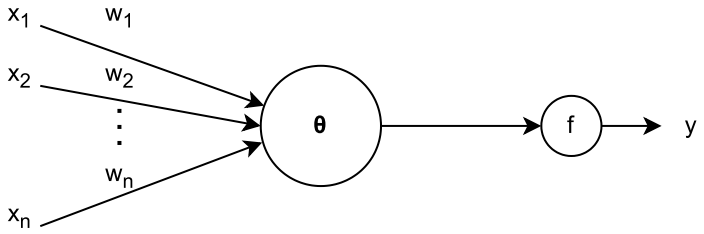
Architektura neuronových sítí

Neuronové sítě jsou založeny na fungování lidských neuronů [14]. Základním stavebním blokem neuronové sítě je neuron. Neuron obsahuje lineární funkci, která je následována nelineární funkcí. Ke každému neuronu dále náleží sada parametrů, které se aktualizují během fáze trénování. Mezi tyto parametry patří váhy a bias [58]. Lineární funkce se obvykle realizuje jako suma součinů jednotlivých vstupů a vah od kterých je následně odečten bias. Nelineární funkci se říká aktivační funkce a zpravidla má skokový nebo spojitý charakter. Aktivační funkce slouží k transformaci výstupu do podoby, která je vhodná jako vstup dalšího neuronu. Mezi typické aktivační funkce patří ReLU [11], leaky ReLU [64], sigmoida nebo hyperbolický tangens. Průběh těchto aktivačních funkcí je zobrazen na obrázku 4.1. Další nelineární funkcí, která se používá v neuronových sítích, je tzv. pooling funkce. Ta slouží k redukci velikosti při zachování dominantních vlastností. Často používanými funkcemi jsou max pooling (vybírá maximální hodnoty) a mean pooling (vybírá průměrné hodnoty).



Obrázek 4.1: Porovnání průběhu aktivačních funkcí.

Označíme-li vstupy neuronu jako $x_1, x_2, \ldots, x_n$, váhy neuronu jako $w_1, w_2, \ldots, w_n$, bias jako θ, aktivační funkci jako f a výstup y, pak je architektura jednoho neuronu neuronové sítě zobrazena na obrázku 4.2.

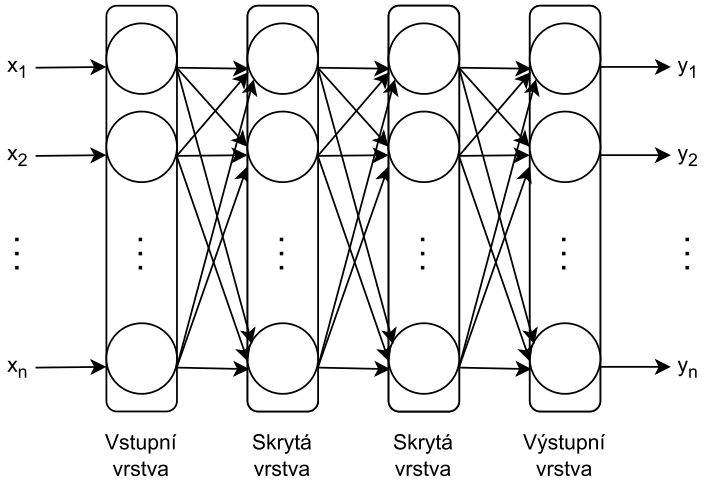


Obrázek 4.2: Architektura jednoho neuronu neuronové sítě.

Výstup jednoho neuronu neuronové sítě se vypočítá podle vzorce 4.1.

$$y = f(\sum_{i=1}^{n}(x_i w_i) - \theta) \tag{4.1}$$

Jednotlivé neurony je možné skládat za sebe a vytvořit strukturu neuronové sítě, která je tvořena ze vstupní vrstvy, skrytých vrstev a výstupní vrstvy. Vstupní data jsou zpracována při postupném průchodu jednotlivými vrstvami neuronové sítě. Tomuto procesu se říká dopředný krok (nebo také *forward pass*). Vícevrstvá neuronová síť, která spojuje každý neuron jedné vrstvy s každým neuronem následující vrstvy, se nazývá vícevrstvý perceptron [56]. Tato architektura se dvěma skrytými vrstvami je zobrazena na obrázku 4.3. Obecně může být skrytých vrstev libovolný počet.



Obrázek 4.3: Architektura vícevrstvé neuronové sítě typu vícevrstvý perceptron.

Na výstup neuronů výstupní vrstvy se často aplikuje funkce *softmax*. Tato funkce normalizuje vektory na rozdělení pravděpodobnosti nad předpokládanými třídami vstupu. Pokud označíme počet požadovaných klasifikovaných tříd jako C, pak se *softmax* funkce pro výstup jednoho neuronu výstupní vrstvy y_i, který reprezentuje třídu i, spočítá jako 4.2.

$$\text{softmax}(y_i) = \frac{e^{y_i}}{\sum_{j=1}^{C} e^{y_j}}, \quad i \in [1, \ldots, C] \tag{4.2}$$

Mezi další druhy architektur neuronových sítí patří například konvoluční, rekurentní nebo grafové neuronové sítě. Konvoluční neuronové sítě využívají konvoluční vrstvy, které aplikují na vstup jeden nebo více konvolučních filtrů [17]. Výhodou této architektury je udržení prostorové informace a často se používají při zpracování obrazu. Rekurentní neuronové sítě v každém kroku přijímají jeden nový vstup a zpětnou vazbu, která je vypočítána z výsledku předešlého kroku [40]. Výhodou je zapamatování si informace z předchozích kroků a kombinace předchozích a současných informací. Mezi nejznámější architektury rekurentních neuronových sítích patří LSTM (Long-Short-Term-Memory), která využívá hradla k rozhodnutí, které informace zahrnout pro další výpočet, a které zapomenout [21].

Grafové neuronové sítě

Další možnou architekturou neuronových sítí jsou grafové neuronové sítě (GNN), které jsou navržené pro práci s daty popsanými pomocí grafů. GNN se používají ke klasifikaci a učení se reprezentace uzlů grafu nebo celých grafů. GNNs se dělí na rekurentní, konvoluční a attention-based. GNN použité pro učení se reprezentace se nazývají *embeddery* a trénují se například pro vytvoření reprezentací webových elementů [22]. Grafové neuronové sítě jsou vhodné pokud je možné reprezentovat vstupní data jako graf. Velké množství dat je možné reprezentovat pomocí grafů. Patří mezi ně například molekuly, dopravní nebo sociální sítě a se správným předzpracováním i webové stránky.

Graf G lze popsat dvojicí $G = (V, E)$, kde V je množina uzlů a E množina hran takových, že $E \subseteq V \times V$. Jedna hrana specifikuje pár uzlů z V, které jsou propojeny. Každý uzel u z V má vektor vlastností $x_u \in \mathbb{R}$ [62]. V některých případech jsou připojeny i vlastnosti k hranám grafu, ale v této práci se pracuje pouze s vlastnostmi uzlu. Samotné grafové neuronové sítě poté při výpočtu pracují s okolím uzlu. Okolí uzlu u lze definovat jako množinu uzlů $N_u = \{v|(u, v) \in E \vee (v, u) \in E\}$. V závislosti na typu GNN se poté liší způsob s jakým se při výpočtu s okolím uzlu pracuje. U konvolučních neuronových sítí se používají konvoluční vrstvy pro agregaci vlastností uzlu a jeho okolí [25]. Attention-based grafové neuronové sítě využívají attention mechanismus, který dovoluje dynamicky a adaptivně upravovat váhy jednotlivých uzlů z okolí a tím se naučit důležitost vztahů mezi různými uzly grafu [63].

Jak bylo zmíněno, grafové neuronové sítě je možné použít i pro reprezentaci webové stránky. Přístup je založen na šíření vlastností přes hrany v grafu uzlů. Stejně jako u získávání kontextuálních vlastností (viz 3.4) se musí definovat sada hran, funkce pro přenos zpráv mezi uzly a agregační funkce ke spojení informací v uzlu [18]. Pro klasifikaci uzlů v grafových neuronových sítí se využívá kromě vlastností jednotlivých uzlů i lokální a globální informace o struktuře grafu. Grafové neuronové sítě iterativně propagují informace z jednotlivých uzlů do jejich sousedních uzlů, což umožňuje uzlům získávat informace o svém okolí. Při použití klasické sítě typu vícevrstvý perceptron se kontext uzlu nebere v potaz a pracuje se pouze s vlastnostmi jednotlivých uzlů.

Z DOM stromu se často vytvářejí heterofilní grafy. Na rozdíl o homofilních grafů, kde spojené uzly jsou si podobné, v heterofilních grafech mají propojené uzly rozdílné vlastnosti. U takových grafů se často reprezentují uzly a jejich sousedi zvlášť a agregují se informace z více vzdálených sousedů se zachováním všech dílčích reprezentací [67].

Reprezentace webové stránky pomocí GNN je možná více způsoby. Například lze zkonstruovat graf, kde uzly jsou textové uzly a hrany se dělí na horizontální, vertikální a DOM hrany. Horizontální a vertikální hrany jsou mezi uzly, které sdílejí horizontální nebo vertikální polohu a není mezi nimi žádný další textový fragment. DOM hrany jsou mezi uzly

sdílející rodiče v DOM stromu [37]. Další možnou reprezentací, použitou i v této práci, je z výsledného stromu segmentačního algoritmu vytvořit graf, kde hrana reprezentuje vztah rodič-potomek.

Trénování neuronových sítí

Neuronové sítě se trénují na trénovací datové sadě. Trénovací datová sada zpravidla vzniká definováním cílových atributů (například název, popis a cena produktu). V případě webových stránek se jedná o cílový uzel obsahující požadovanou informaci. Následně se provádí anotace těchto atributů ve vstupních datech. Při trénování modelu se aktualizují parametry jednotlivých neuronů s cílem minimalizovat tzv. ztrátovou funkci, která je závislá na vahách neuronů. Pro aktualizaci vah se používá algoritmus zpětného šíření chyby (anglicky *backpropagation*). Tento algoritmus funguje na principu učení s učitelem a je založený na metodě gradientního sestupu. V algoritmu nejprve do neuronové sítě vstupují postupně vzorky z trénovací datové sady a vypočítají se výstupy sítě. Výstupy sítě se porovnají s požadovanými výstupy (trénovací datová sada je anotovaná) a vypočte se chyba pomocí ztrátové funkce. Na základě chyby se vypočítá gradient chybové funkce a provede se sestupný gradientní krok, který spočívá v úpravě vah neuronů tak, aby klesla hodnota chyby. Úprava vah postupuje zpětně od výstupní vrstvy ke vstupní vrstvě. Tento proces aktualizace vah probíhá iterativně ve snaze naleznout optimální hodnoty vah neuronů sítě. Výpočet gradientů je výpočetně náročný, a proto se často backpropagation algoritmus aplikuje na podmnožiny vstupních dat, které se nazývají *batches*. Aktualizace vah tedy probíhá po každé batch. Celá datová sada může být použita pro trénování vícekrát, kde jednomu použití se říká *epocha*. Je vhodné pro každou epochu data v trénovací datové sadě náhodně zamíchat. Iterativní proces se zpravidla zastavuje po zadaném počtu epoch nebo při konvergenci ztrátové funkce.

Pro vícetřídní klasifikaci se obvykle jako ztrátová funkce používá křížová entropie (anglicky *cross-entropy*). Křížová entropie se počítá z normalizovaného vektoru pravděpodobností nad předpokládanými třídami výstupu (tento vektor je možný získat pomocí softmax funkce) a odpovídajícího vektoru požadovaných tříd, který je často ve formátu *one-hot encoding*, tedy na indexu požadované správné třídy je jedna a nuly všude jinde.

Mezi největší problémy u trénování neuronových sítí patří tzv. *overfitting*, který způsobuje přilnutí modelu k trénovací datové sadě a špatné zobecnění pro jiná, dříve neviděná data. Je tedy vhodné model průběžně testovat na nových datech a případně použít regularizační techniky jako je využití a postupné snižování koeficientu učení, který určuje jak moc se mění váhy jednotlivých neuronů. Váhový úbytek je další regularizační technika, kdy se při výpočtu ztrátové funkce pracuje i s hodnotami vah [29]. Dropout s overfittingem bojuje náhodnou deaktivací některých neuronů při trénování neuronové sítě [60]. U trénování se často používají optimalizační techniky k nalezení optimálních řešení vzhledem ke ztrátové funkci. Mezi takové patří například ADAM (Adaptive Moment Estimation), který počítá adaptivní koeficient učení pro každý parametr [24].

Testování neuronových sítí

Pro evaluaci se používá datová sada, která nebyla použita při trénování. Takové sadě se říká testovací datová sada a je také anotována. Model neuronové sítě se na testovací datové sadě testuje, a díky tomu se zjistí jak výkonný je testovaný model na nových a dříve neviděných datech.

Metriky pro úlohu klasifikace

U úlohy klasifikace se pomocí výstupu modelu každý testovaný vzorek dat zařadí do jedné z předem určených tříd. Nejjednodušší metrikou pro evaluaci neuronových sítí pro úlohu klasifikace je přesnost. Ta udává procento správně klasifikovaných vzorků z testovací datové sady, tedy jakému procentu vzorkům z testovací datové sady byla predikována správná třída.

Základem přesnosti a dalších metrik pro evaluaci klasifikace neuronové sítě je kontingenční tabulka. Předpokládejme klasifikaci elementů do tříd. V tabulce 4.1 je znázorněna kontingenční tabulka pro tento úkol. Porovnává se predikovaná hodnota modelu (zda element patří nebo nepatří do dané třídy) a skutečná hodnota z anotovaných dat. Případy **TP** (skutečně pozitivní, true positive) a **TN** (skutečně negativní, true negative) označuje situace, kdy se predikce modelu shoduje se správným stavem. Naopak **FN** (falešně negativní, false negative) a **FP** (falešně pozitivní, false positive) říkají, že třída měla být predikována a nebyla, respektive, že třída neměla být predikována a byla.

		Predikce modelu	
		Element patří do třídy	Element nepatří do třídy
Správná hodnota	Element patří do třídy	**TP**	**FN**
	Element nepatří do třídy	**FP**	**TN**

Tabulka 4.1: Ukázka kontingenční tabulky pro klasifikaci elementů do tříd.

Přesnost (anglicky *accuracy*) se počítá jako součet správných predikcí (TP a TN) dělený součtem všech možných stavů z tabulky (TP, FN, FP a TN), jak je zobrazeno v rovnici 4.3. Nevýhodou je zavádějící vysoká přesnost při třídě s dominantním počtem vzorků. Pokud by například velká většina elementů do třídy nepatřila může model predikovat pouze tento případ a stejně dosahovat vysoké přesnosti.

$$Presnost = \frac{TP + TN}{TP + TN + FP + FN} \tag{4.3}$$

Z těchto důvodů je často používaná metrika F_1. Tato metrika pracuje se senzitivitou (anglicky *recall*) a precizností (anglicky *precision*). Senzitivita (viz 4.4) se vypočítá jako počet skutečně pozitivních (TP) dělený celkovým počtem správných hodnot (TP + FN). Metrika udává jak moc se dá věřit modelu, že zařadil do třídy všechny elementy, které tam opravdu patřily.

$$Senzitivita = \frac{TP}{TP + FN} \tag{4.4}$$

Preciznost (viz 4.5) se vypočítá jako počet skutečně pozitivních (TP) dělený počtem všech, které model predikoval, že patří do třídy (TP + FP). Tato metrika udává jak moc se dá věřit modelu pokud predikoval, že element do třídy patří, tak tam opravdu patřil.

$$Preciznost = \frac{TP}{TP + FP} \tag{4.5}$$

Senzitivita a preciznost jsou obecně nepřímo úměrné. Senzitivita nebere při výpočtu v potaz falešně pozitivní (FP) třídy, preciznost zase nepočítá s falešně negativními (FN)

třídami. Metrika F_1 kombinuje senzitivitu a preciznost pomocí vzorce 4.6 a poskytuje tak kompromis zahrnující obě tyto metriky.

$$F_1 = \frac{2 \cdot Senzitivita \cdot Preciznost}{Senzitivita + Preciznost} \tag{4.6}$$

Metriky pro úlohu nominace

V úloze extrakce informací jde o hledání konkrétních informací ve velkém množství elementů. Pokud víme, že se všechny informace ve velkém množství elementů vyskytují v daném počtu, můžeme místo o klasifikaci mluvit o nominaci. Nominace tedy nepracuje s každým vzorkem dat zvlášť ale vybírá pouze předem daný počet vzorků u kterých testuje, zda do dané třídy patřily. O pojmu nominace v širším kontextu dále pojednává navazující sekce o extrakci informací z webových dokumentů.

