UNIVERSITY OF LJUBLJANA
FACULTY OF COMPUTER AND INFORMATION SCIENCE

Tomaž Kovačič

Evaluating Web Content Extraction Algorithms

DIPLOMA THESIS
AT THE UNIVERSITY STUDY PROGRAM

Mentor: doc. dr. Zoran Bosnić

Ljubljana, 2012

UNIVERZA V LJUBLJANI
FAKULTETA ZA RAČUNALNIŠTVO IN INFORMATIKO

Tomaž Kovačič

Vrednotenje algoritmov za izdelavo izvlečkov spletnih vsebin

DIPLOMSKO DELO
NA UNIVERZITETNEM ŠTUDIJU

Mentor: doc. dr. Zoran Bosnić

Ljubljana, 2012

Univerza v Ljubljani
Fakulteta za računalništvo in informatiko

Tržaška 25
1001 Ljubljana, Slovenija
telefon: 01 476 84 11
01 476 83 87
faks: 01 426 46 47
01 476 87 11
www.fri.uni-lj.si
e-mail: dekanat@fri.uni-lj.si

No. of dissertation: 01824/2012
Date: 03.04.2012

University of Ljubljana, Faculty of Computer and Information Science issues the following dissertation:

Candidate: **TOMAŽ KOVAČIĆ**

Title: **EVALUATING WEB CONTENT EXTRACTION ALGORITHMS**

Type of dissertation: Undergraduate dissertation

Topic of the thesis:

In the last years, the world wide web has become an important information medium. However, its contents are mostly unstructured and intertwined with other non-related streams of content (e.g. ads). In practice, this calls for automatically solving tasks for extraction of a specifically targeted web page parts.

The thesis shall provide an overview and analysis of different existing algorithms for web content extraction. The candidate should survey the related work and experimentally evaluate a chosen set of algorithms.

Mentor:

Assist. Prof. Zoran Bosnić

Dean:

Prof. Nikolaj Zimic, PhD

Fakulteta za računalništvo in informatiko
2
UNIVERZA V LJUBLJANI

Univerza v Ljubljani
Fakulteta za računalništvo in informatiko

Tržaška 25
1001 Ljubljana, Slovenija
telefon: 01 476 84 11
01 476 83 87
faks: 01 426 46 47
01 476 87 11
www.fri.uni-lj.si
e-mail: dekanat@fri.uni-lj.si

Št. naloge: 01824/2012
Datum: 03.04.2012

Univerza v Ljubljani, Fakulteta za računalništvo in informatiko izdaja naslednjo nalogo:

Kandidat: **TOMAŽ KOVAČIĆ**

Naslov: **VREDNOTENJE ALGORITMOV ZA IZDELAVO IZVLEČKOV SPLETNIH VSEBIN**

EVALUATING WEB CONTENT EXTRACTION ALGORITHMS

Vrsta naloge: Diplomsko delo univerzitetnega študija

Tematika naloge:

V zadnjih letih je splet postal pomemben informacijski medij. Istočasno pa splet odlikuje tudi nestrukturiranost vsebin in to, da je hkrati tudi oglasni medij, v katerem se prispevki prepletajo z glavno vsebino strani. V praksi se zato velikokrat srečamo z nalogo narediti izvlečke (ekstrahirati) le določeno, ciljano, vsebino spletnih strani.
V diplomskem delu naj kandidat preuči različne obstoječe algoritme, namenjene opravljanju opisane naloge. V delu naj naredi pregled področja, prikazane algoritme pa naj tudi eksperimentalno evalvira in ovrednoti.

Mentor:

doc. dr. Zoran Bosnić

Dekan:

prof. dr. Nikolaj Zimic

Fakulteta za računalništvo in informatiko • UNIVERZA V LJUBLJANI • 2

IZJAVA O AVTORSTVU

diplomskega dela

Spodaj podpisani Tomaž Kovačič,

z vpisno številko 63050153,

sem avtor diplomskega dela z naslovom:

Vrednotenje algoritmov za izdelavo izvlečkov spletnih vsebin

S svojim podpisom zagotavljam, da:

- sem diplomsko delo izdelal samostojno pod mentorstvom
  doc. dr. Zorana Bosnića
- so elektronska oblika diplomskega dela, naslov (slov., angl.), povzetek (slov., angl.) ter kljune besede (slov., angl.) identični s tiskano obliko diplomskega dela
- soglašam z javno objavo elektronske oblike diplomskega dela v zbirki "Dela FRI".

V Ljubljani, dne 30.05.2012 Podpis avtorja:

Zahvala

Najprej gre zahvala vsem, ki ste me tekom nastajanja diplomskega dela spodbujali in mi nudili moralno in strokovno podporo. Zahvaljujem svojemu mentorju, doc. dr. Zoranu Bosniću, za potrpežljivost in res dobrodošlo pomoč pri usklajevnju detajlov na začetku prvotnega raziskovalnega dela ter pri opravljanju formalnosti ob koncu postopka zagovora ter oddaje diplomske naloge.

Iskrena hvala gre tudi kolegom iz podjetja Zemanta, ki so mi nudili nepogrešljivo strokovno pomoč in težko prigarane izkušnje iz zelo specifične domene s katero se ukvarja tudi to delo. Mag. Dušan Omerčevič, Andraž Tori ter Tomaž Šolc so mi že v začetku dali zagon ter motivacijo za raziskavo tega zelo razpršenega a hkrati za industrijo zelo pomembnega področja. Kasneje se je s svojimi akademskimi izkušnjami moji ekipi neuradnih mentorjev ter sodelavcev pridružila še dr. Mateja Verlič. Omenjenim se ponovno zahvaljujem za vso izkazano pomoč ter podporo tekom nastajanja tega dela.

Hvala tudi vsem avtorjem vidnejših člankov, ki sem jih tekom samih raziskav imel priložnost osebno spoznati in so mi nudili marsikateri koristen nasvet oziroma so mi omogočili dostop do njihove programske opreme: dr. Christian Kohlschütter, dr. Jeffrey Pasternack in dr. Thomas Gottron. Zaslužni so tudi avtorji komercialne programske opreme: Michael Tung in mag. John Lehman.

Mami in Očetu

Contents

List of Used Abbreviations

API Application Programming Interface

CSS Cascading Style Sheets

DOM Document Object Model

CE Content Extraction

GUI Graphical User Interface

HTTP HyperText Transfer Protocol

HTML HyperText Markup Language

NLP Natural Language Processing

TE Text Extractor, Text Extraction

URL Uniform Resource Locator

Povzetek

V diplomskem delu smo opravili nalogo kvantitativnega in kvalitativnega vrednotenja algoritmov za izdelavo izvlečkov spletnih vsebin. Poleg vrednotenja smo opravili tudi širok pregled omenjenih algoritmov in povezanih znanstvenih del ter primerov aplikacije algoritmov na različna področja v industrijski in znanstveni sferi. Za potrebe vrednotenja smo izdelali ogrodje za integracijo knjižnic in spletnih storitev, ki implementirajo različne algoritme s sposobnostjo izdelave izvlečkov s poljubnih spletnih strani. Ogrodje nam je omogočilo pridobitev potrebnih izvlečkov za vse integrirane algoritme na dveh različnih naborih dokumentov. Za omenjene nabore dokumentov smo pridobili človeško označene izvlečke, kar je skupaj s primerjavo rezultata posameznega algoritma služilo kot glavno sredstvo za končno kvantitativno vrednotenje.

Omenjeni algoritmi za izdelavo izvlečkov spletnih vsebin so sposobni na vhodu sprejeti poljubno spletno mesto in na izhodu vrniti le prečiščeno tekstovno vsebino. Pri tem algoritem odstrani vse pogosto uporabljene gradnike sodobnih spletnih strani, kot so: navigacijski bloki, reklamne pasice, glavo in nogo, vrinjeno vsebino in ostale gradnike, ki ne prispevajo h glavni vsebini, ampak so dodani zaradi ohranjanje kontinuitete, koherence in semantične povezanosti spletnih mest znotraj spletne strani.

Glavni cilj in motivacija za to delo sta združitev ugotovitev in raziskav razpršenih znanstvenih del po različnih revijah, konferencah in znanstvenih delavnicah po eni strani ter prizadevanja in potrebe industrije po drugi strani, v zadnjem desetletju na področju izdelave izvlečkov na nivoju spletnih mest ter spletnih strani. Eden izmed glavnih ciljev je tudi postaviti obstoječe metode iz te domene v zmogljivostni kontekst.

Prvi cilj smo dosegli s pregledom več kot 30 znanstvenih del, ki se nanašajo na širšo domeno ekstrakcije spletnih vsebin ter pregledom komercialnih spletnih storitev in odprto-kodnih knjižnic. Slednji cilj smo uresničili z evalvacijo 12 različnih orodij in njihovih implementacijskih variacij, kar skupaj znaša 17 različnih algoritmov, ki smo jih vrednotili znotraj dveh človeško označenih naborov dokumentov.

Tekom pregleda področja smo ugotovili, da imamo opravka s štirimi abstraktnimi, a zelo povezanimi kategorijami algoritmov za ekstrakcijo spletnih vsebin. Vse algoritme smo tako klasificirali v eno izmed naslednjih kategorij:

Algoritmi, ki temeljijo na ovojnicah za izluščevanje vsebine. Ti algoritmi preko internega mehanizma sklepanja ustvarijo t.i. ovojnico oziroma proceduro, ki narekuje pravila, kako iz danega spletnega mesta izvleči ciljano vsebino. Ponavadi mehanizem sklepanja deluje tako, da opazuje podobnosti med spletnimi mesti znotraj iste spletne strani.

Segmentacijski algoritmi. Algoritmi znotraj tega razreda so sposobni spletno mesto razdeliti na semantično, vsebinsko oziroma vizualno podobne segmente.

Algoritmi za detekcijo spletnih predlog. Večina današnjih spletnih mest znotraj iste spletne strani, je zgrajenih s pomočjo spletnih predlog. Te predloge ovijejo glavno vsebino z gradniki, kot so: navigacija, glava in noga, reklamne pasice in ostali običajni spletni segmenti. Algoritmi v tej kategoriji na podlagi videne množice spletnih mest poskušajo poiskati dele spletnih mest, ki so bili ustvarjeni s prej omenjeno predlogo.

Algoritmi za izdelavo izvlečkov glavne tekstovne vsebine. Čeprav je ta kategorija algoritmov močno povezana s prejšnjo, so algoritmi v tej kategoriji sposobni glede na dano poljubno spletno mesto (brez ohranjanja internega stanja o spletnih straneh) določiti, kaj predstavlja glavno tekstovno vsebino (npr: glavno besedilo članka znotraj novičarskega spletnega mesta) in pri tem odstraniti vse šumne gradnike spletnega mesta.

Čeprav smo v našem delu naredili pregled vseh omenjenih kategorij, smo se v poglavju, ki je posvečeno vrednotenju, omejili samo na zadnjo kategorijo algoritmov.

Poleg pregleda samih algoritmov smo v naše delo vključili tudi pregled glavnih sredstev, ki jih potrebujemo za kvantitativno vrednotenje. Pod sredstva vključujemo podatke oziroma nabore spletnih mest, katerih izvlečke je predhodno ustvaril človeški ekspert in metrike, ki pokažejo, kako se izvlečki posameznih algoritmov razlikujejo od tistih, ki jih je izdelal ekspert.

Izbrali smo dve klasični metriki s področja strojnega učenja, to sta preciznost in priklic. V našem delu smo jih naprej postavili v klasičnen kontekst strojnega učenja in bolj znani kontekst vrednotenja spletnih iskalnikov. Nato pa smo jih postavili v kontekst vrednotenja algoritmov za izdelavo izvlečkov.

Poleg preciznosti in priklica smo v omenjeni kontekst postavili tudi F-oceno oziroma poenostavljeno različico F_1-oceno. Pri končni definiciji metrik smo upoštevali konvergenco definicij med sorodnimi znanstvenimi deli, ki tudi vključujejo vrednotenje omenjenih algoritmov. Prav tako smo vključili tudi diskusijo o uporabni različnih nivojev razdrobljenosti modela dokumenta pri izračunu preciznosti in priklica za posamezen dokument v naboru.

Za izračun omenjenih metrik smo potrebovali tudi podatke. Izbrali smo dva specifična nabora dokumentov, ki poleg ekspertno izdelanih izvlečkov vsebuje tudi neobdelane HTML dokumente, ki predstavljajo različna spletna mesta. Oba dva nabora dokumentov predstavljata svojo domeno. Prvi nabor dokumentov vsebuje različna spletna mesta iz različnih koncpetualnih domen in je bil sestavljen v okviru zmogljivostnega znanstvenega tekmovanja imenovanega Cleaneval. Drugi pa predstavlja domeno spletnih mest, ki vsebujejo novičarske članke, ki so se v daljšem časovnem obdobju pojavili na agregatu svetovnih novic Google News.

Nadaljevali smo z opisom implementacije našega evaluacijskega ogrodja, ki smo ga zgradili za potrebe diplomske naloge.

Ogrodje smo zgradili s ciljem za čim lažjo integracijo in evalvacijo različnih algoritmov, ki so implementirani v različnih programskih jezikih ali pa so na voljo le v obliki spletne storitve.

Omenjenemu cilju smo zadostili tako, da smo okoli knjižnic sprogramirali ovojnice, ki glavno funkcionalnost algoritma izpostavi kot spletno storitev na podlagi protokola HTTP. Takšne storitve smo potem zlahka integrirali v evalvacijsko ogrodje.

Te algoritme želimo vrednotiti na dveh naborih dokumentov, ki imajo različne formate za ekspertno izdelane izvlečke ter imajo pripete različne metapodatke. Zajezitev takšnih variacij je bil naslednji cilj arhitekture ogrodja, saj mora biti ogrodje sposobno abstrahirati in poenotiti takšen ne-determinizem med različnimi nabori dokumentov.

V zaključnem poglavju diplomskega dela smo predstavili kvantitativne rezultate vrednotenja, kjer smo podali tudi diskusijo in interpretacijo o samih rezultatih. Poleg tabelaričnega prikaza izračunov metrik smo rezultate predstavili tudi v vizualni obliki s pomočjo grafov. Vizualizacija rezultatov vključuje tudi ekvidistančno distribucijo meritev preciznosti, priklica ter ocene F_1 za posamezne dokumente v naboru. Obravnavali smo tudi robne primere metrik in napake implementacijske narave posameznega algoritma.

Ključne besede:

algoritem, izvlečki, vrednotenje, spletno mesto

Abstract

Today's web has proved to be a vast and valuable resource of information. A large portion of written textual content is published in the form of HTML documents on news sites, blogs, forums and other web sites. Automated extraction of such content from an arbitrary HTML structure has proved to be a non-trivial problem, since the underlying technology and the authors themselves who deliver the content to the end user, tend to wrap the main content with boilerplate elements as navigation, header, footer, advertisements, site-maps and other noisy content.

In this diploma thesis we focus on the review and evaluation of algorithms capable of extracting main textual content from an arbitrary web page. Throughout this work we refer to this category of algorithms as text extractors (TE). The main goal is to provide an overview across the sparse literature of content extraction and to evaluate the performance of text extraction methods based on golden standard data.

First we provide a detailed survey of methods that we categorize into four categories: wrapper based, template detection, web page segmentation and text extraction (TE) approaches to content extraction. For the evaluation part of our task we focus only on the last category of algorithms. We continue by exploring the means for creating an evaluation environment, which consists of metrics and data. We use two datasets, one representing a cross-domain collection of documents and the other containing mostly news type web pages and then evaluate various TE using precision, recall and F_1-score.

We conclude our work by presenting the final evaluation results of 17 algorithms on two datasets. Our results analysis includes raw numbers, discussion and various visualizations that help us to interpret the results.

Key words:

algorithm, extraction, evaluation, web page

Chapter 1

Introduction

1.1 Problem Definition

Due to the rapid growth of the internet we get a large part of our everyday information from various sources such as news portals, article archives, blogs, encyclopaedias, forums, online magazines and others. Such vast amounts of textual information fragmented across web pages accessible to everyone have attracted many parties from both academia and industry who wish to harvest its value and implicit knowledge.

Traditionally the main content area (depicted in *Figure 1.1*) of a single web page that contains some form of textual information is wrapped with other common building blocks as menus, headers, footers, advertisements, site maps, disclaimers and other types of content that fall in the category of boilerplate content. Usually this is due to the the technology that delivers a web page to the user.

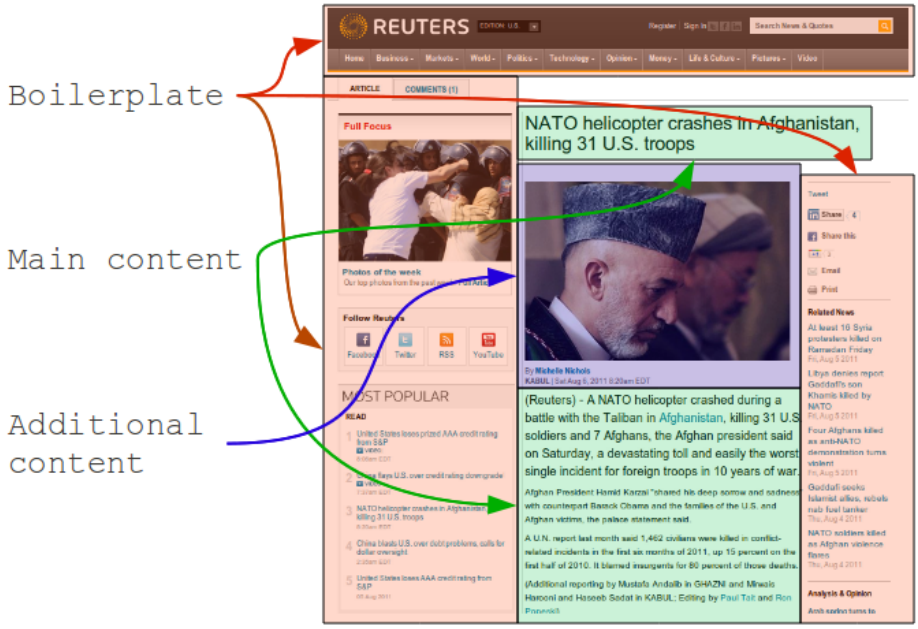


Figure 1.1: Typical structure of a news web page.
Source: `http://www.reuters.com/article/2011/08/06/us-afghanistan-violence-idUSTRE7750UW20110806`

Before presenting the motivation for developing content extraction methods, let us first explore the technology that delivers large parts of modern-day web content. In the following generalized example we follow a user request through the server pipeline all the way to the corresponding response in the form of formatted HTML content.

1. Typically, the user requests a resource using a client (web browser).

2. The request is propagated to the web server via HTTP.

3. The server than executes a procedure that handles the given request. Such a procedure might include logic for querying a database, reading from a file or some other form of data retrieval.

4. The retrieved data must be then formatted using the appropriate HTML syntax. This part is typically done using HTML templates. The template mechanism ensures that all web pages from the same resource share a common layout.

