

BACHELOR THESIS  
COMPUTING SCIENCE



RADBOUD UNIVERSITY

---

# Web classification using DMOZ

---

*Author:*  
Lisa Hoek  
s1009553

*First supervisor/assessor:*  
Prof.dr.ir, Arjen P. de Vries  
a.devries@cs.ru.nl

*Second assessor:*  
Prof.dr.ir, Djoerd Hiemstra  
hiemstra@cs.ru.nl

January 17, 2021

## **Abstract**

Web classification is the process of assigning a Web page to a category out of a pre-defined set of labels. For a classifier to predict well, it needs a proper training data set to learn how to classify. The DMOZ directory is very large Web taxonomy consisting of Web pages and a certain topic given to each URL. Classification on Web taxonomies has to address a number of challenges that make it a non-trivial problem: a large number of different categories, a deep hierarchy and class imbalance including many sparsely populated categories. This research creates a dataset derived from the DMOZ directory, providing a proper dataset for future research in Web classification, and conducts an experiment with different selections of labels, showing the performance and difficulties that arise in the process of Web classification.

*Keywords: Web classification, categorisation, DMOZ, Open Directory Project, Common Crawl*

# Contents

<b>1</b>	<b>Introduction</b>	<b>3</b>
<b>2</b>	<b>Preliminaries</b>	<b>6</b>
2.1	DMOZ data . . . . .	6
2.2	Common Crawl . . . . .	6
2.3	Spark . . . . .	8
2.4	Classification . . . . .	10
2.5	Natural language processing . . . . .	11
<b>3</b>	<b>Research</b>	<b>13</b>
3.1	Data pre-processing . . . . .	13
3.1.1	Pre-processing DMOZ data . . . . .	13
3.1.2	Pre-processing Common Crawl URL index . . . . .	14
3.1.3	Executing join . . . . .	14
3.1.4	Getting Web content . . . . .	15
3.1.5	Cleaning Web content . . . . .	16
3.2	Classification . . . . .	17
3.2.1	Preparing different datasets for classification . . . . .	17
3.2.2	Training the classifier . . . . .	19
<b>4</b>	<b>Results</b>	<b>20</b>
4.1	Classifier I - 13 categories . . . . .	20
4.2	Classifier II - 277 categories . . . . .	22
4.3	Classifier III - 30 categories (more balanced) . . . . .	23
<b>5</b>	<b>Conclusions</b>	<b>26</b>
<b>6</b>	<b>Future Work</b>	<b>28</b>
6.1	Expanding data . . . . .	28
6.2	Improving cleaned content of the Web pages . . . . .	29
6.3	Improving the classifier . . . . .	30
6.4	Personalised Web search . . . . .	30
<b>A</b>	<b>Evaluation scores</b>	<b>33</b>

<b>B</b>	<b>Confusion matrices</b>	<b>36</b>
<b>C</b>	<b>Code examples</b>	<b>40</b>

# Chapter 1

## Introduction

Classification is a widely used machine learning method. However, for Web classification, there are more challenges compared to normal text classification, making it a complex area in which research is not that far. Before a classifier can make predictions (regarding their goal), one needs a proper dataset to train the classification model with. Only then, it could become relevant to use the built classifier for its or other applications. Currently, Web classification is not that feasible to implement as a stand-alone feature in large search engines as a feature like PageRank in Google Search.

Web classification, however, is useful for several tasks. It can not only improve query results in a search engine, but it also assists development of Web directories, can help filtering Web content, and is essential to focused crawlers or vertical (domain-specific) search engines [17]. In case of the latter, with the internet divided in categories, one could take a subset with a topic of personal interest and store this on their own computer, creating opportunity for an offline search engine.

The effectiveness of good classification algorithms such as logistic regression has not been thoroughly investigated on very large Web taxonomies [15], for example the DMOZ directory [12]. Classification on Web taxonomies has to address a number of challenges that make it a non-trivial problem: a large number of different categories, a deep hierarchy and class imbalance including many sparsely populated categories. This makes applying classification algorithms to such datasets with thousands of categories very difficult; sparsely populated categories do not have a lot of data to learn from, and on the other hand, a few considerably larger categories will dominate the classification, i.e., most data get predicted as these classes.

In this research, we create a dataset consisting of 514,506 Web pages divided over 13 maincategories, 451 second-level categories and 110,860 different categories in total, derived from the DMOZ directory in combination with data from the Common Crawl to retrieve the Web page's content. By

doing this, we provide a proper dataset (with a topic and cleaned plain text for each Web page), which can be used for further research in Web content classification.

Also, we conduct an experiment asking ourselves: *“How do the topics of the DMOZ dataset perform as labels for a single-label Web classifier?”*. We show that a classifier using the 13 maincategories performs quite well, but the more classes are added, the more difficult it gets for the classifier to predict correctly, which is in line with the difficulties described in [15]. Already for the 13 maincategories, class imbalance has a huge impact on the performance of the classifier.

A related study proposing how to deal with the sparsity of the DMOZ data is conducted by [10]. They explored different data augmentation techniques on the DMOZ data and showed that it significantly improved the classification performance, more than traditional classifiers. Other research [1] used a classifier from DMOZ data to show that their approach (using social annotation for Web directory extension) worked better than just classifying on content. [19] used the DMOZ categories as a variable in their framework for personalised Web search. They used a text-based classifier trained with logistic regression to obtain classes for each document in the index. That way, they improved retrieval performance for one-word queries with high ambiguity.

Bennett, Svore and Dumais [3] used the DMOZ categories as classification space to classify query results. Together with the usage of click data from user logs, they presented a framework that enhances the ranking of search engines. It builds on the assumption that user clicks not only serve as relevance signal for the clicked URL, but also for other URLs belonging to that same category. So, they trained a classifier with logistic regression using the whole content of the Web pages and 219 second-level labels (the ones that had over one thousand DMOZ Web pages). This model was then applied on real-world Web-scale test data. For each page, they compared the URL and Query class distribution and calculated several features. These features were then stored during indexing time of the collection. They were the first using offline classification during indexing for a large-scale search engine. The ranker with their features significantly improved the search engine’s results over a competitive baseline.

Bennett et al. showed classification of Web data can be used to derive ranking features which are compact enough to still work efficiently in large-scale Web search engines. They did not, however, use the URL class itself as a feature in ranking. This is not yet used in search engines because it requires high precision of its classifier. Also, knowledge about which labels or which multi-level ordering to use for the classes in Web classification is needed and still lacking, up to this point.

The remainder of this thesis is structured as follows. Chapter 2 gives a basic understanding of the sources and tools we have used. Chapter 3 describes the pre-processing phase (what we have done to end up with our final dataset) and the classification process. The evaluation of the classifiers is dealt with in Chapter 4. The conclusions are stated in Chapter 5 and in Chapter 6, improvements and ideas for future research are given. Appendices A, B and C contain the evaluation scores per class for each classifier, the confusion matrices, and some important code examples that were used in this research, respectively.

## Chapter 2

# Preliminaries

### 2.1 DMOZ data

DMOZ is a directory of World Wide Web links, actively maintained from 1998 till 2017. Links to Web pages contain a certain topic built in a hierarchical way. Top levels are, for example, ‘Business’ and ‘Sports’ and deeper in the directory exist lower levels with a path like ‘Science/Technology/Space’ and ‘Recreation/Outdoors/Fishing’.

The community who voluntarily constructed and maintained DMOZ was also known as the Open Directory Project (ODP). Currently, they have continued independent from Mozilla under the name of Curlie [5] and are working on a successor version of the DMOZ directory. As of 2020, it does not seem that they will be publishing any renewed data anytime soon, just as the old data due to technical issues. To obtain the XML file containing DMOZ’s old RDF dump, one must use the Wayback Machine [2] to go to the latest available dump around the 12th of march, 2017. `content.rdf.u8` is the DMOZ data consisting of ~4 million Web pages. Overall, each entry consists of a topic, title, description, and the URL of the Web page. Figure 2.1 shows a code snippet of the file.

### 2.2 Common Crawl

The Common Crawl [9] is a non-profit organisation that crawls the Web. Their Web crawler visits billions of pages and each month, it stores its raw crawl data into Web ARChive (WARC) format. This consists of information how the data was requested, metadata on the crawl process itself and the HTTP response from the websites it contacts.

WARC files contain many WARC records (one record corresponds to one contact the crawler makes with a Web page). So, if one wants to obtain the content of a specific page, one should include which bits to read from the WARC file. This is done by including the WARC record’s offset and length



```

<?xml version="1.0" encoding="UTF-8"?>
<RDF xmlns:r="http://www.w3.org/TR/RDF/" xmlns:d="http://purl.org/dc/elements/1.0/" xmlns="http://dmoz.org/rdf/">
  <!-- Generated at 2017-03-12 00:03:03 EST from DMOZ 2.0 -->
  <Topic r:id="">
    <catid>1</catid>
  </Topic>
  <Topic r:id="Top/Arts">
    <catid>381773</catid>
  </Topic>
  <Topic r:id="Top/Arts/Animation">
    <catid>423945</catid>
    <link1 r:resource="http://www.awn.com/"></link1>
    <link r:resource="http://animation.about.com/"></link>
    <link r:resource="http://www.toonhound.com/"></link>
    <link r:resource="http://www.digitalmediafx.com/Features/animationhistory.html"></link>
    <link r:resource="http://www.animated-divots.net/"></link>
  </Topic>
  <ExternalPage about="http://www.awn.com/">
    <d:title>Animation World Network</d:title>
    <d:description>Provides information resources to the international animation community. Features include searchable database archives, monthly magazine, web animation guide, the Animation Village, discussion forums and other useful resources.</d:description>
    <priority>1</priority>
  </ExternalPage>
  <ExternalPage about="http://animation.about.com/">
    <d:title>About.com: Animation Guide</d:title>
    <d:description>Keep up with developments in online animation for all skill levels. Download tools, and seek inspiration from online work.</d:description>
  </ExternalPage>
  <ExternalPage about="http://www.toonhound.com/">
    <d:title>Toonhound</d:title>
    <d:description>British cartoon, animation and comic strip creations - links, reviews and news from the UK.</d:description>
  </ExternalPage>
  <ExternalPage about="http://www.digitalmediafx.com/Features/animationhistory.html">
    <d:title>Digital Media FX: The History of Animation</d:title>
    <d:description>Michael Crandol takes an exhaustive look at the history of animation and animators/ visionaries like Max Fleisher, Walter Lantz, and Otto Messmer.</d:description>
  </ExternalPage>
  <ExternalPage about="http://www.animated-divots.net/">
    <d:title>Richard's Animated Divots</d:title>
    <d:description>Chronology of animated movies, television programs, and short cartoons. Includes animation filmographies and a list of anime television series.</d:description>
  </ExternalPage>

```

Figure 2.1: Start of DMOZ's RDF dump content.rdf.u8

to the HTTP request to the Common Crawl's server. For example, one can use the Python package Requests and use the code below to retrieve one single WARC record as shown in Figure 2.2.

```

url = "https://commoncrawl.s3.amazonaws.com/" + warc_filename
response = requests.get(url, headers={'Range': 'bytes={}-{}'.format(offset, offset+length-1)})

```

The Common Crawl also provides WAT and WET files which store computed metadata and extracted plain text from the Web data, respectively. The latter removes the need of converting HTML content to plain text yourself, which is quite handy as many tasks only require textual information. However, the Common Crawl does not include offsets in WAT and WET files as it does to each WARC record, so one cannot easily retrieve the WET plain text of one specific Web page without first creating additional data structures. For convenience, this thesis has used the WARC files with their offsets. Consequently, we did the HTML cleaning ourselves.

```

WARC/1.0
WARC-Type: response
WARC-Date: 2014-08-02T09:52:13Z
WARC-Record-ID:
Content-Length: 43428
Content-Type: application/http; msgtype=response
WARC-Warcinfo-ID:
WARC-Concurrent-To:
WARC-IP-Address: 212.58.244.61
WARC-Target-URI: http://news.bbc.co.uk/2/hi/africa/3414345.stm
WARC-Payload-Digest: sha1:M63W6MNGFDWXDSLTHF7GWUPCJUH4JK3J
WARC-Block-Digest: sha1:YHKQUSBOS4CLYFEKQDVGJ4570APD6IJO
WARC-Truncated: length

HTTP/1.1 200 OK
Server: Apache
Vary: X-CDN
Cache-Control: max-age=0
Content-Type: text/html
Date: Sat, 02 Aug 2014 09:52:13 GMT
Expires: Sat, 02 Aug 2014 09:52:13 GMT
Connection: close
Set-Cookie: BBC-UID=...; expires=Sun, 02-Aug-15 09:52:13 GMT; path=/; domain=bbc.co.uk;

<!doctype html public "-//W3C//DTD HTML 4.0 Transitional//EN" "http://www.w3.org/TR/REC-html40/loose.dtd">
<html>
<head>
<title>
    BBC NEWS | Africa | Namibia braces for Nujoma exit
</title>
...

