DEGREE PROJECT IN THE FIELD OF TECHNOLOGY
INFORMATION AND COMMUNICATION TECHNOLOGY
AND THE MAIN FIELD OF STUDY
COMPUTER SCIENCE AND ENGINEERING,
SECOND CYCLE, 30 CREDITS
STOCKHOLM, SWEDEN 2020

Web Scraping using Machine Learning

VICTOR CARLE

KTH ROYAL INSTITUTE OF TECHNOLOGY
SCHOOL OF ELECTRICAL ENGINEERING AND COMPUTER SCIENCE

Web Scraping using Machine Learning

VICTOR CARLE

Master in Computer Science
Date: March 2020
Supervisor: Somayeh Aghanavesi
Examiner: Olov Engwall
School of Electrical Engineering and Computer Science
Host company: Söderberg & Partners
Swedish title: Webbskrapning med maskininlärning

Abstract

This thesis explores the possibilities of creating a robust Web Scraping algorithm, designed to continously scrape a specific website even though the HTML code is altered. The algorithm is intended to be used on websites that have a repetitive HTML structure containing data that can be scraped. A repetitive HTML structure often displays; news articles, videos, books, etc. This creates code in the HTML which is repeated many times, as the only thing different between the things displayed are for example titles. A good example would be Youtube. The scraper works through using text classification of words in the code of the HTML, training a Support Vector Machine to recognize the words or variable names. Classification of the words surrounding the sought-after data is done with the assumption that the future HTML of a website will be similar to the current HTML, this in turn allows for robust scraping to be performed. To evaluate its performance a web archive is used in which the performance of the algorithm is back-tested on past versions of the site to hopefully get an idea of what the performance in the future might look like. The algorithm achieves varying results depending on a large variety of variables within the websites themselves as well as the past versions of the websites. The best performance was achieved on Yahoo news achieving an accuracy of 90 % dating back three months from the time the scraper stopped working.

Sammanfattning

Den här rapporten undersöker vad som krävs för att skapa en robust webb-skrapare, designad för att kontinuerligt kunna skrapa en specifik hemsida trots att den underliggande HTML-koden förändras. En algoritm presenteras som är lämplig för hemsidor med en repetitiv HTML-struktur. En repetitiv HTML struktur innebär ofta att det visas saker såsom nyhetsartiklar, videos, böcker och så vidare. Det innebär att samma HTML-kod återanvänds ett flertal gånger, då det enda som skiljer de här sakerna åt är exempelvis deras titlar. Ett bra exempel är hemsidan Youtube. Skraparen funkar genom att använda text-klassificering av ord som finns i HTML-koden, på så sätt kan maskinlärnings-algoritmen, support vector machine, känna igen den kod som omger datan som är eftersökt på hemsidan. För att möjliggöra detta så förvandlas HTML-koden, samt relevant metadata, till vektorer med hjälp av bag-of-words-modellen. Efter omvandlingen kan vektorerna matas in i maskinlärnings-modellen och klassifiera datan. Algoritmen testas på äldre versioner utav hemsidan tagna från ett webarkiv för att förhoppningsvis få en bra bild utav vad framtida prestationer skulle kunna vara. Algoritmen uppnår varierande resultat baserat på en stor mängd variabler inom hemsidan samt de äldre versionerna av hemsidorna. Algoritmen presterade bäst på Yahoo news där den uppnådde 90 % träffsäkerhet på äldre sidor.

Contents

Chapter 1

Introduction

Since the creation of the internet there has been a vast increase in the amount of data available for consumption, this data is only expected to continue growing at a rapid pace [1]. Due to the free nature of the internet, this data can be displayed in a variety of different ways which is often intended to appeal to humans. Making sense of the data and structuring it conveniently is a difficult task, due to the sheer amount of it.

Retrieving data automatically is primarily achieved through web scraping when there is no direct access to the source of data. Web scraping is a term used for automatically retrieving data from the internet and structuring it in a useful manner. The standard scraping algorithm makes use of static paths to navigate the program through the HTML to the sought-after data. Using static paths in a non-static environment results in the scrapers breaking as the HTML is updated requiring an update of the code which performs the scraping. The frequency at which this must be done is largely dependent on the website and might not be very often. If a business relies on a web scraper working day and night, the breaking of it could cause a large disruption of their services. In such a case the algorithm presented in this thesis might be of interest.

This thesis proposes an algorithm to tackle the issue of creating a robust scraper that can withstand small changes in the HTML and continue scraping without interruption. The algorithm is focused on the HTML-code surrounding the data, and its metadata, to be able to identify the parts of the code which contain the data and extract it. Training the algorithm on solely the current version of the website requires careful use of the data to be able to achieve a high level of accuracy and also puts some restraints on which sites can be scraped. Namely, the site must contain repetitive content. Examples of websites containing repetitive content are; Youtube, in which multiple videos are displayed,

and News sites in which multiple articles are displayed.

As HTML is code in a text format the data has to be formatted into numerical form for the ML-model. The pre-processing of the data is done through text feature extraction and uses the bag-of-words model in conjunction with additional adjustments to create optimal performance. The pre-processed data is input into a Support Vector Machine to classify the data and extract the data which was originally sought-after. The performance of the algorithm is tested on older versions of the websites, from the moment at which the static paths no longer works, to make sure there has been a change of large enough significance to disrupt a static scraper.

There are a large number of variables that affect the performance of the algorithm which are discussed in depth in the thesis. The purpose of this discussion is to give an understanding of the issues with creating a robust scaper and for which sites it might be a viable option. As the research within the constraints of a site-specific scraper has been fairly limited this is a fairly unexplored topic. The thesis therfore provdes a lot of new insight into this topic.

1.1 Background

This chapter of the thesis is intended to; provide an understanding of Web Scraping and its issues, how this thesis explores solving parts of these issues, and a summary of what has been done in the field so far.

1.2 Web Scraping

Web Scraping is the act of programmatically retrieving data from the internet. Web Scraping in its most basic form lacks any sort of generalizing capabilities, when a website is altered, even slightly, the scraper breaks. The use of static paths to retrieve data from a non-static environment is the main culprit. Machine Learning is one of the main tools utilized in trying to achieve a robust scraper. In this thesis robust refers to not susceptible to small changes in the design of the web site. When such a scraper has been created it can, hopefully, continuously scrape a site on the internet for as long as the changes made to the website are relatively small. This can be referred to as a site-specific scraper; it only has to be able to scrape one website, but it must be able to do so when the HTML is changed and with a high level of accuracy. Thus having far less need for maintenance of it as well as allowing for unsupervised scraping.

1.2.1 Web Scraping Fundamentals

The basic principles of Web Scraping are to create a program that in an automated fashion retrieves unstructured data from a website and adds structure to it. This data is often encapsulated as plain text within the HTML of a website, which is the standard language in which websites are written [2].

There are a multitude of ways in which web scraping can be implemented, many of which are discussed in the paper: Web Scraping: State-of-the-Art and Areas of Application [3]. The complexity of these implementations vary widely. The complexity depends mainly on one key factor; how general the web scraping algorithm is. The static nature of scraping results in binary outcomes, either 100 % accuracy, or 0 %. To avoid this, generalization is key.

Listing 1.1: Example of a simple HTML page

```
<html>
    <head>
        <title>Title of page</title>
    </head>
    <body>
        <p>Relevant Data</p>
        <p>Irrelevant Data</p>
    </body>
</html>
```

The commonly used browsers such as; Internet Explorer, Google Chrome, and Firefox interpret the HTML code and display the sites with which we are familiar. The code displayed in listing 1.1 shows the essence of HTML, namely, tags [4]. The tags constrict which type of content is allowed to be written between the opening-and-closing tag. An example would be; only being allowed to write plain text between the paragraph tags, , and . They also create "paths" which can be followed, leading to certain parts of the code. The path to the paragraph with "Relevant Data" can look something like this: $html \rightarrow$ body $\rightarrow p[0]$, this path would lead to the first node with the tag p, containing "Relevant Data". These paths are often defined through CSS selectors [1] or Xpaths [5], these are different ways of navigating the HTML structure. XPath will be commonly used throughout the thesis and stands for XML Path Language. It is a query language used to select nodes in an XML document, in this case the HTML.

