**Leibniz Universität Hannover**
Fakultät für Elektrotechnik und Informatik
Institut für Verteilte Systeme
Fachgebiet Wissensbasierte Systeme

Bachelor Thesis

# Joint Approach to Boilerplate Detection in Web Archives

Author:
**Rafał Leśniak**

Examiner: **Prof. Dr. Wolfgang Nejdl**
Second examiner: **Prof. Dr. Heribert Vollmer**

Supervisor:
**Helge Holzmann, M.Sc.**

21.01.2017

## Abstract

Web archives are a valuable source of information for researchers and the public. In the process of corpora building researchers need tools and methods for the identification and removal of boilerplate text, which is not related to the main content, from HTML documents. While working with large Web archives the process of content extraction can consume large amounts of time and resources, therefore, we present a method for boilerplate detection, which leverages unique properties of HTML documents stored in Web archives. We show how documents extracted from Web archives can be grouped by template similarity in order to find common HTML elements among them and how those elements can be classified as boilerplate or content based on their occurrence in the group. In this work we define a workflow which aims for automatic HTML document annotation suitable for content extraction, discuss a method of parallelization of this process and evaluate our approach in a software solution. Finally, we show, that our method can be applied for content extraction, where simple bit-pattern based classifier reaches 91.8% recall of content with 98.2% precision for boilerplate.

## Zusammenfassung

Web Archive sind eine wertvolle Informationsquelle für Forscher und die Allgemeinheit. Viele Forscher benötigen passende Werkzeuge und Methoden um Elemente, wie Navigation und Werbung - die Boilerplate-Elemente - aus einem HTML Dokument zu löschen. Die Größe eines Archivs spielt eine grosse Rolle, je grösser ein Archiv ist, desto mehr Mittel werden bei der Extraktion vom Inhalten benötigt. In der vorliegenden Arbeit zeigen wir, wie aus einem Web Archiv extrahierte Dokumente, an Hand von der Ähnlichkeit des Templates gruppiert werden können. Welche Eigenschaften von den Dokumenten, die in den Gruppen vorkommen, benutzt werden können um effizient Boilerplate-Elemente erkennen zu können. Wir definieren einen Workflow für automatisierte HTML Annotation, der für Content-Extraktion eingesetzt werden kann. Des weiteren, stellen wir die Parallelisierung dieser Methode vor. Anschliessend evaluieren wir unsere Methoden und zeigen, dass schon die einfache Bit-Muster basierte Methode 91.8% der Inhalt-Elemente mit 98.2% Genauigkeit bei der Erkennung von Boilerplate-Elementen erkennen kann.

# Contents

# List of Figures

# Listings

# List of Tables

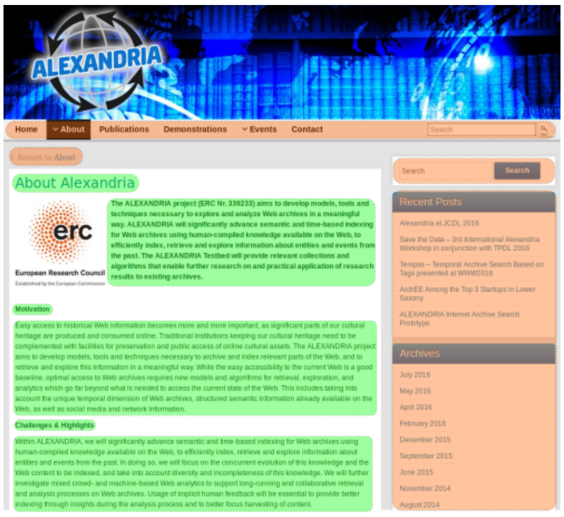
# 1. Introduction

In the Information Age, the number of resources available on World Wide Web (WWW) is growing very fast. News articles, technical documentations, research papers, cooking recipes, tutorials and a large variety of other documents can be found on the Web. Those documents are valuable source of information for users and researches. Web archiving is the process of harvesting and preservation of those information for future researchers, historians and the public. There are various organizations, like the Internet Archive[1], national libraries, national archives and various consortia of organizations, which are involved in archiving important Web content. However, accessing this information still presents a major problem for researchers, thus, new tools and methods need to be developed. One of those problems is the process of finding content in documents. Web archives have few different properties in comparison to Web pages served "live" on the Internet. Web archives usually contain many documents per domain and frequently many versions of a Web site. This allows us to identify changes in, as well, finding common structures shared among a group those documents, which helps us in separation of the content from the rest of the documents.

## 1.1. Motivation

In Information Retrieval researchers are interested in good quality of datasets for their experiments. The task of preparation of such datasets is called *corpora building.* A HTML document is composed of HTML elements and collections of such documents are stored in Web archives. HTML documents contain, besides of the actual content, navigational elements, templates and advertisements, which are typically not related to the main content. Those HTML elements are called *boilerplate.* The presence of boilerplate text, as well, missing parts of the content may have negative impact on experiments. Therefore the identification of the parts of a Web page that hold the content, e.g. the portion of a news page that contains the actual article, is a significant problem that has to be addressed in the process of corpora building [2].

[1]`https://archive.org/`



About Alexandria

The ALEXANDRIA project (ERC Nr. 339233) aims to develop models, tools and techniques necessary to explore and analyze Web archives in a meaningful way. ALEXANDRIA will significantly advance semantic and time-based indexing for Web archives using human-compiled knowledge available on the Web, to efficiently index, retrieve and explore information about entities and events from the past. The ALEXANDRIA Testbed will provide relevant collections and algorithms that enable further research on and practical application of research results to existing archives.

Motivation

Easy access to historical Web information becomes more and more important, as significant parts of our cultural heritage are produced and consumed online. Traditional institutions keeping our cultural heritage need to be complemented with facilities for preservation and public access of online cultural assets. The ALEXANDRIA project aims to develop models, tools and techniques necessary to archive and index relevant parts of the Web, and to retrieve and explore this information in a meaningful way. While the easy accessibility to the current Web is a good baseline, optimal access to Web archives requires new models and algorithms for retrieval, exploration, and analytics which go far beyond what is needed to access the current state of the Web. This includes taking into account the unique temporal dimension of Web archives, structured semantic information already available on the Web, as well as social media and network information.

Challenges & Highlights

Within ALEXANDRIA, we will significantly advance semantic and time-based indexing for Web archives using human-compiled knowledge available on the Web, to efficiently index, retrieve and explore information about entities and events from the past. In doing so, we will focus on the concurrent evolution of this knowledge and the Web content to be indexed, and take into account diversity and incompleteness of this knowledge. We will further investigate mixed crowd- and machine-based Web analytics to support long-running and collaborative retrieval and analysis processes on Web archives. Usage of implicit human feedback will be essential to provide better indexing through insights during the analysis process and to better focus harvesting of content.

Recent Posts

Alexandria at JCDL 2016

Save the Data – 3rd International Alexandria Workshop in conjunction with TPDL 2016

Tempas – Temporal Archive Search Based on Tags presented at WWW2016

ArchEE Among the Top 3 Startups in Lower Saxony

ALEXANDRIA Internet Archive Search Prototype

Archives

July 2016

May 2016

April 2016

February 2016

December 2015

September 2015

June 2015

November 2014

August 2014

Figure 1.1.: Excerpt of the Alexandria Project's Web site showing content (green) and boilerplate (orange).

The Figure 1.1 shows an excerpt of a Web page of the Alexandria Project[2] with elements marked as content and boilerplate. Extracting content for analysis from HTML documents usually can be split up into small number of steps, this includes:

- identification of HTML elements which hold the content
- writing a tool for extraction
- post-processing and storage of the extracted content

Processing HTML documents from one specific domain can be very easy if all or at least large fraction of documents share the *same template*. A template is pre-prepared HTML document that is used as a basis for composing new Web pages [3]. A Formal definition of a template will be discussed in Related Work (see Section 3.2).

Listing 1.1 shows a very simple template with place holders: `${adv1}`, `${adv2}`, `${year}`, `${content}` and `${copyright}`. Those variables will be replaced with data in the generation process of a HTML document. Likely "`${copyright} - ${year}`" and the advertisement will be shared, and the contents of the `${content}` be unique, among pages generated from this template. Thus `${content}` is more interesting than the rest of the place holders. While in the past many Web sites have been manually generated, today's Web sites are more likely generated by Content Management System (CMS) or other Web applications that use some form of templating.

Researchers analyzing HTML documents generated from templates usually do not have access to the templates itself, therefore they are forced to use other methods in order to separate content and boilerplate elements.

[2]`http://alexandria-project.eu/about/about-alexandria-project/`, visited: 2017-01-13

```html
<html><head></head><body>
  <span class="def" style="color:␣blue">
    <ul class="adv"><li>${adv1}</li><li>${adv2}</li></ul>
    ${content}
    ${year} - ${copyright}
  </span>
</body></html>
```

Listing 1.1: template.html - Example of a template HTML document

## 1.2. Content Extraction

The filtering process of finding the main content is called Content Extraction (CE) [4]. This process involves taking samples of documents, reviewing them manually and creating handcrafted rules for CE. Many attempts have been made in the past to automate this process (cf. Related Work 3). Methods like Body-Text-Extraction (BTE), Boilerpipe[3] or jusText[4] exploit the *table layout*, the way Web pages were generated in the past, use heuristics in order to classify elements or concentrated on the contents of *<p>*/*<span>* HTML elements, making assumptions on what the Web pages look like.

However, Web Technology is fast changing. With the evolution of the technology and introduction of HTML5 [5], which allows usage of custom HTML elements [6], making such assumptions on the Web page layout is no longer sufficient, hence new approaches are needed.

Before we can extract content from a HTML document we first need to differentiate between HTML elements that are part of boilerplate and those of the content. One way to do such *classification* would be to inspect the HTML document and annotate those elements by hand.

Consider the following, extremely simplified example: assume we are interested in content of the document *doc1.html* presented in Listing 1.2 and, while working with Web archives, the chances are very high, that we also have a similar document *doc2.html* shown in 1.3. Both documents have been generated from the template presented in Listing 1.1:

```html
<html><head></head><body>
  <span class="def" style="color:␣blue">
    <ul class="adv"><li>Buy More</li><li>New Cars</li></ul>
    Definition of Boilerplate
    2016 - Example Inc. All rights reserved
  </span>
</body></html>
```

Listing 1.2: doc1.html - HTML document we are interested in

```html
<html><head></head><body>
  <span class="def" style="color:␣blue">
    <ul class="adv"><li>Buy More</li><li>New Cars</li></ul>
    <i>Content description</i>
    2016 - Example Inc. All rights reserved
    </span>
</body></html>
```

Listing 1.3: doc2.html - Similar HTML document

[3] `https://github.com/kohlschutter/boilerpipe`, visited: 2017-01-04
[4] `https://github.com/miso-belica/jusText`, visited: 2017-01-04

After short inspection we recognize "*Definition of Boilerplate*" as potential content, the list "*Buy More, New Cars*" and "*2016 - Example Inc. All rights reserved*" as boilerplate since those elements are included in both documents. We notice that the HTML documents share the same template, so it is very easy for us to differentiate those two documents and quickly distinguish between content and boilerplate. For further analysis we write our observation into *doc1.html* using additional HTML elements to markup the elements according to the classification as shown in Listing1.4.

We also make an interesting observation that it is not enough to annotate the whole *<span>* element but, in this case, it is in our interest to mark parts of the text block as boilerplate as we did with "*2016 - Example Inc. All rights reserved*" and the advertisement list. Following this observation we may annotate the document as follows:

```
<html><head></head><body>
  <span class="def" style="color:␣blue">
    <boilerplate>
      <ul class="adv"><li>Buy More</li><li>New Cars</li></ul>
    </boilerplate>
    <content>
      Definition of Boilerplate
    </content>
    <boilerplate>
      2016 - Example Inc. All rights reserved
    </boilerplate>
  </span>
</body></html>
```

Listing 1.4: result.html - Annotated version of doc1.html

Listing 1.4 shows two additional HTML elements that have been inserted to mark the relevant parts of text. In this case *<content>* and *<boilerplate>* elements are used for annotation. From this point onwards the CE is a rather simple task.

For small Web archives it is sufficient to examine a few documents, identify the HTML elements with content and proceed with extraction. However, creation of a bigger corpus, from larger Web archives, can take huge amounts of time and resources to process. The size of a Web archive can vary from few kilobytes[5] to several gigabytes[6] and the number of included documents between few hundreds to several millions. While the analysis of few hundreds of documents can be done by one individual or with crowdsourcing platforms, analysis of millions of documents will consume serious amounts of time and resources. The time needed to process big Web archives sometimes extends the lifespan of a project the researchers are working on.

[5]kilobyte (KB)
[6]gigabyte (GB)

### Difference Between Content and Boilerplate

The concept that *information is the message* has different meanings in different contexts [7, 8]. We can assume that HTML documents, when presented on the Internet are a form of communication between the author (sender of the message) and reader (receiver of the message). A user looking for an answer to any given question, is reading a HTML document, and thus receiving a message. This user will probably concentrate on those parts of the message that contain the answer to the question he had.

While working with Web archives, we assume that the answer to any given question is contained in the documents within a Web archive. The challenging problem is that we do not know where. Therefore, following our observations from the previous section, we can provide simplified definitions of content and boilerplate.

In this work we consider HTML elements shared among a set of HTML documents as *boilerplate* and elements that differ in this set are considered as *content*. Consequently, stripping *boilerplate* elements from a HTML document will still satisfy the objective of the retrieval of “the kind of knowledge that helps the receiver to satisfy his information needs” [7, 9].

## 1.3. Problem Definition

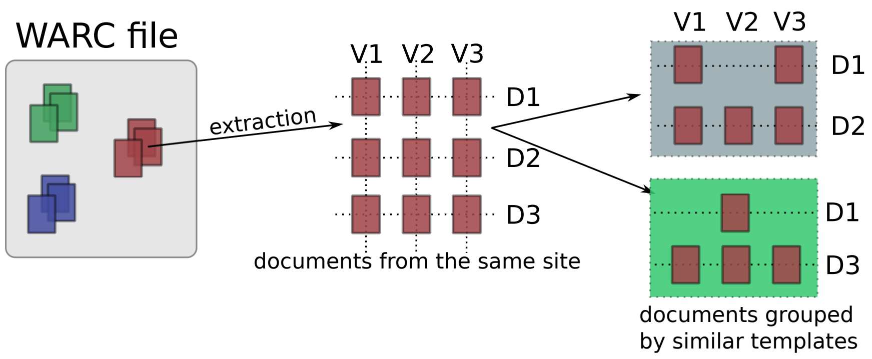


Figure 1.2.: Simplified view of the extraction process.

We are given a Web archive (WARC file) with millions of documents for multiple Web sites[7] (brown, green, blue). For each site we have multiple documents (D1, D2, D3) and their different snapshots (V1, V2, V3) as showed in Figure 1.2. Web pages in the same site share boilerplate elements (see Figure 1.3). Instead of trying to detect boilerplate elements ($A$,$B$) by analyzing the structure of a single HTML document, we compare documents with a similar template (*doc1.html* and *doc2.html*) in order to detect common elements in a joint fashion.

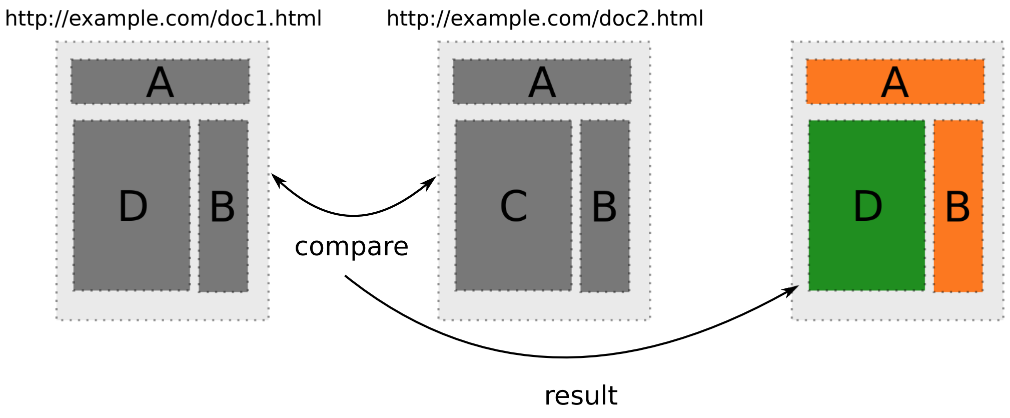


Figure 1.3.: Simplified view of the boilerplate detection process.

We also conduct a qualitative study of our approach and prepare experimental software which demonstrates our method.

[7]crawled from one domain, e.g. `https://www.heise.de`

# 2. Methodology

In this work we aim for defining steps and methods suitable for automated boilerplate detection in sets of HTML documents stored in Web archives. This method should be independent from used programming language, ruleless (without usage of predefined rules for detection of boilerplate in different templates) and suitable for parallelization. It should work with HTML documents stored in Web ARChive (WARC) files which means that only static[1] content is available. In this chapter we describe our Diff Based Content Extraction (DBCE) method in detail.

## 2.1. Process Chain

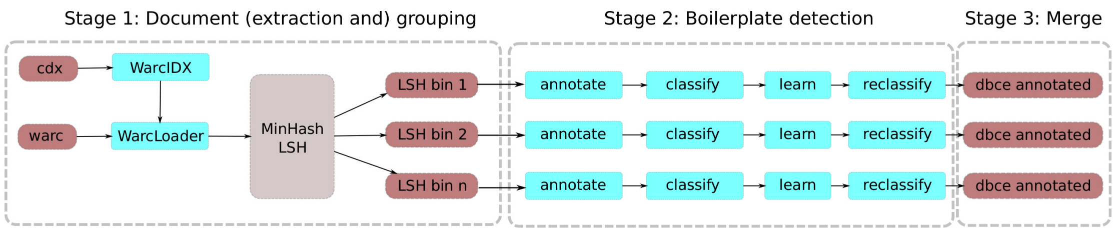


Figure 2.1.: DBCE framework overview.

Figure 2.1 shows the main three stages that a HTML document will pass on its way from a Web archive to the final annotated version. We start with a WARC file and a corresponding index file (CDX). To keep this description as simple as possible, we work with WARC files containing HTML documents for one domain only, but it is possible to have WARC files with mixed domains.

- *Stage 1* is responsible for loading and extraction of the data from WARC files. In this stage the documents will be grouped by their template similarity
- In *Stage 2* we analyze the HTML documents of each group in order to find shared HTML elements. Those elements will be then classified according to simple patterns
- In *Stage 3* the annotated HTML documents will be postprocessed, merged and prepared for storage

The Figure 2.2 shows the simplified view of the DBCE framework. The three stages form a pipeline. For the sake of this explanation we focus on one single CPU and describe the differences to parallel processing in Section 2.4.

