Data Enrichment Pipeline Model for Web Classification Based on Web Scraping and Machine Learning

Evegeniya Samsonova *, Zlatan Morić *, Goran Gvozden *, **, Tomislav Hlupić *, **
* Algebra University, Zagreb, Croatia
** Poslovna inteligencija, Zagreb, Croatia
evgeniya.samsonova@algebra.hr, zlatan.moric@algebra.hr, goran.gvozden@inteligencija.com, tomislav.hlupic@inteligencija.com

***Abstract* — With the digitization of sales and marketing, growth of online platforms, and e-commerce, businesses are now able to operate globally with ease and fewer restrictions. Expanding customer reach also means handling larger data volumes, which is often addressed through automation. Many companies now utilize data pipelines and advanced AI systems for analytics, operational optimization, faster and improved decision-making. This paper will explain in details the creation of an automated lead enrichment pipeline for B2B that focuses on ease of implementation and deployment. Employing a web data extraction tool, machine learning for website classification, a cloud-based database, and an automation and orchestration tool, the software provides a readily implementable solution for smaller enterprises to build and deploy.**

Keywords - automation; data enrichment; machine learning; natural language processing;

I. INTRODUCTION

Digitization and increasing popularity of e-commerce platforms makes operating globally easier for many companies, but accessing a broader customer base comes with additional challenge of managing larger data volumes. As a result, many businesses now depend on automation, complex data pipelines, and AI for analytics and operations. While effective data utilization helps with strategic decision making, and offers a competitive edge and insights into market trends for business growth, disparities in AI and automation adoption persist across businesses and present challenges for some teams.

The 2022 IBM Global AI Adoption Index [1] revealed a significant gap between larger and smaller companies in AI implementation. Smaller teams encounter problems in the adoption of data tech stack. They are limited by financial and human resources, integration complexities and difficulty in balancing immediate needs with long-term goals in data stack implementation. Additionally, projects in smaller businesses are often impacted by the gap between the ever-advancing tools used by Big Tech, and the reality of executing such projects within smaller organizations and their infrastructure.

Moreover, the volumes of some businesses might not often justify the investment needed for complex and latest technologies [2]. Because of this, cost-effective, easier to manage and fast to deploy tools can be an optimal solution for enterprises seeking to integrate automation and AI into their operational workflows.

Focusing on technologies that do not require complex infrastructure, this paper describes the process of creating and deploying a data enrichment pipeline that uses automation and machine learning to extract, process and load necessary data to be used by a business for their needs. The result is a lead enrichment pipeline with components like CRM software, web data extraction tool, a hosted machine-learning model and a cloud-based database for data storage that can be used by an organization to enrich their internal data on leads to be used by the sales, marketing or other teams.

First, the process extraction of web data is discussed, along with some of the most common tools and techniques of collecting web data. The chapter covers the nuances of extracting textual data on the web, and the differences of processing traditional structured corpus and text on the web. Next, the work describes the process used to extract the textual and metadata features for the software. Additionally, model training, algorithm choice, experimentation and results are covered as well.

The paper follows with an overview of the pipeline automation, focusing on the utilized software, the architecture and underlying infrastructure. The study covers the process and outcomes associated with the implementation of such software, followed by limitations and suggestions for deployment.

II. EXTRACTION OF WEB DATA

Lead data enrichment enhances business client profiles by adding relevant details like industry, revenue, and technology stack, or other important information [3] [4]. This process aids in tailoring sales and marketing strategies, identifying high-potential prospects, improving market knowledge, and enhancing conversions and partnerships. It can also be important for a smaller organization that need to create a database of potential prospects at the start of operations. Often, enrichment process involves dedicated teams extracting data from reliable sources and updating databases, performing quality checks. More recently, data extraction and enrichment

processes are becoming automated as manual efforts are unable to cope with the amount of incoming data.

A. Web Data Extraction

Web scraping involves extracting data from the World Wide Web and saving it to a destination, like a database, for analysis or retrieval [5]. This process includes making HTTP requests to the data source, retrieving the resource in various formats (HTML, XML, JSON). After this, the retrieved data can be processed in an integration (ETL) or ingestion (ELT) process for business use.