The simplest practice right now is to create a site-specific scraper in which

[1] https://www.w3schools.com/cssref/css$_s electors.asp$

a person has identified the parts of HTML in which the sought-after data resides, then creating a program that goes to those exact locations and retrieves it. These exact locations are specified through the aforementioned paths. Retrieving the HTML of a website can be achieved through methods such as HTTP requests, or headless browsers such as puppeteer [2]. Puppeteer provides an easy way of interacting with browsers programmatically, this can, in turn, allow for more data to be collected. For example; scrolling down the page can load additional items into the HTML. This is the easiest way of implementing scraping, however, it has some major flaws.

1.2.2 Web Scraping Flaws

As mentioned in the previous section there are some flaws with the easier ways of implementing scraping. Firstly, it has to be done manually for each site, which might be, depending on the scope, infeasible. Secondly, as soon as the HTML-code for the site changes, the algorithm will break. This is due to the exact path which previously located the wanted element no longer leads to an element. This results in quite a bit of maintenance work as the algorithms need to be updated potentially as often as the website is altered.

1.2.3 Advanced Web Scraping

The flaws mentioned in 1.2.2 have resulted in an interest in developing more advanced scraping algorithms, to reduce the need for fragile and maintenance-heavy scraping algorithms. Machine Learning is often used to create advanced scraping algorithms, as it is well suited for the task of generalizing. The two aspects of scraping with which machine learning can help in solving the thesis are; classification of the text data on the site and recognizing patterns within the HTML structure. When the objective is a site-specific scraper the limited amounts of data are a large factor. The aim is to be able to set this up using only what is found on the page as data for the model. This limitation requires the site to have repetitive structures which are explained in section 1.2.4 to be able to extract enough data for the model to work with, as well as ease the identification of these areas.

[2] https://github.com/puppeteer/puppeteer

1.2.4 HTML Patterns

Even websites which are frequently updated often retain a similar structure in the code which represents the repetitive content (this can be easily verified by comparing the sites:s current version to their older versions [3]). Youtube is a great example of this. Even though the website has been updated a fair amount of times during the years, the content contained within the HTML representing the homepage video has remained the same. Even though the blocks are not identical, there are key components which must be present within them, as displayed in listing 1.2. Video length, title and creator are examples of such attributes in the case of Youtube videos, allowing for much easier identification. These components themselves sometimes contain easily classifiable data. Such as the length of a video which has a fixed format, or data which is hard to classify, such as the title of the video. The title of a Youtube video can be near anything and is thus hard to make classifications upon.

Listing 1.2: Simple Youtube example

```
<html>
    <head>
        <title>Youtube</title>
    </head>
    <body>
        <video>
            <title>Youtube video#1</title>
            <length>10:00</length>
            <Noise>********</Noise>
            <creator>Creator#1</creator>
        </video>
        <video>
            <title>Youtube video#2</title>
            <Noise>********</Noise>
            <length>5:00</length>
            <creator>Creator#2</creator>
            <Noise>********</Noise>
        </video>
    </body>
</html>
```

[3] https://archive.org/web/

1.2.5 HTML Nodes

Nodes in the HTML-code are created each time a new tag is added. The nodes content is what is between the opening and closing tag, thus its children are all the tags added within its body. When looking at 1.2 for example, the "video node" has 4 children: title, length, noise and creator. When HTML-code is parsed with a HTML-parser such as HtmlAgilityPack [6] [4] it is common that these nodes contain a lot of metadata, such as its position in the code, how many children it has, etc. The selection of the nodes within the HTML can be done in various ways, meaning there are multiple languages designed to select these nodes. One of which is Xpath [5], which is used commonly throughout the thesis. Nodes in which the innertext attribute contains text, are the nodes which potentially contain the sought-after data.

1.2.6 Text Feature Extraction

The HTML code as well as the data points which are being scraped come in the form of plain text. Before any type of text classification [7] can be performed, there must be text feature extraction. Because ML-algorithms can only use numerical data as input, the text data retrieved must be converted into numbers, this is known as text feature extraction. Once the text feature extraction has been performed the transformed data can be used as an input for a ML-model. The primary focus of the algorithm will be to classify the text, or code, which surrounds the sought-after data. This code does not contain a large vocabulary as there are a limited amount of HTML tags per page, especially within the block constraint. Due to the fact that the changes to the HTML are assumed to be small, the structure of the code can be assumed to be similar. Focusing on the structure of the HTML has been shown to have higher accuracy than a focus on the plain text of the data which is being scraped [8] especially in the context of data which does not have a specific characteristic. A way of representing the data as numbers is through the bag-of-words model. It is common to first remove all the so-called stop-words from the text, these are words that exist in all texts and thus do not provide any extra information to the classification algorithm. Then the bag-of-words model can be created, the model includes a vocabulary, which consists of all the words within the data. As well as a count of each word within the vocabulary. The vocabulary is represented as a vector in which each index represents a word, this index contains the number of times it occurs in the text. This can then be fed into the ML-model. To

[4] https://html-agility-pack.net/

avoid making the vocabulary to large a maximum amount of features is often used, this limits the bag of words to a set amount of the most common words in the vocabulary thus reducing the size of the vector.

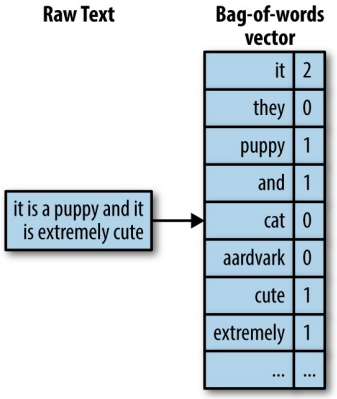

Figure 1.1: A bag of words example

When creating a bag-of-words a so called vectorizer is used, this vectorizer is fit to a corpus from which it extracts all words once. This results in a vector in which each word of the text can be represented as present in a text with a 1, and not present with 0. There are several different ways in which the vectorizer can represent words, discussed at great length in: Vector representations of text data in deep learning [9]. However, the more advanced ways of representing text as data are merely implemented to capture context which is very important in written text. This is not the case in this thesis, as the purpose is to represent text which builds certain paths. Due to the code having a more structured format than normal text the use of the bag-of-words model combined with a vectorizer which instead creates a bag-of-words vector representing a sequence of words, also known as the ngram range, instead of solely one word at a time is ideal. For example, if the ngram range is set to 2, the vectorizer allows for both uni-grams and bi-grams. This allows for keeping the structure of the written code to a certain extent, as it is not unusual for HTML tags to be used multiple times but not as often in the same order. So when looking at the

tags of listing 1.2 a sequence could be [video, title], this would most likely refer to an HTML-node containing the title of a video, instead of just [title] which could refer to perhaps a book title further down in the HTML code. An example of this occurring can be shown in tables 1.1 and 1.2 in which the two sentences are indistinguishable from each other in the case of using only one-word sequences, whereas a difference can be seen when using two-word sequences. This captures more information especially in the context of using it for HTML due to the structural nature of code. To limit the amount the size of the vector retrieved when fitting to a corpus, the number of max features can be adjusted. This can and should be adjusted according to the amount of data available, a lower number of features is most likely better with a smaller amount of data and vice versa.

Sentence/Bag-of-words vector	video	book	title	good
I like this video title, it reminds me of a good book.	1	1	1	1
I like this book title, it reminds me of a good video.	1	1	1	1

Table 1.1: A bag-of-words vector representing a sequence of one word

Sentence/Bag-of-words vector	video, title	I, like	reminds, me	like, this
I like this video title, it reminds me of a good book.	1	1	1	1
I like this book title, it reminds me of a good video.	0	1	1	1

Table 1.2: A bag-of-words vector representing a sequence of two words

To humans, the word "banana" is very similar to the word "bananas". It is to us just a conjugation of the word, however, this is not the case for computers. Thus an additional feature that can be implemented is the use of stemming [10]. Stemming is the act of reducing a word to its most basic form, so stemming "bananas" results in "banana". This is used to try to get slightly altered words to be recognized by the bag-of-words vector, however, its use might be limited.

1.2.7 Support Vector Machines

Support vector machines, SVM for short, are trained through supervised learning, meaning the data on which the model is trained needs to be labeled. It can be used for classification or regression, however, in this thesis is used for classification and will thus only be explained in that context.

An SVM, in a N-dimensional context, classifies data through splitting the N-dimensional space with a hyperplane, such that as many points as possible are classified correctly. If the data is linearly separable, meaning it is possible to separate the data points with a plane and classify them all correctly, it seeks to maximize the margin, the margin is the space between the plane and the nodes closest to the plane. As shown in figure 1.2 there are two ways to separate the data, however, one has a larger margin. A larger margin allows for better classification of outliers as can be seen by the classification of the squares by the different lines.