```

Figure 2.2: Example response of a WARC HTTP request, retrieved from <https://commoncrawl.org/the-data/get-started/>

## 2.3 Spark

When, due to big data tasks, working on a single computer is not feasible anymore, one can choose to process data on a cluster. This makes dividing work over several computers possible. These worker nodes are independent of each other and instructed by their one master node.

Apache Spark [8] is one of the frameworks for distributed data processing. It provides an interface and four components on top of the Spark Core's framework. (Spark SQL, MLlib, Spark Streaming and GraphX). Its default language is Scala, but also Java, Python and R are supported. Using Spark on a cluster requires a cluster manager (e.g., Yarn) and a distributed storage system like Hadoop Distributed File System (HDFS). To easily explore and

analyse data when working on the cluster, Apache Zeppelin is an interactive, Web-based notebook for writing and executing code.

The general idea behind Spark is its Resilient Distributed Datasets (RDDs). An RDD is a read-only, immutable object of data items, distributed over multiple computers in the cluster. When the RDD is not partitioned over multiple nodes, there are no tasks to execute in parallel. Good partitions are of real importance. Efficient handling of these partitions is managed by avoiding intermediate shuffles between them and by distributing the data wisely.

There exist two types of operators to manipulate data from RDDs. Transformations make new RDDs out of the previous one, for example a *map()*-function but also a *groupByKey()* or *join()*. Actions return a result like *count()* or *take(n)*. Transformations have lazy evaluation, so they are only performed when they are followed by an action. If some part of a job has already been performed, it can also re-use the already existing cached RDDs corresponding to that same task.

Over the years, Spark has grown into a complete eco-system consisting of different interfaces, to help users work with the rather low-level abstraction of the RDD. An example that we use extensively in this thesis is Spark SQL, a Spark component that introduced the data abstraction DataFrames. One can make a dataframe out of an RDD and perform operations on it. It even facilitates using SQL as a language once a so-called (global) temporary view is created. One can apply transformations and actions on data in the dataframe, but also create their own user-defined functions (UDFs) which will then be applied to each row of the dataframe.

Another component of the Spark eco-system that has been deployed in this thesis are the MLlib libraries, that provide scalable versions of common machine learning algorithms, like the Logistic Regression classifier.

Pyspark [21] is the Python API for Spark. It is not only useful if you are already familiar with Python and prefer it over Scala, but also enables using Python packages on the cluster. Python code for a single computer can also be easily transformed into something in Pyspark that works large-scale on a cluster.

The standard format to save data on the file system is in Parquet files because it preserves the partitions. Other formats like CSV are also supported and can be transformed into a dataframe but are slower to use. Efficient data storage and distribution is of importance for fast executions. Partitions are often made based on a certain key feature like for employee data their working department. This way, queries within a certain partition (department) can be executed faster, because it does not have to look at the other partitions. If partitioning the data results in uneven partition sizes, due to skew in the value distribution in the original data source, it is also better to distribute the data into so-called ‘buckets’, where the system will apply hashing to create these approximately equi-sized groups. This optimises the

worker nodes because the amount of processing for each bucket becomes about the same.

## 2.4 Classification

Classification is the process in which items are given a certain label based on its features. In the case of text classification, one gives as input a piece of text and the classifier predicts (out of the given categories) which category suits best, for example a category like ‘Sports’ for a text from a sports magazine. Web classification resembles the simpler problem of text classification, but the content of Web pages is often messy and needs a lot of cleaning before it is transformed into a useful text snippet. For example, HTML consists of tags which need to be removed. Several toolkits are available for this, like BeautifulSoup [18] for Python. After cleaning the Web content, often some natural language processing is done as well (see next section).

When all input data is pre-processed, some part of it is kept separate as test data (often between 20% and 40%) and the rest is used as training data for the classifier. A classifier can be built using a certain Machine Learning algorithm. There exist many different algorithms, the one used in this thesis is the (multinomial) logistic regression classifier. Details about this kind of classifier can be found here [20].

A classifier is trained by giving it the feature vectors and labels of the training data. A feature vector contains all information about that certain data item. After transforming the training data, the model can be applied to the test data as well and make predictions based on their feature vectors. Because the test data also has an original label, it can be compared to the predicted label. The accuracy is the percentage (or factor between 0 and 1) of correctly classified items in the test data.

A good evaluation method is the confusion matrix. This shows for all data items the predicted label versus the true label. In the case of binary classification, True Positives (TP) are all labels correctly classified as True; True Negatives (TN) are all labels correctly classified as False, False Positives (FP) are all incorrectly classified as True; and False Negatives (FN) are all labels incorrectly classified as False. In the case of multinomial classification, it is kind of similar except all classes have their own TP, TN, FP and FN. For example, the FP of class A are all items classified as A while they belong to a different class and FN are all items from A classified as a different class.

With these four values, measurements like precision, recall and F1 measure can be calculated. (Equation 2.1). Precision is the number of correct items compared to all items predicted as this class. Recall is the number of correct items compared to the number that truly belong to that class. F1 is a measurement that shows the combination of the two.

$$P = \frac{T_P}{T_P + F_P} \quad R = \frac{T_P}{T_P + F_N} \quad F1 = 2 \frac{P * R}{P + R} \quad (2.1)$$

Accuracy but also precision, recall and F1 are not a good evaluation method when classes are imbalanced. For example, for fraud detection only a few percent consists of fraud and if the classifier just classifies every data item as no fraud, the accuracy is almost 100% while the classifier did not detect a single fraud item. Better measurements are done when looking at each individual class. This introduces the measurements weighted precision, weighted recall and weighted F1, (equation 2.2, 2.3 and 2.4 respectively).

$$PPV_w = \frac{1}{N} \sum_{l \in L} PPV(l) * \sum_{i=0}^{N-1} \delta^\#(y_i - l) \quad (2.2)$$

$$TPR_w = \frac{1}{N} \sum_{l \in L} TPR(l) * \sum_{i=0}^{N-1} \delta^\#(y_i - l) \quad (2.3)$$

$$F_w(\beta) = \frac{1}{N} \sum_{l \in L} F(\beta, l) * \sum_{i=0}^{N-1} \delta^\#(y_i - l) \quad (2.4)$$

Here,  $L$  is defined as the set of classes.  $y$  as the prediction vector and  $\delta^\#$  is a modified delta function  $\delta(x)$  which is 1 if  $x = 0$  and 0 otherwise.

Improving the classification model can be done by, for example, tuning the hyper-parameters of the algorithm. Due to time constraints, this is not included in this thesis.

## 2.5 Natural language processing

Natural language processing is the field which concerns how human readable text should be processed by machines. NLTK [16] is the most used natural language processing toolkit for Python. Relevant features are Word segmentation (Tokenization), Stop word removal, Lemmatization and Stemming.

When tokenizing, a chunk of text is split up in separate words, this is for most languages of course done by splitting on space, so the string of text is transformed into an array of words.

Stop words are frequently used words that often do not carry additional meaning, such as ‘the’, ‘a’, ‘for’, ‘but’. Filtering out this useless data saves a lot of space and processing time later. For this reason, Stop word removal is one of the standard procedures for text-mining applications.

Lemmatization is the task of changing each word back to its certain base form if possible. This way, plural forms return to their singular value and verbs like ‘walked’ and ‘walks’ are all changed to their base form ‘walk’.

Lemmatisation is done by looking up the lemmas (base forms) in a dictionary.

Word stemming is a process which looks like lemmatisation, but root forms do not necessarily need to be an existing word. An example is the root form of ‘trouble’, which is ‘troubl’. This way, more words will merge into the same root. For example, ‘computed’, ‘computing’, but also ‘computer’ and ‘computational’ stem down to the root ‘comput’. There are two error measurements in stemming: understemming and overstemming. The latter is the case when words like ‘computer’ and ‘computational’ get stemmed down to the same root, as in the previous example, even though one would have liked to keep them separate.

Thus, for natural language processing there is no general best method, and it varies which steps to include and how strict to be in those.

# Chapter 3

## Research

### 3.1 Data pre-processing

This section describes the process that was made to obtain the Cleaned Common Crawl Content (CCCC) in a dataframe together with its category labels (retrieved from DMOZ). Briefly said, the DMOZ data containing URLs and topics needs to be joined with the Common Crawl data consisting of URLs and WARC information. After joining, the WARC information is used to obtain the HTML content for each Web page. Lastly, the content is cleaned and a dataframe remains consisting of *maincategory*, *categories*, *url*, *title* and *cleaned\_content*. This dataframe (stored as CCCC parquet table) is used for training and testing the classifier.

#### 3.1.1 Pre-processing DMOZ data

To begin with, the DMOZ data is transformed into a dataframe. We used Python 3.6 in a Google Colab notebook for this. To get our hands on the data, we read the data as done in “URL classification using DMOZ data” by Utsav [22] and built a mini-classifier with Sklearn based on the description text and topics.

Then, we moved on to the actual pre-processing. We decided to read the data in a slightly different way than suggested by [22]. We started off by reading the content of `content.rdf.u8` and splitting it on `</ExternalPage>` (instead of newlines). As can be seen in the file in Figure 2.1, splitting on `</ExternalPage>` will make sure the ‘title’, ‘description’ and ‘topic’ of a URL stay together in one string with the URL.

Then, the ‘url’, ‘title’ and ‘topic’ are extracted from each snippet. This is done with regular expressions. In case the snippet is lacking a URL, title or topic, it will not be added to the dataframe. In case there were multiple URLs, titles or topics in the snippet, the first one is taken.

The dataframe is formed by taking the first part of the topic as *maincategory*, the whole topic as *categories*, the *url* and *title*. This resulted in

a dataframe of 3,573,026 entries (against Utsav’s 3,529,057 entries). This difference can be explained by looking at Figure 2.1 again. The first URL contains a title, description, priority and topic. Utsav’s processing skipped this snippet, because of the ‘priority’ disrupting the standard format title-description-topic.