Metrika přesnost nominace se zaměřuje pouze na případy, kdy element do třídy patří. Navíc nepočítá s třídou znázorňující absenci informace. Tato metrika říká v jakém procentu případů je model schopen nalézt požadované informace pokud se na stránce nachází. Pokud označíme počet požadovaných informací na jedné stránce jako C a fakt, že model pro informaci i nalezl správný element jako n_i (n_i je 1, pokud model element nalezl, 0 v opačném případě), pak se přesnost nominace pro spočítá jako 4.7.

$$PresnostNominace = \frac{\sum_{i=1}^{C} n_i}{C} \tag{4.7}$$

Metrika prediktivní přesnosti spočítá v nalezení jednoho elementu pro každou požadovanou informaci. Ze všech možných elementů se tedy pomocí výstupů modelu vybere pouze jeden, u kterého se testuje, zda danou informaci měl nebo neměl obsahovat. Prediktivní přesnost se počítá stejně jako přesnost nominace (viz 4.7), rozdílem ale je, že fakt n_i není získán z prozkoumání více elementů, ale pouze jednoho vybraného [22].

Extrakce informací z webových dokumentů

Pro extrakci informací z webových dokumentů je cílem získat modely, které jsou schopny vytvořit kvalitní reprezentace pro jakýkoliv typ webového elementu.

U extrakce informací z webových dokumentů je možné místo pojmu klasifikace použít pojem *nominace*. Klasifikace se zabývá zařazováním vstupních vzorků do různých tříd. Nominace je považována za složitější úlohu. U nominace probíhá porovnávání velkého množství elementů stránky a výběr konečného počtu elementů pro každou třídu [22]. V následující práci je zaměření na trénování modelu pro účely klasifikace, ale dále se bude pracovat s oběma pojmy a budou se zkoumat jak metriky pro klasifikaci, tak metriky pro nominaci. V případě proměnného počtu cílových extrahovaných informací pro různé webové stránky může být vhodnější použít klasifikaci, která určuje třídu pro každý element zvlášť. Naopak pokud máme pevný počet informací, které chceme extrahovat z každého webového dokumentu, je vhodnější pracovat s nominací, kterou zajímá právě tento pevný počet informací pro každý dokument.

Architektura neuronových sítí pro úkol extrakce z webových dokumentů se může skládat ze dvou částí. První část tzv. *embedder* bere vstup (například ve formě DOM stromu) a produkuje reprezentaci webové stránky. Druhá část *klasifikátor* bere jako vstup zmíněné reprezentace a jeho výstupem je pravděpodobnost náležitosti elementů k dané třídě.

4.4 Existující přístupy

V následující sekci jsou popsány existující řešení využívající strojového učení v oblasti extrakce informací z webových nebo jiných dokumentů. U každé práce jsou na závěr shrnuty techniky a poznatky, které se ukázaly jako přínosné a jsou v určitém směru využity i při návrhu systému této práce.

Extrakce kontextuální informací z vědeckých publikací

Práce Stefana Klampfla a Romana Kerna [26] se zabývá použitím metod strojového učení pro automatickou extrakci kontextuálních informací z vědeckých publikací. V práci se zabývají anotací PDF dokumentů, které jsou podobně jako webové stránky primárně určeny pro prezentaci a chybí jim dostatečné strukturální informace. PDF dokument je nejprve rozdělen na bloky, na které se aplikuje učení s učitelem pro klasifikaci do meta-data kategorií, jako jsou například název, autor nebo abstrakt práce.

Konkrétně jsou použity mechanismy Maximum Entropy (ME) [2] a Beam Search [53]. Vlastnosti bloku použité pro klasifikaci jsou získány z rozložení, formátování, slov (uvnitř i okolo zkoumaného textového bloku) a seznamu běžných jmen. Pro generování vlastností pro slova je použit jazykový model. Rozložení je popsáno pomocí proměnných, které určují kde se textový blok nachází (např. *vlevo nahoře, uprostřed, vpravo dole,* ...). Podobně je popsáno i formátování, kde možnosti jsou např. *je malý font, je tučně, je zarovnáno doprava,* atd. Dále se používají také heuristické vlastnosti jako např. *obsahuje e-mail, obsahuje číslice,* atd. Vlastnosti se specifickými třídami jsou poté získány i pro samotná slova z bloku (např. *je časté slovo, je první v bloku,* ...).

Algoritmy učení s učitelem jsou použity i pro klasifikaci slov z referencí a citací. Zde je použita technologie CRF (Conditional Random Field) [31] pomocí open-source systému ParsCit [9]. Navíc jsou přidány vlastnosti o rozložení a formátování.

Práce ukazuje, že rozdělení složitého dokumentu na bloky a získání pozičních informací (jako například kde se nachází jednotlivé textové bloky) může být užitečné pro správnou extrakci požadovaných bloků.

Kategorizace webových dokumentů pomocí sémantických grafů

Následující práce se zabývá kategorizací webových stránek pomocí sémantických grafů a strojového učení [5]. Více o sémantických grafech je zmíněno v kapitole 4.2. Pro kategorizaci se používají jak metody učení s učitelem, tak metody učení bez učitele. Díky trénování modelu pomocí sémantických vlastností se podařilo dosáhnout uspokojivých výsledků, hlavně pak metodou učení bez učitele. Výsledky jsou porovnávány vzhledem k reprezentaci pomocí mechanismu TF-IDF, který je popsán v kapitole 4.2.

Práce se zabývá klasifikací a aglomerací. Pro klasifikaci je použita technologie SVM (*Support Vector Machine*) [8] a pro aglomeraci technologie SOM (*Self Organizing Maps*) [27]. Jako datová sada je použit korpus Reuters článků. Klasifikace zde probíhala do tří kategorií. Další datová sada byla získána pomocí web-scrapingu novinových článků z pěti kategorií. Výsledky experimentů ukázaly, že použití sémantických vah pro strojové učení na Reuters vykazovalo lepší výsledky pro oba typy strojové učení - s učitelem (SVM) i bez učitele (SOM). Na druhé datové sadě si nejlépe vedla reprezentace pomocí sémantických vah a SOM technologie.

V práci je možné vidět, že získání grafové reprezentace webové stránky může vést k zajímavým výsledkům.

Extrakce obsahu webu

Tato práce [66] se zaměřuje na extrakci méně strukturovaného obsahu stránek, v tomto případě novinových článků. Výsledkem je aplikace, která klasifikuje textové bloky pomocí spojení vizuálních a jazykově nezávislých vlastností. Mezi vizuální informace patří barva textu, jeho velikost a styl, rozložení webu, velikost bloku atd. Více o vizuální reprezentaci webové stránky se nachází v sekci 3.3. Podobné bloky jsou ve výsledné datové sadě sloučeny pomocí shlukování a označeny pomocí LCS algoritmu (více viz 4.2).

Stejně jako v předchozím přístupu je použit klasifikátor SVM. Vstupem klasifikátoru jsou vizuální vlastnosti zmíněné výše. Cílem klasifikátoru je klasifikovat jednotlivé bloky na obsahové a ne-obsahové. Kvůli nevyváženosti počtu obsahových a ne-obsahových bloků se použily třídní váhy pro zvýšení výsledné váhy pozitivních záznamů.

Pro klasifikaci se používaly tři přístupy. V prvním z nich se klasifikátor trénoval odděleně na každém webovém místě. Tento přístup fungoval dobře. Následně se klasifikátor trénoval na datové sadě obsahující webové stránky z více webových míst. Bohužel tento přístup dopadl nejhůře. Nakonec se druhý přístup upravil tak, že při trénování se z datové sady (obsahující webové stránky různých webových míst) vybírala trénovací data náhodně, což napomohlo k lepším výsledkům.

Výsledky práce ukazují, že vytvoření systému, který funguje nezávisle na webovém místě je složitější úloha, než extrakce informací z webových stránek se stejnou nebo velmi podobnou strukturou.

Hluboké neuronové sítě pro extrakci informací z webových stránek

Následující práce [16] využívá konvoluční neuronové sítě pro extrakci webových informací. Pro reprezentaci vizuálního a textového obsahu se používá metoda prostorového kódování, která dovoluje zakódovat tyto informace do jedné neuronové sítě. Reprezentace pomocí kódování vizuálních a textových informací do neuronové sítě dodává značnou flexibilitu, a tvoří tak reprezentaci vhodnou pro úlohy nezávislé na webovém místě.

Data z DOM stromu jsou převedena do 2D roviny a pomocí technik detekce objektů se naleznou elementy obsahující požadované informace. Z DOM stromu se extrahují uzly včetně jejich pozice. Textové uzly se poté použijí jako vstup pro neuronovou síť. Listové uzly se používají jako kandidáti pro klasifikaci.

Konvoluční neuronová síť zpracovává vizuální a textový kontext kandidátních elementů a predikuje jejich pravděpodobnost výskytu v jedné z předem definovaných tříd. Třídy jsou specifické pro danou úlohu extrakce. Pro klasifikaci elementů založenou na jejich absolutní pozici je použit model prostorové pravděpodobnostní distribuce. Výsledná pravděpodobnost pro predikci třídy elementu je tedy dána kontextovou i poziční pravděpodobností, kde obě tyto pravděpodobnosti jsou na sobě nezávislé.

Neuronová síť přijímá tři typy vstupů: screenshoty, textové mapy (viz 3.4) a kandidátní boxy. Screenshoty se zpracovávají pomocí tří konvolučních vrstev, textové mapy pomocí jedné. Výsledky vrstev se spojí a zpracují poslední konvoluční vrstvou, která reprezentuje finální vlastnosti webové stránky. Pomocí ROI poolingu [15] se získá seznam ohraničujících oblastí, které se projektují do poslední vrstvy a extrahují se vlastnosti, které prostorově korespondují s konkrétními částmi stránky. Vektor vlastností obsahuje 86 elementů pro každého kandidáta a používá se pro klasifikaci do výsledných tříd pomocí lineárního klasifikátoru a softmax funkce.

Samotné trénování používá stochastický gradientní sestup se ztrátovou funkcí cross-entropy. Trénuje se dávkově, kdy jedna dávka obsahuje dva obrázky se 100 kandidátními

elementy. Pro potlačení přetrénování se používá augmentace datové sady náhodnou změnou odstínu a invertování barev stránky s 15% pravděpodobností.

Separátní model se používá pro aproximaci prostorové pravděpodobnostní distribuce třídy. Distribuce se modeluje konvolucí matice frekvence třídy s 2D diskrétním gaussovským jádrem. Pravděpodobnost třídy vstupního elementu (daného souřadnicemi jeho ohraničujícího boxu) je vypočítána jako průměr pixelů, které pokrývá.

Výsledný systém je použit na úlohu nalezení elementů na stránce produktu (jméno produktu, obrázek produktu a finální cena). Datová sada je sesbírána ze 39 online prodejců z různých segmentů. Jako baseline byla použita prostorová distribuce, a poté také její kombinace s textovou heuristikou. Nejlepších výsledků bylo dosaženo pomocí kombinace neuronové sítě a prostorového modelu. Tento výsledek naznačuje, že neuronová síť je schopna zachytit cílové elementy i v netriviálních situacích. Součástí experimentů bylo také natrénování neuronových sítí používajících pouze textové mapy nebo pouze screenshoty.

Z výsledků práce plyne, že textová data sama nejsou dostačující pro úkol extrakce informací z webových stránek. Kombinace textových a vizuálních vstupů produkovala nejlepší výsledky.

FreeDOM

FreeDOM je framework pro extrakci strukturovaných informací z webových dokumentů, který je založený na použití neuronových sítích [33]. Framework se skládá ze dvou fází. V první fázi se učí reprezentace každého DOM uzlu pomocí kombinace textové a HTML informace. Druhá fáze používá relační neuronové sítě pro zachycení dalších vazeb a sémantické příbuznosti. Spojení obou fází dovoluje generalizovat extrakci i pro dříve neviděné webové stránky a místa. Framework byl trénován a testován na datové sadě SWDE (více viz 5.5).

První fáze funguje na úrovni uzlů DOM stromu. Pro každý uzel se učí reprezentaci a klasifikuje tyto uzly do jedné z předem definovaných tříd. Reprezentace uzlu je získána kombinací textových informací a HTML elementů z blízkého okolí uzlu. Pro účely zakódování textových informací se používají GloVE word embeddingy [51], konvoluční a LSTM neuronové sítě. Kromě textu samotného uzlu je v reprezentaci zahrnut i typ textu uzlu (např. jestli text obsahuje čísla, datum nebo URL), text předcházející tomuto uzlu a HTML značka uzlu. Pro klasifikaci je použita jednoduchá vícevrstvá neuronová síť.

Druhá fáze pracuje na globální úrovni, snaží se zachytit vztahy mezi vzdálenějšími uzly a tím se přiblížit k zahrnutí informace o vizuální struktuře webové stránky. Použita je relační neuronová síť. Tato fáze řeší problémy první fáze, kdy uzly byly nesprávně klasifikovány kvůli velké lokální podobnosti k uzlům ze špatné třídy nebo kvůli slabým lokálním informacím v uzlu. Z výstupů první fáze se vybere pouze určitý počet uzlů s nejvyšším skóre a vytvoří z nich páry uzlů. Vzdálenost mezi dvojicemi uzlů je modelována pomocí XPath sekvencí a pozičních embeddingů, které představují zakódovanou informaci o pozici bloku ve stránce. Reprezentace XPath vzdálenosti je získána z natrénované sítě LSTM. Cílem druhé fáze je naučit se predikovat třídu (skládající se ze dvou hodnot, každá pro jeden uzel) dvojice uzlů, která odpovídá tomu, zda uzly v dvojici obsahují nebo neobsahují hledanou hodnotu. Výstupem pro každou hledanou hodnotu je pak uzel, kterému byla v párech nejčastěji predikována daná hodnota. K tomuto úkolu je stejně jako v první fázi použita vícevrstvá neuronová síť.

Největší výhodou FreeDOMu je zobecnění modelu pro dobrý výkon na neviděných webových místech i s použitím poměrně malého počtu webových míst pro trénování. Nevýhodou

může potenciálně být zanedbání přímé vizuální informace, která může být užitečná pro extrakci informací. Framework se zaměřuje pouze na text, DOM strom a HTML zdrojový kód a neprovádí renderování webové stránky pro získání reprezentace prezentované uživateli.

Grafové neuronové sítě pro učení nad webovými elementy

Tato práce [22] se zabývá adaptací grafových neuronových sítí (GNN) na DOM strom webových stránek (více viz 4.3). Konvoluční GNN je dále natrénována a použita pro klasifikaci elementů. Pro tento úkol je použitá datová sada Klarna (viz 5.5), ve které jsou webové stránky pro prodej produktů s anotovanou cenou, názvem a obrázkem produktu a tlačítky pro přidání do košíku a přesměrování do košíku. Cílem práce je nalezení těchto anotovaných atributů ve webovém dokumentu. I přesto je zde velké zaměření na obecnost řešení. Cílem je, aby architektura vytvářela reprezentace, které lze použít jak pro extrakci atributů, tak pro jiné úlohy klasifikace na různých šablonách webových stránek.

Práce pracuje se třemi druhy GNN, konkrétně se jedná o rekurentní GNN, konvoluční GNN a attention based GNN. V práci je porovnáváno 10 architektur patřících do zmíněných druhů. Každá GNN pracuje jako embedder, který je následovaný jednovrstvou plně propojenou neuronovou sítí, která slouží jako klasifikátor.

Pro každý uzel se definují vlastnosti stylu, které zahrnují ohraničující box elementů, styl a velikost fontu, počet obrázků v podstromu, viditelnost a HTML tag. Kromě těchto vlastností se ještě experimentuje s vložením textových vlastností pomocí word embeddingů z *Universal Sentence Encoder* [6]. Pro porovnání je použit FreeDOM [33], což je metoda učení se reprezentace navržená přímo pro DOM stromy, která silně závisí na textu a zpracování přirozené řeči (viz předchozí sekce).

V práci je také navržena nová dvoufázová metoda trénování. Metoda dovoluje trénování na malé podmnožině elementů z každé stránky z trénovací množiny. Během trénování se hodnotí elementy na stránce a přidávají se nové, tzv. *confusing* elementy. Confusing element je neanotovaný element, který získal velké skóre pro určitou třídu. V první fázi se tedy vytvoří datová sada ze všech anotovaných elementů, některých náhodných neanotovaných elementů a confusing elementů. Tato datová sada se v druhé fázi využije pro klasické trénování. Výhodou této metody je menší trénovací datová sada a zároveň zlepšení přesnosti klasifikace.

Výsledky ukazují, že nejlépe si vedla vícevrstvá konvoluční grafová neuronová síť. To naznačuje, že velká část kontextu potřebného ke správné klasifikaci je koncentrována v okolí elementu. Zavedením textových vlastností se ještě zvedla úspěšnost tohoto modelu.