5. The formatted data is then returned back to the user in the form of an HTML document.

The intriguing parts of this process, in the domain of our problem, are those where structured data get wrapped into HTML format (stages 3 and 4). In this particular stage, developers insert all the navigation, design and advertisement-related segments of the web page. Wrapping the data in such a fashion introduces a wide array of cross-domain problems reaching out to the sub-fields of information retrieval, machine learning and data mining.

Let us now define notable problem definitions related to this work:

Problem 1. *Given a set of HTML documents believed to share a common layout, detect the underlying structure of the template .*

Problem 2. *Given an HTML document (or many documents with similar layout), infer which parts can be segmented together based on their visual and semantic cohesiveness.*

Problem 3. *Given a single HTML document, extract the meaningful textual content by removing or detecting all the boilerplate elements.*

Note that these problem definitions are closely related and solving them might share the same principles. The main focus of this work is the evaluation of approaches that solve the 3. problem statement.

Now let us explore the motivation for developing algorithms that solve previously listed problems.

Engineers that build search engines and researchers that compile large text corpora from HTML documents retrieved from the web are only interested in the main content of the web page and treat boilerplate content as noise that would either reduce the information-retrieval capacity of a search engine or corrupt the analysis of the corpora. Apart from web search and text corpora compilation there are also many other use cases where cleaning out noisy elements from HTML documents is considered necessary. Most notable applications include: improving near-duplicate detection methods, adaptive viewing on mobile devices, web browser utilities for improved reading experience and tools for generating printer-friendly formats.

We now perceive the motivation behind devising algorithms depicted in *Figure 1.2* that are capable of extracting meaningful content from arbitrary HTML documents by using a wide range of prediction models, heuristics and prior knowledge.

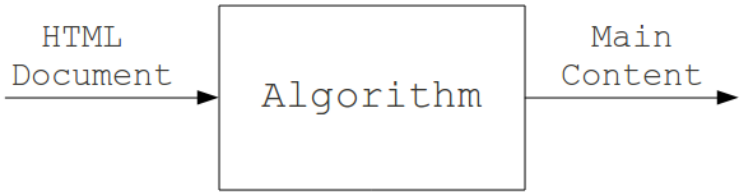


Figure 1.2: The generalized text extraction algorithm receives an arbitrary HTML document and produces the main textual content without the boilerplate elements in the output.

With an increasing demand for methods with such capabilities researchers and companies have created various libraries, web APIs and other types of similar software. To address the problem of performance oversight of these methods, we employ commonly used metrics from the field of information retrieval, computed over evaluation data in order to obtain valuable performance comparisons that can be used for further refinement of the methods developed.

In this work we mainly focus on evaluation, which has proven to be non-trivial due to sparse literature and research, unstandardized metrics, unstable implementations with varying quality of documentation and distinct features, evaluative datasets with varying quality, relevancy and representativeness.

1.2 Goals and Requirements

Throughout this work we pursued the following main objectives:

- First and foremost we must examine the sparse literature of text extraction methods in order to **produce a broader overview** of existing and related solutions. We examine the merits of each scientific paper relevant to the subject and include a short review of each work, respectively. Besides the review we also compare specific features of each method (e.g. language dependency).
- **Review the related work** of evaluating text extraction algorithms.
- By inspecting related works we select the appropriate **metrics** to conduct the evaluation. We explore the background of each metric and put them into the context of our task. We also define calculation procedures and handling of border cases of each metric, respectively.
- We **examine the existing datasets** and compare them against our requirements. Our requirements include clear annotation guidelines, existence of annotated and raw documents, appropriate document collection size, english only content, relevance and representativeness.
- Next we **develop a framework for evaluating various implementations of text extraction algorithms**. The framework has to be able to seamlessly perform the task of content extraction and evaluation on different datasets. The framework has to flexible enough so we can quickly integrate new algorithms, datasets and metrics.
- To conclude our work we provide an **interpretation of the evaluation's results**. To aid our effort, we extend our framework to produce visualizations of the results using various charts.

1.3 Terminology

In order to eliminate the ambiguity of certain words and phrases used throughout this work, we start by providing a contextual portrayal of such terminology.

Web page. In our context a web page is a resource that was published on the web and was requested by a client application, usually a web browser. By issuing a request for such resource we then retrieve the data that usually comes in the form of an HTML file and related media such as images, cascading style sheets (CSS), scripts and other media.

Web site. Since web pages come in the form of HTML-formatted data, they usually utilize *hyperlinks* in order to link related web pages which can be retrieved from the same server via the same domain name. Web pages retrieved from the same web site usually share numerous attributes as common layout, related textual content, shared visual perception and so on. In the context of this work we refer web sites when we point out the relationship between various web pages or when we allude to the shared resource of web pages.

(HTML) Document. We refer to documents in several contexts, but generally a document is a single file with HTML-encoded data. In the context of evaluation we refer to documents when defining metrics.

Web page cluster. A web page cluster is simply a set of documents which were derived from the same template. The cluster might have been obtained by a crawling process limited to a single web site. When rendered in a web browser alongside related media, such documents share overall visual perception and layout.

(Document) Dataset. In the context of evaluation we refer to datasets as collections of documents representing a domain of web sites.

Noise. In the context of retrieval, noise is considered all the unwanted information that was retrieved alongside relevant information. To put noise in our context, we define noise as elements of HTML-encoded data that impair the goals which an algorithm tends to achieve. For example, if the goal is to extract the main textual content, noise is defined as all the navigation, advertisement and layout-related content of the given web page.

Content extraction (CE). CE is the process of extracting content from the given resource. In our context a resource can be either a web site, a web page or a whole dataset. The content is defined as the main sought after entity of the extraction goal. For example, if our goal is to automatically extract the news text from an arbitrary news portal, the content is the article's text. Similarly, if we are to program a custom procedure that extracts a list of books and corresponding prices from the search results of an online book shop, the content is the tabular form of book titles and book prices.

Web page linearisation. Several content extraction techniques use a linearised web page representation as the main algorithm input. A linearised web page is usually created by first parsing it into a DOM tree and then the leaves of the tree are heuristically (the heuristic varies across implementations) joined together in a bottom up fashion. We end up with a sequence of semi-uniform content blocks.

Golden standard. In the context of our evaluation, we refer to the golden standard as the human annotated dataset upon which the test is based.

Text Extraction and Text Extractor (TE) Throughout this work we used a rather general term *text extractor* to distinguish a specific class of algorithms that solve the 3. problem statement defined in *Section 1.1.* These algorithms have the capability of extracting the main textual content from an arbitrary web page. To clarify, we might also use other terms to denote the same category – e.g. boilerplate removal algorithm, content block extractor or other terms frequently used by fellow authors.

Chapter 2

Field Overview

In this chapter we first provide a survey of related work dedicated to our evaluation. Next we overview the related work of content extraction (CE) and other related algorithms and their applications to various domains.

We covered the most notable and frequently-cited works that were were published in various scientific journals and conference proceedings. For those included in our evaluation framework (to be presented in the following chapters) we provided a more thorough examination. Apart from works published in academic literature, we also review some generally popular software and commercial solutions for CE.

Although we did our best to provide a general overview, we recognize our study is not fully comprehensive, due to the sparsity of the available literature.

2.1 Related Evaluation Work

In spite of the fact that nearly all works which present a particular CE approach include an evaluation of their methods alongside competing methods, there are not many that are specifically dedicated to evaluation itself.

2.1.1 Evaluation-Specific

In 2002, Laender et al. [30] presented a paper in which they covered a variety of CE tools. First they developed a *taxonomy for characterizing web data extraction tools* (they refer to CE as "web data extraction") and grouped the reviewed tools to their respective category. Some of the categories in the taxonomy include: HTML-aware tools, NLP-based tools and wrapper induction tools. We adopted some of the main concepts of this taxonomy in our overview of CE algorithms. They do not specifically evaluate the included tools on any data, but they provide a careful qualitative analysis. The most notable features of the analysis include: degree of automation (the amount of manual work), support for extracting complex structures and support for non-HTML sources.

In a more recent study, Gottron [19] evaluated a range of CE algorithms on HTML documents. The author provides the definition of metrics (mainly precision, recall and F1-score) used in his analysis alongside a clarification about different document representations based on levels of granularity. We found this paper especially useful since it gave us some initial intuition about document representation and framework design that we applied to our evaluation environment. CE tools included in the framework were evaluated in terms of precision, recall and F1-score on several small datasets.

The Cleaneval shared task project by *ACL Special Interest Group on Web as Corpus* [3] established a competition for cleaning arbitrary web pages, with the intention of harvesting a corpus that could be used for language and linguistic research. For the competitive part of the project, organizers provided a smaller development and a larger evaluation dataset of human annotated [4] HTML documents selected from arbitrary web sources. The final evaluation dataset was also included in our evaluation framework, since it met our scale and representativeness requirements.

Prior to this work we also published an overview and evaluation results using a smaller set of text extraction tools on our website. Alongside limited evaluation results [24], we produced a short field overview [28], a list of related works [27] and a brief comparison of CE tools [25]. We also reviewed some details of our evaluation approach [26]. Although the findings were never academically reviewed, we argue that the content itself provides a good starting point for fellow researchers.

2.1.2 Evaluation Approaches Within Related Works

As noted in the beginning of this section, many algorithm authors include performance evaluation and comparison against other competing algorithms. In this section we provide an overview of examples of such evaluations focusing especially on the metric definitions. We only inspect the publications that adopt relevant metric concepts.

Debnath et al. [12, 13] adopted a similar evaluation approach from Lin and Ho [31]. Both works used precision, recall and F1-score metrics and define content blocks as retrieval items. The downfall of using content blocks for calculating precision and recall is biased towards content blocks with little or no content in them. Such content blocks are considered equally as important as blocks with large text volume. The downside of this concept was also noted by Gottron [19].

Gottron also devised the CombineE framework [20] for testing, evaluating and optimising various combinations of different CE algorithms. Similar to our evaluation approach and similar to our evaluation framework, CombineE uses word sequences to represent the extracted and relevant text. The framework first breaks both retrieved and relevant text into word sequences and then computes the longest common subsequence to measure precision and recall.

Weninger and Hsu [45] defined two metrics in their evaluation environment, one based on longest common subsequence and the other based on edit distance. The two proposed metrics were not included in the extended work by Weninger et al. [46] where they adopted a standardized interpretation of precision and recall which were computed on word sets.

Pasternack and Roth [37] compared their algorithm against another competing method using metrics and data established by the Cleaneval shared task. They also included results based on similar metric concepts employed by previously mentioned Gottron works (precision and recall computed on word sequences). The latter results were computed on various single source and multi source datasets.

The work of Kohlschütter et al. [23] included an evaluation of their own algorithm and comparison against three Cleaneval contestants and the method devised by Pasternack and Roth [37]. The results were computed using the Cleaneval evaluation dataset and a collection of news articles harvested from Google News. We included the latter dataset in our evaluation framework, since it provided a good representation of news content.

2.2 Overview of Content Extraction Tools

Most of the related CE tools developed and studied over the last decade can roughly be arranged into four intersecting categories:

- In the following section we start our overview with **wrapper based** methods. Wrapper is a procedure that is capable of extracting data from HTML documents that were created using a similar template. These methods revolve around GUI applications that mitigate the tedious task of writing wrappers and wrapper induction techniques that produce the wrapper hypothesis automatically. The latter method is a step towards automated CE.

- We continue by listing a number of tools that address the problem of **detecting template elements** of HTML documents across web pages that are usually collected from a single web site. Some of these methods provide solutions for removing template noise from web search systems.

- Next we review algorithms and tools dedicated to **web page segmentation**. Usually a web page is presented by placing coherent pieces of content in proximity to another, so users can easily identify the conceptual structure. Segmentation tools are capable of detecting similar units of content using various heuristics and learning algorithms.

- Last, we provide a survey of methods that are closely related to our evaluation task. These methods are capable of removing boilerplate elements (navigation, footer, header, advertisements etc.) and extracting only the main textual content. Throughout this study we simply refer to them as **text extractors** (TE).

Alternatively we could group all the methods into two larger categories as suggested by Pomikalek [38] in his dissertation: **web site level methods** that operate on a set of documents from the same web site and **web page level methods** which take a single web page on the input and output information about the page structure.

2.2.1 Wrapper Based

One of the most obvious ways to extract data from a web page is to program a wrapper procedure. Such a program usually uses regular expressions and parsers integrated into a simple logic that follows a pattern of HTML elements

to the the sought after content. It is apparent that such a procedure must be programmed individually for every set of HTML documents that are derived from the same template. Such manual work is the bottleneck of this approach since wrapper programming is known to be a tedious, error-prone and time consuming task. Altogether, wrappers have proven to be a very useful technique when extracting tabular, listed and other semi-structured data.

The repetitive task of producing wrapper procedures can be mitigated by using wrapper generation tools as *XWRAP* by Liu et al. [32] or *NoDoSE* by Adelberg [1]. Both methods heavily rely on user interaction and user-labelled data to generate wrapper logic.

Introducing machine learning to the wrapper generation process leads us to wrapper induction techniques. A typical scheme of the wrapper induction process can be observed in *Figure 2.1*. Similarly, the input to the wrapper induction algorithm is a set of human-labelled HTML documents based on the same template. We feed the labelled training data to the learning algorithm that induces the wrapper logic. Wrapper induction was extensively studied by Kushmerick [29], who proposed wrapper classes which correspond to different styles of learning biases. To facilitate for more complex structures in a web page, Muslea et al. [36] propose the *STALKER* wrapper induction algorithm. While *STALKER* introduced some improvements regarding the magnitude of training data, the bottleneck of prior data labelling is still present in both approaches.

One attempt to eliminate costly manual work from the wrapper generation process was *ROADRUNNER* by Crescenzi et al. [11], where the wrapper logic is inferred by comparing two similar web pages and detecting the invariant and changing parts of both documents.

The *webstemmer* tool by Shinyama [41] further improved the wrapper induction process by utilizing a similarity criterion and a clustering algorithm to detect invariant features across a cluster of news-type web pages.

Any web data is likely to change over time and so are web sites that constantly get redesigned and such changes usually invalidate the wrapper. The concept of gradual structural evolution of web sites introduces a whole new category of problems to wrapper based methods such as wrapper maintenance [22], detection of template alteration and others that reach beyond the scope of this overview.

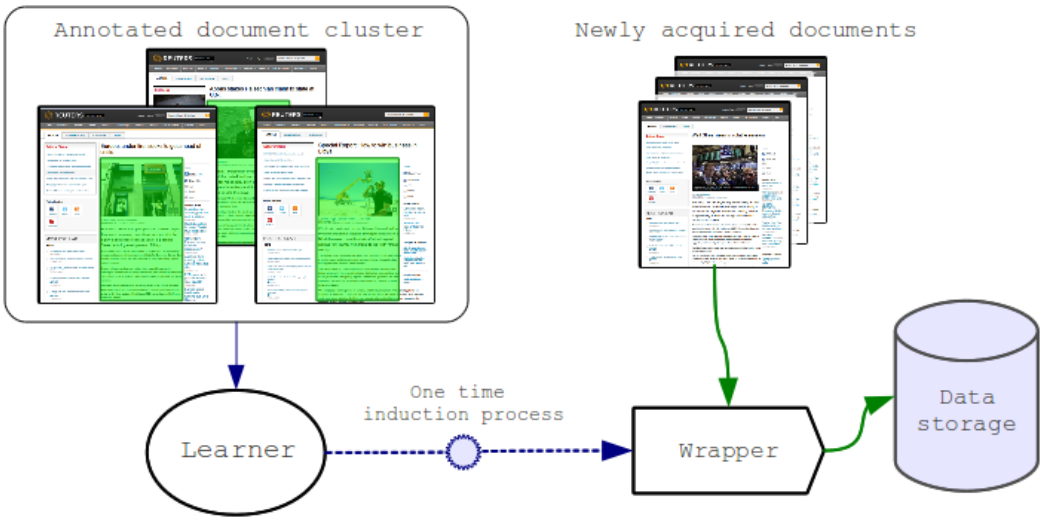


Figure 2.1: Typical scheme of the wrapper induction process. The blue lines indicate the induction process and the green lines indicate the extraction of newly acquired data.

2.2.2 Template Detection

Template detection is another phrase coined by researchers who have built algorithms for detecting noisy elements of the document and exposing the underlying template structure. Similarly to some wrapper induction methods, some template detection approaches use a cluster of documents based on the same underlying shared structure to detect "template" segments.

Template detection methods are closely related to CE approaches since detecting noisy elements of a web page and exposing the underlying template structure can be utilized for CE purposes.

To further expose the motivation for template detection methods we present the findings of Gibson et al. [17]. They observe that the volume of template material on the web, measured in bytes, is varying from $40 - 50\%$. Since this percentage in expected to grow, these findings pose a challenge for web search and data warehousing technologies, where detecting template-related content could slim down the costly storage and caching capacity of such systems.

One of the early contributions to this field was made by Bar-Yossef and Rajagopalan [5]. They introduced *pagelets* as non-intersecting regions of a website that have a distinct functionality.

Yi et al. [47] presented a method that employs a tree structure and an

information-based measure that outperformed the Bar-Yossef and Rajagopalan [5] approach. The input to the overall algorithm is a document cluster obtained from the same web site, which is then used to construct a generalized layout structure named *Site Style Tree* where an entropy based measure is employed to detect noisy elements.

Vieira et al. [44] exposed some downsides of the approach taken by Yi et al. [47] method such as requiring multiple passes over the a reasonably large cluster of web pages at the algorithm input Their improvement was based on an algorithm that finds an optimal mapping between DOM trees in a top-down fashion. In other words, they reduced the problem of template detection to finding a repeatable sub-tree structure among the input DOM trees. The problem of finding an optimal mapping between two trees is generally known as the *Tree Edit Distance* (TED) problem, which is analogous to the Levenshtein distance that applies to strings.
Alongside template detection, TED has other areas of application, for example, using TED for comparing abstract syntax trees of Java programs [40] and detecting data listing regions inside an HTML document [48].

Another algorithm pipeline dedicated to discovering informative content blocks from template-based HTML documents is *WISDOM* by Kao et al. [21]. Their overall system is capable of extracting informative blocks of content from a DOM tree of the web page by employing a top down search algorithm.

Lin and Ho [31], define an approach named *InfoDiscoverer* which is capable of detecting template elements in web page clusters by observing the occurrence frequency of extracted terms from blocks of content. Their approach is based on segmenting individual web pages into blocks of content by relying on a quite outdated assumption that a website is wrapped around a single large HTML `<table>` structure.

Recognizing that template content can impair web search, Chen et al. [9], tackled the template detection by measuring similarities between blocks of content (and subsequently removing noisy blocks) during the inverted index build process used by search engines.

2.2.3 Web Page Segmentation

In *Section 1.1* we provided a formal definition (*Problem 2.*) of the web page segmentation (WPS) problem. In this section we first expand the formal definition of WPS algorithms and then we proceed with the related works.