[1] JavaScript and any additional content will not be fetched from the Web

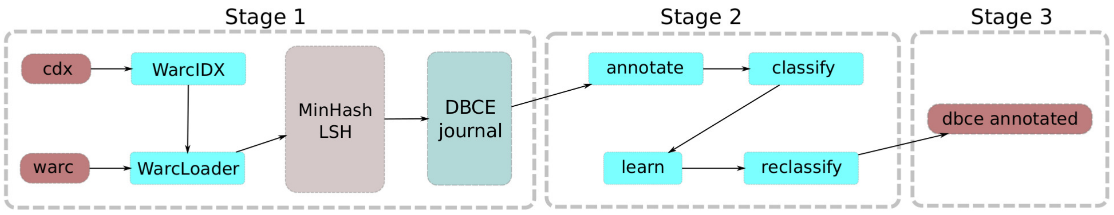


Figure 2.2.: Simplified view of the DBCE Framework.

## 2.2. Stage 1: Document Extraction and Grouping

### Working with Web Archives

A WARC file is a container file format for concatenation of multiple resources (data objects), stored in one file. Each resource is consisting of a set of simple text headers and an arbitrary block of data [10]. For faster random access of the contents of a WARC file an Index (CDX) file may be available. The CDX file may contain such information as: URL of the resource, crawl timestamp, offset in the WARC file, type of the resource, length of the entry, unique id of the entry (UUID[2]). With those information we refer to an HTML document from now on as *WARC entry* which is unambiguously identified by the tuple *(domain, path, timestamp, uuid)*. A WARC file may contain multiple snapshots of a single HTML document. Such snapshots will be stored with different *timestamp* and *uuid* thus representing different WARC entries.

The goal is to group documents based on their template similarity and to achieve this we use Min-wise independent permutations (MinHash) and locality sensitive hashing (LSH), which we describe in detail in the next Section 2.2.

Once the WARC entries have been extracted from the WARC file we sanitize the HTML documents. This includes removing JavaScript, comments, iframes. Sanitization is necessary part of the CE, it reduces potential sources of errors while parsing the HTML documents. After the sanitization step we compute MinHash and index the WARC entry using LSH for further querying. LSH indexing is done for whole document, thus the MinHash signature must be computed prior to indexing. Processed WARC entries are stored in a data structure *DBCEJournal*, which allows us to query the LSH indexes and collect information during the analysis.

### MinHash

The basic idea used in our approach is to group HTML documents based on their template similarity. This process can be described as the process of finding near duplicates in document sets. Finding duplicates of any data can be achieved for example by using cryptographic signatures on the data. Cryptographic signatures use hash functions to compute a signature over given data such that, same data yields the same signature.

[2]Universally Unique Identifier

This is sufficient for finding exact duplicates but in our case, we are interested in finding documents that are *roughly the same* thus we need another methods to solve this problem.

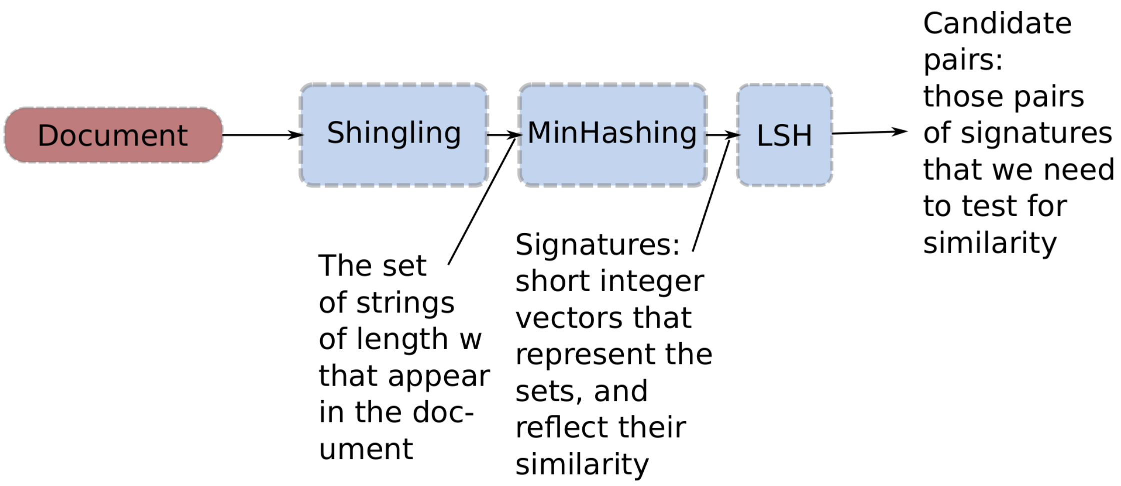

Document
Shingling
MinHashing
LSH
Candidate pairs: those pairs of signatures that we need to test for similarity
The set of strings of length w that appear in the document
Signatures: short integer vectors that represent the sets, and reflect their similarity

Figure 2.3.: MinHashing process. The big picture. [1]

The standard distances defined on strings (like Hamming or Levenshtein) can not represent the concept of *roughly the same*, furthermore the computation of these distances requires pairwise comparison of whole documents. For a large collection of documents this is not feasible and a sampling mechanism per document is necessary [11]. Andrei Z. Broder proposed two mathematical concepts *resemblance* and *containment.*

Given two documents *A,B* the resemblance $r(A, B)$ is a number between 0 and 1, such that when the resemblance is close to 1 it is likely that the documents are roughly the same. The containment $c(A, B)$ of A in B is a number between 0 and 1 that, when close to 1, indicates that A is roughly contained within B. The resemblance has the additional property that

$$d(A, B) = 1 - r(A, B) \tag{2.1}$$

is a metric, which is useful for the design of algorithms intended to cluster a collection of documents into sets of closely resembling documents [11]. The resemblance and containment are expressed as set intersection problems. Each document $D$ can be represented as a set of tokens (those can be words, characters or something else) and has associated set of subsequeces $S(D, w)$. A contiguous subsequece contained in $D$ is called a *shingle.* Given a document $D$ we can associate to it its *w-shingling* defined as the bag of all shingles of size $w$ contained in $D$. So for instance the 4-shingling of *(a,rose,is,a,rose,is,a,rose)* is the bag {*(a,rose,is,a), (rose,is,a,rose), (is,a,rose,is), (a,rose,is,a), (rose,is,a,rose)*}.

To compute the resemblance of two documents it suffices to keep for each document a "sketch" of three to eight hundred bytes consisting of a collection of fingerprints of shingles. Fingerprints are short tags for larger objects. A fingerprint can be computed by a hashing function which uniquely identify substantial blocks of data [12]. Now we can represent a document $D$ as a set of numbers $S_D$ that is the result of fingerprinted shingles[13].

For given two documents $A, B$ the *resemblance* is defined as

$$r(A,B) = \frac{|S_A \cap S_B|}{|S_A \cup S_B|} = sim(A,B) \tag{2.2}$$

which is also the Jaccard similarity[1]. Now we are able to find near-duplicate documents based on the Jaccard similarity. However, suppose we need to find near-duplicates among $N = 1$ million document. The logical step would be to compute pairwise Jaccard similarities for every pair of the documents. This process would take about 5 days at $10^6$ comparisons/sec. For $N = 10$ million it would take more than a year [1].

The process of MinHashing is a way for converting large sets to short signatures while preserving the similarity. The goal is to find a hash function $h(\cdot)$ such that if $sim(S_A, S_B)$ is high, then with high probability $h(S_A) = h(S_B)$ and if $sim(S_A, S_B)$ is low, then with high probability $h(S_A) \neq h(S_B)$. Such suitable hash function for the Jaccard similarity is called Min-Hashing.

First we need another representation of $S_D$ for all documents. We create a boolean matrix which columns ($C$) are sets (documents) and rows ($R$) are elements (shingles). Each cell $x_{e,s}$ becomes 1 if and only if $e$ is a member of $s$. The signature of a column can be computed by a hash function $h(C)$. This would be one signature for all rows in one column. Now we can define a fixed number $p$ of random permutation $\pi$ of the rows. Compute the MinHash $h_\pi(C)$ which is the index of the first (in the permuted order $\pi$) row in which column $C$ has value 1.

$$h_\pi = min_\pi \pi(C) \tag{2.3}$$

The MinHash signature $sig(C)$ is also a list of $p$ indexes of first rows with 1 in column $C$. The MinHash property is defined as

$$Pr[h_\pi(C_A) = h_\pi(C_B)] = sim(C_A, C_B) \tag{2.4}$$

which means that the similarity of two signatures is the fraction of the hash functions in which they agree. We can now store the MinHash signatures as matrices. Columns are the documents and the rows are the $p$ signatures. We have now "compressed" long bit vectors into short signatures.

### Locality-sensitive Hashing for MinHash

The problem with pairwise comparison still persists. To solve this problem we need to focus on pairs of signatures that are likely to be from similar documents. The basic idea of Locality-sensitive Hashing (LSH) is to find documents with Jaccard similarity of at least $s$, for some similarity threshold, e.g., $s >= 0.8$. To achieve this we hash columns of the signature matrix M (cf. Fig. 2.4) several times to many buckets. Each pair of documents that hashes into the same bucket is a candidate pair.

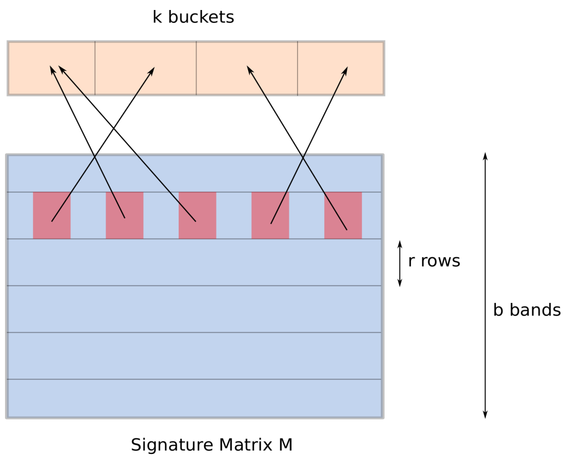


Figure 2.4.: Hashing into k buckets.[1]

We divide the matrix $M$ into $b$ bands of $r$ rows. For each band, hash its portion of each column to a hash table with $k$ buckets. From now on we need to compute the Jaccard similarity only for documents which hashed into the same bucket. Due to the nature of this method there are some trade-offs. Depending on the value of $r$ and $b$ the number of false positives and false negatives would vary [1, 14].

### Similarity of HTML Documents

In further steps we are interested in the analysis of *similar* HTML documents. As we described before the definition of similar HTML documents in our approach is based on the similarity of the template.

Extracting relevant information about the template includes removing all text elements and stripping all attributes from all HTML tags. Consider our example HTML documents *doc1.html* and *doc2.html*, after the preparation process those document would look like this:

```
<html><head></head><body>
  <span>
    <ul><li></li><li></li></ul>
  </span>
</body></html>
```

Listing 2.1: doc1.html - processed for LSH indexing

```
<html><head></head><body>
  <span>
    <ul><li></li><li></li></ul>
    <i></i>
  </span>
</body></html>
```

Listing 2.2: doc2.html - processed for LSH indexing

### Layer Representation of a DOM Tree

A HTML document can be represented as Document Object Model (DOM) tree [15, 16, 17]. For better understanding what actually happens while we process the HTML

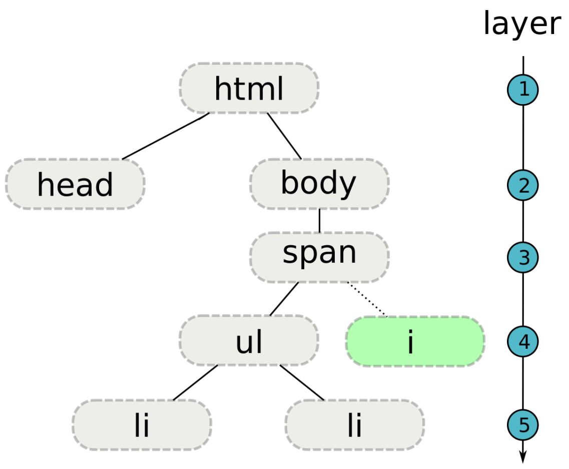


Figure 2.5.: DOM tree representation of our doc1.html (gray) and doc2.html (gray and green).

documents we have adapted *DepthFirstSearch* algorithm to convert DOM trees into a nested list of DOM elements. Such representation is necessary to reduce the complexity of our method and define natural boundaries of DOM element's neighborhood.

The DOM tree presented in Figure 2.5 can be represented as a following list:

```
1      [ [ [ html ] ],
2        [ [ head, body ] ],
3        [ [ span ] ],
4        [ [ ul ] ],
5        [ [ li, li ] ] ]
```

Listing 2.3: DOM Tree of the doc1.html stored as a nested list

```
1      [ [ [ html ] ],
2        [ [ head, body ] ],
3        [ [ span ] ],
4        [ [ ul ], [ li ] ],
5        [ [ li, li ] ] ]
```

Listing 2.4: DOM Tree of the doc2.html stored as a nested list

The line numbers of the Listing 2.3 corresponds to the layer number showed in Figure 2.5. In first layer there is only <*html*>, in second <*head*> and <*body*> and so on. Listing 2.4 shows nested list representation of the *doc2.html*. The only difference is in line 4, where the <*span*> element has two children: an unsorted list <*ul*> and <*i*> element.

General representation of nested lists is shown in Listing 2.5.

```
    layers = [
      [ [ nodes in layer 1] ], // root
      [
        [ nodes in layer 2 parent 1 ],
        [ nodes in layer 2 parent 2 ]
      ], [
        [ nodes in layer n parent 1 ],
        ...,
        [ nodes in layer n parent y ]
      ]
    ]
```

Listing 2.5: Example of a DOM Tree stored as a nested list

Each sublist holds DOM elements/nodes that are in the same layer and share the same parent. The algorithm, which translates a DOM tree into a nested list, is based on *DepthFirstSearch* as presented in Listing 2.6.

```python
def get_layers(tree):
    # Translate DOM tree into nested lists
    level = 0
    level_nodes = OrderedDict()
    dfs(tree, level, level_nodes)
    return level_nodes

def dfs(tree, level, level_nodes):
    # Skip if node is a string
    if is_string(tree):
        return

    _lvl_nodes = []
    for node in tree.children:
        # Traverse the tree to the bottom
        dfs(node, level + 1, level_nodes)
        # Check if node is a HTML tag
        if is_tag(node):
            node.dbce_layer = level
            _lvl_nodes.insert(0, node)
    _ = level_nodes.setdefault(level, [])
    _.insert(0, _lvl_nodes)
    return
```

Listing 2.6: Python implementation of DepthFirstSearch algorithm used for converting DOM Tree to nested lists

The *get_layers(tree)* function's input is a DOM tree which represents a parsed HTML document. The function initializes the nested list and performs recursive *DepthFirst-Search* using the *dfs* function which filters out all text nodes and creates list of all children sharing the same parent.

### Token Generator for MinHash

MinHash (or the min-wise independent permutations) is a probabilistic data structure for computing the Jaccard similarity between datasets. We compute MinHash over sanitized and striped HTML documents. The idea behind this scheme is that two similar templates should have similar signatures. Our approach is to generate long unique tokens that describe the template. One way would be to use XPath[3] or Cascading Style Sheets (CSS)[4] selectors for each DOM tree node, however in our tests we have achieved better results generating tokens like this *'0_0_2_ul/0_0_2_body/0_0_2_html/0_0_2_[document]'*. Next listing shows the actual implementation.

```
for level in sorted(layers.keys()):
 node_group_nr = 0
 for nodes in layers[level]:
  node_nr = 0
  for node in nodes:
   p_path = []
   for _ in node.parentGenerator():
    p_path.append("{}_{}_{}_{}".format(node_nr,
            node_group_nr,
            level,
            _.name))

   p_path.insert(0, "{}_{}_{}_{}".format(node_nr,
            node_group_nr,
            level,
            node.name))

   entry['minhash'].update('/'.join(p_path).encode('utf8'))
   node_nr += 1

  mhf_cnt += 1
  node_group_nr += 1

logger.info('MinHash␣computed␣based␣on␣{}␣features'.format(mhf_cnt))
self.compute_lsh(entry)
```

Listing 2.7: MinHash token generator (dbce/DBCEJournal.py)

The algorithm presented in Listing 2.7 traverses the DOM Tree layer by layer, node by node and generates tokens which are used to feed the MinHash algorithm. After the MinHash has been computed it is used for indexing in LSH.

At the end of this stage we have already extracted, sanitized all WARC entries and indexed them in LSH. The DBCEJournal holds all of those relevant information for one single domain. We can now iterate over all entries in DBCEJournal and proceed with the next stage (see Section 2.3).

[3]https://www.w3.org/TR/xpath/
[4]https://www.w3.org/Style/CSS/

## 2.3. Stage 2: Boilerplate Classification

In this stage of the process we are working with the DBCEJournal and one selected WARC entry (*wEnCur*). For given *wEnCur* we can query the LSH for similar HTML documents and store the result in a list (*SimDocList*) which is sorted descending by similarity. This list will be used to build new data container for further steps.

### Cross Construction

Having one or more similar documents stored in *SimDocList* we can arrange them in special data structure we call *cross*. First we split the *SimDocList* into two disjoint sets *SimDocListVertical* and *SimDocListHorizontal* such that *SimDocListVertical* includes WARC entries that do **not** share the same URL-path and *SimDocListHorizontal* includes WARC entries with **the same** URL-path building (if sorted by crawl timestamp) a timeseries.

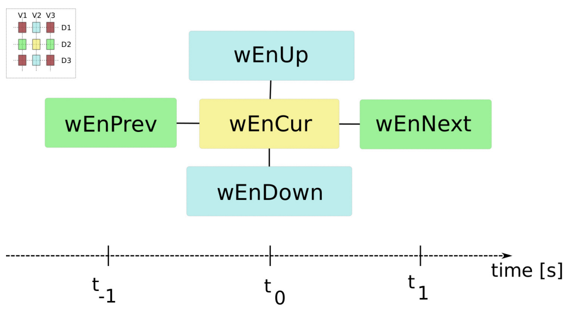


Figure 2.6.: Cross data structure showing the timeline axis.

From now on we build the *cross* as follows: we place in the center *wEnCur*, above the first entry in *SimDocListVertical* which has the highest similarity with *wEnCur* and call it *wEnUp*. The second entry from *SimDocListVertical* (if present) will be placed below *wEnCur*, and we call it *wEnDown*.

The left and right side will be filled with different snapshots (if present) of the same URL-path for *wEnCur*. On the left we place the previous *wEnPrev* and on the right the next *wEnNext* versions of *wEnCur*. We select such *wEnNext* and *wEnPrev* entries from *SimDocListHorizontal*, that the time difference between the snapshots is minimal.

From now on we will work only with the *cross* since it includes up to five WARC entries, arranged in such way that we can easily discern which WARC entry is which. Those characteristic will be used in annotation process. It is also important to notice that we need at least *wEnCur* and *wEnUp* in order to differentiate between those documents. Comparison only between *wEnCur* and *wEnPrev* or *wEnNext* would not yield any useful results, therefore, this requirement must be satisfied, otherwise we terminate here.

