HOCHSCHULE FÜR ANGEWANDTE WISSENSCHAFTEN HAMBURG
Hamburg University of Applied Sciences

Development and software implementation of a preliminary model to identify the probability of search engine optimization on webpages

SEO-Effekt Working Paper 3
Version 0.6

Sebastian Sünkler & Dirk Lewandowski
in collaboration with Nurce Yagci

SEO-Effekt

searchstudies.org/seo-effekt

Abstract

This working paper describes the development and implementation of the software to determine the probability of search engine optimization of a web page. The software tool is written in Python with various programming libraries to scrape data from the web, analyze source codes of URLs, analyze defined indicators, perform retrieval and text matching tasks, and write and read data from a PostgreSQL database. We use a model-view-controller pattern to ensure the flexibility of the software in terms of extending its capabilities and replacing modules. Probability analysis is performed by a rule-based classifier built on our model of potential SEO indicators defined by the state of research in the project (expert interviews, literature reviews, and experiments with multiple test cases). The current state of the software allows us to determine whether SEO actions have been taken for a given URL. We collect URLs either by scraping commercial search engines or by uploading lists of web pages. For each URL, we analyze the HTML code, from which we extract information using SEO and analysis tools. We also extract SEO indicators at the page and site level (e.g., page descriptions and page speed). We complement this approach by using lists of manually classified websites. An analysis based on three datasets with a total of 2,043 search queries and 263,790 results shows that a large proportion of the pages found in Google are at least probably optimized, which is in line with statements by SEO experts that it is very difficult to gain visibility in search engines without applying SEO techniques.

Author: Sebastian Sünkler & Dirk Lewandowski

In collaboration with: Nurce Yagci

Institution: Hamburg University of Applied Sciences (HAW)

Faculty of Design, Media, Information

Department of Information

Finkenau 35

22081 Hamburg

E-Mail: sebastian.suenkler@haw-hamburg.de

dirk.lewandowski@haw-hamburg.de

nurce.yagci@haw-hamburg.de

www.searchstudies.org/seo-effekt

Table of Contents

Figures and Tables

1 Introduction

Within the research project SEO-Effekt we aim to describe the external influence on the search engine results through search engine optimization (SEO) from the perspective of the participating stakeholder groups. We use a multimethod approach to conduct our research broken down into to study parts (see Figure 1): One part is about the user perspective and uses methods like expert interviews (Schultheiß 2021b), representative online studies (Schultheiß 2021a) to get an understanding about the users on two levels. On the first level are the users of methods on their web content to get better positions in the ranking on Search Engine Result Pages (SERPs). On the other level we examine the actual users of search engines to get a better understanding about their opinions and their knowledge of Search Engine Optimization in general. The second part is focused on data analysis and software development. In this part, we develop a data model and a classifier to determine the probability of Search Engine Optimization on search result pages of major search engines. Both parts are not independent from each other, e.g., the results of the expert interviews are important to extend the preliminary model of potential SEO factors which are necessary to develop methods to identify the degree to wthich these results have been optimized.

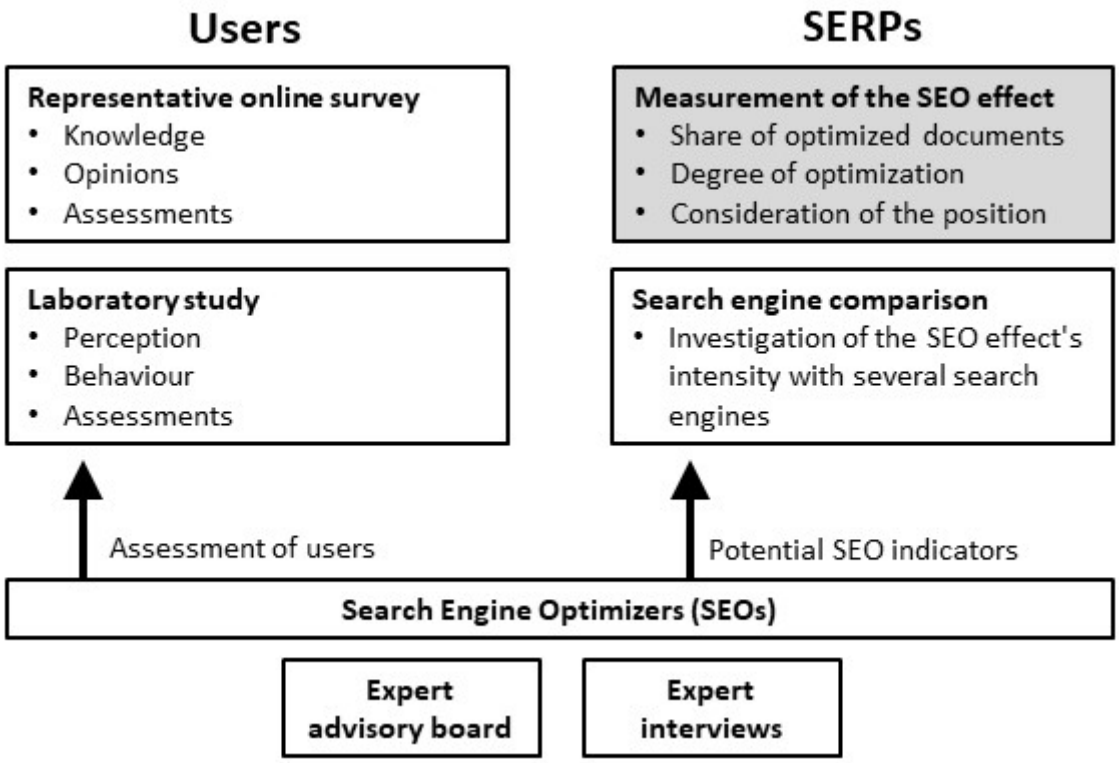

Figure 1: Interplay of the study components

This working paper describes the research on processing and evaluating indicators to determine the use of SEO methods, based on search results. The paper is organized as follows: First, we describe the current data model and our rule-based classifier for measuring the probability of search engine optimization on a URL. We describe the indictors and explain which of them we use for our classification. We present the results and discussion of our evaluation of potential SEO indicators in terms of their meaningfulness, measurability, and ease of technical implementation. We also provide a market research of plugins and tools used for optimizing web documents with respect to search engine optimization, as well as our research results on classifying websites into specific categories, which are useful to make assumptions about the use of SEO on websites to a specific category (e.g., informed by the literature and our interviews with SEO experts, we assume that news services on the web are most likely to use measures of SEO to get better positions in search engines). The following section details our approach to identifying the likelihood of SEO on web pages and our initial results of using machine learning algorithms to improve our approaches. We then describe the software architecture, database model, and current state of the various apps and scripts we developed for scraping the major search engines, extracting SEO indicators from HTML sources, and classifying search results with respect to SEO usage. The next section deals with various test cases and studies that we conducted to support the software development and for some exploratory analysis to gather information for the development of the data model. Finally, we summarize the results and its implications for the final data model to be used for all further research on measuring the SEO effect.

2 Identifying SEO on a webpage

Our discussion of SEO factors and their implementation in our system is based on an extensive review of the professional literature (e.g., Enge, Spencer, and Stricchiola 2015; Erlhofer 2019) and interviews with SEO experts (Schultheiß and Lewandowski 2021). Based on the discussion, we first developed a model of possible indicators (see Figure 2) and evaluated these indicators in terms of their validity and the possibility to implement them in our software. In total, our model consists of 47 factors that we prioritized for implementation in our system. While the model from the interviews contains 62 indicators, after evaluating these indicators we decided to implement 47 of them in our software. The criteria for selecting these indicators were primarily based on whether they could be captured using our technical approaches. Therefore, we initially focused on so-called onsite factors, most of which can be extracted from HTML source code. When compiling the indicators, we also decided to partially orient ourselves only on the categories from the model and to further differentiate some of the features mentioned there or to derive other indicators from them. One of our indicators, for example, is whether descriptions (i.e., the HTML description tag) are used. This indicator was not mentioned by the experts in the interviews. However, not using a description clearly indicates that no SEO measures have been taken, as descriptions are a standard practice in SEO.

In the current state of development, we use 20 factors from the list for our rule-based classifier to determine the likelihood of SEO on a web page. However, since these are the ones that have been deemed most fruitful by experts and researchers, we are confident that we can reliably identify optimized content already at this stage.

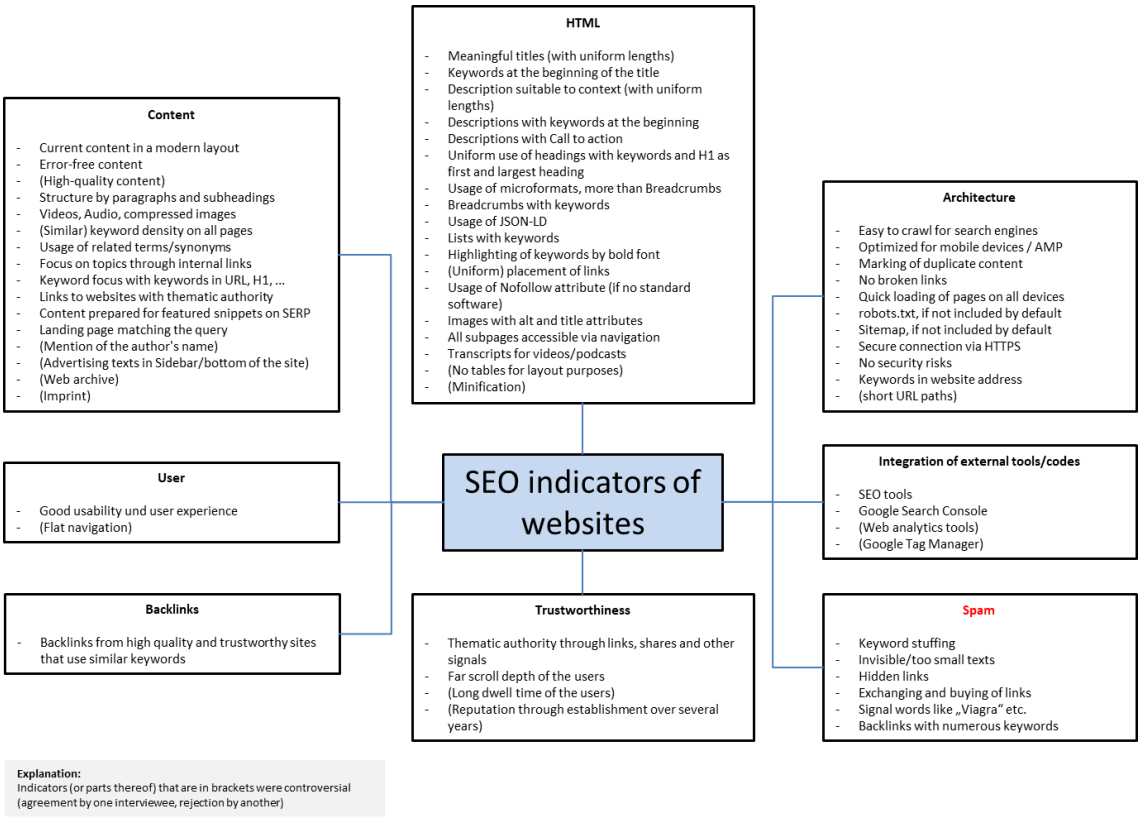

Figure 2: SEO Indicators of websites (Schultheiß 2021b)

The SEO indicators are very important for our approach to identify SEO, but they are only a part of it. Our approach combines automatic identification of SEO indicators from web pages and websites with manually created lists of optimized and non-optimized websites. We also use machine learning methods (see section 2.6) to evaluate better ways to perform the classifications. The main initial goal is to better evaluate the quality of the indicators. For example, we want to evaluate the extent to which we can use the features from the collected data for our classifications without the help of list matching. We are also using algorithms to determine the quality of the data. Our approaches for semi-automatic identification of SEO on web pages and the machine learning approaches used are discussed in detail in the following sections.

