Jan Dukaczewski
Main Supervisor: Bert Bredeweg
Second Reader: Victor de Boer
Vrije Universiteit Amsterdam
July 22nd, 2017

Bachelor Project Computer Science: Thesis

A Special Purpose Search Toolkit for Identifying Qualities in Data:

A Case Study in Finding Sustainable Businesses

Abstract

This paper presents research in the domain of Information Retrieval in the case of finding sustainable businesses. The goal was to come up with a set of tools and methods that work best for retrieving relevant entries from data sets, based on automatically identified qualities of the data. The work consisted of studying, describing, and evaluating the literature on the topic to establish the state of the art in Information Retrieval, building tools for crawling the web and accessing databases, and coupling them with different search methods. The contribution is evaluating and combining the accepted best practices in the field, and suggesting improvements and further research. The conclusion was that crawling is not a desirable method for acquiring data of unknown structure, database scraping performing much better. Another finding was that the applied classifiers' accuracy varies widely based on the characteristics of a data set, and recommendation of the classifiers to use is included, depending on the data set.

1 Introduction

1.1 Background Information

The more informed humans are, the better decisions they can make. As the amount of information produced by humans and devices around them is larger than ever, big data projects have become the keystone of applied Computer Science in the last decade [7]. However, with big data come big challenges. Information retrieval is the primary one, as large databases are becoming exceedingly hard to search. In the context of the Worldwide Web, one speaks of an Indexability Wall [2], where it has become impossible to index and track changes in all the documents because of their sheer quantity. Because of that, automated retrieval is often not accurate enough, and its results are not sufficiently relevant, but retrieving data manually proves too slow, and projects go abandoned. A project in that domain was started by Enactus, a student organization at the University of Amsterdam. bewustindebuurt.nl is a search-and-discover engine focused on sustainable businesses in the user's vicinity. However, their methodology of assembling data by hand has proven too slow, and the project was put on hold. It is highly probable that with an automated tool capable of crawling the web in search of sustainable businesses, the database could be filled much faster, and the project could scale up. This paper researched the possibility to come up with methods that enable that. The question asked is; what combination of retrieval, learning, and classification methods and features allows one best to automatically scrape available data for the purpose of identifying certain qualities in it. To answer the question, the paper evaluated the authors' efforts to populate the database using the results returned by the different tools and methods.

1.2 State of the Art, Related Work, and the Paper's Contribution

General purpose crawlers have developed substantially since the beginning of the century, but so has the Internet [7]. They are not able to keep up with the exponential growth of the volume of data, as by the time the crawler finishes indexing one part of the Web, the data it has collected about another faraway node has already become outdated. Even with relatively up to date indexing, information retrieval is hard, because what the user is searching for is hidden in an equivocal heap of information [1]. Furthermore, a challenge is posed by the *vocabulary mismatch problem*, meaning users often type different keywords to describe their search than what the authors use in the documents the user would find relevant. [4] These problems become amplified when expert knowledge is considered; domain-specific entries that are not commonly looked up. Search engines, such as Google, have learned to mitigate

these issues, deciding to serve the majority; a keyword are usually interpreted in the way a layperson would mean it; e.g. a search for "how to kill a zombie" returns eight results related to the popular culture notion of a zombie apocalypse, but not one showing how to get rid of a zombie processes in a Unix operating system (google.com, query: "how to kill a zombie", 15 Jul 2017).

That is why general purpose search engines cannot be successfully used to retrieve expert or domain specific knowledge. Several types of crawlers have been developed in response to this. One is focused (topic-driven) crawlers, which use a predicate of some form to limit the search results. Such a predicate can be the domain, a PageRank or BackLink value, such as in the crawler described by Almpanidis et al. [1], or an external dictionary that provides some information about a page before the page is visited. Focused crawlers are also known as vertical search engines, as their focus is on depth of the search tree, and not its breadth. The strategies vary by application, but a focused crawler always uses a classification algorithm to choose the pages to crawl.

Chau et al. [2] proposed an alternative; a general crawler, or one with a very liberal heuristic, whose results are filtered using Machine Learning algorithms after they have been retrieved. They noticed that finding a good starting URL is enough to retrieve many relevant results, however, the problem that remains is removing the irrelevant search results. The methods they explore range from measuring the occurrences of keywords in retrieved documents, TF/IDF matching, and binary classification using Naïve Bayes, Support Vector Machines, Neural Networks, and an unsupervised learning method of K-nearest neighbors for clustering. Interestingly, both Fabrizio [13] and Thornsten [13] praise SVMs for their *universality in text-classification tasks*, while Pang et al. consider them *unsuitable for binary categorization* [11]. A more in-depth description of these methods in the context of Natural Language Processing, alongside a comparison with Ensemble methods, bagging, and boosting, can be found in Màrquez's work in [9]. Other researchers have tried to apply the Bayesian classification directly as a heuristic for a focused crawler [3]. Because the inaccuracy caused by the classifiers can multiply the effect of an imperfect set of results, human classification is always needed before the results are accepted [2]. While Machine Learning is usually applied to text analysis in the context of sentiment analysis [2, 3, 8, 9], sentiment analysis itself can be seen as a special case of binary classification which can be conducted regardless of whether it is emotion or objective information that is being classified.