An ideal WPS algorithm is a human. Once the page is rendered, we humans have the capacity to visually and semantically detect coherent segments of the web page. We tend to discriminate between segments based on their functionality and content. The reason why the content creators tend to group data into segments is usually related to user experience and various design paradigms. Content creators want users to foresee the indented structure of the web site so they can easily find the sought after content. Therefore, a WPS algorithm would ideally detect segments just as good as a human would be able to detect them in a rendered web page.

WPS can be hypothetically and concretely applied to: discovering informative blocks of content as a part of a larger algorithm pipeline, link clustering during web crawling, removing ad cluttered and noisy segments, adapting web pages when browsing on small screens [2], improving web search heuristics by boosting the rank of documents that match the query terms across fewer segments as noted by [7].

The rest of this section is dedicated to the WPS related works.

One of the earliest WPS attempts was made by Chen et al. [10], as the part of adaptive browsing of web pages on small hand-held devices. The overall idea was to partition a web page into segments that can then be browsed individually, since screens of hand-held devices at that time (2003) were particularly small with limited resolution.

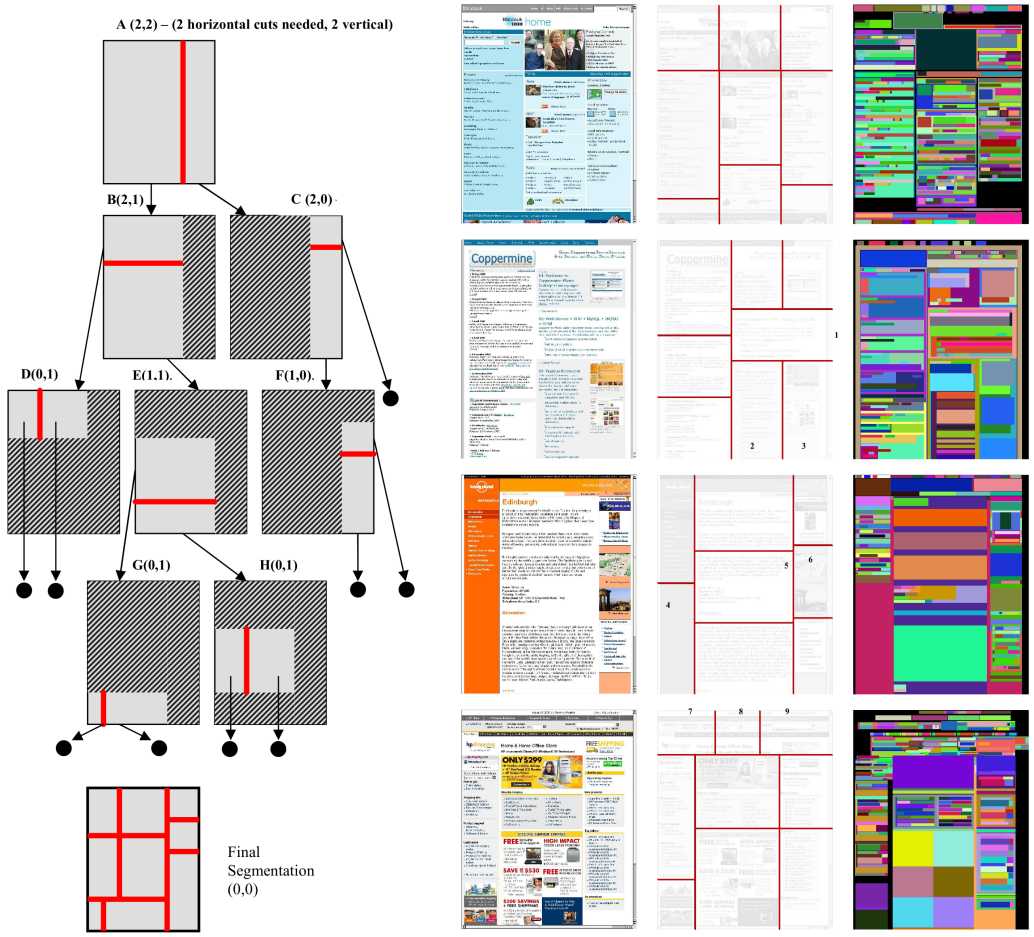
Another approach that tried to improve the browsing experience on mobile devices with small screens was presented by Baluja [2]. The authors argued that an intelligent way of segmenting a web page produced better browsing experience for mobile browsers at that time (2006). Concretely, their method is based on the observation that a web page segmentation problem can be reduced to the problem of inducing a decision tree during the learning phase of a typical classification task. Parts of the segmentation process can be observed on *Figures 2.2(a) and 2.2(b).*

VIPS by Cai et al. [6] is another attempt to leverage the visual perception of the web page in order to segment it into coherent blocks of content. The overall algorithm uses a top-down approach that starts by rendering the web

page using a browser and later segmenting the page by searching for separators between coherent segments of content.

Chakrabarti et al. [7] reduced the web page segmentation problem to the minimization problem by employing an adapted weighted graph structure.

Another site level segmentation method was developed by Fernades et al. [15], that suits web sites that are data intensive (e.g. web forums, institutional sites, e-commerce sites and similar). Such web sites usually contain large numbers of web pages whose sole purpose is to encode large amounts of data.



(a) Recursive segmentation process (b) Examples of segmentations

Figure 2.2: The recursive segmentation process 2.2(a) broken down into individual recursive steps.
Some examples of segmentations 2.2(b). The left column holds screen shots of four different web pages. The middle column shows the output of the segmentation algorithm and the column to the right holds the rendered DOM elements in various colours.
Taken from Baluja [2] – Both figures were included in our work after obtaining written permission from the author.

2.2.4 Text Extraction

In this subsection we reviewed the methods that are most relevant to our evaluation. Note that throughout this work we use a rather general term *text extractor* (TE) to distinguish this class of algorithms from the ones that were reviewed in the previous subsections (*2.2.1, 2.2.2, 2.2.3*). Just to clarify, we could also use other terms to denote the same category – e.g. boilerplate removal algorithm, content block extractor and or other terms frequently used by fellow authors.

Let us shortly revisit the problem that was defined in the introductory chapter: *Problem 3.* – "*Given a single HTML document, extract the meaningful textual content by removing or detecting all the boilerplate elements.*"

In the remainder of this section, we start by reviewing some of the methods that we did not include in our evaluation task due to either unavailable implementation (to the best of our knowledge) or non-trivial reimplementation, just to cover the breadth of the methods in this class. We then continue with an in-depth survey of methods that were either included in our evaluation task or have proven to be a novel TE approach. We also included a method that does not have a scholarly background, but was made famous on the World Wide Web and consequently also widely adopted and ported to many other languages - the readability *bookmarklet* [1].

Chakrabarti et al. [8] developed a page level *"templateness classifier"*, by generating training data from a dataset of randomly selected web sites across the web and extracting features from the compiled data. Their work can be boiled down to three main steps: compiling the training dataset by scraping web pages from the *Yahoo!*[2] search engine and heuristically labelling DOM nodes, preprocessing the dataset by annotating DOM nodes with a wide array of visual and heuristic features and employing the annotated data to train several logistic regression classifiers and finally smoothing the final classifier output that detects noisy content.

Gibson et al. [16] reduced the problem of TE to sequence labelling, whereas elements in the sequence are uniform fragments of text that were extracted from the DOM tree of the web page. Their method was applied to identify content blocks in news-type web pages.

[1] The rather popular and widely adopted term, "bookmarklet", is used to indicate a short piece of JavaScript code that gets executed when you click on a particular link or in the case of a bookmarklet, on a browser bookmark.

[2] `http://yahoo.com`

Note that the idea of *linearising* a web page DOM tree representation into a sequence of uniform text fragments or content blocks is a widely used technique continuously referred to throughout this work.
The authors argue that the sequence labelling approach is more appropriate when extracting non contiguous pieces of content that are interrupted by advertisement or other types of "in-line" noise. Before feeding the sequence to the statistical sequence labelling model, they extracted a wide array of content block level features as per block bag of words, extracted named entities, number of words inside an anchor tag[3], word count and others. According to their evaluation experiment the CRF variant of their overall algorithm has yielded the best results among the three developed models.

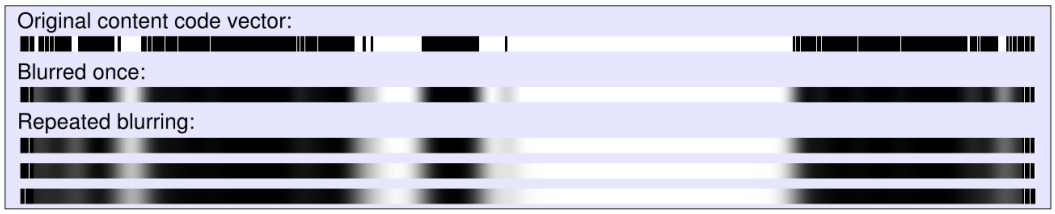


Figure 2.3: Visualization of a Content Code Vector. White - content class, black - code class.
Taken from Gottron [18] – The figure was included in our work after obtaining written permission from the author.

In a study by Gottron [18] a new approach to TE named Content Code Blurring was presented in order to address a simple observation that the main content segment of a web page contains mainly text and a fairly small amount of HTML code. Apart from the previous method that utilized a DOM-level content block linearized web page representation, this approach considers an even lower level of representation of a web page for the overall algorithm input – the character level sequence, where each respective character of the document is labelled as content or code class based on whether it belongs to a HTML tag or is considered to be part visible text. The author also applied a token-based representation where the document is broken down into HTML tag and word tokens. The algorithm operates on a sequence that represents individual tokens (or characters). First the sequence is initialized with real numbers, concretely, 1.0 for code class and 0.0 for content class. The author refers to

[3]tag: `<a> ... </a>`

this structure as the content code vector (CCV). The CCV is then subjected to multiple Gaussian filter passes in order to blur the border between content and code regions which can be observed in *Figure 2.3*. The content is then extracted using a fixed threshold to predict the final content region in a CCV.

The CleanEval shared task [3] set a new standard for evaluating TE algorithms at the time and also established a competitive platform for various researchers in this field.

The best performing algorithm for the 2007 CleanEval competition was the algorithm named Victor by Marek, Spousta & Pecina [42, 35]. Similarly to Gibson et al. [16], the Victor algorithm is built on the same principles of sequence labelling. Concretely, annotating the linearized document with various text and structure based features and classifying the annotated blocks as content or noise.

A tool named NCleaner by Evert [14] was originally based on an experiment that validated the applicability of n-gram based models to TE. The NCleaner algorithm pipeline can be summarized as follows: first they process the web page with a console based browser that retains the original paragraph block structure of text, next they fed text blocks into two pre-trained n-gram character-based language models, one trained to detect the boilerplate class of text and the other to detect the content class of text. A simple rule then determines if the block is retained or deleted from the final prediction. Concretely, if the boilerplate model predicts a higher probability as the opposite main content model then the block is discarded.
The author experimented with two different n-gram models. The first "lexical" model was trained on raw text and was very limiting since it was applicable only to documents that contain English content, the second "non-lexical" model was developed in order to facilitate the shortcomings of the former model, since it was trained on obfuscated text. The author wanted to validate that normalizing non-numeric characters to letter `a` and numbers to number `1` would produce a model that would mainly rely on punctuation and text features for the correct class prediction.

In a study by Pasternack & Roth [37] the authors explore an algorithm that is based on finding maximum subsequences in a vector of scores retrieved from a local token-level classifier in order to detect the main content of a web page. The overall algorithm starts by tokenizing the incoming HTML document by stemming all words and normalizing numeric tokens. Next local token level Naive Bayes classifier applies scores to the incoming sequence of tokens by observing two features: a token trigram and the current unclosed tag. A token

trigram is considered to be a vector of two consecutive tokens after the current position in the sequence of tokens. Concretely, given a sequence of tokens $t_1, t_2, ..., t_n$ the i-th token trigram is a vector $< t_i, t_{i+1}, t_{i+2} >$. The authors specify the second feature, "the most recent unclosed tag" as the tag that was opened before observing the i-th token in the sequence.
The algorithm then proceeds by transforming the output of the classifier by subtracting 0.5 from all results, thereby centring all values around 0 on the interval $[-0.5, 0.5]$.
The pipeline then employs the maximal subsequence segmentation to obtain the highest k-subsequences as the prediction for the start and finish of k-regions of text. This optimization can be solved in $O(n \cdot log(k))$ where n is the size of the whole sequence and k the number of sub-sequences we want to extract.

Next we review the formalization of the maximal subsequence optimization, where we are seeking only a single subsequence instead of k-subsequences:

$$(a, b) = argmax_{(x,y)} \sum_{i=x}^{y} s_i \tag{2.1}$$

Given a sequence of real valued scores $< s_1, s_2, ...s_n >$ we are seeking for a subsequence $< s_a, s_a + 1, ..., s_b >$ where the sum of the elements is maximal.

Given this optimization algorithm and the scores obtained by the local classifier, the final prediction is made by generalizing the local token-level prediction into a global prediction. Therefore the local classifier does not have to be entirely accurate or especially tuned for this task.

Kohlschütter et al. [23] (this paper provides the theoretical background for the boilerpipe library[4] that we included in our evaluation) and argues that a highly competitive algorithm can be built by observing only "shallow text features" of a particular web page instead of using the visual cues or functional properties of the text.
The overall method starts by typically linearising the incoming web page into uniform text chunks, based on the bottom layers of the DOM tree and annotating them with qualitative linguistic features, structural features and text-densitometric features. The authors experimented with decision trees and linear support vector machines for the learning task, which was conducted using a human labelled training set that was also used in our evaluation task. The learned classifier model is then employed to produce the final prediction for text blocks of a newly seen web page.

[4] `http://code.google.com/p/boilerpipe/`

The work of Weninger et al. [45, 46], is wrapped around another HTML document representation named the Text-to-Tag Ratio (TTR) array. The overall algorithm of constructing such an array can be observed in *Algorithm 1.*, whereas the array itself is depicted in *Figure 2.4.*

Algorithm 1 TRR array construction algorithm

Require: $h \leftarrow$ *HTML source code*
1: $h \leftarrow$ strip tags: `script`, `remark`
2: $h \leftarrow$ remove empty lines
3: **for** each line k to `numLines(`h`)` **do**
4: $x \leftarrow$ number of non-tag ASCII characters in $h[k]$
5: $y \leftarrow$ number of tags in $h[k]$
6: **if** $y = 0$ **then**
7: $TTRArray[k] \leftarrow x$
8: **else**
9: $TTRArray[k] \leftarrow x/y$
10: **end if**
11: **end for**
12: **return** $TTRArray$

As seen in *Algorithm 1.* the algorithm first strips tags from the original document and removes empty lines. It proceeds by computing a ratio between the number of text characters and the number of tags found in the current line. The ratio is computed for each line in the document, until it produces an array of positive real valued numbers.

In order to detect the main content in a document represented as a TTR array, the authors have applied several clustering techniques and developed a heuristic: "For each k in TTRArray, the higher the TTR is for an element k relative to the mean TTR of the entire array the more likely that k represents a line of content-text within the HTML-page." (a quote from [45]).
The authors experimented with several well known clustering techniques, including Expectation Maximization, Farthest-First and K-means, but they also applied a heuristic approach to clustering where the standard deviation of the smoothed TTR array was used as a cutoff threshold (depicted in *Figure 2.4*). They also experimented with "prediction clustering" where the clustering algorithm tries to detect sudden anomalies (drops or increases) while iterating through the array.

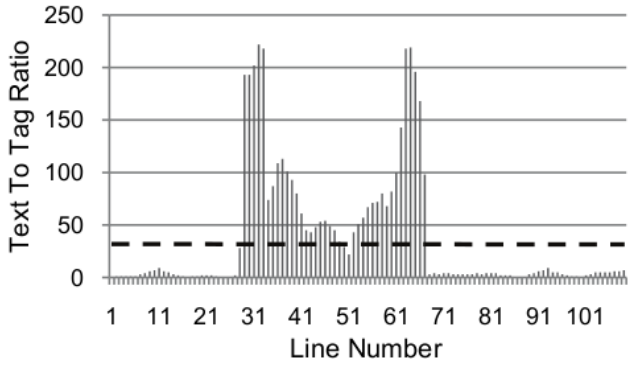


Figure 2.4: A smoothed TTR array depicted as a histogram, where the horizontal dashed line depicts the cutoff threshold.
Taken from Weninger et al. [45] – The figure was included in our work after obtaining written permission from the author.

In a recent study by Sun et al. [43], the authors proposed an algorithm that focuses mainly on text density features and heuristics derived through observation and experimentation. They employed one fairly common metric named text density that is analogous to the construction of the TRR array by Weninger et al. [45], except that the variant by Sun et al. is based on DOM nodes not lines in an HTML document. Through observation, the authors devised another novel feature named "Composite Text Density" (CTD). They argue that this feature is more informative that the regular text density, since it penalizes DOM nodes with too many hyperlinks that usually contain boilerplate content. The extraction phase itself was done by first propagating the sum of text density (or CTD) scores to the parents of the DOM nodes and by setting the cutoff threshold as the text density of the `<body>` DOM node. The algorithm then traverses the DOM structure and if it encounters a node with the text density higher then the set threshold it proceeds to find the child with the maximum propagated summed text density in the sub-tree of the original node. All the children found in this fashion are then marked as content.

In his dissertation Pomikalek [38] presented the jusText[5] algorithm which we also included in our evaluation environment. Similarly to other methods already reviewed in this section, jusText also employed a linearised document representation, by splitting text chunks on a fixed set of HTML tags. The author first applied a per block classifier, that annotates blocks with four classes - bad, good, near-good, and too-short.

[5]Source code can be found at: `http://code.google.com/p/justext/`

The overall algorithm then proceeds to re-evaluate the near-good and too-short classes provided by the initial classifier in order to capture a page-wide context of the block classification prediction. The "context sensitive classification" was based on the observation that the near-good and too-short labelled content blocks usually contain good content if they are grouped tightly with good blocks on both sides. The re-evaluation procedure is depicted in *Figure 2.5*.

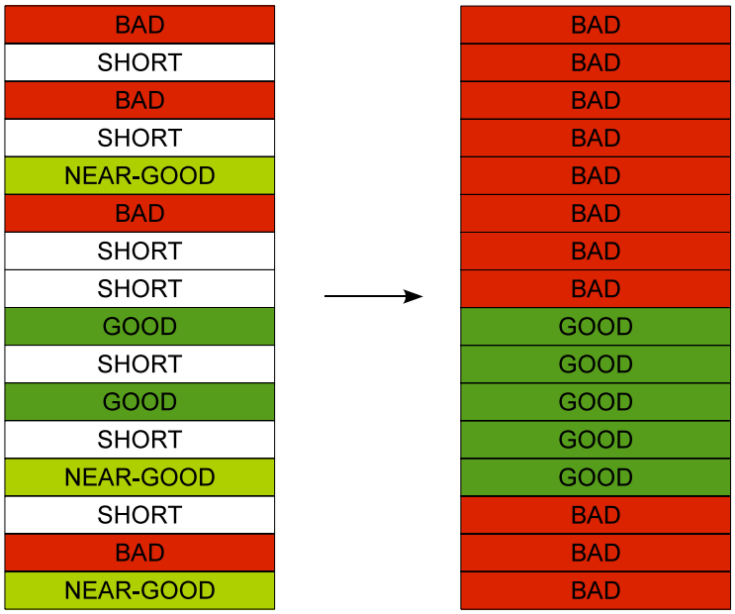


Figure 2.5: Visualization of the context sensitive classification in jusText.
Taken from Pomikalek [38] – The figure was included in our work after obtaining written permission from the author.