Kapitola 5

Návrh systému

V následující kapitole je nejprve formulována úloha řešená navrženým systémem. Návrh celkové architektury systému se skládá ze dvou částí: trénování modelu neuronových sítí a použití modelu neuronových sítí. Pro obě tyto části je třeba vhodně předzpracovat vstupní data. Po návrhu předzpracování těchto dat je popsán i návrh samotného modelu, který se také skládá ze dvou částí: *embedder* pro zachycení vlastností oblastí webových stránek a *klasifikátor*, který na základě vlastností z *embedderu* dané oblasti klasifikuje do tříd vzhledem k formulované úloze. Popis datových sad, které je možné v systému využít, je poslední částí této kapitoly.

5.1 Formulace úlohy

Cílem práce je systém schopný extrahovat klíčové informace z webových stránek produktových webových míst. Každá stránka obsahuje jeden konkrétní produkt. Mezi extrahované informace patří: název produktu, cena produktu, hlavní obrázek produktu a tlačítka pro přidání a přesměrování do košíku. Systém umožňuje extrakci informací i pro webové stránky z dříve neviděných webových míst v různých jazycích.

Vstupem systému je webová stránka. Webovou stránku je možné reprezentovat DOM stromem nebo jiným stromovým modelem (například *AreaTree*). Výstupem je množina cílových uzlů stromu, kde každý uzel může reprezentovat jednu z extrahovaných informací. Cílový uzel je konkrétní oblast webové stránky, která může obsahovat prostý text, obrázek atd. Například pro název produktu je cílový uzel textový fragment a pro obrázek produktu *<img>* oblast obsahující obrázek. Pro každou stránku a extrahovanou informaci by měl být výstupem obecně libovolný počet uzlů, které korespondují s danou informací.

Výsledný systém by měl brát v úvahu nejen zdrojový kód webových stránek, ale i vizuální vlastnosti částí stránek jako je jejich pozice, velikost textu nebo barva. V případě použití textových vlastností by měl být kladen důraz na jazykovou nezávislost. Systém by měl být správně testován a evaluace by měla být provedena pomocí směrodatných metrik. Pro trénování a testování modelu systému by se měla použít aktuální datová sada, kvůli rychlému postupu a změnám ve vývoji webových stránek.

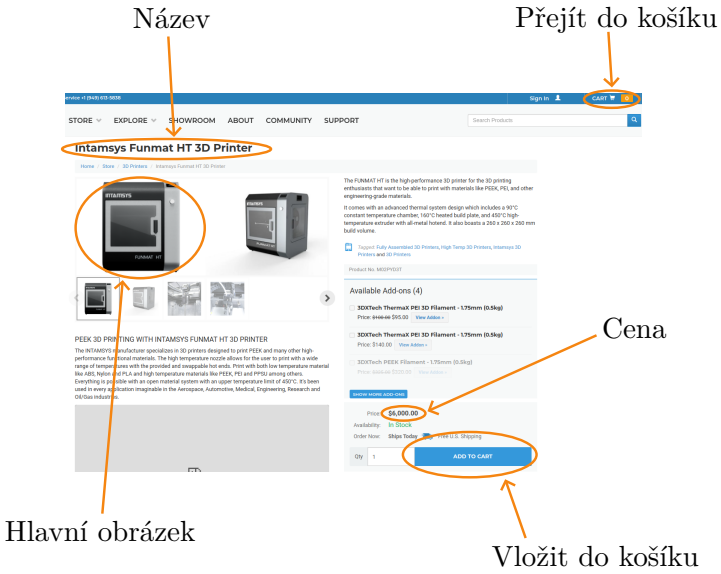
5.2 Návrh celkové architektury

Architektura systému se dělí na dvě části. V první fázi se pomocí datové sady natrénuje model pro klasifikaci částí webové stránky. Datová sada se skládá z anotovaných webových

stránek, jejichž hlavním obsahem je produkt, který je nabízený k prodeji. Každá stránka má anotované následující části:

- název produktu
- cena produktu
- obrázek produktu
- tlačítko pro přidání produktu do košíku
- tlačítko pro přesměrování do košíku

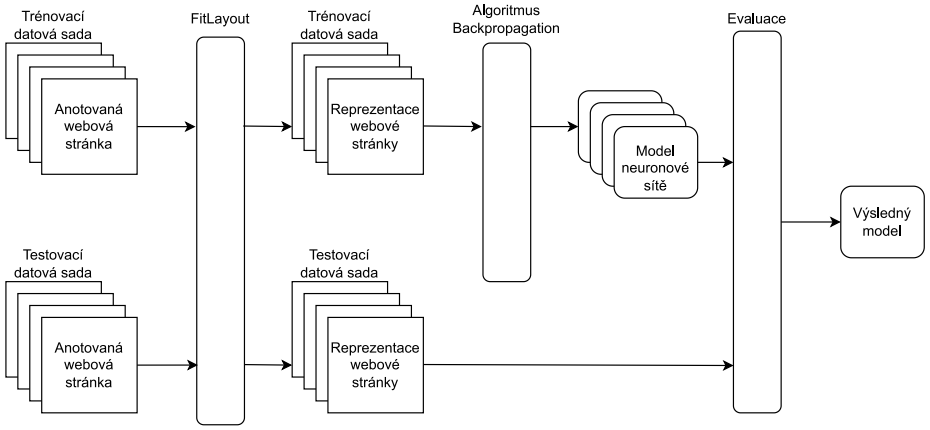
Na obrázku 5.1 je vidět příklad takové stránky s vyznačenými požadovanými informacemi.



Obrázek 5.1: Webová stránka s vyznačenými požadovanými informacemi.

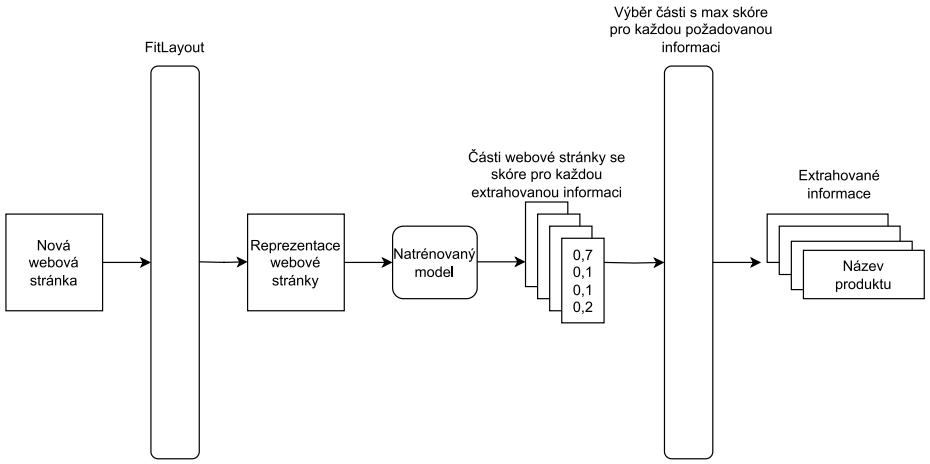
Datová sada je rozdělena na trénovací a testovací část. Webové stránky z datové sady nejprve vstupují do nástroje FitLayout (viz 3.5), pomocí kterého jsou reprezentovány ve formě vhodné pro trénování modelu. Trénovací data se použijí na trénování modelu, zatímco testovací data pro jeho evaluaci. Výsledkem trénování je model, který evaluací dosahoval nejlepších výsledků pro klasifikaci částí webových stránek. Jako model bude použita neuronová síť, popsaná v sekci 5.4. Technologie neuronových sítí je zvolena z důvodů její vhodnosti pro klasifikaci a schopnosti naučit se správně klasifikovat části webových stránek s různou a měnící se strukturou. Pro trénování bude použit algoritmus *backpropagation*, který je založen na principu zpětného šíření chyby a metodě gradientního sestupu (více viz 4.3). Výsledná architektura pro trénování a získání výsledného modelu je zobrazena na obrázku 5.2.

Druhá fáze se zabývá samotnou extrakcí požadovaných informací z nové, dříve neviděné webové stránky. Podobně jako u první fáze je třeba nejprve webovou stránku reprezentovat pro kompatibilitu s modelem. Opět se tedy použije nástroj FitLayout pro získání požadované reprezentace. Výsledek reprezentace se použije na natrénovaný model z první části, který klasifikuje jednotlivé části webové stránky. V závislosti na požadované úloze je poté



Obrázek 5.2: Architektura pro získání výsledného modelu pro klasifikaci částí webové stránky.

možné pomocí skóre vybrat vyhovující části webové stránky, které obsahují dané informace. U některého typu úloh může stačit vybrat jednu část podle maximálního skóre napříč všemi částmi. Jiné úlohy zase mohou obsahovat proměnný počet požadovaných informací a je tedy vhodnější vybírat více částí podle maximálního skóre pro každou danou část zvlášť. Architektura pro extrakci požadovaných informací, kde je použit přístup výběru jedné oblasti pro každou informaci, je zobrazena na obrázku 5.3.

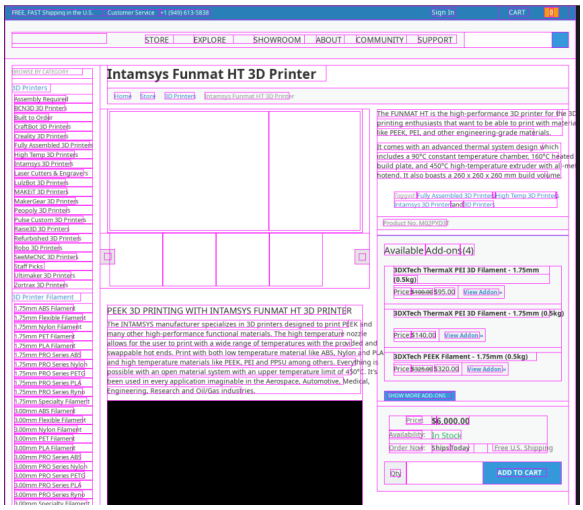


Obrázek 5.3: Architektura pro extrakci požadovaných informací z webové stránky.

5.3 Návrh předzpracování dat

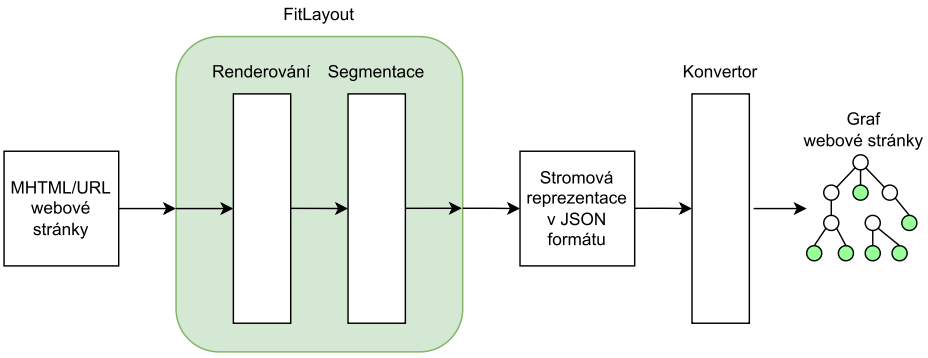
Vstupem pro předzpracování dat jsou webové stránky ve formátu *MHTML*. Další možností vstupu je zadání URL stránky. Webové stránky jsou nejprve reprezentovány pomocí vizuální reprezentace, která je získána pomocí nástroje FitLayout (viz sekce 3.5). Stránky jsou vyrenderovány, a poté je provedena segmentace pro seskupení vizuálních oblastí stránky. Na obrázku 5.4 je vidět výstup segmentace, kde jednotlivé vizuální oblasti jsou organizovány ve stromové struktuře. Kořenem toho stromu je oblast představující celou webovou stránku, která je rodičem pro další menší oblasti stránky. Každá oblast ve stromu může stejně jako

kořen obsahovat další menší oblasti, kterým je rodičem. Listové uzly jsou nejmenší reprezentovanou oblastí a mohou obsahovat požadované extrahované informace. Ne-listové oblasti obsahují kontextové informace o poloze listových uzlů a slouží tedy jako nositelé informace o vztazích mezi danými uzly. Každá oblast má sadu vlastností jako je například úroveň zanoření, její pozice na stránce, barevné vlastnosti nebo vlastnosti textu. Takto reprezentované oblasti jsou uloženy ve formátu JSON se zachováním informace o struktuře stromu.



Obrázek 5.4: Stromová struktura webové stránky získaná segmentačním algoritmem.

Následně jsou tyto stromové reprezentace z formátu JSON převedeny na grafy, kde každá oblast představuje uzel grafu a vazby mezi uzly znázorňují vztah rodiče a potomka ze stromové struktury. Listy stromu jsou reprezentovány speciálními uzly, které jsou cílem klasifikace. Zbylé ne-listové uzly slouží pro propojení listových uzlů, čímž znázorňují jejich vzájemnou polohu. Celé schéma předzpracování je zobrazeno na obrázku 5.5.

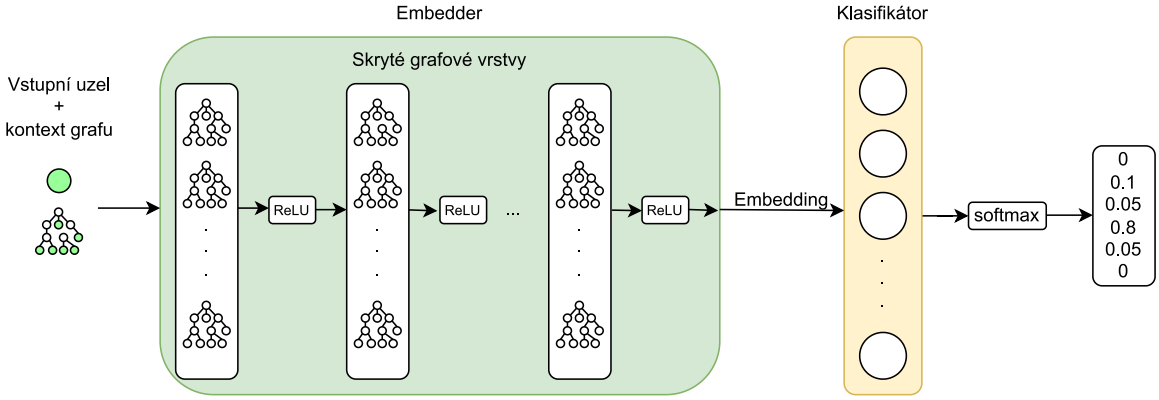


Obrázek 5.5: Návrh předzpracování vstupních webových stránek.

5.4 Návrh modelu strojového učení

Model pro klasifikaci webových stránek je založen na neuronových sítích. Navržený model má dvě části: embedder, který slouží k zachycení vlastností webové stránky a klasifikátor pro určení, zda jednotlivé části obsahují požadovanou informaci. Embedder se skládá

z vícevrstvé grafové neuronové sítě, zatímco klasifikátor představuje klasickou lineární neuronovou vrstvu. Schéma modelu je zobrazeno na obrázku 5.6. Grafové neuronové sítě jsou použity z důvodu, že pracují s informacemi o vztazích mezi jednotlivými oblastmi. Cílem klasifikace jsou pouze listové uzly grafu, které představují nejmenší možné oblasti webové stránky, které mohou obsahovat požadovanou informaci. Zbylé ne-listové oblasti se učí k zachycení vzájemné polohy listových oblastí a slouží jako prostředník pro předávání si informací mezi listovými oblastmi. Právě předáváním informací mezi uzly grafu se vyznačují grafové neuronové sítě.



Obrázek 5.6: Schéma modelu pro klasifikaci webových stránek.

Vstupem do první části modelu (embedderu) je uzel grafu reprezentující jednu oblast webové stránky a kontext celého grafu, tedy kontext jedné webové stránky. Embedder se skládá z grafových neuronových vrstev. Počet těchto vrstev je jedním z parametrů modelu a je možné s ním experimentovat. Stejně tak je parametrem velikost jednotlivých vrstev. V případě, že by nebyly použity grafové neuronové vrstvy, byl by vstupem embedderu pouze jeden uzel a nebyla by použita informace o struktuře grafu, a tedy ani informace o vzájemné poloze jednotlivých oblastí webové stránky. Vstupní uzel je reprezentován vektorem jeho vlastností, jehož velikost závisí na počtu extrahovaných vlastností, který je stejný pro každou oblast webové stránky. Velikost kontextu grafu závisí na velikosti grafu a může být proměnná pro různé vstupní webové stránky. Jednodušší webové stránky jsou obecně reprezentovány menšími grafy s méně uzly a hranami, naopak pro komplexnější webové stránky je třeba větších grafů s více uzly a hranami.

Po provedení operací každé skryté grafové vrstvy následuje aktivační funkce ReLU. Obecně je možné použít jakoukoliv nelineární aktivační funkci. Výstupem embedderu je embedding o zvolené velikosti, který reprezentuje vlastnosti daného uzlu vzhledem ke kontextu grafu a požadované úloze. *Embedding* tedy představuje vektor vlastností jedné oblasti webové stránky, který byl získán z vlastností dané oblasti, ale i vlastností okolních oblastí, pro účely klasifikace oblasti do předem stanovených tříd.