While multiple solutions exist online, organizations might choose to depend on their internal teams for their web scraping needs. At the cost of a more complex infrastructure needs, handling the process internally allows for more precise customization of the scraping project for business goals. Additionally, internal teams are also more familiar with processes and objectives, and are able to tailor solutions to specific requirements and unique cases.

B. Libraries and Tools

Multiple tools exist for web scraping projects and parsing of web data [6]. Some of the most popular are urllib.request [7] library for HTTP requests, BeautifulSoup [8], a Python library known for its simplicity in parsing HTML and XML documents, Scrapy [9] for more advanced needs, Apache Nutch [10] for Big Data applications with complex architectures, and lxml [11] for parsing and manipulating structured HTML and XML data. While some may entail unnecessary overhead for smaller projects due to intricate setup and technical demands, simpler tools such as requests can, often at the cost of limited functionality, offer quicker setup and execution for smaller teams without the complexity of robust frameworks.

After comparing the functionalities of all, the choice was made to use using BeautifulSoup paired with the requests library. The choice does not only stem from the ease of implementation and user-friendliness of both tools, but also from practical reasons. In broad crawling, multiple websites are scraped instead of a crawler extracting data from a single website and all of its pages. As a result, the web data extraction tool navigates various websites with distinct structures and content, some of which may be poorly formatted. With the challenges of handling variable website structures and potential formatting issues, the robust parsing capabilities of BeautifulSoup proved to be essential, and the capacity to handle imperfect HTML made it an optimal parsing choice.

On the other hand, the requests library is optimal for most scraping projects due to its simplicity, flexibility, and ease of use. It provides a straightforward interface for making HTTP requests, handling cookies, and managing sessions, making it well-suited for most web scraping tasks. Additionally, it integrates seamlessly with other Python libraries. The lightweight nature of requests makes it efficient for basic scraping needs without unnecessary overhead.

The scraper, designed for broad crawling, was developed with custom multi-threading functionality to enhance web scraping speed by processing requests simultaneously, utilizing available CPU cores. The maximum number of threads was set at 30, dynamically adjusted based on the incoming data to prevent resource over-allocation.

C. Text Parsing

The text and metadata used for website classification were parsed and stored in a separate database during the initial request to reduce repetitive requests to the website. The contact page was preferred for contact information, and the home page served as a fallback for missing data. Contact page was detected using text parsing techniques in Python. If not detected, or a request fails, a home page was used.

There is a lot of text on a website, not limited to the text visible to a website visitor. The HTML structure, consisting of headings, paragraphs, and semantic tags, provides valuable insights into a business [12]. Navigation elements like menus and presence of shopping carts can provide further clues, for example whether or not a website uses e-commerce functionalities. Additionally, metadata often consist of information created for search engines and SEO purposes, and play a crucial role in page ranking. This can be utilized for a lot of cases, and extraction of web page content, specifically the title, metadata descriptions, and keywords, could yield meaningful features for classification. Metadata can also offer insights into languages, locale, and the intended audience or geographic region of a company. However, meta tags might not always prove to be the best case to use for all web classification projects, as many meta tags are often ignored by modern websites, which might make them unreliable for some cases [13].

The web data were extracted using text parsing and processing techniques in Python. Text extraction and e-commerce detection was done using a combination of Python string parsing, Regular Expression (RegEx) functions, and BeautifulSoup HTML parsing were used.

III. Web Text Classification

Web content classification often involves visiting web pages to categorize them based on their content. This can be done manually, but this can be slow and inconsistent. To address the growing need for efficient classification, machine learning and automation have emerged as solutions. Typically, a website's home page suffices for classification, summarizing the business's purpose and page content. For other cases, more pages might be necessary. For example, e-commerce sites may prioritize shop and product pages, but in most scenarios the home page can provides sufficient data for classification.

Despite various experiments with different features for classifying web content, the use of text on web pages remains a prevalent approach. The main benefit of using textual data on the web is the ease and speed of extraction, its reliability, and a wider choice of classification algorithms that can be used to obtain desired results.