Perhaps one of the best known pieces TE software is the readability bookmarklet[6] that originated as an experiment of a company named *Arc90*[7]. Readability comes in the form of JavaScript code that gets inserted into and executed in a web page when the user clicks on a browser bookmark. The injected TE procedure then manipulates the DOM tree and removes the boilerplate elements.
The TE algorithm of readability is not backed by a strong theoretic foundation, but it is perceived to work well on popular news portals and blog type

[6]Source code available at: `http://code.google.com/p/arc90labs-readability/`
[7]`http://arc90.com/`

web pages, simply because it comes in a form of a very usable reading tool for browsers. Due to its popularity the overall algorithm was also ported to other languages as python, ruby, Java and C#.
The algorithm is constructed by numerous hand-crafted rules and heuristics that were developed through intuition and experimentation. It can can roughly be reviewed by the *Algorithm 2.* outline:

Algorithm 2 Readability algorithm outline

Require: $h \leftarrow$ *HTML source code*

1: $dom \leftarrow$ convert h to DOM
2: $dom \leftarrow$ remove `script` nodes
3: $title \leftarrow$ heuristically extract the title from `<title>` node
4: $dom \leftarrow$ remove nodes whose class names match a hand-crafted list of non-content class names
5: **for** each paragraph p in dom **do**
6: `// proceed by assigning a score to each paragraph`
7: $p.score \leftarrow$ assign a fixed score based on text density
8: `// propagate scores to ancestor nodes`
9: $parent \leftarrow getParentNode(p)$
10: $parent.score \leftarrow parent.score + p.score$
11: $grandParent \leftarrow getGrandParentNode(p)$
12: $grandParent.score \leftarrow grandParent.score + (p.score/2)$
13: **end for**
14: $topCandidate \leftarrow$ extract the top candidate node based on propagated scores in the previous FOR loop
15: $topCandidate \leftarrow findSiblingContent(topCandidate)$ `// heuristically determine if sibling nodes contain content`
16: **if** $length(topCanidate.text) < 250$ **then**
17: repeat the procedure with relaxed parameters
18: **else**
19: **return** $< title, topCandidate.text >$
20: **end if**

Commercial Web Services

Besides open source libraries and experimental implementations provided by researchers, there are also a number of commercial web-based APIs that can be directly used for TE. Since these web services fit well into the TE category and were easy to incorporate into our evaluation environment, we decided to include them.

In *Table 2.1* we list commercial web APIs that were used in our evaluation task and briefly review their features.

Note that the reviewed features were based on inspecting the respective documentation and direct correspondence with the support staff of the respective service.

Name	Features
Diffbot - Article API	• Uses a model that was trained on visual features for article-type pages. (The service supports extraction of structured data from other types of web pages - e.g. news portal indexes) • Implemented as an HTTP-based service with a JSON[8] encoded response. • Besides extracting the main content it can extract the title, embedded media objects, author name and other metadata. • Available at: `http://www.diffbot.com/docs/api/article`
Repustate - Clean HTML	• States that it "replicates the Readability functionality". • Implemented as an HTTP-based API with a JSON or XML encoded response. • Extracts only the main text of a web page. • Available at: `https://www.repustate.com/docs/#api-7`
Extractiv	• Capable of extracting the main content and annotating it with additional metadata, entities (and their respective types) and relations from the extracted content. • Implemented as an HTTP-based API with a JSON or XML encoded response. • Available at: `http://wiki.extractiv.com/w/page/30267488/On-DemandPlatform`
Alchemy API - Text Extraction	• Capable of extracting the main content. • Implemented as an HTTP-based service with a JSON, XML or RDF[9] encoded response. • As a product, Alchemy API offers a wide array of other text analysis related services including sentiment analysis, text categorization, language detection and entity extraction. • Available at: `http://www.alchemyapi.com/api/text/`

Table 2.1: Listing of commercial text extraction web APIs and their features.

Other software

Since TE tools are in high demand for a niche industry, a wide array of open source software was produced by various engineers.

The Goose[10], to expose the core functionality in a way that was suitable for our evaluation framework. The overall core algorithm uses a very similar heuristic that readability does and is also capable of extracting pictures and thumbnails from the given web page.

As mentioned, readability was ported to many languages and the deviation from the original core functionality varies among the language ports. For our evaluation task we experimented with a Python port[12], since our evaluation framework was also written in Python and a port[13] to Node.js[14], which was built on top *jsDOM*, a DOM API implementation in JavaScript.

[10]The original repository is available at `https://github.com/jiminoc/goose/wiki` library was also included in our evaluation task, but for our purposes we had to modify the source code[11]

[12]Available at: `https://github.com/gfxmonk/python-readability`

[13]Available at: `https://github.com/arrix/node-readability`

[14]Node.js is a server side implementation of the JavaScript runtime.

2.3 Applications

Content extraction in general is a very applicable field to many corners of academia and industry. Here we provide a brief review of some notable applications of research and some cases where the product itself depends heavily on TE, web page segmentation and template detection methods.

Perhaps one of the most notable applications of research, outdated due to the advances of mobile devices, is the aforementioned work of Baluja [2]. At the time, some hand held devices provided a very simple interface for browsing web pages. The web page was segmented into nine equally-sized rectangular regions that map against the numeric keyboard of the cellular phone. Baluja argued that such oversimplified segmentation can be improved by performing intelligent non-uniform segmentation that follows visual and structural cues in a web page.

One fairly obvious application of the whole field in general is the application of template detection and segmentation to web search and information retrieval. Let us briefly review some previously mentioned applications:

- Chen et al. [9], approached template detection by adding a detection component in the index building pipeline for large scale search engines.
- Although Chakrabarti et al. [7] did not directly apply his segmentation method to web search, they noted that information retrieval could obviously benefit from web page segmentation by inserting bias for multi word queries that span multiple detected segments of a web page.

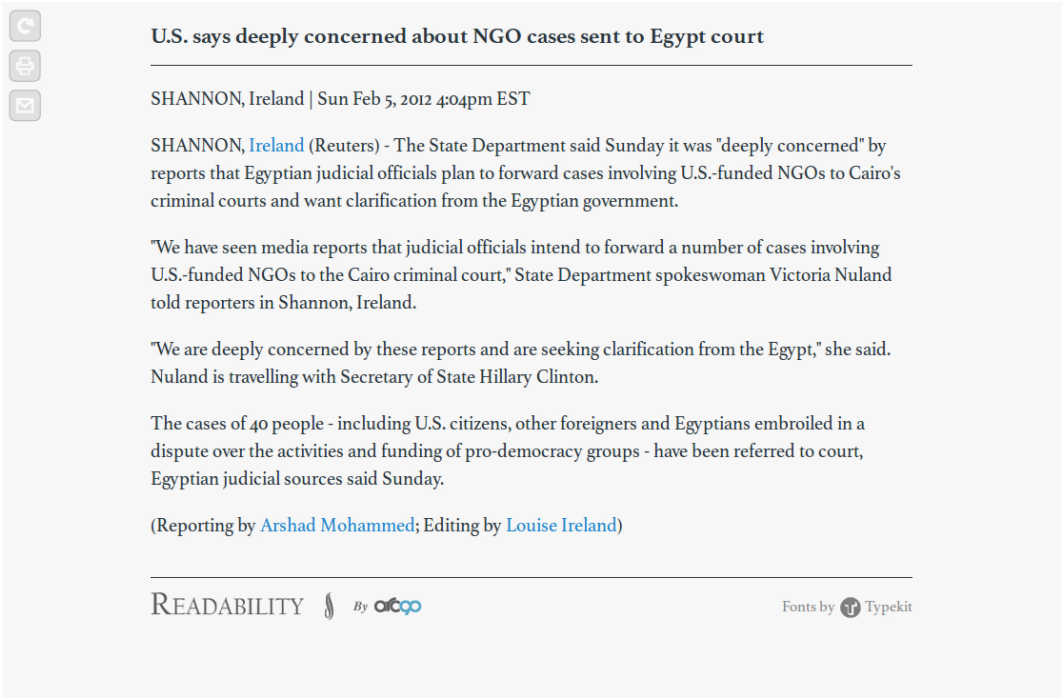
The authors of Chakrabarti et al. [7] also applied segmentation as a preprocessing layer in the near-duplicate detection pipeline based on shingling.[15]. In a typical shingling-based near-duplicate detection algorithm, all segments of a web page are considered to be equally important. As a proof of concept they first extracted the largest segment of a web page and then shingled only the content under the designated segment instead of using the whole web page as classic shingle-based near-duplicate detection algorithms do.

Another obvious application for TE was exposed by the CleanEval shared task [3], where the original goal was to employ competitive research in order

[15] *Shingling* is a well know approach to near duplicate detection on the world wide web. A *k-shingle* is considered to be a set of k consecutive terms in a document. Where two documents can be represented as a set of shingles. Comparing such sets by an overlap measure is the basis of a near duplicate detection algorithm. A more detailed description of *shingling* can be found in Manning et al. [33]

to yield top-performing methods for web data preparation. Concretely, a high performing TE algorithm could be used as a noise removal component in a pipeline for harvesting large web-crawled corpora, which can later be used for linguistic and language-related research.

As mentioned earlier, the readability bookmarklet was originally developed to improve the reading experience of ad-cluttered news and article content. The user first creates a bookmark on the browser window panel with embedded JavaScript code. The user then browses onto the desired web page and clicks on the bookmark. When clicked, the embedded code is executed and it injects additional JavaScript code in the browsed web page which then performs the main TE algorithm. When the main content is extracted, the original DOM tree gets replaced by a content placeholder that provides a much cleaner reading interface. Such a placeholder can be viewed on *Figure 2.6*.



U.S. says deeply concerned about NGO cases sent to Egypt court

SHANNON, Ireland | Sun Feb 5, 2012 4:04pm EST

SHANNON, Ireland (Reuters) - The State Department said Sunday it was "deeply concerned" by reports that Egyptian judicial officials plan to forward cases involving U.S.-funded NGOs to Cairo's criminal courts and want clarification from the Egyptian government.

"We have seen media reports that judicial officials intend to forward a number of cases involving U.S.-funded NGOs to the Cairo criminal court," State Department spokeswoman Victoria Nuland told reporters in Shannon, Ireland.

"We are deeply concerned by these reports and are seeking clarification from the Egypt," she said. Nuland is travelling with Secretary of State Hillary Clinton.

The cases of 40 people - including U.S. citizens, other foreigners and Egyptians embroiled in a dispute over the activities and funding of pro-democracy groups - have been referred to court, Egyptian judicial sources said Sunday.

(Reporting by Arshad Mohammed; Editing by Louise Ireland)

READABILITY By arc90 Fonts by Typekit

Figure 2.6: A Reuters web page enhanced by the readability bookmarklet. Source: `http://www.reuters.com/article/2012/02/05/us-egypt-ngo-usa-idUSTRE8140PX20120205`

Similar to readability, the instapaper service[16] provides delivery of the extracted content to dedicated hand-held reading devices such as the Amazon Kindle.

A TE algorithm is also being used by a service named print-friendly[17] which tries to improve the experience of printing web pages. In some cases, when printing a web page onto paper, we are left with pages that contain the actual content we wanted to print and useless pages that contain the boilerplate content. The service also comes in a form of a bookmarklet. When clicked, the user is presented with the extracted content, which can be manually cleaned before the user decides that the content is ready for printing.

TE can also be used for creating web feeds[18] that contain full article content, since a large portion of web feed entries come only with a short segment of the article. In order to obtain a web feed with full content, the service must first crawl the web pages the feed originally links to, extract the content and map the content back to the feed. A service named *Full Text RSS Feed Builder*[19] implements such functionality.

[16] `http://www.instapaper.com/`

[17] `http://www.printfriendly.com/`

[18] A web feed is a data format for delivering frequently updated content to readers. (e.g. RSS and Atom XML-based formats)

[19] `http://fulltextrssfeed.com/`

Chapter 3

Means for an Evaluation Environment

In order to measure the performance of TE algorithms one must secure the following two means:

1. **Dataset(s)** – A collection of document pairs. Each pair consists of a raw HTML document and a human-labelled document. Concretely, we need a human expert who will lay down the golden standard for each document.

2. **A Metric** that is suitable for this specific domain and is capable of measuring how well does the output of the TE algorithm compares to the golden standard set by the human expert across the dataset.

In this chapter we first explore the means for the evaluation task of this work and identify the main problems. We continue by studying the two main means for our evaluation task - the metrics and datasets, that we employed.

3.1 Problem Identification

When gathering all the means necessary for evaluating TE algorithms one quickly encounters the following array of problems:

- unstandardised metrics with varying interpretations or implementations,
- sparse dataset documentation,
- unstandardised annotation guidelines for building the golden standard,

- unstandardised dataset annotation format,
- varying dataset scale and breadth,
- absence or lack of dataset metadata and related media (e.g. information about original encoding, CSS files, images),
- absence or lack of raw data (i.e. dataset does not contain the HTML document in its original form).

We will refer to some these problems throughout the remainder of this chapter.

3.2 Datasets

In order to perform an evaluation of various TE a dataset must meet the following requirements:

- Raw HTML documents must be included so we can pass them various TE and gather results.
- The dataset must include documents that were annotated for content by humans.
- The dataset domain must fit the context of TE. Concretely, the dataset must contain web pages that hold some form of textual content (i.e. article, news and blog pages).
- The dataset must be sparse, that is, it must contain web pages from many web sites since TE algorithms are designed to extract content from arbitrary web sites without prior knowledge or internal state of a particular web site.
- The content of the dataset documents has to be predominantly written in the English language, since some TE are language dependant and usually fitted to the English language.

While conducting our research, we immediately encountered two datasets that fulfilled all of the previously listed requirements: the Cleaneval dataset[1] and the L3S-GN1 dataset[2].

[1] At the time of the initial phase of our research, the dataset was available at `http://cleaneval.sigwac.org.uk/`

[2] The dataset is publicly available at `http://www.l3s.de/ kohlschuetter/boilerplate/`

3.2.1 L3S-GN1 Dataset

The dataset codenamed L3S-GN1 was compiled and made publicly available by the authors of Kohlschütter et al. [23]. This dataset fits into a very narrow domain of news-type web pages, since it contains 621 web pages that hold news articles from 408 distinct web sites (with a maximum of 12 web pages per web site). The final web pages themselves were selected from a much larger pool (concretely: 254,000 web pages from 7,854 web sites) that was acquired by scraping the Google News stream for a period of approximately 6 months.

The data was labelled using a "web browser based" text annotator by 7 different people. The text was labelled with 5 explicit and 1 implicit label: `headline`, `fulltext`, `supplemental` (e.g. article image captions, article-published timestamps and other similar metadata), `user comments`, `related content` (e.g. links to related articles and/or content).
Any text that was not annotated with any of the explicit labels is treated as `non-content` text.

The labels are presented as `<span>` HTML tags wrapped around individual blocks of text with special classes assigned to them. Each `<span>` class represents one label. An example of an L3S-GN1 formatted document can be observed in *Figure 3.1.*

```
...
<h1>
        <span class="x-nc-sel1">Film/TV directors, producers plan contract talks</span>
</h1>
<div class="timestampHeader">
        <span class="x-nc-sel3">Fri Jan 11, 2008 6:45pm EST</span>
</div>
...
```

Figure 3.1: Example of a L3S-GN1 labelled document. Note the class names of `<span>` tags. Each label has its own tag class assigned with the following structure `x-nc-sel[numbers from 0 to 5]`. In the short except above, the tag class `x-nc-sel1` denotes the `headline` label and `x-nc-sel3` denotes the `supplemental` label of text.

The dataset ships in a single WARC[3] file or as compressed individual files. Raw and labelled documents come in pairs whose file names match through a unique ID. The dataset also contains a mapping between unique document ID pairs and original URLs.

[3]The WARC File Format (ISO 28500) `http://bibnum.bnf.fr/WARC/`

To the best of our knowledge, there were no annotation guidelines provided alongside the dataset.

3.2.2 Cleaneval Dataset

The Cleaneval dataset was compiled as a part of the Cleaneval shared task and competition that took place in 2007 on extracting text from arbitrary web pages. The aim of this competition was to produce best performing means for TE in order to be able to prepare web pages for a web scraped corpora that can later be utilized for "linguistic and language technology research and development".

In order to create a competitive environment for all participants, the organizers have compiled two[4] human annotated datasets (golden standard) for benchmarking and development purposes. The smaller development set of documents contained 58 documents and was distributed to all competitors while the larger set containing 681 documents was used for evaluation purposes. In our evaluation task we utilized the latter dataset.

Cleaneval provides the annotators with detailed guidelines [4], which propose a set of definitions of boilerplate elements of an arbitrary web page. The annotators were required to open the web page in a browser in one window and the preprocessed text in a text editor in the other window. The preprocessing step was made by a custom script that removed all tags, embedded CSS and JavaScript from the document and outputted blocks of text separated by whitespace.

With the browser in one window and the text editor in another window the annotator is then required to remove all boilerplate text blocks from the preprocessed text. As previously mentioned Cleaneval defines a definition of boilerplate content:

> Boilerplate is often machine-generated, and includes (but it is not necessarily limited to):
>
> - navigation information,
> - internal and external link lists,
> - copyright notices and other legal information,

[4]Besides the English language based datasets organizers also provided a development and evaluation dataset that predominantly contains content written in the Chinese language, but for our evaluation purposes we only employed the two English language based datasets.

- standard header, footer and template materials that are repeated across (a subset of) the pages of the same site,
- advertisements,
- web-spam, such as automated postings by spammers to blogs.

Quote taken from [4].

After boilerplate removal the annotators then added predefined code back to the cleaned text blocks. Concretely, `<p>`, `<h>`, `<l>` opening tags were added the beginning of each text block that represented either a paragraph, heading or a list member.

An example of the annotation procedure in the Cleaneval format of a web page from Reuters news portal[5] can be observed in *Figure 3.2*.

```
...
<div class="column2 gridPanel grid8">
<h1>Insight: The Wall Street gold rush in foreclosed homes</h1>
<div class="facebookRec" tns="no">
...
<span id="midArticle_0"></span><span class="focusParagraph">
<p><span class="articleLocation">NEW YORK</span> (Reuters) - Dan Magder
recently gave up a top job with private equity firm Lone Star Funds to
strike out on his own and become a landlord.</p>
</span><span id="midArticle_1"></span><p>He's joining a growing list
of big and small investors who see fat profits to be made in renting
out foreclosed homes, especially now the U.S. government is moving
ahead with a trial project to sell big pools of single-family homes
that Fannie Mae currently owns in some of the hardest-hit
housing markets.</p><span id="midArticle_2"></span>
...
```

```
<h1>Insight: The Wall Street gold rush in foreclosed homes

<p> NEW YORK (Reuters) - Dan Magder recently gave up a top job with
private equity firm Lone Star Funds to strike out on his own and
become a landlord.

<p> He's joining a growing list of big and small investors who see
fat profits to be made in renting out foreclosed homes, especially
now the U.S. government is moving ahead with a trial project to sell
big pools of single-family homes that Fannie Mae currently owns in
some of the hardest-hit housing markets.
```

Figure 3.2: Example of a Cleaneval annotation procedure. In the figure above, one can observe the HTML markup of a Reuters news web page and the respective annotation result on the figure below.