Druhá část modelu, která navazuje na embedder, je klasifikátor. Ten má za úkol na základě poskytnutého embeddingu klasifikovat daný uzel. Výstupem po aplikaci funkce softmax je tedy vektor o velikosti počtu tříd, jehož interpretací je možné zjistit jestli daný uzel obsahuje požadovanou informaci a případně o jakou informaci se jedná. Klasifikátor je implementován jako klasická lineární neuronová vrstva. Podle návrhu ze sekce 5.2 každá webová stránka obsahuje pět anotovaných oblastí. Počet tříd je šest – jedna pro každou požadovanou informaci a jedna třída představující, že uzel žádnou požadovanou informaci

neobsahuje. V kontextu klasifikace oblastí jsou výstupem klasifikátoru následující pravděpodobnosti, jejichž součet je 1:

- pravděpodobnost, že oblast žádnou informaci neobsahuje
- pravděpodobnost, že oblast představuje název produktu
- pravděpodobnost, že oblast představuje cenu produktu
- pravděpodobnost, že oblast představuje obrázek produktu
- pravděpodobnost, že oblast představuje tlačítko pro přidání produktu do košíku (neboli tlačítko *Koupit*)
- pravděpodobnost, že oblast představuje tlačítko pro přesměrování do košíku (neboli tlačítko *Košík*)

Interpretace výstupů klasifikátoru závisí na počtu oblastí, které je třeba pro každou požadovanou informaci získat. V případě pevného počtu oblastí pro každou informaci, je možné ze všech oblastí vybrat tento počet oblastí s maximální pravděpodobností, že představují danou informaci. V opačném případě je možné procházet oblast po oblasti a určovat zda informaci obsahují podle maximální pravděpodobnosti uvnitř této oblasti.

5.5 Datové sady

V následující sekci se zaměříme na existující datové sady, které je možné použít k trénovaní a evaluaci modelů pro extrakci klíčových informací o produktu z webové stránky. Je třeba, aby datové sady byly ve správném formátu a požadované informace k extrakci byly anotovány. Detailněji si popíšeme datovou sadu *Klarna*, která je dále použita pro experimenty v následující práci, ale zmíníme i další existující datové sady.

The Klarna Product Page Dataset

The Klarna Product Page Dataset [22] je datová sada veřejně dostupných webových stránek. Jedná se o webové stránky online prodávaných produktů na rozdílných webových místech. Datová sada obsahuje offline snímek 51 701 produktových stránek od 8 175 různých prodejců z 8 různých zemí (Německo, USA, Velká Británie, Finsko, Rakousko, Švédsko, Norsko a Nizozemsko). Datová sada tedy obsahuje webové stránky v 6 různých jazycích – němčina, angličtina, finština, švédština, norština a nizozemština. Kompletní pohled na počty webových míst a webových stránek podle jednotlivých zemí a jazyků je zobrazen v tabulce 5.1. Webové stránky byly sesbírány mezi rokem 2018 a 2019. Pro každou stránku je anotováno 5 elementů: cena, obrázek, název produktu a tlačítka pro přidání do košíku a přesměrování do košíku. Anotace je přímo v HTML kódu pomocí speciálního atributu *klarna_ai_label*. Anotace proběhla manuálně lidským analytikem.

Datová sada nabízí tři možnosti reprezentace. Prvním z nich je MHTML formát, který je nejméně ztrátový, protože je možné tento formát vyrenderovat přímo v prohlížeči. MHTML kromě HTML obsahuje všechny obrázky a zdroje potřebné k renderování. Nevýhodou je, že všechen JavaScript ze stránek je ztracen. WTL (WebTraversalLibrary) formát vzniká načtením MHTML stránek pomocí chromium-based prohlížeče. WTL formát poskytuje HTML

Země	Jazyk	Počet webových míst	Počet webových stránek
Německo	němčina	2 941	16 765
USA	angličtina	1 794	11 003
Velká Británie	angličtina	1 360	11 144
Finsko	finština	1 125	5 623
Rakousko	němčina	899	1 316
Švédsko	švédština	619	4 866
Norsko	norština	180	852
Nizozemsko	nizozemština	130	123
Celkem		**8 175**	**51 701**

Tabulka 5.1: Přehled *Klarna* datové sady podle jednotlivých zemí a jazyků webových míst a webových stránek.

renderovaného DOM stromu a další metadata (např. ohraničující boxy elementů nebo velikost fontu). Třetí formát je ve formě screenshotů načtených stránek, které korespondují s WTL formátem.

Datová sada je rozdělena na trénovací a testovací část. Ve trénovací části se nachází 80 % dat a v testovací zbylých 20 %. Datová sada je vhodná pro úlohy klasifikace elementů a učení se reprezentace webových stránek.

Další datové sady

Structured Web Data Extraction (SWDE) datová sada [20] obsahuje 124 291 webových stránek z 80 webových míst. Sada obsahuje anotované listové uzly, které obsahují text. Každá webová stránka obsahuje jeden datový záznam s detailními informacemi o entitě. Celkem je anotováno 32 atributů. Datová sada se zaměřuje na 8 témat, ze kterých se odvíjí anotované atributy. Témata jsou: automobil, kniha, kamera, práce, film, NBA hráč, restaurace a univerzita. Například pro automobil jsou anotované atributy: model, cena, motor a spotřeba paliva. Atributy jsou anotovány jako prostý text. Pro každou webovou stránku datová sada obsahuje HTML soubor a URL. Nevýhodou HTML formátu je ztráta vizuální informace získané například pomocí CSS. Další nevýhodou je poměrně velké stáří datové sady a za předpokladu, že struktura a rozložení webových stránek se v čase mění, může mít model trénovaný na SWDE datové sadě problém při testování na neviděných moderních webových stránkách.

Další datové sady by bylo možné získat pomocí web wrapperů (viz 4.2). Například webová stránka Apify[1] slouží jako platforma pro web scraping. Mimo jiné obsahuje již existující nástroje pro získání strukturovaných informací (včetně názvu, ceny a popisu produktu) z některých českých e-shopů. Pomocí Apify je možné získat nejen HTML soubory, ale i další zdroje jako jsou například CSS soubory, JavaScript soubory a obrázky. Díky tomu je možné uložené stránky vykreslit ve virtuálním prohlížeči a získat vizuální informace.

[1] https://apify.com/

Kapitola 6

Implementace

Implementaci je možné z hlediska použitých programovacích jazyků rozdělit na dvě části. V první části probíhá předzpracování vstupní webové stránky do vhodné reprezentace pomocí nástroje FitLayout. Tato část je implementována v objektově orientovaném jazyce Java. Tento programovací jazyk byl zvolen z důvodu, že samotný nástroj FitLayout je napsán ve stejném jazyce, a proto použití stejného jazyka poskytne kompatibilitu a větší možnosti využití nástroje. Druhá část se převážně zabývá samotným trénováním a použitím neuronových sítí. Pro tyto účely byl zvolen vysokoúrovňový jazyk Python, který se hojně používá v odvětví strojového učení a obsahuje tedy i podpůrné knihovny, které je možno v práci využít.

Všechny zdrojové kódy jsou dokumentovány pomocí *docstringů*. *Docstring* je řetězcový literál ve zdrojovém kódu, který se podobně jako komentář používá k dokumentaci určitého úseku kódu. Na rozdíl od klasických komentářů se řetězce *docstrigů* při parsování ze zdrojového stromu neodstraňují, ale zůstávají zachovány po celou dobu běhu programu. Z *docstrigů* je také možné generovat dokumentaci v různých formátech. Pro Python byla vygenerována dokumentace ve formátu *HTML* pomocí nástroje *Sphinx*. Tato dokumentace se nachází v adresáři *learn/docs/*. Podobně pro Java zdrojové kódy se pomocí nástroje *JavaDocs* vygenerovala *HTML* dokumentace do adresáře *represent/docs/*.

Pro veškerou správu kódu byl použit distribuovaný systém správy verzí *git*. a to konkrétně pomocí webového *git* repozitáře *GitLab*. Tento repozitář byl použit i pro sledování chyb a nedokončených částí, čehož bylo využito při vývoji systému.

6.1 Předzpracování vstupních dat

Cílem předzpracování vstupních dat je transformace vstupních webových stránek do reprezentace vhodné k použití metod strojového učení. V procesu transformace se z webové stránky získávají vlastnosti důležité k extrakci informací z webové stránky. Jako vstupní datová sada byla zvolena Klarna datová sada webových stránek nabízejících produkty (viz 5.5). Ta obsahuje webové stránky ve formátu *MHTML*. Tento formát webových stránek je získaný ze souboru *HTML* a kódovaný kódováním *MIME*. Díky tomu zahrnuje navíc zdroje propojené z webové stránky, jako jsou například obrázky, animace, zvukové soubory atd. Druhou podporovanou možností vstupu je zadání *URL* stránky. Zadání *URL* je vhodné pro přímé použití natrénovaného modelu.

Oba tyto formáty je možné vyrenderovat pomocí nástroje puppeteer, který používá Chromium prohlížeč a je dostupný z nástroje FitLayout. Renderování probíhá do rozlišení

$1200 \times$ *height*, kde *height* závisí na výšce vyrenderované stránky. Pro správnou funkčnost renderování je třeba mít lokálně nastavený backend pro pupeteeer a cestu k němu aplikaci předat. Vyrenderovanou stránku je možné uložit jako *Page* ve formátu XML a jako screenshot ve formátu PNG.

Následně se nad vyrenderovanou stránkou provede segmentace. K segmentaci je použita segmentační metoda zpřístupněná nástrojem FitLayout [3]. Metoda vybere všechny viditelné boxy – tedy boxy s viditelným obsahem. Poté se z každého boxu vyrobí vizuální oblast. Vizuální oblasti spolu tvoří stromovou strukturu, která se nazývá *AreaTree*. Výsledky lze uložit ve stejných formátech jako základní vyrenderovanou stránku. Kromě toho je ještě implementován pokročilejší vlastní způsob uložení výsledku segmentace do formátu JSON.

Aplikace podporuje zpracování celé datové sady obsahující více *MHTML* stránek najednou. Při tomto zpracování se vyřazují některé stránky pro lepší kvalitu výsledné datové sady. Některé vstupní stránky není možné vyrenderovat, a proto se ve výsledné datové sadě neobjeví. Dalším možným defektem je chybějící *MHTML* soubor. Specificky pro trénovací a testovací webové stránky se vyřazují i takové stránky, které nemají po vyrenderování anotované důležité informace (název a cena produktu). Výsledkem je datová sada obsahující 42 012 webových stránek. Celkem bylo vyřazeno 9 689 vzorků z nichž 7 868 neobsahovalo anotované důležité informace, 1 788 neobsahovalo *MHTML* soubor a 33 nebylo možné bezchybně vyrenderovat.

Výsledkem segmentace pro každou webovou stránky je tedy tzv. *AreaTree*, který se uložený ve formátu JSON považuje jako výsledek předzpracování vstupních webových stránek, a bude dále použit pro další zpracování k trénování a použití modelu strojového učení k extrakci informací. Tento výsledný JSON obsahuje vizuální oblasti, které mají následující vlastnosti: id, úroveň zanoření, pozice na stránce (ve formátu x_1, x_2, y_1, y_2), barva pozadí, průměrná velikost fontu a průměrné styly fontu (kolik textu je tučně a kurzívou). Každá oblast může obsahovat seznam dalších oblastí, kterým je rodičem. Listová oblast navíc obsahuje box (vizuální část, která může obsahovat text nebo jiný obsah) s dalšími rozšiřujícími vlastnostmi: barva písma, použitý font, průměrné podtržení, množství textu přesahujícího řádek, informaci o tom, zda se jedná o nahrazený box (např. obrázek nebo objekt) a text samotný. Pro trénovací a testovací datovou sadu je u boxů také nalezeno jestli obsahuje hledanou informaci a případně jakou. Typ požadované informace je získán prohledáním atributů boxů (případě jejich přímých rodičů). Pokud je nalezen atribut *klarna-ai-label*, lze pomocí jeho hodnoty zjistit o jakou informaci se jedná.

JSON reprezentace obsahujících oblasti s popsanými vlastnostmi tvoří základní datovou sadu, která poté pokračuje k dalšímu zpracování. Základní datová sada byla později rozšířena o další vlastnosti oblastí, mezi které patří: výška a šířka oblasti, informace, zda je oblast explicitně oddělená od ostatních oblastí a informace, zda je oblast horizontální nebo vertikální oddělovač. Navíc jsou k boxům přidány vlastnosti: typ boxu, typ zobrazení boxu, HTML značka a informace o tom jestli je box viditelný a viditelně oddělený od ostatního obsahu.

Popsaný postup je implementován v jazyce Java a je možné celý proces předzpracování ovládat pomocí vytvořené CLI aplikace. Aplikace dokáže předzpracovat jak MHTML soubory pro účely trénování, tak stránky z poskytnuté URL pro účely využití natrénovaných modelů. Detailní návod pro vytvoření a použití této aplikace je k dispozici v příloze B.

6.2 Převedení základní datové sady na grafy

Jelikož navržený model neuronové sítě (viz 5.4) očekává na vstupu graf a jeden jeho uzel, je třeba nejprve webové stránky ze základní datové sady na grafy převést. Cílem této části je postupně pro každý vzorek dat načíst JSON obsahující reprezentovanou webovou stránku, vytvořit graf ze získaných informací a případně graf uložit. Tato část aplikace je již implementována v jazyce Python.

Stěžejní pro práci s grafy je knihovna *PyTorch Geometric*, která je založena na *PyTorch* knihovně [49]. *PyTorch Geometric* dovoluje modelování a trénování grafových neuronových sítí, jelikož obsahuje množství různých metod pro hluboké učení využívající grafy a jiné netradiční struktury. K tomu poskytuje i podpůrné nástroje pro práci s daty a datovými sadami. Pro reprezentaci jednoho grafu je použit objekt *Data* právě z *PyTorch Geometric*. Tento objekt popisuje homogenní graf, který v sobě drží atributy pro uzly, hrany a celkový graf. Navíc poskytuje další funkcionalitu pro analýzu grafu a práci s tenzory atributů. Tenzor je zobecněný vektor, který může mít více indexů. Pro vytvoření datového objektu reprezentujícího graf jsou potřeba následující atributy:

- **x** – tenzor vlastností uzlu o velikosti *[pocet_uzlu, pocet_vlastnosti_uzlu]*
- **edge_index** – tenzor hran v grafu o velikosti *[2, pocet_hran]*
- **y** – tenzor tříd uzlů (označení jakou požadovanou informaci uzel obsahuje) o velikosti *[pocet_uzlu]*
- **train_mask** – tenzor pro vymaskování uzlů, které jsou určeny k trénování o velikosti *[pocet_uzlu]*

Základní datová sada

Při konverzi se postupně načtou a zpracují všechny vzorky základní datové sady. V jednom vzorku formátu JSON se rekurzivně zpracovávají reprezentované oblasti. Pro danou oblast se do tenzoru uloží její vlastnosti. Číselné vlastnosti je možné uložit rovnou. Vlastnosti představující barvu se převedou do číselné podoby z hexadecimální reprezentace barvy. Logické proměnné (například informace jestli je oblast explicitně oddělená od ostatních) se také převedou do číselné podoby, kde pravdivá hodnota je převedena na 1, nepravdivá naopak na 0. Mezi speciální textové informace patří použitý font a HTML element boxu, u kterých není předem znám obor hodnot. Řešením je vytvoření perzistentních slovníků pro tyto dvě vlastnosti. Pokud konvertor narazí na novou hodnotu fontu nebo elementu, která se ještě nenachází v odpovídajícím slovníku, přidá tuto hodnotu do daného perzistentního slovníku a přiřadí ji novou jedinečnou číselnou reprezentaci. Do tenzoru vlastností se poté přidají tyto číselné reprezentace ze slovníků, které jsou unikátní pro každou možnou hodnotu textových vlastností. U textových vlastností, u kterých je předem znám obor hodnot (například typ boxu), se rovnou hodnota zakóduje indexem z oboru hodnot vlastnosti. Jak bylo zmíněno v předchozí sekci 6.1, některé vlastnosti jsou pouze u oblastí obsahující box. Jelikož je nutné zachovat stejný počet vlastností pro každou oblast, jsou tyto specifické vlastnosti u oblastí neobsahující box nahrazeny hodnotou -1. Výsledný vektor vlastností pro každý uzel grafů ze základní datové sady obsahuje 14 položek.

