Information-Extraction from websites with a NER-Approach Group Web-B-Gone

Jan Braker and **Lorenz Heinemann** and **Josephine Selig**
Student at Leipzig University
Computer Science (M.Sc.)
{jb64vyso, lh31ghzy, js12ziwi}@studserv.uni-leipzig.de

Tobias Schreieder
Student at Leipzig University
Data Science (M.Sc.)
fp83rusi@studserv.uni-leipzig.de

Abstract

Finding specific information on different websites is a problem that is of great importance in many research and application areas. Search engines in particular benefit from this technology. Therefore, in this thesis, an attempt is made to examine and test new approaches to extract information from unseen websites. This paper suggests two classification methods to categorize the web page and determine the extraction attributes. Two extraction models were developed, each pursuing contrary approaches. The first model just tries to memorize the HTML structure and positions of the attributes on a webpage, without taking the text content into account. The second approach, on the other hand, only receives the text visible on the website as input and tries to extract the information using a trained NER (Named Entity Recognition) model. A third model is examined in this work, in which both approaches are combined and the individual advantages are used as far as possible. The SWDE dataset formed the basis for training. Surprisingly, the results of the NER-model in particular can surpass previous results found in the literature without much optimization and fine tuning. The model based on the HTML structure and the combined model could not keep up there.

Keywords

Named Entity Recognition (NER), Natural Language Processing (NLP), Machine Learning, Spacy, HTML-Template, Website Classification, Convolutional Neural Network

1 Introduction

The internet has almost reached 5 billion users at the beginning of 2022. Each of them visit different websites almost every day. The internet is an important part of everyday life these days. Many processes are initiated and controlled by the exchange of information via the World Wide Web. Websites serve information pages for a number of applications. Almost every business and company has a website for digital presence. Opening hours, employees, locations and company descriptions can often be found quickly and clearly for new customers and interested parties. An online shop presents its products with associated prices and information such as article number, size and description.

Each page is structured differently and the information is kept in different places. Finding this and separating marginal and important information is often not a problem for a person and happens within seconds. However, this is not quite so easy for a computer, since different website structures, layouts or categories make it difficult to find the information.

In the age of automation, however, it would be desirable if computer programs could automatically search web pages and extract important information. This would make it possible to create databases about products in online shops, or to automatically update the opening hours of available shops in the city.

An example is the search engine Google, which tries to provide the user with an answer to a question that may be asked in a query as quickly as possible by providing direct answers. The technique of extracting information from websites has been improved more and more in recent years and, according to *Sparktoro*, the number of "no-click searches" is increasing. This is solely due to the fact that many search queries are answered directly in an information box at the top. In 2016, only every second search query on mobile devices ended without a click on a listed page (Fishkin, 2018).

In 2020 it was already over 77% (Fishkin, 2020). This clearly shows how important it is to further research this topic and to test new approaches for better extraction and to optimize existing ones.
The two approaches presented below follow different approaches. The first model, called the structure-model, assumes that information is often in the same places on a website. Once the model has seen a web page with the same structure, information could be reliably extracted from all pages with the same structure.
The second model only looks at the text on a web page. The assumption here is that the model learns to recognize names, prices, and other information from the actual and neighboring words.
In the end, an attempt should be made to combine the advantages of both models in a combined approach.
For training the SWDE dataset from Microsoft is used, which contains real-world websites (Hao et al., 2011). These are labeled for information on the page and are already divided into categories. The dataset should also form the basis for the evaluation of the models presented here to be better comparable to other papers.

2 Related Work

The research field of page extraction has a wide variety of use cases and their corresponding approaches. Chang et al. proposes a comparison of the different extraction tools by task domain, automation degree and the techniques used and provides an overview of the existing methods (Chang et al., 2006).
In the work of Wang et al. it was shown that structured text fields can be reliably extracted from websites (Wang et al., 2022). They achieved this by tokenizing and analyzing the HTML of the page.
The work of Carlson et al. attempts to use fewer annotated pages of a domain to infer all pages found there (Carlson and Schafer, 2008). The pages are semi-structured. This could be particularly useful for sites such as online stores, where only a tiny number of pages need to be examined and labeled to then reliably extract information from all remaining pages.
Gogar et al. present a method that does not have to be tailored to a particular website and can extract information from previously unseen pages and their unknown structure. Their new method uses a convolutional neural net, which combines visual and textual content, as well as relative positioning. A new method for textual encoding, called "Tree Maps" that corresponds to spatial text encoding is also proposed (Gogar et al., 2016).
The paper of Foley et al. researches the extraction of local events, especially to four fields of title, date, time and location, using distant supervision and not being reliant on microdata or schemas. For this, they invented a method to join several independent field scoring functions. Furthermore, they use a greedy selection and grouping approach to prevent overlapping and optimize the quality of the obtained events (Foley et al., 2015).

Since the web pages were provided without image embedding or any, no analysis based on OCR approaches or relative positions can be carried out. From this the work concentrates on the structural learning of HTML positions and the text analysis on the web page.

2.1 Classification of webpages

In this paper, we want to create an approach that works for multiple extraction tasks, but still filters the webpages into topics to get the corresponding attributes that have to be extracted.
There are multiple types of classification. Binary classification splits webpages into two classes, for example in the research of detecting phishing websites (Hodžić et al., 2016). Furthermore, there are multiclass problems, as presented in this paper, where instead of splitting the set of web pages, the classifier assigns them to labels corresponding to a category (Safae et al., 2018).
Aburrous et al. use fuzzy logic and data mining algorithms to create a model that evaluates if a web page is a fishing web page by the criteria URL, domain identity, security, encryption and association rule algorithms (Aburrous et al., 2009). In the work of Mohammad et al. another approach, using a self-structuring neural network to cope with the pace of changes of phishing websites (Mohammad et al., 2014).
Kwon et al. researched ways to use a k-nearest neighbour approach for webpage classification and augmented it with a feature selection method and term-weighting using mark-up HTML tags (Kwon and Lee, 2000). A classification applying hidden Markov tree models present Tian et al. and base it on generated DOM trees.