The maincategory ‘World’ is dropped because it contains non-English Web pages. In this thesis, we only focus on building an English text classifier. Also, the category ‘Regional’ is dropped. This category splits on regions first, something that does not belong to the rest of the maincategories. The classifier will learn to predict topics like sport, science, etc. – not regions. The size of the dataframe, after dropping these two categories, is 979,845 records.

The Python code used for the DMOZ processing can be found in Appendix C, code example 1. Figure 3.1 shows the current head of the dataframe.

	maincategory	categories	url	title
0	Arts	Top/Arts/Animation	http://www.awn.com/	Animation World Network
1	Arts	Top/Arts/Animation	http://animation.about.com/	About.com: Animation Guide
2	Arts	Top/Arts/Animation	http://www.toonhound.com/	Toonhound
3	Arts	Top/Arts/Animation	http://www.digitalmediafx.com/Features/animati...	Digital Media FX: The History of Animation
4	Arts	Top/Arts/Animation	http://www.animated-divots.net/	Richard's Animated Divots

Figure 3.1: First 5 records of the DMOZ dataframe

### 3.1.2 Pre-processing Common Crawl URL index

Now, we have moved on to working in Zeppelin with Spark 2.4.4 on an Amazon EMR 6.0.0 cluster. The Common Crawl’s CC-INDEX table that resides on their AWS S3 cluster is in our Zeppelin notebook represented by a dataframe, to combine conveniently with the DMOZ data. We have used both CC-MAIN-2017-13 (March 2017) which is most in line with the date of the DMOZ data and the crawl CC-MAIN-2017-04 (January 2017) which is a bit earlier. From these dataframes, all columns are dropped, except for *url*, *warc\_filename*, *warc\_record\_offset* and *warc\_record\_length*. Both dataframes contain ~3 billion Web pages.

### 3.1.3 Executing join

The JOIN-operator is used to align the two datasets, matching on URLs that are equal in both datasets. Executing this is simple with an SQL query like this:

```
val joinDF = spark.sql("SELECT maincategory, categories, title,
c.url, warc_filename, warc_record_offset, warc_record_length
```



```
FROM global_temp.dmoz d INNER JOIN global_temp.cc c WHERE d.url
== c.url").cache()
```

It performs an inner join on the two global temporary views that are made from the DMOZ and CC dataframes. In Appendix C, code example 2 shows the general process how the DMOZ and CC dataframes are created, the join is performed, and the joined data is stored in the file system.

With the use of the Common Crawl data from January 2017, the join resulted in 575,745 entries with a fine execution time of half an hour (and only ~15 minutes when scaling up to ten worker nodes). The joined dataset is around half the size of the original dataset. Unfortunately, there existed also duplicate rows, because only 438,445 entries were left after taking all unique URLs.

The goal is to end up with as large a dataframe as possible, i.e., the size of the original DMOZ dataframe (979,845 records), because we are trying to link the WARC information to (preferably) all DMOZ Web pages. To increase the number of matched entries, we decided to look at crawl CC-MAIN-2020-45 (October 2020) and crawl CC-MAIN-2017-13 (March 2017) as well. The latest version of the Common Crawl data (October 2020) resulted in only 63,056 matches, confirming our beliefs it is better to use Common Crawl data from around the same time as the DMOZ data. The joined dataframe with March 2017 data consisted of 589,022 matches and after removing the duplicates 400,307 entries. This is about the same size as the set of January 2017.

Hoping there were many distinct URLs, we then took (after matching with DMOZ) the union of the two dataframes from January and March 2017. Luckily, this increased the final dataframe to 514,506 Web pages. Figure 3.2 shows the head of the current dataframe.

### 3.1.4 Getting Web content

In the next step, the four columns to the right of the dataframe in Figure 3.2 are used to extract the HTML content from the WARC file. With an HTTP request to the Common Crawl server, one can obtain the HTML content that was fetched and put in the WARC file. A request consists of the URL `"https://commoncrawl.s3.amazonaws.com/" + warc_filename` and can be given certain headers. For this research, we give the range of bytes to read as a header (which are the `warc_record_offset` and `warc_record_offset+length-1`).

The returned data is decompressed using Python library `zlib`, decoded based on 'utf-8' and read by Python package `BeautifulSoup` to parse the HTML and convert it to plain text only, using function `'get_text'`. Now, we end up with a string consisting of three parts: information about how the information was requested, metadata and the cleaned HTML content. Only

maincategory	categories	title	url	warc_filename warc_record_offset warc_record_length
	Society Top/Society/Histo...	23rd North Caroli...	http://6thnjvol.h...	crawl-data/CC-MAI...  3222345  4853
	Science Top/Science/Math/...	AAECC-16	http://aaecc.free...	crawl-data/CC-MAI...  4593198  1257
	Business Top/Business/Publ...	Association of Ar...	http://aaronline...	crawl-data/CC-MAI...  4196817  10776
	Sports Top/Sports/Events...	ABC News: Summer ...	http://abcnews.go...	crawl-data/CC-MAI...  5520785  12465
	Arts Top/Arts/Literatu...	Miwu	http://absurdism...	crawl-data/CC-MAI...  7158904  1311
	Society Top/Society/Relig...	Addyston Baptist ...	http://addystonba...	crawl-data/CC-MAI...  8342299  5940
	Reference Top/Reference/Edu...	Administrators.Net	http://administra...	crawl-data/CC-MAI...  7269974  10457
	Arts Top/Arts/Literatu...	Akilan: Tamil aut...	http://akilan6.tr...	crawl-data/CC-MAI...  12857033  8263
	Society Top/Society/Histo...	Schon, Alan Morris	http://alanschon...	crawl-data/CC-MAI...  7851843  2701
	Health Top/Health/Medici...	Alaska Regional H...	http://alaskaregi...	crawl-data/CC-MAI...  8750178  33807
	Arts Top/Arts/Literatu...	The Astrid Lindgr...	http://alassociat...	crawl-data/CC-MAI...  14449004  6733
	Recreation Top/Recreation/Du...	Albany Hunting	http://albanytexa...	crawl-data/CC-MAI...  14461904  10473
	Sports Top/Sports/Cricke...	Aldenhams Cricket ...	http://aldenhamcr...	crawl-data/CC-MAI...  13344353  23321
	Recreation Top/Recreation/Fo...	Ed's Ale Haus	http://alehaus.ed...	crawl-data/CC-MAI...  9170278  2313
	Health Top/Health/Altern...	Aligned Coaching	http://alignedcoa...	crawl-data/CC-MAI...  12401431  2896
	Arts Top/Arts/Movies/T...	Allreaders Review...	http://allreaders...	crawl-data/CC-MAI...  10948176  26290
	Arts Top/Arts/Movies/T...	Allreaders Review...	http://allreaders...	crawl-data/CC-MAI...  11443671  26203
	Arts Top/Arts/Movies/T...	Allreaders.com Re...	http://allreaders...	crawl-data/CC-MAI...  14999392  24538
	Arts Top/Arts/Movies/T...	Allreaders.com: S...	http://allreaders...	crawl-data/CC-MAI...  12973935  24429
	Arts Top/Arts/Movies/T...	Allreaders.com: T...	http://allreaders...	crawl-data/CC-MAI...  11683455  25374

only showing top 20 rows

Figure 3.2: First 20 records after joining DMOZ and Common Crawl

the last one is of our interest and kept as column *content* in the dataframe.

This code was first tested on a small sample outside the cluster. In Appendix C, code example 3 shows the Python function. Once on the cluster, we hit some obstacles. The Python package Requests (used to send the HTTP request to the Common Crawl server), was not installed by default on the cluster's driver. Additionally, installing a library on the driver does not mean the worker nodes are able to use it. For this, the packages need to be passed along to the worker nodes. A guide to do this can be found here [6].

Once all content was collected, we wrote the result to the file system for possible later use. The now redundant columns *warc\_filename*, *warc\_record\_offset* and *warc\_record\_length* are dropped and the dataframe is saved as parquet table CCC. We specified to partition by the data's main-categories and bucket by its categories to improve further SQL queries on the categories.

### 3.1.5 Cleaning Web content

Before using the *content* column as input for the classifier, it should be cleaned. We used a Python function written by Idil Ismiguzel [14] for this and slightly adjusted it.

First, all characters are converted to lower case and the white spaces are stripped from the content. Then, the text is searched for contractions, which are replaced by their longer forms if present. The list of English contractions from Wikipedia is used here [13]. Then, all unwanted characters or words are removed as good as possible with regular expressions. For example, Unicodes like the non-breaking space '\xa0' are replaced by a normal space

‘ ’ with the regular expression:

```
text = re.sub(r`[\^ \x00-\x7F]+'`, ` `, text)
```

Stopwords are removed from the text with Python package NLTK. It has many more features of which we use the WordPunctTokenizer and the WordNetLemmatizer. As last step, we join the array of words back to a full string, because this is nicer to give as input to the classifier.

In Appendix C, code example 4 shows the process described above. On the cluster, we put this function in a user-defined function (UDF) and performed it on the *content* column of the dataframe, creating a new column called *cleaned\_content*. Column *content* is dropped. Again, the dataframe is stored as parquet table as described in the previous section, but now under the name CCCC.

## 3.2 Classification

### 3.2.1 Preparing different datasets for classification

In this research, we decided to build three different classifiers (instead of one elaborate classifier), since we wanted to focus on the distribution of categories. Classifier I labels on *maincategory*. Classifier II uses all second-level *categories* such as ‘Arts/Movies’ instead of ‘Arts’. (Categories with less than 100 Web pages are discarded). And the last classifier has clever distributed labels to make the data less imbalanced: from the three largest categories (Arts, Business and Society), its largest sub-categories are taken out and used as a separate category. For example, ‘Arts’ is split up in ‘Arts/Music’, ‘Arts/Movies’, ‘Arts/Literature’, ‘Arts/Performing Arts’ and ‘Arts/Visual Arts’ and the rest of the data remains in class ‘Arts’. Where to stop splitting can be automated but is done by human judgement in this thesis.

Figure 3.3 shows the number of Web pages for each maincategory. The first three classes are considerably larger than the others. Figure 3.4 shows the distribution of labels when the second level is taken (and the ones with less than 100 Web pages removed). The class imbalance increased even more compared to the first-level categories. Figure 3.5 is the third dataset with a better distribution. The y-axis is set the same as in Figure 3.3 to show we reduced class imbalance.

The code how we created the second and third dataset can be found in the beginning of the classifiers’ Notebooks [11].