For the software, language detection was utilized to filter non-English content before processing. The web

scraper extracts business details, social media profiles, descriptions, phones, emails, and software information. The scraper was created to ensure that only two requests are made per a single website, avoiding server overload.

Home page text, including titles, tables, and paragraphs. Additionally, web links were extracted for detection of contact or shop pages. The scraper also extracted necessary metadata like web page title, description, and keywords that can be very useful for categorization as they often describe the main topic of the website and the company and are used for SEO purposes. In theory, these tags can provide more valuable information to a classifier text on the home page, given their purpose to describe the nature of the website. Having this in mind, a decision was made to experiment and train the model on web page text, metadata tags, as well as the text extracted from the <img> tags that often contains a textual description of the website's images.

In the Table I, the distribution of categories and the top 10 most words per category are shown.

A. *Model Training*

For classification, a selection of five algorithms was chosen based on past studies, the dataset characteristics, and their varying complexities. The algorithms compared were support vector machines, multinomial logistic regression, random forest classifier, a perceptron one layer neural network model as well as a passive-aggressive classifier online-learning algorithm similar to the perceptron [14].

The two text representation techniques, TF-IDF and word embeddings , were used and their performances were compared in all tested models. For word representation, the Gensim library [15], using Word2Vec embeddings, was employed. Having gained popularity in 2013, word embeddings revolutionized the field of natural language processing, offering a word representation that can help capture some meaning in the corpus [16]. Trained from scratch with default parameters (vector size 100, window size 5, minimum count 5), the Skip-Gram algorithm, predicting context words from a target word, was then chosen for its effectiveness in representing multiple word meanings and rare words.

Hyperparameter tuning was done for all models using RandomizedSearchCV hyperparameter combinations, and cross-validation. For TF-IDF, the vectorizer and the classifier were tuned. For Word2Vec, the neural network embedding model was trained, but only the final classifier was tuned.

B. *Model Results*

The experimentation step revealed that linear models like multinomial logistic regression, perceptron, and passive-aggressive classifier favored TF-IDF text representation technique, while non-linear algorithms like SVC with a non-linear kernel and random forest classifier performed better with Word2Vec embeddings.

Based on the results of all models, the passive-aggressive classifier was selected as the final model, with TF-IDF representation of text. On the given problem, the default model with TF-IDF achieved a cross-validated score of 0.688 and 0.697 after hyperparameter tuning.

TABLE I. TOP 10 MOST COMMON WORDS FOR EACH CATEGORY

Category Name	Top 10 Most Common Words
Travel	*'hotels', 'pm', 'travel', 'flight', 'book', 'hotel', 'holiday', 'new', 'offer', 'resort'*
Social Networking and Messaging	*'chat', 'online', 'use', 'free', 'be', 'room', 'new', 'get', 'app', 'service'*
Streaming Services	*'video', 'stream', 'be', 'new', 'watch', 'get', 'tv', 'apple', 'live', 'use'*
Sports	*'be', 'league', 'sport', 'cup', 'world', 'new', 'bet', 'news', 'team', 'football'*
News	*'be', 'news', 'new', 'world', 'pm', 'us', 's', 'say', 'august', 'sep'*
Photography	*'photography', 'camera', 'image', 'photo', 'use', 'new', 'work', 'be', 'get', 'one'*
Law and Government	*'state', 'form', 'close', 'organisation', 'information', 'new', 'government', 'be', 'service', 'act'*
Health and Fitness	*'cancer', 'health', 'be', 'cells', 'use', 'get', 'body', 'care', 'test', 'help'*
Games	*'game', 'chess', 'be', 'play', 'new', 'amp', 'download', 'page', 'news', 'one'*
E-Commerce	*'car', 'insurance', 'n', 'cheapest', 'c', 'tip', 'd', 'shop', 'a', 'gift'*
Forums	*'be', 'new', 'book', 'post', 'read', 'ago', 'august', 'one', 'use', 'aug'*
Food	*'recipes', 'recipe', 'be', 'make', 'food', 'yang', 'cook', 'dan', 'indian', 'read'*
Education	*'science', 'be', 'learn', 'use', 'online', 'university', 'us', 'new', 'students', 'research'*
Computers and Technology	*'use', 'be', 'new', 'make', 'c', 'one', 'get', 'file', 'work', 'program'*
Business/Corporate	*'bank', 'service', 'business', 'us', 'market', 'august', 'home', 'use', 'read', 'financial'*