They present a two-phase approach, classifying with an HMT-based (hidden markov tree) classifier, and denoising and pruning for higher accuracy (Tian et al., 2004).
A more similar classification technique to the one used in the paper is proposed by Karthikeyan et al. with scraping information from web documents. They first classify the training set with keyword matching. Furthermore, they utilize logistic regression - recursive feature elimination (LR-RFE) to create the best candidate feature set, which is then used to train a back-propagation neural network (Karthikeyan et al., 2019).

2.2 Named Entity Recognition

Named Entity Recognition, or NER for short, comes from the research field of Natural Language Processing (NLP). The aim is to analyze the natural language in text and to automatically recognize proper names in the context and assign them to predefined categories. Therefore, different algorithms and rule-based approaches are used. Proper nouns are single words or phrases such as a person's name, company name, place, or event (Mansouri et al., 2008).
By using a NER approach and extracting and classifying important entities from a text, information for the overall understanding of the text can be gained. The automation of NER makes it possible to process large amount of text in short time using dictionary-based, rule-based and artificial intelligence-based implementations (Luber, 2022). The process always takes place in two phases. In the first stage, the entities must be recognized. The challenge here is that the same names and word sequences can appear in different variants or formats. The recognized words are then classified and assigned to one of the given categories.

Processes from the field of artificial intelligence achieve the greatest success rate. Models are trained on training data and can then reliably recognize entities from unknown texts. The *BiLSTM-CRF models* are a particularly successful model architecture. BiLSTM-CRF is short for Bidirectional Long Short-term Memory with Conditional Random Field. These models are artificial neural networks equipped with bidirectional Long Short-term Memory (LSTM) and a CRF layer (Conditional Random Field Layer) (Luber, 2022). The bidirectional LSTM layer enclose a forward- and a backward-pass of an LSTM layer, followed by the stacking of the sequences returned by both passes (figure 1). The second layer applies a dense classification layer to every position of the stacked sequences (Timmermann, 2020).

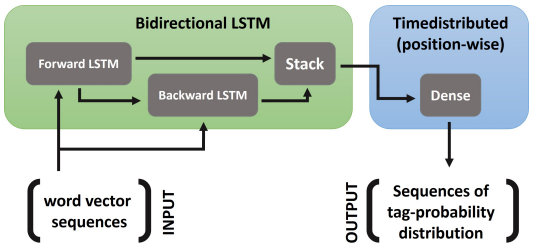

Figure 1: General structure of a BiLSTM network with layers offered by Keras (Inspired by Timmermann (2020))

2.3 Spacy

There are different libraries for the implementation of these procedures. Since *Python* is used in this work, the free open-source library *spaCy* was chosen. The project, published in 2015 by Matthew Honnibal and Ines Montani, offers various methods and procedures for fast natural language processing with large texts (spaCy, 2016b). With spaCy you can also train your own models for a variety of text analysis tasks. A large number of pre-trained models for different languages are already available. For these purposes the *en_core_web_md* model was used in this work.
With the *EntityRecognizer* (spaCy, 2016a), spaCy provides a pipeline-component that can identify non-overlapping areas in a text as entities and assign them to one of several previously defined categories. This makes it very easy to implement a NER analysis on texts. By default, some categories are already supported. If more are required or other analyzes are desired, a model based on a pre-trained spaCy-Model can be trained and optimized with own training examples. This is also the chosen approach in this work.

3 Methods

We split the Problem of information extraction from an arbitrary website into two main problems. First, we have to decide which set of attributes should be extracted. We achieve this by assigning the website to one of eight categories (section 4). Each category is characterized by a number of attributes to be extracted from the related webpages.

```
restruc_swde/
└── <Category file name>/
    ├── domains.json
    └── W<First two chars of url hash>/
        └── W<Full 16 char url hash>/
            ├── groundtruth.json
            ├── website.htm
            └── website-url.txt
```

Figure 2: Restructured directory structure of the SWDE dataset.

Since the classification of websites is a well-studied problem (Al-Ghuribi and Alshomrani, 2013), the focus of this work is the evaluation of different attribute extraction models. The first extraction model uses only the HTML structure of the website to learn a template. This template is then used to predict the position of the attribute and extract it (section 5.1). Since this approach only considers the HTML and not the content of a website, two different NER-models were developed for the attribute extraction based on the website content. First, the pre-trained spaCy-model *en_core_web_md* was fine-tuned on the SWDE dataset (section 5.2). Additionally, an own LSTM-model was developed (section 5.3). To combine the advantages of both approaches, the template-recognition-model and the spaCy-NER-model were combined into the template-NER-model (section 5.4).

The SWDE dataset used in this work, was restructured. For each website, a unique ID from the URL hash was generated and the websites were saved in a new directory structure described in figure 2. This enables easy and high-performance access to the individual files, like the ground-truth for every single website.

4 Website classification

Website classification can not only be divided by the type of classification problem, but also by the classification subject. Webpages can be topic-based or genre-based classified as defined in the paper by Choi and Yao (Choi and Yao, 2005). In our dataset, we classify the pages by content. The categories are: auto, book, camera, job, movie, NBA-player, restaurant and university.
There already exists a wide variety of techniques to classify websites, as stated in section 2.1. We have chosen to use two approaches, a simple keyword-matching model and a text-based neural network, both use the text extracted through the *Beautiful Soup* library.

4.1 Keyword-Matching

Keyword Matching is a simple method to classify data based on a manually prepared batch of words that hint to the category of the text. It is often used to label training data in an easy and efficient way. Keyword Matching is not only easy to implement and expand, but also a nice baseline for classifiers. The text category is decided over the highest matching score of the keyword sets, as displayed in the equation 1 (Karthikeyan et al., 2019).

$$matchingscore = \frac{categoryMatches}{allMatches} \quad (1)$$

In addition to the category and attribute names, related terms that come from a simple vocabulary search on the Internet were used as keywords.