Chau et al. also mention the problem of equivocal user input, such as with the word “cancer”, which can refer to the disease, but also the zodiac sign. That is why another big topic in information retrieval is intent modeling, discussed in [4]. Because the phrases used by search engine users and content creators vary, and sometimes are contrary to each other, such as is common when the author describes the problem, e.g. “air pollution”, and the user searches for the desired effect, e.g. “clean air”. The authors therefore suggest that by *interactively modeling user intent*, namely, suggesting phrases to users mid-search, the search performance can be improved by more than 100% [4]. Dividing the keywords suggested into “explore” and “exploit” categories lets one both accurately model the intent, and, at the same time, satisfy the user by returning highly relevant results [6]. Furthermore, feature selection is sometimes used to give more weight to certain features, which are more likely to be associated with relevant results [10].

Hofmann, K. et al. recognize user input to be noisy and biased [5]. The authors say that context is so significant a determinant of user feedback that it is unfeasible to control all the variables sufficiently to produce usefully unbiased feedback. They studied how the inorganic factors such as placement and font choice, as well as the organic, such as personal preferences, make users click on different links, although semantically seen, these should not have had any impact. They suggested a method to minimize this bias using what they call *probabilistic interleave* [5]. If one assumes their perspective on user feedback, an attempt can be made to resolve the equivocal vocabulary issue on a machine level, as described in [8] by Petasis, who suggested Named Entity Recognition and Part of Speech Tagging as two methods that can be used to detect the intended meaning of the term in a document.

Although extensive research has been done in the domain, many questions remain unanswered. This paper aims to find out whether applying relevance-ranking methods to the results retrieved by general purpose crawlers can replace focused crawling without sacrificing flexibility for accuracy, in places where focused crawling is not a possibility due to the unknown structure and extent of the domain. The work also investigates how the vocabulary mismatch problem and the problem of equivocal input can be mitigated by applying natural language processing methods such as using synonyms and hypernyms. Finally, the paper discusses Machine Learning methods, as reported by Chau et al. [2], for classifying results on relevance.

2 Methods Employed

2.1 Rationale and Overview

The architecture of the toolkit is dictated by the character of problem the paper tackles The tool helps populate a database of sustainable businesses, and it is likely that the information of interest is available on different websites concerned with the topic, and in Open Data files that describe such places. Therefore, handling the data acquisition using a crawler, and an API tool is logical.

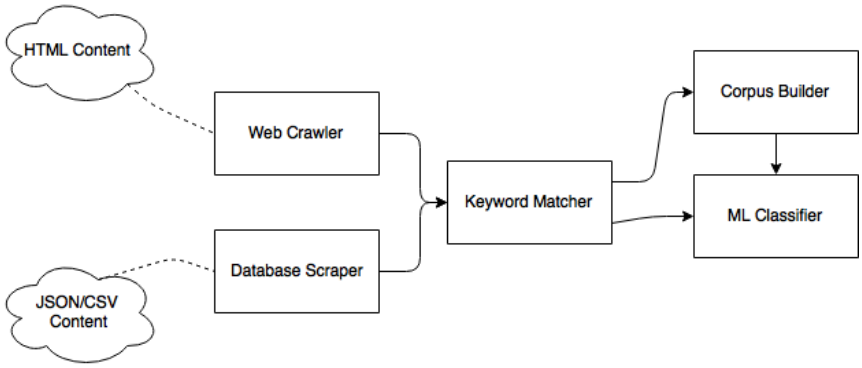

Figure 1: Components Overview.

Based on the design decisions described in Section 1, and the desire to evaluate and improve on the methods used by the research community, the authors have built a toolkit consisting of the following parts: (1) Web Crawler, (2) Database Scraper, (3) Machine Learning Classifier. Furthermore, they have constructed two helper tools that are instantiated by the main programs and facilitate their operation. These are (4) Keyword Matcher, and (5) Corpus Builder. The Keyword Matcher is used by (1) and (2) to find relevant entries. The Corpus Builder is used to build a bi-categorial collection based on keywords and the input from either (1) and (2) which can be divided into a training set and a test set for (3). (3) uses several Naïve Bayes and SVM classifiers, alongside Logistic Regression to classify the input. A guide for the toolkit can be found in the appendix.

