# Maestro: An Extensible General-Purpose Data Gathering and Classification Platform

***Alexandre Magalhães Serra***
*INESC-ID, Instituto Superior Técnico, Universidade de Lisboa*
*Lisboa, Portugal* *alexandre.serra@tecnico.ulisboa.pt*

***Alberto Rodrigues da Silva***
*INESC-ID, Instituto Superior Técnico, Universidade de Lisboa*
*Lisboa, Portugal* *alberto.silva@tecnico.ulisboa.pt*

***Jacinto Estima***
*INESC-ID, Lisboa, Portugal*
*CISUC, Dep. of Informatics Engineering, University of Coimbra*
*Coimbra, Portugal* *estima@dei.uc.pt*

**Abstract**

Researchers who want to gather and classify data on a specific topic are doomed to use several tools in a tedious process given the lack of software tools to collect data from multiple sources for posterior analysis and classification. Our study addresses these issues by designing a novel software platform named Maestro that automatically gathers, classifies, and provides specific datasets from a dynamic set of configurable components (plugins). Extensibility is Maestro's main feature, which allows new plugins to be incrementally added by the core team or other developers without changing the source code. To evaluate this proposal and support the discussion, a simple working example with images of the former U.S. president, Donald Trump and his facial expressions is shown.

**Keywords:** Data Gathering, Data Classification, Data Providing, Data Science

## 1. Introduction

The World Wide Web, hereafter referred to as Web, has become the most extensive wealth of information in human civilization, containing information about virtually every topic. More so, ever since the dawn of the so-called "Web 2.0" and the popularization of "The Web as a platform" [8] where users were able to produce their content, this trend has only increased over these last decades. During the last few years, there have been several efforts to combine Web crawling with machine learning (ML) [2–4]. These technologies usually involve data gathering through Web crawlers and then feeding the obtained data to a ML model, either as training data or for classification. A key advantage of Web crawlers compared to prepared data sets is providing timely real-world data, which allows ML models to generalize better to unseen scenarios. Since ML requires many examples to generalize well and the Web is the humankind's biggest source of data, some works that use data from the Web to train ML algorithms have been proposed [5, 7, 10]. However, since the scope of this work is not to research ML algorithms, we leverage already trained ML algorithms and treat them as regular algorithms.

This paper proposes and discusses a novel platform for data gathering and classification, named Maestro (available at https://maestro.ml). Maestro is an extensible and configurable system that allows users to create and customize search contexts that automatically collect and classify data through multiple plugins, and then, the resulting data from this process can be exported or fed another application services.

This paper is structured in 7 sections. Section 2 introduces the background concepts that are related to the techniques Maestro uses. Section 3 introduces Maestro key concepts and overviews its workflow. Section 4 presents the Maestro software architecture, namely its modular and pluggable architecture. Section 5 shows the use of Maestro based on a simple and illustrative example. Section 6 compares and discusses the related works. Finally, section 7 wraps up and gives an insight into future research.

## 2. Background

Web crawlers are software systems that wander through the Web by analyzing Web pages content and extracting relevant data. The two major types of crawlers are [6]: generic Web crawlers and focused Web crawlers. The goal of the former is to traverse through as many pages as possible to index and rank them. The latter type of Web crawler tries to find more fine-grained information about a specific end target.

The results of Web crawlers can be obtained in two ways: by crawling them or accessing them through an API. Most providers block the first option, therefore, we mainly use APIs to access this data. Usually, APIs make available a set of filtering parameters that can be very useful when narrowing the results to the user's preferences. Most data sources can be accessed through APIs. Popular examples that are used in our system are social network APIs. These allow accessing the social network's inner data, such as posts, and filtering it by criteria like hashtags, location, and date. For instance, Twitter API [16] allows finding tweets through many parameters.

On the other hand, data classification algorithms match a data input into an output label. It is considerably associated with ML algorithms, but that does not need to be the case. For instance, a simple classification algorithm can use a set of conditional expressions to fit the data in some class. However, that usually does not scale very well, which is why artificial intelligence classification algorithms have become so popular with techniques like decision trees, convolutional neural networks, or transformers [9].