### Annotation Process

Having similar HTML documents we can proceed with annotation. In this step we use slightly modified version of *lxml.html.diff.html_annotate*[5] algorithm, which is a XML toolkit for Python[6]. The annotation algorithm works on a group of documents. The original version of this algorithm can be used to annotate HTML documents, where only the first occurrence of a HTML element is stored. We extended the annotation function, such that it allows us to store all occurrences. The basic strategy is to split the documents into logical tokens, then a diff of each of the versions is made to track in which document a token appears. This implementation treats HTML documents as a sequence of tokens similarly to *HtmlDiff* introduced by Ball et al. [18].

Our annotation process expects, as an input, a list of tuples *(string: body of the html, string: annotation)*. The tuples must be set in this order *(wEnCur, wEnUp, wEnDown, wEnPrev, wEnNext)*. Elements that can not be provided at given position are left empty.

```
to_be_annotated = [
  (wEnCur.html.body , 'cur'),
  (wEnUp.html.body , 'up'),
  (wEnDown.html.body, 'down'),
  (wEnPrev.html.body, 'prev'),
  (wEnNext.html.body, 'next')
]
```

Listing 2.8: Input of the annotation process

The annotation function compares *wEnCur* with each of the other entries in the given order:

```
def html_annotate(to_be_annotated):
  # wEnCur with wEnUp
  diff(to_be_annotated[0][0], to_be_annotated[1][0], to_be_annotated[1][1])
  # wEnCur with wEnDown
  diff(to_be_annotated[0][0], to_be_annotated[2][0], to_be_annotated[2][1])
  # wEnCur with wEnPrev
  diff(to_be_annotated[0][0], to_be_annotated[3][0], to_be_annotated[3][1])
  # wEnCur with wEnNext
  diff(to_be_annotated[0][0], to_be_annotated[4][0], to_be_annotated[4][1])
```

Listing 2.9: Input of the annotation process

and in each diffing step, the occurrence information of the HTML elements will be attached to *wEnCur*. Using this method only HTML elements that were in *wEnCur* are being annotated.

Assuming that we give the annotation algorithm input prepared for our examples 1.2 and 1.3, the output will look like this:

[5] `https://github.com/lxml/lxml/blob/ef52eb5cff33d61b8a4195bfadb01b2fb034daf9/src/lxml/html/diff.py#L31`, visited: 2016-12-21

[6] `https://github.com/lxml/lxml/`, visited: 2016-12-21

```
 1 <html><head></head><body>
 2   <ul class="adv">
 3     <li>
 4       <dbce class="dbce-cur dbce-up">Buy More</dbce>
 5     </li><li>
 6       <dbce class="dbce-cur dbce-up">New Cars</dbce>
 7     </li>
 8   </ul>
 9   <span class='def' style='color: blue'>
10     <dbce class="dbce-cur">
11       Definition of Boilerplate
12     </dbce>
13     <dbce class="dbce-cur dbce-up">
14       2016 - Example Inc. All rights reserved
15     </dbce>
16   </span>
17 </body></html>
```

Listing 2.10: ann-result.html - Intermediate result after first annotation

As we mentioned in the introduction, we do not annotate the original HTML elements, but the text elements. We do this by inserting special *<dbce>* tags, where the annotation is stored as a class attribute. By using this method we do not need to hold any shared states in memory and the document can be simply forwarded to other instances in the pipeline. Now we have a HTML document with additional, annotated DOM elements. The next step is to classify the *<dbce>* tags.

### Cross Encoding

The *cross* can be encoded as a bit array. The order of bits in this array corresponds to the order of the tuples we feed into the annotation algorithm. Every WARC entry that is present in the cross, is represented as a *1* and each missing one as *0*. Following the scheme, when working with full *cross*, the encoding will result in *(1,1,1,1,1)* array and for the example 2.10 with *(1,1,0,0,0)* . This simple scheme can be translated into English language as follows: “this HTML document is annotated version of *wEnCur* and *wEnUp*”.

As for the *<dbce>* elements, we do the same encoding, which allows us to determine the occurrence of a DOM element in each version of a HTML document. Joining those two bit arrays we create a tuple of tuples as shown in Figure 2.7

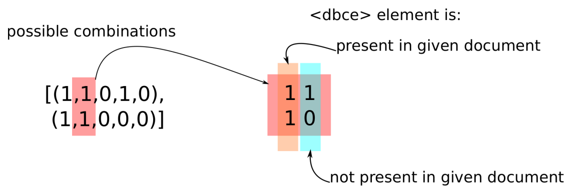


Figure 2.7.: Example of the encoding scheme.

Following this scheme the encoding of *<dbce>* elements would look like this *[(1,1,0,0,0), (1,1,0,0,0)]* for the *<dbce>* in line 6 and *[(1,1,0,0,0), (1,0,0,0,0)]* the one in line 10 of the Listing 2.10.

## Bit-Pattern Based Pre-classification

Humans are very good at pattern recognition. Given any kind of document, depending on context, we are able to quickly find and process interesting content. When presented two documents with the same template (think of an invoice or any kind of business letter) we can compare those documents mentally and fixate our attention on paragraphs that differ in those documents since those paragraphs contain information we may seek.

Lets assume that we are working with a *cross* which can be encoded as a bit array *(1,1,1,1,1)* and whose one *<dbce>* tag is encoded as *(1,1,1,1,1)*. Such a DOM element occurs in all WARC entries and it is very likely that this element is part of the *boilerplate*. When some another DOM element was encoded to *(1,0,0,1,1)* it would mean that it is part of the *content* since it is shared across all WARC entries in the horizontal axis of the cross. Classification of *(1,0,1,0,1)* is not that obvious, it could be anything between changes in the template, dynamic DOM elements (advertisements) to editorial changes in an article. In this case we classify such element as *invalid*.

Following this example we propose a simple method for classification of HTML elements based on their occurrence in the cross. For the chosen encoding scheme we have identified eight possible combinations that describe valid *crosses*.

Starting with the simplest *cross* $:= (wEnCur, wEnUp, wEnDown, wEnPrev, wEnNext) = (1,1,0,0,0)$ we have only two combinations for *<dbce>* encoding: $(1,1,0,0,0)$ and $(1,0,0,0,0)$. A *<dbce>* element will be classified as *content* if it occurs only in $wEnCur$. B.1

The *<dbce>* tags can be classified as content when $wEnCur$, $wEnUp$, $wEnDown$ are present and those tags occur only in $wEnCur$. B.2

The previous two examples did not contain the timeseries components. In addition to $wEnCur$ and $wEnUp$ we may have newer or older versions of $wEnCur$. In this case the cross will be encoded as follow: $(1,1,0,0,1)$ or $(1,1,0,1,0)$ and we classify *<dbce>* element as content only if it occurs in $wEnCur$ and $wEnPrev$ or $wEnNext$. B.3

If both $wEnPrev$ and $wEnNext$ are available in addition to $wEnCur$ and $wEnUp$ the *<dbce>* elements that occurred in all versions of the horizontal axis are classified as content. B.4

The Bit-Patterns have been derived from empirical observations we did during the preparation work and testing. The complete list of the classifications has been listed in Appendix B. Patterns not listed in the appendix will be classified as *invalid*.

Starting from the deepest layer we process the groups of nodes in each layer using this simple bit-pattern classification. After one group of HTML elements, sharing the same parent, have been classified we postprocess this group and if all nodes share the same class or the number of elements in this group equals one, we set the parent's class to the class of its child element.

### Machine Learning

In the previous section we described a simple way to classify HTML elements based on their occurrence in the *cross*. However, not all bit-patterns can be classified manually. To resolve this problem we decided to train a classification model using Machine Learning (ML) methods. We have tested a variety of classifiers and will discuss the results in Section 4.

To train a ML model we need sufficient number of features to construct a vector which will be used for learning and classification. As short reminder - we are interested in proper classification of all remaining <*dbce*> HTML elements which are marked with the class *invalid*. The idea is to use the pre-classified elements and train the ML model.

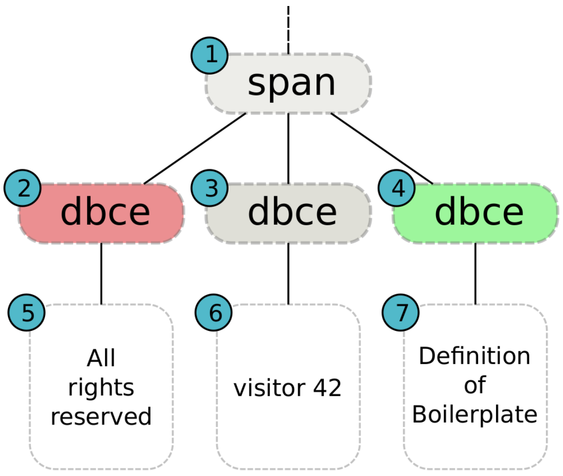


Figure 2.8.: Excerpt of a hypothetical DOM Tree with pre-classified HTML elements: green content, red boilerplate, gray invalid and white boxes contain text.

We take the Figure 2.8 as an example. The elements *5,6,7* are plain text. We collect information about those nodes, this includes the text length, number of lines, number of words. For HTML tags we collect number of tags, number of children, and text statistics, which would be the sum of all children's counters. In case of an <*a*> tag we would also collect text length of the *a href* element.

We start at the bottom of the tree and collect statistical data for all nodes in this layer. Then we go to the upper layer and process elements *2,3,4* and so on. While processing a HTML element we compute text and link density which is part of the *Densitometric Classifier* as Kohlschütter et al. proposed in their work on Boilerpipe [19]. All of those metrics are collected in *stage 1* but can be computed any time, since we do not change the text and the <*dbce*> tags are excluded from counted tags.

## Feature Extraction

For machine learning purposes, we collect several metrics for <*dbce*> nodes and use them as learning instances. Lets assume, that we are interested in features describing DOM node[7] *3* ($C$). The previous ($P$) and next ($N$) node are *2, 4*. If one or both nodes are not available we use internal dummy node instead. The following table contains all features describing one <*dbce*> node.

| Feature | Description |
|---|---|
| $P_{nid}$ | ID of the previous node's name |
| $C_{nid}$ | ID of the current node's name |
| $N_{nid}$ | ID of the next node |
| $C_{class}$ | sorted and hashed class attribute of the node |
| $P_{ld}$ | previous node's link density |
| $P_{td}$ | previous node's text density |
| $C_{ld}$ | current node's link density |
| $C_{td}$ | current node's text density |
| $N_{ld}$ | next node's link density |
| $N_{td}$ | next node's text density |
| $P_{wc}$ | previous node's word count |
| $C_{wc}$ | current node's word count |
| $N_{wc}$ | next node's word count |
| $P_{tl}$ | previous node's text length |
| $C_{tl}$ | current node's text length |
| $N_{tl}$ | next node's text length |
| $P_{tc}$ | previous node's tag count |
| $C_{tc}$ | current node's tag count |
| $N_{tc}$ | next node's tag count |
| $R_1$ | ratio text length to tag count |

Table 2.1.: Features describing one <*dbce*> node

Those metrics are stored as a vector. In some cases a vector may have the same values as one already "seen" but belonging to different class. If such conflict is detected both classes are discarded and removed from the learning set.

[7]representation of a HTML element in DOM tree

### Reclassification Process

As we described in the previous section not all bit-arrays can be unambiguously classified. The impact of missing classification can be seen in Figure 2.9a which shows excerpt of a Wired.com article on machine learning[8]. The text with green background was classified as content, yellow one is *invalid*. The missing classification will certainly impact the accuracy of CE resulting in missing parts of the sentences.

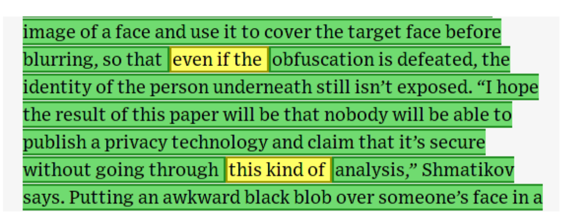


image of a face and use it to cover the target face before
blurring, so that even if the obfuscation is defeated, the
identity of the person underneath still isn't exposed. "I hope
the result of this paper will be that nobody will be able to
publish a privacy technology and claim that it's secure
without going through this kind of analysis," Shmatikov
says. Putting an awkward black blob over someone's face in a

(a) Pre-classified Document

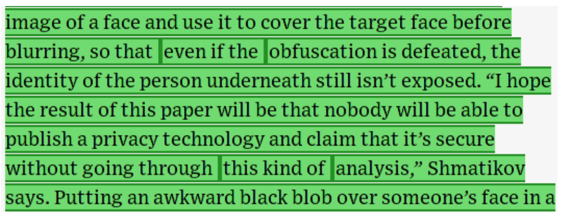


image of a face and use it to cover the target face before
blurring, so that even if the obfuscation is defeated, the
identity of the person underneath still isn't exposed. "I hope
the result of this paper will be that nobody will be able to
publish a privacy technology and claim that it's secure
without going through this kind of analysis," Shmatikov
says. Putting an awkward black blob over someone's face in a

(b) Reclassified Document

Figure 2.9.: Fragment of a Wired.com article.

We use the pre-classified <*dbce*> elements for training the model for given *cross* and then use it to reclassify all <*dbce*> tags. Optimal result would look like the one presented in Figure 2.9b.

Our approach is language, template independent. We do not use any language specific *stop-word lists*. We have tested this method on sites in German, English and Polish language with promising results and will describe the chosen classifiers in Section 4.

## 2.4. Stage 3: Postprocess

In this stage we already annotated all <*dbce*> nodes. The last step is to prepare final HTML document for *wEnCur*. We need to inject the original < *head* > ... < /*head* > back into the document. In our tests we have replaced all <*dbce*> tags with <*span*> and custom CSS for better visual feedback.

### Parallelization of the Process

We described in previous sections the way a raw HTML document passes from WARC file to its annotated version which can be used for CE. The whole pipeline can be executed on a single CPU, however the analyzed data can be passed between the stages as plain text thus it is possible to parallelize the whole process. We discuss the theoretical possibility to execute the process on parallel systems. We assume that a parallel system is build using multiple CPUs that can communicate with each other. We do not distinguish between one node with multiple CPUs and a cluster of networked nodes. The basic

[8]`https://www.wired.com/2016/09/machine-learning-can-identify-pixelated-faces-researchers-show/`, visited: 2016-09-26 15:01:16

principles should be the same. From now on we use the term *node* for a processing unit (CPU/cluster node).

First change to the pipeline would be in *stage 1* (see Section 2.2), where we do not query the *LSH index* but assign a node per *LSH bin/bucket*. Each bin contains WARC entries that, in some degree, have common HTML elements, thus, we can proceed with *stage 2* (cf. Section 2.3) and create *crosses* for all WARC entries in a bin. One of the LSH properties is, that some documents may be hashed into multiple bins therefore in *stage 3* a merging routine is needed. Such routine could use similar approach to our bit-pattern classifier. HTML elements which are the same in all annotated WARC entries can inherit the class and for conflicting classes ( boilerplate vs. content ) a machine learning approach could be used. Due to time constrains we were not able to test this method.

The pipeline can, theoretically be implemented in MapReduce manner. *Stage 1* would be a one Map-Reduce step, since accessing the WARC files and LSH can be implemented in such systems [2, 20]. *Stage 2* would be Map step where all LSH bins will be proceeded resulting in a list of annotated WARC entries followed by *stage 3* where the merging process, as shown in Figure 2.10, would be the reducer.

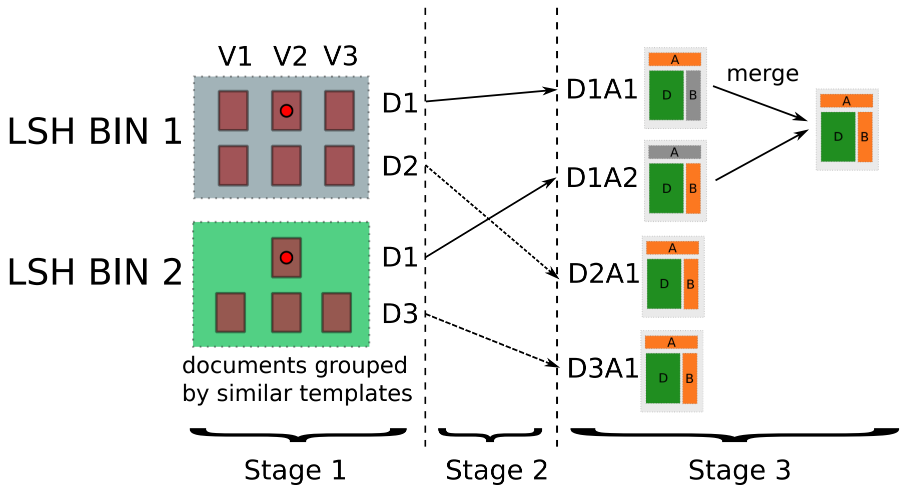


Figure 2.10.: Simplified view of the parallelized process including merge in stage 3.

The Figure 2.10 shows the situation in which HTML document *D1V2* (marked with red dot) has been hashed into two bins in *stage 1*. At the end of *stage 2* this document will have two different annotations (*D1A1*, *D1A2*), thus a merging procedure is needed.

# 3. Related Work

The extraction of content from HTML documents have been subject for research in the past. Single document *content extraction* and multidocument *template detection* are the two main fields of research. The focus of this section is on few selected works on which our approach builds upon and those whose problem solutions are not sufficient for our needs but are still closely related to our work.

It is worth mentioning that there is a competition named Cleaneval[1], which is a shared task and competitive evaluation on the topic of cleaning arbitrary Web pages, with the goal of preparing Web data for use as a corpus for linguistic and language technology research and development [21]. Cleaneval's dataset contains 740 documents but there are only 96 documents that have at least two documents per domain which is not sufficient for us, thus we do not test our method on this dataset.

## 3.1. Single Document Content Extraction

Single document CE (SDCE) methods focus on text extraction from single documents, without knowledge about the templates. SDCE concentrates on text extraction and usually misses pictures, media and tables.

One of the contestant of the Cleaneval competition was the BTE [22] method. BTE determines the largest contiguous text area with the least amount of HTML tags and marks it as "full text". The heuristic has been derived from an observations that the tag density within boilerplate text is higher than within fulltext content [23].

Kohlschütter et al. analyzed a set of shallow text features for classification of individual text elements in a Web page [19]. They derived a simple stochastic model for describing boilerplate creation process and with help of that model a simple heuristics for classification. In their work Kohlschütter et al. show that usage of *text density* and *link density* can be used for classification of blocks without significant accuracy loss. The *Densitometric Classifier* has been adapted in our work.

The tool jusText[2] based on the work of Jan Pomikálek classifies text blocks in two phases [24]. In the context-free classification each block will be assigned to one of four classes: *good* - main content, *bad* - boilerplate, *short* - which is too short to make reliable decision and *near-good* which is between good and short. Then in the context-sensitive classification the *near-good* and *short* blocks will be reclassified based on the observation that boilerplate elements are surrounded by other boilerplate elements and content blocks vice versa. JusText algorithm uses lists of language specific stop-words and prior to the

[1]`https://cleaneval.sigwac.org.uk/`, visited: 2017-01-04

[2]`https://github.com/miso-belica/jusText`, visited: 2016-12-21

classification the language of the analyzed document must be known, following jusText does not fulfill our needs.