4.2 Text-Classification network

Text classification is an essential subdivision of natural language processing with a range of options to choose from. Our approach uses a neural network for text summary and classification.
To handle text inputs, we have to start out with using a text vectorization layer, which handles unstructured natural language texts extracted from the HTML page and converts them into a tensor containing the vocabulary indices. To obtain an end-to-end model, the text vectorization layer is embedded as the first layer. An explicit maximum sequence length is set because of the convolutional layers.
Following, two one-dimensional convolutional layers extract features from the text vectors and a global maximum pooling is used to summarize them. This method originates from image classification and feature extraction. In a simplified form, a convolutional layer works like a sliding window function on a matrix. Mathematically, it is the sum of an element-wise dot product of the input data and a weight array that is the so-called kernel (Jing, 2020). Global max pooling sub-samples the input and generates just one classification output. A following dropout layer prevents overfitting, so half of the nodes will be randomly reset to zero while training.

We decided to use a hidden vanilla layer before the output layer, so a nonlinear activation function such as rectified linear unit can be applied and backpropagation for the network is enabled.
For the output, a softmax function is used, so a vector of probabilities per category is returned and the highest one is the predicted category. Sparse categorical cross-entropy is our selection for model compilation, as it uses cross-entropy, but the category does not have to be one-hot encoded before being inserted into the model for training. It will return a probability matrix with a probability between 0 and 1 for each class and input. Adam is a suited optimization function for our model, especially due to our use of massive data. It extends gradient descent optimization and its advantages lie in computational efficiency and little memory requirements (Kingma and Ba, 2014).

5 Information extraction

After the webpage has been assigned a category, the attributes to be extracted from the page are determined. This task is attempted to be solved in the following chapter using two different approaches. The first one is a Template-Recognition-Model, or TR-model for short, and the second one is a NER-Model.

5.1 Template-Recognition-Model

By analyzing the HTML structure of a website and considering the corresponding HTML-tag-positions of the attributes, it is possible to learn a template for these websites. The template-recognition model uses this approach to construct a tree, from which each leaf represents an attribute position. For the extraction of attributes for unseen pages, each candidate's position is scored against the leaf positions and the candidate with the highest score gets extracted.

Training

The aim of the learning step is to create a template tree with the positions of the attributes on the webpage. Here, for each attribute one separate tree is generated. The positions of different attributes in one tree would interfere with one another. Henceforth, a template tree describes the structural relationships for one specific attribute regarding the positions on the different websites.
The first step to construct the tree for an attribute is finding the ground-truth in the HTML of a specific website. This step needs a mapping of all visible text and its position to search for the attribute ground truth (see figure 3). The text position mapping is created from the website HTML. It is a recursive process and starts with the body HTML tag as input. From there, each child tag is analyzed. If the child tag contains text, the current HTML position and the preprocessed text are added to the mapping. The method is then called recursively for all child nodes, i.e. all HTML tags underneath. Further, the tags need to be filtered, as it may lead to incorrect results otherwise because not every tag is visible on a website or desired in the mapping. HTML comments are ignored, but also the navigation bar shouldn't be used for the mapping. Some misleading attributes can be contained in the navigation bar, for example the team names of NBA players. Websites with multiple team names in the navigation bar make it impossible for the model to extract the correct tokens. A more detailed description of the problem can be found down below in chapter 5.2.

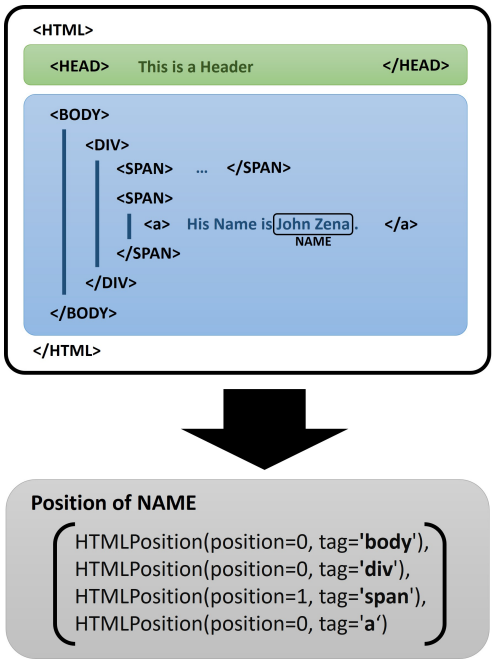

Figure 3: Text position mapping of the HTML-Structure for a website.

In the text position mapping, the ground-truth is searched, and its position is added to the template tree. Additionally, it is needed to differentiate if the complete text under a HTML-tag or only a part of it represents the ground-truth. Therefore, a bitmap is created, which represents the correct words in the found text that represent the ground-truth.

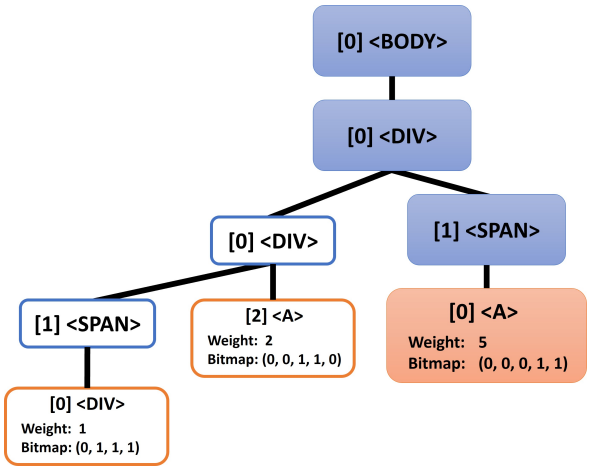

Figure 4: The created mapping for one attribute with the added position of the Name from figure 3

Figure 4 shows a learned template tree. Each node in this tree is a HTML position with a specific tag. Further, a number represents the position in the parent HTML tag. For example, in figure 3 the span tag has position 1, because it is the second child. Each tree leaf also has a corresponding bitmap and a weight factor, which shows how often this leaf was found in websites from the training dataset.

Extraction

For the extraction of an attribute from a given website, the learned template tree for that attribute is needed. To generate possible candidates, the HTML of the website is used to create a text position mapping. The text position mapping follows the same process as specified in the learning step. In a first step, the candidates are filtered to reduce their number. Different filter rules can be applied for different attributes, for example the height attribute is a number, so all candidates without numbers can be removed. The remaining candidates are scored against the leafs of the template tree. The candidate with the highest score gets extracted, and the bitmap is applied. If needed, the top-k candidates can be returned.