Once the best model was selected from the candidates, further experimentation involved testing whether adding metadata content helped the classifier improve its performance and whether metadata was a reliable feature should a web page not have any home text available. Further experiments tested the impact of using metadata (keywords, description, and <img> alt attribute) on model performance, exploring their potential use as alternatives when the scraper fails to extract any text from the home page. The experiment results revealed multiple things.

First, the addition of img <alt> tag textual information worsened the model results, adding a lot of image specific noise that is irrelevant to general topic of the website. On the other hand, keywords and description meta tags on their own proved to be sufficient for model training and provided similar model performance, but because keyword tags are no longer a requirement from most modern search engines - they are often omitted from many modern websites, making them unreliable for classification as the only feature source. Combining the best aspects of both, the final model was trained on the home page text, as well as the description and keyword tags, which improved model performance, provided the model more data to train on, and also offered alternatives for sources of text. For example, if a web scraper fails to extract or the home page text is too short or missing - metadata and keywords can provide additional content. Because metadata are part of the HTML structure, they load independently of the rest of the page. Due to this, using it also provides a reliable fallback option for dynamically generated web content that cannot be scraped using requests and similar libraries without a headless browser.

Testing the final model trained on home page text, keywords and description on unseen data resulted in the final macro f1 score of 0.737.

C. *Observations*

The results highlight the effectiveness of simple but robust text representation techniques such as TF-IDF in classifying web content, but also the particularities of working with textual data from the web. Specifically, web scraped data often lacks relationships between its components.

Given the challenges of poorly structured HTML and media-rich websites, good preprocessing is crucial to remove noise from web scraped data. This makes web text classification distinct from traditional text classification, and final corpus consists of many individual words, titles, and sentences that might be thematically related but do not often form a meaningful corpus. Because of this, Word2Vec models, based on co-occurrence patterns, struggled to capture document-specific information. While it is the ability to understand context that once made embedding models so popular, for this problem simpler bag-of-words models proved more suitable. Additionally, the size of the dataset could also hinder the power of Word2Vec, making simpler algorithms better for the task.

IV. PIPELINE AUTOMATION

A. *Software Choice*

MongoDB [17] was selected as the database for the software. It is a good choice for building a data pipeline due to its flexibility, ease of use, and ability to handle datasets efficiently. The setup does not require extensive infrastructure and hardware resources, making it optimal choice for both small and large projects. Its NoSQL database is beneficial when working with data sets that evolve or change frequently. It can store data with varying structures in the same database without major schema modifications. It stores data in a document format, making it well-suited for representing datasets with complex or hierarchical structures. The main reason for choosing MongoDB instead of relational databases lies in the fact that the data used in the pipelines and CRM are utilized through APIs based on JSON. As MongoDB is a document database which works with semi-structured data without additional object-relational mapping, mitigating additional mapping overhead in the system.

Apache Airflow [18] was used as an orchestration tool. It is an open-source platform revolutionizing automation by orchestrating complex workflows across distributed system components. Using Directed Acyclic Graphs (DAGs), Airflow allows users to define task sequences, dependencies, and execution conditions, facilitating automation of processes like data extraction and transformation [19]. Its user-friendly interface and modular architecture make it accessible and support step-by-step automation, while connectors enable integration with various data sources, databases, APIs, and cloud services. Companies can efficiently automate tasks, enhance data processing, and scale their automation efforts gradually with Airflow.

The software was deployed using containerization, ensuring consistent and reproducible environments and improving resource utilization.