## 3.2. Multidocument Template Detection

The methods of CE described earlier deal with single documents making assumptions on how the content looks like and how the documents are structured. The research field focusing on the analysis of collections of HTML documents in order to determine the underlying structure of the template is called *template detection*. The knowledge of the template structure allows to locate the main content. This includes the ability, besides of text extraction, to extract pictures, media and tables as valid content.

Bar-Yossef and Rajagopalan described the problem of template discovery as a task of Web *information retrieval* and Web *data mining* [3]. They gave a formal definition of *template* which has been adopted by other authors. In their work Bar-Yossef and Rajagopalan introduced the term *pagelet* which “is a self-contained logical region within a page that has a well defined topic or functionality”. Following by definition of a *template* which “. . . is a pre-prepared master HTML shell page that is used as a basis for composing new Web pages”. Thus a template is a collection of pagelets. The task of template detection on a graph of hyperlinked documents $G = (V_G, E_G)$ corresponds to finding all the templates and their pagelets which are part of the documents in $V_G$. Bar-Yossef and Rajagopalan’s method requires full de-duplication of the document set since their algorithms are sensitive to recurrent contents.

*InfoDiscoverer* has been developed by Lin and Ho which is based on the idea, that template generated contents appear more frequently [25]. In this method the documents are disassembled into blocks of text and for each block entropy value, based on the frequency of the words in the text blocks, is calculated. The authors assume that all documents of a website are based on the same template which is not suitable for our needs.

Ma, Goharian, Chowdhury and Chung developed a less complicated method in which they considered only the text fragments in Web documents and developed an approach for detecting redundancy [26]. The *document frequency based filter* is a very simple approach based on the document frequency of text fragments. The authors segmented Web pages according to a quite small predefined set of tags. The text contained in each of the resulting document segments is considered to be an information unit. Ma et al. then calculated the document frequency for each unit and stored it with the according text fragment in a suitable data structure. All text fragments which have a document frequency above a certain threshold are considered redundant thus part of the template.

The last two approaches assume that the underlying template of the Web pages is based on the “table layout” which is also not sufficient for our needs.

## 3.3. Combining a Segmentation-Like and Density-Based Approaches

The first attempt to combine the methods of CE algorithms and template detection used on collections of Web documents was introduced by Thomas Gottron in his PhD Thesis [27]. Starting from a single initial document they use a set of hyperlinked Web pages to build the required training set for the Template Detection automatically. By clustering the documents in this set according to their underlying templates they clean the training set from documents based on different templates. This method is very similar to ours, but we are not interested in the detection of any specific templates used by CMSs like WordPress, Joomla! and Typo3.

# 4. Experiments

For the verification of our method we have developed a test framework in *python3.5*[1] and manually created a ground truth to evaluate the results. Unit testing has been incorporated into the process using *tox*[2], which has been used to prepared dedicated environment, to analysis the code coverage and perform simple sanity checks using *pylint*[3]. A list of modules and dependencies used in our framework can be found in Appendix E.

For our tests we have collected a small number of HTML documents using the Web2Warc[4] crawler. A short summary of the collected data is presented in following table:

| Domain | Number of documents | Number of snapshots |
|---|---|---|
| `http://www.chefkoch.de` | 3 | 12 |
| `http://www.spiegel.de` | 3 | 12 |
| `http://www.heise.de` | 6 | 28 |
| `https://www.wired.com` | 3 | 12 |
| Total | 15 | 64 |

Table 4.1.: Summary of collected documents

We have created one WARC file and a corresponding CDX file per domain which then have been used for our test series. For the sake of this experiment we have choosen a random seed *2047226289*, which is used to seed all functions depending on randomness, to preserve the reproducibility of the tests.

In this section we will describe the dataset, annotation of the ground truth, testcases used and evaluate the results.

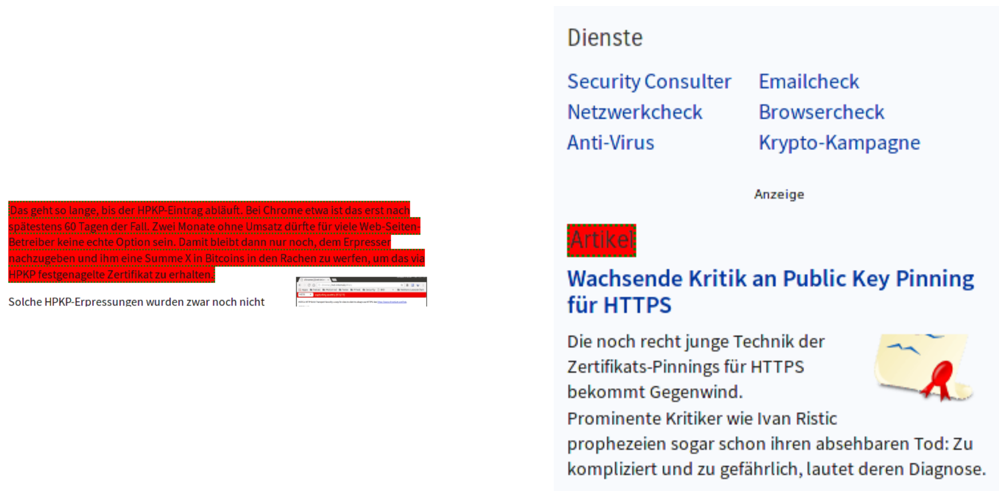
## 4.1. Preparation of the Dataset

We have collected information on all crosses for all the WARC entries in our dataset. For each domain two URL-paths have been picked at random. For each of those URL-paths two crosses have been randomly selected, such that both crosses differ in the number of available documents included in the cross. As mentioned in Section 2.3, a cross describes a set of WARC entries sharing similar templates. Every cross is identified by a *cross-unique-id*, which is a concatenation of hashed IDs[5] of each entry in the cross.

[1]`https://www.python.org`, visited: 2016-01-04
[2]`https://pypi.python.org/pypi/tox`, visited: 2016-01-04
[3]`https://pylint.org/`, visited: 2016-01-04
[4]`https://github.com/helgeho/Web2Warc`, visited: 2016-01-04
[5]the UUID of an WARC entry

### Ground Truth

We have 16 crosses for eight URL-Paths of four domains. We used *selenium*[6] for manual annotation of the chosen crosses. We incorporated the manual annotation process into the DBCE framework. A python script executes *stage 1* of the DBCE method for each domain (as described in Section 2.2) and presents the HTML document with highlighted DOM elements one by one in the selenium driver, asking the annotator for classification (content/boilerplate). A few examples can be found below:



Das geht so lange, bis der HPKP-Eintrag abläuft. Bei Chrome etwa ist das erst nach spätestens 60 Tagen der Fall. Zwei Monate ohne Umsatz dürfte für viele Web-Seiten-Betreiber keine echte Option sein. Damit bleibt dann nur noch, dem Erpresser nachzugeben und ihm eine Summe X in Bitcoins in den Rachen zu werfen, um das via HPKP festgenagelte Zertifikat zu erhalten.

Solche HPKP-Erpressungen wurden zwar noch nicht

Dienste

Security Consulter  Emailcheck
Netzwerkcheck  Browsercheck
Anti-Virus  Krypto-Kampagne

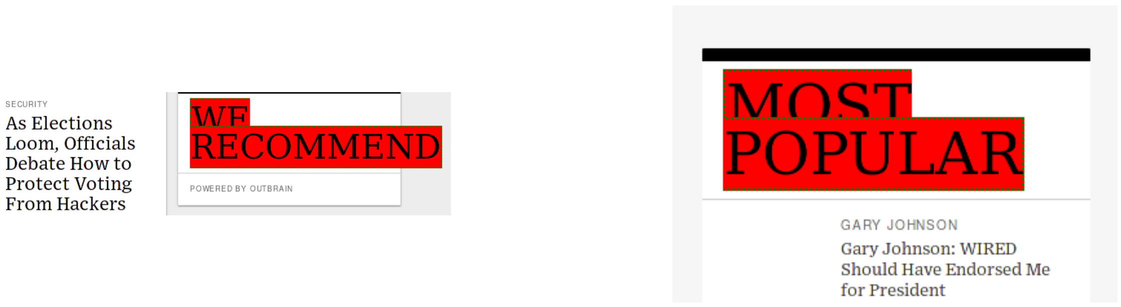
Anzeige

Artikel

Wachsende Kritik an Public Key Pinning für HTTPS

Die noch recht junge Technik der Zertifikats-Pinnings für HTTPS bekommt Gegenwind.
Prominente Kritiker wie Ivan Ristic prophezeien sogar schon ihren absehbaren Tod: Zu kompliziert und zu gefährlich, lautet deren Diagnose.

Figure 4.1.: Excerpt of www.heise.de article in annotation process.



SECURITY
As Elections Loom, Officials Debate How to Protect Voting From Hackers

WE RECOMMEND

POWERED BY OUTBRAIN

MOST POPULAR

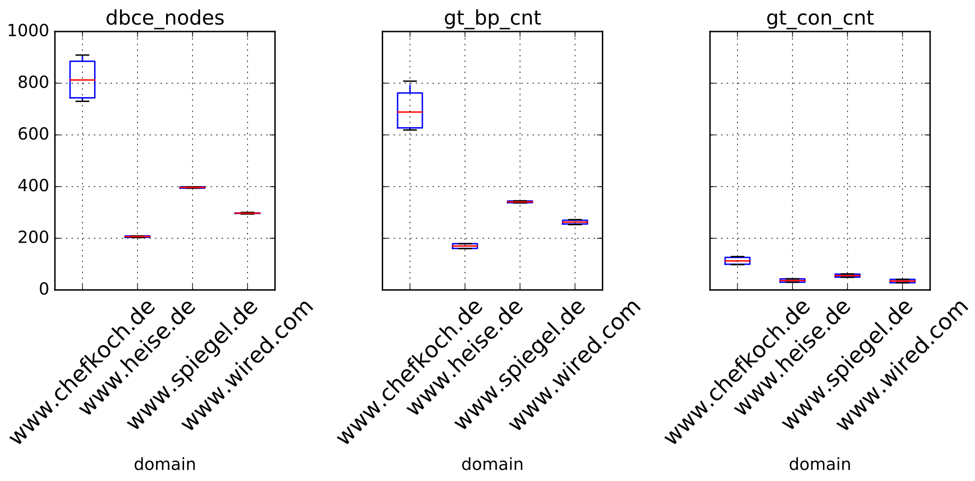
GARY JOHNSON
Gary Johnson: WIRED Should Have Endorsed Me for President

Figure 4.2.: Excerpt of www.wired.com article in annotation process.

The annotation has been done by one person and the result has been verified and accepted during a peer-review. Selenium actually renders the page and it is possible to check if DOM elements are visible. All invisible elements are automatically annotated as *boilerplate*.

Figure 4.3 shows an overview over the number of <*dbce*> nodes (dbce_nodes) and the number of annotated nodes in the ground truth. We have annotated over 98% of

[6]`http://www.seleniumhq.org/`, visited: 2016-01-04



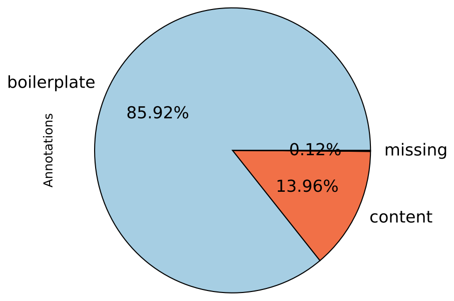


Figure 4.3.: Statistics over annotated DBCE nodes in ground truth.

the 6855 <*dbce*> nodes in all crosses. Our dataset contains 5897 *boilerplate* and 958 *content* elements. The average[7] ratio of *boilerplate/content* is 4.82. The number of *boilerplate* (gt_bt_cnt) nodes is much higher than *content* (gt_con_cnt). Thus misleading accuracy, of the results using ML, is almost certain. Furthermore, a negative impact on classification performance is expected for most methods. Nevertheless, addressing this problem is out of the scope of this work and we leave the problem of balancing the classes trough over-/under-sampling or by other methods [28] to the future work preparing the baseline.

[7]arithmetic mean

## 4.2. Test Cases

For the evaluation we have defined six tests. Each of those tests evaluates different parameters and approaches used in *stage 2*. We focus the discussion to *accuracy* of the classifier. In Appendix D we present the measurements of accuracy, precision, recall and f1-score for both classes (boilerplate and content) per domain in each test. In the testcases, if not otherwise stated, we use KNeighborsClassifier[8] which has been selected by a range of tests in the development phase.

```
    KNeighborsClassifier(algorithm='auto', leaf_size=30, metric='minkowski',
      metric_params=None, n_jobs=1, n_neighbors=5, p=2,
      weights='uniform')
```

Listing 4.1: Default parameters used for KNeighborsClassifier

The parameter tuning will be described in Section 5. We also defined a default threshold for Jaccard similarity $lsh\_threshold = 0.6$ for all domains besides of www.chefkoch.de for which the threshold has been set to 0.3 for testing purposes and because of the fact, that the documents for this domain are very *dynamic*. We will discuss this property in Section 5.

### Testcase T001

Our baseline for the test is the bit-pattern based classification as described in Section 2.3. The test framework classifies the <*dbce*> nodes using the bit patterns derived from the cross without using machine learning methods.

### Testcase T002

This testcase is an extension of *T001*, where all <*dbce*> tags classified as *invalid* are automatically reclassified as *boilerplate*.

### Testcase T003

In addition to *T001* we use machine learning for reclassification purpose. In the learning phase we use vectors that describe each <*dbce*> node and their neighborhood. This includes the previous and next node in the same layer group.

### Testcase T004

The vector used in testcase *T003* will be expanded to the vector describing the parent neighborhood of the analyzed <*dbce*> node. Which we call 2-layer neighborhood window.

[8]sklearn.neighbors.KNeighborsClassifier module of the scikit-learn toolkit, `http://scikit-learn.org/stable/`, visited: 2017-01-10

### Testcase T005

In this testcase the vector used in *T004* is expanded to 3-layer neighborhood window. Including current, parent and parent's parent neighborhood.

### Testcase T006

For this testcase a different classifier has been selected and a test run has been executed. The learning instances are the same as in *T005*. The selected classifier is a RandomForestClassifier[9]:

```
RandomForestClassifier(bootstrap=True, class_weight=None, criterion='gini',
  max_depth=None, max_features='auto', max_leaf_nodes=None,
  min_samples_leaf=1, min_samples_split=2,
  min_weight_fraction_leaf=0.0, n_estimators=10, n_jobs=1,
  oob_score=False, random_state=None, verbose=0,
  warm_start=False)
```

Listing 4.2: Default parameters used for RandomForestClassifier

## 4.3. Evaluation of the Results

In this section we will evaluate the results starting with timing analysis, which shows a correlation between the number of DOM elements in HTML document and the time needed for execution. Followed by the evaluation of the bit-pattern based classification and performance changes between the experiments.

### Timing Analysis

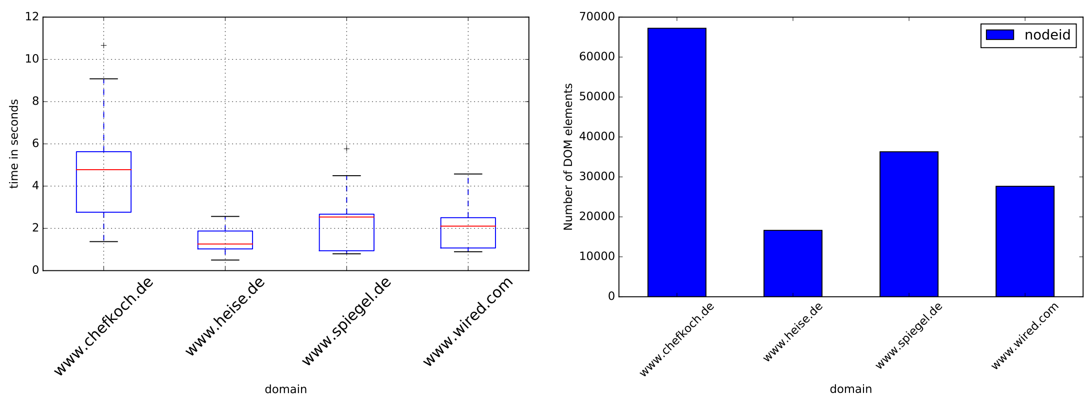


Figure 4.4.: Timing analysis of the process (left) and overall number of DOM elements processed by domain (right).

[9]module sklearn.ensemble.RandomForestClassifier

The Figure 4.4 shows the summary of consumed time in all tests per domain and the number of all DOM elements analyzed, per domain, for all crosses used in our experiment. The time needed to process one cross depends on the number of DOM elements in all WARC entries in this cross. In our experiment we used *BeautifulSoup 4*[10] and *lxml* for parsing the HTML documents. During the tests we have encountered some performance issues with BeautifulSoup and the way it access the attributes. We were able to limit the number of attribute accesses in the BeautifulSoup API using a data structure that cached all relevant information about each node in a HTML document. This includes the class, tag name and the parent.

Following Figures 4.5 and 4.6 show the time spent in each phase of *stage 2*. *annotate* is the first phase in which the *cross* will be prepared followed by *classify* where all <*dbce*> nodes in *wEnCur* will be classified using the bit-patter based classifier. *classify_pp* is the time in which we postprocess the <*dbce*> nodes and propagate the class informations to the parent (cf. Section 2.3). Most of the time is used by the learn and reclassification phase, which mostly depends on choosen classifier and the number of features that have been used.

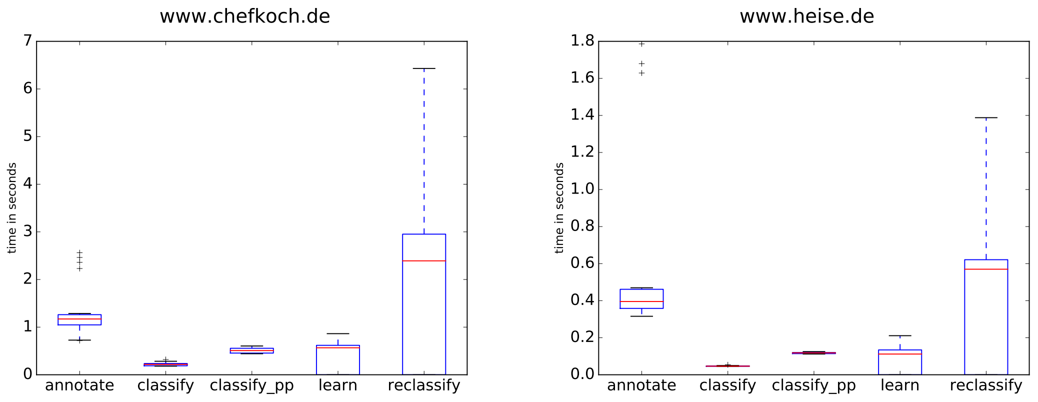


Figure 4.5.: Timing analysis of the stages for www.chefkoch.de and www.heise.de.