Rozšířená datová sada

V předchozí sekci 6.2 byla popsána základní datová sada a rozšířená datová sada. Základní datová sada je převedena na grafy zmíněným postupem. V rozšířené datové sadě jsou přidány vlastnosti navíc, ale proběhla i úprava již existujících vlastností. V první řadě byly odstraněny vlastnosti týkající se barev pozadí a textu. Pomocí experimentů se ukázalo, že vlastnosti barev měly negativní vliv na úspěšnost trénování modelů. Dalším rozdílem je, že se v rozšířené datové sadě normalizovaly vlastnosti pozice oblasti, její výšky, šířky a použitého fontu. V neposlední řadě se přidala jednoduchá textová informace pomocí knihovny *sklearn* [50] a funkce *HashingVectorizer*. Cílem této funkce je převedení textu do vektoru číselných reprezentací. Tato vektorizace používá metodu hashování k nalezení číselné reprezentace vstupního textu. Jelikož cílem je získat pouze jednoduchou doplňující informaci k tenzoru vlastností, je zvolena výsledná velikost číselného vektoru textových vlastností o pěti prvcích. K oblastem neobsahující box (takže ani text) se přidá vektor pěti hodnot -1. Takto malá velikost reprezentace textu může způsobit kolize hashe. Rozšířené a komplexní reprezentaci textových vlastností se věnuje až další datová sada. Výsledný vektor vlastností pro každý uzel grafů z rozšířené datové sady obsahuje 23 položek.

Datová sada s komplexními textovými vlastnostmi

V původních převedených grafech se nepočítá s rozšířenou reprezentací textových informací boxových oblastí. Jelikož ale textové vlastnosti nesou důležitou informaci, která může být užitečná k úloze extrakce, byla vytvořena nová datová sada, navazující na rozšířenou datovou sadu, která obsahuje i textové vlastnosti získané pomocí strojového učení. K původnímu tenzoru vlastností uzlu je přidán embedding získaný ze *SentenceTransformeru* [54]. Tento transformer je založen na technologii BERT [12]. Knihovna *sentence_transformers* obsahuje i předtrénované jazykové modely, které jsou schopné pro textový vstup libovolné délky generovat jeho vektorovou číselnou reprezentaci fixní velikosti. Jelikož se v datové sadě nachází webové stránky z více zemí a celkový systém cílí na jazykovou nezávislost, je použit vícejazyčný předtrénovaný model *paraphrase-multilingual-MiniLM-L12-v2*. Oblasti bez boxu text neobsahují, a proto je k jejich tenzoru vlastností přidán *sentence embedding* prázdného řetězce. Výsledný vektor vlastností pro každý uzel grafů z textové datové sady obsahuje 405 položek.

Další náležitosti grafu

Oblasti se zpracovávají rekurzivně a je tedy možné rovnou přidávat informaci o hranách grafu. Každá oblast (kromě kořenové) přidává do tenzoru hran dvojici obsahující svůj identifikátor a identifikátor rodiče. Jelikož se jedná o neorientovaný graf, přidá se i hrana opačná, která obsahuje dvojici identifikátor rodiče a identifikátor zpracovávané hrany. Pokud oblast obsahuje box, přidá se do tenzoru tříd číselná hodnota označující požadovanou informaci, kterou box nese. Případně číselná hodnota představující informaci, že boxová oblast žádnou extrahovanou informaci neobsahuje. U oblastí neobsahující box se do tenzoru tříd přidá -1. Tenzor trénovací masky obsahuje logické hodnoty – pravdivou hodnotu, pokud se jedná o oblast s boxem a negativní hodnotu, pokud jde o oblast bez boxu.

Výsledné grafy

Výsledný graf se vytvoří z tenzorů vlastností, hran, tříd a trénovací masky. Graf je možný perzistentně uložit na disk ve formátu *.pt*. Kromě samotných grafů se ještě uloží seznam všech vymaskovaných tříd z datové sady. Tento seznam je poté možné využít například k určení vah jednotlivých tříd pro váhovanou ztrátovou funkci. I když lze JSON převádět na graf až před samotným použitím, ukázalo se pro účely trénování vhodnější grafy vytvořit dopředu a perzistentně je uložit na disk. První výhodou je zrychlení následného trénování neuronových sítí jelikož není třeba převádět JSONy na grafy před každým kolem trénování. Druhou výhodou je úspora místa. Zatímco základní datová sada v JSON formátu má velikost 8,7 GiB, stejná sada převedená a uložená v grafech je velká pouze 1,8 GiB.

Celá datová sada se reprezentuje pomocí objektu třídy *WebDataset*, která je založena na bázové třídě *InMemoryDataset* z *Pytorch Geometric*. Při inicializaci objektu je možné načíst z disku datovou sadu již v grafové podobě nebo na grafy základní datovou sadu převést až při samotném vytváření objektu. Nejdůležitější metody objektu vrací délku datové sady a jeden vzorek dat na zadaném indexu. Tyto metody je důležité mít naimplementovány, aby bylo možné z *WebDatasetu* vytvořit *DataLoader*. *DataLoader* spojuje datové vzorky z *WebDatasetu* do dávek dat (tzv. batches). Objekty *DataLoaderu* pak slouží jako vstup pro trénování a testování modelů.

6.3 Model neuronové sítě

Model neuronové sítě je reprezentován objektem třídy, která dědí z bázové třídy *Module* z *torch* knihovny. Tento model se podle návrhu z 5.4 skládá ze dvou částí: embedder a klasifikátor. Při vytváření objektu se nejprve inicializují vrstvy obou částí sítě.

První vrstva embedderu vždy přijímá vstup o velikosti počtu vlastností jednoho uzlu grafu. Výstup první vrstvy, který odpovídá velikostem vstupů a výstupů následujících skrytých vrstev je jedním z argumentů modelu. Argumentem modelu je i samotný počet skrytých vrstev. Poslední vrstva embedderu bere vstup velikosti předchozí skryté vrstvy a generuje embedding. Velikost embeddingu je dalším nastavitelným parametrem modelu. Na výběr je ze tří modelů, které se liší typem vrstev embedderu: grafová konvoluční síť (GCN) [25], grafová síť založená na *attention* mechanismu (GAT) [63] nebo síť typu vícevrstvý perceptron (MLP). První dva modely používají grafové konvoluční vrstvy získané z knihovny *torch_geometric*. Třetí typ (MLP) se skládá z obyčejných lineárních vrstev z knihovny *torch*.

Druhá část modelu je klasifikátor. Ten se skládá z jedné vrstvy přijímající vstup o velikosti embeddingu z embedderu a vrací výstup o velikosti počtu klasifikovaných tříd. Klasifikátor může být realizován buď jako lineární vrstva nebo vrstvou odpovídající typu vrstev embedderu, který se liší dle zvoleného modelu.

Model má funkci *forward*, která představuje dopředný krok neuronové sítě a definuje jeho výpočet. Vstupem funkce je tenzor vlastností uzlu a u grafových modelů i kontext grafu. Při dopředném výpočtu se výsledek získá postupným průchodem vstupu všemi definovanými vrstvami modelu. Před každou vrstvou je možné zavolat funkci *dropout*, která deaktivuje zadaný náhodný počet neuronů dané vrstvy. U embedderu se na výsledek vrstev, před vstupem do další vrstvy, aplikuje aktivační funkce. Na výběr je z následujících aktivačních funkcí: *ReLU*, *ELU*, *Leaky ReLU*, hyperbolický tangens, sigmoida a softmax. Na výstup klasifikátoru je aplikovaná funkce softmax.

6.4 Trénování modelů neuronové sítě

Trénování modelů neuronových sítí probíhá přes objekt implementované třídy *Train*. Ta přijímá *DataLoader* trénovací a validační datové sady. Pro jednu epochu se trénuje model na celých trénovacích datech. Validace validační datovou sadou probíhá po zvoleném počtu epoch (více k validaci v následující sekci 6.5). Podle výsledků validace se rozhodne, zda se uloží současný stav modelu. Model se ukládá podle nejlepší dosažené úspěšnosti volitelné metriky. Ve výchozím nastavení se pro porovnávání modelů používá metrika skóre F1.

Trénovací smyčka

Samotný úkon trénování v jedné epoše probíhá na batches z *DataLoaderu*. Pro každou batch (předem zvolený počet vzorků) se nejprve vynulují gradienty v optimalizátoru. Optimalizátory slouží k nalezení optimálního řešení k definované ztrátové funkci. Cílem je nalezení takových parametrů modelu, aby se minimalizovala *loss* ztrátové funkce. Jako optimalizátor je zvolený *AdamW* [39]. *AdamW* je stochastická optimalizační metoda pracující s gradienty a adaptivní mírou učení, která upravuje implementaci poklesu vah oddělením samotné operace snižování vah od aktualizace gradientů. Poté se provede dopředný krok modelu neuronové sítě (viz 6.3). Pokud se jedná o grafovou síť, předává se modelu kromě tenzoru vlastností uzlu i kontext grafu.

Výsledné predikce vrácené modelem a skutečné požadované třídy slouží jako vstup do ztrátové funkce, která vrací *loss* (neboli chybu). *Loss* určuje míru chyby mezi predikcí modelu a očekávanou třídou. Pro výpočet ztrátové funkce se použije trénovací maska, která bere v potaz pouze uzly, které představují oblasti s boxem. Z *loss* se následně derivují gradienty a pomocí optimalizátoru se upraví parametry modelu.

Na konci každé epochy se volá jeden krok plánovače. Jako plánovač je zvolen *Cosine Annealing* z knihovny *torch*. Tento plánovač zajišťuje vysokou míru učení na začátku, která se ale opakovaně rapidně zmenšuje na minimální hodnotu předtím, než se zase rapidně zvýší. Tento postup simuluje restart učícího se procesu s použitím již dobře natrénovaných vah jako počátečního bodu modelu po restartu. Postupu, kde jsou zachovány váhy modelu po restartu, se říká *warm restart* [38].

Ztrátové funkce

V systému je k dispozici několik druhů ztrátových funkcí. Základní ztrátová funkce používaná pro klasifikaci do více tříd je *Cross Entropy Loss* (křížová entropie), která je dostupná z knihovny *torch*. Pokud máme C tříd a označíme pravděpodobnost, že vstup patří do třídy i jako y_i, predikci modelu pro tuto třídu jako p_i, pak se se ztrátová funkce *Cross Entropy Loss* $\mathcal{L}$ spočítá jako 6.1. Pravděpodobnost, že vstup skutečně patří do třídy je v tomto případě binární a vektor y obsahuje 1 na jediném indexu, který odpovídá správné třídě.

$$\mathcal{L}_{CEL} = -\sum_{i=1}^{C} y_i \log\left(p_i\right) \tag{6.1}$$

Tato ztrátová funkce podporuje váhování, které je vhodné u nevyvážených datových sad. Jelikož tato sada je nevyvážená (dominantní počet boxových oblastní neobsahuje hledanou informaci), je váhovaný *Cross Entropy Loss* jednou z možností ztrátových funkcí. Třídy s menším počtem zastoupení v datové sadě mají vyšší váhu než třídy dominantní. Pokud

označíme váhu třídy i jako w_i a ostatní proměnné zůstanou oproti klasické *Cross Entropy Loss* neměnné, pak se váhovaná *Cross Entropy Loss* spočítá jako 6.2.

$$\mathcal{L}_{WCEL} = -\sum_{i=1}^{C} w_i y_i \log\left(p_i\right) \tag{6.2}$$

Ztrátová funkce *Focal Loss* používá modulační člen k *Cross Entropy Loss* funkci, aby se učení zaměřilo na silně špatně klasifikované vzorky. Jedná se tedy o dynamicky škálovanou *Cross Entropy Loss*, kde škálovací faktor klesá až k nule s rostoucí pravděpodobností predikce správné třídy. Díky tomuto faktoru je možné automaticky snížit *loss* snadno klasifikovatelných vzorků a tím donutit model zaměřit se na těžko klasifikovatelné vzorky [35]. Pokud škálovací faktor označíme jako γ, pak se rovnice pro *Focal Loss* spočítá jako 6.3. Podobně jako *Cross Entropy Loss*, tak i *Focal Loss* je možno váhovat.

$$\mathcal{L}_{FL} = -\sum_{i=1}^{C} y_i (1 - p_i)^{\gamma} \cdot \log\left(p_i\right) \tag{6.3}$$

Další možností je použití *Class-balanced* ztrátových funkcí z knihovny *balanced_loss*. *Class-balanced* přístup je založen na tom, že s rostoucím počtem vzorků se další přínos nového přidaného vzorku může snižovat. Obecně se dá říct, že čím více dat, tím lépe. Ale pokud dochází k překrývání informací mezi daty, pak se snižuje i přínos, který může model z těchto dat získat. Cílem je tedy získat pro trénování sadu efektivního počtu vzorků dat, aby tato sada neobsahovala velmi podobné a překrývající se vzorky. Tohoto se dosáhne pomocí úpravy *loss*, kde je výsledná *loss* snížena u nových vzorků, které se překrývají s již použitými vzorky [10]. *Class-balanced* přístup je možný použít jak pro *Cross Entropy Loss*, tak pro *Focal Loss*. Pokud označíme efektivní počet vzorků pro danou třídu i E_i, pak se *Class-balanced loss* pro ztrátovou funkci $\mathcal{L}_x$ vypočítá jako 6.4.

$$\mathcal{L}_{CBx} = \frac{1}{E_i} \cdot \mathcal{L}_x \tag{6.4}$$

6.5 Testování modelů neuronové sítě

Pro testování modelů se používá *DataLoader* testovací nebo validační datové sady. U těchto *DataLoaderů* je nastavena velikost batche 1, grafy se tedy berou postupně jeden po druhém. V jednom kroku testování se vezme jeden vzorek dat a provede se na něm dopředný výpočet zadaného modelu. U testování je možné model specifikovat, při validaci se zpravidla jedná o momentálně trénovaný model. Pro získání přesnosti a skóre F1 se z výstupu modelu pro každou boxovou oblast vybere jako predikovaná třída ta, která má největší pravděpodobnost. Pomocí predikovaných tříd oblastí a skutečných tříd oblastí se vypočítá skóre F1 a přesnost pomocí knihovny *sklearn* [50]. Dále se zjistí průměrný počet predikovaných tříd. Tato metrika může odhalit jestli se model snaží predikovat všechny možné třídy, které se v datové sadě nacházejí.

Pro nominační přesnost se nejprve vyberou pro každou třídu do ní predikované oblasti. Počítá se zde pouze se třídami, které představují konkrétní informaci. Není tedy brána v potaz třída určující, že oblast žádnou informaci nenese. Poté se zkontroluje jestli se v grafu daná třída vyskytuje a pokud ano, zda vybrané oblasti opravdu danou třídu měly obsahovat, čímž se získá přesnost nominace. Přesnost nominace je možné určit zvlášť pro každou třídu s informační hodnotou a nebo jako jednu hodnotu zprůměrováním přesností nominace

všech těchto tříd. Na podobném principu je vypočítána i prediktivní přesnost, která slouží k porovnání s výsledky *Klarna* práce [22]. Zde se také nejprve vyberou pro každou informační třídu predikované oblasti, ale z nich se vybere pouze jedna nejpravděpodobnější u níž se zjistí, zda opravdu obsahovala požadovanou informaci. Jde tedy říci, že mezitím co nominační přesnost pracuje při výpočtu přesnosti se všemi predikovanými oblastmi, prediktivní přesnost pracuje pouze s jednou oblastí pro každou třídu.

Speciálně pro testování jsou získány ještě další metriky z knihovny *sklearn*. Funkce *classification_report* vypisuje preciznost, senzitivitu, F1 skóre a podporu (počet vzorků pro každou třídu). Tyto metriky vypisuje i jako makro nebo váhovaný průměr ze všech tříd. Součástí je také přesnost, která ale už byla získána i dříve. Další funkce *confusion_matrix* vypisuje celou matici záměn pro všechny třídy datové sady.

6.6 Výsledná aplikace pro trénování a testování modelů

Spojením jednotlivých částí implementace vzniká aplikace pro trénování a testování modelů neuronových sítí. Aplikace přijímá následující argumenty:

- **action** – akce, která se má provést (trénování, testování nebo jejich kombinace)
- **train_dataset** – trénovací datová sada
- **test_dataset** – testovací datová sada
- **input_type** – formát datových sad (JSONy nebo grafy)
- **fonts** – perzistentní slovník fontů
- **elements** – perzistentní slovník HTML elementů
- **hyperparams** – sada nakonfigurovaných hyper-parametrů sítě a trénování
- **input_model** – vstupní model pro testování
- **output_model** – výstupní model z trénování

Aplikace argumenty zpracuje a podle zadané akce volá jednotlivé části implementace. Pro akci trénování se trénovací datová sada rozdělí na trénovací a validační v poměru 80:20. Na konci běhu se vypíše celkový čas výpočtu. Detailnější popis instalace a použití aplikace je uveden v příloze C.