The scoring of two HTML positions h_1, h_2 is not a trivial task as the comparison function that checks equality does not consider added HTML tags. An example HTML position can be seen in figure 3 in the lower area with a gray background. If two websites from the same domain, one with an added advertisement, are compared, the position of the attribute is mostly the same, except for one tag position in the beginning. To consider this, a custom scoring function was defined, using n-grams of the position. First, the length difference between the two positions h_1, h_2 multiplied by -2 builds the initial score. After that, the score uses all n-grams of the HTML position h_1 with a length of $n = 5$. For each n-gram g it is checked, if the order of tags exists in the HTML-position h_2. For that, only the tag is considered, not the position-attribute. If this is the case, 1 is added to the score. Additionally, it is checked if the positions of g are the same in the HTML position h_2. Is this true, then 1 is added again to the final score. These tests are performed for all n-grams and in the end the final score is multiplied by the weight of the corresponding leaf node. This result represents the similarity-score between the HTML positions h_1, h_2. With this heuristic method, a computationally expensive alignment of the positions is dispensed and a lot of time is saved.

5.2 NER-Extraction with spaCy

The second approach tries to go the opposite way and deliberately ignores the HTML structure. The reasons for this are the non-constant HTML structures within pages of a domain due to advertising, navigation bars or dynamic content. However, what remains unaffected by all these points is the order of the text on the page in relation to one another. For example, whether the advertising or other elements in the website header change, the text in the main part remains unaffected. In particular, the adjacent words of relevant text excerpts are preserved. The assumption of the second approach is thus that dynamic content changes the HTML structure of the underlying elements, but the adjacent text of the relevant tokens mostly remains the same. This property is considered by a method called Named Entity Recognition (NER), which tries to filter out the entities of a text. If a NER-model is trained on the categories of the desired attributes, the relevant text excerpts from a website may be extracted.

Pre-trained spaCy-model

First, *spaCy* and the provided English language model *en_core_web_md* are used for this, since the web pages of the SWDE dataset are in English and only the medium model has word embedding. The model was trained on English texts from the Internet and can already recognize entities from different categories.

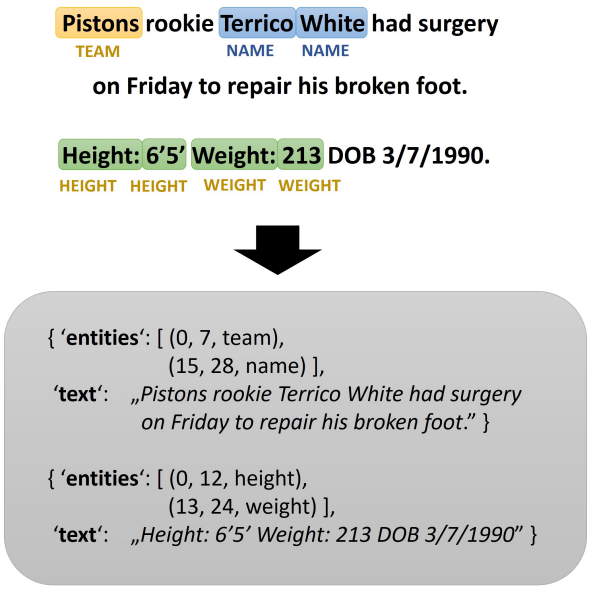

Figure 5: Processing the text of a Website to the desired format for spaCy.

This includes personal names, places or companies. However, there are other attribute categories in the SWDE dataset that cannot yet be extracted. For this purpose, the model must also be trained with additional training examples. To do this, the visible texts on the websites are extracted using the *Beautiful Soup* library and the correct attribute-values from the ground-truth are marked and saved in the following format (figure 5).

Filtering the HTML

The text was limited to the visible text on the website. Comments, script inclusions and other non-visible tags have been filtered out.
Furthermore, it was noticeable that the attributes appeared several times in the text at different places. For example, an NBA player's team name was not only in the player's description, but also in game reports found further down the page. This is not problematic, since the model can also learn to recognize team names from such information. However, the name also appeared in the navigation bar. There it stood alongside all the other team names in the league. This navigation bar does not change for different pages of different players from different clubs. The model thus learns to recognize and mark all team names from the navigation bar. The problem is similar to that in chapter 5.1. The output of the NER-model was therefore very imprecise at the beginning for the team-attribute, since all team names were outputted. The solution was to filter out the navigation bar and the footer of a website.
It should be noted that the spaCy-model can only work with attributes that do not overlap. That means a token can only belong to exactly one category. The data must be tested for this and if there is an overlap, the attribute with the shorter token length is removed.

Pipeline

A language processing pipeline is then set up. Depending on the definition, this includes various steps. For NER there is a supplied pipeline from spaCy, which already includes a tokenizer, lemmatizer and batching. At the end, the pipeline returns a *Doc*-object, which can be used to train the spaCy-model (spaCy).
40 epochs were trained with a batch size of 32. An optimization with regard to batch size and learning rate did not take place.

Candidate determination

The model then assigns a label to each token from the text of the web page. If a token does not belong to a category, a wildcard symbol is used. When several words in a row are given the same label, they are combined and the tokens are considered as a single solution candidate. An example here would be consecutive first and last names, which are then suggested as a coherent solution for the Name attribute.
More than one solution candidate is often found on a website. All candidates are pooled and screened for subsequences. If one candidate is a subset (a substring) of another, the longer candidate is used. Furthermore, the frequency of the candidates is counted, since, for example, a name appears several times on the page and, ideally, is recognized several times by the model. The candidates are ordered by frequency and the top k are returned. The k was set to 3 for the evaluation in this work.

5.3 NER-Extraction with own LSTM-Model

In addition to the spaCy-model, which is based on a pre-trained model, it should also be tested how an own LSTM model works and how it performs in comparison. The reason for this investigation is that the *en_core_web_md* model was trained on continuous text.

However, it is significantly different to the texts that the network receives in this work. Continuous text consists of sentences that follow grammatical rules. This gives adjacent words a clear relationship that can be learned by the model. However, in the extracted texts from the websites, this relationship does not always exist. Since all of the text found on the website is simply concatenated, adjacent words do not always have a relationship and the text appears randomly strung together. It cannot be ruled out that a completely retrained network will cope better with this text structure, which is why it was decided to train an own LSTM model.