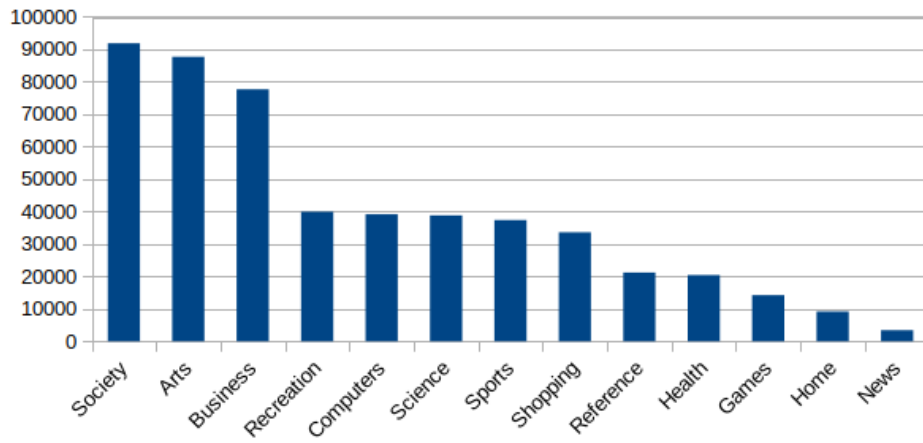


Figure 3.3: Nr of Web pages per category, classifier I

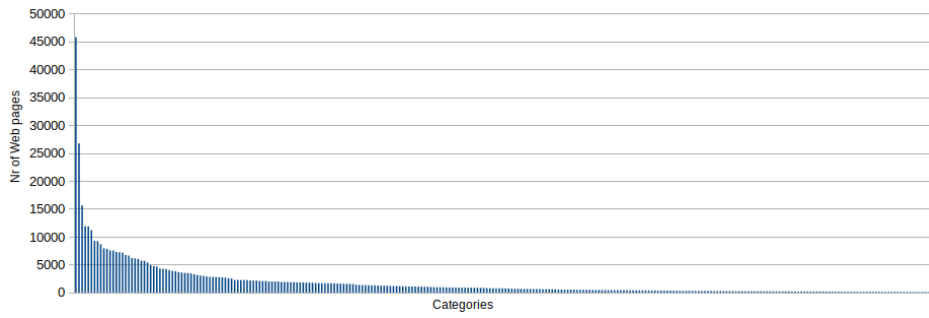


Figure 3.4: Nr of Web pages per category, classifier II

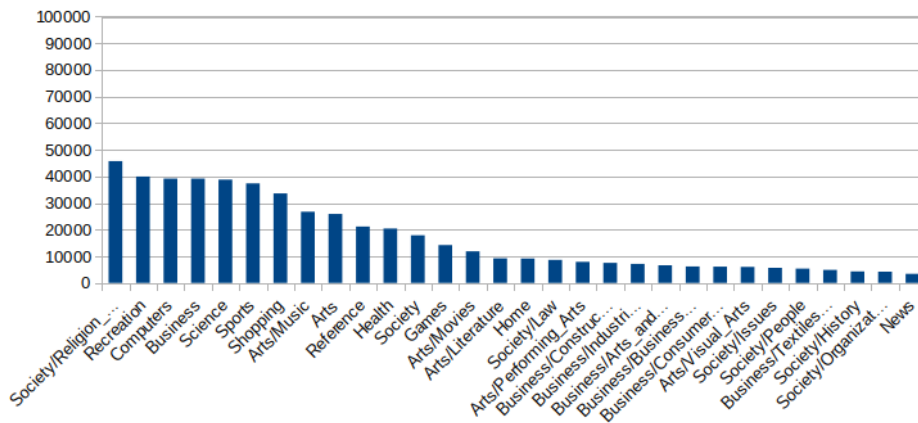


Figure 3.5: Nr of Web pages per category, classifier III

### 3.2.2 Training the classifier

For each dataset, the following procedure is followed using Spark MLlib.

First of all, a classifier cannot be trained with strings as labels, so the `StringIndexer` is used to transform the *maincategory* into integer labels. Then, a random split is made which divides the data into 75% training and 25% testing data.

A pipeline is created which consist of the stages ‘tokenizer’, ‘hashingTF’ and ‘lr’. The first one uses the `Tokenizer` and transforms the *cleaned\_content* into an array of words. Tokenizing in an earlier phase of the process (when the *cleaned\_content* was made) was not possible, because the next transformer `HashingTF` did not accept our own tokenized array. The `HashingTF` transforms the output of the tokenizer into feature vectors. These are the term frequencies of the words. The last stage of the pipeline uses the `LogisticRegression` module to train and apply a logistic regression classifier. When this pipeline is fitted on the training data, it takes the columns *label* and *features* and starts training the model. Due to time constraints, we opted to use default hyper-parameters as is and defer their optimisation to later work.

Training a classifier with a lot of data and many labels, requires a lot of memory on the cluster. When training classifier II, we hit the obstacle of not having enough memory available. We fixed this by claiming 32 GB on the driver. Though, this is not advisable when multiple people are using the same cluster.

Once the model is applied to the training data, we can predict labels by transforming the test data. We compare the *prediction* column with their original label. To convert the integer labels from *prediction* back to their corresponding category name, we use the `IndexToString` transformer.

Evaluation is done by instantiating a metrics object which requires the columns *prediction* and *label*. It can calculate all sorts of evaluation statistics. We also print the confusion matrix. These values are then used in a Python program [11] to plot the confusion matrix with Seaborn heatmap [23] and Matplotlib.pyplot [7].

In Appendix C, code example 5 shows the general approach that was used for training, testing, and evaluating the three classifiers. For the detailed code, we refer to the three Zeppelin Notebooks on our Github page [11].

## Chapter 4

# Results

This chapter shows the evaluation results of the three classifiers. The accuracy, weighted precision, weighted recall and weighted F1 measure are calculated. Unfortunately, Spark MLlib cannot compute the AUROC for a multi-class problem, but we do show the confusion matrices for classifier I and III, and the normalised confusion matrices for all three classifiers. Normalisation is done by dividing each entry by the number of true label Web pages of that category.

### 4.1 Classifier I - 13 categories

The first classifier categorises Web pages in the following 13 categories: ‘Society’, ‘Arts’, ‘Business’, ‘Recreation’, ‘Computers’, ‘Science’, ‘Sports’, ‘Shopping’, ‘Reference’, ‘Health’, ‘Games’, ‘Home’ and ‘News’.

Figure 4.1 shows a random sample of the test data. On average, 11 out of 20 are classified correctly.

Table 4.1 shows the overall statistics. Accuracy on its own does not tell us a lot how good the classifier performs. The weighted measures normalise the data such that the amount of data per class is taken into account. More than half of the test data is classified correctly, 54%. Although clearly the performance is much better than random choosing one of the 13 labels, it is unclear if the effectiveness is already good enough for practical applications, as for example, a feature in search engines.

Accuracy	0.54
Weighted precision	0.58
Weighted recall	0.54
Weighted F(1) Score	0.53

Table 4.1: Evaluation scores Classifier I

maincategory	url	words	predicted_category	correct
Computers	http://www.the-sz...	[royal, read, r, ...]	Computers	true
Society	http://ndpr.nd.ed...	[window, nreum, n...]	Society	true
Sports	http://www.thurle...	[thurles, golf, c...]	Arts	false
Sports	http://redwoodfal...	[redwood, fall, g...]	Reference	false
Arts	http://www.imdb.c...	[var, ue, t0, win...]	Arts	true
Reference	http://www.hangar...	[function, id, {, ...]	Recreation	false
Science	http://www.ucmp.b...	[introduction, sc...]	Science	true
Shopping	http://www.finess...	[finesse, pinstri...]	Recreation	false
Business	http://www.mcleod...	[trucking, softwa...]	Business	true
Computers	http://www.newmil...	[new, millennium, ...]	Computers	true
News	http://www.newsqu...	[wonderpluginslid...]	Sports	false
Recreation	http://webpace.w...	[radio, controlle...]	Recreation	true
Reference	http://cos.arizon...	[ua, college, sci...]	Science	false
Shopping	http://www.metros...	[metro, swim, sho...]	Shopping	true
Arts	http://www.imdb.c...	[var, ue, t0, win...]	Arts	true
Arts	http://www.fabulo...	[fabulous, emma, ...]	Society	false
Computers	http://www.whetst...	[online, classroo...]	Business	false
Science	http://www.outlaw...	[geometrical, dim...]	Science	true
Recreation	http://people.del...	[mcconaughey, bird...]	Recreation	true
Shopping	http://www.aurell...	[fantasy, art, ni...]	Arts	false

only showing top 20 rows

Figure 4.1: Random sample taken from test set Classifier I

Figure A.1 shows the statistics per class. In here, the first three classes have a high recall (0.64-0.70) and low precision (0.43-0.50). Then, it switches the other way around and the remaining categories have a rather low recall (0.16-0.49) but higher precision (0.58-0.83). This can be explained by looking at the confusion matrix in Figure B.1. The data we worked with has quite some class imbalance. The first three categories ‘Society’, ‘Arts’ and ‘Business’ are considerably larger than the other ones. When the classifier was trained with the training data, it learnt a lot of items need to be classified as ‘Society’, ‘Arts’ or ‘Business’. So, a lot of items from the test data are classified as these three categories while their true label was something else. The last category ‘News’ is also relatively small compared to the middle classes, making it hard for the classifier to predict this label. News items are more often classified as ‘Society’, ‘Arts’ or ‘Business’ than ‘News’ itself, so its recall is really low.

Figure B.2 shows the normalised confusion matrix. Here, the colours and values give a better indication per class how good the classifier is. Visible is a diagonal line representing all True Positives. However, the first three columns are relatively darker showing a lot of items get classified incorrectly

as one of the three largest classes.

## 4.2 Classifier II - 277 categories

The second classifier contains 277 categories from the second level like ‘Arts/Movies’. Initially, this resulted in 451 different categories, but a lot were rather small. After removing all categories which had less than 100 Web pages in it, the number of categories downgraded to 277 and the number of Web pages in the dataframe became 509,082 (originally 514,506).

Figure 4.2 shows a random sample of the test data. One can see the categories are more detailed than the 13 maincategories.

categories	url	words	predicted_category	correct
Society/Religion_...	http://emmanuel.o...	[eag, function, s...	Society/Religion_...	true
Society/Religion_...	http://www.newadv...	[catholic, encycl...	Society/Religion_...	true
Society/Religion_...	https://www.umhef...	[united, methodis...	Society/Religion_...	true
Arts/Music	http://userpages....	[thomas, moore, p...	Arts/Music	true
Arts/Music	http://www.angelf...	[sarah, westlife,...	Arts/Music	true
Arts/Music	http://www.plugin...	[pluginmusic, com...	Arts/Music	true
Science/Math	http://cage.ugent...	[mackie, 2002, ma...	Reference/Education	false
Games/Video_Games	http://freedom.g...	[freedom, home, ...	Health/Medicine	false
Games/Video_Games	http://pc.gamespy...	[portal, pc, game...	Games/Video_Games	true
Games/Video_Games	http://www.gamefa...	[feel, magic, xy,...	Games/Video_Games	true
Society/People	http://robmello.b...	[var, tpapp, {}, ...	Society/Religion_...	false
Society/People	http://www.hirezf...	[albion, fuzz, bo...	Society/Ethnicity	false
Reference/Education	http://www.librar...	[ucsb, library, i...	Society/Religion_...	false
Business/Textiles...	http://www.avatro...	[avatron, teollis...	Society/Law	false
Business/Textiles...	http://www.perman...	[window, nreum, n...	Sports/Hockey	false
Arts/Literature	http://www.readin...	[window, nreum, n...	Arts/Performing_Arts	false
Computers/Graphics	http://kyomo.free...	[dollz, united, >...	Arts/Music	false
Society/Law	http://m-i-n-a.org/	[metric, importer...	Computers/Software	false
Business/Arts_and...	http://www.alanhu...	[portrait, photog...	Business/Arts_and...	true
Business/Arts_and...	http://www.edgemo...	[edgemont, video,...	Business/Construc...	false

only showing top 20 rows

Figure 4.2: Random sample taken from test set Classifier II

Table 4.2 shows the statistics of the classifier. The values are rather low compared to the previous classifier. The more categories, the more difficult it is for the classifier to predict correctly.