2.1 Semi-automatic approach to identify SEO on a webpage

Our approach to identifying search engine optimization on a webpage is a combination of analyzing relevant SEO indicators from webpages and the classification of URLs based on categories with optimized and not optimized websites. It is important to stress here that the aim is not to measure whether these SEO approaches have been *successful* (i.e., the page actually receives a higher visibility). The approach is illustrated in Figure 3 and can be divided into three major steps:

1. Fetching the URL to extract the HTML code and metadata
2. Generating the input for the rule-based classifier on three stages and
3. Determining the probability of SEO on the URL.

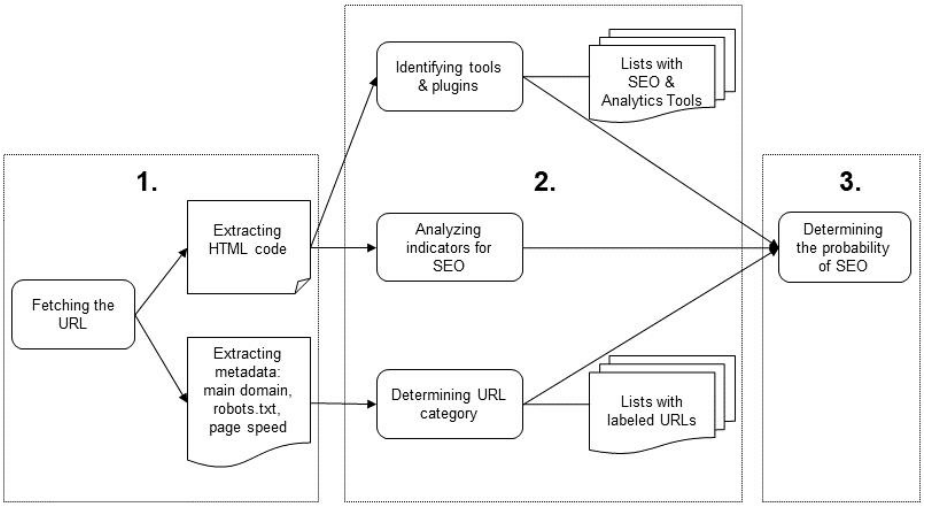

Figure 3: Semi-automatic approach to identify SEO on a webpage

In the first step, our software scrapes search engine result pages and saves the URLs. Thereby, the HTML source code as well as some additional data like the main domain, the content of robots.txt and the loading speed of the URL are captured. Subsequently, SEO indicators as well as information on SEO plugins and analytics tools are extracted from the source code and the URL is matched with lists containing optimized and non-optimized documents. The results from this step are the basis for the final evaluation. The individual processes from the second and third steps are explained in more detail below.

2.2 Identification of plugins and tools

For the identification of SEO plugins and tools, we have created lists that contain both the name of the plugin and a search pattern to find clues about its use in the source code of the URL. A use of SEO plugins is a clear indication that optimization measures are being used. Web analytics are a very

good indicator of a commercial interest on the part of the provider, and since it is precisely commercial providers that carry out measures for good positions in search engine rankings, analytics tools are also a good indicator. In the case of SEO plugins, the list currently includes 58 plugins[1] and in the case of Analytics tools, 54 tools have been manually identified so far[2] in a dataset of 30,000 stored source codes. The following excerpts from the HTML source code show one example each for an SEO plugin and an Analytics tool:

SEO-Plugin: Yoast SEO Plugin[3]

HTML-Code: <!--This site is optimized with the Yoast SEO plugin v12.4 - https://yoast.com/wordpress/plugins/seo/-->

Search pattern: "*yoast seo*"

Analytics Tool: Google Analytics[4]

HTML-Code: <!-- Google Analytics --> <script>(function(i,s,o,g,r,a,m){i['GoogleAnalyticsObject']=r;…

Search pattern: "*google analytics*"

SEO Plugins and Analytics Tools are the most important indicators so far but we also examined the 30,000 stored source codes to evaluate the use of plugins and tools from the following categories:

- **Advertisement Services:** Commercial websites tend to use advertisement services, e.g., Google Ads. We assume that if we found such a service on a website that the provider has a commercial intent and so it is likely that providers with such intent use search engine optimization. As serving advertisements on the web is concentrated towards a few major players, our list consists of 4 advertisement services[5].
- **Caching Tools:** The loading speed of a website is important to enhance the usability and the joy of use of a website and it is relevant for the ranking position on a Search Engine Result Page. These

[1] https://osf.io/7w9ha/

[2] https://osf.io/5dez6/

[3] https://yoast.com/wordpress/plugins/seo/

[4] https://analytics.google.com/analytics/web/

[5] https://osf.io/9xes5/

reasons let us assume that the use of tools to cache the content on a webpage is an indicator for SEO, too. We identified 14 tools that enhance the loading speed so far[6].

- **Content and Microdata Tools:** Some webpages use tools to manage and enrich the content on their webpages. Such tools support tagging content, e.g., Microdata-Markup according to schema.org or to support the accessibility and responsivity. The list of identified tools consists of 25 entries[7]. We also defined Microdata-Markup as especially relevant for SEO. We save findings of microdata as a separate indicator.
- **Social Plugins:** The presence of content on social media is also very important for content providers on the web. Therefore, we evaluated the use of social media plugins on webpages, too. Such plugins help to share content on several social media platforms. Our list of such tools consists of 5 plugins[8].

2.3 Categorization of the site based on known domains

For our approach, we combine automatic identification of SEO indicators from web pages and websites with manual generation of lists of optimized and non-optimized websites. We follow this approach because search engine results are not evenly distributed, i.e., there are only a relatively small number of websites that make up a large portion of the URLs shown (Petrescu 2014) and clicked in the top results (Goel et al. 2010).This means that by manually classifying a limited number of websites, we can already detect a relatively large portion of optimized pages. For the classification, we compiled lists of websites by category, which we subdivided based on the likelihood of search engine optimization in the categories. In total, six categories were defined. The first category is a list of clients of SEO agencies, since it ist save to say that clients of such agencies perform search engine optimization. The agencies were identified by researching specialist portals, and the websites for the list were determined on the basis of customer references on the agencies' sites. This approach enabled a total of 1,004 websites to be identified. Another category refers to websites that most definitely do not perform search engine optimization. Since our approach is still under development, it is difficult to make statements about such sites. Therefore, only Wikipedia is currently in this list, as it is known from Wikipedia that no active SEO measures are used there. For the other categories, a data set with 13,403 documents was evaluated manually. The URLs were classified and divided into the remaining categories. We started from the possible interests of the page operators. Thus, it is defined that clear commercial intentions increase the probability of SEO. The category that contains the most documents

[6] https://osf.io/267cs/

[7] https://osf.io/5jq4f/ & https://osf.io/h3v9z/

[8] https://osf.io/xvmz5/

is news websites such as spiegel.de. A total of 1,203 news sites were found in the data set[9]. The other categories are online stores (178 pages)[10], websites that display advertisements (325 pages)[11] and business websites (72 pages)[12]. However, the lists are not mutually exclusive. Overlaps are possible. The lists are extended continuously.

2.4 Analysis of the SEO indicators

We extract information directly from the HTML source code of the pages and also store other features such as the domain, the contents of robots.txt and the page speed (time elapsed until the page is fully loaded). The data obtained from the source code is, for example, the page title (Title Tag), the page description (Description Tag) and Nofollow links. Furthermore, it is checked whether a sitemap is available. With regard to the robots.txt, typical content is searched for that indicates SEO. A robots.txt is used to give instructions to search engine crawlers. We measure whether it is present and contains SEO-relevant instructions (e.g., explicit exclusions of content on the domain for the crawlers). Finally, we measure the speed of the pages, as one of the fundamental technical factors in search engine optimization is optimizing the pages for fast loading. The advantage in this approach is that we can act independently of external indicators, since we only access directly available technical information. The combination with the evaluation of the plugins and tools as well as the categorization of the URL by matching it with known domains creates a basis for the assessment of search engine optimization. A complete overview of the features used for our rule-based classification can be found in Table 1. The complete lists of all indicators we collect can be found in the project's OSF repository[13].

Table 1: Current list of indicators to determine the probabilty of SEO

Indicator	Description
Tools and plugins	
SEO Tools	Tools that dedicatedly support SEO measures, e.g., Yoast SEO Plugin
Analytics Tools	Tools that are used for website analytics, e.g., Google Analytics
URL lists	
SEO customers	Customers of search engine optimization agencies (manually collected; 1,004 items)
News websites	List of news websites (manually collected; 1,203 items)
Online shops	List of websites (manually collected; 178 items)
Business websites	List of business websites (manually collected; 72 items)
Websites with ads	List of websites showing ads (manually collected; 325 items)
Not optimized	List of websites know not to be optimized (manually collected; 1 item)

[9] https://osf.io/e4djx/

[10] https://osf.io/v4axp/

[11] https://osf.io/pmh6e/

[12] https://osf.io/gqkfx/

[13] https://osf.io/vhsyg/

Indicators for SEO	
Microdata and schema.org	Use of microdata or schema.org on a website to define the context of the data, e.g., JSON-LD
Online advertisements	Use of contextual and affiliate marketing on a website, e.g., Google Ads
HTTPS	Usage of Hypertext Transfer Protocol Secure
SEO in robots.txt	SEO indicators in robots.txt of a website, e.g., crawl-delay
Sitemap	Use of a sitemap on a website
Viewport	Definition of a viewport for a responsive design
Nofollow links	Use of tags on the website to instruct search engines to ignore the target of the link for ranking purposes
Canonical links	Use of canonical tags on the website to prevent duplicate content issues
Pagespeed	Loading time of a website
Description	Use of a site description
Title	Use of a site title
Open Graph Tags	Use of Open Graph Tags for social media

2.5 Rule-based Classifier

We built a system that automatically queries Google, collects result URLs and result positions from the SERPs, collects the result documents, analyzes the HTML code, and checks the URLs against our database of already known websites. We will describe the technical implementation in this paper in chapter 3. Results are classified according to the following rules:

- A result is **most probably optimized** when either an SEO tool was found in the HTML code, it is on the list of news websites, on the list of customers of SEO agencies, or is on the list of websites with ads, or has at least one microdata. Apart from the manually classified sets, we use microdata as an indicator here as microdata is used particularly for SEO purposes.
- A result is **probably optimized** when it is not classified as definitely optimized, the item is not classified as not optimized, and it fulfills at least one of the following criteria: (1) It is on the list of shops or business websites, (2) it uses Analytics Tools or advertisement services like gujAd, (3) it uses https, (4) it has SEO indicators in its robots.txt, e.g., crawl-delay or Sitemap, (5) the website has a sitemap, (6) a viewport is defined, (7) it has at least one nofollow link or canonical link, (8) its loading time is less than 3 seconds.
- A result is **probably not optimized** when it is not definitely optimized, it is not classified as not optimized, and it fulfills at least one of the following criteria: (1) the description tag is empty, (2) the title tag is empty, (3) it has no Open Graph tags.
- A result is **most probably not optimized** when it is on the list of not optimized websites.