2.2 Crawler Software Choices

Most crawlers built rely on a known website structure, and are used on a specific website where elements of a certain type can be retrieved using a tag that corresponds to the content class name, e.g. “ct” or “content”. In this case, the crawler has to scrape different websites, possibly with vastly varied structure and nomenclature. Unlike in the focused crawlers researched in [1] and [3], flexibility is a necessity here, even if it somewhat compromises accuracy, as it is only usable if it can be applied universally.

The crawler was built in Python, with the help of the module Beautiful Soup 4 (BS4)[1]. Python offers flexibility and is convenient for network coding and language processing, where various modules can be used to automatize the interpretation and categorization of content. Beautiful Soup allows for HTML, XML parsing, automatically finding links and other content in websites, and unifying the encoding.

2.3 Crawler Operation

The crawler downloads the source code of one page, identifies key elements in it, such as href hyperlinks, or paragraphs of content, and prints them to the user console. Using keywords and their synonyms, URLs of the relevant documents are returned. The main method (get_page_data(page_url)) crawls a page, stores the found hyperlinks in a list, and then recursively calls on itself with the URLs as arguments. A depth argument to the main method call prevents crawling trees of infinite depth. When that depth is reached, get_page_data(page_url) still prints the content of the page that it is scraping, but it does not call on itself with the URLs it finds there, and the program returns. A global variable, urls_visited[] keeps track of the visited URLs which prevents looping back to the homepage or other pages referenced from within. The scraping function find_all uses argument 'div' which retrieves all content containers in the website.

Custom headers are used to circumvent the reluctance to crawling that may be exhibited by some websites. The crawler disguises itself as a web browser; headers = {'user-agent': 'Mozilla/5.0'}. However, to ensure that no unauthorized crawling takes place, the program reads robots.txt and only crawls the parts of the website that are publicly available.

The crawler features a boolean setting for filtering out all URLs that lead to websites outside the page's domain. This behavior limitation may be desirable in certain situations, such as when scraping the content, but not in others, such as when attempting to discover new websites that may contain valuable content that are linked to from the current website. Another boolean prevents the crawler from crawling pages in its own domain in the first step of recursive descent.

2.4 Keyword Scraper Operation

[1] BS4 is developed by Leonard Richardson. https://www.crummy.com/software/BeautifulSoup/. Last updated and accessed July 2017.

The keyword search functionality was implemented using additional libraries; *Re*[2] (regular expressions) is used to sanitize the text from special signs, punctuation and anything that is not an alphabetic string of characters. *Operator* is used to sort the frequency table. *Stop_words*[3] is a very useful library that lets one remove the common "stop words" such as "and", "or", "the", etc. This package works in English, as well as a number of other languages, including Dutch. As the definite article and conjunctions are the most common words in every Germanic language, this lets one remove those highly prevalent yet irrelevant words from the results. The paragraphs are acquired using an HTTP request, and a library for turning HTML into plain text. However, instead of printing the text, they are sanitized and stored in a list. Then, a function iterates over the list and counts the occurrences of each word. The total sum of words is also included, so the occurrences can be expressed in terms of frequency distribution. The words and their frequencies are then appended to a new list of tuples which can then be printed or passed on for further processing.

The crawler gets to a page, runs the scraper function to create a word-frequency table for it, and appends it to a dictionary, in a tuple with the page's URL. The frequencies are analyzed during execution. The function takes in as an argument the series of page URLs with a list of word-frequency tuples for each page created by the crawler. It returns URLs whose word-frequency lists have a high occurrence of custom specified keywords. In this way, a number of pages is crawled, and each page's relevance is assessed right away.

2.4.1 Keyword Crawler-Scraper Pseudocode

This is the pseudocode for the crawler-scraper. For simplicity, special cases such as the domain restrictions are not included.

```
crawled_urls []
search_keywords []
retrieved_keywords []
starting_url := 'http://www.example.com'

function crawl (url):
        page_text = page_to_text(url)
```

[2,3] Re and Stop Words are part of the Python Standard Library. Python Software Foundation https://docs.python.org/2/library/re.html

```
        page_urls [] = page_text.find_type('a href')
        page_keywords [] = page_text.find_type('content')
        for url in page_urls do
                crawl(url)
        end
        retrieved_keywords.add(scrape(page_keywords[])
end

function scrape (keywords[]):
        found_keywords []
        for keyword in keywords do
                if keyword in search_keywords then
                        found_keywords.add(keyword)
                end
        end
        return found_keywords
end
```

2.5 Parsing JSON data from Open Data sources

2.5.1 Software Choices

On the level of code, downloading JSON data with Python is much less complicated than downloading and parsing HTML content. JSON objects are easily convertible to Python dictionaries which, in turn, can be iterated over, browsed, and consolidated with other lists. The authors have used Python packages "urllib2" for getting the data from the server, and "json" for parsing the JSON contents. Because the nature of the data remains the same as in the HTML crawler, the only difference being the file format – JSON instead of HTML, JSON can still be searched using the keyword-based search engine.