Assuming the classes behave in a similar style as in the previous classifier, we decided to calculate statistics for the first 10 classes, 10 classes in the middle and 10 at the end instead of giving the statistics for all 277 categories. These statistics can be found in Table A.2. In here, one can see the precision and recall do not consistently grow or decrease anymore as it did for the previous classifier. There are quite some classes with high precision and/or



Accuracy	0.36
Weighted precision	0.41
Weighted recall	0.36
Weighted F(1) Score	0.33

Table 4.2: Evaluation scores Classifier II

recall. Take, for example, category ‘Sports/Squash’. It has a  $P = 0.88$  and  $R = 0.24$ . This is rather high for a class with just a bit more than 100 Web pages in the dataset. Apparently, the classifier can predict this category really good which is probably because of the distinct word ‘Squash’. Some more ambiguous classes like ‘Society/Future’ have a precision and recall of both zero. This indicates that the selection of labels also hugely affects the performance of the classifier. Categories should have some unique characteristics like the name of a certain sport to classify on.

In the normalised confusion matrix shown in Figure B.3, one can distinguish a diagonal line for all True positives. It could have been a more consistent dark line; some categories fail and have a quite light TP, others are remarkable dark (for example an entry like ‘Sports/Squash’. Again, the first set of columns are considerably darker than other columns, because the classes are imbalanced. The first three categories have 45,752, 26,729 and 15,625 Web pages while the last classes have only just above a hundred. One can also notice some darker cells at random places (not on the diagonal TP line). These are probably due to some topics overlapping in real life as well, for example a Web page from ‘Science/Math’ predicted as ‘Reference/Education’ as visible in Figure 4.2.

### 4.3 Classifier III - 30 categories (more balanced)

This third classifier is made with more balanced data to see what class imbalance does to precision and recall. The intuition behind it is that categories that are really big, probably have subcategories that can be distinguished well by the classifier.

The target classification consists of the same maincategories ‘Recreation’, ‘Computers’, ‘Science’, ‘Sports’, ‘Shopping’, ‘Reference’, ‘Health’, ‘Games’, ‘Home’ and ‘News’ but the three originally largest categories ‘Society’, ‘Arts’, ‘Business’ are now split up using the second-level categories. These are ‘Society/Religion\_and\_Spirituality’, ‘Society/Law’, ‘Society/Issues’, ‘Society/People’, ‘Society/History’, ‘Society/Organizations’, ‘Arts/Music’, ‘Arts/Movies’, ‘Arts/Literature’, ‘Arts/Performing\_Arts’, ‘Arts/Visual\_Arts’, ‘Business/Construction\_and\_Maintenance’, ‘Business/Industrial\_Goods\_and\_Services’, ‘Business/Arts\_and\_Entertainment’, ‘Business/Business\_Services’, ‘Business/-

Consumer\_Goods\_and\_Services’ and ‘Business/Textiles\_and\_Nonwovens’. The Web pages belonging to rather small subcategories are unioned into the three general ‘Society’, ‘Arts’, ‘Business’ categories, resulting in 30 categories in total. As already mentioned in Section 3.2.1, we choose this division based on our own judgement. The process, however, could be automated.

Figure 4.3 shows a random sample of the test data.

categories	url	words	predicted_category	correct
Society/Religion_...	http://www.newadv...	[catholic, encycl...	Society/Religion_...	true
Society/Religion_...	http://www.emanue...	[emanuel, luthera...	Society/Religion_...	true
Society/Religion_...	http://www.newadv...	[catholic, encycl...	Society/Religion_...	true
Society/Religion_...	http://www.thezen...	[thezensite, teis...	Society/Religion_...	true
Science	http://www.fields...	[field, institute...	Science	true
Society/Religion_...	http://www.sbbc.c...	[sbbc, beat, mess...	Society/Religion_...	true
Computers	http://www.h5.com/	[h5, e, discovery...	Business	false
Business	http://www.blueri...	[blue, ridge, tra...	Shopping	false
Business	http://www.pan-pi...	[wire, drawing, m...	Business	true
Science	http://www.antiqu...	[antiquity, man, ...]	Science	true
Business	http://www.alpaca...	[error, page, can...	Computers	false
Business	http://www.carein...	[st, louis, misso...	Health	false
Business	http://iechristma...	[inland, empire, ...]	Computers	false
Computers	http://www.tutori...	[blender, game, e...	Health	false
Recreation	http://www.pbase...	[pbase, com, impo...	Recreation	true
Recreation	https://www.caese...	[caeses, software...	Computers	false
Recreation	http://www.greyho...	[home, greyhound, ...]	Recreation	true
Society	http://www.crstru...	[body, {, backgro...	Society/Religion_...	false
Health	http://www.mcmc.net/	[mid, columbia, m...	Business	false
Arts/Music	http://www.ipl.or...	[ipl2, page, move...	Science	false

only showing top 20 rows

Figure 4.3: Random sample taken from test set Classifier III

Table 4.3 shows the statistics for this classifier. This third division of labels lowered all scores compared to the first classifier.

Accuracy	0.45
Weighted precision	0.48
Weighted recall	0.45
Weighted F(1) Score	0.43

Table 4.3: Evaluation scores Classifier III

Figure B.4 is the confusion matrix for this classifier. Notice that the three classifiers are trained and tested with three different random data splits, so one cannot compare the absolute numbers from one confusion matrix with another. The normalised confusion matrices on the other hand

can be compared to each other.

In Figure B.5, a good diagonal line is visible at the left top. Also, the first set of columns is not that dark anymore as in the confusion matrix for the first classifier. However, another problem now occurs. A lot of Web pages were classified as ‘Business’ while it belonged to one of its subgroups like ‘Business/Business\_Services’ and ‘Business/Industrial\_Goods\_and\_Services’. This leads to the diagonal TP line not being that present anymore at the right bottom for these subcategories. Their recall is around 0.1. As a consequence, the total accuracy is rather low in table 4.3 and does not tell us a lot about the performance of the classifier compared to the others. Other subcategories such as ‘Arts/Movies’ and ‘Arts/Literature’ seem to perform well with a precision of 0.65 and 0.60 and a recall of 0.65 and 0.39, respectively.

The recall increased for all maincategories from ‘Recreation’ till ‘News’, when comparing Table A.1 with Table A.3. However, the precision lowered for all categories. This can be attributed to the fact that more TP arose for these classes because they became relatively larger in size compared to the now multiple second-level categories. Splitting the largest three categories did solve the class imbalance, resolved the problem of a lot of items being wrongly classified as one of these largest classes, and increased their recalls. The F1 scores stay about the same.

## Chapter 5

# Conclusions

In this thesis, we have built a dataset consisting of ~0.5 million Web pages categorised with a certain topic. Three classifiers were trained each given a different selection of labels. The first classifier seems to perform best, but it also has the least categories to classify on. The second classifier has so many categories, the overall performance lowered quite a bit. However, still a lot of distinct categories such as ‘Sports/Squash’ had a high precision and recall even when they did not have a lot of training data and there was quite some class imbalance. Conquering this class imbalance, the input to classifier III had the three largest main categories split up in multiple subcategories. This resulted in a higher recall for all normal maincategories, but the classifier was not that great at distinguishing Web pages between the different kinds of ‘Business’ subcategories (lowering the overall performance). Subcategories such as ‘Arts/Movies’ and ‘Arts/Literature’ classified really well on the other hand.

The results of the three classifiers combined indicate that there is much potential for classification of the Web, as long as the selection of labels is clear and distinct. The hierarchy of the DMOZ dataset is carefully human-chosen, but one can argue whether this categorisation is effective when applied on a classification model. The hierarchies are too deep, some categories are really similar to another and there is lots of class imbalance (a few considerably larger categories, and many sparsely populated categories), making it hard for a classifier to predict Web pages when there are many categories to chose from.

We would recommend neither of the three classifiers, as they do not seem to be sufficient enough for practical applications. Each have their own flaws, however, give an indication what to keep in mind when building upon this current research. 13 categories is not sufficient to represent the Web for goals we envision, classifier II shows it is able to predict classes with little data quite correct as long as its topic is distinct from others, and the third classifier shows categories should indeed not overlap when using single-label

classification. For a proposal of a better classification, we refer to the future work in Section 6.3.

## Chapter 6

# Future Work

Building a classifier for Web data is a long process, not only involving the training of the classifier, but also many pre-processing steps of getting the proper training data. Because there are so many steps and choices to be made, there are also lots of opportunities for improvements.

### 6.1 Expanding data

The DMOZ dump data consists of ~4 million Web pages. After dropping the ‘World’ and ‘Regional’ categories, the size of the dataset reduces to ~1 million Web pages. The category ‘World’ cannot be used when building an English text classifier. However, as future work, the category ‘Regional’ could be added to the data. This category has a difficult hierarchy with paths such as ‘Regional/Europe/United\_Kingdom/Recreation\_and\_Sports/Sports/Country\_Sports/Fishing’. The Web pages within this category could be transferred to the category ‘Recreation/Outdoors/Fishing’ which is already one of the labels in the data.

Something similar occurs with the ‘Kids&Teens’ directory. This directory resides in a different XML-file than `content.rdf.u8`, namely `kt-content.rdf.u8`. This XML-file also has categories like ‘Arts’ and ‘Computers’, but contain new categories like ‘Teen Life’ and ‘School Time’ as well. One can merge these similar categories with the standard DMOZ categories and come up with an idea to work with the new categories (e.g., adding them to the labels as new categories, not including them, etc).

Lastly, data could be expanded by improving the joining phase of the DMOZ and Common Crawl. In this step, the data reduced from 979,845 entries to 514,506 entries, simply because the DMOZ URL did not exist in the Common Crawl. More Common Crawl buckets (from other months) could be added to match on. Ideally, all DMOZ URLs match to some record in the Common Crawl, future work can find out whether this is possible.

## 6.2 Improving cleaned content of the Web pages

For the content of each Web page, this thesis included some basic cleaning plus stop word removal and lemmatisation. Another natural language processing task is stemming which can be executed in future to see whether this improves the classification.

Another feature of NLTK, the package that was used in this thesis, is the ability to remove non-English words. We decided not to use this function in the end, because it removed too many words. See the example Web content in Figure 6.1 for clarification. As future work, another function inside NLTK or a package other than NLTK could be found that works better for removing non-English words. This will also remove unwanted HTML data that was missed during tag-removal.

```
web_content = '[robert, louis, stevenson, holdings, robert, louis, stevensonnotable, editions, inthomas, cooper, librarythis, page, provides, information, on, editions, of, works, by, robert, louisstevenson, that, may, be, of, particular, interest, to, scholars, it, is, by, no, meansan, exhaustive, list, of, all, thomas, cooper, library, s, stevenson, holdings, noris, it, a, complete, list, of, stevenson, editions, held, in, the, rare, books, room, such, lists, are, available, through, uscan, the, texts, have, been, separated, into, seven, groups, of, works, withinwhich, they, are, listed, chronologically, by, first, publication, in, book, form, the, seven, groups, are, novelsshort, storiespoetryessaystravelslettersmiscellaneous, return, to, scottishliterary, holdingsreturnto, rare, booksand, special, collections, this, page, last, updated, 27, may, 2003, by, the, department, of, rare, books, and, special, collections]'
```

```
web_content(without stopwords) = '[robert, louis, stevenson, holdings, robert, louis, stevensonnotable, editions, inthomas, cooper, librarythis, page, provides, information, editions, works, robert, louisstevenson, may, particular, interest, scholars, meansan, exhaustive, list, thomas, cooper, library, stevenson, holdings, noris, complete, list, stevenson, editions, held, rare, books, room, lists, available, uscan, texts, separated, seven, groups, works, withinwhich, listed, chronologically, first, publication, book, form, seven, groups, novelsshort, storiespoetryessaystravelslettersmiscellaneous, return, scottishliterary, holdingsreturnto, rare, booksand, special, collections, page, last, updated, 27, may, 2003, department, rare, books, special, collections]'
```

```
web_content(without stopwords and non-English words) = '[cooper, page, information, works, may, particular, interest, exhaustive, list, cooper, library, complete, list, rare, room, available, seven, works, listed, chronologically, first, publication, book, form, seven, return, rare, special, page, last, may, department, rare, special]'
```

Figure 6.1: In this research is chosen for box two. The boxes are a visualisation of which stop words are removed and which non-English words would be removed if continued cleaning. One can see non-English words being removed, but also some named entities such as ‘robert louis stevenson’ which could be useful information. All plurals are removed as well, meaning lemmatisation or stemming needs to be performed first before removing non-English words could become an option.