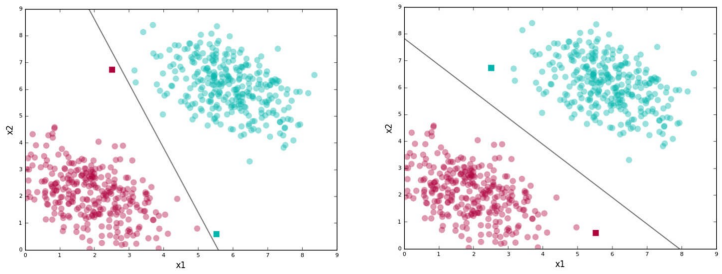

Figure 1.2: An example of optimizing the margin

The data is not always as easily separable as the data in figure 1.2 and demanding 100 % accuracy on the training data can lead to poor performance on test data. Using what is called a soft-margin to allow for a certain amount of miss classification whilst gaining a greater margin might improve accuracy on the training data, this can be especially useful when the data sets have outliers which overlap. In figure 1.3 the parameter C can be used to control the amount of miss classification allowed at the cost/gain of the margin. Examples of how the parameter C affects performance are displayed in figure 1.3 A wider margin is likely to perform better in cases in which overfitting is an issue and vice versa.

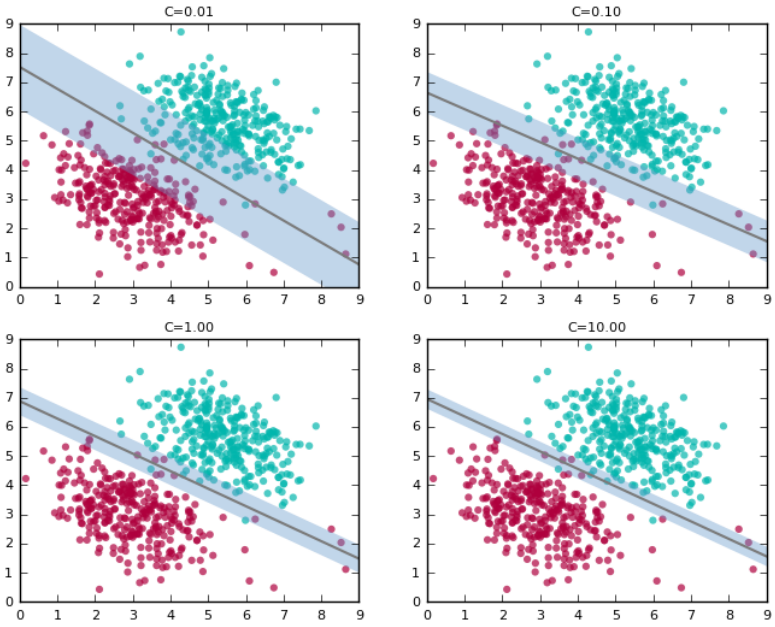

Figure 1.3: An example of the trade-off between a wide margin and a high accuracy on training data

When data is not linearly separable in its current dimensions, a kernel function can be used. The kernel functions purpose is to take two points in the current space and calculate the dot product onto the projected space. If the correct kernel function is applied to the data it can become linearly separable. An example of this is displayed in figure 1.4.

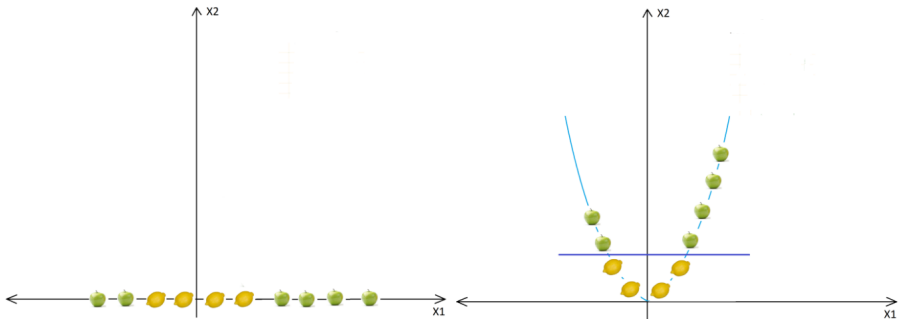

Figure 1.4: Displaying a non-linearly separable data set, on the left, transformed with the kernel function: $f(x) = x^2$ to a linearly separable data set, on the right.

How the coordinates of the points in the new dimension are calculated is dependant on the kernel function. When no projection is used and the dot products are computed in the original space it is called a linear kernel. The linear kernel is advantageous when the data has a large number of features, this is due to it being unlikely that there will be lots of overlap in the feature space,

thus quite likely that the data is linearly separable [11]. It is also much more efficient computationally. Linear Support vector machines have been found to work well for the classification of text [12] [11].

1.3 Ethics of Web Scraping

There are quite a few aspects of scraping that are ethically questionable [13]. A few examples of ethical issues regarding scraping are:

- Aggressively scraping can cause certain sites to slow down or even crash.
- Potentially reducing the value of a site as its information can now be found elsewhere. This redirection of internet traffic might results in a loss of monetization, through for example ads.
- The legal aspects of scraping are still a grey zone, thus can be taken advantage of.

1.4 Purpose

The purpose of this thesis is to answer the following questions:

- Can a web scraper, with the help of machine learning, reliably scrape different types of websites even when small changes are made to the HTML-code of the website?
- If such a web scraper can be created, what are its limitations?

These questions will be answered through proposing and implementing an algorithm that is then tested on various sites to get a good grasp of which issues the algorithm might encounter and why.

1.5 Related Work

The more advanced ML-based scraping algorithms tend to focus on the process of in some way structuring the large amounts of unstructured data available on the internet. In these algorithms the focus lies more on decent accuracy over a large variety of pages [14, 15, 16, 17], rather than a high accuracy on just one. This is not congruent with the goals of this thesis but they do provide a lot of inspiration as to how data on the websites can be utilized. The authors

of the paper: Classification of Web Site by Naive-Bayes and Convolutional Neural Networks [18] train a deep learner to classify whether a page might pose a threat to users who are not familiar with the world wide web. It does this by detecting whether or not personal information is requested on the site.

The paper which has achieved the goal of this thesis is; Classification-Based Adaptive Web Scraper [19]. They use a technique in which they identify reoccurring blocks of code in the HTML, which must contain certain predefined attributes. These blocks along with the rest of the HTML can be used as data for supervised learning. They managed to achieve near-perfect accuracy when training on the current version of a site then testing on an older version of that site. The issue with their paper [19] is the fact that they do not test the performance of the model on other websites, thus having a high risk of achieving results that may not be reproducible on other sites. They also do not go into detail of the limitations of their algorithm because the site on which tests were done most likely is very well suited for their algorithm and not running into issues that might be present elsewhere.

Chapter 2

Methods

This chapter aims to provide a clear picture of how the data which is used for training can be retrieved, how it is formatted, and the restrictions to which web pages can be scraped.

2.1 Websites suitable for the algorithm

The amount of data that can be collected is highly dependant on the specific site for which the goal is to create a robust scraper, as this is the only source of data. This results in having to take the following limitations into consideration when searching for a website on which the algorithm can be implemented:

- Not all types of websites are suitable for the scraping algorithm implemented in this thesis, they must have similar repetitive content.
- The website must have its past versions available on WaybackMachine. The algorithm can be tested on altered versions of the website retrieved from the archive.
- The accuracy of the algorithm can be affected by the amount of data that can be retrieved from the web page, how many blocks are available.
- The ML-model must not be exposed to more data than needed when performing classification to maximize accuracy, minimizing the amount of data it has to classify which is not within the blocks.

An additional limitation is that not all website owners want them to be scraped. It is quite common for a website to incorporate a robots.txt file [20]. The robots file is a web standard text file containing instructions that essentially

inform you about the scraping behavior which will be tolerated by the hosts of the site. The file often contains instructions on which parts of the site are allowed to be accessed as well as giving a rate at which the site can be scraped to avoid straining their servers. It is considered proper scraping practice to follow these rules but they are merely guidelines and it is up to the host of the page to punish the breaking of the rules. Punishment is often given through barring the IP-address of the offender from using the site.