With Maestro, we do not intend to create a new Web crawler but, instead, use existing ones to support its gathering feature because Web crawlers require a significant amount of resources such as computation power and memory. It is not our goal to dig into these aspects. Likewise, we do not intend to research classification algorithms in the scope of this work. We just considered that these algorithms are ready-to-use programs that can be accessed through standard interfaces. In the particular case of ML applications, these algorithms shall be already trained and ready to be properly integrated and used.

## 3. Maestro Key Concepts and Workflow

Maestro can be described from two perspectives, namely the user perspective and the system's perspective. Some concepts overlap, and others are invisible to its users to prevent overwhelming them with technical concepts.

### 3.1. Key Concepts: Organizations, Users, and Search Contexts

Maestro is a multi-user and multi-organization platform. Users can create and manage "Search Contexts", which is Maestro core concept. In a Search Context, users define what they want to search for, classify, and deliver based on configurable parameters.

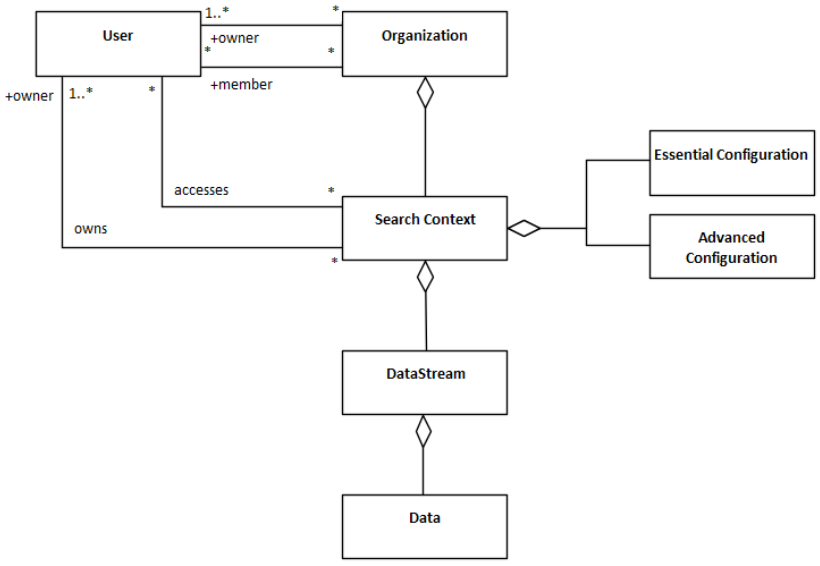


**Fig. 1.** Maestro top-level concepts (UML class diagram).

An organization may be created by a user who becomes its owner. This owner may invite other users that become organization members. An owner may then manage the permissions of organization members and perform other access control actions. Depending on the permissions set by the owner, organization members may consult or manage the organization's search contexts. The data gathered in a Search Context is a datastream, i.e., a collection of data objects. The datastream is progressively enhanced according to the Search Context's configurations. Users can customize the behavior of the Search Context by tailoring the results for a concrete use case. The essential configuration object is required and represents the bare minimum configuration a Search Context needs to execute, and it includes fields such as the search term and keywords. Complementary, the advanced configuration allows for further refine the potential output results, allowing for much more fine-grained control.

### 3.2. Maestro Workflow

The Maestro workflow is a sequence of steps that Maestro must follow to fully process a Search Context. This workflow works asynchronously, meaning that when a Search Context starts, the server processes the request in the background but without preventing its ability to interact with the user. A process is spawned to handle this workflow and concludes when the job finishes, as shown in **Fig. 2**.