B. *Architecture*

A high-level architecture diagram is shown on the Figure 1. A CRM system stores original data and is used by the sales and marketing teams, as well as anyone in the company managing leads. URLs entering the CRM can be

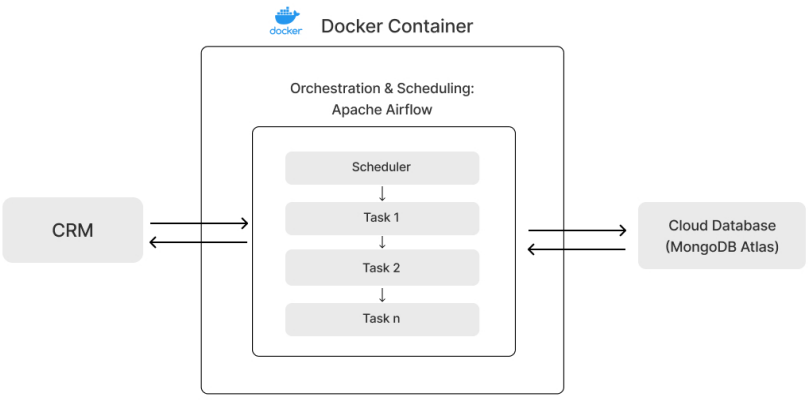

Figure 1. A high-level architecture diagram of the software.

added manually or collected through various channels, including social media marketing, form integration on company websites, and or embedded forms, or other sources.

This acts as a starting point for the pipeline and stores a list of website URLs to be processed. The data also has a unique identifier field and last scraped date to allow tracking processed leads, but also for teams working with the CRM to be able to assess the freshness of the data and understand how recently the lead was updated.

The CRM automatically updates a last updated column if a team member manually edits a lead or if any other changes have been made. The scrape timestamp is kept separate from this logic to allow for correct tracking of automated data updates. If a team member updates a property of a lead – the pipeline will not overwrite the data, unless instructed.

Each pipeline step is represented as an Airflow task. Scheduling options allow users to automate pipeline execution at regular intervals, such as hourly or daily. The user-friendly interface permits manual task runs before the scheduled time, if necessary.

Airflow can handle task failures, send notifications, log errors, or perform specified actions based on requirements. If a task fails, data are scraped on the next scheduled run. A retry mechanism was not implemented in the current scope of the software.

The DAG defines the schedule for the pipeline to run and four PythonOperator tasks representing each of the four steps of the pipeline. For this pipeline, a daily run was scheduled. Since each task fetches data from a source, processes those data, and pushes them to the next location – the code is designed to succeed if there are no new items to process. This also covers cases where no new leads have been added during the previous day, and all the leads have already been processed.

To decide whether there are any leads to process, the data are fetched from the CRM in the first task. A *scrape_date* field is used as a reference to ensure that documents that have been scraped before will not be passed through the rest of the pipeline, unless marked to do so. The task will also insert into the CRM only the documents that do not share the same unique identifier with any document in that collection, which is set to be scraped using a property called *to_scrape.* All other tasks use a Boolean sync property in each document in a MongoDB collection stored by the previous task.

The second task accepts incoming leads and sends them through a web data extraction tool to gather contact details, phone numbers, social media links, and additional data. This script also identifies used software, detects e-commerce platforms, and tags leads based on their business type - such as whether or not the business is an e-commerce business. This task uses *to_scrape* properly to filter out and process leads that should be scraped. The next task performs classification by running the machine learning model on the incoming text. The classification task is only processing websites whose document has to_categorize property set to True. Finally, the final task task pushes the processed and enriched properties to the CRM.

The synchronization allows to save on computational time that is billed by cloud platforms based on duration and avoids processing all items every time. The setup also helps preserve data in the event of a task failure. On the next successful run, the pipeline detects whether there are any documents that have not been processed in the previous run. Should a user need historical data, this can be obtained by accessing the data in the MongoDB collection.

The file used for software extraction mapping is also stored in the same folder during the text parsing step and can be updated to adapt the extraction to business needs - for example, if detection of different software is needed.

MongoDB stores data as documents that are stored in three collections - one for the incoming links, another one with enriched data and scraped output, with all the enriched data including the full-page text needed for classification. A third collection stores the final enriched data, that no longer includes the textual information, but includes the final predicted category for the website. This data are then fed to the CRM.