Our rule-based approach is under continuous evaluation and is constantly being improved with tests and through machine learning. Currently, very simple rules and indicators for ranking web pages have

been identified and defined. With further work, rules will be refined, also to allow a better gradation of the degree of probability. At the end of the development, a hybrid system should emerge in which both classifications based on machine learning and rule-based classification complement each other to determine a more accurate probability. In addition, on the basis of further analyses, regression analyses and determination of probability scores from the classifications as information on the SEO probability is also to be made possible.

2.6 Machine learning approaches

While we already use the rule-based approach based on relevant SEO indicators in practice, we also conduct research using machine learning (ML) techniques to provide a better assessment of how search engine optimization is used on websites. For our evaluations, we used a dataset that was collected until December 2020. This dataset is available on OSF and was used to gain initial insights into the use of search engine optimization in various subject areas (see test cases reported in section 4).

Evaluation of the dataset

The dataset used has 457,851 entries with 63 features. The data type of 52 features is numeric, while 12 contain non-numeric values. Predominantly, machine learning algorithms can only process numeric values, so this differentiation is necessary. Fifteen of the 63 features are determined with the help of references. These references are separate lists that are manually created and therefore introduce an imprecision factor into the data. These characteristics will be referred to as "external characteristics" in the following. An example of an external feature is "source_shop". It is a binary feature and makes a statement about whether the respective search results are an online store. For this purpose, a csv file is referenced in which domains of known online stores are listed. If the domain of the search result matches a domain in the list, the value of "source_shop" is set to true. Since the list is not a complete list, this feature is not as reliable as features that can be formed from the content of the search result.

Class distribution as input for the ML-Evaluation

The rule-based classification of search results uses 20 features to classify search results into the following classes:

Not optimized (0)

Probably not optimized (1)

Probably optimized (2)

Most likely optimized (3)

250,155 of the entries were classified as "most likely optimized", 203,091 as "probably optimized", 657 as "probably not optimized" and 3,948 as "not optimized". The distribution of classes is not evenly distributed. This must be taken into account when evaluating the algorithms and ML models. Four metrics are used to evaluate the metrics. These are matched to the dataset.

Selection of ML-Algorithms

The scikit-learn library provides three options for calculating Precision, Recall, and F1 for a multi-class classification: micro, macro, and weighted. These options refer to the method of calculating individual metrics for each class to a mean value. While the weighted method is intended to compensate for unbalanced data, this method places a greater emphasis on larger classes. Therefore, the macro method is better suited to make a statement across classes. The 11 selected algorithms are tested by a 5-fold cross validation. Here, the data set is split using the StratifiedShuffleSplit method. This method ensures that the class distribution is the same in all splits. The test size is 66%, thus the size of the training dataset is 33%. Although the GradientBoostingClassifier performs best in all metrics with over 99%, the computation time is much longer than other algorithms. The GaussianNB classifier is therefore used for all data preprocessing steps.

Table 2: Selection of ML-Algorithms

Algorithm	Balanced Accuracy	Precision macro	Recall macro	F1 macro
GradientBoostingClassifier	99,74 %	99,23 %	99,74 %	99,49 %
GaussianNB	97,04 %	75,75 %	97,04 %	75,85 %
BernoulliNB	93,91 %	73,30 %	93,91 %	73,44 %
DecisionTreeClassifier	89,33 %	91,98 %	89,33 %	90,51 %
LinearSVC	80,39 %	80,25 %	80,39 %	77,89 %
ExtraTreesClassifier	79,48 %	96,15 %	79,48 %	82,48 %
RandomForestClassifier	79,14 %	98,80 %	79,14 %	82,18 %
AdaBoostClassifier	69,79 %	71,76 %	69,79 %	68,91 %
KNeighborsClassifier	53,56 %	58,96 %	53,56 %	54,59 %
SGDClassifier	51,47 %	52,31 %	51,47 %	50,45 %

SVC	31,49 %	38,84 %	31,49 %	29,59 %
RadiusNeighborsClassifier	4,50 %	75,27 %	3,60 %	6,71 %

Data preprocessing

Missing values

The feature "position" describes the position of the search result on the SERP. This column is empty in about 7.7% of the entries in the data set. Two other features also have missing values, but these contents were coded with a negative value. Table 3 shows how removing error coding improves performance. Again, 5-fold stratified cross-validation was used, with a test size of 66%. The most common techniques to handle missing data are compared. These are encoding the missing values with a placeholder (in this case -1), imputing the missing value by the mean of the feature or removing the sample in which a value for one feature is missing. Removing samples with missing data has the biggest positive impact on the data, increasing the accuracy from 96% to 99% and the f1 from 75% to 80%.

Table 3: Comparision of methods to find the best way to handle missing values

Method	Balanced Accuracy	Precision macro	Recall macro	F1 macro
error encoded (-1)	97,04 %	75,75 %	97,04 %	75,85 %
errors dropped	97,52 %	78,05 %	97,52 %	80,44 %
errors imputed (mean)	93,14 %	74,99 %	93,14 %	71,80 %

Scaling of values

Machine Learning algorithms generally perform better when the values are on an even scale. As such, different methods of scaling the data are compared. Since most of the features in the data set are either binary (0 or 1) or have very low variance (except for speed), the algorithm performs best when no scaler is applied.

Table 4: Selection of method for scaling values

Method	Balanced Accuracy	Precision macro	Recall macro	F1 macro
no scaling	97,04 %	75,75 %	97,04 %	75,85 %
MaxAbsScaler()	95,95 %	75,47 %	95,95 %	74,33 %

MinMaxScaler()	95,95 %	75,47 %	95,95 %	74,32 %
StandardScaler()	95,80 %	75,44 %	95,80 %	74,16 %
Normalizer()	77,77 %	56,02 %	77,77 %	59,66 %

Class balancing

The dataset is highly imbalanced, with a class distribution of roughly 48, 48, 1 and 1% per class. As such, methods to over or under sample these classes are tested. A combination of both SMOTE and Tomek, in which the higher classes are under sampled and the lower classes are over sampled, boosts f1 from 75% to 97% while the accuracy score also increases from 96% to 99%.

Table 5: Selection of method for balance the classes

Methode	Balanced Accuracy	Precision macro	Recall macro	F1 macro
no sampling	97,04 %	75,75 %	97,04 %	75,85 %
RandomOverSampler	97,05 %	97,08 %	97,05 %	97,04 %
RandomUnderSampler	96,72 %	96,72 %	96,72 %	96,70 %
SMOTETomek	97,80 %	97,84 %	97,80 %	97,80 %

Feature reduction

The Gaussian Naive Bayes Algorithm was also used to reduce the number of features that are used for the model. Different approaches to feature reduction are tested and compared. The benchmarks are the full data (Accuracy: 99,48%, F1: 99,47%) and the data with external features removed (Accuracy: 81,62%, F1: 79%).

The following sklearn methods to reduce features are tested: SelectKBest (k=30), SelectPercentile (percentile=70%), SelectFpr(alpha=0.05) with both ANOVA and Chi-Squared as parameters, as well as VarianceThreshold(threshold=0). The lowest number of features in combination with the highest Accuracy and F1 is SelectKBest_ANOVA. With 30 features the model scores an accuracy of 99,59% and F1 of 99,59%.

Table 6: Comparision of methods to reduce the features

method	feature_count	accuracy_mean	f1_mean
SelectKBest_ANOVA	*30*	*0.995963*	*0.995920*
SelectPercentile_ANOVA	34	0.995761	0.995718

SelectPercentile_chi2	34	0.995093	0.995051
VarianceThreshold	49	0.994816	0.994738
SelectFpr_ANOVA	49	0.994816	0.994738
SelectFpr_chi2	49	0.994816	0.994738
SelectKBest_chi2	30	0.971516	0.970796
External Features	34	0.816252	0.799008

To find the ideal number of features, SelectKBest is tested with k = 1 to k = 49. The best result is k=42, with an accuracy and f1 of 99.66% each. k=42 includes 12 of 15 external features.

In the line charts, three distinct increases of f1 are visible. k = 6 (acc: 93.8%, f1: 93.5%, ext_f: 2), k = 25 (acc: 97.28%, f1: 97.2%, ext_f: 5) and k = 29 (acc: 99.59%, f1: 99.59%, ext_f: 7). The selection at this step needs to take f1 into account but also the number of external features and how important the reduction of features is. Reducing the features gives us the opportunity to enhance our model and to boost the overall performance of our classification processes.

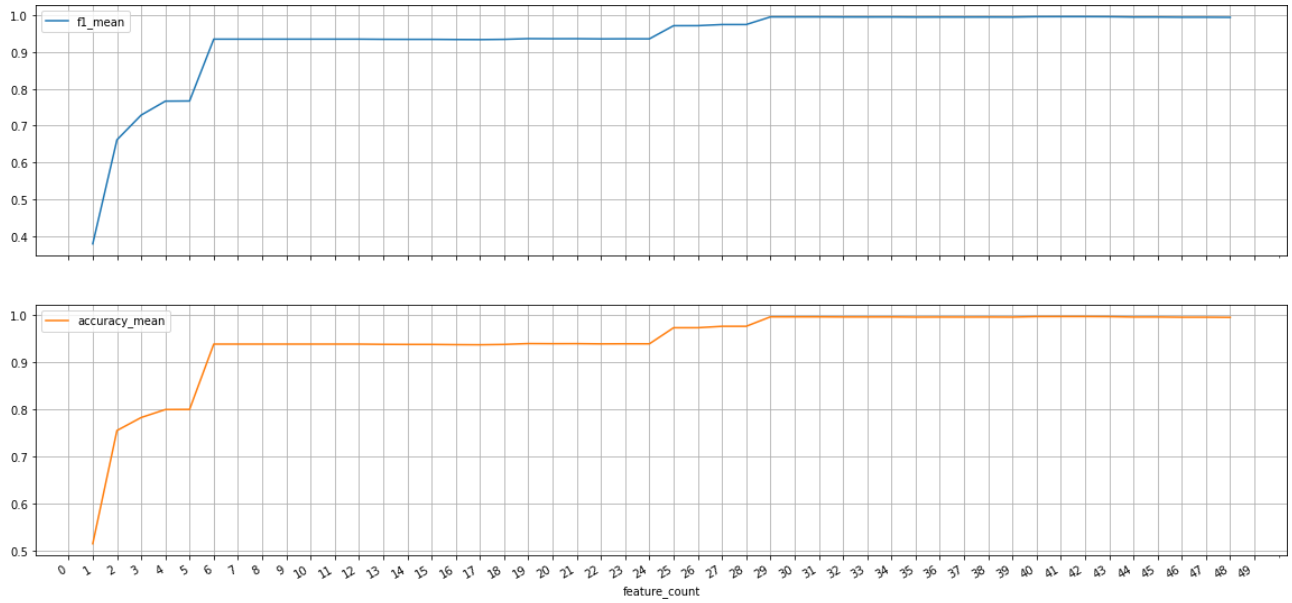

Figure 4: Comparison of SelectKBest performance

Table 7: Comparison of reduced feature performances (SelectKBest)