K práci je také vytvořena jednoduchá aplikace pro použití modelů na webových stránkách zadaných pomocí jejich URL adresy. Tato aplikace vychází z návrhu architektury pro extrakci požadovaných informací z webové stránky (viz 5.3). Vstupem aplikace je URL webové stránky a cesta k modelu, který se pro extrakci použije. Webová stránka se vykreslí a segmentuje pomocí Java CLI aplikace pro předzpracování vstupních dat (viz 6.1). Následně se použije model pro získání predikcí a pro každou informaci se z oblastí klasifikovaných do odpovídající třídy vybere ta nejpravděpodobnější. Tato oblast se dohledá v původní JSON reprezentaci stránky a vypíše se její pozice a text. Stejně jako pro předchozí aplikaci je návod k instalaci a použití aplikace popsán v příloze C.

Kapitola 7

Experimenty

V práci je experimentováno s různými architekturami sítí a parametry jejich trénování. Experimenty se převážně zaměřují na dvě úlohy a k nim patřící metriky. První a primární úlohou je klasifikace, kde stěžejními metrikami jsou celková přesnost klasifikace uzlů a skóre F1. Tyto metriky ale mohou být zkreslené kvůli velkému množství uzlů, které žádnou informaci neobsahují. Je tedy možné, že klasifikátor predikuje ve velkém množství majoritní třídu, ale i tak dosahuje poměrně velké úspěšnosti klasifikace. Kvůli tomu je experimentováno i s alternativní úlohou nominace a tedy s metrikou nominační přesnosti. Ta říká v kolika procentech případů dokáže model najít cílovou informaci, pokud se na stránce opravdu nachází.

7.1 Hyper-parametry a datové sady

Výsledná aplikace pro testování a trénování modelů (viz 6.6) přijímá jako argument sadu hyper-parametrů ve formátu JSON. Tento JSON obsahuje 21 konfigurovatelných parametrů. Jedním z nich je *seed* pro *torch* knihovnu, který se použije pro inicializaci všech generátorů náhodných čísel používaných touto knihovnou. Použití stejného *seedu* je vhodné pro reprodukovatelnost výsledků *torch* knihovny.

Implementovaný model sítě (viz 6.3) používá osm parametrů. Prvním z nich architektura embedderu, kde je na výběr ze tří možností: grafová konvoluční síť (GCN), grafová síť založená na *attention* mechanismu (GAT) nebo síť typu vícevrstvý perceptron (MLP). Počet skrytých vrstev, jejich velikost a velikost výstupu embedderu jsou dalšími parametry modelu. Před operací každé skryté vrstvy je možnost využít funkci *dropout* s nastavitelnou pravděpodobností. Nastavitelná je i aktivační funkce, která je volána vždy po operaci skryté vrstvy. Jako aktivační funkci je možné použít: *ReLU*, *ELU*, *Leaky ReLU*, hyperbolický tangens, sigmoidu nebo softmax. Poslední dva parametry se týkají druhé části modelu, a to klasifikátoru. V původním návrhu v kapitole 5.4 je znázorněn klasifikátor používající jednu lineární vrstvu. Pomocí parametrů je možné nastavit, aby se tento klasifikátor neskládal z lineární vrstvy, ale z vrstvy stejného typu jako je typ embedderu. Poslední parametr určuje, zda se na výstup klasifikátoru aplikuje funkce softmax.

Pro učení neuronových sítí se používá *AdamW* optimalizátor, který má tři nastavitelné argumenty: míru učení, koeficient pro pokles váhy a parametr *eps*, který se používá pro zlepšení numerické stability. Typ ztrátové funkce pro výpočet chyby je dalším parametrem. Možnosti ztrátových funkcí jsou: *Cross Entropy Loss*, *Weighted Cross Entropy Loss*, *Class-balanced Cross Entropy Loss*, *Focal Loss*, *Weighted Focal Loss* a *Class-balanced Focal*

Loss. Tyto funkce jsou detailněji popsány v sekci 6.4. Ztrátové funkce typu *Focal Loss* mají další konfigurovatelný parametr: škálovací faktor γ. *Class-balanced* funkce zase pro výpočet efektivního počtu vzorků využívají parametr β. Nakonec *Cross Entropy Loss* funkce dovolují nastavit parametr určující míru *label smoothing* při výpočtu chyby. *Label smoothing* je regularizační technika, která snižuje pravděpodobnost správné třídy a naopak přidává pravděpodobnost ostatním (nesprávným) třídám.

Zbylých pět hyper-parametrů se týká samotné trénovací a validační smyčky. Nastavuje se celkový počet epoch na kterých se bude trénovat, stejně tak jako velikost jednoho *batche* dat. Trénovací datovou sadu může být vhodné náhodně zamíchat, a proto je možnost náhodného zamíchání trénovací datové sady dalším parametrem trénování. Jelikož operace validace je časově náročná a nemusí se vyplatit validovat po každé epoše, je možné zvolit po kolika epochách bude validace probíhat. Poslední hyper-parametr určuje na základě jaké metriky se budou porovnávat a ukládat validované modely. Příklad nastavení všech hyper-parametrů ve formátu JSON je zobrazen v příloze D.

Kromě nastavení hyper-parametrů je experimentováno s různými datovými sadami. V sekci 6.2 jsou popsány tři datové sady: základní datová sada, rozšířená datová sada a datová sada s komplexními textovými vlastnostmi. Všechny tyto sady vychází z původní datové sady *The Klarna Product Page Dataset* (viz 5.5), ale liší se použitými vizuálními a textovými vlastnostmi, které se z daných webových stránek získávají.

7.2 Architektura embedderu

Prvním z nastavitelných parametrů modelu, se kterým se experimentovalo, je architektura embedderu. V práci jsou implementovány tři možné architektury. Dvě grafové architektury: grafová konvoluční síť (GCN) a grafová síť založená na *attention* mechanismu (GAT). Třetí možností je síť typu vícevrstvý perceptron (MLP). Experimentováno bylo se všemi třemi druhy architektur. Pro každou architekturu bylo provedeno několik experimentů s různými ztrátovými funkcemi. Nejlepší výsledky pro každou z architektur jsou zobrazeny v tabulce 7.1. Všechny tři metriky nemusely být dosaženy jedním modelem s danou architekturou, ale vždy je vybrána největší hodnota pro každou metriku napříč všemi natrénovanými modely dané architektury. Pro embedder se ukázala jako nejlepší architektura grafová síť založena na attention mechanismu (GAT). Zároveň je vidět, že grafové neuronové sítě si vedly lépe než MLP. To naznačuje, že informace o vztazích mezi jednotlivými oblastmi jsou důležité pro úlohu klasifikace jednotlivých částí webových stránek. V dalších experimentech se proto pracuje s typem embedderu **GAT**.

	Přesnost	F1	Přesnost nominace
MLP	0,9567	0,9453	0,3019
GCN	0,9623	0,9469	0,3280
GAT	**0,9680**	**0,9598**	**0,8581**

Tabulka 7.1: Výsledek experimentů s architekturou embedderu.

7.3 Ztrátová funkce

Při experimentech se ukázalo, že nevyváženost datové sady pro klasifikační úlohu (většina klasifikovaných uzlů neobsahuje informaci) může být problém. Modely mohou být spíše kon-

zervativní v predikování minoritních tříd, čímž klesá nominační přesnost. Ve snaze bojovat s tímto problémem bylo vyzkoušeno více ztrátových funkcí, které jsou detailněji popsány v sekci 6.4. V tabulce 7.2 je vidět srovnání jednotlivých ztrátových funkcí. Ukázalo se, že některé ztrátové funkce dosahují dobrých výsledků pro úlohu klasifikace, jiné zase dominují v úloze nominace. U prvních experimentů si pro metriky přesnost a skóre F1 nejlépe vedl klasický *Cross Entropy Loss*, ale pro přesnost nominace nejlepších výsledků dosáhl váhovaný *Focal Loss*. Jelikož funkce *Class-balanced Focal Loss* v úloze nominace vykazuje na první pohled nejhorších výsledků, je dále experimentováno pouze z ostatními ztrátovými funkcemi.

	Přesnost	F1	Přesnost nominace
Cross Entropy Loss	**0,9680**	**0,9598**	0,4655
Weighted Cross Entropy Loss	0,7893	0,8545	0,6228
Class-balanced Cross Entropy Loss	0,9660	0,9553	0,3832
Focal Loss	0,9496	0,9437	0,4081
Weighted Focal Loss	0,7079	0,8017	**0,8581**
Class-balanced Focal Loss	0,9334	0,9283	0,1368

Tabulka 7.2: Výsledek experimentů se ztrátovými funkcemi.

7.4 Lineární klasifikátor

Jak bylo zmíněno v sekci 7.1, lineární klasifikátor je volitelným nastavením modelu. V případě nepoužití lineárního klasifikátoru je použita další vrstva embedderu (u následujících experimentů se pak jedná o další *GAT* vrstvu), která bere na vstup embedding a vrací výstup o velikosti počtu klasifikovaných tříd. Pro různé nastavení velikosti skryté vrstvy a ztrátové funkce bylo experimentováno jak s použitím, tak nepoužitím lineárního klasifikátoru. Nejlepší výsledky podle zvolené ztrátové funkce jsou zobrazeny v tabulce 7.3. Výsledky ukazují, že se vyplatí používat lineární klasifikátor na embedding embedderu,/ jak pro úlohu klasifikace, tak pro úlohu nominace. Proto je v dalších experimentech vždy lineární klasifikátor nastaven.

7.5 Velikost skryté vrstvy

Současně s použitím lineárního klasifikátoru bylo experimentováno i s různou velikostí vstupů a výstupů skrytých vrstev embedderu. Cílem bylo nalezení ideální velikosti skrytých vrstev vzhledem ke ztrátové funkci. Ideální velikosti se lišily pro různé ztrátové funkce, ale obecně se tato velikost skryté vrstvy pohybovalo okolo 100 - 300 vstupů a výstupů. Nižší velikosti dosahovaly horších výsledků. U vrstev vyšších než 300 se přestaly výsledky zlepšovat a navíc s vyšší velikostí skrytých vrstev roste i časová náročnost trénování a použití daných modelů. Příklad pro ztrátovou funkci *Cross Entropy Loss* s použitím lineárního klasifikátoru je zobrazen v tabulce 7.4.

	Lineární klasifikátor	Přesnost	F1	Přesnost nominace
Cross Entropy Loss	ANO	**0,9755**	**0,9737**	0,6577
	NE	0,9674	0,9668	0,6196
Weighted Cross Entropy Loss	ANO	0,8007	0,8630	**0,9202**
	NE	0,8660	0,9052	0,7676
Class-balanced Cross Entropy Loss	ANO	0,9740	0,9714	0,6683
	NE	0,9725	0,9696	0,6099
Focal Loss	ANO	0,9141	0,9357	0,8695
	NE	0,9183	0,9380	0,8197
Weighted Focal Loss	ANO	0,7566	0,8369	0,8978
	NE	0,8590	0,9025	0,7860

Tabulka 7.3: Výsledek experimentů s použitím lineárního klasifikátoru.

Velikost skrytých vrstev	Přesnost	F1	Přesnost nominace
5	0,9691	0,9623	0,5394
10	0,9710	0,9661	0,5503
50	0,9733	0,9706	0,6202
100	0,9745	0,9722	**0,6704**
200	**0,9755**	**0,9737**	0,6577
300	0,9748	0,9721	0,6173
400	0,9738	0,9707	0,5989

Tabulka 7.4: Výsledek experimentů s velikostí vstupů a výstupů skrytých vrstev.

7.6 Další datové sady

Všechny předchozí experimenty byly prováděny na základní datové sadě. Kromě základní datové sady ale byla vytvořena i sada rozšířená a z ní vycházející sada s komplexními textovými vlastnostmi. Učení modelů na nových datových sadách vychází ze zkušeností nastavení hyper-parametrů základní datové sady. V tabulce 7.5 jsou zobrazeny nejlepší dosažené výsledky pro každou datovou sadu jak pro úlohu klasifikace, tak pro úlohu nominace. Z výsledků je možné vidět, že dává smysl se zaměřovat nejen na samotné hyper-parametry modelu a trénování, ale i na vlastnosti oblastí, které jsou pro toto trénování použity. Rozšířená datová sada, u které byly přidány, odebrány i upraveny vlastnosti oblastí, dosahuje lepších výsledků než datová sada základní. Úplně nejlepších výsledků pro úlohu klasifikace i nominace bylo dosaženo ještě přidáním textových vlastností získaných pomocí technologií strojového učení.

Zároveň se při experimentech na dalších datových sadách ukázalo výhodné využívat funkci *softmax* na výstupy modelu. Ztrátové funkce, použité při trénování modelů, v sobě automaticky obsahují i *softmax* funkci. Tato funkce tedy není explicitně volána, protože již je součástí ztrátových funkcí. Avšak při testování a validaci modelů se ztrátová funkce nepoužívá a není tedy použita ani funkce *softmax*. Proto při experimentech na dalších datových sadách byla funkce přidána na výstupy modelu pro testování a validaci. Na metriky klasifikace, které pracují s každou oblastí zvlášť, tato úprava nemá žádný vliv, avšak u nominace (respektive i prediktivní přesnosti), kde se pracuje s porovnáváním více oblastí mezi sebou, vede normalizace výstupů modelu k lepším výsledkům. Tyto lepší výsledky pramení

z toho, že po použití funkce *softmax* obsahuje výstup pro každou oblast pravděpodobnosti jednotlivých tříd jejichž součet je vždy 1. Díky tomu jsou všechny výstupy pro oblasti v normalizovaném formátu pravděpodobností tříd a je tedy možné oblasti mezi sebou lépe porovnávat.

Datová sada	Úloha	Přesnost	F1	Přesnost nominace
Základní datová sada	Klasifikace	0,9755	0,9737	0,8180
	Nominace	0,8007	0,8630	0,9202
Rozšířená datová sada	Klasifikace	0,9790	0,9786	0,8747
	Nominace	0,8427	0,8886	0,9458
Textová datová sada	Klasifikace	**0,9838**	**0,9837**	0,9400
	Nominace	0,9162	0,9369	**0,9783**

Tabulka 7.5: Výsledek experimentů s datovými sadami.

7.7 Porovnání s jinými pracemi

V práci *Graph Neural Networks for Nomination and Representation Learning of Web Elements* [22] (zkráceně *Klarna*), popsané v sekci 4.4, jsou trénovány grafové neuronové sítě na stejné datové datové sadě jako je ta, která je použita jako startovní bod pro experimenty této práce. Díky tomu je možné přemýšlet o porovnání výsledků nejlepších modelů s touto prací.

Klarna přístup k využití grafových neuronových sítí se ale v mnohém liší od přístupu této práce. V první řadě je grafová reprezentace získána přímo z DOM stromu stránky, na rozdíl od použití vizuální reprezentace vyrenderované stránky. To vede k tomu, že vlivem vyrenderování není možné některé stránky z datové sady v této práci použít, z důvodů uvedených v sekci 6.1. Také se může stát, že jeden DOM element, představující jednu extrahovanou informaci, se vlivem segmentace rozdělí na více boxových oblastí. To znamená, že obecně není možné říct, že se v této práci na každé stránce požadovaná informace nachází pouze jednou, protože se může skládat z více oblastí. Což vede k dalšímu rozdílu, že v práci *Klarna* je velké zaměření na predikci vždy jedné oblasti pro každou informaci, mezitím co tato práce se obecně zaměřuje na klasifikaci s libovolným počtem požadovaných oblastí pro jeden druh informace.

Přesnost nominace použitá v této práci říká v jakém procentu případů je model schopen nalézt informace na webové stránce pokud se na stránce opravdu nachází. V zmíněné práci je použita metrika prediktivní přesnosti, kde je vybrán jeden uzel pro každou třídu a pro každou stránku. Jelikož vlivem předzpracování, které se zaměřuje na vizuální reprezentaci stránky, jsou některé požadované informace ztraceny a jiné se vlivem jemnosti segmentace mohou nacházet na stránce vícekrát, byla v této práci místo prediktivní přesnosti použita právě přesnost nominace pro vyhodnocování modelů. I když trénování modelů nebylo optimalizováno pro prediktivní přesnost, byly nejlepší modely na tuto metriku testovány pro možnost porovnání s výsledky *Klarna* práce. Výsledky porovnání metriky prediktivní přesnosti pro každou požadovanou informaci, navíc i s technologií *FreeDOM* [33], jsou zobrazeny v tabulce 7.6. *FreeDOM* implementace používá text i v netextovém formátu, a proto se její výsledky nemění s použitými textovými vlastnostmi.

Na výsledcích je možné vidět, že i když se následující práce primárně zaměřovala na klasifikaci, tak i pro prediktivní přesnost dosahuje slušných výsledků. Oproti práci *Klarna*

	Textové vlastnosti	Název	Cena	Hlavní obrázek	Tlačítko "Koupit"	Tlačítko "Košík"	Průměr
FreeDOM	NE	0,645	0,245	0,020	0,379	0,061	0,281
	ANO	0,645	0,245	0,020	0,379	0,061	0,281
Klarna GCN-MEAN-T	NE	0,778	0,659	0,594	0,772	0,616	0,709
	ANO	0,811	0,653	0,497	0,911	0,671	0,733
Vytvořený model	NE	0,833	0,754	**0,893**	0,701	0,829	0,802
	ANO	**0,869**	**0,924**	0,888	**0,920**	**0,913**	**0,903**

Tabulka 7.6: Výsledek porovnání vytvořeného modulu s ostatními pracemi metrikou prediktivní přesnosti.