In each collection MongoDB automatically creates an id field for each document. This acts as a unique internal identifier and has a significant analytical value as it stores the hashed timestamp of the object creation. This makes it possible to use it for future analysis, track metrics and historical data over time.

Configuration parameters governing the parallelism and concurrency within the workflow execution were adapted for the use case. The maximum number of task instances allowed to run concurrently across all DAGs, allowing only one task instance can execute at any given time. Maximum number of active DAG runs that can be executed simultaneously for each DAG were also set to 1, indicating that only one instance of a particular DAG can be active at any given time. The maximum number of active task instances that can be executed concurrently within a single DAG were set to 4, allowing up to 4 tasks within the same DAG to execute concurrently.

These settings help control the degree of parallelism and resource utilization within Airflow, allowing users to fine-tune the execution behavior based on their specific requirements and resource constraints. Allowing only one DAG to run at a time helped ensure there is no accidental parallel running of the pipeline which could create clashes because of simultaneous insertions.

V. LIMITATIONS

The software and machine learning solution have certain limitations. Firstly, the classifier was trained on a specific sample of websites with 15 categories. If a website from a different category enters the pipeline, the model will try to classify it into one of the defined categories. While this is needed by some businesses, it might not be suitable for other needs.

Another constraint of the model is its reliance on an English corpus, limiting its performance to English-language websites. Given the diversity of languages on the web, businesses dealing with local companies may encounter challenges. To improve data quality and avoid

running the model on languages other than English, a language detection library lingua-py [20] was utilized in the study. Alternative solutions, such as using a translation API before model input, exist, but they can be costly for extensive texts and are influenced by translation quality. For best results, it is recommended to train separate models for each language or use a multilingual model.

Another limitation of the study is its exclusive focus on training a model from scratch, without exploring transfer learning and pre-trained model fine-tuning. While acknowledging the potential of such methods, this was not the primary focus of this work.

As the company expands, scalability and performance become priorities. The web scraper is adjusted with multithreading for handling more input URLs. Yet, the Machine Learning model's prediction for each pipeline item might still slow the process down. To address this, future scaling can involve using tools like Kubernetes to distribute the computational workload.

In deployment, managing data storage and size is crucial. Storing textual data for prediction in a separate collection may eventually result in significant storage requirements, potentially reaching hundreds of gigabytes. Future considerations may involve compression or deletion of text used solely for classification, as it is unnecessary for enrichment.

The software can be extended to automatically update data on a scheduled basis. For instance, implementing a rule could trigger the pipeline to enrich companies not updated within a set period, ensuring CRM data stay current. However, this feature was not implemented in this study. Furthermore, integration of external APIs and other data sources into the pipeline can enhance the CRM's completeness by providing additional company information.

The pipeline is deployed using containers, making it ready for fast deployment on any cloud provider or an on-premises server. However, more extensive testing is required for high-scale deployments to ensure the software meets desired quality standards and produces the expected outcome.

VI. CONCLUSION

Creating a deployable data processing and enrichment pipeline with an internal web extraction tool, a cloud-based database, and an open-source scheduling platform can be a solution for businesses aiming to avoid substantial upfront investments or lacking large internal technical expertise. The experiment demonstrates that, despite the limitations, it is possible to create and deploy a reliable, tailored software that can be used by companies seeking to leverage their data for automated data ingestion pipelines with machine learning, without the need for complex infrastructure.