[5] http://www.reuters.com/article/2012/03/20/us-usa-foreclosures-investors-idUSBRE82J12M20120320

The dataset was harvested by issuing Google search queries and downloading the retrieved web pages. Therefore, the Cleaneval dataset does not cover a specific domain of web data as the L3S-GN1 dataset does for news type web pages.

3.3 Metrics

In the field of classic information retrieval (IR) two metrics are predominantly used to evaluate the effectiveness of IR systems – **precision** and **recall**. In order to put these two metrics in the context of evaluating the performance of TE algorithms, we first explored their classic IR meaning and later provided the definition of the targeted context.

3.3.1 Precision and Recall

Both metrics can be defined alongside a modified confusion matrix[6] in *Table 3.1* (as seen in Manning et al. [34])

	relevant	non-relevant
retrieved	true positives (TP)	false positives (FP)
not retrieved	false negatives (FN)	true negatives (TN)

Table 3.1: Confusion matrix as the visualization of results of an IR system. Rows contain instance counts of the predicted class and columns contain the ground truth – instance counts of the actual class.

Precision and recall can now be defined in the following equations 3.1 and 3.2.

$$Precision = \frac{TP}{TP + FP} \tag{3.1}$$

$$Recall = \frac{TP}{TP + FN} \tag{3.2}$$

[6]The confusion matrix was adopted from the classic artificial intelligence field where it usually provides a visualization of the results of a supervised learning algorithm.

Note that in a classic IR environment we are usually operating with a large set of skewed data. Therefore well-known metrics such as classification accuracy (*Equation 3.3*) are misleading. Concretely, if we would be presented with a trivial IR system that always predicts the non-relevant class operating on large scale data typically encountered in a web search environment. The accuracy of such a flawed IR system would be very close to 100% since the small scale of TP gets lost in the numerator of the accuracy equation (*3.3*).

$$Accuracy = \frac{TP + TN}{\#(total\ dataset\ instances)} \tag{3.3}$$

Since we are faced with two metrics used to evaluate the effectiveness of a single system, we must be able to calculate the trade-off. For this purpose we use the **F score**. F score is a weighted harmonic mean of precision and recall, defined by the following equation:

$$F_\beta\ score = \frac{(\beta^2 + 1)P \cdot R}{\beta^2 \cdot P + R} \tag{3.4}$$

Where P stands for precision and R for recall and $\beta \in [0, \infty]$, where β is used to emphasize the importance of either precision or recall. With $\beta > 1$ we emphasize recall and conversely with $beta < 1$ we emphasize precision. With $beta = 1$ we get a balanced score named $F_{\beta=1}$ or F_1 score. We can now eliminate β from the equation and write down the final formula for the F_1 score below:

$$F_1\ score = \frac{2 \cdot P \cdot R}{P + R} \tag{3.5}$$

Let us now rewrite the precision and recall equations in terms of relevant and retrieved items produced by a IR system and then continue within the TE context.

$$Precision = \frac{\#(relevant\ items\ retrieved)}{\#(retrieved\ items)} \tag{3.6}$$

$$Recall = \frac{\#(relevant\ items\ retrieved)}{\#(relevant\ items)} \tag{3.7}$$

3.3.2 Document models

Consider a typical IR system as a search engine, where the items are documents stored in an inverted index, but in a TE context the items are fragments of content on an arbitrary web page.
In order to define what a fragment of content is we inspect several document models on different levels of granularity (some document models were documented by Gottron [19]):

Bag of words (BOW) – BOW model, being one of the classic document models used in the field of natural language processing, only retains a set of words and their frequency in the original document. Since it discards the word order and any grammatical structure, this document model is not suitable for evaluating TE algorithms. Concretely, if we encounter multiple instances of the word in both content and non-content regions of a web page, we have no way of determining the class of the respective word instances.

Set of words – A document represented as a set of words discards even more information about the original document as the BOW model does. It only retains a unique set of words found in the document.

Sequence of characters – This is the lowest level of granularity for a document model. It is only suitable if we decode each character correctly from it's byte representation. Conversely, such model would introduce a fair amount of fragmentation when calculating the sequence intersection between two instances of such a document model.

Sequence of words (SOW) – Words in the exact same sequence from the original document is arguably the most suitable document model for the task of evaluating TE algorithms.

Sequence of text blocks – We can also separate text into blocks by retaining paragraphs and other test separators that are found in the original HTML structure. As mentioned in *Section 2.1.2*, such a document model is biased towards content blocks with little or no content in them. Such text blocks are considered equally important as blocks with large text volume.

Since we were dealing with multiple datasets, with different golden standard formats and unreliable information about the encoding of raw and golden

standard data, we decided that it was best to utilize the Sequence of words (SOW) document model for our evaluation task.

3.3.3 Precision and Recall in Text Extraction Context

Precision and recall for TE evaluation are computed by first obtaining the result of the TE algorithm for each document in the dataset. Next, we put the result and the respective golden standard document into SOW form. We continue by computing the intersection between both sequences using a well-known sequence matching algorithm. We then use the length of the intersection and lengths of both sequences to compute the per-document precision and recall:

$$P_i = \frac{|Seq_{relevant} \cap Seq_{retrieved}|}{|Seq_{retrieved}|} \tag{3.8}$$

$$R_i = \frac{|Seq_{relevant} \cap Seq_{retrieved}|}{|Seq_{relevant}|} \tag{3.9}$$

Where P_i and R_i are per-document precision and recall of the i-th result – golden standard pair. $|Seq_{relevant} \cap Seq_{retrieved}|$ denotes the length of the intersection of two sequences, where $Seq_{relevant}$ stands for the SOW of the golden standard and $Seq_{retrieved}$ is used to denote the result of the TE algorithm. Similarly $|Seq_{relevant}|$ and $|Seq_{retrieved}|$ denote the length of individual sequences.

With P_i and R_i defined we can also define the per-document F_1 score:

$$F1_i = \frac{2 \cdot P_i \cdot R_i}{P_i + R_i} \tag{3.10}$$

We compute P_i, R_i and $F1_i$ for each document in the dataset and then compute the average overall per-document measurements:

$$P_{avg} = \frac{1}{N} \cdot \sum_{i=1}^{N} P_i \tag{3.11}$$

$$R_{avg} = \frac{1}{N} \cdot \sum_{i=1}^{N} R_i \tag{3.12}$$

$$F1_{avg} = \frac{1}{N} \cdot \sum_{i=1}^{N} F1_i \tag{3.13}$$

Where N is the number of documents in the dataset.

With the final formal definition, we can now provide a practical interpretation of such metrics in the context of TE performance.

- If we observe high recall this implies that the algorithm is good at correctly extracting the entire relevant content, with possible chunks of non-content text.
- Conversely, if we observe high precision, this indicates the capability of cleaning out non-content text chunks from the retrieved text produced at the algorithm output.

During our study of related works we encountered a very similar interpretation of precision and recall to that which we presented above. This particular observation was [43, 46, 37] already mentioned in *Chapter 2*.

Conversely, the Cleaneval shared task used a metric based on the Levenshtein distance, a measure that yields the amount of difference between two strings in terms of the number of edit operations as deletion, substitution and insertion required to transform one string into the other. Although, the Levenshtein distance originally operates on the character level, Cleaneval adapted the measure by eliminating substitutions and modified it in order to operate on the token level since they also adopted the SOW model.

3.3.4 Boundary Cases

Since we are dealing with unreliable web data to compute precision and recall we must also cover all boundary cases of both metrics. Concretely, we must account for per-document precision, recall and F1 score boundary cases listed in *Table 3.2*.

Precision	**Recall**	**F1 Score**	**Description**
0	0	∞	Mismatch – Both $Seq_{relevant}$ and $Seq_{retrieved}$ sequences are not empty, but they don't intersect.
∞	0	NaN	The TE extracts no content in the given document, thus $Seq_{retrieved}$ is empty.
0	∞	NaN	The golden standard document is empty, thus $Seq_{relevant}$ is empty.
∞	∞	NaN	The TE extracts no content in the given document and the golden standard document is empty, thus $Seq_{retrieved}$ and $Seq_{relevant}$ are both empty.

Table 3.2: Boundary cases of precision, recall and F1 score

Chapter 4

Evaluation Environment Implementation

In order to perform the evaluation task, it was necessary to build a non-trivial amount of infrastructure that would allow us to gather evaluation results from several TE libraries written in different languages, commercial and research-only web services and a script that is intended to run in a browser.

To be able to perform such a task we set out to implement a TE evaluation framework with the following architectural requirements and goals:

- the integration of TE software must be seamless,
- all TE should be abstracted using a simple and shared interface,
- the framework must support evaluation on multiple datasets, with different golden standard formats and metadata,
- the framework must be capable of storing and later parsing raw TE results in arbitrary formats,
- the computation of precision, recall and F1 score presented in *Section 3.3.3* must be supported,
- the framework must be capable of capturing all errors and boundary cases of previously mentioned metrics,
- the capability of producing visualization of the results must also be supported.

In this chapter, we will first present the overall architecture of our evaluation environment. Next we will review the actual implementation and its details: raw data format, dataset preprocessing, implementation of library service wrappers and details related to the metric computation.

4.1 Evaluation Framework Architecture

As mentioned in the introduction to this chapter, all TE software that we included in our evaluation environment was made available either as libraries written in different programming languages, as web services or as a browser application. In order to meet the requirement that dictates us to provide a shared interface for all TE tools, we decided to expose the majority of existing TE libraries as simple HTTP-based web services since we will already have to integrate the ones that come only in the form of a web service.

We chose the Python programming language to be the core of our environment, since we had to glue together many components and libraries for parsing, storing and serializing data, communicating with various services and producing visualizations. Python already includes a very strong set of built-in tools and has proven and tested third-party libraries that cover all of our requirements for such a task.

In *Figure 4.1* we lay out the overview of the general architecture of our evaluation framework. As previously mentioned, we exposed most of the freely available TE libraries as web services which function alongside other remotely available commercial and research-only services, exposed to the framework through a shared interface.

At the core of our framework we implemented three major components:

Data manipulator – The main data manipulation component is responsible for storing and loading TE results, loading and decoding raw data and passing it to the TE components and producing error logs.

Evaluation manager – In this component domain is everything related to the actual evaluation: computing and serializing the per-document metric measurements and computing statistics over per-document measurements. It also contains logic for parsing different formats of the golden standard.

Services wrapper – This component contains all the framework-side TE wrappers that implement a shared interface.

The core of the framework is also connected to two local file-system storages:

Dataset storage – a collection of files that contain raw HTML documents and golden standard documents.

Result storage – for each extractor and each document in each dataset we store the raw result from each TE in its original format.

Since we wanted to keep the storage as simple as possible, we only use the local file-system and store all the files in a predefined directory structure. The data manipulation component then relies on this directory structure and naming scheme in order to retrieve the designated contents of the file.

4.2 Dataset Storage and Preprocessing

As previously mentioned we use the local file system in combination with a directory and file naming schema to implement the dataset storage. Since both datasets used in our evaluation environment (Cleaneval and L3S-GN1) come with their own document naming conventions, metadata format and golden standard format we decided that we will introduce a separate layer of logic that extracts and serializes metadata into a YAML[1] formatted file for each dataset. The Cleaneval dataset also required us to introduce a preprocessing step that will be presented in the remainder of this section.

[1] YAML is a human friendly data serialization format. `http://yaml.org/`

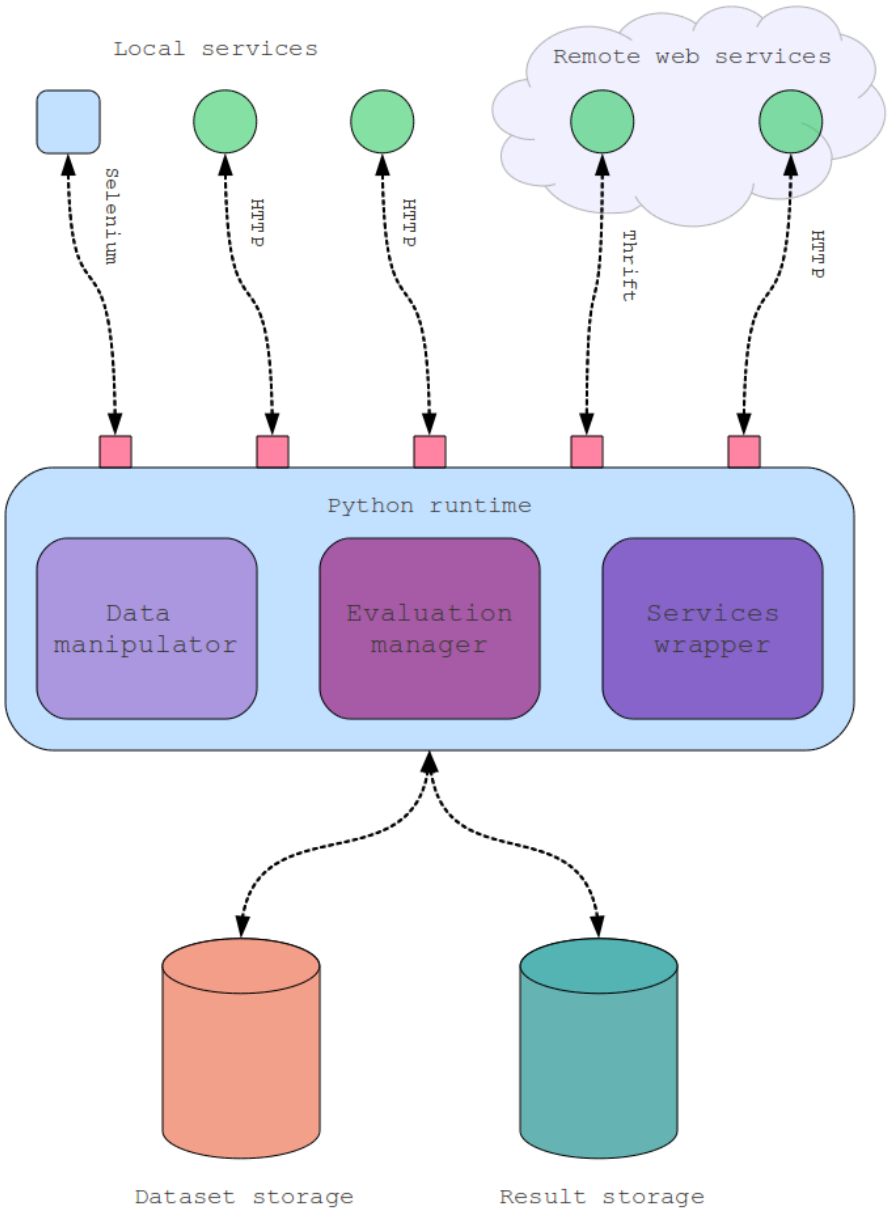


Figure 4.1: A high level abstraction of the evaluation framework architecture, where green circles represent various remote and local web based services exposed via HTTP or Thrift, with the exception of the blue square that stands for the browser that was controlled via the Selenium web driver. The red squares denote a common interface for all TE services to the core of the Python-based TE evaluation framework. The framework consists of three main components that are connected with two storages.

First we rearranged the files of both datasets in the following directory structure:

```
.
‘-- datasets
    |-- cleaneval-final
    |   |-- clean
    |   |-- raw
    |   ‘-- meta.yaml
    |
    |
    ‘-- google-news
        |-- clean
        |-- raw
        ‘-- meta.yaml
```

Where the `clean` directory contains the the golden standard documents and the `raw` directory holds the raw HTML documents. Each dataset also contains the `meta.yaml` file that serves a mapping between clean documents and their raw counterpart along with other metadata that was extracted from the dataset.

The `meta.yaml` file for each dataset was produced by a script that goes through all the raw and clean documents of both datasets and extracts a list of attributes for each raw / golden standard document pair. Let us now observe some examples of the YAML serialized metadata extracted from both datasets and the preprocessing step that was done only on the Cleaneval dataset.

For L3S-GN1 dataset the script extracts the following attributes for a single raw / golden standard document pair:

```
- clean: 4a3cd9c0-153b-4e30-bf72-6c0cd5133e1e.html
  clean_encoding: utf-8
  id: 4a3cd9c0-153b-4e30-bf72-6c0cd5133e1e
  meta:
    encoding_confidence: 0.87624999999999997
  raw: 4a3cd9c0-153b-4e30-bf72-6c0cd5133e1e.html
  raw_encoding: utf-8
```

Where `id` is reserved for a unique identifier for each raw / golden standard document pair and its value is used to map the TE results back to the original document pair. The `clean` and `raw` attributes hold the file names of the golden standard and their raw HTML counterpart in their respective directories. The `clean_encoding` and `raw_encoding` fields contain the name of the detected encoding for their respective files. The `meta` attribute is reserved for any dataset-specific attributes. In the case of L3S-GN1 we used `meta` to store the confidence of the encoding detection algorithm if we could not extract encoding from the HTML `<meta>` tag or other standard encoding information markup, since, to the best of our knowledge, the L3S-GN1 dataset does not contain any information about the original encoding of the files.

Alongside the similar metadata extraction for document pairs, the Cleaneval dataset required a raw data preprocessing step in order to remove a special XML tag which can be observed on *Figure 4.2* that wrapped the whole raw document and contained some metadata about the document itself.

```
<text id="http://flakmag.com/film/sunshine.html"
    title="Flak Magazine: Review of  Sunshine State, 8-9-02"
    encoding="iso-8859-1">

<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 3.2 Final//EN">
<html>
<head>

...

</text>
```

Figure 4.2: Example of the Cleaneval raw data format

During our evaluation we noticed that some TE require that the `<html>` DOM node is available in the root of the DOM tree, so we applied a preprocessing step using regular expressions to remove the opening and closing `<text>` tag from raw documents.

Prior to deleting the `<text>` tag we also extracted all the metadata and stored it into the `meta.yaml` file, where we store the following attributes for each document pair in the dataset:

```
- clean: 763.txt
  clean_encoding: utf-8
  id: '763'
  meta:
    encoding: iso-8859-1
    id: http://www.tribuneindia.com/2002/20020831/nation.htm
    title: The Tribune, Chandigarh, India - Nation
  raw: 763.html
  raw_encoding: iso8859-1
```

Where the `meta` field contains all the data that was extracted from the `<text>` tag.

With both datasets in the prescribed directory structure and the extracted meta data, the data manipulation component is now ready to serve the contents of the raw or golden standard files to TE wrappers or to the evaluation manager component.