## 6.3 Improving the classifier

In this thesis, the classification model stays quite basic. No hyper-parameters were tuned to improve the performance of the classifier. As future research, cross-validation should be performed to find the best combination of hyper-parameters.

A better classifier could also arise when the labels are better defined. Classifier II and III show there is potency in splitting the categories into more subcategories, as long as the categories stay distinct. More investigations for classifiers are possible; (1) a training could be performed on all lowest levels of all categories, (2) on all third-level categories, or (3) one with an intuitively clever selection of labels to make them as distinct as possible. The trick is to come up with a different categorisation than the standard DMOZ categories such that it enhances Web classification.

For this latter research idea using the DMOZ categories, labels should be split based on distinctiveness of the emerging subcategories. For example, splitting ‘Sports’ results in subcategories such as ‘Sports/Bowling’ and ‘Sports/Golf’ but also in ‘Sports/Water\_sports’ and ‘Sports/Winter\_sports’. One can argue the last two are still not distinct enough as a label for the classifier and one must take their subcategories again as final label, i.e., categories like ‘Sports/Water\_sports/Kitesurfing’ and ‘Sports/Winter\_sports/Snowboarding’.

## 6.4 Personalised Web search

Once satisfaction is reached of improving the classification, future research could continue to build upon work by Bennett et al. [3] by looking if the predicted classes can also be stored as feature during indexing time and improve query results. Research can also be directed towards personalised Web search in the form of offline search engines. Once Web pages are classified in certain topics, it becomes relatively easy to retrieve a certain subset of personal interest and store this offline on your computer. We suggest using ChatNoir [4] if research is performed in this direction, which is a search engine using Common Crawl data from January 2017 as document collection. One could investigate whether the ranking worsens once a query is executed on only a subset of the document collection (with the same topic of the query) or actually improves because users want their query results to be in the same class as the query itself.



# Bibliography

- [1] Sadegh Aliakbary, Hassan Abolhassani, Hossein Rahmani, and Behrooz Nobakht. Web page classification using social tags. In *2009 International Conference on Computational Science and Engineering*, volume 4, pages 588–593, 2009.
- [2] The Internet Archive. Wayback machine. <https://archive.org/web/>.
- [3] Paul N. Bennett, Krysta Svore, and Susan T. Dumais. Classification-enhanced ranking. In *Proceedings of the 19th International Conference on World Wide Web, WWW '10*, page 111–120, New York, NY, USA, 2010. Association for Computing Machinery.
- [4] Janek Bevendorff, Benno Stein, Matthias Hagen, and Martin Potthast. Elastic ChatNoir: Search Engine for the ClueWeb and the Common Crawl. In Leif Azzopardi, Allan Hanbury, Gabriella Pasi, and Benjamin Piwowarski, editors, *Advances in Information Retrieval. 40th European Conference on IR Research (ECIR 2018)*, Lecture Notes in Computer Science, Berlin Heidelberg New York, March 2018. Springer.
- [5] Curlie. About curlie. <https://curlie.org/docs/en/about.html>.
- [6] Arjen P. de Vries. Zeppelin on redbad, 2020. <https://rubigdata.github.io/course/admin/zeppelin.html>.
- [7] The Matplotlib development team. `matplotlib.pyplot.plot`. [https://matplotlib.org/3.2.0/api/\\_as\\_gen/matplotlib.pyplot.plot.html](https://matplotlib.org/3.2.0/api/_as_gen/matplotlib.pyplot.plot.html).
- [8] The Apache Software Foundation. Apache spark. <https://spark.apache.org/>.
- [9] The Common Crawl Foundation. Common crawl. <https://commoncrawl.org/>.
- [10] JongWoo Ha, Jung-Hyun Lee, Won-Jun Jang, Yong-Ku Lee, and SangKeun Lee. Toward robust classification using the open directory project. In *2014 International Conference on Data Science and Advanced Analytics (DSAA)*, pages 607–612, 2014.

- [11] Lisa Hoek. Web classification using dmoz. <https://github.com/LisaHoek/dmozclassification>.
- [12] AOL Inc. Dmoz - the directory of the web. <https://dmoz-odp.org/>.
- [13] Wikimedia Foundation Inc. List of english contractions. [https://en.wikipedia.org/wiki/Wikipedia%3aList\\_of\\_English\\_contractions](https://en.wikipedia.org/wiki/Wikipedia%3aList_of_English_contractions).
- [14] Idil Ismiguzel. Nlp-with-python, May 2020. <https://github.com/Idilismiguzel/NLP-with-Python>.
- [15] Sathi T. Marath, Michael Shepherd, Evangelos Milios, and Jack Duffy. Large-scale web page classification. In *2014 47th Hawaii International Conference on System Sciences*, pages 1813–1822, 2014.
- [16] NLTK project. Natural language toolkit. <https://www.nltk.org/>.
- [17] Xiaoguang Qi and Brian D. Davison. Web page classification: Features and algorithms. 41(2), February 2009.
- [18] Leonard Richardson. beautifulsoup4. <https://pypi.org/project/beautifulsoup4/>.
- [19] David Sontag, Kevyn Collins-Thompson, Paul N. Bennett, Ryen W. White, Susan Dumais, and Bodo Billerbeck. Probabilistic models for personalizing web search. In *Proceedings of the Fifth ACM International Conference on Web Search and Data Mining, WSDM '12*, page 433–442, New York, NY, USA, 2012. Association for Computing Machinery.
- [20] Apache Spark. Mllib: Main guide - multinomial logistic regression. <https://spark.apache.org/docs/2.4.4/ml-classification-regression.html#multinomial-logistic-regression>.
- [21] Apache Spark. Pyspark 3.0.1 documentation. <https://spark.apache.org/docs/latest/api/python/index.html>.
- [22] Utsav. Building a url classifier using dmoz data, January 2017. <https://utatds.github.io/2017-01-18-URL-classification-using-DMOZ-data/>.
- [23] Michael Waskom. Seaborn.heatmap. <https://seaborn.pydata.org/generated/seaborn.heatmap.html>.

## Appendix A

### Evaluation scores

<b>Class</b>	<b>Precision</b>	<b>Recall</b>	<b>F(1) Score</b>
Society	0.50	0.70	0.58
Arts	0.52	0.70	0.59
Business	0.43	0.64	0.51
Recreation	0.62	0.37	0.47
Computers	0.68	0.46	0.55
Science	0.67	0.44	0.53
Sports	0.77	0.49	0.60
Shopping	0.60	0.36	0.45
Reference	0.58	0.28	0.38
Health	0.73	0.38	0.50
Games	0.83	0.36	0.50
Home	0.72	0.31	0.43
News	0.67	0.16	0.25

Table A.1: Evaluation scores Classifier I

Label	Class	Precision	Recall	F(1) Score
0	Society/Religion_and_Spirituality	0.26	0.81	0.40
1	Arts/Music	0.26	0.69	0.37
2	Reference/Education	0.30	0.46	0.36
3	Arts/Movies	0.61	0.47	0.53
4	Computers/Software	0.32	0.45	0.37
5	Science/Biology	0.69	0.67	0.68
6	Games/Video_Games	0.53	0.51	0.52
7	Arts/Literature	0.60	0.41	0.49
8	Society/Law	0.48	0.55	0.51
9	Arts/Performing_Arts	0.37	0.24	0.29
⋮				
134	Games/Gambling	0.69	0.34	0.45
135	Home/Consumer_Information	0.56	0.10	0.16
136	Recreation/Living_History	0.69	0.27	0.38
137	Society/Military	0.38	0.74	0.12
138	Business/Aerospace_and_Defense	0.08	0.01	0.01
139	Society/Government	0.39	0.08	0.13
140	Business/Environment	0.65	0.10	0.17
141	Business/Materials	0.13	0.01	0.02
142	Shopping/Toys_and_Games	0.08	0.01	0.01
143	Business/Real_Estate	0.64	0.13	0.21
⋮				
267	Sports/Squash	0.88	0.24	0.38
268	Health/Senses	0.0	0.0	0.0
269	Society/Future	0.0	0.0	0.0
270	Sports/Darts	0.5	0.07	0.12
271	Games/Conventions	0.0	0.0	0.0
272	Computers/Artificial_Life	0.33	0.05	0.08
273	Arts/Digital	0.0	0.0	0.0
274	Shopping/Weddings	0.0	0.0	0.0
275	Business/News_and_Media	0.0	0.0	0.0
276	Sports/Badminton	0.5	0.10	0.16

Table A.2: Evaluation scores first, middle and last 10 classes Classifier II

<b>Class</b>	<b>Precision</b>	<b>Recall</b>	<b>F(1) Score</b>
Society/Religion_and_Spirituality	0.42	0.75	0.54
Recreation	0.35	0.55	0.43
Computers	0.50	0.59	0.54
Business	0.29	0.49	0.37
Science	0.49	0.56	0.53
Sports	0.59	0.56	0.58
Shopping	0.40	0.50	0.44
Arts/Music	0.69	0.43	0.53
Arts	0.43	0.26	0.32
Reference	0.52	0.31	0.39
Health	0.64	0.41	0.50
Society	0.55	0.28	0.37
Games	0.30	0.48	0.37
Arts/Movies	0.65	0.65	0.65
Arts/Literature	0.60	0.39	0.47
Home	0.64	0.30	0.41
Society/Law	0.75	0.42	0.54
Arts/Performing_Arts	0.63	0.10	0.18
Business/Construction_and_Maintenance	0.45	0.11	0.18
Business/Industrial_Goods_and_Services	0.52	0.17	0.26
Business/Arts_and_Entertainment	0.53	0.10	0.17
Business/Business_Services	0.22	0.05	0.08
Business/Consumer_Goods_and_Services	0.11	0.01	0.02
Arts/Visual_Arts	0.55	0.05	0.09
Society/Issues	0.51	0.18	0.26
Society/People	0.13	0.03	0.04
Business/Textiles_and_Nonwovens	0.44	0.13	0.20
Society/History	0.55	0.28	0.37
Society/Organizations	0.74	0.13	0.23
News	0.18	0.19	0.18