REFERENCES

[1] *"IBM Global AI Adoption Index 2022 New* research *commissioned by IBM in partnership with Morning Consult,"* 2022. Accessed: 11.01.2024. [Online]. Available: https://www.ibm.com/downloads/cas/GVAGA3JP

[2] A. Hopkins and S. Booth, "Machine learning practices outside big tech: how resource constraints challenge responsible development," *Proceedings of the 2021 AAAI/ACM Conference on AI, Ethics, and Society*, Jul. 2021. Accessed: 10.01.2024. [Online]. doi: https://doi.org/10.1145/3461702.3462527

[3] P. Gokhale and P. Joshi, "A binary classification approach to lead identification and qualification," *Communications in computer and information science*, pp. 279–291, Jan. 2018. Accessed: 16.01.2024. [Online]. doi: https://doi.org/10.1007/978-981-13-1423-0_30

[4] J. Gitlin, "*What is lead enrichment? And how can automation elevate your lead enrichment process?*," Workato, Aug. 20, 2021. Accessed: 13.01.2024. [Online]. Available: https://www.workato.com/the-connector/what-is-lead-enrichment/

[5] B. Zhao, "Web scraping," *Encyclopedia of Big Data*, pp. 1–3, 2017. Accessed: 14.01.2024. [Online]. doi: https://doi.org/10.1007/978-3-319-32001-4_483-1.

[6] E. Uzun, T. Yerlikaya, and O. Kırat, "Comparison of python libraries used for web data extraction," in *7th International Scientific Conference 'TechSys 2018' – Engineering, Technologies and Systems,* Technical University of Sofia, Plovdiv. May 17-19, 2018, pp. 108-113.

[7] J. Goerzen, "Web client access," in *Foundations of Python Network Programming*, Apress eBooks, pp. 113–126, Jan. 2004. Accessed: 19.01.2024. [Online]. doi: https://doi.org/10.1007/978-1-4302-0752-8_6.

[8] G. L. Hajba, "Using Beautiful Soup," in *Website Scraping with Python*, pp. 41–96, 2018. Accessed: 19.01.2014. [Online]. doi: https://doi.org/10.1007/978-1-4842-3925-4_3

[9] D. Myers, and J. W. McGuffee, "Choosing Scrapy," *Journal of Computing Sciences in Colleges,* vol. 31, pp. 83-89, October 2015. Accessed: 19.01.2024. [Online]. Available: https://dl.acm.org/doi/abs/10.5555/2831373.2831387

[10] *Apache NutchTM.* Accessed: 11.01.2024. [Online]. Available: https://nutch.apache.org/

[11] *lxml - Processing XML and HTML with Python,* Accessed: 11.01.2024. [Online]. Available: https://lxml.de/

[12] A. Patil, and B. Pawar, "Automated classification of web sites using naive bayesian algorithm," Accessed: 11.01.2024. [Online]. https://www.semanticscholar.org/paper/Automated-Classification-of-Web-Sites-using-Naive-Patil-Pawar/d1a069bd1c58473ea0e66d8c4f8a41c5cd69fa34

[13] J. M. Pierre, "On the automated classification of web sites," *Computer and Information Science*, vol. 6, p. 0, Feb. 2001, Accessed: 20.12.2023. [Online]. Available: https://arxiv.org/abs/cs/0102002v1

[14] *1. Supervised learning — scikit-learn 1.4.0 documentation.* Accessed: 20.01.2024. [Online]. Available: https://scikit-learn.org/stable/supervised_learning.html

[15] *Gensim: topic modelling for humans.* Accessed: 20.01.2024. [Online]. Available: https://radimrehurek.com/gensim/auto_examples/

[16] S. Vajjala, B. Majumder, A. Gupta, and H. Surana, *Practical Natural Language Processing*. O'Reilly Media, Inc., 2020.

[17] S. Bradshaw, E. Brazi, and K. Chodorow, "*MongoDB: The Definitive Guide, 3rd Edition.*" Accessed: 24.12.2023. [Online]. Available: https://www.oreilly.com/library/view/mongodb-the-definitive/9781491954454/

[18] *Apache Airflow*. Accessed: Sep. 20, 2023. [Online]. Available: https://airflow.apache.org/

[19] B. Harenslak and J. de Ruiter, "Data Pipelines With Apache Airflow," p. 454, May 2021, Accessed: Sep. 20, 2023. [Online]. Available: https://www.oreilly.com/library/view/data-pipelines-with/9781617296901/

[20] *pemistahl/lingua-py*. Accessed: Sep. 25, 2023. [Online]. Available: https://github.com/pemistahl/lingua-py