K (Feature Count)	6	25	29	42
External Features	2	5	7	12
F1	0.9354	0.972	0.9959	0.9966
Feature Names	['check https', 'check og', 'check viewport', 'robots_txt', 'source news', 'source not optimized']	['position', 'speed', 'check canonical', 'check description', 'check external links', 'check https', 'check internal links', 'check kw_count', 'check kw_in_description-og-property', 'check kw_in_meta-description', 'check kw_in_meta-og', 'check kw_in_title', 'check kw_in_title-og', 'check nofollow', 'check og', 'check title', 'check viewport', 'check word_count', 'robots_txt', 'source ads', 'source news', 'source not optimized', 'source top', 'tools analytics count', 'url length']	['position', 'speed', 'check canonical', 'check description', 'check external links', 'check https', 'check internal links', 'check kw_count', 'check kw_in_description-og-property', 'check kw_in_link-text', 'check kw_in_meta-description', 'check kw_in_meta-og', 'check kw_in_title', 'check kw_in_title-og', 'check nofollow', 'check og', 'check sitemap', 'check title', 'check viewport', 'check word_count', 'robots_txt', 'source ads', 'source known', 'source news', 'source not optimized', 'source top', 'tools analytics count', 'tools seo count', 'url length']	['position', 'speed', 'check canonical', 'check description', 'check external links', 'check h1', 'check https', 'check internal links', 'check kw_count', 'check kw_density', 'check kw_in_description-og-property', 'check kw_in_href', 'check kw_in_link-text', 'check kw_in_meta-content', 'check kw_in_meta-description', 'check kw_in_meta-og', 'check kw_in_source', 'check kw_in_title', 'check kw_in_title-meta', 'check kw_in_title-og', 'check kw_in_url', 'check nofollow', 'check og', 'check sitemap', 'check title', 'check viewport', 'check word_count', 'check wordpress', 'micros counter', 'robots_txt', 'source ads', 'source known', 'source news', 'source not optimized', 'source search engine', 'source shop', 'source top', 'tools ads count', 'tools analytics count', 'tools content count', 'tools seo count', 'url length']

2.7 Current status and next steps.

In summary, we have been able to develop good initial approaches for determining SEO on websites. We have already evaluated various datasets using the previous approach of rule-based classification to determine the probability of SEO in search results (see section 4). However, since this approach has some weaknesses, we are using machine learning to evaluate how we can achieve better data quality and more reliable classifications overall. In doing so, we also want to focus more on internal features in order to reduce the manual maintenance of lists or to weight them

differently, respectively. In addition, we are currently developing a method to output a statement about the degree of optimization in the form of a probabliy score. In future work, we also plan to use methods from unsupervised learning to determine further indicators. These are to be obtained directly from the source texts, since we have so far relied on typical indicators from the literature on search engine optimization as well as on an evaluation by experts.

3 Software Development

The software tool for analyzing search results according to the probability of search engine optimization is developed in Python. We chose Python as a programming language because it offers very good possibilities to perform data analysis and write lightweight web applications. Several external programming libraries support our goals of developing a performant application that allows us to scrape web content, analyze the scrapped data, and is also compatible with popular machine learning approaches. The goals of our software in detail are stated as follows:

- Scraping of search results from commercial search engines to identify indicators to measure the grade of search engine optimization of websites.
- Extracting and preparing the content of websites for further analysis processes.
- Evaluating the scraped webpages and extracted indicators to examine the probability of SEO by using a rule-based classificator and supervised machine learning based classification approaches.

The software is based on the prevalent model view control architecture to ensure stability and further development. This architecture divides the typical parts of a software by unlinking the graphical user interface (view layer), the data processing (control layer) and the data layer. Consequently, this means that the various parts are developed largely independently of each other, which means that modules can be replaced with little effort. Figure 5 shows the software architecture.

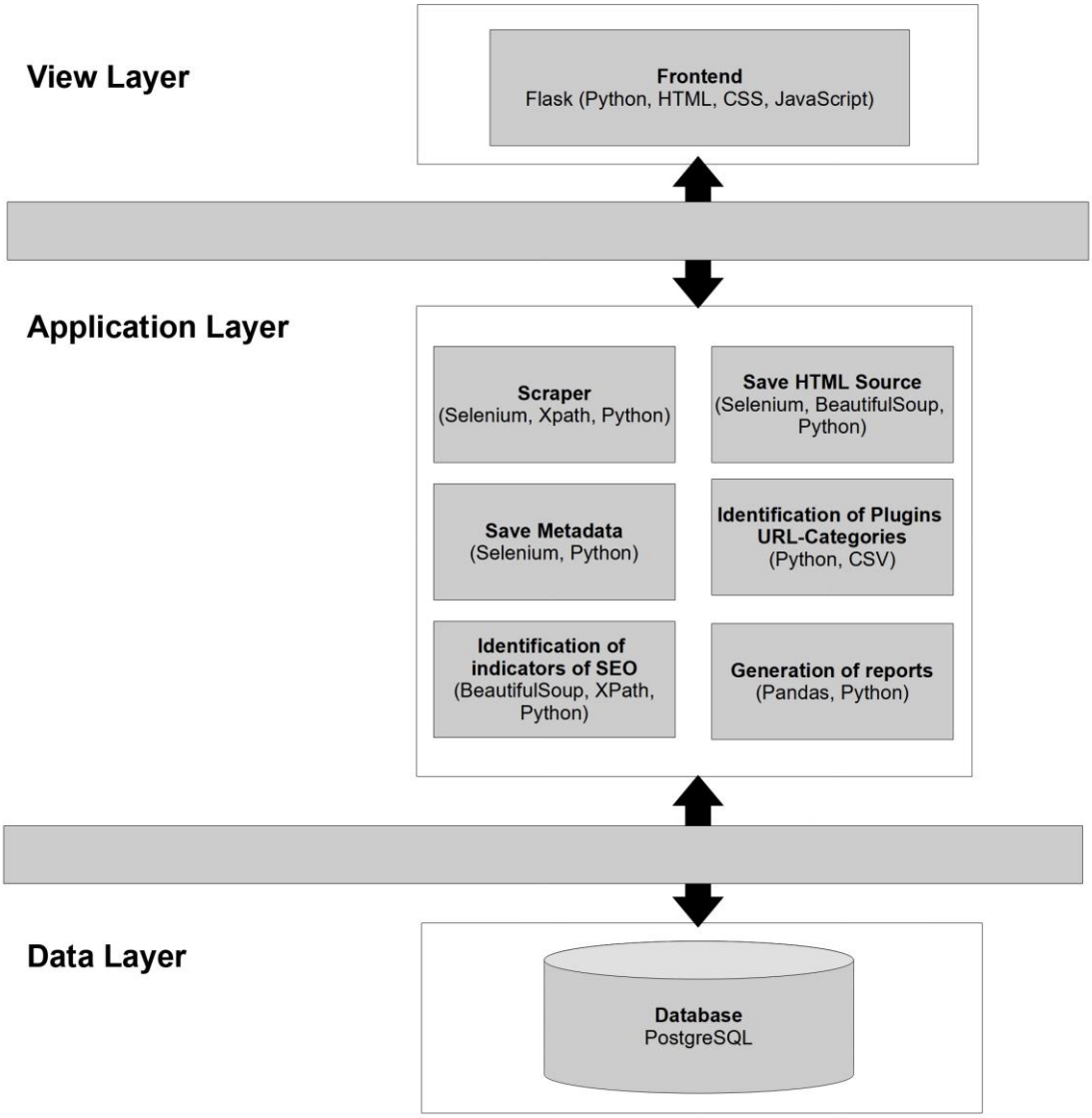

Figure 5: MVC-Architecture

Each layer consists of several modules for different tasks that can be replaced within the layers. The data layer is mainly the database of the application (see 3.2) which stores all data, e.g., search results and evaluation results to identify the probability of search engine optimization of the search results. PostgreSQL is the database management system for the application. It was chosen due to its capabilities to handle huge amounts of data and the licensing of that system[14]. The license allows to use, copy, modify for any purpose, without fee, and without a written agreement. The Control layer consists of several applications based on third-party libraries for designing tests, scraping search

[14] https://www.postgresql.org/about/licence/

results, and extracting the content of websites (see section 3.1.2). The layer is also the interface inbetween the database and the view. All data processing takes place here before it is stored in the database and it also makes the data available on the view layer. The view layer is based on the frontend framework Flask[15]) because of its good integration possibilities on Python. It is responsible for taking the user inputs to create studies, to administrate search queries and to configure via a graphical user interface. The view communicates directly with the control layer to exchange data. The following sections describe the software implementation in detail. Some snippets of source code will be used to explain our development. The whole source code is available in the project's OSF repository[16].

3.1 Modules and scripts

Our software framework is based on several self-developed modules and scripts for performing studies with search engine results. We followed the MVC architecture and developed modules for each layer. We have structured these modules according to their tasks, as shown in Table 8.

Table 8: Modules and scripts

Module	Description
App	Application to scrape search results and to identify and store indicators for SEO.
Main	Main script to start the background scripts for parallel processing using a job management.
Indicators	Application to identify SEO-Indicators in source codes of webpages.
Classifier	Rule-based classifier to determine the probability of SEO on a webpage.
Results	Application to export results from the database to save as CSV file for further processing.
Scraper	Application to scrape search engines and to store source codes in the database.
DB	
Connect	Script to connect the app to the database.
Evaluations	Class to read and write to the evaluations table.
Queries	Class to read and write to the queries table.
Results	Class to read and write to the results table.
Scraper	Class to read and write to the scraper table.
Sources	Class to read and write to the sources table.
Studies	Class to read and write to the studies table.
Libs	
Helpers	Class with self-developed functions to enable wildcard search patterns and to

[15] https://flask.palletsprojects.com/

[16] https://osf.io/amfvj/

	escape HTML special characters.
Evaluations	Class to communicate with interrelated DB class.
Queries	Class to communicate with interrelated DB class.
Results	Class to communicate with interrelated DB class and to store the source code of search results.
Scrapers	Class to communicate with interrelated DB class and to scrape search engines.
Sources	Class to communicate with interrelated DB class.
Studies	Class to communicate with interrelated DB class.
Demo	
App	Web application developed in Flask to demonstrate our rule-based approach to identify the probability of SEO on a single webpage.
Libs	Script to identify indicators of SEO on a single webpage and to classify the page.
Templates	HTML templates for the web application.
Gui	
App	Web application developed in Flask to manage studies and search queries.
Templates	HTML templates for the web application.
Tests	
Scraper	Test application to check for the functionality of the web scrapers.

3.1.1 Communication of application to database

The software framework follows a pattern that ensures a free choice of the database management system to make the software as compatible as possible. Currently, it supports the use of all popular DBMS like MySQL, PostgreSQL and MariaDB. Our pattern is to separate the database operations and a layer of applications that perform the database processes. The following code snippet show this approach:

/db/queries.py

```
#database class for search queries
class Queries:
  def __init__(self, cursor): #constructor
    self.cursor = cursor #database cursor

  #function to write a single query to the queries table
  def insertQuery(cursor, studies_id, queries_query):
    cursor.execute("INSERT INTO queries VALUES(%s, %s);", (studies_id, queries_query,))
```

/libs/queries.py

```
#application class for search queries
```

```
#import database class with connector based of pycopg2
from db.connect import DB

#import queries class to communicate with the database and to store data in the queries table
from db.queries import Queries as DB_Queries

#function to write query to table
  def insertQuery(study_id, query):
    db = DB() #initialize database connection
    #call function from database class
    DB_Queries.insertQuery(db.cursor, studies_id, queries_id) #call database class to
   insert query
    db.DBDisconnect() #disconnect from database
```

As can be seen from this snippet, the program calls the necessary functions to write a search query to the database. The advantage of this approach is that the DBMS and the SQL query are interchangeable as long as we call the specific connector for it.