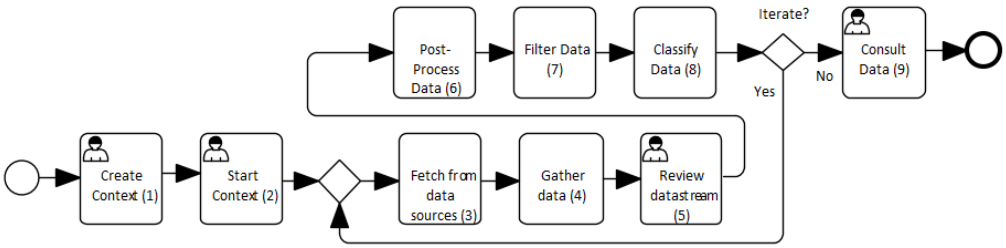


**Fig. 2.** Maestro workflow (BPMN process diagram).

After creating the Search Context and (at least) the definition of the core configuration, the user can trigger its execution. The system starts by fetching the URLs from its data sources (fetchers) and downloads the results to its file system (through the gatherer). Next, if the user wants to inspect the obtained datastream manually, the system stops and waits for the user to resume the execution. Once it is resumed, the post-processing step starts, enhancing the datastream. The data objects can then be filtered in the filtering stage, which is the last step responsible for improving the datastream according to the defined configurations. The datastream is then classified and, if the number of objects is greater than a defined minimum, this datastream is sent through a webhook to a target service. Otherwise, the workflow repeats, as suggested in **Fig. 2**.

## 4. Maestro Platform

Maestro is defined as a modular and extensible platform decomposed into a set of well-defined plugins. These plugins are executed asynchronously and sequentially with the datastream obtained from previous steps.

Due to its extensibility, Maestro allows new components to be added to enhance its capabilities. New fetchers, gatherers, post-processors, filters, and classifiers can be added easily. A developer uses Maestro's documentation to build a compatible component and submits it for review. If it is accepted by a Maestro admin, then it will be added to the system through Maestro's back-office, which allows managing all the existing components (at the time of writing, only Python plugins are supported, but integration with other languages is a possibility).

Maestro provides a hierarchy of internal plugins, explained in the following sections. Currently, Maestro supports plugins such as: "Bing Image API" to fetch URLs for images; "Default Image Gatherer" downloads images from URLs; "Exif retriever" post-processor extracts metadata from images; location and date filters based on the retrieved metadata; "Facial expression classifier" classifies images with people regarding the expression on their faces; webhook provider allows for server-to-server asynchronous communication.

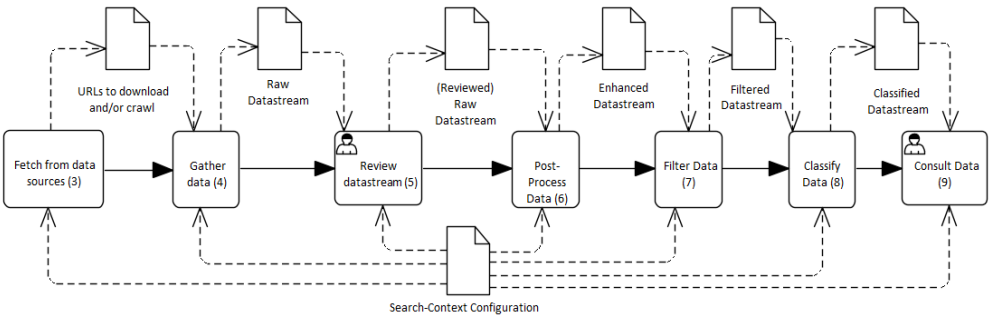


**Fig. 3.** Maestro workflow (BPMN process diagram).

**Fig. 3** emphasizes the flow and evolution of the datastream. As referred above, the datastream represents the set of all data objects associated to the search context which, during the execution of the workflow, are transformed with the goal of matching to the configurations the user set for the search context.

Each individual data object of the datastream has two major components: the data itself (e.g., the image, text, sound), and the respective metadata. The latter is continuously updated throughout the execution of the search context and plays a crucial role on the refinement of the search context.

For example, considering that a search context was configured to look for images of flood events in a particular region. In stage (3), URLs are fetched from several sources including Twitter which may provide the location where the tweets were posted. Therefore a "location" field can be defined in the data object's metadata and set it to this location obtained from Twitter. In stage (6), the "Exif retriever" may be used to extract Exif [1] metadata from the images, which can include the latitude and longitude of where it was taken. We can then update the "location" field in the data object's metadata to this value because the location extracted from the image's metadata is more accurate than the one obtained from the tweets. Finally, in the filtering stage (7), the search context could be configured to exclude data objects that were not in a given location.