## Evaluation of the Bit-Pattern Classification and Reclassification

The Figure 4.7 shows an overview over accuracy of the test cases (see Section 4.2). The accuracy of the bit-pattern based classification (*T001*), varies between 81% and 97%, which is a very promising result[11]. As mentioned before, we do not use any heuristics to classify the DOM elements, thus the accuracy may vary depending on cross, time difference between snapshots as well as the number of changes between the WARC entries. All those factors are subject for research in future work.

[10] `https://www.crummy.com/software/BeautifulSoup/bs4/doc/`, visited: 2017-01-04

[11] All statistical analysis has been conducted using default parameters with scikit-learn and pandas python modules

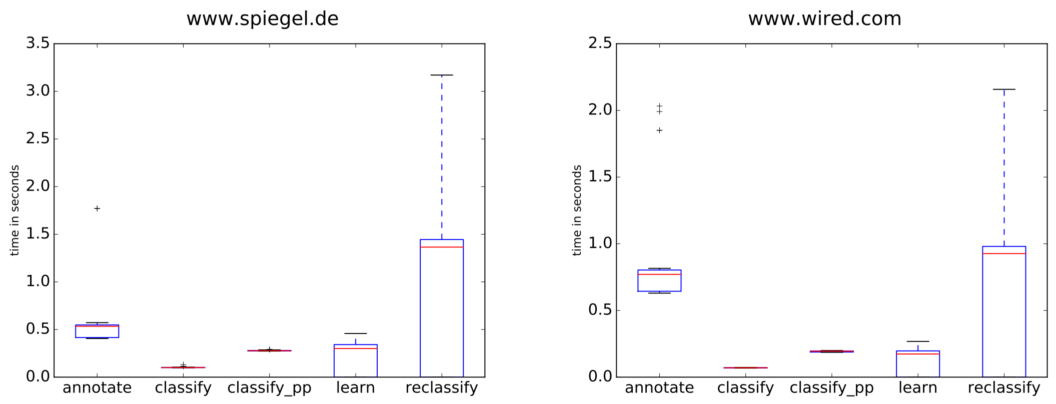


Figure 4.6.: Timing analysis of the stages for www.spiegel.de and www.wired.com.

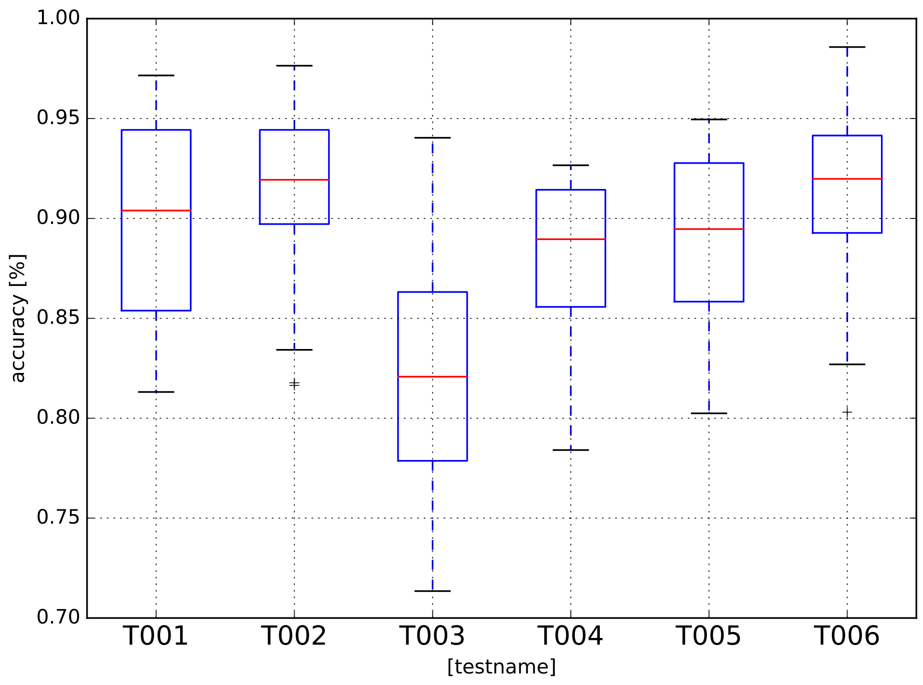


Figure 4.7.: Accuracy of the classification grouped by testname.

The test case *T002* performed slightly better than *T001*. In this case we reclassify all *invalid* nodes as *boilerplate*. This result indicates, that there is some relation between the *invalid* nodes and the number of WARC entries in the cross, which we do not fully understand yet. In *T003* we observe a significant drop in accuracy. This shows that it is not sufficient to look only at the direct neighborhood of a node. The accuracy of the reclassification increases while including the neighborhood of node's parent (*T004*).

Clearly the choosen features have impact on the accuracy. Expanding the vector to parent's parent node yields on average better results as can been seen in *T005*. However, even with such a wide range of features *T005* performs slightly worse than bit-pattern based classification. In *T006* we have used RandomForestClassifier which performed slightly better than KNeighborsClassifier used in *T003, T004* and *T005*.

This shows clearly that feature extraction and the selection of the classifier impacts the accuracy of the classification and should be researched in future work. Figure 4.8 shows a more finegrained view of classification performance for combinations of domains and methods. While the results for www.chefkoch.de and www.wired.com show the typical drop in accuracy between *T001* and *T003*, www.heise.de and www.spiegel.de are outliers as shown in Figure 4.8.

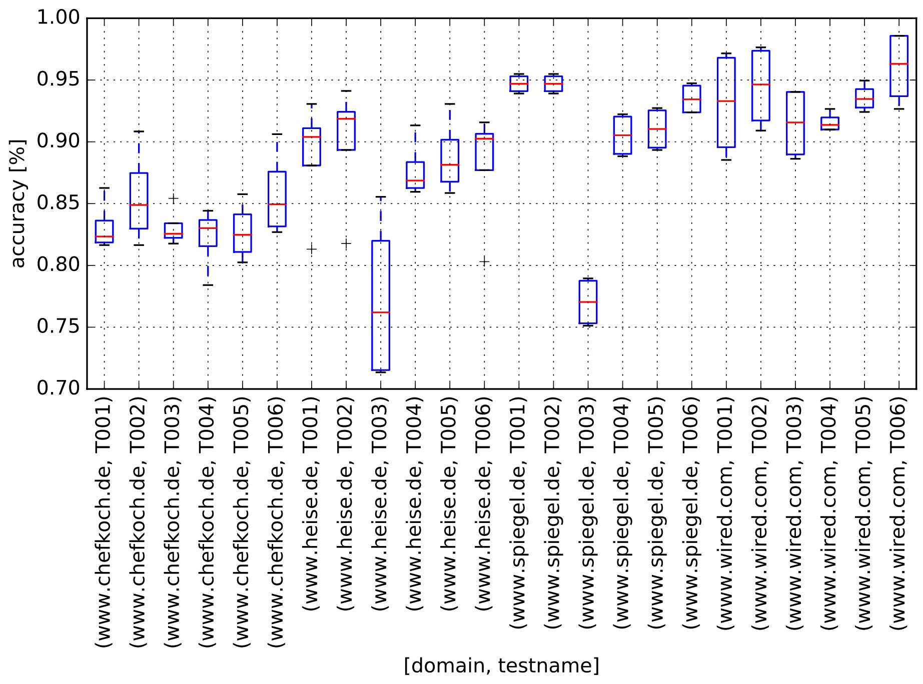


Figure 4.8.: Accuracy of the classification grouped by domain.

The drop is about 0.20 which can be explained by the fact that the classes in the learning set are not balanced, thus ML algorithms have more instances of *boilerplate* than *content* and tend to prefer the first one in classification. We will show an example of the result and discuss this problem in further sections.

In the following we take a closer look at the classification results of the three best performing variations of our method – *T001*, *T002*, and *T006*.

| | **T001** | **T002** | **T006** |
|---|---|---|---|
| Accuracy | 88.3% | **89.9**% | 89.6% |
| | *Content Performance* | | |
| Precision | 59% | 59% | **63**% |
| Recall | **91.8**% | 90.9% | 86.1% |
| F1-Score | 71.8% | 71.6% | **72.8**% |
| | *Boilerplate Performance* | | |
| Precision | 98.2% | **98.4**% | 97.1% |
| Recall | 87.7% | 89.7% | **90.2**% |
| F1-Score | 92.7% | **93.8**% | 93.5% |

Table 4.2.: Measurements summary over the best performing methods.

The Table 4.2 shows a summary of measurements over the best performing methods and the complete measurements are presented in Appendix D (cf. Figures D.1,D.2 and D.3).

The accuracy of our baseline experiment *T001*, with 88.3%, is already quite high. With 91.8% recall of content and 98.2% precision for boilerplate we can conclude, that the method is conservative in detecting the boilerplate – discarding content only with high confidence, which is desired behavior.

The *T002* method achieved the best overall accuracy, with 89.9%. Similar to before, precision for boilerplate (98.4%) and recall for content (90.9%) are very high, while errors are reflected in lower precision for content (59%). However, compared to the results for *T001*, recall for the content class is slightly lower (90.9%), due to some of the *invalid* elements that are classified as boilerplate by default in this configuration, having been assigned class label *content* during the manual ground truth creation process.

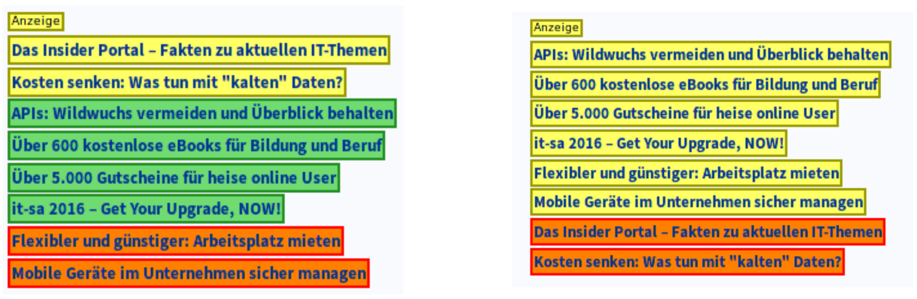
Overall accuracy of the *T003* method, with 89.6%, is slightly higher than *T001* with 88.3%. The gains in classification performance are made in terms of filtering boilerplate, with an increased precision value w.r.t. content class of 63% (in comparison to 59% previously). However, recall on the other hand is reduced to 86.1% in comparison to 91.8%. Apparently, using a machine learning algorithm in conjunction with the imbalanced training set introduces a different trade-off, where more boilerplate is filtered, while some content is lost. It is worth noting that we do not evaluate relative importance of individual content elements, i.e. whether any of the key content elements (e.g. a div element with a large body of text) is lost in the process. Such a change in relative performances per class might well be desirable for some use cases depending on this. Furthermore, balancing the training set in future work might address the reduction in recall of the content class, as well as lead to overall improvements of classification performance.

# 5. Discussion

## Static vs Dynamic HTML Pages

Our methods shows very good results for HTML documents that have more static nature. Example of such HTML documents are news articles from www.spiegel.de or www.wired.com. Documents from www.heise.de, depending on the passed time between snapshots, show more changes in boilerplate elements. Which we simply call “dynamic” nature. The Figure 5.1 shows excerpt of two HTML documents from www.heise.de, where the advertisements are dynamically embedded into the content of the HTML document. While taking snapshots of documents from www.heise.de, www.spiegel.de and www.wired.com in very short timespans (under few seconds to few minutes) will yield almost the same content, doing the same with documents from www.chefkoch.de will result in very different content. This may vary, depending on the age of a recipe, probably older recipes will have lower rate of comments thus content will stay static over time. Finding the right time span between snapshots can be very hard depending on the type of the website one is interested in.

Since our method groups documents based on the template similarity, changes in advertisements or other static locations in HTML document can be processed without any problems. Such elements will be marked as “invalid” depending on the current cross encoding, because the template does not change. The impact of cross “fullness” can be seen in the excerpts of `http://www.heise.de` in following figure:


Anzeige
Das Insider Portal – Fakten zu aktuellen IT-Themen
Kosten senken: Was tun mit "kalten" Daten?
APIs: Wildwuchs vermeiden und Überblick behalten
Über 600 kostenlose eBooks für Bildung und Beruf
Über 5.000 Gutscheine für heise online User
it-sa 2016 – Get Your Upgrade, NOW!
Flexibler und günstiger: Arbeitsplatz mieten
Mobile Geräte im Unternehmen sicher managen

Anzeige
APIs: Wildwuchs vermeiden und Überblick behalten
Über 600 kostenlose eBooks für Bildung und Beruf
Über 5.000 Gutscheine für heise online User
it-sa 2016 – Get Your Upgrade, NOW!
Flexibler und günstiger: Arbeitsplatz mieten
Mobile Geräte im Unternehmen sicher managen
Das Insider Portal – Fakten zu aktuellen IT-Themen
Kosten senken: Was tun mit "kalten" Daten?

Figure 5.1.: Partial cross on the left, uuid:47346f56-5d34-4fcb-b175-18e2bb9d864d and full cross on right uuid:a3c71aa6-cca0-488c-971c-2c12c746e1ed (cf. Table A.2).

Elements highlighted in yellow are of the class *invalid*. For the document on the right, since the testcase *T002* showed promising results, it could be possible to automatically classify all those elements as boilerplate. To properly classify the green elements in the document on the left we need ML like methods. The *content* classification is based on the fact, that those elements are shared among the horizontal axis in the cross and never occurred in the vertical axis.

Chefkoch's documents contain user comments which can be commented again. New comments change, by addition of another HTML elements, the template. Such changes between snapshots would sort the documents into different LSH bins reducing the accuracy or even, depending on the number of used bins, leave such HTML document without matches (similarity threshold would not be satisfied). The default, minimal, Jaccard similarity (0.6) was not satisfied for documents from www.chefkoch.de. Thus, as we mentioned in Section 4.2, we have changed the minimal Jaccard similarity to 0.3 for this domain. Following, the templates of the WARC entries in any cross, are not very similar. We conclude that the accuracy of our methods depends on the similarity of the templates. However, the relevance must be checked in future work.

## Reclassification

We introduced the ML components in hope to solve the problem with stop words being classified as boilerplate. However, we think that this problem can be solved by other methods. One would be to use the densitometric classifier or the method used by jusText where a given "invalid" element has a previous ($P$) and next ($N$) neighbors. If the $P_{td}$ and $N_{td}$ are "large enough" and both neighbors are *content*, then we could reclassify the "invalid" element as content too.

### Imbalanced Datasets

Based on the results of the bit-pattern classification we know, that there are more boilerplate than content instances. Balancing the datasets can be done by variety of methods like over-/under-sampling [28].

## Hidden Data

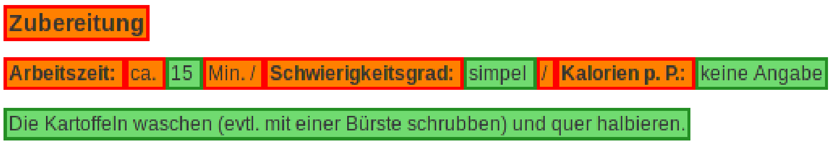
The problem with imbalanced datasets can be amplified by DOM elements that are in the HTML document but have a CSS property "hidden". A Web browser would not render such element, thus the question is: "Are such elements part of the content or boilerplate?". If the user can not see it, then it certainly will not answer the question the user had. In our method we do not have the ability to filter such elements out of the learning set.

```
<span class="visually-hidden">
 <span class="dbce-class-invalid␣dbce-marker␣dbce-bits-10000">
  <dbce class="cur-Gah5aipa">
   Microsoft Bets Its Future on a Reprogrammable Computer Chip
  </dbce>
 </span>
</span>
```

Listing 5.1: Excerpt of a wired.com article showing an DOM element which is not visible in Web browser

## Recurrent Elements

Some sites may serve HTML documents which contain the same content on purpose, like cooking recipes from `http://www.chefkoch.de`. Following figures shows an excerpt of a recipe, where elements of the document *uuid:014ff911-194d-4c81-9ee0-b9a5928d66c6* have been classified as boilerplate (orange).

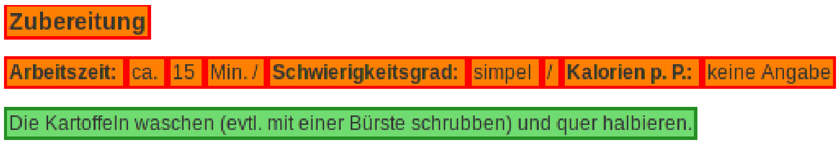


Zubereitung

Arbeitszeit: ca. 15 Min. / Schwierigkeitsgrad: simpel / Kalorien p. P.: keine Angabe

Die Kartoffeln waschen (evtl. mit einer Bürste schrubben) und quer halbieren.

Figure 5.2.: Testcase T001.



Zubereitung

Arbeitszeit: ca. 15 Min. / Schwierigkeitsgrad: simpel / Kalorien p. P.: keine Angabe

Die Kartoffeln waschen (evtl. mit einer Bürste schrubben) und quer halbieren.

Figure 5.3.: Testcase T006.

The Figure 5.2 shows the result of the bit-pattern based classification (*T001*) and Figure 5.3 the result of the ML based reclassification (*T006*). Both tests failed to classify the elements properly which can be explained by the fact, that all of those elements are common in the cross and the lack of learning instances for the ML methods.

## Parameter Tuning

We have conducted our experiments using default parameters of the estimators in order to generate baseline results and show that our methods can be used in further experiments. We expect better results by tuning the estimator parameters by using GridSearchCV for exhaustive search or randomized parameter optimization using RandomizedSearchCV [29].

# 6. Conclusions and Future Work

**Conclusions.** In this work, we have presented our joint approach to boilerplate detection in Web archives. We described a way to group WARC entries based on their template similarity using MinHash and LSH. The data structure *cross* has been introduced, which allowed us to make simple, bit-pattern based, classification of HTML elements in sets of similar HTML documents. We implemented a test framework which implements the three stages of our boilerplate detection method and conducted a series of tests in order to evaluate the bit-pattern based classification. We tested different ways to improve the classification using ML methods. We also described one way to parallelize the pipeline using MapReduce. The evaluation of our methods showed that *T001* is very conservative in detecting the boilerplate, reaching 91.8% recall of content with 98.2% precision for boilerplate. Depending on the use case, *T002* can be applied, which is more aggressive in boilerplate detection. With 89.7% recall and 98.4% for precision for boilerplate we were still able to retrieve 90.9% of content.

**Future Work.** The bit-pattern based classification yields promising results, however due to time constrains and insufficient data sets, we were not able to compare our approach to other algorithms, thus the preparation of larger data sets is needed. Encoding of the *cross* does not cover all possible combinations, beside of *content, boilerplate* classes an *invalid* class has been introduced. With respect to the results of the evaluation of the method *T002* in which we have purposefully reclassified the *invalid* class as *boilerplate*, further evaluation of the bit-patterns is necessary. We proposed a range of different features and parameters used for machine learning. In the future, a method for balancing the instances used for ML should be evaluated, as well the used classifiers. The accuracy of our method seems to depend on the "fullness" of the cross and the timespan between taken snapshots of a Web document. This may be related to *T002* which performs better than *T001*, thus the impact of changing timespan between snapshots and number of entries in the cross should be analyzed as well.