2.2 Retrieving the HTML

Retrieving HTML code is done in Visual Studio, in c# with the help of a headless browser called puppeteer[1]. The headless browser must be instantiated, then given a URL to which it can navigate. Once the browser has arrived at the page a method can be called which retrieves the HTML which is then written to a text file for easy access. The reason for the use of a headless browser is that it allows for easy retrieval of the HTML as well as navigation of the page. The navigation is useful when for example, scrolling down on YouTube to create a larger batch of training data as it results in a larger number of videos being loaded into the HTML. Retrieving the HTML can also be done completely manually by opening the website in google chrome, right-click, and press inspect. This action will display the underlying HTML of the site which in turn can be copied and pasted into a text file. The headless browser still has the advantage of being able to automatically retrieving data from multiple websites and thus reducing the amount of manual labor needed.

2.3 Extracting relevant Data from the HTML

To easily be able to traverse the HTML text, it is loaded into an HTML-parser [2], this allows for the different nodes of the HTML to be represented as objects from which data can be extracted as well as navigate the HTML structure easily. The object contains mostly metadata, for example: which its parent nodes are, its location in the code, its child nodes, its text content.

To extract relevant data from the parsed HTML two components are needed; the XPath of the repetitive blocks as well as the names of the nodes containing the data which the algorithm is intended to scrape and classify. This must be done manually by looking through the actual HTML of the website and

[1] https://github.com/puppeteer/puppeteer

[2] https://html-agility-pack.net/

identifying an XPath that targets all the blocks from which the data can be extracted.

Once the blocks are extracted, the nodes which are present within all the blocks are retrieved. This is done by taking all the nodes in the first block and iteratively comparing it to all the other blocks, removing the nodes which they do not have in common. These nodes must contain the data which is sought-after. When iterating over the content of the blocks, each node has its metadata appended to a string and is labeled. Everything which is labeled 0 are nodes that are not a part of the wanted nodes and the others are labeled 1.

The data on which the trained model is tested is retrieved from the web archive called WaybackMachine [3] through the same process mentioned in 2.2. However, as the point is to never have seen this site the use of XPaths is no longer an alternative. Instead, the algorithm simply counts the number of times a node occurs within the HTML and if it occurs more than a set number of times, it is included. The number of times is specific to a site and the aim is to get rid of as much data as possible which is not contained within the blocks. This must still be general enough to not potentially exclude the blocks from future websites, so it is better to choose this number conservatively.

2.4 Relevant Metadata

The HTML nodes from which the training data is created uses metadata retrieved by the parsing of the HTML turning nodes into objects. This metadata is what is used to create a picture of the structure of the HTML code surrounding and inside of the node. This picture is created through choosing different parameters from the metadata which are deemed as general enough to not allow overfitting, but specific enough to differentiate between the other nodes in the block. When choosing the parameters it is generally best to look at the specific website structure. For example; the node names are very specific to the data and are not present anywhere else in the HTML, this could allow for overfitting as the algorithm might only classify nodes correctly by their names. Whereas these might not be the exact names on future versions of the website. On the other hand, having node names that are present everywhere in the HTML does not allow the algorithm to detect differences at all. This is unique for all sites and can be taken into account on a case by case basis.

[3] https://archive.org/web/

The following parameters are used for the algorithm in the experiments performed in this thesis:

- The name of the node.
- The name of the parent of the node.
- The XPath of the node.
- The classes of the node, if any.
- The amount children of the node, if any.
- The length of the text of the node.

These parameters are picked to combine the structure of the HTML with the names of the nodes to give the algorithm insight of both aspects and hopefully balance bias and variance.

2.5 Data Formatting

All the data extracted is text, for the ML-model to be able to interpret it the text must be converted into numerical form. This is done with the help of the python library sklearns CountVectorizer [4]. The CountVectorizers fit and transform method is given a corpus of data, which is the metadata of the HTML nodes which are being scraped. This creates a CountVectorizer which can be fitted to a piece of text and return a bag-of-words vector, as displayed in figure 1.1.

The parameters which are used and altered to create a CountVectorizer suited for a specific webpage are; max features and ngram range explained in section 1.2.6. The CountVectorizer is given these two parameters when it is being fit to the corpus, to adjust the dimensions of the bag-of-words model which is output. The best values for these parameters are decided by iterating over different values and looking at the accuracy for each specific site.

The different parameters, shown in section 2.4, are transformed with the CountVectorizer creating a vector for each parameter. These vectors are then appended to each other, one for each node extracted from the HTML. This creates a matrix of data, which has as many rows as there are nodes and the number of parameters times the number of features columns.

[4] https://scikit-learn.org/stable/modules/generated/sklearn.feature$_extraction.text.CountVectorizer.html$

2.6 Support Vector Machine

The machine learning model which receives the formatted data as an input is sklearns LinearSVC [5]. The SVM is set to converge for 5000 iterations, has a tolerance of $1e^{-}5$ which is referred to as the parameter C in section 1.2.7 and its random state set to 0.

After the ML-model has been trained on the entirety of the data received from the original website, the model is given the extracted data from the past versions, classifying all of the nodes as either one or zero, one being the node which is supposed to contain the sought-after data. The data on which the algorithm is tested is from past sites and thus not labeled. The nodes classified as 1 must be manually interpreted and identify whether or not they contain the correct nodes. This is done through identifying the node name which is deemed to be the original node from the current-day website.

[5] https://scikit-learn.org/stable/modules/generated/sklearn.svm.LinearSVC.html

Chapter 3

Evaluation

This section is intended to deliver a clear picture of what has been achieved by the algorithm put in place. As well as how the algorithm is evaluated.

3.1 Assessing the performance of the algorithm

Setting up tests to measure performance requires the following:

- A way to extract HTML from the targeted website, which preferably contains repetitive structures for good results.
- Identification of the Xpath to a block of code containing the sought-after data, by looking at the HTML of the website.
- Identification of the name of the node within that block containing the data, by looking at the HTML of the website.
- Setting the number of repetitions in which one might expect repetitive content to appear.
- Useful, but not necessary, to implement a filter, which filters the node's inner text attribute based on what the data commonly looks like. This is done in order to minimize the amount of data which has to be classified.

Once this has been done the algorithm can be run, the output can be filtered by nodes that have been classified as the node containing data. The nodes classified as 1, must further be filtered by a unique identifier for the node, to evaluate how many of the nodes labeled as 1 were the nodes from which the data is supposed to be extracted.

3.2 Reproducing the tests

Depending on which node a person determines to be the correct node the accuracy might differ, as this is done manually. However, using the same parameters as well as the same identification method for the correct node will provide identical results.

3.3 Measurement of results

To get a clear picture of how well the model performs a confusion matrix will be used. The confusion matrix is used as opposed to solely accuracy due to it showcasing algorithms performance in terms of how many of the nodes labeled 1 are correct as well as the overall accuracy.

The websites in the results have been chosen in line with the limitations mentioned in section 2.1. The confusion matrix will be calculated from the point of the XPath breaking too, 1 month at a time, 3 months back in time. This is to give a picture of the algorithm's performance over extended periods of time. The scraping will be done based on one data point in the block and the performance will be measured by the algorithm's ability to classify that node. The graphs will display two lines; the normal line looks at the overall accuracy of the model whereas the dotted line gives better insight into how much of the data labeled as 1 is the correct data, also know as the sensitivity of the algorithm [21]. The normal line is calculated by:

$$Accuracy = \frac{TruePositives + TrueNegatives}{Total} \tag{3.1}$$

and the dotted line is calculated by:

$$Sensitivity = \frac{TruePositives}{TruePositives + FalsePositives}. \tag{3.2}$$

3.4 Results

3.4.1 Fox News

To determine which sequence length is optimal for performance the algorithm is ran with 4 different sequence lengths and the number of features set to 15. The graph 3.1 shows that sequence length: 1 provides great sensitivity, but poor accuracy. The lines quickly flatten after that and performance differences are very small.