The model was created in Python using the Keras (Keras-Team) library. Strictly speaking, it is a BiLSTM model that takes a word-vector as input and outputs the probabilities of each token for belonging to a category. Word embedding is also done using the *en_core_web_md* model provided by spaCy. However, no pre-trained model is used as a basis for training. 256 filters were used in the LSTM layer. It was trained between 3 and 10 epochs with a batch size of 32 (Timmermann, 2020). After considering the results, an optimization with regard to the variable sizes was dispensed.

5.4 Combined Template-NER-Model

Finally, an attempt will be made to combine the theoretical advantages of both approaches, the TR-model and the NER-model. For this purpose, both models are first trained separately. When information should be extracted from a web page, it is first entered into the TR-model. All text in an HTML block is seen as a potential candidate. It is then checked for each candidate whether there is a leaf node with an attribute in the stored structure tree. This would mean that the training dataset already contained a website that had a searched attribute in the same position. If so, the text is extracted with the appropriate bitmask as an attribute.
If there is no match in the structure tree for an attribute, the NER-model is used. The idea is, that the template of the website should be unknown to the TR-model and therefore no assumptions can be made regarding the position of the attributes. For this reason only the text can be used for extracting the correct tokens for the attribute with the NER-model.

6 Evaluation

Following the description of the methods, the approaches presented will now be evaluated to be able to compare their performance. In addition, characteristics of the dataset will be revealed through an exploratory data analysis.

6.1 Dataset

The SWDE dataset is divided into 8 categories. A category contains thematically related websites, such as “Car”, “Restaurant”, or the category "NBA Player", which is primarily used for the following evaluation. In each category the different websites are assigned to one of 10 domains. Examples are "nba.com" or "sports.yahoo.com". Furthermore, each category is assigned three to five attributes, such as "name" or "team", whose values are to be extracted from the website. In total, the SWDE dataset includes more than 124,000 websites with given ground-truth (Hao et al., 2011).

To identify characteristics of the dataset, an exploratory data analysis was first performed for the attributes of all categories on the labeled ground-truth. Crucial metrics for this are the average length of the given solutions (Len.), the average number of words in a solution (Tok.), the average number of different solutions for the same attribute (Sol.), and the proportion of missing solutions (Mis.). The results for the category "NBA Player" can be seen in table 1. Based on this analysis, the category "NBA Player" was chosen to train and evaluate the model approaches because few values are missing and a token length between 1 and 3 is easy to handle in the beginning. Later, the same analysis was performed on the results of the extractions to detect anomalies.
Further analysis of the dataset has shown that some inconsistencies exist between ground-truth and actual text. These include, for example, a different number of spaces, incorrectly extracted protected spaces, or extracted points at the end. Separate errors also occur within individual attributes. For example, in the "Name" attribute, in many cases the player’s position is additionally specified without spaces to the name. Furthermore, in some cases not all correct solutions are listed in the ground-truth. In order to be able to compare the extracted solutions of the models with the ground-truth, a comprehensive preprocessing is performed on all texts.

Attribute	Len.	Tok.	Sol.	Mis.
name	13.45	2.31	1.01	0.00
team	13.32	1.92	0.90	0.10
height	5.49	1.59	1.00	0.00
weight	7.12	1.78	1.00	0.00

Table 1: Exploratory data analysis results for the category "NBA Player" calculated on the preprocessed ground-truth for a better comparison. The average length of a solution (Len.) is represented by the number of contained characters. The average number of tokens (Tok.) reflects the actual number of tokens per solution and attribute. The average number of solutions (Sol.) illustrates the number of solutions per attribute. The proportion of missing solutions (Mis.) is calculated by dividing the number of missing solutions per attribute by the number of attributes in the ground-truth.

On the one hand, punctuations, stop words and wrong spaces are removed. On the other hand, the previously mentioned name errors are corrected by inserting spaces. Finally, lemmatization and lowercasing are applied for the comparison of the solutions. This is necessary to minimize the number of False Negative Extractions, since every smallest deviation of the strings leads to an Exact Match of 0.

6.2 Website Classification Models

Two different types of train-test data splits are used to train and evaluate the classification and extraction models. On the one hand, a website-split is used, where websites are randomly assigned to the train or test set. The model thus knows the domain and patterns on it can be learned. On the other hand, a domain-split is applied to test the model on websites of unknown domains. To measure the performance of the classification models, the metrics recall, precision and F1-score are calculated. The macro-averaging method is used to calculate the metric scores (scikit learn, 2022). For all models, the performance is determined on the basis of the out-of-sample prediction as well as on the basis of the in-sample prediction in order to identify possible overfitting.

6.2.1 Evaluation Keyword-Matching

First, the performance of the Keyword-Matching model will be examined. Since no training data is required for this, only an out-of-sample prediction is performed on the entire dataset. Table 2 shows the evaluation metrics calculated for this purpose. With an F1-score of just under 98%, almost all websites are classified correctly. The worst category is "Movie" with approximately 90% correct classifications. The main error here are false positives, since it is mainly websites in the "Job" category that are incorrectly classified as "Movie".

Metric	Value
Recall	0.9783
Precision	0.9798
F1	0.9783

Table 2: Evaluation results of Keyword-Matching Model. The entire dataset was used for testing the model. The macro-averaging method was used to calculate the scores.

6.2.2 Evaluation Text-Classification network

In addition to Keyword-Matching, attempts were made to achieve even better classification results with the Text-Classification network. In table 3 the results with a website-split as well as with a domain-split are listed. Both models were trained and tested with a train-test-split of 0.7 on a randomly drawn dataset sample of 30,000 websites. Already with this, classification results of over 99% could be achieved with a website-split. The in-sample forecast also hardly deviates from this with an F1-score of 0.9989. Therefore, the train data size was not increased further. The domain-split results with an F1-score of 66% are clearly below those with website-split. When classifying websites of unknown domains, these deviations are to be expected. Here, a clear difference to the in-sample F1-score of 1.0 can be seen.