Table A.3: Evaluation scores Classifier III

## Appendix B

### Confusion matrices

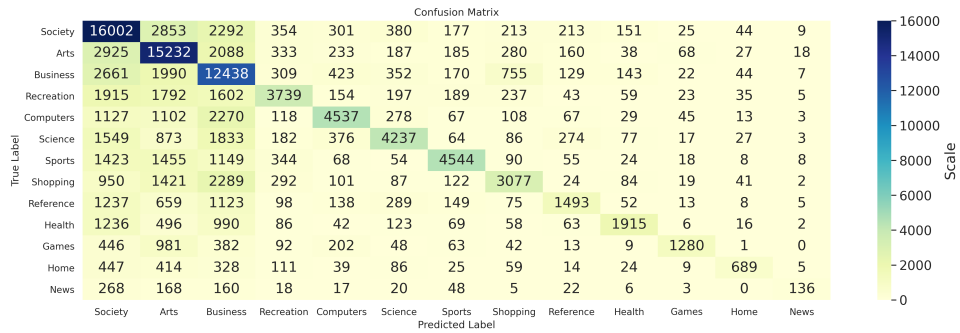


Figure B.1: Confusion matrix for Classifier I with 13 labels

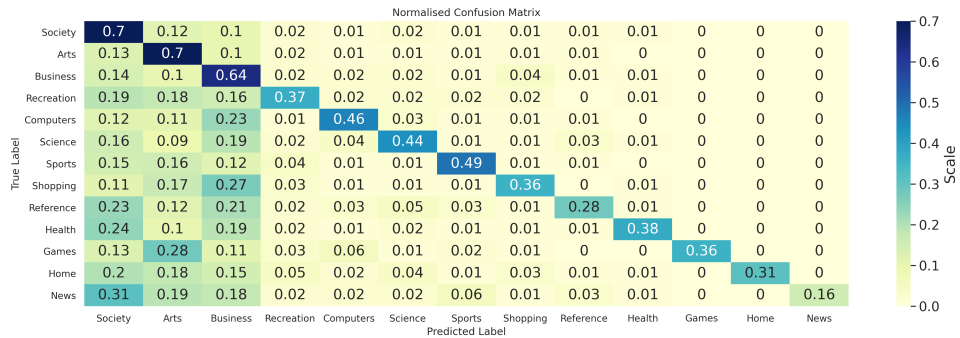


Figure B.2: Normalised confusion matrix for Classifier I with 13 labels

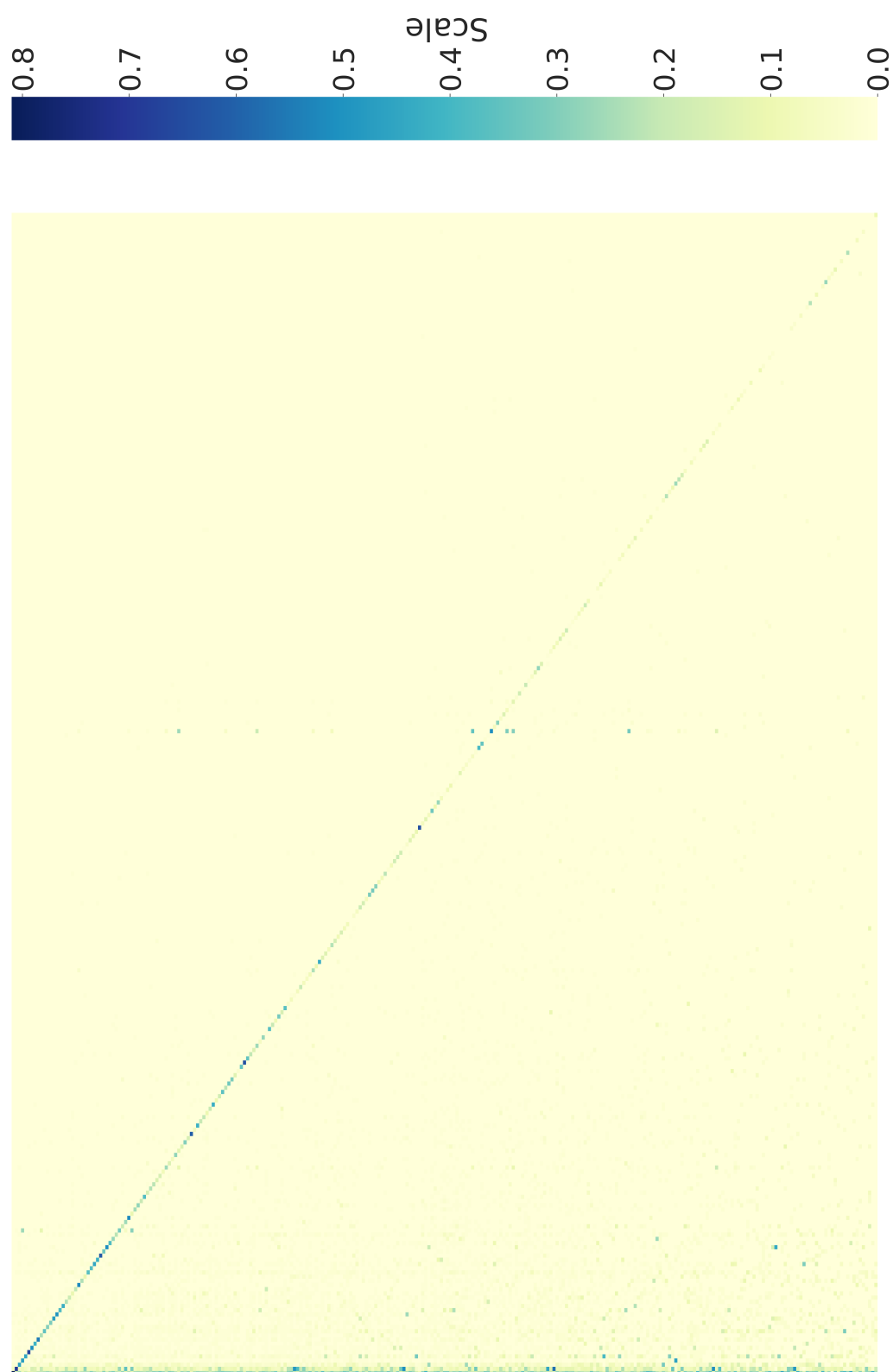


Figure B.3: Normalised confusion matrix for Classifier II with 277 labels

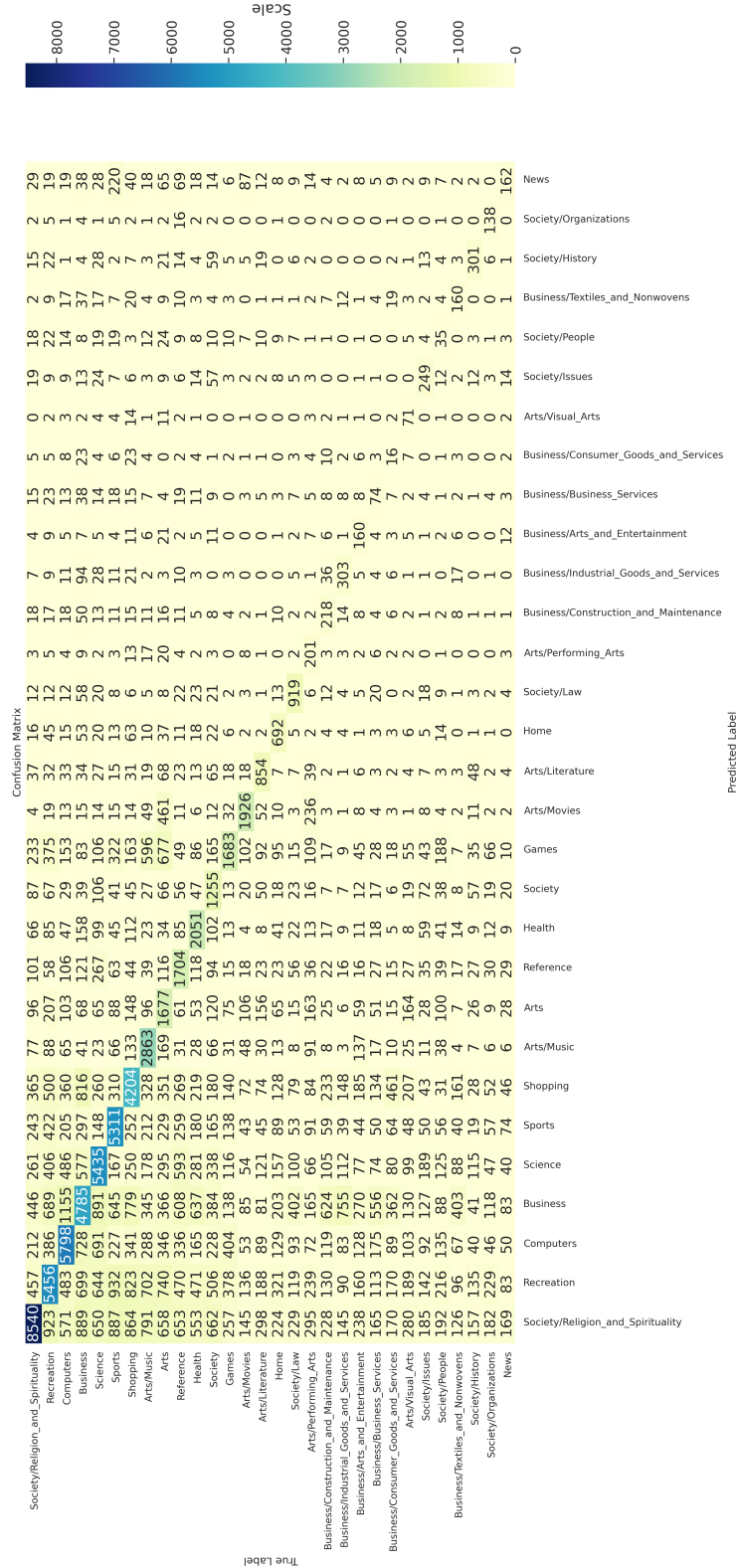
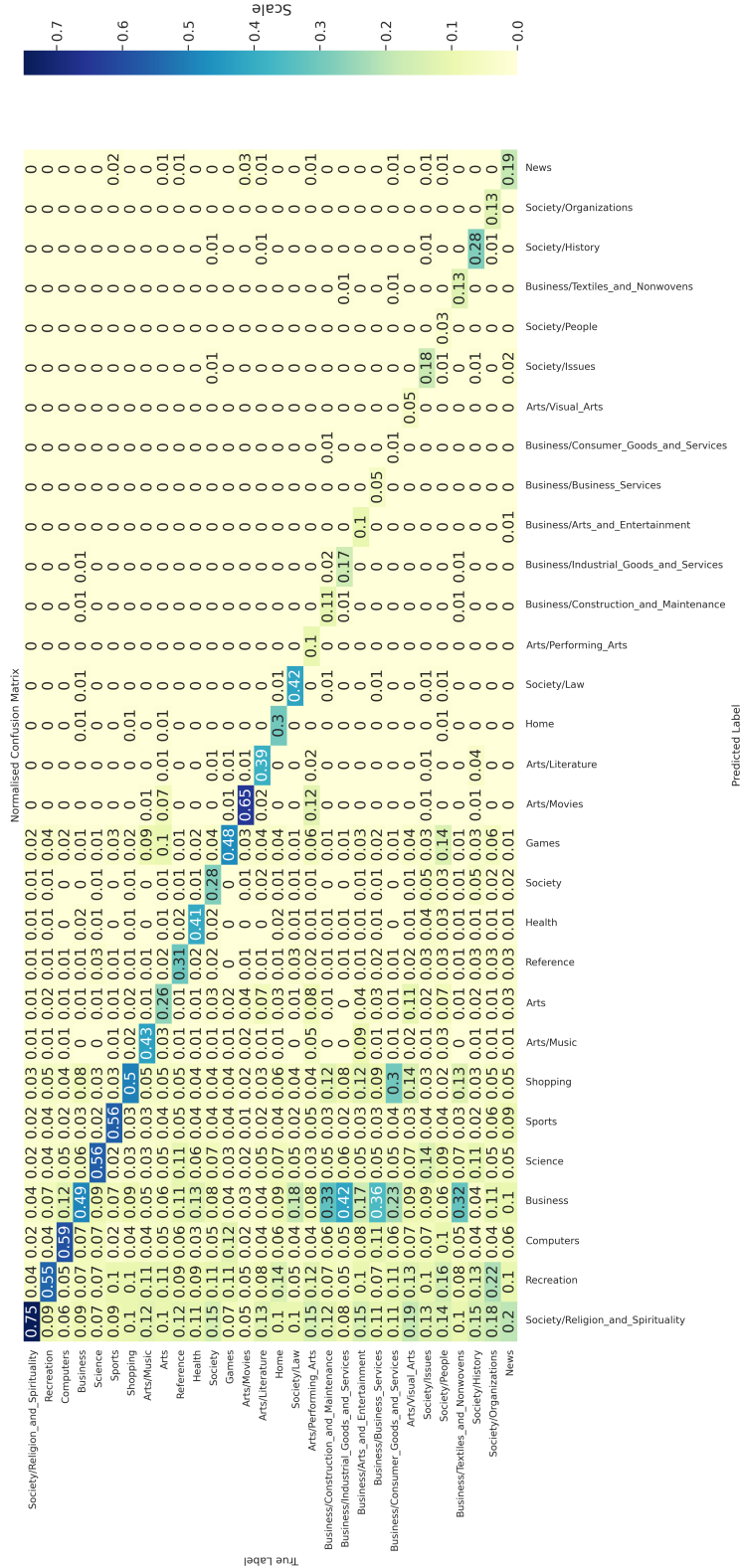