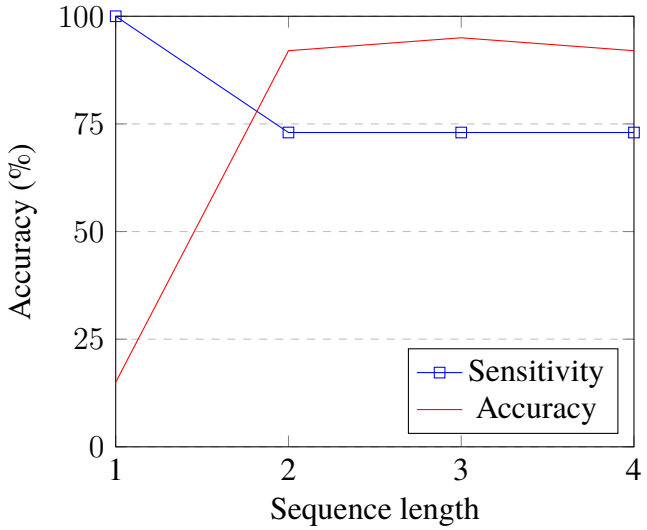

Figure 3.1: This graph displays the accuracy at which the algorithm correctly labels the article node, from Fox News HTML on 2019-12-14, as 1, with 4 different sequence lengths and the number of features set to 15. This is to determine which sequence length is optimal.

To determine which number of features is optimal for performance the algorithm is ran with 4 different numbers of features and the sequence length set to 3, which achieved the best performance in the previous test. The graph 3.1 shows that a number of features of 15 drastically improves performance, with a steep drop after.

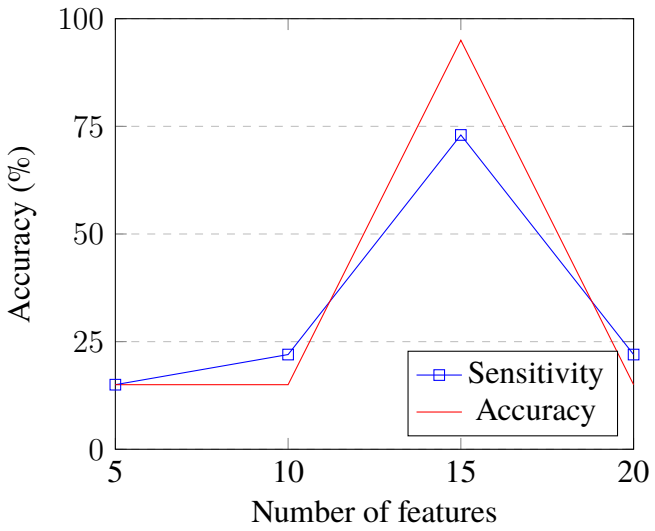

Figure 3.2: This graph displays the accuracy at which the algorithm correctly labels the article node, from Fox News HTML on 2019-12-14, as 1, with 4 different numbers of features and the sequence length set to 3. This is to determine which number of features is optimal.

The graph 3.3 displays the performance of the algorithm with the parameters providing the best results from the previous trials. This is tested on 3 different versions of the site retrieved 1 month apart from each other, after the XPath on the original site broke. This is done to asses for how long the scraping algorithm can provide decent results.

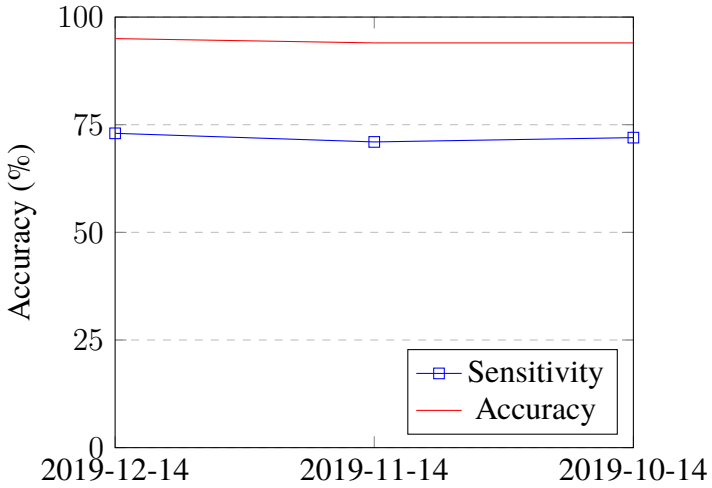

Figure 3.3: This graph displays the accuracy at which the algorithm correctly labels the article node, from Fox News HTML on 3 different dates, as 1, with a sequence length of 3 and the number of features set to 15. This is done to measure the performance of the algorithm with optimal features.

This matrix displays the exact numbers from which the graph 3.3 is calcu-

lated, to provide context as to how many nodes were involved.

Fox News	2020/05/10	2019/12/14	2019/11/14	2019/10/14
True Positive	276	514	466	484
False Positive	0	192	188	190
True Negative	828	2806	2614	2640
False Negative	0	0	0	0
Total	1104	3512	3268	3314

Table 3.1: Confusion matrix displaying the results achieved by the algorithm, to get an understanding of the amount of nodes involved and give context to the graphs. Scraping news titles from Fox News on 4 different dates, with a word sequence set to 3 and a number of features of 15.

The following vector represents the bag-of-words vector extracted from the Fox News corpus, this provides greater insight into what the algorithm uses to classify the HTML nodes and why it might struggle or perform well. The number displayed is the index in the vector and the order is from most to least common.

'article': 0, 'div': 3, 'html': 11, 'header': 10, 'article div': 1,
'html body': 12, 'div div': 5, 'div section': 9, 'section div': 14,
'div article': 4, 'html body div': 13, 'body div div': 2, 'div div div': 7,
'div div section': 8, 'div div article': 6'

Table 3.2: The bag-of-words vector extracted from the Fox News corpus, to be able to understand what it has deemed the structure of the nodes HTML looks like. Displayed in the order of how commonly they occur.

3.4.2 Youtube

To determine which sequence length (ngram-range)Vector representations of text data in deep learning is optimal for performance the algorithm is ran with 4 different sequence lengths and the number of features set to 15. The graph 3.4 shows how the algorithm is practically unaffected by changes in the sequence length.

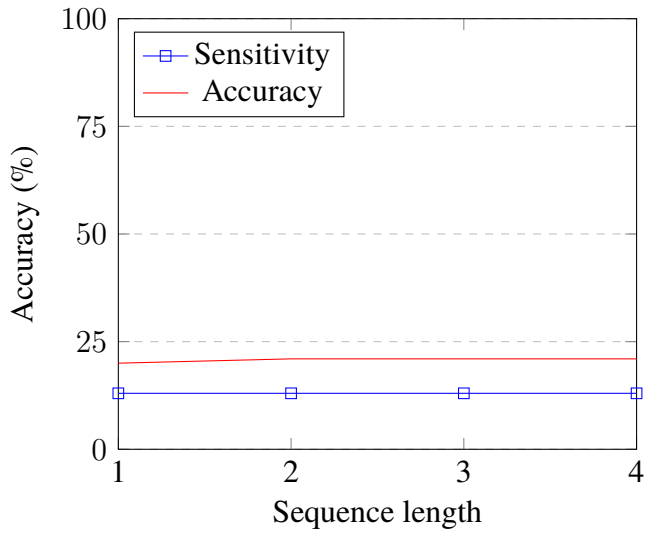

Figure 3.4: This graph displays the accuracy at which the algorithm correctly labels the Video Title node, from Youtube HTML on 2020-03-03, as 1, with 4 different sequence lengths and the number of features set to 15. This is to determine which sequence length is optimal.

To determine which number of features is optimal for performance the algorithm is ran with 4 different numbers of features and the sequence length set to 3, which achieved the best performance in the previous test. The graph 3.5 shows a slight increase in performance with the number of features set to 20.

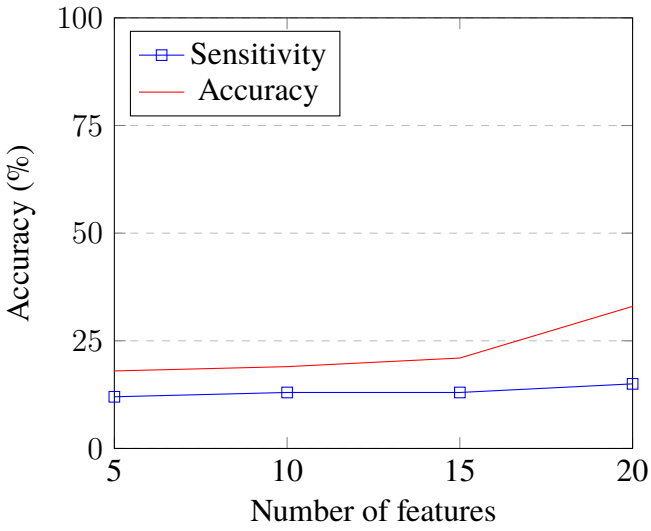

Figure 3.5: This graph displays the accuracy at which the algorithm correctly labels the Video Title node, from Youtube HTML on 2020-03-03, as 1, with 4 different numbers of features and the sequence length set to 3. This is to determine which number of features is optimal.