# References

[1] Jure Leskovec, Anand Rajaraman, and Jeffrey David Ullman. Mining of massive datasets, 2014. URL `http://www.mmds.org/#book`. [Online; accessed 2017-01-04].

[2] Helge Holzmann, Vinay Goel, and Avishek Anand. Archivespark: Efficient web archive access, extraction and derivation. In *Proceedings of the 16th ACM/IEEE-CS on Joint Conference on Digital Libraries*, pages 83–92. ACM, 2016.

[3] Ziv Bar-Yossef and Sridhar Rajagopalan. Template detection via data mining and its applications. In *Proceedings of the 11th international conference on World Wide Web*, pages 580–591. ACM, 2002.

[4] AFR Rahman, H Alam, and R Hartono. Content extraction from html documents. In *1st Int. Workshop on Web Document Analysis (WDA2001)*, pages 1–4, 2001.

[5] The World Wide Web Consortium. A vocabulary and associated apis for html and xhtml, 2014. URL `https://www.w3.org/TR/html5/`. [Online; accessed 2017-01-04].

[6] The World Wide Web Consortium. Custom elements, 2016. URL `https://www.w3.org/TR/custom-elements/`. [Online; accessed 2017-01-04].

[7] Wolfgang G Stock and Mechtild Stock. Handbook of information science, 2013.

[8] Claude Elwood Shannon. A mathematical theory of communication. *ACM SIGMOBILE Mobile Computing and Communications Review*, 5(1):3–55, 2001.

[9] Luciano Floridi. *Information: A very short introduction.* OUP Oxford, 2010.

[10] ISO 28500:2009(E). Information and documentation - the warc file format. Technical report, International Organization for Standardization, 2009.

[11] Andrei Z Broder. On the resemblance and containment of documents. In *Compression and Complexity of Sequences 1997. Proceedings*, pages 21–29. IEEE, 1997.

[12] Michael O Rabin et al. *Fingerprinting by random polynomials.* Center for Research in Computing Techn., Aiken Computation Laboratory, Univ., 1981.

[13] Andrei Z Broder. Identifying and filtering near-duplicate documents. In *Annual Symposium on Combinatorial Pattern Matching*, pages 1–10. Springer Berlin Heidelberg, 2000.

[14] Edith Cohen, Mayur Datar, Shinji Fujiwara, Aristides Gionis, Piotr Indyk, Rajeev Motwani, Jeffrey D Ullman, and Cheng Yang. Finding interesting associations without support pruning. *IEEE Transactions on Knowledge and Data Engineering*, 13(1):64–78, 2001.

[15] The World Wide Web Consortium. Document object model (dom) level 1 specification, 1998. URL `https://www.w3.org/TR/REC-DOM-Level-1/`. [Online; accessed 2017-01-04].

[16] The World Wide Web Consortium. Document object model (dom) level 2 specification, 2000. URL `https://www.w3.org/TR/DOM-Level-2/`. [Online; accessed 2017-01-04].

[17] The World Wide Web Consortium. Document object model (dom) level 3 specification, 2004. URL `https://www.w3.org/TR/DOM-Level-3-Core/`. [Online; accessed 2017-01-04].

[18] Thomas Ball and Fred Douglis. An internet difference engine and its applications. In *Compcon'96.'Technologies for the Information Superhighway'Digest of Papers*, pages 71–76. IEEE, 1996.

[19] Christian Kohlschütter, Peter Fankhauser, and Wolfgang Nejdl. Boilerplate detection using shallow text features. In *Proceedings of the Third ACM International Conference on Web Search and Data Mining*, WSDM '10, pages 441–450, New York, NY, USA, 2010. ACM. ISBN 978-1-60558-889-6. doi: 10.1145/1718487.1718542. URL `http://doi.acm.org/10.1145/1718487.1718542`.

[20] Bahman Bahmani, Ashish Goel, and Rajendra Shinde. Efficient distributed locality sensitive hashing. In *Proceedings of the 21st ACM international conference on Information and knowledge management*, pages 2174–2178. ACM, 2012.

[21] Adam Kilgarriff Marco Baroni, Francis Chantree and Serge Sharoff. Cleaneval: a competition for cleaning web pages. In *Proceedings of the Sixth International Conference on Language Resources and Evaluation (LREC'08)*, Marrakech, Morocco, may 2008. European Language Resources Association (ELRA). ISBN 2-9517408-4-0. http://www.lrec-conf.org/proceedings/lrec2008/.

[22] Aidan Finn. Bte: Body text extraction, 2001. URL `http://www.aidanf.net/posts/bte-body-text-extraction`. [Online; accessed 2016-12-13].

[23] Aidan Finn, Nicholas Kushmerick, and Barry Smyth. Fact or fiction: Content classification for digital libraries, 2001.

[24] Jan Pomikálek. *Removing boilerplate and duplicate content from web corpora.* PhD thesis, Masaryk university, Faculty of informatics, Brno, Czech Republic, 2011.

[25] Shian-Hua Lin and Jan-Ming Ho. Discovering informative content blocks from web documents. In *Proceedings of the eighth ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 588–593. ACM, 2002.

[26] Ling Ma, Nazli Goharian, Abdur Chowdhury, and Misun Chung. Extracting unstructured data from template generated web documents. In *Proceedings of the twelfth international conference on Information and knowledge management*, pages 512–515. ACM, 2003.

[27] Thomas Gottron. *Content extraction: Identifying the main content in HTML documents*. PhD thesis, Johannes Gutenberg-Universität, 2008.

[28] Nitesh V Chawla. Data mining for imbalanced datasets: An overview. In *Data mining and knowledge discovery handbook*, pages 875–886. Springer, 2009.

[29] Scikit learn developers. Tuning the hyper-parameters of an estimator, 2016. URL `http://scikit-learn.org/stable/modules/grid_search.html#grid-search`. [Online; accessed 2017-01-04].

# Appendices

# Glossary

| | |
|---|---|
| accuracy | $$Accuracy = \frac{TP + TN}{TP + TN + FP + FN} \quad (.1)$$ with True Positive (TP), True Negative (TN), False Positive (FP) and False Negative (FN), is a weighted arithmetic mean of precision and inverse precision as well as a weighted arithmetic mean of recall and inverse recall. 33 |
| f1-score | $$F1 = 2 \cdot \frac{precision \cdot recall}{precision + recall} \quad (.2)$$ is a measure of a test's accuracy. 42 |
| HTML element | part of a HTML document representing a logical unit, e.g. paragraph, link or an image. 13 |
| precision | $$Precision = \frac{TP}{TP + FP} \quad (.3)$$ is the percentage of matches which are correct matches. 42 |
| recall | $$Recall = \frac{TP}{TP + FN} \quad (.4)$$ is the fraction of the documents that are relevant to the query that are successfully retrieved. 42 |
| template | A master HTML doucment, which provides a common layout of a Web page in a content management systems. Templates are filled with data in the process of page generation. 14 |
| Web browser | is a software application used for retrieving, presenting and traversing information resources on the World Wide Web. 50 |

| | |
|---|---|
| Web page | document used in World Wide Web, which can be presented by a Web browser. 13 |
| XPath | XPath is a language for addressing parts of an XML document, designed to be used by both XSLT and XPointer. 26 |

# Abbreviations

**BTE** Body-Text-Extraction. 15

**CE** Content Extraction. 15

**CMS** Content Management System. 14

**CSS** Cascading Style Sheets. 26

**DBCE** Diff Based Content Extraction. 19

**DOM** Document Object Model. 23

**FN** False Negative. 61

**FP** False Positive. 61

**GB** gigabyte. 16

**KB** kilobyte. 16

**LSH** locality sensitive hashing. 20

**MinHash** Min-wise independent permutations. 20

**ML** Machine Learning. 31

**SDCE** Single document CE. 35

**TN** True Negative. 61

**TP** True Positive. 61

**WARC** Web ARChive. 19

# A. List of URLs we used for our experiment

Short summary of URL-paths with associated uuid, timestamps and encoded crosses:

| Attribute | Value |
|---|---|
| uuid | 014ff911-194d-4c81-9ee0-b9a5928d66c6 |
| URL-path | /rezepte/839381188998764/Ofenkartoffeln-mit-frischen-Kraeutern.html |
| timestamp | 20160921224549 |
| cross bits | 11111 |
| uuid | 7860509f-9400-4f5e-8786-58fb4fb00f17 |
| URL-path | /rezepte/839381188998764/Ofenkartoffeln-mit-frischen-Kraeutern.html |
| timestamp | 20161022030429 |
| cross bits | 11110 |
| uuid | a9799eb9-5a0b-45a8-a236-905547cbef3c |
| URL-path | /rezepte/932451198676837/Original-Wiener-Schnitzel-mit-Petersilienkartoffeln.html |
| timestamp | 20161022030418 |
| cross bits | 11110 |
| uuid | 0e480386-a02f-4d28-aabc-9442ae2840b8 |
| URL-path | /rezepte/932451198676837/Original-Wiener-Schnitzel-mit-Petersilienkartoffeln.html |
| timestamp | 20160921224544 |
| cross bits | 11010 |

Table A.1.: WARC entries for www.chefkoch.de

| Attribute | Value |
|---|---|
| uuid | 47346f56-5d34-4fcb-b175-18e2bb9d864d |
| URL-path | /security/artikel/Wachsende-Kritik-an-Public-Key-Pinning-fuer-HTTPS-3324703.html |
| timestamp | 20160921224212 |
| cross bits | 11101 |
| uuid | a3c71aa6-cca0-488c-971c-2c12c746e1ed |
| URL-path | /security/artikel/Wachsende-Kritik-an-Public-Key-Pinning-fuer-HTTPS-3324703.html |
| timestamp | 20160921224546 |
| cross bits | 11111 |
| uuid | 4929bee0-0df6-45e4-b1cf-35b8df45a1db |
| URL-path | /security/meldung/Forscher-demonstriert-erfolgreichen-Angriff-auf-iPhone-Sperre-3326826.html |
| timestamp | 20160921224547 |
| cross bits | 11111 |
| uuid | 49cd031f-77e0-4d61-9b39-b1f968b99b09 |
| URL-path | /security/meldung/Forscher-demonstriert-erfolgreichen-Angriff-auf-iPhone-Sperre-3326826.html |
| timestamp | 20160926150113 |
| cross bits | 11110 |

Table A.2.: WARC entries for www.heise.de

| Attribute | Value |
|---|---|
| uuid | 5dcac983-cf64-4d60-a83c-0aa13ac36ab7 |
| URL-path | /wirtschaft/unternehmen/subventionsskandal-in-chinas-e-auto-industrie-a-1112035.html |
| timestamp | 20160926150118 |
| cross bits | 11010 |
| uuid | 957ae5dc-edb1-4f0e-bacd-e01599a36fea |
| URL-path | /wirtschaft/unternehmen/subventionsskandal-in-chinas-e-auto-industrie-a-1112035.html |
| timestamp | 20160921224550 |
| cross bits | 11011 |
| uuid | c1b6bfcd-9b7a-452d-a57a-887928284ecf |
| URL-path | /wirtschaft/unternehmen/hanjin-pleite-treibt-container-frachtraten-nach-oben-a-1112029.html |
| timestamp | 20160926150108 |
| cross bits | 11010 |
| uuid | 3b071332-86a3-4b8d-9a80-e60090e27eb3 |
| URL-path | /wirtschaft/unternehmen/hanjin-pleite-treibt-container-frachtraten-nach-oben-a-1112029.html |
| timestamp | 20160921224544 |
| cross bits | 11011 |

Table A.3.: WARC entries for www.spiegel.de

| Attribute | Value |
|---|---|
| uuid | 03808d91-4f61-4471-bcd1-81f835323138 |
| URL-path | /2016/09/machine-learning-can-identify-pixelated-faces-researchers-show/ |
| timestamp | 20160926150116 |
| cross bits | 11110 |
| uuid | 6e2b623c-ba6d-4396-b4ec-0c1d1c699fa0 |
| URL-path | /2016/09/machine-learning-can-identify-pixelated-faces-researchers-show/ |
| timestamp | 20160921224549 |
| cross bits | 11111 |
| uuid | 8c742c8d-5b22-432e-84e8-f586be537690 |
| URL-path | /2016/09/police-trace-cellphones-ieds-like-ones-nyc/ |
| timestamp | 20160921224548 |
| cross bits | 11111 |
| uuid | a8d7c494-b428-4cf9-8055-7b04382bdca7 |
| URL-path | /2016/09/police-trace-cellphones-ieds-like-ones-nyc/ |
| timestamp | 20160926150114 |
| cross bits | 11110 |

Table A.4.: WARC entries for www.wired.com

# B. Bit-Pattern Classification

Table B.1.: Classification of crosses with two entries

| cross bits | dbce-tag bits | classification |
|---|---|---|
| (1, 1, 0, 0, 0) | (1, 0, 0, 0, 0) | dbce-class-content |
| | (1, 1, 0, 0, 0) | dbce-class-bp |

Table B.2.: Classification of crosses with three entries (v1)

| cross bits | dbce-tag bits | classification |
|---|---|---|
| (1, 1, 1, 0, 0) | (1, 0, 0, 0, 0) | dbce-class-content |
| | (1, 0, 1, 0, 0) | dbce-class-bp |
| | (1, 1, 0, 0, 0) | dbce-class-bp |
| | (1, 1, 1, 0, 0) | dbce-class-bp |

Table B.3.: Classification of crosses with three entries (v2)

| cross bits | dbce-tag bits | classification |
|---|---|---|
| (1, 1, 0, 0, 1) | (1, 0, 0, 0, 0) | dbce-class-bp |
| | (1, 0, 0, 0, 1) | dbce-class-content |
| | (1, 1, 0, 0, 0) | dbce-class-bp |
| | (1, 1, 0, 0, 1) | dbce-class-bp |
| (1, 1, 0, 1, 0) | (1, 0, 0, 0, 0) | dbce-class-bp |
| | (1, 0, 0, 1, 0) | dbce-class-content |
| | (1, 1, 0, 0, 0) | dbce-class-bp |
| | (1, 1, 0, 1, 0) | dbce-class-bp |

Table B.4.: Classification of crosses with up to four entries

| cross bits | dbce-tag bits | classification |
|---|---|---|
| (1, 1, 0, 1, 1) | (1, 0, 0, 0, 0) | dbce-class-bp |
| | (1, 0, 0, 0, 1) | dbce-class-bp |
| | (1, 0, 0, 1, 0) | dbce-class-bp |
| | (1, 0, 0, 1, 1) | dbce-class-content |
| | (1, 1, 0, 0, 0) | dbce-class-bp |
| | (1, 1, 0, 0, 1) | dbce-class-bp |
| | (1, 1, 0, 1, 0) | dbce-class-bp |
| | (1, 1, 0, 1, 1) | dbce-class-bp |

Table B.5.: Classification of crosses with up to five entries

| cross bits | dbce-tag bits | classification |
|---|---|---|
| (1, 1, 1, 0, 1) | (1, 0, 0, 0, 1) | dbce-class-content |
| | (1, 1, 1, 0, 1) | dbce-class-bp |
| (1, 1, 1, 1, 0) | (1, 0, 0, 1, 0) | dbce-class-content |
| | (1, 1, 1, 1, 0) | dbce-class-bp |
| (1, 1, 1, 1, 1) | (1, 0, 0, 1, 1) | dbce-class-content |
| | (1, 1, 1, 1, 1) | dbce-class-bp |

# C. Screenshots of annotated HTML documents

## www.heise.de

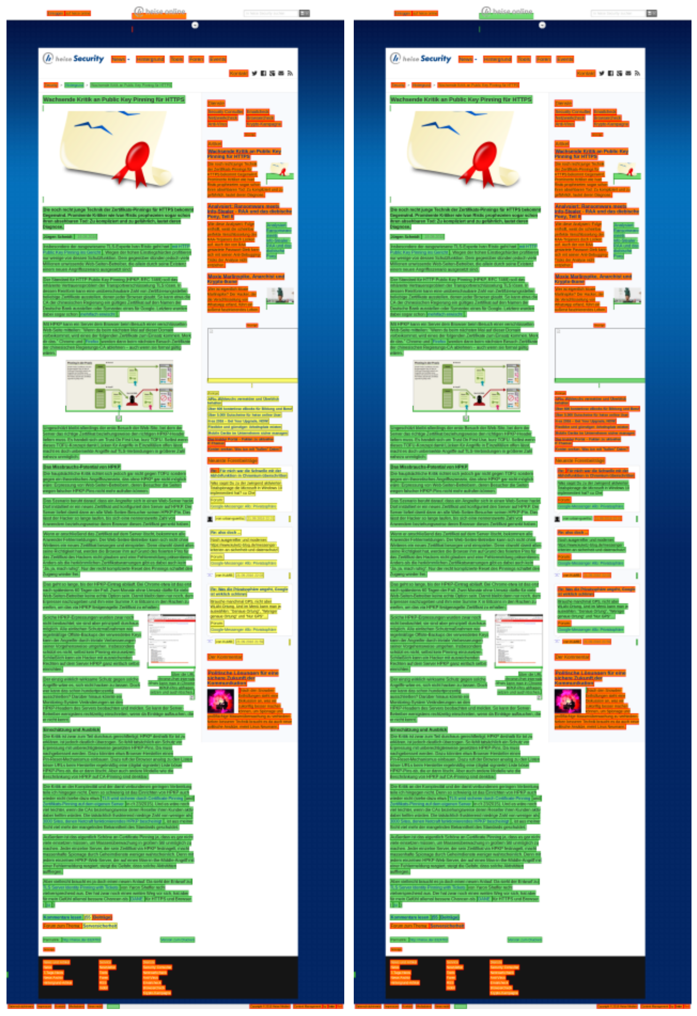


Figure C.1.: Document uuid: a3c71aa6-cca0-488c-971c-2c12c746e1ed. On the left bit-pattern based annotation and on the right result of the T006.