Since some commercial TE APIs required that the raw document must be passed to the HTTP call as a public URL, it is also worth mentioning that we created a script that deploys the raw counterparts of each dataset to a remote server and makes them publicly available via HTTP.

4.3 Integration of Text Extraction Services

As already noted in the *Section 4.1*, where we presented the overall architecture, we exposed most of the freely available TE libraries as HTTP-based web services and integrated them into our evaluation framework alongside other remote commercial and research-only services. We also integrated one remote service that was made available to us remotely via Thrift[2].

Most libraries used were written in Java, concretely, Goose library, Boilerpipe library and the TRR implementation [45]. To expose the core functionalities of these libraries we built a standalone HTTP application using the play framework[3].

[2]Apache Thrift is a cross-language framework for creating and exposing services. `http://thrift.apache.org/`

[3]Play is a lightweight framework for developing web applications in Java. `http://www.playframework.org/`

Similarly we also implemented a HTTP-based application in node.js[4] for the node.js port of the Readability bookmarklet.

A single TE service was integrated using a wrapper that implements an interface shared among all integrated TE. An example of such a wrapper around the commercial Alchemy API TE can be observed in *Figure 4.3*.

There were three exceptions regarding our integration of TE tools. Two TE algorithms were already implemented in Python so we simply imported the respective libraries and implemented the wrapper logic. The third exception was the integration of the core functionality of the original Readability bookmarklet.

Since Readability was made to run in a browser we had to utilize the Selenium web browser automation framework[5] and its WebDriver API[6] using the Mozilla Firefox browser. Concretely, we instructed the Firefox WebDriver to execute a JavaScript function when the page is loaded in the browser. The JavaScript then injects the DOM tree with a `<script>` node that executes the Readability algorithm. Since the original implementation by arc90 labs (i.e. the authors of Readability) inserted some additional boilerplate elements after the extraction process for self promotion and better reading experience, we modified it in way that allowed us to capture failures to extract content and we also removed the part of the code that inserted additional boilerplate elements in the extraction result.

After the script finished we simply used WebDriver to retrieve the contents of a certain `<div>` node in the DOM tree that contained the extracted content.

4.4 Computing per Document Measurements

In *Section 3.3.3* we already presented the formal definition of how to compute precision, recall and F1 score in the TE context. Here we review implementation details of how to compute the numerators of *Equations 3.8 and 3.9* denoted as $|Seq_{relevant} \cap Seq_{retrieved}|$ where $Seq_{relevant}$ stands for the sequence of relevant words and $Seq_{retrieved}$ for the sequence of words retrieved by the TE algorithm. Also recall that we used the sequence of words model for both the golden standard text and the retrieved text from TE.

[4]Node.js is a platform for building server side applications in JavaScript. `http://nodejs.org/`

[5]`http://seleniumhq.org/`

[6]`http://seleniumhq.org/docs/03_webdriver.html`

```python
class BaseExtractor(object):
    '''Extractor base class to prescribe a common representation'''

    NAME = ''# unique name
    SLUG = ''# unique slug name ([a-z_]+)
    FORMAT = ''# txt|html|json|xml

    def __init__(self, data_instance):
        self.data_instance = data_instance

    def extract(self):
        '''Returns unformatted extractor resposne'''
        pass

    @classmethod
    def formatted_result(cls, result_string):
        pass
```

```python
class AlchemyExtractor(BaseExtractor):
    '''Alchemy API extractor'''

    NAME = 'Alchemy API'
    SLUG = 'alchemy'
    FORMAT = 'json'

    @check_content_status
    @return_content
    def extract(self):
        html = self.data_instance.get_raw_html()
        req = Request(
            'http://access.alchemyapi.com/calls/html/HTMLGetText',
            data = {'apikey':settings.ALCHEMY_API_KEY,
                    'html': html.encode(self.data_instance.raw_encoding,'ignore'),
                    'outputMode':'json'
            }

        )
        return req.post()

    def _content_status(self):
        js = json.loads(self._content, encoding = 'utf8')
        if js['status'] == 'ERROR':
            raise ContentExtractorError(js['statusInfo'].encode('utf8','ignore'))

    @classmethod
    def formatted_result(cls, result_string):
        js = json.loads(result_string, encoding = 'utf8')
        return TextResultFormat(js['text'].encode('utf8','ignore'))
```

Figure 4.3: Example of TE wrapper and the shared TE interface. The `BaseExtractor` class prescribes a shared interface for all TE wrappers. The `AlchemyExtractor` class is an implementation of such an interface for the commercial web service Alchemy API TE. It overrides the main class attributes `NAME`, `SLUG` and `FORMAT` and it implements two core methods: the `extract` method is called by the data manipulation component for each document in the dataset and it it also relies on some framework internals that call the private method `_content_status` for each extraction call in order to capture any errors. The evaluation manager then calls the `formatted_result` class function that contains all the business logic for obtaining the result in the shared format across all TE included in the framework.

The evaluation manager component of our evaluation framework contains logic that parses the golden standard format for both datasets and retrieves only the cleaned text that is later put into the SOW representation. Recall from *Section 3.2* that the Cleaneval format simply prepends some additional opening tags to each paragraph of the golden standard text, whereas the L3S-GN1 dataset inserts specific class names for `<span>` tags to wrap nodes that contain different classes of content.

Retrieving the cleaned text from the Cleaneval golden standard format was done by simply removing the opening tags using regular expressions. For the L3S-GN1 dataset we first had to decide which classes of labelled text should be included in the golden standard. We decided to use only classes named *full text* and *headline* as the golden standard since they seemed to contain equivalent content as the Cleaneval annotation guidelines seemed to prescribe.

After obtaining the golden standard text from the human annotated document, we parsed the stored result and obtained the retrieved text for the TE. We obtained such a document pair for every document in the dataset and for every TE.

Now we converted the text to the SOW model by means of the text tokenization procedure presented in *Algorithm 3*.

Algorithm 3 Text Tokenization Procedure

Require: $t \leftarrow$ *raw text*

`// replaces punctuation with whitespace`

1: $t \leftarrow$ remove_punctuation(t)
2: $t \leftarrow$ remove_control_characters(t)
3: $t \leftarrow$ remove_non_ASCII_characters(t)
4: $t \leftarrow$ lowercase_all_characters(t)
5: $Seq \leftarrow$ split_into_tokens(t) `// goes through the character sequence and splits the text into tokens upon whitespace`
6: **return** Seq

With both $Seq_{relevant}$ and $Seq_{retrieved}$ sequences, where the former contains the SOW from the golden standard and the latter holds the SOW from the TE result we computed the intersection between the two.

We computed the intersection between both sequences of words by utilizing the Python built-in `difflib` library for matching arbitrary sequences[7]. This

[7]Evert [14] also utilized python `difflib` to compute the intersection between two relevant and retrieved word sequences.

library contains an adapted implementation of the algorithm published in 1980 by Ratcliff and Obershelp[8]. The implementation details of the final intersection computation and the consequent computation of per-document precision and recall can be observed in *Figure 4.4*.

With the length of the intersection of both sequences, we continued to compute and store the precision, recall, and F1 score for every document in the dataset. Finally we computed the average of such per-document measurements for all three metrics using *Equations 3.11, 3.12 and 3.13*.

```
1  class TextOnlyEvaluator(BaseEvaluator):
2
3      def get_eval_results(self):
4
5          s = difflib.SequenceMatcher()
6          rel = self.relevant.get_word_seq()
7          ret = self.retrieved.get_word_seq()
8
9          s.set_seqs(rel, ret)
10         matches = s.get_matching_blocks()[:-1]
11
12         rel_union_ret = sum(i.size for i in matches) if len(matches) > 0 else 0
13
14         precision = float(rel_union_ret) / float(len(ret)) \
15                     if len(ret) > 0 else float('inf')
16         recall = float(rel_union_ret) / float(len(rel)) \
17                     if len(rel) > 0 else float('inf')
18
19         # nan when prec or recall are inf
20         f1_score = (2. * precision * recall)/(precision + recall) \
21                     if precision + recall > 0 else float('inf')
22
23         return Result(precision, recall, f1_score, self.id)
```

Figure 4.4: First (line 6) we obtained the relevant sequence of words form the golden standard and the retrieved sequence of words (line 7) from the TE. In lines 9–10 we computed the lengths of the matching regions of both sequences (where we ignore the last dummy region which just holds the lengths of both sequences). With the total length of the matching region of both sequences computed in line 12, we compute precision, recall and F1 score from lines 14 to 20.

[8]We did not manage to obtain the original paper, but the publication by Ratclif and Metzener [39] from 1988 brings forth insight about the original algorithm.

Chapter 5

Evaluation Results

In this chapter we present the final evaluation results produced by the framework presented in *Chapter 4*. We evaluated 12 distinct TE tools and several of their implementation variations, resulting in a total of 17 TE algorithms. The algorithms were evaluated on two datasets presented in *Section 3.2*.

In the remainder of this chapter, we first provide a detailed list of all evaluated TE. We proceed with the presentation of evaluation and finally we interpret and discuss the final results.

5.1 Final set of Text Extraction Algorithms

Let us now present all TE algorithms that we included in our evaluation environment.

As noted in the introductory part of this chapter we included 12 distinct TE that were reviewed in *Section 2.2.4* and several implementation variations. Such implementation variations differ either in terms of the programming language used, internal state or different parameter values. By including such variations we end up with a total of 17 TE methods that are listed in *Table 5.1*.

Name	Variation	Reference	Description

Boilerpipe	Default
	Article
	Article Sentence
		`Boilerpipe DEF`
		`Boilerpipe ART`
		`Boilerpipe SENT`
			The boilerpipe library was written by the authors of Kohlschütter et al. [23]. The library contains several different variations that are implemented as Java classes.

jusText library		`JusText`	Open source python library presented in Pomikalek [38].
Goose library		`Goose`	Open source Java library presented in *Section 2.2.4*.
MSS		`MSS`	TE method presented by Pasternack & Roth [37] was made available to us as a web service by the authors.

Readability	Python
	Node.js
	Original
		`Python Readability`
		`Node Readability`
		`Readability`
			The original readability bookmarklet and the two ports presented in *Section 2.2.4*.

Alchemy API		`Alchemy API`	The commercial web service was reviewed in *Table 2.1*.
Diffbot API		`Diffbot`	The commercial web service was reviewed in *Table 2.1*.
Extractiv API		`Extractiv`	The commercial web service was reviewed in *Table 2.1*.
Repustate API		`Repustate`	The commercial web service was reviewed in *Table 2.1*.
Zextractor		`Zextractor`	The TE service internally used by Zemanta Ltd. made available via Thrift.
NCleaner	English Non Lexical	`NClerner En` `NClerner NonLex`	Library by Evert [14] and the two n-gram model variations. The english n-gram model and the non lexical n-gram model.
TTR		`TTR`	TE by Weninger et al. [45]. We obtained the freely availably source code from the author's web site.

Table 5.1: Final list of evaluated TE methods. The column *variation* lists implementation variations of a specific TE method. The *reference* column provides a reference name for a specific algorithm used throughout this chapter.

Note that for the Readability algorithm we included the original JavaScript implementation, a Python port and a port to Node.js. We first evaluated the latter two implementations, where we observed a high number of boundary cases alongside low precision and recall which lead us to question the quality and representativeness of the two implementations of this algorithm. We decided to evaluate the original JavaScript implementation from Arc90 Labs, which reduced the number of boundary cases and has proved to perform well on both datasets.

5.2 Precision, Recall and F1-score

Let us now review the final evaluation results in terms of average precision, recall and F1-score over per-document measurements as already defined by *Equations 3.11, 3.12 and 3.13*.

The results for each algorithm can be observed in the tabular form in *Table 5.2* and *Table 5.3* for L3S-GN1 and Cleaneval dataset, respectively.

L3S-GN1 dataset			
Text Extractor	***Precision***	***Recall***	***F1-score***
`Boilerpipe ART`	0.9312	0.9554	**0.9388**
`Readability`	0.9233	**0.9699**	0.9379
`Boilerpipe SENT`	**0.9624**	0.9193	0.9362
`Diffbot`	0.9240	0.9679	0.9351
`JusText`	0.9121	0.9168	0.9012
`Goose`	0.9340	0.8866	0.9008
`Repustate`	0.9171	0.9069	0.8978
`Alchemy API`	0.9356	0.8760	0.8883
`Boilerpipe DEF`	0.8628	0.9376	0.8872
`MSS`	0.9070	0.8819	0.8746
`Extractiv`	0.8237	0.9564	0.8703
`NCleaner En`	0.7421	0.9467	0.8120
`Zextractor`	0.8497	0.8057	0.8032
`Node Readability`	0.6879	0.9682	0.7891
`TTR`	0.6538	0.9523	0.7473
`NCleaner NonLex`	0.6128	0.9590	0.7226
`Python Readability`	0.6375	0.8255	0.6808

Table 5.2: Average precision, recall and F1-score for L3S-GN1 dataset. Sorted by F1-score.

Cleaneval dataset			
Text Extractor	***Precision***	***Recall***	***F1-score***
`Readability`	0.9345	0.9035	**0.9009**
`NCleaner En`	0.9233	0.8949	0.8972
`Repustate`	0.9400	0.8892	0.8960
`JusText`	**0.9619**	0.8668	0.8954
`NCleaner NonLex`	0.8820	**0.9268**	0.8924
`Diffbot`	0.9321	0.8899	0.8910
`Extractiv`	0.9345	0.8709	0.8873
`TTR`	0.8911	0.8888	0.8737
`Boilerpipe DEF`	0.9313	0.8561	0.8720
`Alchemy API`	0.9504	0.8281	0.8542
`Boilerpipe ART`	0.9485	0.7643	0.8041
`Python Readability`	0.8407	0.8332	0.8031
`Zextractor`	0.9164	0.7625	0.8004
`Boilerpipe SENT`	0.9587	0.7373	0.7919
`Goose`	0.9343	0.7189	0.7697
`Node Readability`	0.7037	0.8779	0.7530
`MSS`	0.9109	0.6994	0.7182

Table 5.3: Average precision, recall and F1-score for the Cleaneval dataset. Sorted by F1-score.

We also provide a visualization of the results using bar charts in *Figure 5.1* and *Figure 5.2* for both datasets.

Note that we sorted the algorithms in descending order on the horizontal axis for each metric respectively. Naturally, the vertical axis denotes the computed average for each metric. Each bar was also equipped with an error bar that represents the standard deviation of per-document metric measurements. We argue that the variability of per document metric calculations are also an important characteristic of a TE algorithm.

We observed that the top performing algorithm in terms of F1-score for the Cleneval dataset was `Readability`, which was rather surprising, given the fact that this algorithm has no particular scholarly or theoretic background, but was built using domain-specific intuition and experimentation.

Similarly, the top performing algorithm in terms of precision for the same dataset, `JusText` also relied on tuning some classification parameters by intuition.

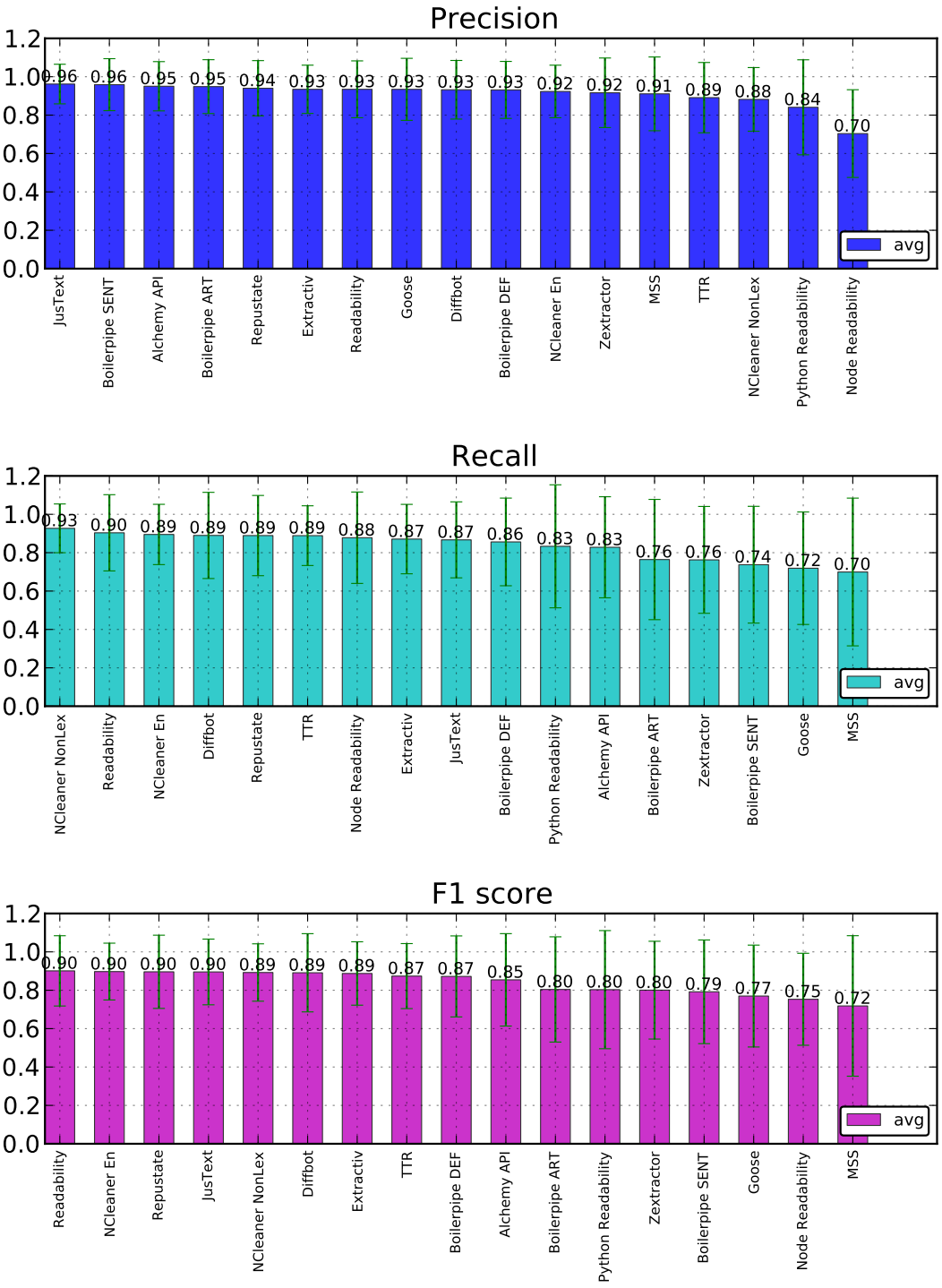


Figure 5.1: Cleaneval dataset results.