[22] a *FreeDom* [33] je dosažena vetší průměrná prediktivní přesnost i prediktivní přesnost pro všechny druhy extrahované informace.

7.8 Shrnutí experimentů

Kromě zmíněných experimentů bylo dále experimentováno s dalšími hyper-parametry, jako například s velikostí embeddingu pro různé velikosti skrytých vrstev nebo s přidáváním dalších vrstev embedderu. Celkem bylo provedeno přes 250 experimentů. Menší část experimentů s trénováním modelů probíhala lokálně na grafické kartě *NVIDIA GeForce GTX 1050 Ti*. Větší část výpočtů byla možná díky školou poskytnutým serverům s grafickými kartami *NVIDIA RTX A5000*.

Nejlepší model pro klasifikaci i nominaci byl trénován na datové sadě s komplexními textovými vlastnostmi a jeho architektura je založena na embedderu s grafovými neuronovými sítěmi a lineárním klasifikátoru. Pro klasifikaci bylo dosaženo přesnosti **0,9838** a skóre F1 **0,9837** modelem obsahujícím tři skryté vrstvy embedderu o velikosti 20 a trénováním se ztrátovou funkcí *Cross Entropy Loss*. Nejlepší model pro úlohu nominace dosahoval nominační přesnosti **0,9783**. Tento model také obsahoval třívrstvý embedder s velikostí skrytých vrstev 20, ale byl trénován se ztrátovou funkcí *Weighted Cross Entropy Loss*. Detailnější pohled s nominační přesností zvlášť pro každou požadovanou informaci je zobrazen v tabulce 7.7. Z výsledků je vidět, že nejhůře si model vede při nacházení názvu produktu a to i přes to, že byly použity komplexní textové vlastnosti. Naopak nejlepší výsledky dosahuje pro nalezení obrázku a ceny produktu.

	Název produktu	Cena produktu	Hlavní obrázek produktu	Tlačítko "Koupit"	Tlačítko "Košík"	Průměr
Nominační přesnost	0,9569	0,9854	0,9954	0,9742	0,9794	0,9783

Tabulka 7.7: Výsledky nominační přesnosti nejlepšího modelu pro každou požadovanou informaci.

S porovnáním s již existujícími pracemi se ukázalo, že přístup předvedený v této práci se s jejich výsledky může rovnat a dokonce je i předčí. Podrobný výpis metrik pro nejlepší klasifikační model je zobrazen v příloze E. Stejné metriky, ale pro nejlepší nominační model, jsou k nalezení v příloze F.

Další experimenty by se mohly zaměřovat jak na další nastavení a kombinaci hyperparametrů, tak na další úpravy vektoru vlastností extrahovaných pro jednotlivé oblasti. Zajímavým přístupem by mohlo být dotrénování (tzv. *fine-tuning*) již předtrénovaného *SentenceEmbedderu* specificky pro tuto úlohu.

Kapitola 8

Závěr

Práce se zabývá použitím metod strojového učení pro automatickou extrakci informací z webových stránek. Nejprve je v kapitole 2 definován pojem web mining, jež se zabývá právě procesem automatického získávání informací a znalostí z webových dat. Konkrétněji je popsáno jeho odvětví web content mining, tedy extrahování užitečných informací z obsahu webových dokumentů, včetně souvisejících problémů a úskalí spojených s extrakcí informací z různých webových míst.

Následně jsou uvedeny způsoby reprezentace webové stránky. V kapitole 3 se možnosti reprezentace prochází od reprezentace pouze textového obsahu, přes DOM strom až po vizuální reprezentaci. V práci se dále pro získání vhodné vizuální reprezentace používá nástroj FitLayout, a proto je v této části i podrobněji rozveden. Strojové učení představuje kapitola 4. Kromě klasických technologií je detailněji představeno použití neuronových sítí. Kromě samotných metod jsou popsány již existující přístupy a řešení, které se zabývají extrakcí informací s důrazem na práce využívající právě neuronové sítě. U jednotlivých řešení jsou vyzdviženy jejich klady a důležité poznatky, které vedly k efektivní extrakci.

V kapitole 5 je formulována úloha a cíl práce a navržena architektura trénování a extrakce informací. Popsán je jak návrh celkové architektury, tak jednotlivých částí – předzpracování a reprezentace webové stránky a modelu neuronové sítě. Pro trénování navržených modelů je použita datová sada Klarna, která je také popsána v této kapitole.

Kapitola 6 popisuje implementaci systému navrženého v předchozí kapitole. Nad implementovaným systémem jsou prováděny experimenty popsané v kapitole 7. Nejlepší natrénované modely dosahují přesnosti 98,38 % s F1 skóre 0,9837. Pro účely extrakce je dosažena úspěšnost nalezení požadované informace až 97,83 %.

Grafové neuronové sítě se ukazují jako vhodné pro zachycení vlastností jednotlivých částí webových stránek a vztahů mezi nimi. Na výsledný embedding je možné využít lineární klasifikátor a díky tomu extrahovat požadované informace z webových stránek. Výhodou tohoto přístupu je vysoká nezávislost na konkrétní struktuře webové stránky, a tedy i nezávislost na konkrétním webovém místě. Současně je kladen důraz na vysokou jazykovou nezávislost vytvořených modelů.

Možnosti budoucí práce spočívají v pokračujícím experimentování s nastavením a kombinací parametrů modelu. Další možností je přidání jiných zajímavých vlastností k uzlu, včetně textových vlastností získaných dotrénováním vícejazyčného modelu specificky pro tuto úlohu. K větší robustnosti klasifikátoru by také mohlo vést vytvoření a použití nových a aktuálních datových sad webových stránek, například pomocí web scrapingu.

Literatura

[1] Alahmadi, A., Joorabchi, A. a Mahdi, A. E. A new text representation scheme combining Bag-of-Words and Bag-of-Concepts approaches for automatic text classification. In: IEEE. *2013 7th IEEE GCC Conference and Exhibition (GCC)*. IEEE, 2013, s. 108–113. ISBN 9781479907229.

[2] Berger, A. L., Pietra, V. J. D. a Pietra, S. A. D. A Maximum Entropy Approach to Natural Language Processing. *Comput. Linguist.* 1. vyd. Cambridge, MA, USA: MIT Press. mar 1996, sv. 22, č. 1, s. 39–71. ISSN 0891-2017.

[3] Burget, R. Automatic Document Structure Detection for Data Integration. In: Abramowicz, W., ed. *Business Information Systems*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2007, s. 391–397. ISBN 978-3-540-72035-5.

[4] Cai, D., Yu, S., Wen, J.-R. a Ma, W.-Y. *VIPS: a Vision-based Page Segmentation Algorithm*. MSR-TR-2003-79. November 2003. 28 s. Dostupné z: https://www.microsoft.com/en-us/research/publication/vips-a-vision-based-page-segmentation-algorithm/.

[5] Camastra, F., Ciaramella, A., Placitelli, A. a Staiano, A. Machine Learning-Based Web Documents Categorization by Semantic Graphs. In: Bassis, S., Esposito, A. a Morabito, F. C., ed. *Advances in Neural Networks: Computational and Theoretical Issues*. Cham: Springer International Publishing, 2015, s. 75–82. DOI: 10.1007/978-3-319-18164-6_8. ISBN 978-3-319-18164-6. Dostupné z: https://doi.org/10.1007/978-3-319-18164-6_8.

[6] Cer, D., Yang, Y., Kong, S.-y., Hua, N., Limtiaco, N. et al. Universal Sentence Encoder for English. In: Blanco, E. a Lu, W., ed. *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*. Brussels, Belgium: Association for Computational Linguistics, Listopad 2018, s. 169–174. DOI: 10.18653/v1/D18-2029. ISBN 9781510874176. Dostupné z: https://aclanthology.org/D18-2029.

[7] Cooley, R., Mobasher, B. a Srivastava, J. Web mining: Information and pattern discovery on the world wide web. In: IEEE. *Proceedings ninth IEEE international conference on tools with artificial intelligence*. 1997, s. 558–567. ISBN 0818682035.

[8] Cortes, C. a Vapnik, V. Support-Vector Networks. *Mach. Learn.* 1. vyd. USA: Kluwer Academic Publishers. sep 1995, sv. 20, č. 3, s. 273–297. DOI: 10.1023/A:1022627411411. ISSN 0885-6125. Dostupné z: https://doi.org/10.1023/A:1022627411411.

[9] Councill, C. L. G. I. a Kan, M.-Y. ParsCit: an Open-source CRF Reference String Parsing Package. In: Calzolari, N., Choukri, K., Maegaard, B., Mariani, J., Odijk, J. et al., ed. *Proceedings of the Sixth International Language Resources and Evaluation (LREC'08)*. Marrakech, Morocco: European Language Resources Association (ELRA), Květen 2008. ISBN 2951740840.

[10] Cui, Y., Jia, M., Lin, T.-Y., Song, Y. a Belongie, S. Class-Balanced Loss Based on Effective Number of Samples. In: IEEE. *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. 2019, s. 9260–9269. DOI: 10.1109/CVPR.2019.00949. ISBN 9781728132938.

[11] Dahl, G. E., Sainath, T. N. a Hinton, G. E. Improving deep neural networks for LVCSR using rectified linear units and dropout. In: IEEE. *2013 IEEE International Conference on Acoustics, Speech and Signal Processing*. IEEE, 2013, s. 8609–8613. ISBN 1479903566.

[12] Devlin, J., Chang, M.-W., Lee, K. a Toutanova, K. *BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding*. 2018. Cite arxiv:1810.04805Comment: 13 pages. Dostupné z: http://arxiv.org/abs/1810.04805.

[13] Fellbaum, C., ed. *WordNet: An Electronic Lexical Database*. 1. vyd. Cambridge, MA: MIT Press, 1998. Language, Speech, and Communication. ISBN 9780262061971.

[14] Fitch, F. B. Warren S. McCulloch and Walter Pitts. A logical calculus of the ideas immanent in nervous activity. Bulletin of mathematical biophysics, vol. 5 (1943), pp. 115–133. *Journal of Symbolic Logic*. 1. vyd. Cambridge University Press. 1944, sv. 9, č. 2, s. 49–50. DOI: 10.2307/2268029.

[15] Girshick, R. *Fast R-CNN*. arXiv, 2015. DOI: 10.48550/ARXIV.1504.08083. Dostupné z: https://arxiv.org/abs/1504.08083.

[16] Gogar, T., Hubacek, O. a Sedivy, J. Deep Neural Networks for Web Page Information Extraction. In: Maglogiannis, I., Iliadis, L., Papaleonidas, A. a Chochliouros, I., ed. *Artificial Intelligence Applications and Innovations*. Cham: Springer International Publishing, 2016, sv. 475, s. 154–163. IFIP Advances in Information and Communication Technology. ISBN 9783319449432.

[17] Goodfellow, I., Bengio, Y. a Courville, A. *Deep Learning*. 1. vyd. MIT Press, 2016. Adaptive computation and machine learning. ISBN 9780262035613. Dostupné z: https://books.google.co.in/books?id=Np9SDQAAQBAJ.

[18] Hamilton, W. L. Graph Representation Learning. *Synthesis Lectures on Artificial Intelligence and Machine Learning*. 1. vyd. Morgan and Claypool. 2020, sv. 14, č. 3, s. 1–159.

[19] Han, J., Kamber, M. a Pei, J. 13 - Data Mining Trends and Research Frontiers. In: Han, J., Kamber, M. a Pei, J., ed. *Data Mining (Third Edition)*. Third Edition. Boston: Morgan Kaufmann, 2012, s. 585–631. The Morgan Kaufmann Series in Data Management Systems. DOI: https://doi.org/10.1016/B978-0-12-381479-1.00013-7. ISBN 978-0-12-381479-1. Dostupné z: https://www.sciencedirect.com/science/article/pii/B9780123814791000137.

[20] Hao, Q., Cai, R., Pang, Y. a Zhang, L. From One Tree to a Forest: A Unified Solution for Structured Web Data Extraction. In: Ma, W.-Y., Nie, J.-y., Baeza Yates, R., Chua, T.-S. a Croft, B., ed. *Proceedings of the 34th International ACM SIGIR Conference on Research and Development in Information Retrieval*. New York, NY, USA: Association for Computing Machinery, 2011, s. 775–784. SIGIR '11. DOI: 10.1145/2009916.2010020. ISBN 9781450307574. Dostupné z: https://doi.org/10.1145/2009916.2010020.

[21] Hochreiter, S. a Schmidhuber, J. Long Short-term Memory. *Neural computation*. 1. vyd. Prosinec 1997, sv. 9, č. 8, s. 1735–80. DOI: 10.1162/neco.1997.9.8.1735.

[22] Hotti, A., Risuleo, R. S., Magureanu, S., Moradi, A. a Lagergren, J. *The Klarna Product Page Dataset: A Realistic Benchmark for Web Representation Learning*. 2021.

[23] Jing, K. a Xu, J. A Survey on Neural Network Language Models. *CoRR*. 1. vyd. 2019, abs/1906.03591, č. 1. Dostupné z: http://arxiv.org/abs/1906.03591.

[24] Kingma, D. P. a Ba, J. Adam: A method for stochastic optimization. *ArXiv preprint arXiv:1412.6980*. 1. vyd. 2014, č. 1.

[25] Kipf, T. N. a Welling, M. Semi-Supervised Classification with Graph Convolutional Networks. *CoRR*. 1. vyd. 2016, abs/1609.02907, č. 1. Dostupné z: http://arxiv.org/abs/1609.02907.

[26] Klampfl, S. a Kern, R. Machine Learning Techniques for Automatically Extracting Contextual Information from Scientific Publications. In: Gandon, F., Cabrio, E., Stankovic, M. a Zimmermann, A., ed. *Semantic Web Evaluation Challenges*. Cham: Springer International Publishing, 2015, s. 105–116. ISBN 978-3-319-25518-7.

[27] Kohonen, T. The self-organizing map. *Proceedings of the IEEE*. 1. vyd. 1990, sv. 78, č. 9, s. 1464–1480. DOI: 10.1109/5.58325.

[28] Kosala, R. a Blockeel, H. Web Mining Research: A Survey. *ACM SIGKDD Explorations Newsletter*. 1. vyd. Prosinec 2001, sv. 2, č. 1. DOI: 10.1145/360402.360406.

[29] Krogh, A. a Hertz, J. A. A simple weight decay can improve generalization. In: Lippman, D. S., Moody, J. E. a Touretzky, D. S., ed. *Advances in Neural Information Processing Systems 4*. Morgan Kaufmann, 1992, s. 950–957. ISBN 9780262561457.

[30] Kushmerick, N., Weld, D. S. a Doorenbos, R. Wrapper Induction for Information Extraction. In: Pollack, M. E., ed. *Proc. IJCAI-97*. 1997. ISBN 15558604804. Dostupné z: https://citeseerx.ist.psu.edu/document?repid=rep1&type=pdf&doi=50101a449f1687a5bfcfc6206c18197300391e89.

[31] Lafferty, J. D., McCallum, A. a Pereira, F. C. N. Conditional Random Fields: Probabilistic Models for Segmenting and Labeling Sequence Data. In: Danyluk, A., ed. *Proceedings of the Eighteenth International Conference on Machine Learning*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2001, s. 282–289. ICML '01. ISBN 1558607781.

[32] Liddy, E. D. Natural Language Processing. In: McDonald, J. D. a Levine Clark, M., ed. *Encyclopedia of Library and Information Science*. 2. vyd. New York: Marcel Dekker Inc, Květen 2003. ISBN 9780824742591.

[33] Lin, B. Y., Sheng, Y., Vo, N. a Tata, S. FreeDOM: A Transferable Neural Architecture for Structured Information Extraction on Web Documents. *CoRR*. 1. vyd. 2020, abs/2010.10755, č. 1. Dostupné z: https://arxiv.org/abs/2010.10755.

[34] Lin, D. An Information-Theoretic Definition of Similarity. In: Shavlik, J. W., ed. *Proceedings of the Fifteenth International Conference on Machine Learning*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 1998, s. 296–304. ICML '98. ISBN 1558605568.

[35] Lin, T., Goyal, P., Girshick, R. B., He, K. a Dollár, P. Focal Loss for Dense Object Detection. *CoRR*. 1. vyd. 2017, abs/1708.02002, č. 1. Dostupné z: http://arxiv.org/abs/1708.02002.

[36] Liu, S., Li, Y. a Fan, B. Hierarchical RNN for Few-Shot Information Extraction Learning. In: Zhou, Q., Miao, Q., Wang, H., Xie, W., Wang, Y. et al., ed. *Data Science*. Singapore: Springer Singapore, 2018, s. 227–239. ISBN 978-981-13-2206-8.