2.5.2 TF/IDF

The hypothesis was that the accuracy can be increased by the means of feature extraction for Term Frequency / Inverse Document Frequency (TF/IDF) based classification, employing a dictionary (lexicon/thesaurus) for the keywords and the words used in the searched documents. This could be further augmented by machine learning methods such as Logistic Regression and Support Vector Machines, finding similar documents and removing less relevant search results from the retrieved set. The basis for this is the research done in [2]

which shows that these methods exhibit significantly higher accuracy than keyword search, especially in smaller and incomplete data sets.

The authors tokenized the retrieved descriptions and used them as documents for the TF/IDF matching. The corpus can be seen as all the descriptions in the database. When the scores were calculated with the sustainable descriptions as documents, and all descriptions as the corpus, a set of features characteristic of sustainable businesses' descriptions was created.

TF lets one understand what role the keywords play in a document, but its limitation is that it only uses the document itself as reference. Because of this, one does not know how the document ranks in the collection. TF/IDF lets one ignore the general features of the collection, as the focus is on the specifics of particular documents. This is applicable in web crawling, as the crawler operates on sets of retrieved pages to rank them on relevance, and even more so in Open Data retrieval, because the descriptions are necessarily homogenous in some regards, and finding the features related to sustainability appears more promising than a simple keyword search.

2.5.3 Document Classification

Supervised learning methods such as Logistic Regression and SVM can be applied to the retrieved features to find similar documents. The number of features, or words from the dictionary that are used varies depending on the purpose. Unlike in keyword matching, one does not decide on the keywords that shall be used; the algorithm takes e.g. 100, 1000 or 10 thousand most common features from the training set, and searches the test set for their instances to classify the documents. Depending on the application, a different number of features is used, and the more general the classification, the less features need be used to find correlation [3]. In this case, the application is very domain specific, so it is reasonable to use in excess of the 3000 most common words to find a usable correlation.

For classification, the authors implemented several Naïve Bayes classifiers, Linear Models of SGD and Linear Regression, Logistic Regression, and Support Vector Machines. This is based on other research in the area, and a comparison of the results with those of [2] is in the evaluation section.

All Naïve Bayes classifiers are based on the idea of independent probabilities, but the mathematical background differs. This research employed a Multinomial NB which does not use the vector of samples (features) directly, but calculates a polynomial that most accurately

matches the feature set. Bernoulli NB was also used, which treats features as booleans rather than frequency values. The hypothesis is that Bernoulli would perform the best, as it gives more importance to the existence of certain features, such as the keywords beforehand, than the frequencies of these. This is especially useful for shorter texts and smaller sample sizes, which is the case in this data set. SGD and Linear Regression yield a baseline for the hyper-linear methods that are employed – Logistic Regression and Support Vector Machines.

2.5.4 Input for Classifiers

Every supervised classifier needs two categories of classification; positive and negative examples. The issue with the data sets is that only examples known, or likely, to be positive are available, as there are no databases containing businesses of similar category that are known not to be sustainable. This lack has been mitigated using improvised methods which are not industry standard, but applicable in this case. The authors came up with three categories. The first category encompasses documents which are said to describe sustainable businesses according to the sources described in the previous paragraph. The second, new category, is descriptions which are unlikely to be associated with unsustainable businesses – this category was created based on the keyword search. This method creates significant bias, and it is only justified for testing the software, to augment the database which would otherwise be too small to use. When the tool is used on new datasets, the user is responsible for the training set consisting of independent examples and not being an effect of any other prior classification. The authors justified using this suboptimal method by the goal of the research, which is not to produce accurate results per se, but to explore different technologies in the field.

3 Results and Evaluation

3.1 Experimental Setup

The experiment was run in accordance with the research community guidelines. The authors did have expectations as to the performance of the methods, but could not estimate their rendition in this case specifically. Therefore, the programs were run on the data sets sequentially, independent of each other, the results being juxtaposed with each other. Because the data sets were suboptimal, the programs were also run with external data sets to obtain more accurate results and reduce bias in performance measures.