There are some relationships between datastream, data and metadata. The datastream is composed by a set of data objects that move together along the workflow. The size of this datastream is variable, since it depends on how many data objects the system is able to gather from the URLs provided by the fetchers. Data is an abstract element whose properties are implemented by one of its specializations. The type of data that the search context is configured to gather determines the instance of the data element that is used, and the data subclasses have specific properties for that datatype. The metadata element can hold multiple properties related to the data object, such as: source of the data, post-processing, and classification results when it was gathered. The flexibility of the metadata element is supported using a non-structured data model to store these properties, which allows to store arbitrary key-value pairs in JSON format. This enables metadata properties to be incrementally added to the data objects through new plugins without the need of changing the underlying model.

**Fetchers**. Fetchers are a central component of Maestro because they support the entry point of the whole process. Fetchers retrieve URLs from a data source and transfer those to the workflow's next step. These data sources can be an API that exposes endpoints to access a specific service, an RSS feed, or any source that returns URL links to documents or media files. Fetcher services allow some sort of filtering or tuning to be defined on the request. For example, some APIs allow querying for data in a specific date interval. To take advantage of these capabilities, Maestro provides to the fetchers relevant information about the Search Context configuration, allowing them to use this information to retrieve more meaningful results. Multiple fetchers can be used for the same Search Context. The greater the number of fetchers is used, the more URLs are obtained and, therefore, more data objects can be possibly retrieved.

**Gatherers**. Gatherers are responsible for turning the URLs obtained by the Fetchers into actual data that can be further processed. These URLs can lead to two types of data: media files or hypertext markup language (HTML) documents. URLs for media files are

straightforward because they point directly to the media file. Thus, the gatherer's only job is downloading that file to Maestro's filesystem. Links that point to HTML documents require a different approach. Because relevant data may be nested inside the document object model (DOM) or inside webpages linked by the original document. Hence, a Web crawler is used to handle the parsing and traversal through the documents' structure and find URLs to related webpages. The crawler's approach to extracting data from documents can vary depending on the use case, making it impossible to have a single fit for all cases. So, Maestro provides a simple implementation of a crawler for each supported data type and leaves room for users to add their implementation of a crawler written in Scrapy [15].

**Post-Processors**. After gathering as much data as possible, the post-processing stage and the filtering stage try to tailor the datastream to the user's preferences. Two types of post-processors are possible: data manipulation and metadata retrieval post-processors. The former kind allows manipulating the individual data objects to uniformize them. For example, image data objects can be rotated, sound data objects can have background noise, and text data objects can contain slang. It may be beneficial to have all images in portrait, clean sound objects, and slang-free text. A post-processor can be used for these applications. The latter kind extracts some metadata from the data object that can be used in the following stage (filtering) to remove some data objects from the datastream. For example, one may want to retrieve EXIF [1] metadata from images, which contain information such as location and date where the image was taken. This information is associated with each data object and can be used in the next stage to remove data objects.

**Filters**. Filters remove data objects from the datastream according to their actual data or by the metadata associated with them, extracted in the previous stage. Filters can therefore be of one of two types: data filters or metadata filters. The result of a filter is a conditional expression that states whether the data object matched the filtering condition or not. Data filters check for some properties associated with the data object itself. For instance, one may want to filter out grayscale images or intense sounds. Metadata filters conditionally remove data objects from the datastream based on the metadata associated with them collected in the post-processors. There is a high correlation between post-processors and filters. Thus, some operations can be done on both ends. In the image orientation example, we may rotate the images to portrait on the post-processor, filter the non-rotated images on the filters or do both. This highlights the difference in approaches one may have when building a Search Context.

**Classifiers**. So far, previous components have been refining the datastream to curate it to the user's preference. The classifiers represent one of Maestro's main characteristics: the ability to classify the data gathered from multiple sources. Classifiers can be a wide range of programs. They can vary from simple decision trees to highly complex ML algorithms. Maestro handles them always in the same way: as a black box. The system passes the data objects to the classifier, and it outputs another data object whose type can be different from the input. An image classification algorithm usually receives as input an image and outputs a text string. It is worth mentioning that Maestro, unlike other similar applications, expects them to be already trained when handling ML programs.