3.1.2 Scraper

Since we are studying the probability of search engine optimization in search results, we need a scraper that is capable of automatically querying commercial search engines like Google and storing the results. For scraping, we use the query of search engines based on GET parameters to process our search queries and the desired result positions. The typical search query URL from Google with the search query and position looks like this:

*https://www.google.de/search?**q**=QUERY&**start**=POS*

In this URL, both the localization (here Google Germany, **google.de**) and the search query with the parameter **q** and the start position **start** are specified. If we now query Google, we replace q with a search query from our database and set a start value that

begins at 0. We generate a total of 32 scraper jobs per query, since we want to capture up to 320 search results. We chose this approach because our preliminary studies showed that Google typically never returns more than a total of 320 search results per query. The search URL is generated dynamically in each case. The components of this URL are stored in a configuration file. This file also stores where the search results can be found in the source code of a Google search results page. For scraping, we use the XML-based query language XPath, with which a DOM of a web page can be systematically queried. With XPath, XML-typical expressions are queried, which in the case of web pages refer to specific HTML elements and formatting. So, the URLs of the search results at Google.de can be read out over the parameter "//div[@class='tF2Cxc']/div[@class='yuRUbf']/a/@href". One of the main problems with scraping commercial search engines is that they can block an automated query after a certain number of queries in a certain period of time. They then respond with a prompt for a captcha. Currently, our software acts to automatically detect if Google rejects the request. To do this, it checks the displayed search results for a hint to display the captcha. If this is the case, the scraping job is set to unsuccessful. Via further scripts these unsuccessful requests are reset and continued at a later time. Currently a solution is also being worked on to allow the use of proxy servers and VPM services to prevent Google from blocking our scraping requests.

The technical implementation of scraping and saving the source code of search results is done by using the Selenium webdriver and the HTML library Beautifulsoup. This combination allows us to save web pages as they are displayed in a web browser, i.e., with capturing dynamically generated content and executing Javascript. Other scraping solutions such as cURL are also conceivable. However, these usually have the problem that they do not display dynamic content and the execution of scripts is also not possible. Selenium is a cross-browser test suite for websites and therefore provides an interface to various web browsers. It practically simulates users of web browsers and allows to simulate interactions. It is able to call web pages as they are displayed in a browser. Therefore, we decided to use these capabilities to be able to store the actual source code of search results. This source code is then also the basis for capturing indicators that can point to SEO. For the preparation of the saved source code, we use Beautifulsoup, as this simplifies processing. For example, we can remove unnecessary spaces or simply detect all comments in the text.

We developed a job management system that monitors and performs the scraping tasks. In the first step, scraping jobs are defined automatically when search queries are stored in the system. These jobs are processed in a queue. For each job, a check is made to see if search results are returned for the query by the search engine. If this is the case, these results are stored with their position. If no search results are found for the query, the system checks whether either a prompt for entering a captcha code is displayed or whether the maximum number of search hits has already been reached. Since this hint is also displayed on the search results page, it can be easily checked. The job management is able to assign a status to each job, which are defined as follows:

- 0 = scraping job is unassigned
- 2 = scraping job is in process
- 1 = scraping job is finished successfully
- -1 = scraping job is finished with an error

The status of the job changes during the process. Thus, a job starts as unassigned. When it is assigned, the system checks whether it finished successfully or not. All jobs that ended with an error are reset to status 0 to allow the job to be executed again. The system tries to perform these jobs again after 30 minutes.

3.1.3 Save HTML Source

We determine a large number of the indicators directly from the source code of a web page. To do this, we evaluate the HTML code with the help of various tools (see section 3.1.5). In order to save the source code in its original form, we have installed the Selenium WebDriver[17] for our software tool. With Selenium, it is possible to write automated tests for web pages that can simulate user interactions as if an actual web browser were being used. This makes it possible to capture all dynamic content that cannot be captured by other scraping libraries such as cURL[18]. Selenium offers a variety of interfaces for all major browsers (Firefox, Safari, Edge, Chrome, Internet Explorer). For our scraper we use Mozilla Firefox and use the profile function to use extensions that are necessary to capture in particular the source code of web pages that are difficult to retrieve automatically due to pop ups asking for privacy settings[19]. With the help of

[17] https://www.selenium.dev/projects/

[18] https://curl.se/

[19] https://www.i-dont-care-about-cookies.eu/

this extension, we simulate the confirmation of all cookies and then reload the source code after a time delay. The following code snippet shows our approach to save the source code of a web page.

/libs/results.py

```
#function so save html source
def saveResult(url):
  #set options for Selenium and Mozilla Firefox
  os.environ['MOZ_HEADLESS'] = '0'  #start in headless mode
  options.add_argument("user-data-dir=selenium")  #path to store cookies
  options.log.level = 'error' #log level

  #set profile for Firefox with security options to prevent loading malicious code
  profile = webdriver.FirefoxProfile()
  profile.set_preference("browser.safebrowsing.blockedURIs.enabled", True)
  profile.set_preference("browser.safebrowsing.downloads.enabled", True)
  profile.set_preference("browser.safebrowsing.enabled", True)
  profile.set_preference("browser.safebrowsing.forbiddenURIs.enabled", True)
  profile.set_preference("browser.safebrowsing.malware.enabled", True)
  profile.set_preference("browser.safebrowsing.phishing.enabled", True)
  profile.set_preference("dom.webnotifications.enabled", False);

  #load extension to accept all cookies automatically
  # https://www.i-dont-care-about-cookies.eu/
  profile.add_extension(extension='/path /i_dont_care_abou t_cookies-3.2.7-an+fx.xpi')

  # initialize selenium driver
  driver = webdriver.Firefox(firefox_profile=profile, options=options)
  driver.set_page_load_timeout(60) #max of 60 second to load a page

  #execute selenium driver to scrape source code
```

```
 try:
   driver.get(url) #load url
   time.sleep(10) #wait 10 seconds to render source code; important for redirects
   source = driver.page_source #save source code
 except:
   source = false #set source = false on error

 driver.quit() #close selenium driver

 return source #return source for further processing
```

3.1.4 Save Metadata

Some of the indicators we evaluate are not extracted from the html code. For example, we use the url to check if the providers use https and we measure the page loading speed using a simple script that we run with the help of Selenium. It calculates the seconds needed to load the entire dom of a given web page. We are aware of factors such as mobile device and dependencies on the speed of the server running the software. As a result, we currently only achieve a rough value when determining the loading speed. In future development, however, we will develop agents that simulate both the device used and the location of the user. In addition, we will take several measurements of loading speed and use average values for evaluation as SEO indicators. At the moment, however, a rough guideline value is sufficient, as we use many indicators for rule-based classification.

/apps/sources/pagespeed.py

```
#function to calculate loading speed of an url
def pagespeed(url)
   speed = driver.execute_script( """ var loadTime =
   ((window.performance.timing.domComplete-
   window.performance.timing.navigationStart)/1000); return loadTime; """ )
   driver.quit()
```

```
return speed
```

3.1.5 Identification of indicators of SEO, plugins and URL-categories

The application to gather the indicators consists of partial applications and scripts. This makes the application modular and expandable so that further characteristics can be recorded in the future. This is to ensure that with newly added indicators, an extension of the software can also take place. The main application is a script that first reads the meta information and the source code of the stored search results. Thereby the source text is read in different forms for the further processes. These include the raw source code, a text without HTML markups, and the page as a DOM tree for easier querying with XPath. Furthermore, the search queries for the results are also read in to perform various processes related to the search words. Once all the data is provided, the indicators are acquired one by one. This is done, as mentioned earlier, by executing scripts developed for each indicator or group of indicators.

The scripts are divided into the following categories:

- Checking whether particular content can be found on the web pages, e.g., checking whether a description is used or SEO plugins are used. Most scripts fall into this category.
- Scripts for checking indicators related to keywords, e.g., for calculating keyword density or the occurrence of keywords in the title of the page.
- Scripts on the basis of the URL, so simple means are used to check whether https is used or whether keywords are present in the URL.
- A script is the check of SEO measures in robots.txt.

All scripts are available in the folder apps/indicators[20]. The indicators.py is the main application to execute all scripts.

The following scripts are implemented so far:

Table 9: Scripts to extract indicators

Script	Description
canonical.py	Script for checking the number of canonical links in the source code.
description.py	Script for checking the usage of a site description in the source code. It evaluates if a description could be found in meta tags by the name property or by the open graph tag for site descriptions.

[20] https://osf.io/amfvj/

h1.py	Script for checking the use of h1 – headings in source code.
https.py	Script for checking if https is part of the URL.
indicators.py	Main controller for the extraction of indicators for SEO. Firstly, it reads all unprocessed web pages, their metadata and their stored sources from the database. Secondly, the script executes all individual scripts for the indicators. The controller is modular as it allows adding scripts for storing new indicators easily.
keyword_density.py	Script for calculating the keyword density of a text to a given search query.
keywords.py	Script for running several processes related to the keyword of a search query and the source code. It checks the presence of the keywords in the document page, meta properties, link texts and links, and source code. It uses a config file with XPath-Experessions.
kw_in_url.py	Scrtipt for checking the presence of the keyword in the url of the document.
links.py	Script for counting the number of internal and external links.
micro.py	Script for checking the presence of microdata forms in the source code. It uses a file with names and search patterns of microdata formats.
nofollow.py	Script to count the numner of nofollow links in source code.
og.py	Script to check the presence of open graph tags.
plugins.py	Main script for checking the presence of SEO plugins and analytics tools using csv files with names of the plugins and search patterns to find them in the source code.
robots.py	Script to check the presence of SEO techniques in a robots.txt file. It uses the main domain and scrapes the associated robots.txt. The script also writes the result of the evaluation to any result with the same domain.
sitemap.py	Script for checking the presence of a sitemap.
sources.py	Main script to check the categorization of a domain. The script reads the categorization files with the domains to check if the domain of the result could be found in one of these lists.
title_h1.py	Script to check if the title can also be found in the h1 headings of the document.
title.py	Script for checking the presence of a site title. It evaluates if a title could be found in meta tags by the name property or by the open graph tag for site descriptions.
url_length.py	Script to count the length of a url without the shema (https://www.).
viewport.py	Script to check the presence of a viewport tag.
wordpress.py	Scrtipt to check if the document is generated in wordpress.