Listed in Table 1, are the nine URLs the crawler was deployed on, suggested by the people in the Information Retrieval and Sustainability community. The cutoff depth of the search tree was chosen to be 8, with a time limit of 60 minutes. It was found to be the greatest depth where the crawler would terminate within the time limit for most starting URLs, and the greater the depth, the more results are acquired. To evaluate the JSON parser and the keyword search, the authors scraped several databases of Open Data Amsterdam (data.amsterdam.nl), which was dictated by the local character of the search effort and the type of desired results. Term frequencies were used to assign relevance scores to retrieved entries. These entries were then stored, and categorized using keyword matching into possibly relevant and irrelevant, which was subsequently used to build the corpus for testing the Machine Learning methods. Each classifier was run 100 times to minimize the effect of unaccounted variance in performance measurements on the results, and an 80% to 20% training set to test set ratio was used, because the data set was relatively small. The authors used the data set "movie_reviews" from the NLTK corpora library with a 90%-10% train-test ratio to evaluate the classifiers' performance, as it is a larger, and better pre-processed data set.

3.2.1 Keyword Crawl

Table 1: Relevance and Coverage of Focused Crawling from origin URL

starting URL	# retrieved URLs	# relevant URLs	% relevance	comments
iens.nl	63	8	12.7	stopped manually
thenextgeneration.nl	0	N/A	N/A	blocked
puuruiteten.nl	26	7	26.9	-

starting URL	# retrieved URLs	# relevant URLs	% relevance	comments
yelp.com	27	4	14.8	-
foursquare.com	21	3	14.3	-
google.com (search)	90	2	2.22	stopped manually
bewustindebuurt.nl	16	4	25.0	-
marqt.com	8	1	12.5	only own domain

The results show very low relevance, and have taken a very long time to be collected. One could achieve similar results randomly following links on each page which is, in essence, what the crawler does. Getting responses from the server has taken prohibitively long, rendering the crawler unusable.

Many sustainability-related websites feature little relevant keywords, while many other keywords are not specific enough. Sentences are written in a way that makes it clear for humans, but would require very advanced natural language understanding from machines to interpret the text as having to do with *sustainability*. This poses a significant limitation to the power of keyword-based scraping applications.

A significant challenge is distinguishing the classes in the website constituting textual content. Some pages within a single sample website have paragraphs in divs of class names as varied as *content*, *item-ct*, *item-content*, or *ct*. Searching for the incorrect div type yields no results. If such discrepancies can be found among pages of the same domain, then it is next to impossible to index all the content-bearing category names from different domains and pages that the crawler can encounter.

Finally, certain websites only allow user agents and throw an exception, even if the crawler masks itself as a web browser using the custom headers.

It is hard to objectively evaluate the crawler's performance, as the task that it performs is inherently different from that carried out by the other parts of the tool. Its findings are not pre-determined by the entries in an existing document, and they differ based on the entry point. However, one criterion of evaluation is usability, and due to its limitations in flexibility, speed, and volume of results, it receives a very low usability score.

3.2.2 Keyword Scrape

Table 2: Keyword Scrape Results

Database	# entries	# pos	# likely	# true	% recall	% cov.
EtenEnDrinken	752	246	309	199	80.9	64.4
Shoppen	597	155	202	99	63.9	49.0
UitInAmsterdam	341	83	60	43	51.8	71.7

#entries in DB – total number of entries in the database

#pos retrieved – number of entries marked as relevant based on keywords and retrieved

#likely positive – number of entries that are likely relevant, based on Google search and cross-referencing with other classification methods

#true pos – number of entries in the intersection of "pos retrieved" and "likely pos"

% recall – percentage of entries in "true pos" that are also in "pos retrieved"

% coverage – percentage of entries in "true pos" that are also in "likely pos"

Name	Score
Bar Loulou	23.0769
YAY Health Store & More	21.4287
Koffie Academie	21.4286
Lokaal Spaanders	18.1818
Spirit Amsterdam	18.1818
Frits	17.6472
Staal	16.6668
LAVINIA Good Food	16.2792
Restaurant Moer	15.3846
Het Zwaantje	14.2857

Table 3: Relevance score example: 10 highest scoring businesses

This version has been the most successful attempt at retrieving information about sustainable businesses from any source so far. As it appears in Table 3, the relevance score is only a preliminary measure, and does not have to relate in a meaningful way to the real world criteria classified on, but it does show that the entry is a potential candidate, and it can be used to find out more about the entry and verifying its classification. The authors found the usability of the tool to be very high, because thanks to the database, the user can access any available data related to the entry, such as address and contact details of an establishment.

To evaluate usability in terms of added value, the authors Google searched the names of 25 retrieved places from Table 2 (~ 10% of the total result space) to see whether they could easily be found if not for the program. Of the 25 establishments, 18 had some form of individual media presence, that is, own website, a Facebook page, Twitter account, or a Google Maps pin with at least an email address and a phone number. 5 of those had their own website. Only 7 pages profiled the business as *sustainable* (descriptions, images, etc.).