The graph 3.6 displays the performance of the algorithm with the parameters providing the best results from the previous trials. This is tested on 3 different versions of the site retrieved 1 month apart from each other, after the XPath on the original site broke. This is done to asses for how long the scraping algorithm can provide decent results.

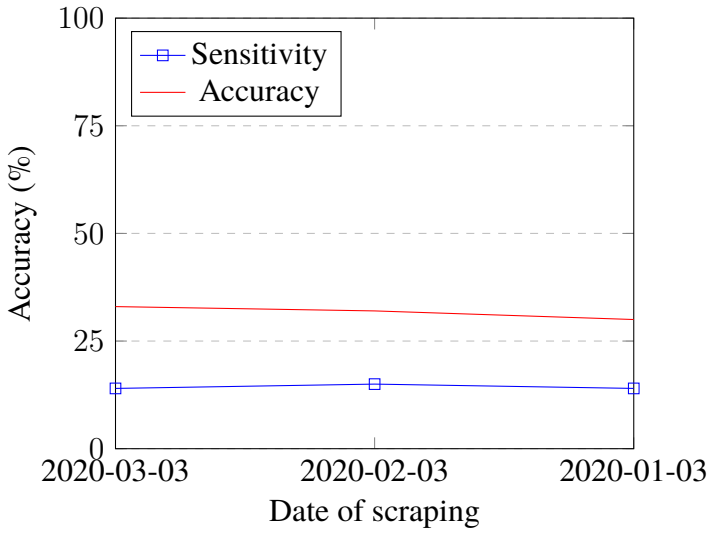

Figure 3.6: This graph displays the accuracy at which the algorithm correctly labels the Video Title node, from Youtube HTML on 3 different dates, as 1, with a sequence length of 3 and the number of features set to 20. This is done to measure the performance of the algorithm with optimal features.

This matrix displays the exact numbers from which the graph 3.6 is calculated, to provide context as to how many nodes were involved.

Youtube	2020/05/10	2020/03/03	2020/02/03	2020/01/03
True Positive	30	120	130	146
False Positive	0	820	918	1022
True Negative	540	94	84	82
False Negative	0	0	0	0
Total	570	1034	1132	1250

Table 3.3: Confusion matrix displaying the results achieved by the algorithm, to get an understanding of the amount of nodes involved and give context to the graphs. Scraping video titles from Youtube on 4 different dates, with a word sequence set to 3 and a number of features of 20.

This vector represents the bag-of-words vector extracted from the Fox News corpus, this provides greater insight into what the algorithm uses to make classifications of nodes and why it might struggle or perform well. The number displayed is the index in the vector and the order is from most to least common.

’ytd’: 16, ’rich’: 10, ’item’: 5, ’renderer’: 7, ’div’: 1, ’browse’: 0,
’grid’: 4, ’video’: 15, ’ytd rich’: 17, ’rich item’: 13, ’item renderer’: 6,
’renderer div’: 8, ’div ytd’: 2, ’rich grid’: 11, ’ytd rich item’: 19,
’rich item renderer’: 14, ’renderer div ytd’: 9, ’div ytd rich’: 3,
’ytd rich grid’: 18, ’rich grid video’: 12

Table 3.4: The bag-of-words vector extracted from the Youtube corpus, to be able to understand what it has deemed the structure of the nodes HTML looks like.. Displayed in the order of how commonly they occur.

3.4.3 Yahoo

The graph 3.7 displays the performance of the algorithm with the parameters providing the best results from the previous trials. This is tested on 3 different versions of the site retrieved 1 month apart from each other, after the XPath on the original site broke. This is done to asses for how long the scraping algorithm can provide decent results.

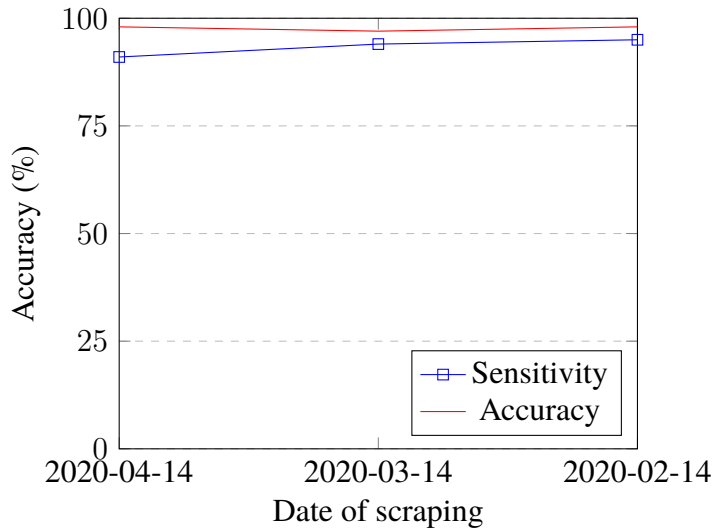

Figure 3.7: This graph displays the accuracy at which the algorithm correctly labels the article node, from Yahoo News HTML on 3 different dates, as 1, with a sequence length of 3 and number of features set to 15. This is done to measure the performance of the algorithm with optimal features.

This matrix displays the exact numbers from which the graph 3.7 is calculated, to provide context as to how many nodes were involved.

Yahoo	2020/05/10	2020/01/31	2019/12/31	2019/11/31
True Positive	140	522	762	994
False Positive	16	50	46	50
True Negative	691	1906	872	1586
False Negative	0	0	6	6
Total	847	2478	1680	2630

Table 3.5: Confusion matrix displaying the results achieved by the algorithm, to get an understanding of the amount of nodes involved and give context to the graphs. Scraping news titles from Yahoo News on 4 different dates, with a word sequence set to 3 and a number of features of 15.

This vector represents the bag-of-words vector extracted from the Yahoo News corpus, this provides greater insight into what the algorithm uses to classify nodes and why it might struggle or perform well. The number displayed is the index in the vector and the order is from most to least common.

'li': 11, 'ul': 13, 'html': 8, 'body': 0, 'div': 3, 'px': 12,
'html body': 9, 'body div': 1, 'div div': 4, 'div ul': 6, 'ul li': 14,
'html body div': 10, 'body div div': 2, 'div div div': 5, 'div ul li': 7

Table 3.6: The bag-of-words vector extracted from the Yahoo corpus, to be able to understand what it has deemed the structure of the nodes HTML looks like. Displayed in the order of how commonly they occur.

Chapter 4

Discussions and Conclusions

This chapter aims to discuss the results achieved by the algorithm as well as the flaws with the results.

Yahoo

The graph 3.7 shows excellent performance, this is likely because only a very minor part of the HTML code changed, this resulted in breaking the Xpath but did not alter the code in any significant way, thus providing excellent results. The differences between the two blocks of HTML code can be seen in listings; 4.1 and 4.2 in which the only node which has changed name is the first node, thus it has optimal conditions for high accuracy. When looking at the bag-of-words vector extracted from the code, the notable part is how it mostly consists of general HTML tags. This is of interest due to its generalizing capabilities when the code-structure remains intact.

Listing 4.1: Simplified HTML of an article from Yahoo News on 2020-05-10

```
<li class="js-stream-content Pos(r)">
    <div class="Py(14px) Pos(r)">
        <div class="Cf">
            <div class="Fl(start) Pos(r) Mt(2px)">
            <div class="Ov(h) Pend(44px) Pstart(25px)">
                <h3 class="Mb(5px)">
                <a href="LINK">
                    <u class="StretchedBox">...</u>
                        TEXT
                </a>
                </h3>
            </div>
            </div>
        </div>
    </div
</li>
```

Listing 4.2: Simplified HTML of an article from Yahoo News on 2020-04-14 with the difference in code highlighted from the previous listing

```
<li class="js-stream-content">
    <div class="Py(14px) Pos(r)">
        <div class="Cf">
            <div class="Fl(start) Pos(r) Mt(2px)">
            <div class="Ov(h) Pend(44px) Pstart(25px)">
                <h3 class="Mb(5px)">
                <a href="LINK">
                    <u class="StretchedBox">...</u>
                        TEXT
                </a>
                </h3>
            </div>
            </div>
        </div>
    </div
</li>
```

Youtube

The graph 3.6 showed the worst performance, this is most likely because the HTML code was altered quite significantly, which can be seen in the comparison between listings: 4.3 and 4.4. This drastic changed not only the structure of the HTML but also the variables themselves. What is also a contributing factor to the poor performance is the very specific bag-of-words vector, these tags are not very general and thus focus less on the structure of the code and more on the tags themselves. When comparing the extracted words from the vector to the HTML it is apparent that none of the words are present. The graphs: 3.4 and 3.5 show practically no changes when the vector is altered, this is because since the bag-of-words vector will essentially be empty when used as an input, apart from its text features, the algorithm cannot distinguish between nodes. It can only classify based upon the text length and the children count, this does not change and thus the performance is fairly identical throughout the graph.