The following snippet shows the script to check the number of canonical links. We use an xpath expression to extract all canonical links from the dom:

/apps/indicators/canonical.py

```
#function to evaluate the number of canonical links

#hash = hash of url

#tree = dom of web page


def canonical(hash, tree):
```

```
xpath = = '//a[@rel="canonical"] | //link[@rel="canonical"]'
module = 'check canonical'
number_of_links = len(tree.xpath(xpath))

#function to write the result to the database
insert_evaluation_result(hash, module, number_of_links)
```

3.1.6 Rule-based classification

Search engine optimization probability is currently calculated using a simple rule-based classifier (see 2.5). For this, the indicators for a web page are read from the database and stored in variables that are processed using else if statements. The following excerpt from the source code shows the procedure for this.

```
/apps/classifier/classifier.py
'''
globals:
   classification_result: result of classification (str)
'''
classification_result = "uncertain"

#check if document is most_probably_not_optimized
if source_not_optimized == 1:
   classification_result = "not optimized"
   not_optimized = 1

#check if document is most probably optimized
if classification_result == "not optimized" and (tools_seo > 0 or source_known == 1 or source_news == 1 or source_ads == 1 or indicator_micros > 0):
   classification_result = "optimized"
   optimized = 1
```

```
#check if document is probably optimized
if optimized == 0 and not_optimized == 0 and (tools_analytics > 0 or source_shop == 1 or source_company == 1 or indicator_https == 1 or indicator_og == 1 or indicator_viewport == 1 or indicator_robots_txt == 1 or indicator_sitemap == 1 or indicator_nofollow > 0 or indicator_canonical > 0 or (indicator_speed < 3 and indicator_speed > 0)):
  classification_result = “probably_optimized“


#scrape two sub pages from url, if document is still not classified and check for titles on the pages. If identical titles found, set indicator_identical_title = 1
if classification_result == “uncertain“:
  indicator_identical_title = check_identical_title(hash)


#check if document is probably_not_optimized
if optimized == 0 and not_optimized == 0:
  if indicator_title == 0 or indicator_description == 0 or indicator_identical_title == 1:
    probably_not_optimized = 1
    classification_result = 'probably_not_optimized'


return classification_result
```

3.1.7 Generation of reports

Currently, the software is able to provide a CSV file with the results for further processing. The results from all studies are exported from the database with the evaluations and the rule-based classification. Since this can involve very large amounts of data, the SQL queries were designed in a way that they are split according to an offset value and the intermediate results for each query are stored in temporary files. This is to prevent the SQL server from returning a timeout. The offset value is defined based on the amount of data in the associated .ini file along with the file name for storing the

data. The app to export the results is available at /apps/results. An example for a results for out test cases (see chapter 4) is available at OSF[21].

3.1.8 Running the software on servers

The main application of the tool is a controller that controls all tasks for recording and evaluating the indicators via time-controlled processes. The processes are called up as threads that run in parallel. This architecture makes it possible to install the application on multiple servers and write the results to the central PostgreSQL database on the database server. We have chosen this architecture to allow especially computationally intensive processes, such as scraping and saving the web pages. All important tasks are divided into additional controllers, thus creating the possibility to run them as separate applications on servers. This way the load of the tasks can be better distributed. Figure 6 shows an example how to run the software by using distributed servers. Two servers run the scrapers to gather search results while two more servers run the classifier in parallel.

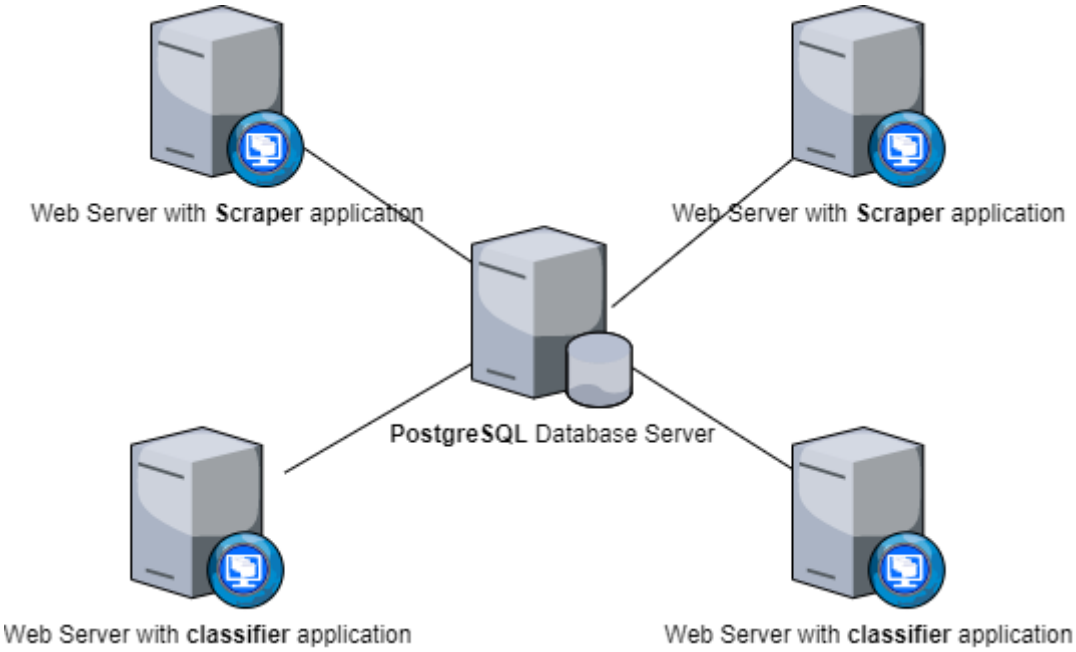

Figure 6: Installation of the software on several servers

[21] https://osf.io/z43jb/

So far, the application has been prototyped and is still under development. For example, many processes currently depend on samples that are randomly created from the database based on SQL queries. The advantage with this method is that for complex queries with a lot of results, samples can be generated quickly for further processes. If sampling is not used, queries can take a very long time and this has a strong impact on application speed. In the future, however, the application should act more intelligently and be controlled on the basis of better job management for threads and processes.

3.2 Database Model

All data from the application is stored in a PostgreSQL database. This includes the metadata on the studies, search queries, search results, the source texts of the web pages as well as the identified indicators and the results of the rule-based classification to assess the probability of SEO on a web page. In the following, the current database model for the software is explained in more detail.

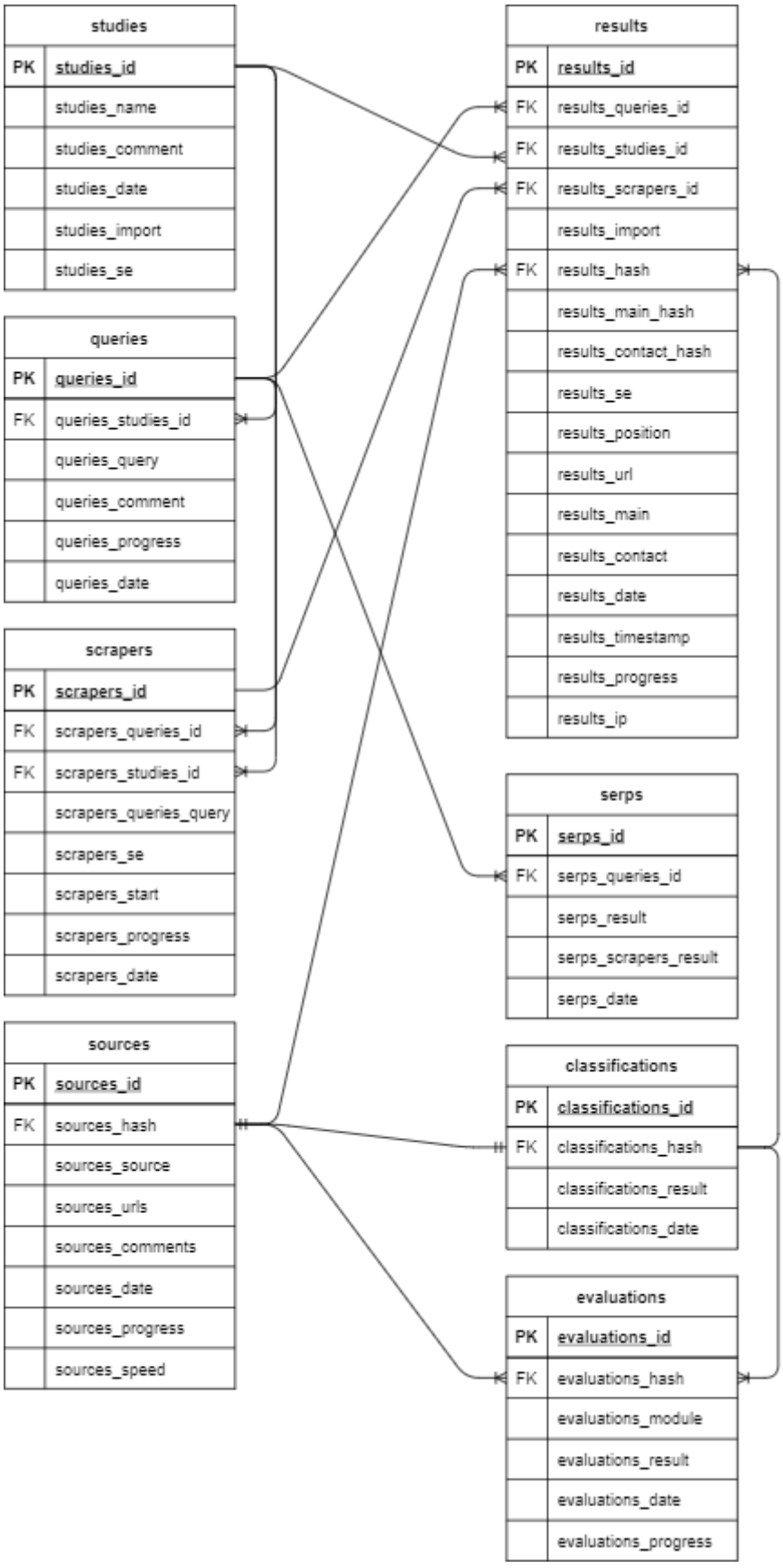

Figure 7: Database Model

Our current database model (Fig. 5) consists of a total of eight tables. These are based on the requirements for our software. For example, many tables contain an MD5 hash that is generated based on the URL of web pages and serves as a link. For easier data collection and to ensure good performance, normalization has been omitted in some tables, as some queries with multiple joins could create severe degradation. In the following, each table is briefly described with its columns.