Therefore, the authors concluded that the tool has a significant impact on discovering entries that would not otherwise be discoverable. The added value of the tool is very high.

As much as cross-checking with existing sources is possible, the obtained results do appear in the databases of bewustindebuurt.nl, thenextgeneration.nl, puuruiteten.nl, and iens.nl with the *biologisch* (*organic*) tab selected, as mentioned in Table 2. Close to 100% of the entries on these websites appear in the results obtained. The other way around, the correlation is much lower, as none of these databases is as comprehensive as this paper's results set.

3.2.3 Augmented Keyword Scrape

Enhancing the keyword set with synonyms (words carrying a similar meaning) and hypernyms (superordinate words) appears very promising, but it has its caveats. WordNet is an English language lexicon. An alternative framework for Dutch is OpenDutch WordNet, but it is incompatible with Python versions below 3.2x, while the other modules used are incompatible with Python versions above 2.7x. To circumvent this issue, the paper attempted to use the English descriptions of places, however, only about 60% of entries in the data set has those. Another attempt was made by translating the keywords and description to English using py-translate, but because the translation module is not aware of the context in which the words are used, the resulting matches were wholly insupportable. This prevented us from obtaining meaningful quantitative results. Therefore, the paper only offers a qualitative description of how the method has influenced the results, and how it can be used in the future to collect better data.

Attempting to put aside the bias and deviation introduced by translating the results or incomplete English language descriptions, it can be shown that the influence of synonyms and hypernyms on the results is related to the extended set. It can be done either with the set of keywords themselves, or by finding synonyms for all words in the description sets (after filtering out stop words etc.). In the first case, as expected, since a larger set of keywords is used, the results differ, and are larger in volume. In the latter case, the results remain the same, but their relevance scores change, and the order shifts, placing some results from previously as low as the bottom quarter of the table in its top 25%. The mean relevance score increases by about 100% in this approach.

3.3 TF/IDF "Most Useful Features"

LHS: Table 4: 10 most informative features, as identified by the NB classifier

RHS: Table 5: 10 most informative features, as identified by the NB classifier, with the keyword features removed

feature	class	ratio
huisgemaakte	pos : neg	25.9 : 1.0
keuze	pos : neg	24.4 : 1.0
kwaliteit	pos : neg	21.5 : 1.0
vers	pos : neg	20.0 : 1.0
licht	pos : neg	15.6 : 1.0
gezonde	pos : neg	14.1 : 1.0
inspireren	pos : neg	12.6 : 1.0
houoven	pos : neg	9.6 : 1.0
voel	pos : neg	8.1 : 1.0
worstjes	pos : neg	6.7 : 1.0

feature	class	ratio
houtoven	pos : neg	9.6 : 1.0
voel	pos : neg	8.1 : 1.2
extra	pos : neg	6.7 : 1.6
vijf	pos : neg	6.7 : 1.5
strakke	pos : neg	6.7 : 1.4
jong	pos : neg	6.7 : 1.3
best	pos : neg	6.7 : 1.2
worstjes	pos : neg	6.7 : 1.0
iedereen	pos : neg	6.1 : 1.0
open	neg : pos	5.9 : 1.0

Because the keywords were previously used as a qualifier for the positive set, it should come as no surprise that many of the features discovered by the Naïve Bayes algorithm listed in Table 4 are the keywords themselves. However, the results have also showed some features with very high positive to negative ratios that were not seen before, such as, among others, “licht” (light, adj.), “inspireren” (to inspire), “houtoven” (wooden oven), “voel” (to feel), and, apparently, “worstjes” (little sausages). Most examples are positive, as words describing sustainability tend to stand out, while the descriptions of negatives are rather homogenous [3]. Table 5 shows the effect of removing the keyword-induced features from the feature set.

3.4 Classifier Results for NB, Linear, Log. Reg., SGD, SVMs

The following are the results of the different classifiers, with 80%-20% training set to test set ratio for the “sustainable” data set. On small data sets, this ratio is considered the best [2], as it minimizes the bias and variance on the test set. Because “movie_reviews” is a bigger data set, the paper used a 90%-10% ratio, which would result in high performance estimate variance if the set were too small, but is appropriate for its size. The classifiers have been run 100 times. The document set has been reshuffled after every run to prevent training and testing on the same data or duplicate results.