[37] Lockard, C., Shiralkar, P., Dong, X. L. a Hajishirzi, H. ZeroShotCeres: Zero-Shot Relation Extraction from Semi-Structured Webpages. *CoRR*. 1. vyd. 2020, abs/2005.07105, č. 1. Dostupné z: https://arxiv.org/abs/2005.07105.

[38] Loshchilov, I. a Hutter, F. SGDR: Stochastic Gradient Descent with Restarts. *CoRR*. 1. vyd. 2016, abs/1608.03983, č. 1. Dostupné z: http://arxiv.org/abs/1608.03983.

[39] Loshchilov, I. a Hutter, F. Fixing Weight Decay Regularization in Adam. *CoRR*. 1. vyd. 2017, abs/1711.05101, č. 1. Dostupné z: http://arxiv.org/abs/1711.05101.

[40] Medsker, L. a Jain, L. *Recurrent Neural Networks: Design and Applications*. 1. vyd. CRC Press, 1999. International Series on Computational Intelligence. ISBN 9781420049176.

[41] Mikolov, T., Chen, K., Corrado, G. a Dean, J. Efficient Estimation of Word Representations in Vector Space. *ArXiv preprint arXiv:1301.3781*. 1. vyd. 2013, č. 1.

[42] Mikolov, T., Sutskever, I., Chen, K., Corrado, G. a Dean, J. Distributed Representations of Words and Phrases and their Compositionality. *Advances in neural information processing systems*. 1. vyd. 2013, č. 1.

[43] Milicka, M. a Burget, R. Information Extraction from Web Sources Based on Multi-aspect Content Analysis. In: Gandon, F., Cabrio, E., Stankovic, M. a Zimmermann, A., ed. *Semantic Web Evaluation Challenges*. Cham: Springer International Publishing, 2015, s. 81–92. ISBN 978-3-319-25518-7.

[44] Mitchell, T. M. *Machine learning*. 1. vyd. Boston: McGraw-Hill, 1997. ISBN 0070428077.

[45] Netcraft. *November 2022 Web Server Survey*. 2022. Accessed: 2022-09-29. Dostupné z: https://news.netcraft.com/archives/2022/09/22/september-2022-web-server-survey.html.

[46] Netcraft. *September 2022 Web Server Survey*. 2022. Accessed: 2022-12-22. Dostupné z: https://news.netcraft.com/archives/2022/11/10/november-2022-web-server-survey.html.

[47] Nicol, G., Hors, A. L., Hegaret, P. L., Byrne, S. B., Champion, M. et al. *Document Object Model (DOM) Level 3 Core Specification*. W3C Recommendation. W3C, září 2021. Https://www.w3.org/TR/2021/SPSD-DOM-Level-3-Core-20210928/.

[48] Pala, K., Čapek, T., Zajíčková, B., Bartůšková, D., Kulková, K. et al. *Czech WordNet 1.9 PDT*. 2011. LINDAT/CLARIAH-CZ digital library at the Institute of Formal and Applied Linguistics (ÚFAL), Faculty of Mathematics and Physics, Charles University. Dostupné z: http://hdl.handle.net/11858/00-097C-0000-0001-4880-3.

[49] Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J. et al. PyTorch: An Imperative Style, High-Performance Deep Learning Library. In: Wallach, H., Larochelle, H., Beygelzimer, A., Alché Buc, F. d', Fox, E. et al., ed. *Advances in Neural Information Processing Systems 32*. Curran Associates, Inc., 2019, s. 8024–8035. ISBN 9781713807933. Dostupné z: http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf.

[50] Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B. et al. Scikit-learn: Machine Learning in Python. *Journal of Machine Learning Research*. 1. vyd. 2011, sv. 12, č. 1, s. 2825–2830.

[51] Pennington, J., Socher, R. a Manning, C. D. Glove: Global Vectors for Word Representation. In: Moschitti, A., Pang, B. a Daelemans, W., ed. *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)*. 2014, sv. 14, s. 1532–1543. ISBN 9781634394789. Dostupné z: https://nlp.stanford.edu/pubs/glove.pdf.

[52] Rajalakshmi, Madeshan, Narayanan a Scholar, U. An Exclusive Study on Unstructured Data Mining with Big Data. *International Journal of Applied Engineering Research*. 1. vyd. Leden 2015, sv. 10, č. 1, s. 3875–3886.

[53] Ratnaparkhi, A. a Marcus, M. P. *Maximum Entropy Models for Natural Language Ambiguity Resolution*. USA, 1998. Disertační práce. University of Pennsylvania. ISBN 0591941120. AAI9840230.

[54] Reimers, N. a Gurevych, I. Sentence-BERT: Sentence Embeddings using Siamese BERT-Networks. *CoRR*. 1. vyd. 2019, abs/1908.10084, č. 1. Dostupné z: http://arxiv.org/abs/1908.10084.

[55] Reimers, N. a Gurevych, I. *Making Monolingual Sentence Embeddings Multilingual using Knowledge Distillation*. 2020.

[56] Ruck, D. W., Rogers, S. K. a Kabrisky, M. Feature selection using a multilayer perceptron. *Journal of Neural Network Computing*. 1. vyd. 1990, sv. 2, č. 2, s. 40–48.

[57] Salton, G. a Buckley, C. Term-weighting approaches in automatic text retrieval. *Information Processing & Management*. 1. vyd. 1988, sv. 24, č. 5, s. 513–523. DOI: https://doi.org/10.1016/0306-4573(88)90021-0. ISSN 0306-4573. Dostupné z: https://www.sciencedirect.com/science/article/pii/0306457388900210.

[58] Shiloh-Perl, L. a Giryes, R. Introduction to deep learning. *CoRR*. 1. vyd. 2020, abs/2003.03253, č. 1. Dostupné z: https://arxiv.org/abs/2003.03253.

[59] Smola, A. a Vishwanathan, S. Introduction to machine learning. *Cambridge University, UK*. 1. vyd. 2008, sv. 32, č. 34, s. 2008.

[60] Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I. a Salakhutdinov, R. Dropout: A Simple Way to Prevent Neural Networks from Overfitting. *Journal of Machine Learning Research*. 1. vyd. 2014, sv. 15, č. 56, s. 1929–1958. Dostupné z: http://jmlr.org/papers/v15/srivastava14a.html.

[61] Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L. et al. Attention is All you Need. In: Guyon, I., Luxburg, U. V., Bengio, S., Wallach, H., Fergus, R. et al., ed. *Advances in Neural Information Processing Systems*. Curran Associates, Inc., 2017, sv. 30. ISBN 9781510860964. Dostupné z: https://proceedings.neurips.cc/paper/2017/file/3f5ee243547dee91fbd053c1c4a845aa-Paper.pdf.

[62] Veličković, P. *Everything is Connected: Graph Neural Networks*. 2023.

[63] Veličković, P., Cucurull, G., Casanova, A., Romero, A., Liò, P. et al. *Graph Attention Networks*. 2018.

[64] Xu, B., Wang, N., Chen, T. a Li, M. Empirical Evaluation of Rectified Activations in Convolutional Network. *CoRR*. 1. vyd. 2015, abs/1505.00853, č. 1. Dostupné z: http://arxiv.org/abs/1505.00853.

[65] Zhou, Y., Sheng, Y., Vo, N., Edmonds, N. a Tata, S. Simplified DOM Trees for Transferable Attribute Extraction from the Web. *CoRR*. 1. vyd. 2021, abs/2101.02415, č. 1. Dostupné z: https://arxiv.org/abs/2101.02415.

[66] Zhou, Z. a Mashuq, M. Web content extraction through machine learning. *Standford Univ*. 1. vyd. 2014, č. 1, s. 1–5.

[67] Zhu, J., Yan, Y., Zhao, L., Heimann, M., Akoglu, L. et al. Beyond Homophily in Graph Neural Networks: Current Limitations and Effective Designs. In: Larochelle, H., Ranzato, M., Hadsell, R., Balcan, M. a Lin, H., ed. *Advances in Neural Information Processing Systems*. Curran Associates, Inc., 2020, sv. 33, s. 7793–7804. ISBN 9781713829546. Dostupné z: https://proceedings.neurips.cc/paper/2020/file/58ae23d878a47004366189884c2f8440-Paper.pdf.

Příloha A

Obsah přiloženého paměťového média

- **datasets** – vytvořené a používané datové sady
- **learn** – zdrojové kódy pro strojové učení v jazyce *Python*
- **represent** – zdrojové kódy pro vizuální reprezentaci webových stránek v jazyce *Java*
- **models** – natrénované modely zmíněné v práci
- **src_latex** – zdrojové soubory textu práce
- **text.pdf** – text práce ve formátu PDF
- **Readme.md** – detailní popis obsahu přiloženého paměťového média

Příloha B

Návod k použití CLI Java aplikace pro předzpracování webových dokumentů

CLI Java aplikace pro předzpracování webových dokumentů je spustitelná pomocí *.jar* souboru. Projekt již obsahuje vytvořenou aplikaci *represent/represent.jar*, ale v případě potřeby je možné zdrojové kódy zkompilovat a aplikaci sestavit nástrojem *Maven* pomocí příkazů:

```
cd represent
mvn -DskipTests clean package install
mvn -P assembly package
```

Aplikace pro svůj běh vyžaduje Javu 11 (konkrétně je použita verze *openjdk version 11.0.18*) a nakonfigurovaný Puppeteer backend, který je možný získat pomocí příkazů:

```
git clone https://github.com/FitLayout/fitlayout-puppeteer.git
cd fitlayout-puppeteer
npm install
```

Aplikaci je možné spustit následujícími příkazy:

```
cd represent
java -jar represent.jar --puppeteer-backend <puppeteer-backend>
--input-type <input-type> --input-path <input-path>
--output-format <output-format> --output-path <output-path>
--train <train>
```

, kde

- **--puppeteer-backend** – Cesta k nakonfigurovanému puppeteer backendu.
- **--input-type** – Určuje v jakém formátu budou vstupní data. Možnosti jsou: *url*, *urls*, *mhtmls*.
- **--input-path** – V případě použití formátu *mhtml* se jedná o cestu k adresáři, jehož struktura odpovídá struktuře MHTML části Klarna datové sady. Pro *urls* je očekávána cesta k textovému souboru obsahujícímu URL adresy oddělené novým řádkem. U *url* vstupního formátu argument přijímá jednu URL webové stránky v textové podobě.

- **--output-format** – argument určuje v jakém formátu budou uloženy zpracované webové stránky. Možnosti jsou:
  - **all** – Ukládá všechny možné formáty: XML, PNG pro vyrenderovanou stránku a JSON, PNG pro segmentovanou stránku.
  - **noscreenshot** – Na rozdíl od *all* neukládá PNG snímky obrazovky.
  - **nopage** – Na rozdíl od *all* ukládá pouze JSON a PNG segmentované stránky.
  - **areatree** – Ukládá pouze JSON segmentované stránky, který je považován jako výstup části předzpracování a je dále využit v práci.
- **--output-path** – Cesta k adresáři, do kterého se uloží výstupy.
- **--train** – V případě, že je tento argument nastaven jako 1, se hledají na stránkách atributy *ai-klarna-label* a ukládají se pouze stránky obsahující anotovaný název a cenu produktu.

Příloha C

Návod k použití Python aplikace pro trénování a testování modelů

Pro spuštění aplikace je třeba Python 3 (konkrétně je použita verze *Python 3.10.10*). Poté je třeba nainstalovat potřebné balíčky například pomocí příkazu:

```
pip install -r learn/requirements.txt
```

Samotná aplikace pro trénování a testování modelů se spustí příkazy:

```
cd learn
python main.py --action <action> --input_type <input_type>
--train_dataset <train_dataset> --test_dataset <test_dataset>
--fonts <fonts> --elements <elements> --hyperparams <hyperparams>
--input_model <input_model> --output_model <output_model>
```

, kde

- **--action** – Požadovaná akce. Možnosti jsou: *train* pro trénování, *test* pro testování a *traintest* pro kombinaci, kde nejlepší natrénovaný model je po ukončení učení otestován.
- **--input_type** – Formát vstupních datových sad. První možností je *json* pro použití přímo JSON formátu segmentované stránky z Java CLI aplikace (viz příloha B), kde se tato datová sada převede na grafy až před samotným použitím dat. Druhou možností je *graph*, přijímající datovou sadu, která již byla převedena na grafy.
- **--train_dataset** – Cesta k adresáři, který obsahuje trénovací datovou sadu.
- **--test_dataset** – Cesta k adresáři, který obsahuje testovací datovou sadu.
- **--fonts** – Cesta k perzistentnímu JSON slovníku fontů pro převod *json* reprezentací na grafy.
- **--elements** – Cesta k perzistentnímu JSON slovníku DOM elementů pro převod *json* reprezentací na grafy.
- **--hyperparams** – Cesta k JSON souboru obsahující nakonfigurované parametry modelu a procesu trénování a testování modelů. Hyper-parametry jsou popsány v sekci 7.1 a příklad tohoto souboru je zobrazen v příloze D.

- **--input_model** – Cesta ke vstupnímu modelu pro testování.
- **--output_model** – Cesta ke výstupnímu modelu z trénování.

Pro využití modelů na webových stránkách zadaných jejich URL adresou, lze použít příkazy:

```
cd learn
python extract.py --url <url> --input_model <input_model>
--puppeteer_backend <puppeteer_backend>
```

, kde

- **--url** – URL webové stránky, ze které se budou informace extrahovat.
- **--input_model** – Cesta k natrénovanému modelu, který se pro extrakci použije.
- **--puppeteer_backend** – Cesta k nakonfigurovanému puppeteer backendu.

Příloha D

Příklad nastavení hyper-parametrů

```
{
  "model_name": "gat",
  "layers": 3,
  "hidden_channels": 20,
  "em_size": 20,
  "dropout": 0.0,
  "activation_function": "relu",
  "linear_classifier": true,
  "softmax": false,
  "seed": 42,
  "loss_function": "cel",
  "gamma": 2.0,
  "beta": 0.999,
  "label_smoothing": 0.0,
  "learning_rate": 0.001,
  "weight_decay": 0.0,
  "eps": 1e-08,
  "batch_size": 32,
  "epochs": 3000,
  "validate_after": 5,
  "save_metric": "f1",
  "shuffle": true
}
```

Příloha E

Podrobné metriky pro nejlepší klasifikační model

```
Accuracy = 98.38, F1 Score = 0.8142 / 0.9837, Avg number of classes: 5.12,
Nomination accuracy: 94.00, Klarna nomination accuracy: 83.99

Nomination accuracy per label:
['0.9252', '0.9630', '0.9280', '0.9515', '0.9321']

Klarna nomination accuracy per label:
['0.8060', '0.8688', '0.8368', '0.8652', '0.8227']

              precision recall f1-score support

           0 0.99  0.99  0.99 1801162
           1 0.84  0.81  0.82  18289
           2 0.88  0.85  0.87  17496
           3 0.74  0.70  0.72  12313
           4 0.85  0.86  0.85  13802
           5 0.63  0.63  0.63  11827

accuracy  0.98 1874889
macro avg  0.82  0.81  0.81 1874889
weighted avg  0.98  0.98  0.98 1874889


Confusion matrix:
[[1786912, 2800, 1987, 2990, 2057, 4416]
 [3481,14761, 13, 1, 24, 9]
 [2526, 8,14940, 0, 4, 18]
 [3712, 1, 10, 8587, 2, 1]
 [1922, 7, 3, 0,11855, 15]
 [4298, 0, 14, 2, 13, 7500]]
```

Příloha F

Podrobné metriky pro nejlepší nominační model

```
Accuracy = 91.62, F1 Score = 0.5686 / 0.9369, Avg number of classes: 5.93,
Nomination accuracy: 97.83, Klarna nomination accuracy: 90.28

Nomination accuracy per label:
['0.9569', '0.9854', '0.9954', '0.9742', '0.9794']

Klarna nomination accuracy per label:
['0.8691', '0.9237', '0.8878', '0.9203', '0.9129']

              precision recall f1-score support

           0 1.00  0.91  0.95 1801162
           1 0.36  0.93  0.51  18289
           2 0.42  0.96  0.59  17496
           3 0.17  0.96  0.29  12313
           4 0.44  0.96  0.60  13802
           5 0.31  0.95  0.46  11827

accuracy  0.92 1874889
macro avg  0.45  0.95  0.57 1874889
weighted avg  0.97  0.92  0.94 1874889


Confusion matrix:
[[1647639,30805,22799,58064,16499,25356]
 [1098,17035, 26, 68, 42, 20]
 [ 599, 28,16793, 22, 27, 27]
 [ 440, 34, 15,11811, 9, 4]
 [ 413, 63, 37, 55,13183, 51]
 [ 480, 1, 33, 30, 22,11261]]
```