**Providers**. The final component of Maestro is the list of providers. These are responsible for providing the classified data to external services for them to use for their use-cases. The most common provider is expected to be the webhooks provider. Through webhooks, a user can specify a REST endpoint for an external service to which the data will be sent in a POST request after the classification step. This requires that the external service is online. If that does not verify, the user responsible for the search context can manually execute this step afterward. One can also download the results directly to the filesystem for manual inspection.

The components above described interact with each other using procedure calls. When one finishes successfully, it calls the next component to continue the process. The data is shared through a relational database, which is accessible from all components.

## 5. Running Example: The Donald Trump's Dataset of Images!

This section shows how to define and execute a simple search context, which intends to collect images of the former U.S. president, Donald Trump, and classify them with a ML algorithm that captures facial expressions, and labels expressions as "Excited" or "Happy".

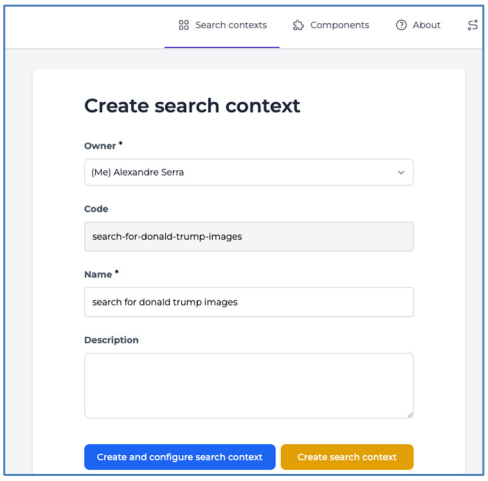


**Fig. 4**. Create a search context view.

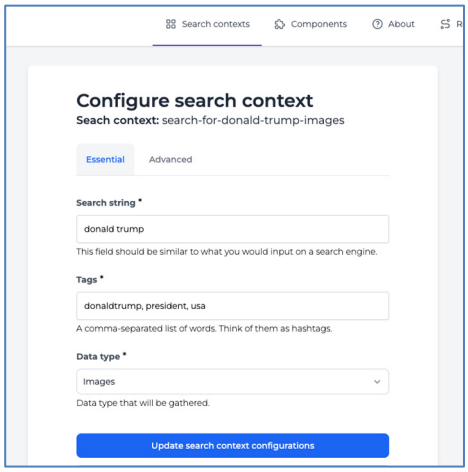


**Fig. 5.** Essential search context configuration.

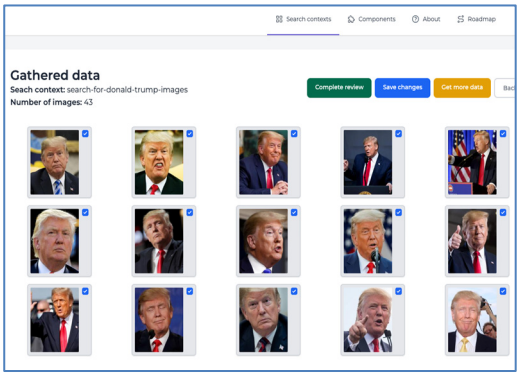


**Fig. 6.** Search context gathering results.

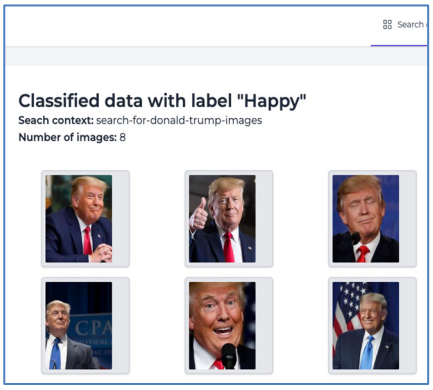


**Fig. 7.** Results after using the "Facial expression" classifier filtered by the label "Happy".

**Task 1**. The user creates a search context through the Maestro's interface as shown in **Fig. 4**. A search context has an owner, a name, and a unique code. After creating the search context, the user must configure it. There are two types of configurations: essential configuration and advanced configuration, where the former is mandatory, and the latter is optional. The essential configuration includes the parameters required for a search context execution. By default, the system automatically defines all other parameters (i.e., configurable in the advanced configuration). The advanced configuration includes parameters that allow tuning the Search Context to be more suitable to the user's preferences but requires more understanding from the users. We only configure the essential configuration for this demonstration, as shown in **Fig. 5**.