Table 6: Classifier Accuracy (%) across 100 runs on "sustainable"

Classifier Accuracy (%)	mean	Q1	Q3	IQR (Q3-Q1)
NB	97.3	95.2	99.1	3.90
MNB	95.5	94.3	96.6	2.30
Bernoulli	85.5	84.1	95.7	11.6
Log. Reg.	98.4	96.1	99.3	3.20
SGD	95.7	93.7	99.3	5.60
SVC (SVM)	71.2	66.5	75.5	9.00
Linear SVC (SVM)	98.6	96.1	99.2	3.10
Nu SVC (SVM)	95.7	95.4	97.9	2.50

Table 7: Classifier Accuracy (%) across 100 runs on "movie_reviews"

Classifier Accuracy (%)	mean	Q1	Q3	IQR (Q3-Q1)
NB	82.5	81.9	83.9	2.00
MNB	88.1	87.4	88.3	0.90
Bernoulli	86.3	82.5	90.0	7.50
Log. Reg.	93.0	91.7	94.5	2.80
SGD	78.6	75.7	79.1	3.40
SVC (SVM)	50.0	49.8	50.9	1.10
Linear SVC (SVM)	96.6	94.2	97.1	2.90
Nu SVC (SVM)	90.2	89.8	91.0	1.20

Q1 – First quartile of the distribution (top 75%)
Q3 – Third quartile of the distribution (top 25%)
IQR – Inter-Quartile Range

The data set is extremely small for the purpose of classifier training. As shown in Table 7, there are only have 246 positive examples (joint categories I and II) and 520 negative examples, of which 274 were randomly removed to give parity to positive and negative examples.

For evaluation purposes, the authors removed the features corresponding to the keywords from the trained classifier feature set before classifying the test set in some runs. This significantly reduces the bias induced by the practice described in the previous paragraph.

Both the homogeneity of the descriptions in the data set, and its size, contribute to the high perceived accuracy of the classifiers. One can, however, also notice that the average IQR for it is almost twice as high as when classifying movie_reviews (5.15 vs. 2.73), showing that classifying the data set suffers from greater inconsistency and uncertainty. In both cases, SVC SVM has the lowest accuracy (Table 6). Interestingly, SGD is the second worst performer in the test data set, yet it achieved one of the best accuracies in the data set. With both data sets, Bernoulli NB has the highest accuracy IQR (11.6 and 7.5), while the most consistent performance is maintained by the Multinomial NB (2.3 and 0.9). Linear SVC SVM performs the best on both data sets, and shows a high degree of consistency. While in this small data set the difference in training times the Naïve Bayes classifiers and Support Vector Machines is insubstantial, it becomes visible in the large movie_reviews data set (Table 7). However, the performance advantage is also much more significant in the latter, making the Linear or Nu SVMs a better choice, unless time is of essence, so as it may be the case in a strict real-time system.

4 Discussion and Further Development Suggestions

4.1 Crawling and Keyword Scraping

The issues mentioned in section 3 cannot be easily resolved with the current state of the Web. Rather, they point to the weaknesses of crawlers when they are deployed on web content of unknown structure. This paper's finding is that a crawler without a heuristic cannot match the performance of focused crawlers, even if very good algorithms are used for filtering the results, because the results cannot be collected efficiently enough to make them useful. Embedded linked data could help build ontologies and add structural information to the crawled websites which could result in more targeted and efficient crawling.

As far as the speed is concerned, a remedy could involve concurrency and a branch-cutting algorithm. The tree of links followed by the crawler can be represented as a binary search tree. In this crawler, the main thread crawls the pages, while worker threads scrape the content and report back on its relevance. If a page's relevance is below the cut threshold, the URLs followed on that page will not be followed and the branch will have reached a dead end. This idea has the potential to increase the speed of crawling while also augmenting accuracy.

4.2 Classifying, finding similar documents

The primary challenge of identifying documents based on similarities is building a large, unbiased corpus for extracting features for classification [7]. In the case of this paper, the data set is small in size and biased due to keywords being used to create the positive and negative categories.

The authors have therefore removed the keywords from the feature set to counteract the bias, which has led to increased reliability in results. Even so, the distinction between positive and negative documents was clear to most classifiers. This can likely be attributed to their homogenous characteristics; unlike the websites classified in [2], or the documents from an indefinite domain used in [5], the descriptions in this research all share a very similar structure, and those related to *sustainability* tend to stand out. That is why the relatively simple mechanisms behind Naïve Bayes classifiers are not surpassed by much by the more complex Logistic Regression or Support Vector Machines in this case. Extrapolation to higher dimensions brings little advantage when the distinction between features on a place is well visible [13]. When evaluating the methods using the NLTK corpus "movie_reviews", which is a

bigger and more diverse set, the results painted a different picture, and confirmed the performance of classifiers established in [2, 12].