Metric	website-split	domain-split
Recall	0.9978	0.6809
Precision	0.9975	0.7261
F1	0.9976	0.6619

Table 3: Evaluation of out-of-sample results of the Text-Classification network with both website and domain-split. The macro-averaging method was used to calculate the scores.

Figure 6 shows the confusion matrix of the Text-Classification network with domain-split. It is noticeable that most classification errors occur for the categories "NBA player" and "Restaurant".

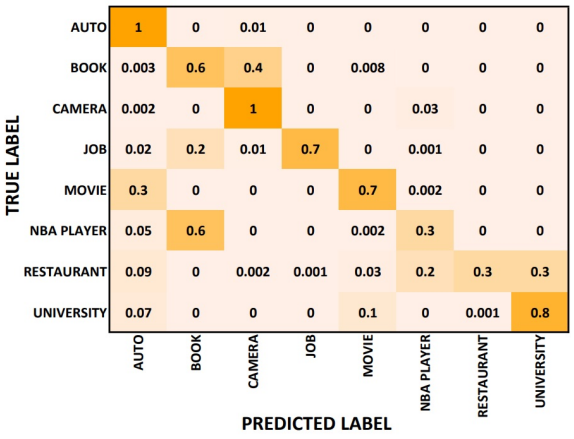

Figure 6: Confusion matrix for the Text-Classification network Model with domain-split. The matrix was normalized over the true labels.

6.3 Extraction Models

The extraction models are evaluated on the basis of string similarities between the ground-truth and the predictions. On the one hand, an exact match is calculated, which reflects the number of predictions that are identical with at least one of the given solutions of the ground-truth. On the other hand, an additional (macro-averaged) F1-score is used for evaluation (Rajpurkar et al., 2016). These two scores have already been applied to evaluate the WebFormer approach for the SWDE dataset (Wang et al., 2022). The error distribution of the WebFormer method has shown that many errors occur by extracting only a partial text (Wang et al., 2022). In order to additionally express the performance by a less restrictive metric and thus classify semantically correct but syntactically slightly deviating solutions as correct nevertheless, a partial match is also computed. This works similarly to the exact match. However, a solution is considered a partial-match if one of the two strings is a substring of the other one and if a value greater than 0.5 is returned when the length of the shorter string is divided by the length of the longer string. Furthermore, stronger preprocessing is applied by removing all spaces beforehand for the comparison. With the partial match it should be excluded as far as possible that false negative classifications occur due to errors in the ground-truth, which could not be eliminated before by the preprocessing.

All metric results are determined in the form of a top 1 and a top 3 evaluation. Top 1 means that only the first and therefore best prediction is considered. In the top 3 evaluation, on the other side, the best three predictions are used to detect whether a model can extract the correct prediction, but weights it incorrectly. With exact match and partial match, a value of 1 is calculated if at least one of the three solutions is a match. With the (macro-averaged) F1-score, however, the maximum of the three values is used.

6.3.1 Evaluation Template-Recognition Model

For the Template-Recognition model, a model with website-split was first trained on the category "NBA Player". Only a small train-test split of 0.15 is used for this, since the model learns the structure of the websites of a domain, which only requires to ensure that the model receives at least one website per domain to train. Table 4 shows the calculated results. It can be seen that the model gives almost the same results for all three metrics. Furthermore, with a value of less than 1 percentage point, there is a very small difference between the top 1 and top 3 results. Thus, the model delivers a correct solution already with the first prediction. In order to exclude the possibility of overfitting, an in-sample evaluation is performed in addition to the out-of-sample evaluation. Therefore, for all models to be evaluated, only the top 1 exact match is specified. The template recognition model achieves an in-sample exact match of 0.7027 and is thus even slightly below the out-of-sample value.

Metric	Top 1	Top 3
Exact Match	0.7205	0.7282
F1	0.7206	0.7283
Partial Match	0.7208	0.7286

Table 4: Evaluation results of Template-Recognition Model for out-of-sample prediction with website-split on category "NBA Player".

The results with domain-split are shown in table 5. It can be seen that for the top 1 evaluation these are 15%, significantly lower than the 72% of the evaluation with website-split. Furthermore, the results deviate significantly from the top 1 exact match of the in-sample evaluation in the amount of 0.7716. These results were expected, since the model learns the structure of the websites of a domain, in this case however, websites of unknown domains are used for testing. It is noticeable that the top 3 values are significantly larger than the top 1 values with approximately 24%.

Metric	Top 1	Top 3
Exact Match	0.1572	0.2442
F1	0.1573	0.2465
Partial Match	0.1575	0.2461

Table 5: Evaluation results of Template-Recognition model for out-of-sample prediction with domain-split on category "NBA Player".

6.3.2 Evaluation NER-Extraction with spaCy

The best extraction results are achieved by the NER-model with spaCy and a website-split. As seen in table 6, about 92% of the extractions are matches. There is only minimal variation between the top 1 and top 3 results. In-sample, the model achieves a top 1 exact match of 0.9211.

Metric	Top 1	Top 3
Exact Match	0.9194	0.9216
F1	0.9209	0.9230
Partial Match	0.9208	0.9232

Table 6: Evaluation results of NER-Extraction with spaCy for out-of-sample prediction with website-split on category "NBA Player".

To evaluate this model, only one model was trained and tested on all websites of the "NBA player" category. In order to exclude that the performance can deviate strongly from this single model run, 4 further models were trained on a dataset sample of 2,000 randomly drawn websites and their standard deviation was calculated. The mean value of the top 1 out-of-sample exact match is 0.9149 with a standard deviation of 0.0093. Therefore, different model runs differ only marginally in the extraction quality.

Similar to the template recognition model, significantly worse results are achieved when the model is tested on websites of unknown domains. The results are shown in table 7. However, the values of approximately 41% to 54% are significantly higher than the results of the Template-Recognition model. It is noticeable that the F1-score is more than 10 percentage points higher than the other two metric results. Thus, the model extracts only a part of the words of a solution correctly. The in-sample prediction reaches a very high value for this setting with a top 1 exact match of 0.9637, which could already be a sign of overfitting. However, no better results could be achieved with shorter training times.

Metric	Top 1	Top 3
Exact Match	0.4065	0.4159
F1	0.5258	0.5352
Partial Match	0.4097	0.4193

Table 7: Evaluation results of NER-Extraction with spaCy for out-of-sample prediction with domain-split on category "NBA Player".