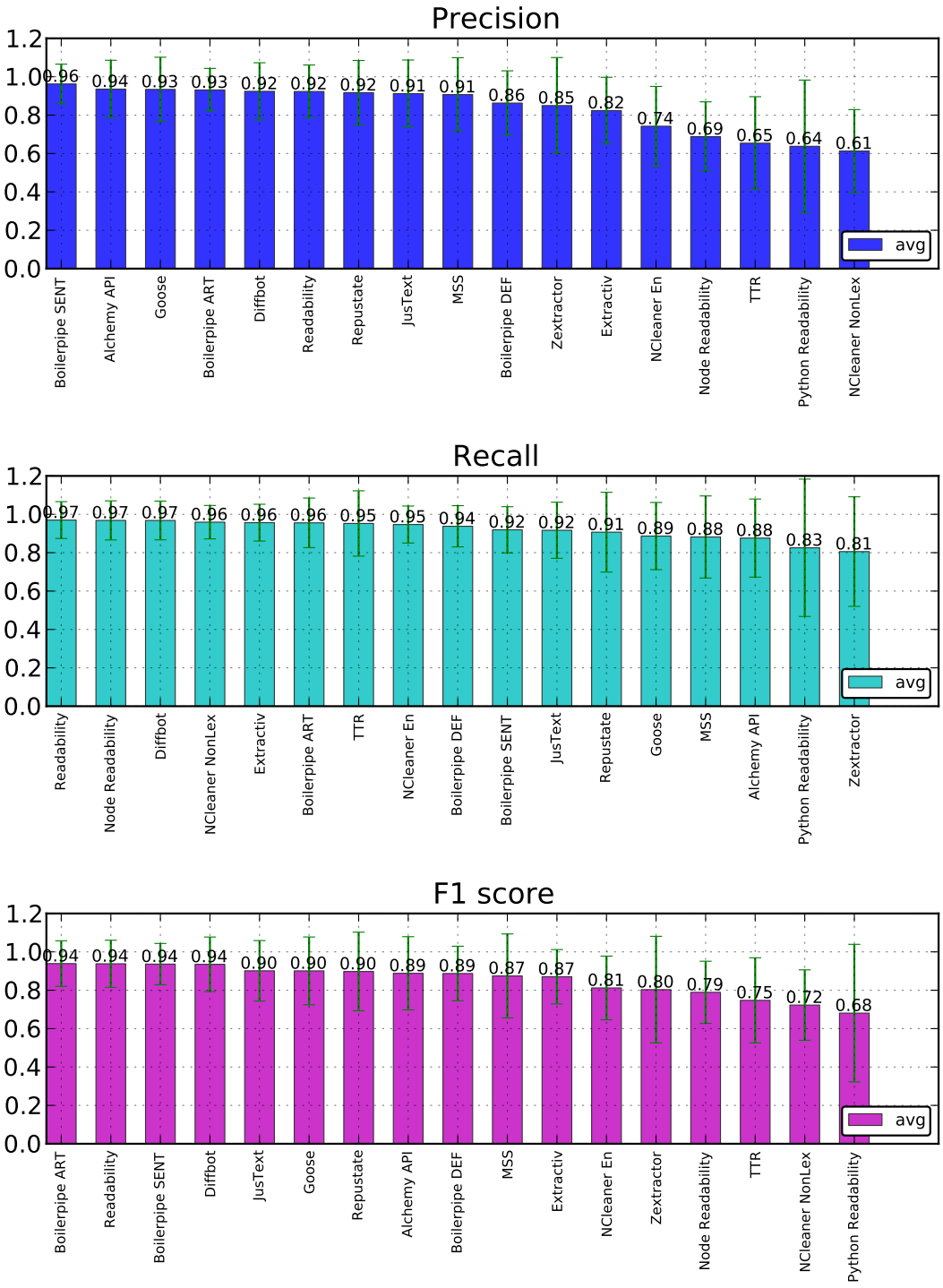


Figure 5.2: L3S-GN1 dataset results.

We also observed poor performance and high variability of the `MSS` algorithm which significantly differed from results of the evaluation benchmark on the same dataset the authors presented in their paper [37]. The authors might have provided us with a different implementation of the algorithm, since our results are clearly distant from theirs.

The top performing algorithm for L3S-GN1 dataset was clearly `Boilerpipe ART` in terms of F1-score and `Boilerpipe SENT` in terms of precision. It was expected that Boilerpipe and its implementation variations would perform well on this domain, given the fact that the authors of the original paper which provided the theoretic background for this library used this dataset to train the block level classifier.

We also noticed a noteworthy lower F1-score for both `NCleaner En` and `NCleaner NonLex` on the L3S-GN1 dataset when compared to the Cleaneval results. Since NCleaner was part of the Cleaneval competition and considering the fact that Cleaneval represents a cross-domain set of web pages which are expected to contain more noise when compared to the news domain, we conclude that NCleaner was over-fitted to the Cleaneval dataset, since it mainly relies on features of the contents of the text itself and not the structure of web pages and other related features.

Similar than NCleaner, the `TTR` algorithm experienced the same drop of performance on L3S-GN1. The manual inspection of a small set of the results returned by the algorithm revealed that the implementation of this algorithm had problems of cleaning out large portions of HTML and JavaScript code. To the best of our knowledge this was the main cause of varying results on this dataset.

All the commercial solutions seem to perform reasonably well on both datasets judging from their resulting F1-score, with Diffbot and Repustate API being the top performing APIs on both L3S-GN1 and Cleaneval datasets. Other commercial APIs seem to follow closely.

We also provided an additional visualization of per-document metric measurements for both datasets in *Figures 5.3 and 5.4*. For each algorithm we plotted the distribution of per-document measurements as histograms with multiple bins per metric (precision, recall and F1-score). Concretely, we divide the $[0, 1]$ interval into 20 equidistant intervals and then count the number of per document measurements for each interval. The count is then reflected in the vertical length of an individual bar in the histogram. We do so for each metric and each algorithm, respectively.

We argue that the visualization of such metric distributions can serve as an

extra feature that provides more information about the performance characteristics of an algorithm on a particular dataset. For example, the data variability across different datasets can easily be detected using this visualization.

Concretely, we observed a particular long tail distribution of precision per document measurements for `NCleaner NonLex` in *Figure 5.3* for the Cleaneval dataset and a flat distribution for precision in *Figure 5.4* for L3S-GN1 dataset. This observation coincides with the conclusion about over-fitting the algorithm to a specific dataset made previously in this section.

For algorithms that experienced high precision or high recall with low variability of per-document measurements (e.g. `Boilerpipe SENT` in *Figure 5.3*), we observe an exponential distribution that captures nearly all measurements in the last bin.

We also observed some weak indications of a binomial distribution of recall for `Python Readability` and `MSS` on the Cleaneval dataset.

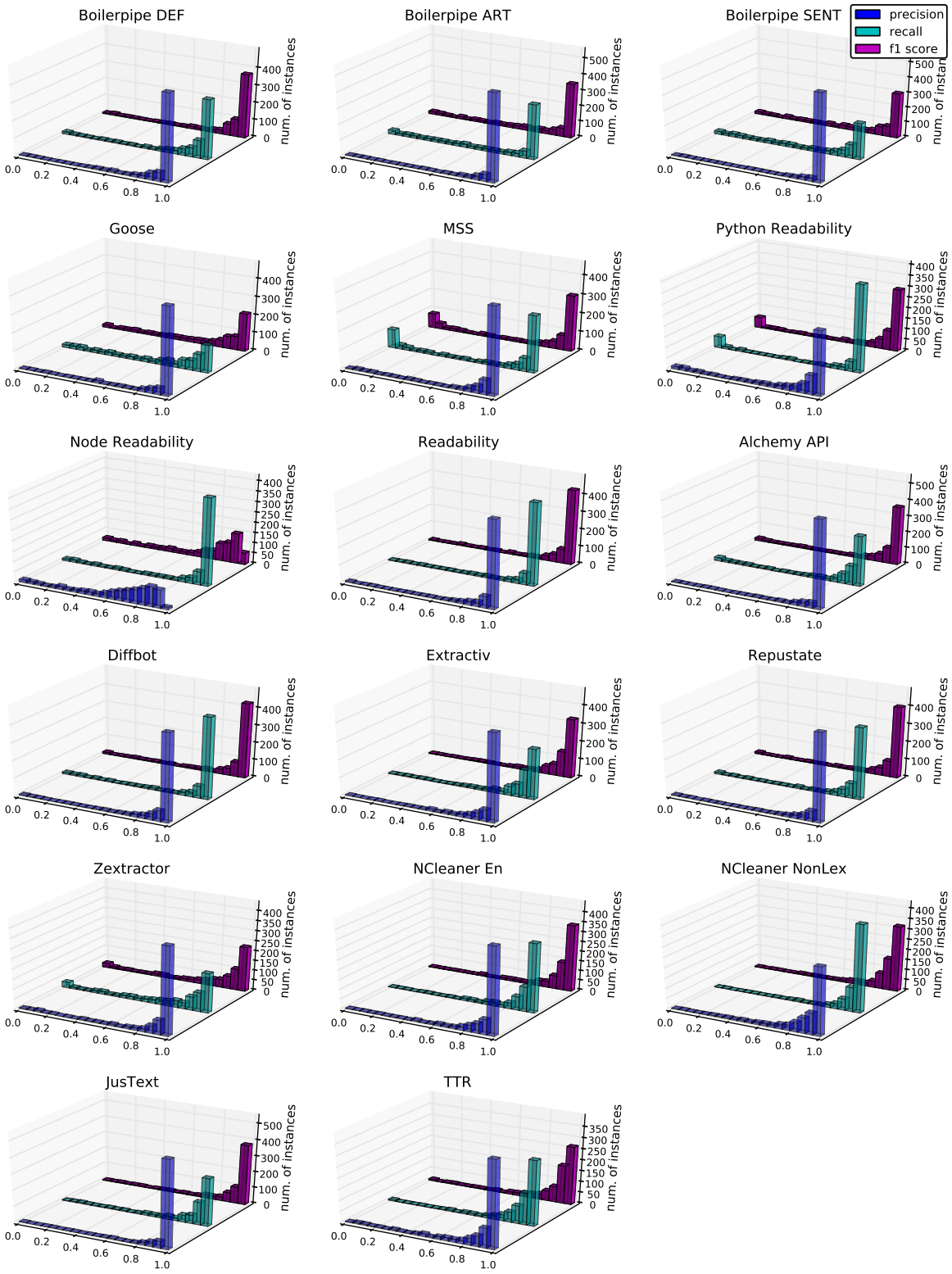


Figure 5.3: Cleaneval per document measurement distributions

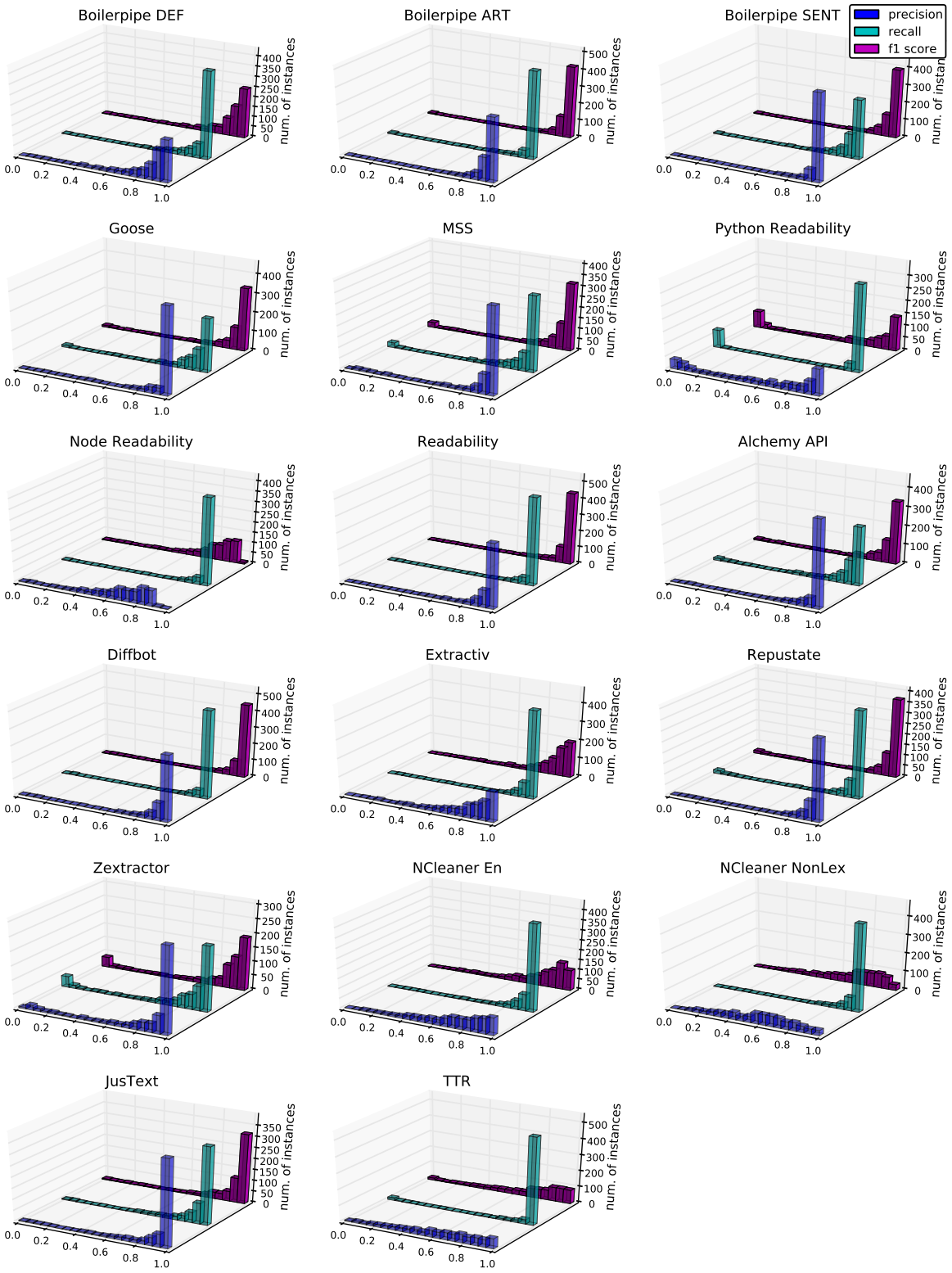


Figure 5.4: L3S-GN1 per document measurement distributions

5.3 Boundary cases

An important part of the evaluation results is also the analysis of boundary cases captured for each algorithm. We defined the boundary cases of per-document measurements in *Section 3.3.4*.

In *Tables 5.4 and 5.5* we provide the boundary case counts for Cleaneval and L3S-GN1 dataset, respectively. Alongside already defined boundary cases, we also provide the counts of an additional boundary case: the number of failed per-document measurements due to implementation specific exceptions of each TE (e.g. parser failures). We also added the number of successful per-document measurements that were used to calculate the the precision, recall and F1-score results in the previous section.

We argue that such an analysis is an important feature of the overall performance of an individual TE since a high number of border cases is usually correlated to high variability of per-document measurements. Without such error analysis, a TE could hypothetically return an empty retrieved set of words for documents that tend to insert a lot of noise into the results of other TE and only retrieve text from documents with easily extractable content. In theory such an extractor could be unfairly favoured by our evaluation environment. Since all erroneous TE exposed by our analysis also experience poor performance, we argue that this was not the case in our evaluation task.

In *Tables 5.4 and 5.5* we first provide a tabular form of boundary case analysis and in *Figures 5.5 and 5.6* we provided a visualization of such data in the form of a stacked bar chart for the Cleaneval and L3S-GN1 dataset, respectively. The left part of each bar chart includes the "successful" portion of per document measurements. Conversely, the right side of the bar chart only contains the portions of erroneous cases.

The visualization of such data exposed some noteworthy outliers of our border case analysis:

- `Node Readability` and `Python Readability` have distinctively high failed cases across both datasets.
- `Goose`, `Repustate`, `JusText` and `Zextractor` consistently accumulate high number of empty extraction results denoted by $|Seq_{ret}| = 0$ across both datasets.

Cleaneval dataset ($n = 681$)						
Text Extractor	$\lvert Seq_{rel}\rvert = 0$	$\lvert Seq_{rel} \cap Seq_{ret}\rvert = 0$	$\lvert Seq_{ret}\rvert = 0$	Mismatch	Failed	Success
`Boilerpipe DEF`	4	2	4	1	0	670
`Boilerpipe ART`	4	2	2	1	0	672
`Boilerpipe SENT`	3	3	11	0	0	664
`Goose`	2	4	57	1	0	617
`MSS`	3	3	4	3	1	667
`Python Readability`	4	2	23	7	52	593
`Node Readability`	5	0	0	0	50	626
`Readability`	4	0	0	0	11	666
`Alchemy API`	4	2	10	1	1	663
`Diffbot`	5	1	3	0	0	672
`Extractiv`	5	1	1	0	3	671
`Repustate`	3	2	18	2	20	636
`Zextractor`	1	5	45	0	2	628
`NCleaner En`	5	1	0	1	0	674
`NCleaner NonLex`	5	1	0	0	0	675
`JusText`	1	5	21	0	0	654
`TTR`	6	0	5	0	0	670

Table 5.4: Cleaneval boundary case analysis. Where n at the top of the table denotes the total number of documents in the dataset.

L3S-GN1 dataset ($n = 621$)						
Text Extractor	$\lvert Seq_{rel}\rvert = 0$	$\lvert Seq_{rel} \cap Seq_{ret}\rvert = 0$	$\lvert Seq_{ret}\rvert = 0$	Mismatch	Failed	Success
`Boilerpipe DEF`	0	0	3	0	0	618
`Boilerpipe ART`	0	0	2	1	0	618
`Boilerpipe SENT`	0	0	5	0	0	616
`Goose`	0	0	36	0	0	585
`MSS`	0	0	6	4	0	611
`Python Readability`	0	0	18	34	104	465
`Node Readability`	0	0	0	0	127	494
`Readability`	0	0	0	0	3	618
`Alchemy API`	0	0	2	0	1	618
`Diffbot`	0	0	0	3	0	618
`Extractiv`	0	0	1	0	1	619
`Repustate`	0	0	24	1	10	586
`Zextractor`	0	0	58	7	1	555
`NCleaner En`	0	0	2	0	0	619
`NCleaner NonLex`	0	0	2	0	0	619
`JusText`	0	0	35	0	0	586
`TTR`	0	0	3	0	0	618

Table 5.5: L3S-GN1 boundary case analysis. Where n at the top of the table denotes the total number of documents in the dataset.