Listing 4.3: Simplified HTML of a video from Youtube on 2020-05-10

```
<ytd-rich-item-renderer class="style-scope ytd-rich-grid-renderer"
    <div id="content" class="style-scope ytd-rich-item-renderer">
    <ytd-rich-grid-video-renderer class="style-scope ytd-rich-item
    <div id="dismissable" class="style-scope ytd-rich-item rendere
        <ytd-thumbnail class="style-scope ytd-rich-grid-video rend
        <div id="details" class="style-scope ytd-rich-grid-video-r
            <a id="avatar-link" "class="yt-simple-endpoint style-s
            <div id="meta" class="style-scope ytd-rich-grid-video-
                <h3 class="style-scope ytd-rich-grid-video-rendere
                    <ytd-badge-supported-renderer class="style-sco
                <a id="video-title-link class="yt-simple-endpoint
                    <yt-formatted-string id="video-title" class="s
                </a>
                </h3>
            <ytd-video-meta-block class="grid style-scope ytd-rich
            <ytd-badge-supported-renderer class="video-badge style
            <div id="buttons" class="style-scope ytd-rich-grid-vid
            </div>
        <div id="menu" class="style-scope ytd-rich-grid-video-rend
        </div>
    </div>
    <div id="dismissed" class="style-scope ytd-rich-grid-video-ren
    </ytd-rich-grid-video-renderer>
    </div>
</ytd-rich-grid-video-renderer>
```

Listing 4.4: Simplified HTML of a video from Youtube on 2020-03-03

```
<li class="yt-shelf-grid-item yt-uix-shelfslider-item">
    <div class="yt-lockup yt-lockup-grid yt-lockup-video
        <div class="yt-lockup-content">
            <h3 class="yt-lockup-title ">
                <a href="LINK">
                    Text Content
                </a>
                <span>class="accessible-description">..
            </h3>
            <div class="yt-lockup-byline yt-ui-ellipsis
            <div class="yt-lockup-meta ">...</div>
        </div>
    </div>
</li>
```

Fox News

The sequence graph 3.1 of Fox News shows how as the sequence length is 1, there is very little for the ML-model to go on, and thus it classifies very few nodes as the correct one, this leads to very high sensitivity in its classifications. However, there are a lot of nodes that are miss-classified due to its very narrow scope of recognizing nodes correctly. The graph testing different numbers of features 3.2 shows that the number of features that receives the best results is 15, this was true for Yahoo as well. Because of the fact that the number of features are incrementally increased by 5, there appears to be a very large spike in performance. This is most likely smoothed out when testing more numbers of features. Due to the tags being found across most of the HTML as seen especially in the Yahoo bag-of-words shown in table 3.6, the number of features cannot be too low, in that case, it will not be able to capture the architecture surrounding the node and the node itself, thus likely overfitting to the primary node and not considering the context of the node. When there are too many features the issue of capturing too much of the surroundings occurs, allowing for too many features to be captured, creating overfitting of the surroundings of the node, no longer being able to detect the node in new contexts. Due to the way the corpus for the vector is created the number of nodes are limited.

The performance of the algorithm on Fox News is somewhat low consid-

ering the very small changes in the HTML, where title changes its name to title-color-default as can be seen in the simplified codes from listings: 4.5 and 4.6.

Listing 4.5: Simplified HTML of a article story on Fox News from 2020-05-10

```
<article class="article">
    <div class="info">
        <header class="info header">
            <h2 class="title title-color-default">
                <a href="LINK">
                    Text Content
                </a>
            </h2>
        </header>
    <div class="meta">...</div>
    </div>
    <div class="m headshot">...</div>
</article>
```

Listing 4.6: Simplified HTML of an article on Fox News from 2019-12-14 with the difference in code highlighted from the previous listing

```
<article class="article">
    <div class="info">
        <header class="info header">
            <h2 class="title">
                <a href="LINK">
                    Text Content
                </a>
            </h2>
        </header>
    <div class="meta">...</div>
    </div>
    <div class="m headshot">...</div>
</article>
```

An explanation for this is the fact that the algorithm is solely trained on the article blocks, and the classification of these are determined to be the correct node classified, however, there are also article story blocks, which are the blocks containing the code for the headlines. These are very similar to the article blocks, as shown in listing 4.7, and thus get classified as such as well by

the algorithm. The accuracy when including these articles much more reflects what one would expect from such a minor change in the HTML.

Listing 4.7: Simplified HTML of a article story on Fox News from 2019-12-14

```
<article class="article story">
    <div class="m">...</div>
    <div class="info">
        <header class="info header">
            <div class="meta">...</div>
            <h2 class="title">
                <a href="LINK">
                    Text Content
                </a>
            </h2>
        </header>
    </div>
    <div class="m headshot">...</div>
</article>
```

4.1 Limitations

There are some issues with the calculation of accuracy which in turn may boost or decrease the accuracy shown in the tables in 3.4. The only labeled data is the training data retrieved from the original page, the other versions are not labeled as this requires the tedious process of finding an Xpath which leads to the sought-after data for each version of the website. Thus solely the name of the wanted node was used to filter the results. Which depending on the website may or may not be a feasible task as not all sites have a name for the node which is unique. For example; FoxNews labels the HTML nodes sought-after as "article" which makes it easy to distinguish from the rest. Whereas some other sites merely label them as "#text" which is a very common name.

Many nodes can contain the sought-after data, due to the inner text attribute of nodes returning the text of itself and all of its descendant nodes. This means that even though the algorithm might not only classify the "correct" node as 1, the algorithm can still correctly scrape the sought-after data. The accuracy of the scraper might not directly reflect how well it might work in a practical scenario.

4.2 Future work and Improvements

The fine-tuning of the algorithm is highly website specific and using past versions of websites can also be implemented as a way to increase the amount of training data. This does, however, require more manual labor which might not always be worth the work when considering the main purpose of the algorithm is to reduce the burden of updating the algorithm every once in a while. In the context of setting up a scraper that does not break and can run no matter what, the increased labor might be worth the results.

The algorithm working lies a lot in the hands of the creator of the website, as no matter how much past data it is trained on; if the web site is altered to a large enough extent it will most likely break. The reason for this is that the simplicity of the algorithm is also its drawback, as is often the case. It looks at the words of which the HTML is created and finds patterns of certain words existing within certain parts of the HTML. If these words are removed, the algorithm no longer works. However, a large altering of the code is not very common from my experience.

A thing to consider is the fact that the words, all though they might be similar, which are altered slightly cause issues for the algorithm and are no longer part of the bag-of-words vocabulary. Even when using stemming these are commonly not words in the normal sense of a word, they are more similar to variable names. These variables might be similarly named due to a multitude of different reasons such as; the same person writing the code, company policies to ease the rewriting of the HTML, etc. Simply put, there are many cases in which stemming just does not cut it. This is were fuzzy matching [22] might be of good use, it allows for the matching of strings which a human would consider similar. This is done through a mathematical formula that weighs the distance between letters in a word, to get a percentage of how similar the words are.

The interesting part of being able to backtest the model on past versions of the site as well as perhaps include past sites in the training data is the fact that the algorithm can be fine-tuned for the specific website based upon the performance upon the past sites. If the past websites are any indication of how the future of the site will be structured, and from my experience it is, It is a good idea to use the algorithm with the past sites in mind. This is, however, not always easy, the web archive contains a vast amount of old versions of websites, but not all. If it exists it is almost always solely the home page of the site, and the other pages cannot be accessed. Providing an additional hindrance.

What stood out when searching for websites that were appropriate for in-

cluding in the results, is how seldom the sites were updated, depending on the type of site. The news sites seemed to be practically stagnant in the HTML, some of which had not changed for years. Whereas sites such as youtube seemed to change quite frequently, still not more often than once every few months though. This leads me to conclude that sites that are altered often seem to be the ones that have to stay at the forefront of their industry by creating an experience that is as good as possible. The age-old news sites have one job, and there are not many ways in which the delivering news can be revolutionized. At least do current news sites not seem to think that is the case.