**Task 2**. The user starts the execution of the search context. This operation triggers a background process on the server, and the user can monitor the progress of this process by clicking the button "Progress" that appears after the context is started.

**Tasks 3..5**. Depending on the configuration, the process stops after the gathering stage. This behavior is configurable by the "Yield after gathering data" parameter, which is enabled by default. This option allows the user to validate the collected data by removing data that he/she might find unfit or even to get more data (Task. 5). Since we did not configure this parameter, it is enabled, and therefore the process stopped after the data gathering (see the result in **Fig. 6**).

**Tasks 6..8**. After making the desired modifications to the dataset, the user can click on the "Continue" button, which continues Maestro's progress, proceeding to the enhancing stage, where the data is post-processed (Task 6) and filtered (Task 7). Then, the

classification (Task 8) starts, and the ML algorithm is used to classify the facial expressions of the former U.S. president. **Fig. 7** shows the images that were identified as "Happy" by the "Facial expression" classifier.

**Task 9**. The collected data, namely the classification results, are sent to an external service connected to the internet through a communication protocol. The progress of these steps can be followed through a progress bar that shows the current execution stage and the operations that each step has already performed.

## 6. Related Work

From our analysis, we did not find any system like Maestro that combines all the features discussed in this innovative platform.

**Table** 1 shows a summary of the main analyzed features. Some tools combined gathering and classification features, using the data gathered from the internet to train neural networks, thus allowing for real-world data.

K. Yanai [10] proposed a system that uses the Image Collector [11, 12] that, based on a set of keywords, collects images from the Web and uses them to train a neural network for image classification. Similarly, Lee et al. [7] described a technique that uses images of cars scraped from Web pages, combining the original images and images derived from the originals by applying transformations like flipping, to train a deep neural network.

Others have applied ML directly to the crawling process. For instance, Johnson et al. [4] use a genetic learning algorithm to learn strategies to rank URLs and order the crawled results. Jiang et al. discuss a technique based on deep reinforcement learning to find the next page to crawl by learning based on a reward system that rewards pages that contain the keyword that is being searched for [2].

inTIME [5] is a platform used in the Cyber-Threat Intelligence (CTI) domain, that allows to identify and extract CTI and security artefacts from various data sources such as webpages and social media networks, classify them through ML algorithms to extract CTI features (e.g., use natural language processing to find if a CVE [13] is being mentioned) and then export to the MISP database, which is an open-source collaborative threat intelligence sharing platform [14]. Like Maestro, inTIME allows users to configure the sources from which the data will be fetched.

Maestro differentiates from this related works because its goal is not only to crawl the Web, or train some classification application; instead, it provides an integrated service capable of both gathering and classifying data, built in an extensible and customizable way. The data processed by this system can thereafter be exported to external services so that these can leverage this data and use it to improve the capabilities to solve problems on their domains. Furthermore, Maestro is built with pluggability at its core, which can be used in various application domains. On the other hand, a disadvantage of its flexible and modular architecture is that it may underperform other more focused systems because it may not be able to employ specific tasks for some narrow use-cases. Moreover, Maestro sets itself as a service, available to be used by other systems as a resource to enhance their capabilities, without having to implement data gathering and classification functionalities. From our perspective, this is one of the key aspects that differentiates Maestro from other tools.

Table 1. Comparison between Maestro and similar works

| Work | Gather data | Enhance data | Classify data | Train classifier | Provide to external services |
|---|---|---|---|---|---|
| Maestro (this paper) | Yes | Yes | Yes | No | Yes |
| inTIME [5] | Yes | Yes | Yes | No | No |
| Image Collector [11, 12] | Yes | No | No | No | No |
| Generic Image… [10] | Yes | No | Yes | Yes | No |

## 7. Conclusion and Future Work

Nowadays we have access to large amounts of information through the internet. Ever since the dawn of Web 2.0, this trend has only increased, because of the active participation of users. However, this can also be a problem because information is now scattered around the Web. Therefore, it is no surprise why search engines like Google Search are on the top

of the most visited websites: with their extensive processing capabilities, they index webpages based on a set of keywords provided by a user, making it easier to find information in the Web. Nonetheless, search engines have their limitations. For instance, they cannot scrape the information from meaningful and highly active sources such as social networks because these systems block such actions, neither can have very fine-grained queries. Maestro provides a system capable of having "smart" searches, that can gather information from multiple data sources like search engines, social networks, and others, and also classify that data, simultaneously exposing a general-purpose architecture that can be used in various domains.