Figure B.4: Confusion matrix for Classifier III with 30 labels





# Appendix C

## Code examples

For the Zeppelin Notebooks and full Python code, we refer to our GitHub repository. [11]

---

```
import os
import re
import pandas as pd
from pydrive.auth import GoogleAuth
from pydrive.drive import GoogleDrive
from google.colab import auth
from oauth2client.client import GoogleCredentials

# Give permission to Google drive
auth.authenticate_user()
gauth = GoogleAuth()
gauth.credentials = GoogleCredentials.get_application_default()
drive = GoogleDrive(gauth)

# Make sure the content.rdf.u8 file is in your google drive.
# The ID is part of the link to the file (click on retrieve link)
id = "INSERT_ID_HERE"
downloaded = drive.CreateFile({'id':id})
downloaded.GetContentFile('content.rdf.u8')

df = pd.DataFrame(columns=['maincategory','categories', 'url', 'title'])

with open('content.rdf.u8', 'r') as contents:
    lines = contents.read().split("</ExternalPage>")

topics_mainlist = []
topics_list = []
titles_list = []
urls_list = []

lines = [str(line) for line in lines]
for line in lines:
    titles = re.findall('<d:Title>(.)</d:Title>', line)
    urls = re.findall('<ExternalPage about="(.)">', line)
```

```

topics = re.findall('<topic>(.)</topic>', line)
if titles and urls and topics:
    topics_mainlist.append(topics[0].split('/')[1])
    topics_list.append(topics[0])
    titles_list.append(titles[0])
    urls_list.append(urls[0])

del lines
del titles
del urls
del topics

df.maincategory = topics_mainlist
df.categories = topics_list
df.url = urls_list
df.title = titles_list

df = df.drop(df[df.maincategory == 'World'].index)
df = df.drop(df[df.maincategory == 'Regional'].index)

from google.colab import drive
drive.mount('drive')

df.to_csv('urls.csv')
!cp urls.csv "drive/My Drive/"

```

---

Example code 1: Python code that was used in Google Colab to obtain the DMOZ data from `content.rdf.u8`

---

```

val urls = s"hdfs://user/lisa/urls.csv"
val dfU = spark.read.option("header",true).option("inferSchema",true).csv(urls)
dfU.createGlobalTempView("odp")

val df = spark.read
    .option("mergeSchema", "true")
    .option("enable.summary-metadata", "false")
    .option("enable.dictionary", "true")
    .option("filterPushdown", "true")
    .parquet("s3://commoncrawl/cc-index/table/cc-main/warc/crawl=CC-MAIN-2017-04/"
        + "subset=warc/")
    .drop("url_host_name", "url_host_tld", "url_host_2nd_last_part",
        "url_host_3rd_last_part", "url_host_4th_last_part", "url_host_5th_last_part",
        "url_host_registry_suffix", "url_host_registered_domain", "url_host_private_suffix",
        "url_host_private_domain", "url_protocol", "url_port", "url_path", "url_query",
        "fetch_time", "fetch_status", "content_digest", "content_mime_type",
        "content_mime_detected", "warc_segment")
df.createGlobalTempView("cc")

val joinDF = spark.sql("SELECT maincategory, categories, title, c.url, " +
    "warc_filename, warc_record_offset, warc_record_length FROM " +
    "global_temp.odp o INNER JOIN global_temp.cc c WHERE o.url == c.url").cache()

```

```

joinDF = joinDF.dropDuplicates("url")

joinDF.write.format("csv").option("header",true).mode("overwrite")
    .option("sep", ",").save(s"hdfs:/user/lisa/join.csv")

```

---

Example code 2: Basic Scala code example of executing the JOIN

---

```

def get_content(filename, offset, length):
    import zlib, requests
    from bs4 import BeautifulSoup

    url = "https://commoncrawl.s3.amazonaws.com/"+filename
    r = requests.get(url, headers={'Range': 'bytes={}-{'
        .format(offset, offset+length-1)})
    # Here we need to do a try-except block
    try:
        data = zlib.decompress(r.content, wbits = zlib.MAX_WBITS | 16)
        html = data.decode('utf-8',errors='ignore')
        soup = BeautifulSoup(html, 'lxml')
        content = soup.get_text()
    except zlib.error:
        content = 'Decompression error; invalid/incomplete request response'
    try:
        warc, header, response = content.strip().split('\r\n\r\n', 2)
        return response
    except:
        return ""

```

---

Example code 3: Python function for getting the Web page's content

---

```

def clean_text(text, remove_stopwords = True):
    import re
    import nltk
    from nltk.corpus import stopwords
    from nltk.stem import WordNetLemmatizer
    nltk.data.path.append('./nltk_data')

    # Convert words to lower case
    text = text.lower()
    # Remove whitespaces
    text = text.strip()

    # Replace contractions with their longer forms
    if True:
        text = text.split()
        new_text = []
        for word in text:
            if word in contractions:
                new_text.append(contractions[word])
            else:
                new_text.append(word)

```

```

        text = " ".join(new_text)
        # Format words and remove unwanted characters
        text = re.sub(r'https?:\/\/.*[\r\n]*', '', text, flags=re.MULTILINE)
        text = re.sub(r'\<a href', ' ', text)
        text = re.sub(r'&', ' ', text)
        text = re.sub(r'[_"-;%()|+&=%.,!?:#$@\[ \]/]', ' ', text)
        text = re.sub(r'\<br />', ' ', text)
        text = re.sub(r'\'', ' ', text)
        text = re.sub(r'[\x00-\x7F]+', ' ', text)
        # Remove stop words and non-English words
        if remove_stopwords:
            text = text.split()
            words = set(nltk.corpus.words.words())
            stops = set(stopwords.words("english"))
            text = [w for w in text if not w in stops]
            #text = [w for w in text if w in words and not w in stops]
            text = " ".join(text)
        # Tokenize each word
        word_list = nltk.WordPunctTokenizer().tokenize(text)

        # Lemmatize each word and put back in one string
        lemm = nltk.stem.WordNetLemmatizer()
        lemmatized_output = ' '.join([lemm.lemmatize(w) for w in word_list])

        # Return cleaned text
        return lemmatized_output

```

---

Example code 4: Adjusted Python function for cleaning the Web content, original version written by Ismiguzel [14]

---

```

%pyspark
from pyspark.ml import Pipeline, PipelineModel
from pyspark.ml.classification import LogisticRegression
from pyspark.ml.feature import HashingTF, Tokenizer, StringIndexer, IndexToString
from pyspark.ml.feature import StringIndexerModel
from pyspark.mllib.evaluation import MulticlassMetrics
from pyspark.sql.functions import col

spark.sql("DROP TABLE IF EXISTS cccc")
spark.sql(
    "CREATE TABLE cccc(categories STRING, title STRING, url STRING, " +
    "cleaned_content STRING, maincategory STRING) USING parquet PARTITIONED " +
    "BY(maincategory) CLUSTERED BY(categories) INTO 12 BUCKETS " +
    "LOCATION 'hdfs:/user/arjen/cccc'"
)
spark.sql("MSCK REPAIR TABLE cccc")

df = spark.table("CCCC")

# Transform the categories to numeric labels
indexer = StringIndexer(inputCol="maincategory", outputCol="label",
    handleInvalid="keep")

```

```

trainpl = Pipeline(stages=[indexer])
indexermodel = trainpl.fit(df)
d = indexermodel.transform(df)

# Split data in training and test data
splits = d.randomSplit([0.75, 0.25], 24)
trainDF = splits[0]
testDF = splits[1]

# Configure an ML pipeline
tokenizer = Tokenizer(inputCol="cleaned_content", outputCol="words")
hashingTF = HashingTF(inputCol=tokenizer.getOutputCol(), outputCol="features")
lr = LogisticRegression(maxIter=10, regParam=0.001)
pipeline = Pipeline(stages=[tokenizer, hashingTF, lr])

# Fit the pipeline to training documents.
model = pipeline.fit(trainDF)

# Make predictions on test documents.
testSet = model.transform(testDF)

# Convert the numeric labels back to the categories
labels = (indexermodel.stages[0]).labels
converter = IndexToString(inputCol="prediction", outputCol="predicted_category",
    labels=labels)
converted = converter.transform(testSet)

# Start evaluation
predictionAndLabels = testSet.select(col("prediction"), col("label")).rdd
metrics = MulticlassMetrics(predictionAndLabels)

# Overall statistics
precision = metrics.precision()
recall = metrics.recall()
f1Score = metrics.fMeasure()
print("Precision = %s" % precision)
print("Recall = %s" % recall)
print("F1 Score = %s" % f1Score)

# Statistics by class
classes = d.select("label").rdd.flatMap(lambda x:x).distinct().collect()
for label in sorted(classes):
    print("Class %s precision = %s" % (dict.get(label), metrics
        .precision(label)))
    print("Class %s recall = %s" % (dict.get(label), metrics.recall(label)))
    print("Class %s F1 Measure = %s" % (dict.get(label), metrics
        .fMeasure(label, beta=1.0)))

# Weighted stats
print("Weighted recall = %s" % metrics.weightedRecall)
print("Weighted precision = %s" % metrics.weightedPrecision)
print("Weighted F(1) Score = %s" % metrics.weightedFMeasure())

# Confusion matrix

```

```
confusionMatrix = metrics.confusionMatrix()
```

---

Example code 5: General code example for training, testing and evaluating the three classifiers