Similar evaluations on the SWDE dataset were performed using the WebFormer method. Here, an exact match of 0.8658 is achieved (Wang et al., 2022). The results of the NER extraction model with spaCy look very promising in comparison. However, it must be noted that the NER spaCy-model has only been evaluated for three of the eight categories. In order to establish a direct comparability, further training efforts have to be made.

6.3.3 Evaluation NER-Extraction with own LSTM-Model

The LSTM model provides very poor results overall. Table 8 shows that the top 1 values are below 1% and also the top 3 values reach a maximum of 1%. Even when considering the in-sample exact match top 1 score, with 0.0024 hardly any correct values are extracted. It is noticeable that only for the attribute "name" values can be extracted at all, whereby these are also mostly wrong, as for example text segments are returned that are far too long. For the other three attributes, the model cannot find any solutions. Due to the poor results, no further evaluation and optimization is performed for this model. Possible reasons for this could be a too small model-size and a too short training. Furthermore, increasing the size of the structure by adding filters in the LSTM layer could lead to better results. However, due to the very good performance of the spaCy-model, no further investigations were carried out here.

Metric	Top 1	Top 3
Exact Match	0.0017	0.0073
F1	0.0032	0.0119
Partial Match	0.0026	0.0097

Table 8: Evaluation results of NER-Extraction with own LSTM-Model for out-of-sample prediction with website-split on category "NBA Player".

6.3.4 Evaluation combined Template-NER-Model

The idea of the Combined Template-NER-model to minimize the weaknesses of the Template-Recognition model and the NER spaCy-model, by a combined consideration, and thus to obtain better results, did not work. As can be seen in table 9, the Combined Template-NER-model achieves very similar results to the NER spaCy-model with scores of about 92%, but performs minimally worse on most metrics. For extracting the solutions, the already presented Template-Recognition Model and NER spaCy-model were used, each with website-split.

Metric	Top 1	Top 3
Exact Match	0.9173	0.9216
F1	0.9193	0.9219
Partial Match	0.9190	0.9218

Table 9: Evaluation results of Combined Template-NER-Model for out-of-sample prediction with website-split on category "NBA Player".

6.3.5 Transferability to other categories

Until now, the models have been tested only on the category "NBA player". In order to determine whether a model can be transferred to other categories without any problems, some additional Template-Recognition models and NER spaCy-models were trained and evaluated for the category "Auto" and "Restaurant" respectively. The results of the out-of-sample exact matches with a website-split are shown in table 10.

Model	Category	Top 1	Top 3
TR.	Auto	0.5813	0.6451
TR.	Restaurant	0.5655	0.6681
NER.	Auto	0.8973	0.9085
NER.	Restaurant	0.9255	0.9413

Table 10: Evaluation results of Template-Recognition Model (TR.) and NER-Extraction with spaCy (NER.) for other categories with website-split. To compare the results the metric Exact Match is used.

The Template-Recognition Model achieves significantly worse results for the two categories than for the "NBA player" category. In addition, significant differences between the top 1 and top 3 results can be seen. The NER spaCy-model achieves very similar results for the three categories tested, with only minimal differences between top 1 and top 3 results at the same time. The NER spaCy-model thus seems to be very well transferable to other categories, while category-dependent adaptations should be considered for the Template-Recognition model.

7 Conclusion

In this work an attempt is made to extract information from websites. The SWDE dataset was used for this purpose.
Each website is first classified to the categories specified by the dataset. A simple model that searches for keywords on the website has already been able to achieve sufficiently good results here. 98% of the approximately 30,000 websites were correctly classified with this. Further approaches for classification with neural networks were tested and precision and recall could be increased to 0.998 if websites from the same domain were already included in the training data. For pages from unknown domains, F1-Score dropped to 0.7, making keyword search superior in the general use case, especially with its faster and computationally less difficult execution.

For each category, a model will then try to extract the specific information. This thesis aims to test two different approaches. The first one relates to the structure of a page's HTML tree to be learned. The actual text is here not considered. By learning the positions of the text excerpts, an exact match score of 0.72 could be achieved. However, this model only works for already known domains. A test with websites from unknown domains quickly showed that the performance drops drastic to 0.16. The second approach is to go the opposite way and only receive the text of a page without the structure from the HTML tree and extract the information using a trained NER-model (Named Entity Recognition). Due to a presumably too small training dataset and too little training time, our own BiLSTM model could not deliver any significant good results. Another model, which is based on the open-source library spaCy, was able to achieve significantly better results. 92% of all attribute values were correctly extracted from the webpages, which is about 8 to 10 percentage-points better than the state-of-the-art techniques found in the literature. This required pre-processing of the values, as there were large discrepancies between the dataset and the actual websites.

It should be noted that only 3 of the 8 categories were evaluated. Furthermore, it is conceivable that the results achieved in this work is not yet the optimum for the used NER technique, since no parameter optimization and fine tuning were carried out with regard to the training of the model. Additionally, due to the age of the dataset, an evaluation of the technology on more up-to-date websites would be recommended for future work. Finally, an attempt was made to combine the two models in order to use the advantages of both technologies. However, the NER-model was already so good that no further significant improvement could be achieved with the additional TR-model.

It should be emphasized again at this point that the NER-model from spaCy was already pre-trained for certain attributes. *Names* could already be recognized by the model in texts before. If the text extractions on unseen websites for the attributes of this work are considered separately, it is noticeable that *names* can even be extracted correctly with over 99%. This suggests that the other attributes can also be recognized much better if the model is trained on them with comparably large resources, as it was the case with *names*. The use of NER networks can thus be a very important and possibly the crucial technique for information extraction from websites and should be researched and investigated further.

Another idea for further work would be to test whether a separate classification is necessary in advance. In theory, the NER-model also recognizes attributes without knowing that they must be included. The category could then be inferred from the extracted attributes if desired. However, this would require a much larger model with more training data, since it would have to include websites of all 8 categories.

Reproducibility

Our complete code basis and our results of the evaluation process can be found in our GitLab repository [1].
We used the SWDE dataset from Hao et al.. It can be found on this website [2].