# D. Precision, Recall and F1-Score Measurements

Table D.1.: Measurements for www.chefkoch.de

| docid | testname | tp | fn | fp | tn | acc | p_bp | p_con | r_bp | r_con | f1_bp | f1_con |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 014ff911-194d-4c81-9ee0-b9a5928d66c6 | T001 | 112.0 | 16.0 | 107.0 | 478.0 | 0.827489 | 0.97 | 0.51 | 0.82 | 0.88 | 0.89 | 0.65 |
| 014ff911-194d-4c81-9ee0-b9a5928d66c6 | T002 | 112.0 | 17.0 | 107.0 | 512.0 | 0.834225 | 0.97 | 0.51 | 0.83 | 0.87 | 0.89 | 0.64 |
| 014ff911-194d-4c81-9ee0-b9a5928d66c6 | T003 | 43.0 | 85.0 | 45.0 | 540.0 | 0.817672 | 0.86 | 0.49 | 0.92 | 0.34 | 0.89 | 0.40 |
| 014ff911-194d-4c81-9ee0-b9a5928d66c6 | T004 | 84.0 | 44.0 | 80.0 | 505.0 | 0.826087 | 0.92 | 0.51 | 0.86 | 0.66 | 0.89 | 0.58 |
| 014ff911-194d-4c81-9ee0-b9a5928d66c6 | T005 | 96.0 | 32.0 | 85.0 | 500.0 | 0.835905 | 0.94 | 0.53 | 0.85 | 0.75 | 0.90 | 0.62 |
| 014ff911-194d-4c81-9ee0-b9a5928d66c6 | T006 | 99.0 | 29.0 | 90.0 | 495.0 | 0.833100 | 0.94 | 0.52 | 0.85 | 0.77 | 0.89 | 0.62 |
| 0e480386-a02f-4d28-aabc-9442ae2840b8 | T001 | 82.0 | 18.0 | 116.0 | 514.0 | 0.816438 | 0.97 | 0.41 | 0.82 | 0.82 | 0.88 | 0.55 |
| 0e480386-a02f-4d28-aabc-9442ae2840b8 | T002 | 82.0 | 18.0 | 116.0 | 514.0 | 0.816438 | 0.97 | 0.41 | 0.82 | 0.82 | 0.88 | 0.55 |
| 0e480386-a02f-4d28-aabc-9442ae2840b8 | T003 | 43.0 | 57.0 | 69.0 | 561.0 | 0.827397 | 0.91 | 0.38 | 0.89 | 0.43 | 0.90 | 0.41 |
| 0e480386-a02f-4d28-aabc-9442ae2840b8 | T004 | 68.0 | 32.0 | 89.0 | 541.0 | 0.834247 | 0.94 | 0.43 | 0.86 | 0.68 | 0.90 | 0.53 |
| 0e480386-a02f-4d28-aabc-9442ae2840b8 | T005 | 72.0 | 28.0 | 108.0 | 522.0 | 0.813699 | 0.95 | 0.40 | 0.83 | 0.72 | 0.88 | 0.51 |
| 0e480386-a02f-4d28-aabc-9442ae2840b8 | T006 | 79.0 | 21.0 | 77.0 | 553.0 | 0.865753 | 0.96 | 0.51 | 0.88 | 0.79 | 0.92 | 0.62 |
| 7860509f-9400-4f5e-8786-58fb4fb00f17 | T001 | 111.0 | 13.0 | 105.0 | 424.0 | 0.819296 | 0.97 | 0.51 | 0.80 | 0.90 | 0.88 | 0.65 |
| 7860509f-9400-4f5e-8786-58fb4fb00f17 | T002 | 111.0 | 14.0 | 105.0 | 642.0 | 0.863532 | 0.98 | 0.51 | 0.86 | 0.89 | 0.92 | 0.65 |
| 7860509f-9400-4f5e-8786-58fb4fb00f17 | T003 | 43.0 | 81.0 | 34.0 | 495.0 | 0.823890 | 0.86 | 0.56 | 0.94 | 0.35 | 0.90 | 0.43 |
| 7860509f-9400-4f5e-8786-58fb4fb00f17 | T004 | 78.0 | 46.0 | 95.0 | 434.0 | 0.784074 | 0.90 | 0.45 | 0.82 | 0.63 | 0.86 | 0.53 |
| 7860509f-9400-4f5e-8786-58fb4fb00f17 | T005 | 91.0 | 33.0 | 96.0 | 433.0 | 0.802450 | 0.93 | 0.49 | 0.82 | 0.73 | 0.87 | 0.59 |
| 7860509f-9400-4f5e-8786-58fb4fb00f17 | T006 | 108.0 | 16.0 | 97.0 | 432.0 | 0.826953 | 0.96 | 0.53 | 0.82 | 0.87 | 0.88 | 0.66 |
| a9799eb9-5a0b-45a8-a236-905547cbef3c | T001 | 82.0 | 15.0 | 67.0 | 433.0 | 0.862647 | 0.97 | 0.55 | 0.87 | 0.85 | 0.91 | 0.67 |
| a9799eb9-5a0b-45a8-a236-905547cbef3c | T002 | 82.0 | 16.0 | 67.0 | 741.0 | 0.908389 | 0.98 | 0.55 | 0.92 | 0.84 | 0.95 | 0.66 |
| a9799eb9-5a0b-45a8-a236-905547cbef3c | T003 | 32.0 | 65.0 | 22.0 | 478.0 | 0.854271 | 0.88 | 0.59 | 0.96 | 0.33 | 0.92 | 0.42 |
| a9799eb9-5a0b-45a8-a236-905547cbef3c | T004 | 68.0 | 29.0 | 64.0 | 436.0 | 0.844221 | 0.94 | 0.52 | 0.87 | 0.70 | 0.90 | 0.59 |
| a9799eb9-5a0b-45a8-a236-905547cbef3c | T005 | 72.0 | 25.0 | 60.0 | 440.0 | 0.857621 | 0.95 | 0.55 | 0.88 | 0.74 | 0.91 | 0.63 |
| a9799eb9-5a0b-45a8-a236-905547cbef3c | T006 | 80.0 | 17.0 | 39.0 | 461.0 | 0.906198 | 0.96 | 0.67 | 0.92 | 0.82 | 0.94 | 0.74 |

Table D.2.: Measurements for www.heise.de

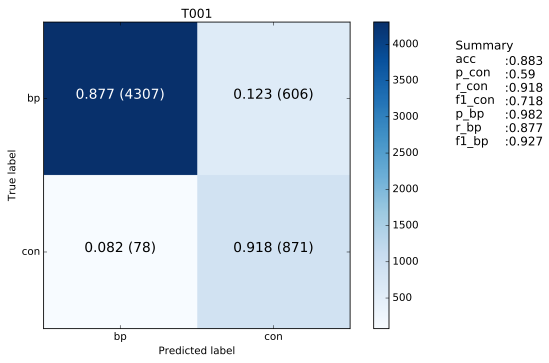
| docid | testname | tp | fn | fp | tn | acc | p_bp | p_con | r_bp | r_con | f1_bp | f1_con |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 47346f56-5d34-4fcb-b175-18e2bb9d864d | T001 | 43.0 | 0.0 | 37.0 | 118.0 | 0.813131 | 1.00 | 0.54 | 0.76 | 1.00 | 0.86 | 0.70 |
| 47346f56-5d34-4fcb-b175-18e2bb9d864d | T002 | 43.0 | 0.0 | 37.0 | 123.0 | 0.817734 | 1.00 | 0.54 | 0.77 | 1.00 | 0.87 | 0.70 |
| 47346f56-5d34-4fcb-b175-18e2bb9d864d | T003 | 27.0 | 16.0 | 22.0 | 133.0 | 0.808081 | 0.89 | 0.55 | 0.86 | 0.63 | 0.88 | 0.59 |
| 47346f56-5d34-4fcb-b175-18e2bb9d864d | T004 | 39.0 | 4.0 | 21.0 | 134.0 | 0.873737 | 0.97 | 0.65 | 0.86 | 0.91 | 0.91 | 0.76 |
| 47346f56-5d34-4fcb-b175-18e2bb9d864d | T005 | 39.0 | 4.0 | 24.0 | 131.0 | 0.858586 | 0.97 | 0.62 | 0.85 | 0.91 | 0.90 | 0.74 |
| 47346f56-5d34-4fcb-b175-18e2bb9d864d | T006 | 39.0 | 4.0 | 35.0 | 120.0 | 0.803030 | 0.97 | 0.53 | 0.77 | 0.91 | 0.86 | 0.67 |
| 4929bee0-0df6-45e4-b1cf-35b8df45a1db | T001 | 30.0 | 0.0 | 17.0 | 129.0 | 0.903409 | 1.00 | 0.64 | 0.88 | 1.00 | 0.94 | 0.78 |
| 4929bee0-0df6-45e4-b1cf-35b8df45a1db | T002 | 30.0 | 0.0 | 17.0 | 162.0 | 0.918660 | 1.00 | 0.64 | 0.91 | 1.00 | 0.95 | 0.78 |
| 4929bee0-0df6-45e4-b1cf-35b8df45a1db | T003 | 13.0 | 17.0 | 33.0 | 113.0 | 0.715909 | 0.87 | 0.28 | 0.77 | 0.43 | 0.82 | 0.34 |
| 4929bee0-0df6-45e4-b1cf-35b8df45a1db | T004 | 18.0 | 12.0 | 12.0 | 134.0 | 0.863636 | 0.92 | 0.60 | 0.92 | 0.60 | 0.92 | 0.60 |
| 4929bee0-0df6-45e4-b1cf-35b8df45a1db | T005 | 21.0 | 9.0 | 10.0 | 136.0 | 0.892045 | 0.94 | 0.68 | 0.93 | 0.70 | 0.93 | 0.69 |
| 4929bee0-0df6-45e4-b1cf-35b8df45a1db | T006 | 29.0 | 1.0 | 16.0 | 130.0 | 0.903409 | 0.99 | 0.64 | 0.89 | 0.97 | 0.94 | 0.77 |
| 49cd031f-77e0-4d61-9b39-b1f968b99b09 | T001 | 30.0 | 0.0 | 17.0 | 131.0 | 0.904494 | 1.00 | 0.64 | 0.89 | 1.00 | 0.94 | 0.78 |
| 49cd031f-77e0-4d61-9b39-b1f968b99b09 | T002 | 30.0 | 0.0 | 17.0 | 162.0 | 0.918660 | 1.00 | 0.64 | 0.91 | 1.00 | 0.95 | 0.78 |
| 49cd031f-77e0-4d61-9b39-b1f968b99b09 | T003 | 12.0 | 18.0 | 33.0 | 115.0 | 0.713483 | 0.86 | 0.27 | 0.78 | 0.40 | 0.82 | 0.32 |
| 49cd031f-77e0-4d61-9b39-b1f968b99b09 | T004 | 18.0 | 12.0 | 13.0 | 135.0 | 0.859551 | 0.92 | 0.58 | 0.91 | 0.60 | 0.92 | 0.59 |
| 49cd031f-77e0-4d61-9b39-b1f968b99b09 | T005 | 21.0 | 9.0 | 14.0 | 134.0 | 0.870787 | 0.94 | 0.60 | 0.91 | 0.70 | 0.92 | 0.65 |
| 49cd031f-77e0-4d61-9b39-b1f968b99b09 | T006 | 29.0 | 1.0 | 14.0 | 134.0 | 0.915730 | 0.99 | 0.67 | 0.91 | 0.97 | 0.95 | 0.79 |
| a3c71aa6-cca0-488c-971c-2c12c746e1ed | T001 | 43.0 | 0.0 | 12.0 | 118.0 | 0.930636 | 1.00 | 0.78 | 0.91 | 1.00 | 0.95 | 0.88 |
| a3c71aa6-cca0-488c-971c-2c12c746e1ed | T002 | 43.0 | 0.0 | 12.0 | 149.0 | 0.941176 | 1.00 | 0.78 | 0.93 | 1.00 | 0.96 | 0.88 |
| a3c71aa6-cca0-488c-971c-2c12c746e1ed | T003 | 22.0 | 21.0 | 4.0 | 126.0 | 0.855491 | 0.86 | 0.85 | 0.97 | 0.51 | 0.91 | 0.64 |
| a3c71aa6-cca0-488c-971c-2c12c746e1ed | T004 | 36.0 | 7.0 | 8.0 | 122.0 | 0.913295 | 0.95 | 0.82 | 0.94 | 0.84 | 0.94 | 0.83 |
| a3c71aa6-cca0-488c-971c-2c12c746e1ed | T005 | 36.0 | 7.0 | 5.0 | 125.0 | 0.930636 | 0.95 | 0.88 | 0.96 | 0.84 | 0.95 | 0.86 |
| a3c71aa6-cca0-488c-971c-2c12c746e1ed | T006 | 41.0 | 2.0 | 15.0 | 115.0 | 0.901734 | 0.98 | 0.73 | 0.88 | 0.95 | 0.93 | 0.83 |

Table D.3.: Measurements for www.spiegel.de

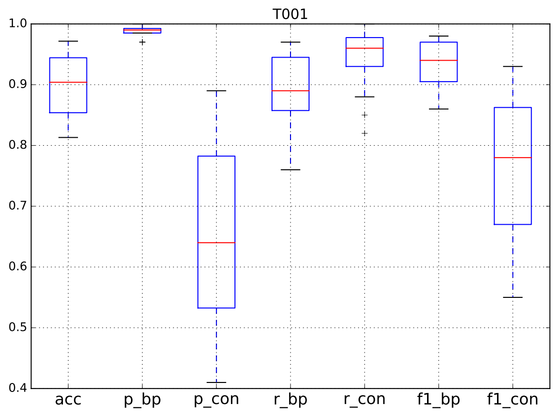
| docid | testname | tp | fn | fp | tn | acc | p_bp | p_con | r_bp | r_con | f1_bp | f1_con |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 3b071332-86a3-4b8d-9a80-e60090e27eb3 | T001 | 47.0 | 2.0 | 21.0 | 324.0 | 0.941624 | 0.99 | 0.69 | 0.94 | 0.96 | 0.97 | 0.80 |
| 3b071332-86a3-4b8d-9a80-e60090e27eb3 | T002 | 47.0 | 2.0 | 21.0 | 324.0 | 0.941624 | 0.99 | 0.69 | 0.94 | 0.96 | 0.97 | 0.80 |
| 3b071332-86a3-4b8d-9a80-e60090e27eb3 | T003 | 25.0 | 24.0 | 73.0 | 272.0 | 0.753807 | 0.92 | 0.26 | 0.79 | 0.51 | 0.85 | 0.34 |
| 3b071332-86a3-4b8d-9a80-e60090e27eb3 | T004 | 42.0 | 7.0 | 36.0 | 309.0 | 0.890863 | 0.98 | 0.54 | 0.90 | 0.86 | 0.93 | 0.66 |
| 3b071332-86a3-4b8d-9a80-e60090e27eb3 | T005 | 42.0 | 7.0 | 34.0 | 311.0 | 0.895939 | 0.98 | 0.55 | 0.90 | 0.86 | 0.94 | 0.67 |
| 3b071332-86a3-4b8d-9a80-e60090e27eb3 | T006 | 46.0 | 3.0 | 27.0 | 318.0 | 0.923858 | 0.99 | 0.63 | 0.92 | 0.94 | 0.95 | 0.75 |
| 5dcac983-cf64-4d60-a83c-0aa13ac36ab7 | T001 | 58.0 | 3.0 | 15.0 | 323.0 | 0.954887 | 0.99 | 0.79 | 0.96 | 0.95 | 0.97 | 0.87 |
| 5dcac983-cf64-4d60-a83c-0aa13ac36ab7 | T002 | 58.0 | 3.0 | 15.0 | 323.0 | 0.954887 | 0.99 | 0.79 | 0.96 | 0.95 | 0.97 | 0.87 |
| 5dcac983-cf64-4d60-a83c-0aa13ac36ab7 | T003 | 28.0 | 33.0 | 51.0 | 287.0 | 0.789474 | 0.90 | 0.35 | 0.85 | 0.46 | 0.87 | 0.40 |
| 5dcac983-cf64-4d60-a83c-0aa13ac36ab7 | T004 | 50.0 | 11.0 | 20.0 | 318.0 | 0.922306 | 0.97 | 0.71 | 0.94 | 0.82 | 0.95 | 0.76 |
| 5dcac983-cf64-4d60-a83c-0aa13ac36ab7 | T005 | 49.0 | 12.0 | 17.0 | 321.0 | 0.927318 | 0.96 | 0.74 | 0.95 | 0.80 | 0.96 | 0.77 |
| 5dcac983-cf64-4d60-a83c-0aa13ac36ab7 | T006 | 51.0 | 10.0 | 11.0 | 327.0 | 0.947368 | 0.97 | 0.82 | 0.97 | 0.84 | 0.97 | 0.83 |
| 957ae5dc-edb1-4f0e-bacd-e01599a36fea | T001 | 58.0 | 4.0 | 15.0 | 322.0 | 0.952381 | 0.99 | 0.79 | 0.96 | 0.94 | 0.97 | 0.86 |
| 957ae5dc-edb1-4f0e-bacd-e01599a36fea | T002 | 58.0 | 4.0 | 15.0 | 322.0 | 0.952381 | 0.99 | 0.79 | 0.96 | 0.94 | 0.97 | 0.86 |
| 957ae5dc-edb1-4f0e-bacd-e01599a36fea | T003 | 28.0 | 34.0 | 51.0 | 286.0 | 0.786967 | 0.89 | 0.35 | 0.85 | 0.45 | 0.87 | 0.40 |
| 957ae5dc-edb1-4f0e-bacd-e01599a36fea | T004 | 50.0 | 12.0 | 20.0 | 317.0 | 0.919799 | 0.96 | 0.71 | 0.94 | 0.81 | 0.95 | 0.76 |
| 957ae5dc-edb1-4f0e-bacd-e01599a36fea | T005 | 49.0 | 13.0 | 17.0 | 320.0 | 0.924812 | 0.96 | 0.74 | 0.95 | 0.79 | 0.96 | 0.77 |
| 957ae5dc-edb1-4f0e-bacd-e01599a36fea | T006 | 53.0 | 9.0 | 13.0 | 324.0 | 0.944862 | 0.97 | 0.80 | 0.96 | 0.85 | 0.97 | 0.83 |
| c1b6bfcd-9b7a-452d-a57a-887928284ecf | T001 | 47.0 | 3.0 | 21.0 | 323.0 | 0.939086 | 0.99 | 0.69 | 0.94 | 0.94 | 0.96 | 0.80 |
| c1b6bfcd-9b7a-452d-a57a-887928284ecf | T002 | 47.0 | 3.0 | 21.0 | 323.0 | 0.939086 | 0.99 | 0.69 | 0.94 | 0.94 | 0.96 | 0.80 |
| c1b6bfcd-9b7a-452d-a57a-887928284ecf | T003 | 25.0 | 25.0 | 73.0 | 271.0 | 0.751269 | 0.92 | 0.26 | 0.79 | 0.50 | 0.85 | 0.34 |
| c1b6bfcd-9b7a-452d-a57a-887928284ecf | T004 | 42.0 | 8.0 | 36.0 | 308.0 | 0.888325 | 0.97 | 0.54 | 0.90 | 0.84 | 0.93 | 0.66 |
| c1b6bfcd-9b7a-452d-a57a-887928284ecf | T005 | 42.0 | 8.0 | 34.0 | 310.0 | 0.893401 | 0.97 | 0.55 | 0.90 | 0.84 | 0.94 | 0.67 |
| c1b6bfcd-9b7a-452d-a57a-887928284ecf | T006 | 45.0 | 5.0 | 25.0 | 319.0 | 0.923858 | 0.98 | 0.64 | 0.93 | 0.90 | 0.96 | 0.75 |