Maestro is still in development, and therefore additional updates soon would be introduced. Designed with modularity and extensibility as first-order priorities, many services in different and unrelated areas would use it to enhance their data processing mechanisms. After developing its core features, we intend to create a collaborative environment where developers can submit their plugins to enhance and extend Maestro and allow it to support more use cases increasingly. We are also planning to perform a quantitative evaluation of the platform with end users. Another interesting idea to further improve the platform is to verify whether the content gathered within a search context is true and has not been maliciously modified.

## Acknowledgements

Research partially funded by FCT UIDB/50021/2020 and 02/SAICT/2017/29360.

## References

1. Japan Electronics and Information Technology Industries Association: Exchangeable image file format for digital still cameras: Exif Version 2.2. 154
2. Jiang, L., Wu, Z., Feng, Q., Liu, J., Zheng, Q.: Efficient deep web crawling using reinforcement learning. In: Pacific-Asia Conference on Knowledge Discovery and Data Mining. pp. 428–439. Springer (2010)
3. Jiang, L., Wu, Z., Zheng, Q., Liu, J.: Learning deep web crawling with diverse features. In: 2009 IEEE/WIC/ACM International Joint Conference on Web Intelligence and Intelligent Agent Technology. pp. 572–575. IEEE (2009)
4. Johnson, J., Tsioutsiouliklis, K., Giles, C.L.: Evolving strategies for focused web crawling. In: Proceedings of the 20th international conference on machine learning (ICML-03) (2003)
5. Koloveas, P., Chantzios, T., Alevizopoulou, S., Skiadopoulos, S., Tryfonopoulos, C.: intime: A machine learning-based framework for gathering and leveraging web data to cyber-threat intelligence. Electronics. 10 (7), 818 (2021)
6. Kumar, M., Bhatia, R., Rattan, D.: A survey of Web crawlers for information retrieval. Wiley Interdiscip. Rev. Data Min. Knowl. Discov. 7 (6), e1218 (2017)
7. Lee, Y., Kang, S.-J.: Web Scraping Crawling-based Automatic Data Augmentation for Deep Neural Networks-based Vehicle Classifications. In: 2019 IEEE International Conference on Consumer Electronics (ICCE). pp. 1–2. IEEE (2019)
8. O'reilly, T.: What is web 2.0. O'Reilly Media, Inc. (2009)
9. Sen, P.C., Hajra, M., Ghosh, M.: Supervised classification algorithms in machine learning: A survey and review. In: Emerging technology in modelling and graphics. Springer (2020)
10. Yanai, K.: Generic image classification using visual knowledge on the web. In: Proceedings of the eleventh ACM international conference on Multimedia. pp. 167–176. (2003)
11. Yanai, K.: Image Collector: An Image-Gathering System From The World-Wide Web Employing Keyword-Based Search Engines. In: ICME. (2001)
12. Yanai, K.: Image collector II: A system for gathering more than one thousand images from the web for one keyword. In: 2003 International Conference on Multimedia and Expo. ICME'03. Proceedings (Cat. No. 03TH8698). p. I–785. IEEE (2003)
13. CVE - CVE, https://cve.mitre.org/, Accessed: April 12, 2022
14. MISP Open Source Threat Intelligence Platform & Open Standards For Threat Information Sharing, https://www.misp-project.org/, Accessed: March 30, 2022
15. Scrapy | A Fast and Powerful Scraping and Web Crawling Framework, https://scrapy.org/, Accessed: March 30, 2022
16. Twitter API | Products | Twitter Developer Platform, https://developer.twitter.com/en/products/twitter-api, Accessed: March 29, 2022