Table 10: Tables and columns

Table / *Column*	Description	Type
studies	Studies are the starting point for the software. Researchers define their test cases here.	
studies_id	Primary key of studies table	int
studies_name	Name of the study	text
studies_comment	Description of the study	text
studies_date	The date when study was created	date
studies_import	Option, if study uses results from a list	int
studies_se	List of scrapers for the study separated by comma, e.g., Google, Bing	text
queries	Table with search queries for a study	
queries_id	Primary key of queries table	int
queries_studies_id	Foreign key to studies table	int
queries_query	Keywords of search query	text
queries_comment	Description of search query	text
queries_progress	Status of query, whether if it was processed or not	int
queries_date	The date when query was created	date
scrapers	Table with scrapers job. The scrapers job will be created by the scrapers app (see 3.1.2)	
scrapers_id	Primary key of scrapers table	int
scrapers_queries_id	Foreign key to queries table	int
scrapers_studies_id	Foreign key to studies table	int
scrapers_queries_query	Keywords of search query	text
scrapers_se	Search engine to scrape	text
scrapers_start	Start value for a scraper as GET_Parameter for the scraper jobs.	int
scrapers_date	Date of scrapers job	date
scrapers_progress	Progress of scrapers job (0 = not assigned, 1 = finished, 2 = is processed, -1 = error)	int
sources	Table with stored sources of search results	
sources_id	Primary key of sources table	int
sources_hash	md5 hash of result url	text
sources_source	HTML source code of the web page	text
sources_urls	List of extracted URLs of the web page	text
sources_comments	List of HTML comments from the source code	text
sources_date	Date when source was saved	date
sources_progress	Progress of source storing (0 = not processed, 1 = finished, 2 = is processed)	int
sources_speed	Calculated loading speed of the web page (see 3.1.4)	int
results	Table with search results	

results_id	Primary key of results table	int
results_queries_id	Foreign key to queries table	int
results_studies_id	Foreign key to studies table	int
results_scrapers_id	Foreign key to scrapers table	int
results_import	Indicator if result was imported from a list	int
results_ip	IP-Adress of result	text
results_hash	md5 hash of result url	text
results_main_hash	md5 hash of top domain	text
results_contact_hash	md5 hash of page with contact data	text
results_se	Search engine from which the result was scraped	text
results_position	Results position in search engine	int
results_url	URL of result	text
results_main	URL of top domain of result	text
results_contact	URL of page with contact data	text
results_date	Date when result was scraped	date
results_timestamp	Timestamd when result was scraped	timestamp
serps	Table with the source code of SERPs and also result of scraping content from a SERP	
serps_id	Primary key of serps table	int
serps_queries_id	Foreign key to queries table	int
serps_result	Source code of SERP	text
serps_scrapers_result	Result of scraping to check if some special content was found on the SERP	int
serps_date	Date when SERP was scraped	date
classifications	Table with classifications results of the rule-based classification	
classifications_id	Primary key of classifications table	int
classifications_hash	md5 hash of result url	text
classifications_result	Determined class	text
classifications_date	Date when classification was processed	date
evaluations	Table with identified values for indicators for SEO (see 2.2)	
evaluations_id	Primary key for evaluations table	int
evaluations_hash	md5 hash of result url	text
evaluations_module	Indicator	text
evaluations_result	Value of indicator	text
evaluations_date	Date when indicator was stored	date
evaluations_progress	Progress of indicator determination	int

3.3 Installation and Dependencies

The tool is developed in Python3. A backward compatibility to Python2 is not given. In the following a short step-by-step guide will show the installation process, so that the tool can be used on any linux web server for example Debian or Ubuntu. For the future, the definition of an installation package is also planned, which should simplify the installation. At the moment, however, only manual installation is possible. Some steps can be skipped, if the requirements are already met. For most installation steps a

super user is needed. So, it is necessary to define a super user first. In the following we give a short manual to install the software using pip and apt to install external packages.

1. Download the source code

The soure code is available at: https://osf.io/amfvj/. The zip file has to be extracted and to be copied to any folder.

2. Install Python3 virtual environments

We recommend to install a virtual environment to install the software and the necessary packages. To install such environment it is necessary to install the package for virtual einvironments first via **sudo apt-get install -y python3-venv**. Next we create the virtual environment with **sudo python3 -m venv seoeffekt**. It is recommened to install the virtual environment for the standard user and not the root.

3. Installation and configuration of the PostgreSQL database

All data of the software will be stored in PostgreSQL database. To use PostgreSQL as DBMS (database management system) we install its client in the virtual environment via **sudo apt-get install -y postgresql libpq-dev postgresql-client postgresql-client-common**. After the installation we create the postgresql user and the database for the application. The sql file “install_db.sql” for the datables can be found at the folder install in the folder with the source code. This example shows how to do that with a user called seo and a databsse called seoeffekt:

1. **sudo -i -u postgres**
2. **createuser seo -P –interactive**
3. **createdb seoeffekt**
4. **psql seoeffekt**
5. **psql -f /installation/install_db.sql**
6. **exit.**

The standard configuration of the database makes it just accessible from localhost. It is fine to leave it that way, if the application will be just used on one local machine. Since our tool is developed with using the data base as central storage for all data, we suggest do make it accessible from external machines. We will change the configuration file of our installed PostgreSQL server, to make that possible. It is necessary to change

to the root user to get access to the configuration file. This example shows how to change the file, using nano as editor:

1. **su root**
2. **find / -name "postgresql.conf"**
3. **nano /etc/postgresql/11/main/postgresql.conf**
4. replace line listen_addresses = 'localhost' with listen_addresses = '*'
5. **find / -name "pg_hba.conf"**
6. add following entries at the very end:
   a. host all all 0.0.0.0/0 md5
   b. host all all ::/0 md5
7. **etc/init.d/postgresql restart**

4. Installation and configuration of the packages for the scraper

The software uses Selenium WebDriver along with Mozilla Firefox and the geckodriver as API to communicate with Firefox to scrape web pages. The latest version of the geckodriver is available at https://github.com/mozilla/geckodriver/. We will also install the latest version of Firefox using the unstable repository of Debian. The installation varies for other linux distributions. The following example shows how we installed the latest version of the packages on a Debian server:

1. **sudo wget https://github.com/mozilla/geckodriver/releases/download/v0.29.0/geckodriver-v0.29.0-linux64.tar.gz**
2. **sudo sh -c 'tar -x geckodriver -zf geckodriver-v0.29.0-linux64.tar.gz -O > /usr/bin/geckodriver'**
3. **sudo chmod +x /usr/bin/geckodriver**
4. **sudo rm geckodriver-v0.29.0-linux64.tar.gz**
5. **sudo nano /etc/apt/sources.list**
6. **add the following line:**

   deb http://deb.debian.org/debian/ unstable main contrib non-free

7. **sudo nano /etc/apt/preferences.d/99pin-unstable**
8. **add the following lines:**

Package: *

Pin: release a=stable

Pin-Priority: 900

Package: *

Pin release a=unstable

Pin-Priority: 10

9. **a sudo apt update**
10. **sudo apt install -y -t unstable firefox**
11. **sudo apt-get install -y curl unzip xvfb libxi6 libgconf-2-4**
12. **sudo apt-get install -y default-jdk**
13. **sudo apt-get -y install gnupg2**
14. **su root**
15. **curl -sS -o - https://dl-ssl.google.com/linux/linux_signing_key.pub | apt-key add -**
16. **apt-get -y update**
17. **apt-get -y upgrade**

5. Installation of the packages for the virtual environment:

The last steps will show how to install the dependencies for the software via pip in the virtual environment using a standard user. The packages include Psycopg as connector to the database, Beautifulsoup for processing scraped data e.g. source code, the Selenium API for Python, Flask as framework for the web application and Pandas for data analysis. First of all we need to activate the virual environment via **source seoeffekt/bin/activate** and also install pip there **sudo apt-get install -y python3-pip**. Next we will install the packages:

1. **pip3 install wheel**
2. **pip3 install selenium**
3. **pip3 install lxml**
4. **pip3 install pandas**

5. **pip3 install beautifulsoup4**
6. **pip3 install apscheduler**
7. **pip3 install psycopg2-binary**
8. **pip3 install request**
9. **pip3 install flask**
10. **pip3 install -U Flask-WTF**
11. **pip3 install python-dotenv**
12. **pip3 install gunicorn flask**

After the installation of all packages, we can deactivate the virtual environment via the command **deactivate**.

6. Configuration of the software:

The connection to the database will be established through Psycopg using a JSON configuration file with the information to the db name, db user, host address and the password for the db user. The file is located at "apps/config/db.ini":

```
{
        "_comment": "Database Connection; change the configuration here to
use a shared database or run it on a localhost",
        "dbname": "seoeffekt",
        "user": "seo",
        "host": "adress oft the database server or localhost",
        "password": "your password"
}
```

It is also necessary to change the file path to the configuration folder at "apps/db/connect":

```
config_path = '/home/your_user_name/your_software_folder/config/'
```

3.4 Demo Tool

The SEO Tool is a web application that allows determining the probability of search engine optimization on a given URL. Users can enter the URL to trigger the analysis by processing all steps from our proposed approach. The application is developed in Python using Flask (micro web framework written in Python), Selenium (portable framework for

testing web applications), Beautiful Soup (package for parsing HTML and XML), and pandas (library for data manipulation and analysis).

Figure 8 is an example of a generated result report. The report is categorized according to the processes of our approach and lists all characteristics:

- **SEO Assessment**: Probability of SEO on the given webpage
- **Tools & Plugins:** Use of SEO Tools and Analytics Tools
- **URL Category:** Categorization of the URL
- **Indicators for SEO:** Overview of Indicators for SEO

The checkboxes next to the identified characteristics show the result of our analyses, while the third column displays the extracted information or provides further explanations. The tool also shows deeper explanations of all processes, which can be opened by clicking on a category. Finally, it also offers the possibility to download the report as csv-file for further processing. The demo tool is available at http://5.189.155.20:5000/

Figure 8: Demo tool

3.5 Tutorial to use the software on a server

At the current state the software is just useable by adding information to the database manually, e.g. studies and search queries. If a study and search queries added, the software can be started via running the main script „main.py“ in the apps folder at „apps/main/“.

3.6 Software as Web application

The development of a web application with a graphical user interface is currently in progress.

4 Test cases

In this section, we present some results from our test studies for the software. The aim of these studies was on the one hand to continuously check the functionality of our developed components and on the other hand to gain initial insights into the use of search engine optimization in search results for various topics. The results shown here are exemplary and reflect a current state, as the software is continuously running to evaluate further search results. However, the results presented already give a first good insight into the importance and spread of search engine optimization as a tool for online marketing. For the application of our methods, three very different datasets were defined, which differ greatly from each other thematically. For example, the most popular search queries on Google were collected by designating them in Google Trends over a defined period of time to generate a diverse collection of search results that are not thematically fixed. The other data sets, on the other hand, are topic-specific. For example, cooperation with the Medienanstalt Hamburg Schleswig-Holstein (Media Authority Hamburg Schleswig-Holstein) enabled search queries to be generated to find potentially criminal documents relating to radical right-wing content. The third dataset was generated using German search queries relating to the Corona pandemic.

4.1 Datasets

To test our system, we used three datasets, which are based on query sets and results collected from Google through screen scraping until September 2020:

- **Google Trends**: This dataset consists of 1,478 queries derived from the Google Trends website, where recently trending queries are listed (https://trends.google.de/trends/?geo=DE). Results for these queries were collected on the day when the query occurred on the list. Data were collected from March to June 2020. The dataset consists of 244,985 results. It became clear that the search queries generally refer to similarly popular topics again and again across all months. These include, amongst others, search queries about Corona, celebrities, TV shows, and sports. This dataset was created to generate a larges and diverse set of search results.

- **Radical right**: This dataset consists of 82 queries provided by the Medienanstalt Hamburg/Schleswig-Holstein (regional media regulation authority). The queries are specific keywords that hint at radical right content. Data were collected in March 2020. This dataset consists of 13,403 results.
- **Coronavirus**: For this dataset, we used 483 top queries from Germany from the Microsoft Bing Coronavirus QuerySet (https://github.com/microsoft/BingCoronavirusQuerySet). Data were collected in September 2020. This dataset consists of 5,402 results.

With various topics and a total of 263,790 result URLs without duplicates, we are confident to have built a large and diverse enough dataset for testing purposes. In the following, we will present and discuss the results for the datasets separately to show if and how results depend on particular datasets and what similarities between the datasets can be found.

4.2 Results

An analysis of the data sets shows that the vast majority of the results are either highly likely or at least likely optimized. Figure 11 shows the respective results. Depending on the dataset, we can see that between 41.9% and 62% of the results found are classified as most likely optimized.