Table D.4.: Measurements for www.wired.com

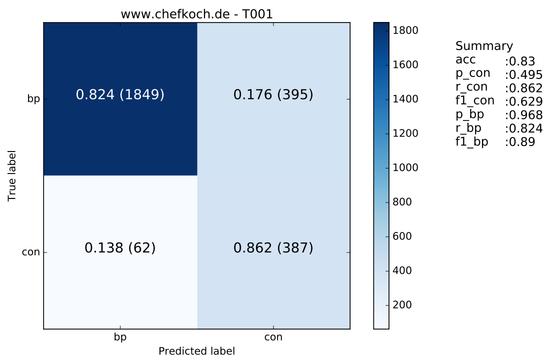
| docid | testname | tp | fn | fp | tn | acc | p_bp | p_con | r_bp | r_con | f1_bp | f1_con |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 03808d91-4f61-4471-bcd1-81f835323138 | T001 | 25.0 | 1.0 | 24.0 | 168.0 | 0.885321 | 0.99 | 0.51 | 0.88 | 0.96 | 0.93 | 0.67 |
| 03808d91-4f61-4471-bcd1-81f835323138 | T002 | 25.0 | 3.0 | 24.0 | 245.0 | 0.909091 | 0.99 | 0.51 | 0.91 | 0.89 | 0.95 | 0.65 |
| 03808d91-4f61-4471-bcd1-81f835323138 | T003 | 14.0 | 12.0 | 1.0 | 191.0 | 0.940367 | 0.94 | 0.93 | 0.99 | 0.54 | 0.97 | 0.68 |
| 03808d91-4f61-4471-bcd1-81f835323138 | T004 | 23.0 | 3.0 | 13.0 | 179.0 | 0.926606 | 0.98 | 0.64 | 0.93 | 0.88 | 0.96 | 0.74 |
| 03808d91-4f61-4471-bcd1-81f835323138 | T005 | 22.0 | 4.0 | 7.0 | 185.0 | 0.949541 | 0.98 | 0.76 | 0.96 | 0.85 | 0.97 | 0.80 |
| 03808d91-4f61-4471-bcd1-81f835323138 | T006 | 22.0 | 4.0 | 12.0 | 180.0 | 0.926606 | 0.98 | 0.65 | 0.94 | 0.85 | 0.96 | 0.73 |
| 6e2b623c-ba6d-4396-b4ec-0c1d1c699fa0 | T001 | 25.0 | 1.0 | 21.0 | 171.0 | 0.899083 | 0.99 | 0.54 | 0.89 | 0.96 | 0.94 | 0.69 |
| 6e2b623c-ba6d-4396-b4ec-0c1d1c699fa0 | T002 | 25.0 | 3.0 | 21.0 | 251.0 | 0.920000 | 0.99 | 0.54 | 0.92 | 0.89 | 0.95 | 0.68 |
| 6e2b623c-ba6d-4396-b4ec-0c1d1c699fa0 | T003 | 14.0 | 12.0 | 1.0 | 191.0 | 0.940367 | 0.94 | 0.93 | 0.99 | 0.54 | 0.97 | 0.68 |
| 6e2b623c-ba6d-4396-b4ec-0c1d1c699fa0 | T004 | 22.0 | 4.0 | 14.0 | 178.0 | 0.917431 | 0.98 | 0.61 | 0.93 | 0.85 | 0.95 | 0.71 |
| 6e2b623c-ba6d-4396-b4ec-0c1d1c699fa0 | T005 | 21.0 | 5.0 | 8.0 | 184.0 | 0.940367 | 0.97 | 0.72 | 0.96 | 0.81 | 0.97 | 0.76 |
| 6e2b623c-ba6d-4396-b4ec-0c1d1c699fa0 | T006 | 22.0 | 4.0 | 9.0 | 183.0 | 0.940367 | 0.98 | 0.71 | 0.95 | 0.85 | 0.97 | 0.77 |
| 8c742c8d-5b22-432e-84e8-f586be537690 | T001 | 39.0 | 1.0 | 5.0 | 166.0 | 0.971564 | 0.99 | 0.89 | 0.97 | 0.97 | 0.98 | 0.93 |
| 8c742c8d-5b22-432e-84e8-f586be537690 | T002 | 39.0 | 2.0 | 5.0 | 251.0 | 0.976431 | 0.99 | 0.89 | 0.98 | 0.95 | 0.99 | 0.92 |
| 8c742c8d-5b22-432e-84e8-f586be537690 | T003 | 21.0 | 19.0 | 4.0 | 167.0 | 0.890995 | 0.90 | 0.84 | 0.98 | 0.53 | 0.94 | 0.65 |
| 8c742c8d-5b22-432e-84e8-f586be537690 | T004 | 32.0 | 8.0 | 11.0 | 160.0 | 0.909953 | 0.95 | 0.74 | 0.94 | 0.80 | 0.94 | 0.77 |
| 8c742c8d-5b22-432e-84e8-f586be537690 | T005 | 32.0 | 8.0 | 7.0 | 164.0 | 0.928910 | 0.95 | 0.82 | 0.96 | 0.80 | 0.96 | 0.81 |
| 8c742c8d-5b22-432e-84e8-f586be537690 | T006 | 37.0 | 3.0 | 0.0 | 171.0 | 0.985782 | 0.98 | 1.00 | 1.00 | 0.93 | 0.99 | 0.96 |
| a8d7c494-b428-4cf9-8055-7b04382bdca7 | T001 | 39.0 | 1.0 | 6.0 | 165.0 | 0.966825 | 0.99 | 0.87 | 0.96 | 0.97 | 0.98 | 0.92 |
| a8d7c494-b428-4cf9-8055-7b04382bdca7 | T002 | 39.0 | 2.0 | 6.0 | 247.0 | 0.972789 | 0.99 | 0.87 | 0.98 | 0.95 | 0.98 | 0.91 |
| a8d7c494-b428-4cf9-8055-7b04382bdca7 | T003 | 21.0 | 19.0 | 5.0 | 166.0 | 0.886256 | 0.90 | 0.81 | 0.97 | 0.53 | 0.93 | 0.64 |
| a8d7c494-b428-4cf9-8055-7b04382bdca7 | T004 | 32.0 | 8.0 | 11.0 | 160.0 | 0.909953 | 0.95 | 0.74 | 0.94 | 0.80 | 0.94 | 0.77 |
| a8d7c494-b428-4cf9-8055-7b04382bdca7 | T005 | 32.0 | 8.0 | 8.0 | 163.0 | 0.924171 | 0.95 | 0.80 | 0.95 | 0.80 | 0.95 | 0.80 |
| a8d7c494-b428-4cf9-8055-7b04382bdca7 | T006 | 37.0 | 3.0 | 0.0 | 171.0 | 0.985782 | 0.98 | 1.00 | 1.00 | 0.93 | 0.99 | 0.96 |



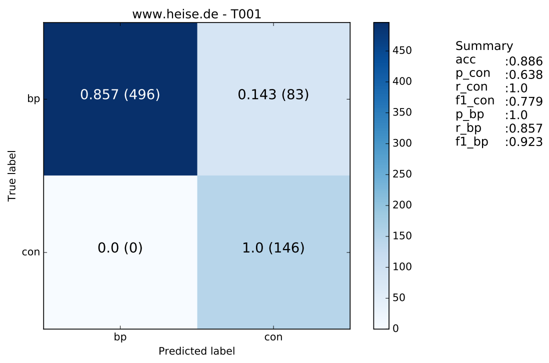
(a) Aggregate Confusion Matrix



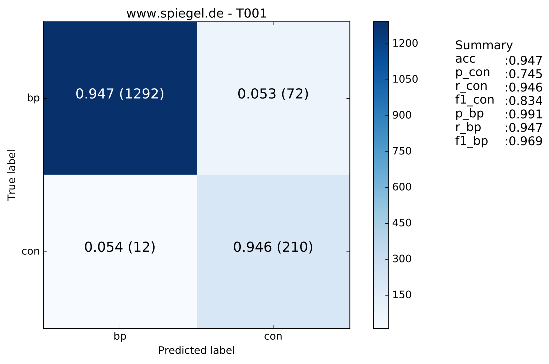
(b) Aggregate Performance



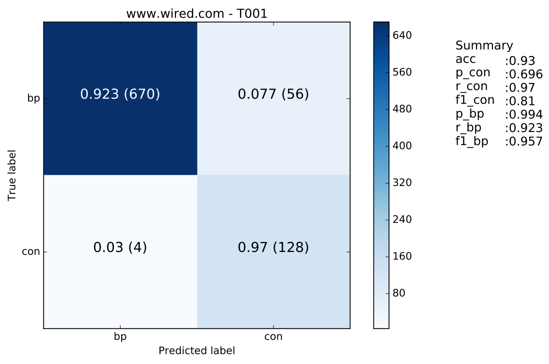
(c) Confusion Matrix chefkoch.de



(d) Confusion Matrix heise.de

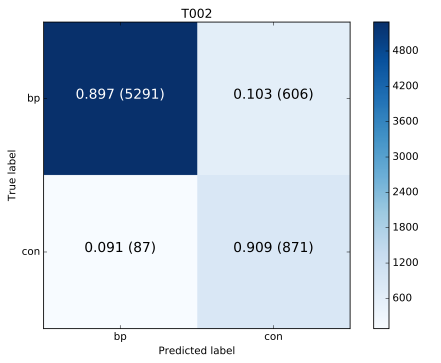


(e) Confusion Matrix spiegel.de



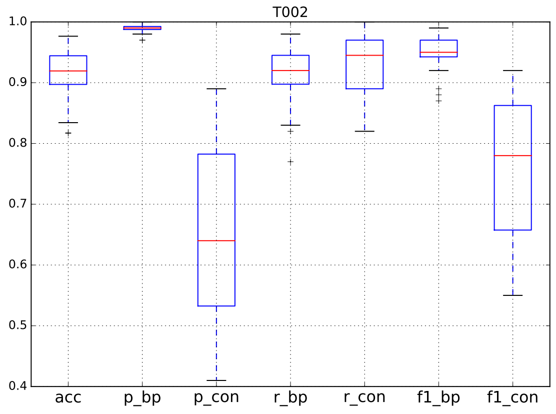
(f) Confusion Matrix wired.com

Figure D.1.: Overview of classification results for the bit-pattern based classification (*T001*).

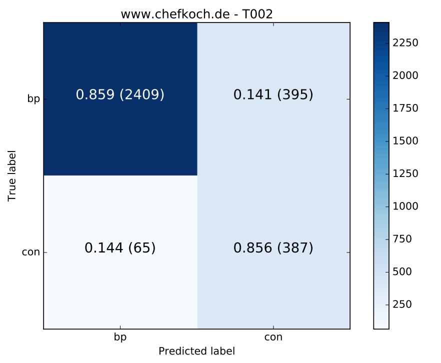


```
Summary
acc    :0.899
p_con  :0.59
r_con  :0.909
f1_con :0.716
p_bp   :0.984
r_bp   :0.897
f1_bp  :0.938
```

(a) Aggregate Confusion Matrix

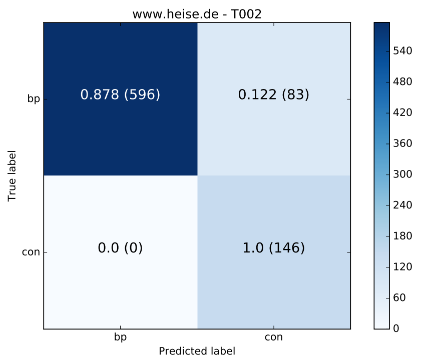


(b) Aggregate Performance



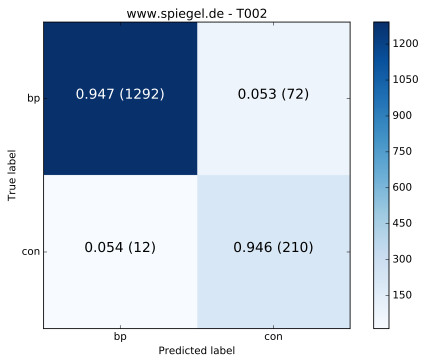
```
Summary
acc    :0.859
p_con  :0.495
r_con  :0.856
f1_con :0.627
p_bp   :0.974
r_bp   :0.859
f1_bp  :0.913
```

(c) Confusion Matrix chefkoch.de



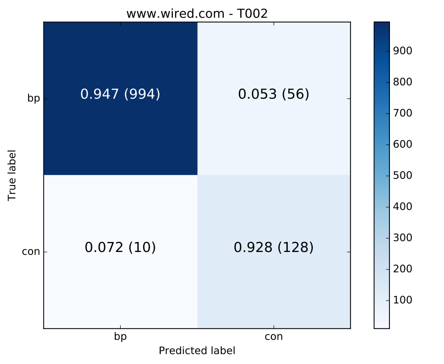
```
Summary
acc    :0.899
p_con  :0.638
r_con  :1.0
f1_con :0.779
p_bp   :1.0
r_bp   :0.878
f1_bp  :0.935
```

(d) Confusion Matrix heise.de



```
Summary
acc    :0.947
p_con  :0.745
r_con  :0.946
f1_con :0.834
p_bp   :0.991
r_bp   :0.947
f1_bp  :0.969
```

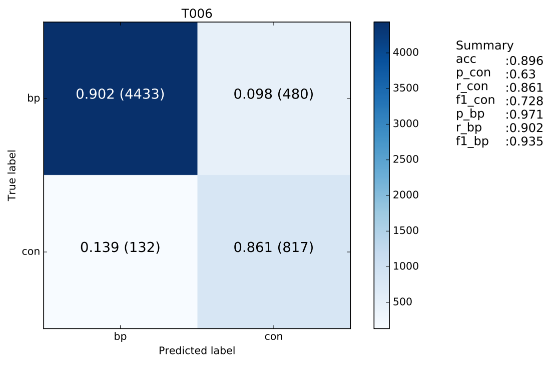
(e) Confusion Matrix spiegel.de



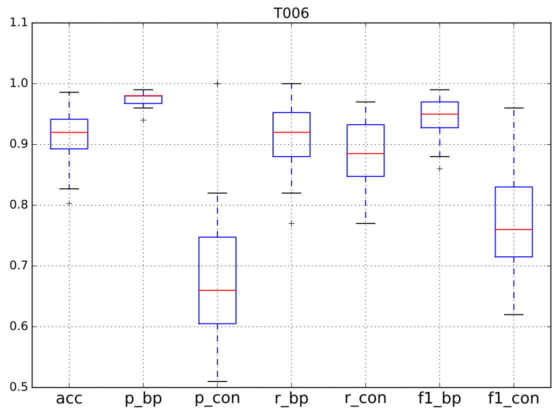
```
Summary
acc    :0.944
p_con  :0.696
r_con  :0.928
f1_con :0.795
p_bp   :0.99
r_bp   :0.947
f1_bp  :0.968
```

(f) Confusion Matrix wired.com

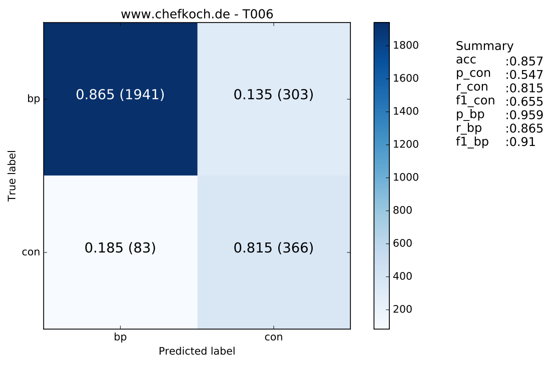
Figure D.2.: Overview of classification results for the extended version of the bit-pattern based classification method (*T002*).



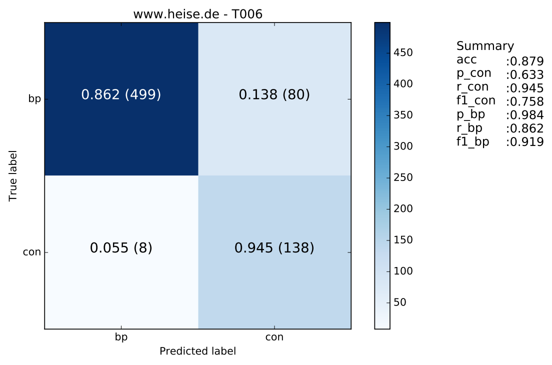
(a) Aggregate Confusion Matrix



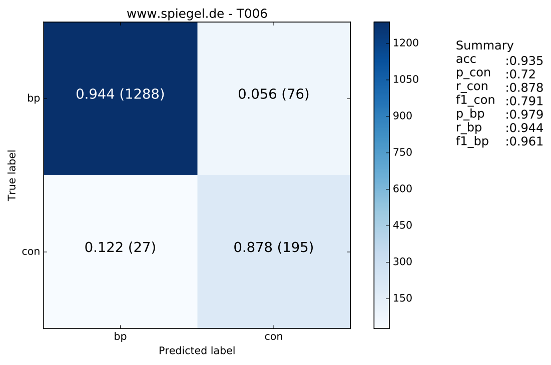
(b) Aggregate Performance



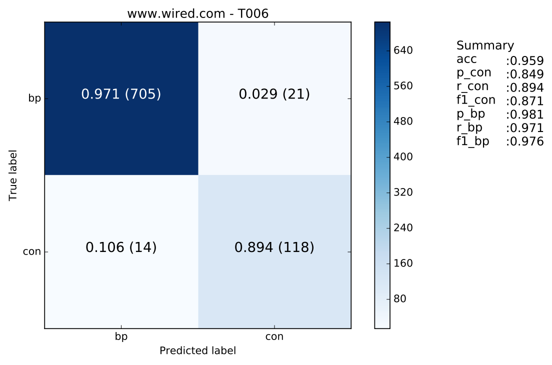
(c) Confusion Matrix chefkoch.de



(d) Confusion Matrix heise.de



(e) Confusion Matrix spiegel.de



(f) Confusion Matrix wired.com

Figure D.3.: Overview of classification results for the re-classification method based on random forests (*T006*).

# E. Python Modules and Tools used in DBCE Framework

- beautifulsoup4-4.5.1 - for easy DOM Tree traversal and DOM modification
- datasketch-0.2.5 - MinHash and LSH implementation
- lxml-3.6.4 - Annotation of HTML documents
- matplotlib-1.5.3 - representation of the result
- numpy-1.11.2 - dependency for scipy and pandas
- pandas-0.19.1 - used for result evaluation
- pywb-0.33.0 - access of the WARC and CDX files
- selenium-3.0.1 - for manual annotation of the ground truth
- scikit-learn-0.18.1 - machine learning
- scipy-0.18.1 - open source library of scientific tools

# Erklärung der Selbstständigkeit

Hiermit versichere ich, dass ich die vorliegende Bachelorarbeit selbständig und ohne fremde Hilfe verfasst und keine anderen als die in der Arbeit angegebenen Quellen und Hilfsmittel verwendet habe. Die Arbeit hat in gleicher oder ähnlicher Form noch keinem anderen Prüfungsamt vorgelegen.

Hannover, den 21.01.2017

________________________________
Rafał Leśniak