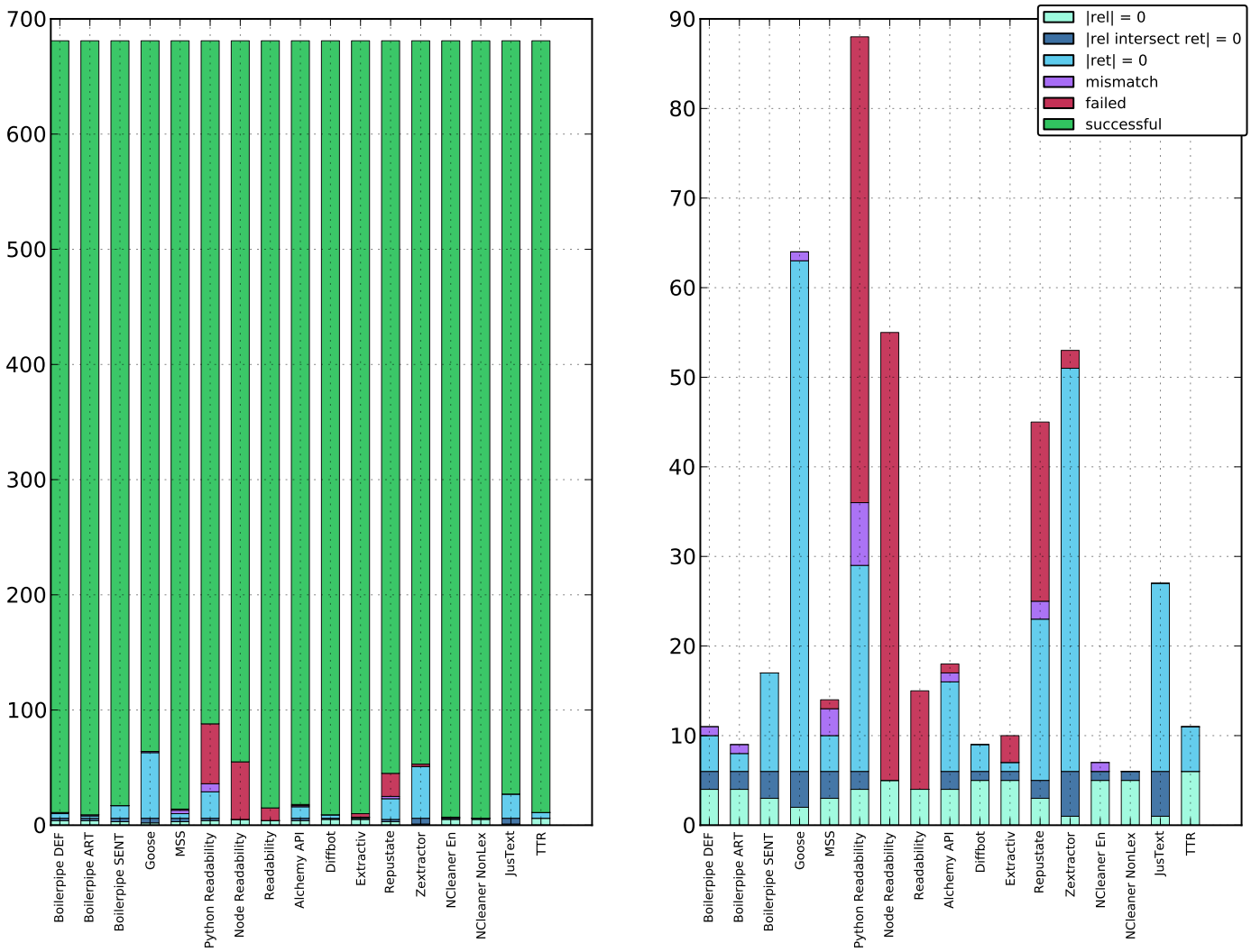


Figure 5.5: Cleaneval boundary case analysis.

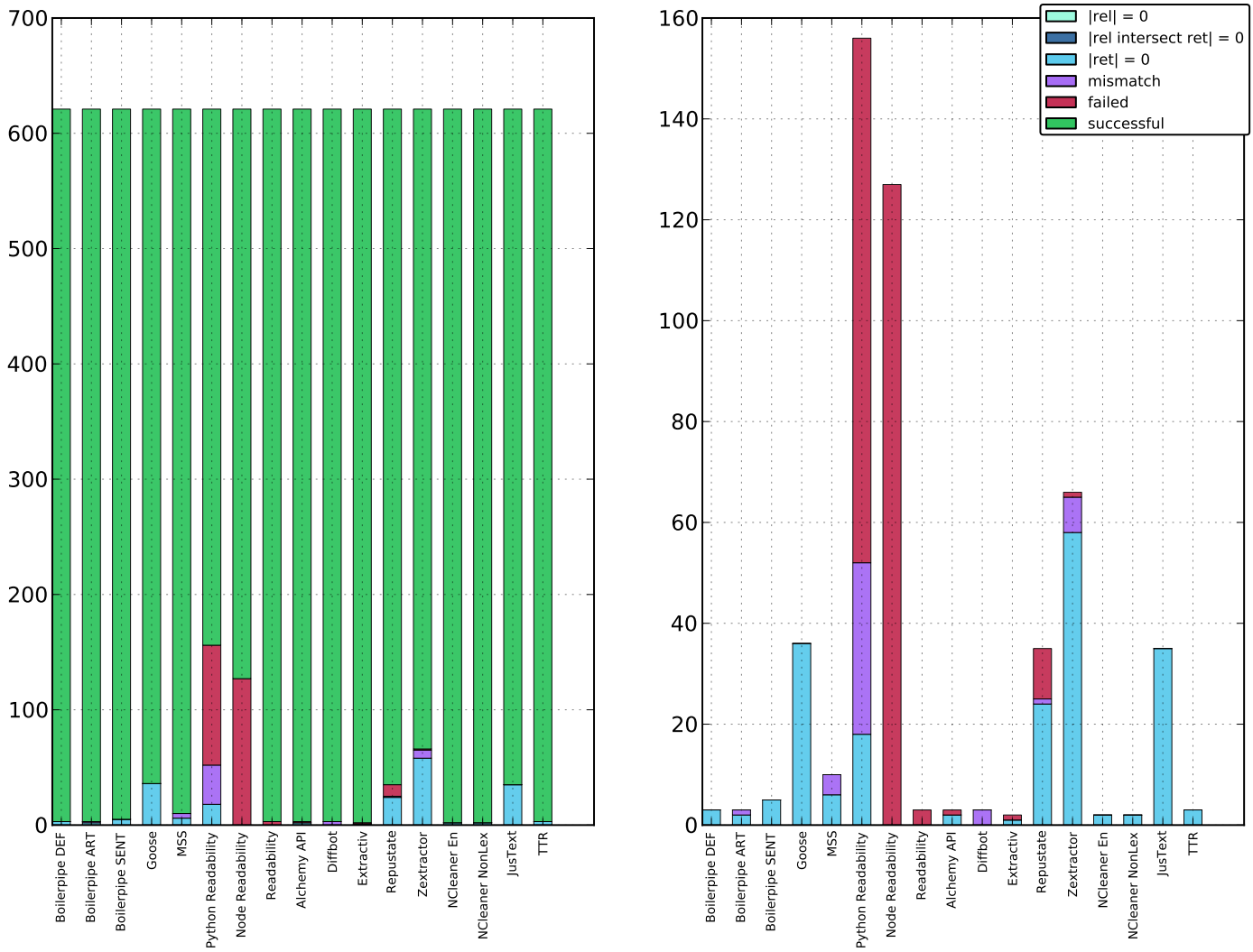


Figure 5.6: L3S-GN1 border case analysis.

Chapter 6

Conclusion

In the first chapter we provided a general definition of content extraction from web data and its importance for the academic and industrial communities. We also defined several sub-categories of this field in general and focused specifically on the category of algorithms capable of extracting the main textual content from arbitrary web pages. We continued with the definition and contextual perspective of terminology used throughout this work.

The second chapter was entirely dedicated to the general overview of the sparse content extraction field. During our examination of the literature we encountered a lack of survey works as well as a lack of related evaluation works. Despite these initial difficulties we managed to review approximately 40 scientific publications related to this field. Our overview chapter roughly categorizes all the reviewed scientific approaches, whether popular, commercial or open-source software into four intersecting categories: wrapper based methods, template detection, web page segmentation methods and finally a more detailed survey of TE approaches, which were the main focus of our evaluation. For TE we also provided an overview of commercial web services and other software with TE capabilities.

The third chapter explored the means for evaluating TE methods, which exposed the sparsity of available resources. We reviewed the features of our evaluation datasets, the Cleaneval dataset which was harvested to benchmark various methods in a competitive scientific task and the L3S-GN1 dataset which was compiled by crawling the Google News stream for a longer period of time. Alongside datasets, we have chosen to evaluate the selected TE methods in terms of precision, recall and F1-score whose definition and computation implementation varied across different related works. We put the selected metrics in the appropriate context of TE and provided formal definitions alongside

special boundary cases that can occur when using a larger set of documents and multiple TE.

The fourth chapter presented the engineering portion of our work that included building a framework for evaluating multiple TE implementations with varying quality on multiple datasets.

In the last chapter we reviewed the final evaluation results of 17 algorithms based on two datasets with more then 600 documents per dataset. First we presented, interpreted and discussed the observed results that included average precision, recall and F1-score over per-document metric measurements, then continued with the presentation and interpretation of the visualization of the same results using firstly bar plots and secondly histograms of the per-document measurement distributions. We concluded the chapter by producing an analysis of boundary cases of per-document measurements that we encountered during our evaluation. Surprisingly we observed high performance for methods that followed intuition and experimentation in contrast to those that relied on complex mathematical models and scientific approaches.

6.1 Further Work

During our study we encountered a particular aspect of evaluation of TE algorithms that arguably carries research value and could be subjected to further research.

Despite the fact that we utilized common metrics and presented them in various forms, we argue that the very same metrics are not informative enough to provide performance oversight over a specific feature of TE algorithms. For example, say we are interested in the ability of cleaning out in-line article noise as in-line article advertisements or picture galleries, or say we are interested in the precise detection of the beginning of the article compared to the rest.

In our the first example case, when using the per-document definition of precision used in this work, small amounts of in-line noise do not penalize precision significantly, since all retrieved words are treated equally. Similarly, in our second example case words in the beginning of the article contribute equally to the metric measurement as words at the end of the article.

In the continuation of this work we plan to explore the possibility of attributing weights to either specific regions or to words of the golden standard in order to provide extra performance measurement capabilities for specific use cases or particular characteristics of TE algorithms. Concretely, if the algorithm would fail to retrieve words with higher weights it would be significantly penalized in the resulting metric, opposed to the failure of extracting words with little importance in respect to the assigned word weight.

List of Figures

List of Tables

List of Algorithms

Bibliography

[1] B. Adelberg, "NoDoSE - A tool for semi-automatically extracting structured and semistructured data from text documents," in *Proc. of the 1998 ACM SIGMOD Int. Conf. on Management of Data*, 1998, pp. 283-294.

[2] S. Baluja, "Browsing on small screens: recasting web-page segmentation into an efficient machine learning framework," in *Proc. of the 15th int. conf. on World Wide Web*, 2006, pp. 33-42.

[3] M. Baroni et al. (2007). *Cleaneval: a competition for cleaning web pages.* [Online]. Available: http://cleaneval.sigwac.org.uk/Cleaneval_LREC_abstract.pdf

[4] M. Baroni et al. (2007). *CLEANEVAL: Guidelines for annotators.* [Online]. Available: http://cleaneval.sigwac.org.uk/annotation_guidelines.html

[5] Z. Bar-Yossef and S. Rajagopalan, "Template detection via data mining and its applications," in *Proc. of the 11th int. conf. on World Wide Web*, 2002, pp. 580-591.

[6] D. Cai et al., "VIPS: a Vision-based Page Segmentation Algorithm," Microsoft Technical Report (MSR-TR-2003-79), 2003.

[7] D. Chakrabarti et al., "A graph-theoretic approach to webpage segmentation," in *Proc. of the 17th int. conf. on World Wide Web*, 2008, pp. 377-386.

[8] D. Chakrabarti et al., "Page-level template detection via isotonic smoothing," in *Proc. of the 16th int. conf on World Wide Web*, 2007, pp. 61-70.

[9] L. Chen et al., "Template detection for large scale search engines," in *Proc. of the 2006 ACM symposium on Applied computing*, 2006, pp. 1094-1098.

[10] Y. Chen et al., "Detecting web page structure for adaptive viewing on small form factor devices," in *Proc. of the 12th int. conf. on World Wide Web*, 2003, pp. 225-233.

[11] V. Crescenzi et al., "RoadRunner: Towards Automatic Data Extraction from Large Web Sites," in *Proc. of the 27th Int. Conf. on Very Large Data Bases*, 2001, pp. 109-118.

[12] S. Debnath et al., "Automatic extraction of informative blocks from webpages," in *SAC '05: Proc. of the 2005 ACM symp. on Applied computing*, 2005, pp. 1722-1726.

[13] S. Debnath et al., "Identifying content blocks from web documents," in *Proc. of the 15th ISMIS 2005 Conf.*, 2005, pp. 285-293.

[14] S. Evert, "A Lightweight and Efficient Tool for Cleaning Web Pages," in *Proc. of the Sixth Int. Conf. on Language Resources and Evaluation*, 2008.

[15] D. Fernandes et al., "A site oriented method for segmenting web pages," in *Proc. of the 34th int. ACM SIGIR conf. on Research and development in Information Retrieval*, 2011, pp. 215-224.

[16] J. Gibson et al., "Adaptive web-page content identification," in *Proc. of the 9th annu. ACM int. workshop on Web information and data management*, 2007, pp. 105-112.

[17] D. Gibson et al., "The Volume and Evolution of Web Page Templates," in *Special interest tracks and posters of the 14th int. conf. on World Wide Web*, 2005, pp. 830-839.

[18] T.Gottron, "Content Code Blurring: A New Approach to Content Extraction," in *Proc. of the 2008 19th Int. Conf. on Database and Expert Systems Application*, 2008, pp. 29-33.

[19] T. Gottron, "Evaluating Content Extraction on HTML Documents," in *Proc. of the 2nd Int. Conf. on Internet Technologies and Applications*, 2007, pp. 123-132.

[20] T. Gottron, "Combining Content Extraction Heuristics: The CombinE System," in *Proc. of the 10th Int. Conf. on Information Integration and Web-based Applications & Services*, 2008, pp. 591-595.

[21] H.Y. Kao et al., "WISDOM: Web Intrapage Informative Structure Mining Based on Document Object Model," *IEEE Trans. on Knowl. and Data Eng.*, vol. 17, pp. 614-627, 2005.

[22] C.A. Knoblock et al., "Wrapper Maintenance: A Machine Learning Approach," *Journal Of Artificial Intelligence Research*, vol. 18, pp. 149-181, 2003.

[23] C. Kohlschütter et al., "Boilerplate detection using shallow text features," in *Proc. of the third ACM int. conf. on Web search and data mining*, 2010, pp. 441-450.

[24] T. Kovacic (2011). *Evaluating Text Extraction Algorithms.* [Online]. Available: http://tomazkovacic.com/blog/122/evaluating-text-extraction-algorithms/

[25] T. Kovacic (2011). *Feature-wise Comparison of HTML Article Text Extractors.* [Online]. Available: http://tomazkovacic.com/blog/98/feature-wise-comparison-of-html-article-text-extractors/

[26] T. Kovacic (2011). *Evaluation Metrics for Text Extraction Algorithms.* [Online]. Available: http://tomazkovacic.com/blog/74/evaluation-metrics-for-text-extraction-algorithms/

[27] T. Kovacic (2011). *List of resources: Article text extraction from HTML documents.* [Online]. Available: http://tomazkovacic.com/blog/56/list-of-resources-article-text-extraction-from-html-documents/

[28] T. Kovacic (2011). *Overview: Extracting article text from HTML documents.* [Online]. Available: http://tomazkovacic.com/blog/14/extracting-article-text-from-html-documents/

[29] N. Kushmerick, "Wrapper induction: efficiency and expressiveness," *Artificial Intelligence*, vol. 118, pp. 15-68, Apr. 2000.

[30] A.H.F. Leander et al., "A Brief Survey of Web Data Extraction Tools," in *SIGMOD Rec.*, Vol. 31, No. 2. , 2002, pp. 84-93.

[31] S. Lin and J. Ho, "Discovering Informative Content Blocks from Web Documents," in *Proc. of ACM SIGKDD'02*, 2002, pp. 588-593.

[32] L. Liu et al., "XWRAP: an XML-enabled wrapper construction system for Web information sources," in *Proc. 16th Int. Conf. on Data Engineering*, 2000, pp. 611-621.

[33] C.D. Manning et al., "Near-duplicates and shingling," in *Introduction to Information Retrieval*, New York, USA, Cambridge University Press, 2008, ch. 19, sec. 6, pp. 400-403.

[34] C.D. Manning et al., "Evaluation of unranked retrieval sets," in *Introduction to Information Retrieval*, New York, USA, Cambridge University Press, 2008, ch. 8, sec. 3, pp. 142-145.

[35] M. Marek et al., "Web page cleaning with conditional random fields," in *Proc. of the Web as Corpus Workshop*, 2007.

[36] I. Muslea et al., "Hierarchical Wrapper Induction for Semistructured Information Sources," *Autonomous Agents and Multi-Agent Systems*, vol. 4, pp. 93-114, Mar. 2001.

[37] J. Pasternack and D. Roth, "Extracting article text from the web with maximum subsequence segmentation," in *Proc. of the 18th int. conf. on World wide web*, 2009, pp. 971-980.

[38] J. Pomikalek, "Removing Boilerplate and Duplicate Content from Web Corpora," Ph.D. dissertation, Faculty of Informatics, Masaryk Univ., Brno, Czech Republic, 2011.

[39] J. W. Ratcliff and D. E. Metzener (1988). *Pattern Matching: The Gestalt Approach*. [Online]. Available: http://drdobbs.com/article/print?articleId=184407970

[40] T. Sager et al., "Detecting similar Java classes using tree algorithms," in *Proc. of the 2006 int. workshop on Mining software repositories*, 2006, pp. 65-71.

[41] Y. Shinyama. (2007, Sep 6). *Webstemmer - How it works?* [Online]. Available: http://www.unixuser.org/ euske/python/webstemmer/howitworks.html

[42] M. Spousta et al., "Victor: the Web-Page Cleaning Tool," in *Proc. of the 4th Web as Corpus Workshop*, 2008, pp. 12-17.

[43] F. Sun et al., "DOM based content extraction via text density," in *Proc. of the 34th int. ACM SIGIR conf. on Research and development in Information Retrieval*, 2011, pp. 245-254.

[44] K. Vieira et al., "A fast and robust method for web page template detection and removal," in *Proc. of the 15th ACM int. conf. on Information and knowledge management*, 2006, pp. 258-267.

[45] T. Weninger and W. H. Hsu, "Text Extraction from the Web via Text-To-Tag Ratio," in *Workshop on Text Info. Ret. at Int. Conf. on Data. and Expert Sys.*, Turin, Italy, 2008.

[46] T. Weninger et al., "CETR - Content Extraction via Tag Ratios," in *Proc. of the 19th Int. Conf. on World Wide Web*, Raleigh, NC, 2010 pp. 971-980.

[47] L. Yi et al., "Eliminating noisy information in Web pages for data mining," in *Proc. of the ninth ACM SIGKDD int. conf. on Knowledge discovery and data mining*, 2003, pp. 296-305.

[48] Y. Zhai and B. Liu, "Web data extraction based on partial tree alignment," in *Proc. of the 14th int. conf. on World Wide Web*, 2005, pp. 76-85.