The same website developed by different people is as dissimilar as the same program being written by different people. These dissimilarities have a pretty big impact on accuracy. Websites that have node names that are very clear and different from the rest greatly improve accuracy versus the ones where everything is labeled very similarly. These types of websites demand more overfitting to produce good accuracy and are thus less likely to be able to adapt to future websites.

When finding pages suitable for this algorithm there were a majority of pages that had to be ruled out, after taking into account that there had to be a repetitive structure within the page, due to mainly two things. Firstly, the Xpath did not stop working even when looking several years back on the versions of the web page. These pages are not very interesting as putting in the effort to create a scraper tolerant to changes seems somewhat useless when it is updated very infrequently. Secondly, the page simply had HTML code that seemed to be randomly generated and had no room at all for extracting words which could provide a way of identifying certain parts of the code.

Since the data which is being classified is scraped in such a way that it simply extracts all repetitive content on the site the amount of data can vary widely. If what is being scraped is amongst one of few nodes repeated the classifier will have an easier time performing classification, the opposite will be true when there is more repetitive content thus more data to look at.

Another avenue that could be explored is the use of data of what is being sought-after, this would very likely greatly enhance the performance of the algorithm. In the tests performed in this thesis, the scraped information was very dynamic, such as titles. Titles generally have very little in common with each other and to find any similarity the dataset would probably have to be much larger. If the goal was to extract for example social security numbers one could on top of the primary classification of structural data classify the text content extracted. This would likely result in high accuracy.

This algorithm is constricted to and relies heavily on the data being a part

of the inner text of a node. This seems to often be the case, however, there are cases in which the data is retrieved from a database whilst clicking on a button for instance. This requires more advanced scraping methods and as such is not within this particular algorithms reach.

4.3 Ethical and Social Impacts of the Algorithm

The ethical aspects of this project rely primarily on decreasing the need for visiting a certain website by providing the information elsewhere. This can potentially create issues with the livelihood of website owners and the person deploying the algorithm gaining monetarily from that instead. However, as long as the wishes of website owners are followed this will not be an issue.

The impacts of being able to extract and organize large amounts of data from the internet could have benefits as well as drawbacks. One could consider third parties scraping facebook, gaining a huge amount of knowledge regarding people's personal lives, which could not be done manually. This data could be used with malicious intent, such as trying to track down people with certain opinions, etc. The upside is tapping into a vast amount of data, which could be used for scraping medical forums and performing sentiment analysis on the data to extract valuable knowledge about the side-effects of drugs [23].

4.4 Conclusions

The algorithm proposed, although having a limited number of websites with which to work, may be useful in niche situations. For certain websites with certain characteristics the algorithm can be implemented to create a scraper which is unaffected by small changes in the HTML. A robust web scraper. It is quite simple yet provides decent results. There are a lot of different issues that can be run into when constructing this type of algorithm, but primarily the HTML-code is the main factor that determines the performance. The primary use case for this type of algorithm is for websites with repetitive content in which a business model relies on being able to consistently scrape a website.

Bibliography

[1] Wadzani Gadzama, Bitrus Joseph, and Ngubdo Aduwamai. "Global Smartphone Ownership, Internet Usage And Their Impacts On Humans". In: (Sept. 2019).

[2] M. Jazayeri. "Some Trends in Web Application Development". In: *Future of Software Engineering (FOSE '07)*. May 2007, pp. 199–213. DOI: 10.1109/FOSE.2007.26.

[3] R. Diouf et al. "Web Scraping: State-of-the-Art and Areas of Application". In: *2019 IEEE International Conference on Big Data (Big Data)*. Dec. 2019, pp. 6040–6042. DOI: 10.1109/BigData47090.2019.9005594.

[4] Ankush Sharma and Aakanksha Sharma. "Introduction to HTML (Hyper Text Markup Language) - A Review Paper". In: *International Journal of Science and Research (IJSR)* 7 (May 2018), pp. 1337–1339.

[5] Tadeusz Pankowski. "Processing XPath Expressions in Relational Databases". In: vol. 2932. Jan. 2004, pp. 265–276. DOI: 10.1007/978-3-540-24618-3_23.

[6] Erdinç Uzun et al. "EVALUATION OF HAP, ANGLESHARP AND HTMLDOCUMENT IN WEB CONTENT EXTRACTION". In: Nov. 2017.

[7] Armand Joulin et al. *Bag of Tricks for Efficient Text Classification*. 2016. arXiv: 1607.01759 [cs.CL].

[8] Yiming Yang, Seán Slattery, and Rayid Ghani. "A Study of Approaches to Hypertext Categorization". In: *Journal of Intelligent Information Systems* 18 (Mar. 2002), pp. 219–241. DOI: 10.1023/A:1013685612819.

[9] Karol Grzegorczyk. *Vector representations of text data in deep learning*. 2019. arXiv: 1901.01695 [cs.CL].

[10] F. Harrag, E. El-Qawasmah, and A. M. S. Al-Salman. “Stemming as a feature reduction technique for Arabic Text Categorization”. In: *2011 10th International Symposium on Programming and Systems*. 2011, pp. 128–133.

[11] Thorsten Joachims. “Text categorization with Support Vector Machines: Learning with many relevant features”. In: *Machine Learning: ECML-98*. Ed. by Claire Nédellec and Céline Rouveirol. Berlin, Heidelberg: Springer Berlin Heidelberg, 1998, pp. 137–142. ISBN: 978-3-540-69781-7.

[12] M Ikonomakis, Sotiris Kotsiantis, and V Tampakas. “Text classification using machine learning techniques.” In: *WSEAS transactions on computers* 4.8 (2005), pp. 966–974.

[13] Vlad Krotov and Leiser Silva. “Legality and Ethics of Web Scraping”. In: Sept. 2018.

[14] Fabio Ciravegna et al. “Learning to Harvest Information for the Semantic Web”. In: May 2004, pp. 312–326. DOI: 10.1007/978-3-540-25956-5_22.

[15] Anna Gentile et al. “Unsupervised Wrapper Induction using Linked Data”. In: June 2013. DOI: 10.1145/2479832.2479845.

[16] Y. Chu et al. “Automatic data extraction of websites using data path matching and alignment”. In: *2015 Fifth International Conference on Digital Information Processing and Communications (ICDIPC)*. Oct. 2015, pp. 60–64. DOI: 10.1109/ICDIPC.2015.7323006.

[17] Giulio Barcaroli et al. “Use of web scraping and text mining techniques in the Istat survey on "Information and Communication Technology in enterprises"”. In: June 2014.

[18] Xueyan Liu and Ryuya Uda. “Classification of Web Site by Naive-Bayes and Convolutional Neural Networks”. In: *Proceedings of the 12th International Conference on Ubiquitous Information Management and Communication*. IMCOM ’18. Langkawi, Malaysia: Association for Computing Machinery, 2018. ISBN: 9781450363853. DOI: 10.1145/3164541.3164581. URL: https://doi-org.focus.lib.kth.se/10.1145/3164541.3164581.

[19] U. B V S et al. “Classification-Based Adaptive Web Scraper”. In: *2017 16th IEEE International Conference on Machine Learning and Applications (ICMLA)*. Dec. 2017, pp. 125–132. DOI: 10.1109/ICMLA.2017.0-168.

[20] M. Drott. “Indexing aids at corporate Websites: The use of robots.txt and META tags”. In: *Inf. Process. Manage.* 38 (Mar. 2002), pp. 209–219. DOI: 10.1016/S0306-4573(01)00039-5.

[21] Wen Zhu, Nancy Zeng, and Ning Wang. “Sensitivity, Specificity, Accuracy, Associated Confidence Interval and ROC Analysis with Practical SAS ® Implementations”. In: *NorthEast SAS users group, health care and life sciences* (Jan. 2010).

[22] Krishna Kalyanathaya, Akila D., and Suseendran G. “A Fuzzy Approach to Approximate String Matching for Text Retrieval in NLP”. In: *Journal of Computational Information Systems* 15 (May 2019), pp. 26–32.

[23] Umamageswari Kumaresan and Kalpana Ramanujam. “Automated scraping of structured data records from health discussion forums using semantic analysis”. In: *Informatics in Medicine Unlocked* 10 (Feb. 2018). DOI: 10.1016/j.imu.2018.01.003.

TRITA -EECS-EX-2020:515