5 Conclusion

Web crawling and document scraping, with today's tools and the current state of the Worldwide Web, is as much an art as it is a science. The biggest disadvantage of general crawlers is their blindness – they follow every URL within specified domains, regardless of how relevant it may be, because the crawler cannot assess the relevance before the page is crawled. While focused crawlers try to counteract that with a search heuristic, that restricts the flexibility of the crawler, making it inapplicable to unstructured sets of links with unknown domains [1, 3]. The variety of tags used for content in websites, text-infused images, and bot blockers all serve to diminish the effectiveness of crawling. That is why this paper suggests to always use documents with structured data, if such are available. Documents in JSON or CSV formats can easily be converted to dictionaries, ordered, and searched using any Natural Language Processing and Machine Learning method, or a combination thereof.

The inherent limitation of this project is its pioneer nature – as the data set is concerned with sustainability, it is limited; consequently, the accuracy measurements have low reliability, and the Naïve Bayes classification, Logistic Regression, and Support Vector Machine accuracies do not follow the usual pattern for text classification. The authors, however, found Logistic Regression and Linear SVC SVMs to be the most accurate, and thus recommend those methods when interpreting incomplete data sets that have not undergone thorough pre-processing. The research community is divided as to the usability of SVMs for binary classification [11, 12], but when evaluating on an external data set, the research in this paper managed to reproduce the results obtained in [2, 12], and found Support Vector Machines to perform the best, unless a large number of documents is used, in which case Naïve Bayes and keyword matching show similar accuracy. Thanks to the flexibility of the software that was built, the program can be used on any data set, both for evaluation purposes and for real world applications.

Bibliography

[1] Almpanidis, George, Constantine Kotropoulos, and Ioannis Pitas. "Combining text and link analysis for focused crawling–An application for vertical search engines." *Information Systems* 32.6 (2007): 886-908.

[2] Chau, Michael, and Hsinchun Chen. "A machine learning approach to web page filtering using content and structure analysis." *Decision Support Systems* 44.2 (2008): 482-494.

[3] Chakrabarti, Soumen, Martin Van den Berg, and Byron Dom. "Focused crawling: a new approach to topic-specific Web resource discovery." *Computer networks* 31.11 (1999): 1623-1640.

[4] Ruotsalo, Tuukka, et al. "Interactive intent modeling: Information discovery beyond search." *Communications of the ACM* 58.1 (2015): 86-92.

[5] Hofmann, Katja, et al. "Learning to rank for information retrieval from user interactions." *ACM SIGWEB Newsletter,(Spring)* 5 (2014): 1-5.

[6] Hofmann, Katja, Shimon Whiteson, and Maarten de Rijke. "Balancing exploration and exploitation in listwise and pairwise online learning to rank for information retrieval." *Information Retrieval* 16.1 (2013): 63-90.

[7] Kitchin, Rob. The data revolution: Big data, open data, data infrastructures and their consequences. Sage, 2014.

[8] Petasis, Georgios P. Machine Learning in Natural Language Processing. Diss. 2011.

[9] Marquez, Lluıs. "Machine learning and natural language processing." Complementary documentation for the conference "Aprendizaje automático aplicado al procesamiento del lenguaje natural", Soria. 2000.

[10] Patočka, Michal. "Machine learning for sentiment analysis." (2013).

[11] Pang, Bo, Lillian Lee, and Shivakumar Vaithyanathan. "Thumbs up?: sentiment classification using machine learning techniques." *Proceedings of the ACL-02 conference on*

Empirical methods in natural language processing-Volume 10. Association for Computational Linguistics, 2002.

[12] Sebastiani, Fabrizio. "Machine learning in automated text categorization." ACM computing surveys (CSUR) 34.1 (2002): 1-47.

[13] Joachims, Thorsten. "Text categorization with support vector machines: Learning with many relevant features." Machine learning: ECML-98 (1998): 137-142.

Appendix A: Source Code

The code can be found in the GitHub repository https://github.com/jfdeverne/Bachelor-Project. This section describes the workings of the tool, and the tool is equipped with usage guides. Argument "-help" prompts explanations for all modes.

The Web Crawler takes as arguments (1) a list of keywords and (2) a URL of the initial webpage. The Keyword Scraper requires (1) a list of keywords as well, and (2) a URL of the .JSON file with the data for scraping. The Corpus Builder parses the entries in a .JSON file into a corpus that can be loaded and used with NLTK and SciKit. It takes (1) a list of keywords, (2) a list of entries' names known to be positive ("verified positives"), and (3) the target .JSON document. The classifier consists of two parts. features_train.py loads the corpus, extracts the features, trains the selected classifier, and save it in a .pickle file. features_train.py also takes an optional list of keywords as argument, if it is provided, it excludes the corresponding features from the feature set used for training, to reduce bias. features_test.py can then be used to classify input using the pre-trained classifier. The Corpus Builder names the files after the entries' names, so the output of features_train consists of all the file names (entry names) classified as positives.