The differences between the datasets can be attributed to the higher proportion of news content in the Trends and Corona datasets compared to the radical right dataset (see table 11). For example, 56% of the results in the Google Trends dataset were news listings (136,806 documents), 45% in the Corona dataset (2,454 documents), and only 34% in the potentially radical right dataset. Overall, 47,177 unique domains were detected across all datasets for 263,790 documents. An evaluation of the top 10 shows that offers from Google in particular can be found very frequently in the documents (books.google.de and youtube.com). Wikipedia is in second place. All other offers are news offers.

Table 11: Distribution of news sources

Domain	Number of documents	Relatively number of documents in %
books.google.de	5.724	12,13
de.wikipedia.org	2.442	5,18
youtube.com	2.122	4,50
t-online.de	2.075	4,40

focus.de	1.915	4,06
welt.de	1.814	3,85
spiegel.de	1.713	3,63
sueddeutsche.de	1.699	3,60
stern.de	1.622	3,44
rtl.de	1.535	3,25

A further analysis in terms of SEO plugins and analytics tools used shows a similar distribution in all data sets. Table 12 shows the distribution.

Table 12: Distribution of SEO Plugins and Analytics Tools

Dataset	Number of documents with SEO-Plugins	Number of documents with Analytics Tools
Google Trends (244.985 documents)	8% (19.515 documents)	36% (87.175 documents)
Radical right (13.403 documents)	8% (1.206 documents)	36% (4.765 documents)
Coronavirus (5.402 documents)	6% (332 documents)	33% (1.793 documents)

It is clear that the number of SEO plugins is not very pronounced across all the data sets. This is due to the fact that these are usually used for websites that are based on the CMS Wordpress which is not usually used for professional news websites. However, an evaluation of the analytics tools shows that most of the websites perform web analytics for their offers, which indicates commercial intentions.

The evaluation of the technical features shows that 96% of all documents use HTTPS (252,519 documents) and Open Graph tags are even set in 98% (257,396 documents). This shows, for example, how high the relevance of social media is for website operators, as almost all providers define these special tags for an optimized display of a preview of their content.

Figure 9 shows the evaluation of the SEO probability in the data sets. It shows that only a small portion of the results were classified as most likely not optimized (1.6% across all datasets). These are all results from Wikipedia, as this is the only website on our list of definitely not optimized pages. A relatively small portion of pages (between 3.5% and 7%) are likely not optimized. For better presentation, we combined the results from most likely not optimized and probably not optimized. We also found a slight overlap of up to 3% across all datasets. The overlap can be explained by the fact that the results were classified by our classifier as both likely optimized and likely not optimized. In summary, we found that a large proportion of the results found in Google are either

definitely or probably optimized. Over 90% of the results found belong to these categories.

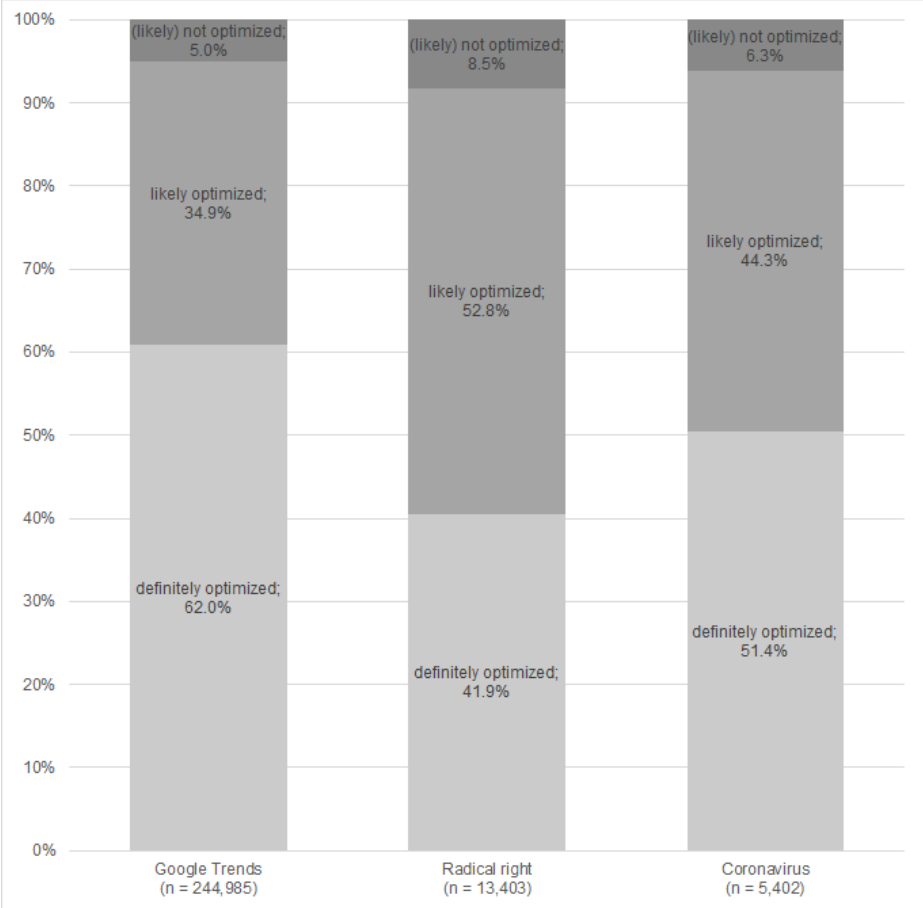

Figure 9: Distribution of results

4.3 Discussion

The first studies to determine the probability of search engine optimization show that it can be reliably assumed that optimized content strongly dominates the search results. Thus, only up to 10% are not or probably not optimized. This is no surprise, as we know that SEO is a multi-billion-dollar business and companies and other stakeholders often depend on the visibility their websites receive from traffic from web search engines. Thus, we can confirm the assumption that search engine optimization has a strong impact on search results, even independent of the topics being searched for.

The data sets differ strongly from each other in terms of content. While the results set on Google Trends covers a broad range of topics according to the popularity of search queries, the results on potentially radical right-wing content and on the Corona virus are very specific and less commercially oriented. However, a statement about whether the influence of search engine optimization has a positive or negative effect on the quality of results cannot be made with the analyses performed, whether with regard to individual search queries or the totality of search queries. For such analyses, the methodology would have to be supplemented, for example, by retrieval studies in which judges evaluate the quality of search results.

Furthermore, it is also necessary to further differentiate the current rule-based approach, since so far only a limited set of features is included in the analysis. In particular, the check for the use of SEO plugins, which is a very strong indicator of search engine optimization, and the check for missing descriptions (description tag), which is a strong indicator of a lack of search engine optimization, are particularly meaningful indicators. The other characteristics we have looked at so far are in part also well suited for assessing SEO, but they lack a clear weighting. A weighting of the indicators will help to create a more distinguished classification. Further characteristics, which for example relate to the keywords in the search queries or also so-called offsite factors such as the number of links to a website, have not yet been included in the model.

With regard to the categorization of websites into rather optimized and non-optimized sites, the identification of a site operator as a client of an SEO agency can be assumed as a safe indicator. In addition, a high search engine optimization can be assumed for a news offer. This is also shown by the closer evaluation of the frequencies in relation to the domains in the data sets. It can be assumed that particularly frequently occurring domains are also optimized, and since news offers in particular appear in the source distribution, it can be assumed that this type of website also uses SEO measures.

In the evaluation of the distribution of the most frequent sources in the data sets, prominent news providers were found in seven out of ten cases with a share of about 3% of the total of all search results. On the one hand, this can be attributed to the predominantly information-oriented search queries for all data sets. On the other hand, we explain this dominance of news services by the fact that search engines prefer reliable sources in the search results, for example, to minimize the risk of misinformation

and fake news (Wingfield, Isaac, and Benner 2016). However, this approach can disadvantage smaller information services and those that are independent of large media companies. Another disadvantage for content providers also exists in the fact that search engine companies' own offerings are listed very prominently in search results. For example, our analysis of the distribution of domains shows that Google's own offerings (Google Books and YouTube) are strongly represented. This kind of disadvantage has already led to legal disputes for several years (Edelman 2015).

The evaluation of all the above aspects shows how complex the influence of various factors on the display of documents in search results is. Thus, our evaluations already show that, in addition to the influence of search engine optimization on the results, aspects such as the diversity of search results and the search engine companies' own offerings are also relevant.

5 Conclusion and future work

The influence of search engine optimization on search results in commercial search engines has hardly been researched so far. With our semi-automated procedure and an implementation in a software tool, we were able to perform initial evaluations of how large the proportion of optimized content is in search results, regardless of topic focus. The approach includes proven approaches to automatically store search results, extract relevant features, and categorize URLs to perform page type by category, split between likely and not likely optimized pages. For these analyses, we have identified features that allow reliable conclusions about the probability of SEO on web pages. However, this model is not yet complete and will be extended by further features, factor analyses, weightings of features and feature groups as well as analyses with machine learning methods for unsupervised learning of the web pages (e.g., by clustering). Related to this, we will also take a closer look at the position of documents in the search engine rankings in order to investigate the correlations between SEO and the positions.

By conducting further studies, we will continue to develop our methodology and further investigate the effect of search engine optimization. In doing so, we will also look more closely at the diversity of sources in the search results and at the distribution of search engine companies' own offers, as it has already been shown that these factors also have an influence on the selection of documents for the search results and on the presentation of the search results.

6 Acknowledgements

The research project “SEO-Effekt” is funded by the German Research Foundation (DFG – Deutsche Forschungsgemeinschaft) from 5/2019 until 7/2021.

7 References

Edelman, Benjamin. 2015. “DOES GOOGLE LEVERAGE MARKET POWER THROUGH TYING AND BUNDLING?” *Journal of Competition Law and Economics* 11(2): 365–400. https://academic.oup.com/jcle/article-lookup/doi/10.1093/joclec/nhv016.

Enge, Eric, Stephan Spencer, and Jessie Stricchiola. 2015. *The Art of SEO: Mastering Search Engine Optimization*. Third edit. Sebastopol, CA: O’Reilly.

Erlhofer, Sebastian. 2019. *Suchmaschinen-Optimierung: Das Umfassende Handbuch*. Rheinwerk Verlag.

Goel, Sharad, Andrei Broder, Evgeniy Gabrilovich, and Bo Pang. 2010. “Anatomy of the Long Tail.” In *Proceedings of the Third ACM International Conference on Web Search and Data Mining - WSDM ’10*, eds. Brian D. Davison, Torsten Suel, Nick Craswell, and Bing Liu. New York, New York, USA: ACM Press, 201.

Petrescu, Philip. 2014. “Google Organic Click-Through Rates in 2014 - Moz.”

Schultheiß, Sebastian. 2021a. *A Representative Online Survey among German Search Engine Users with a Focuson Questions Regardingsearch Engine Optimization (SEO): A Study within TheSEO Effect Project*. https://osf.io/3ukcf/.

Schultheiß, Sebastian.. 2021b. *Expert Interviews with Stakeholder Groups in the Context of Commercial Search Engineswithin the SEO Effect Project*. https://osf.io/5aufr/.

Schultheiß, Sebastian, and Dirk Lewandowski. 2021. “‘Outside the Industry, Nobody Knows What We Do’ SEO as Seen by Search Engine Optimizers and Content Providers.” *Journal of Documentation*.

Wingfield, Nick, Mike Isaac, and Katie Benner. 2016. “Google and Facebook Take Aim at Fake News Sites.” *The New York Times*. https://www.nytimes.com/2016/11/15/technology/google-will-ban-websites-that-host-fake-news-from-using-its-ad-service.html.