[1] https://git.informatik.uni-leipzig.de/jb64vyso/web-b-gone

[2] https://academictorrents.com/details/411576c7e80787e4b40452360f5f24acba9b5159

Author contributions

Jan Braker: Project Structure, Dataset Re-Structuring, Template-Recognition-Model, Model Optimization
Lorenz Heinemann: Research and Related Work, Template-Recognition-Model, NER-Extraction with spaCy, NER-Extraction with own LSTM-Model, Combined Model
Tobias Schreieder: Research, Dataset-Exploration and Analysis, Data Preparation, Model Evaluation and Optimization
Josephine Selig: Research and Related work, Website-Classification with Keyword Matching, Website-Classification with a Text-Classification network

References

Maher Ragheb Aburrous, Alamgir Hossain, Keshav Dahal, and Fadi Thabatah. 2009. Modelling intelligent phishing detection system for e-banking using fuzzy data mining. In *2009 International Conference on CyberWorlds*, pages 265–272.

Sumaia Mohammed Al-Ghuribi and Saleh Alshomrani. 2013. A simple study of webpage text classification algorithms for arabic and english languages. In *2013 International Conference on IT Convergence and Security (ICITCS 2013)*, pages 1–5, Piscataway, NJ. IEEE.

Andrew Carlson and Charles Schafer. 2008. Bootstrapping information extraction from semi-structured web pages. In Walter Daelemans, Bart Goethals, and Katharina Morik, editors, *Machine learning and knowledge discovery in databases*, volume 5211 of *Lecture notes in computer science Lecture notes in artificial intelligence*, pages 195–210. Springer, Berlin.

Chia-Hui Chang, Mohammed Kayed, Moheb Girgis, and Khaled Shaalan. 2006. A survey of web information extraction systems. *Knowledge and Data Engineering, IEEE Transactions on*, 18:1411– 1428.

Ben Choi and Zhongmei Yao. 2005. Web page classification. In *Foundations and Advances in Data Mining*, pages 221–274. Springer.

Rand Fishkin. 2018. Google ctr in 2018: Paid, organic, & no-click searches. [Accessed 27-Aug-2022].

Rand Fishkin. 2020. In 2020, Two Thirds of Google Searches Ended Without a Click - SparkToro — sparktoro.com. [Accessed 27-Aug-2022].

John Foley, Michael Bendersky, and Vanja Josifovski. 2015. Learning to extract local events from the web. In *Proceedings of the 38th International ACM SIGIR Conference on Research and Development in Information Retrieval*, ACM Digital Library, pages 423–432, New York, NY. ACM.

Tomas Gogar, Ondrej Hubacek, and Jan Sedivy. 2016. Deep neural networks for web page information extraction. In Lazaros Iliadis and Ilias Maglogiannis, editors, *Artificial intelligence applications and innovations*, volume 475 of *IFIP Advances in Information and Communication Technology*, pages 154–163. Springer, Cham.

Qiang Hao, Rui Cai, Yanwei Pang, and Lei Zhang. 2011. From one tree to a forest. In *Proceedings of the 34th international ACM SIGIR conference on Research and development in Information Retrieval*, ACM Conferences, page 775, New York, NY. ACM.

Adnan Hodžić, Jasmin Kevrić, and Adem Karadag. 2016. Comparison of machine learning techniques in phishing website classification. In *International Conference on Economic and Social Studies (ICESoS'16)*, pages 249–256.

Hong Jing. 2020. How convolutional layers work in deep learning neural networks? [Accessed 29-Aug-2022].

T Karthikeyan, Karthik Sekaran, D Ranjith, JM Balajee, et al. 2019. Personalized content extraction and text classification using effective web scraping techniques. *International Journal of Web Portals (IJWP)*, 11(2):41–52.

Keras-Team. Keras: the Python deep learning API — keras.io. [Accessed 28-Aug-2022].

Diederik P. Kingma and Jimmy Ba. 2014. Adam: A method for stochastic optimization.

Oh-Woog Kwon and Jong-Hyeok Lee. 2000. Web page classification based on k-nearest neighbor approach. In *Proceedings of the Fifth International Workshop on on Information Retrieval with Asian Languages*, IRAL '00, page 9–15, New York, NY, USA. Association for Computing Machinery.

Dipl.-Ing. (FH) Stefan Luber. 2022. Was ist Named Entity Recognition (NER)? [Accessed 28-Aug-2022].

Alireza Mansouri, Lilly Affendey, and Ali Mamat. 2008. Named entity recognition approaches. *Int J Comp Sci Netw Sec*, 8.

Rami M Mohammad, Fadi Thabtah, and Lee McCluskey. 2014. Predicting phishing websites based on self-structuring neural network. *Neural Computing and Applications*, 25(2):443–458.

Pranav Rajpurkar, Jian Zhang, Konstantin Lopyrev, and Percy Liang. 2016. Squad: 100,000+ questions for machine comprehension of text.

Lassri Safae, Benlahmar El Habib, and Tragha Abderrahim. 2018. A review of machine learning algorithms for web page classification. In *2018 IEEE 5th International Congress on Information Science and Technology (CiSt)*, pages 220–226.

scikit learn. 2022. sklearn.metrics.f1_score. [Accessed 28-Aug-2022].

spaCy. Language Processing Pipelines · spaCy Usage Documentation — spacy.io. [Accessed 28-Aug-2022].

spaCy. 2016a. EntityRecognizer · spaCy API Documentation — spacy.io. [Accessed 28-Aug-2022].

spaCy. 2016b. spaCy · Industrial-strength Natural Language Processing in Python — spacy.io. [Accessed 28-Aug-2022].

Yong-Hong Tian, Tie-Jun Huang, and Wen Gao. 2004. Two-phase web site classification based on hidden markov tree models. *Web Intelligence and Agent Systems: An International Journal*, 2(4):249–264.

Thomas Timmermann. 2020. Take control of named entity recognition with your own Keras model! — blog.codecentric.de. [Accessed 28-Aug-2022].

Qifan Wang, Yi Fang, Anirudh Ravula, Fuli Feng, Xiaojun Quan, and Dongfang Liu. 2022. Webformer: The web-page transformer for structure information extraction.