Real-Time Automated Mapping of Drone Airstrikes

Ayoub Kachkach

Supervised by Naeem Nisar Sheikh

Spring 2019
Real-Time Automated Mapping of Drone Airstrikes

Capstone Report

Student Statement
I have applied ethics to the design process and in the selection of the final proposed design. I have held the safety of the public to be paramount and have addressed this in the presented design wherever may be applicable

Ayoub Kachkach

Approved by the supervisor

Naeem Nisar Sheikh
Acknowledgement

I would like to thank all the colleagues, faculty and staff who made my journey at AUI the best years of my life. Special thanks to professor Naeem Nisar Sheikh for coming up with the topic of this thesis and for his invaluable help throughout. I would also like to express gratitude to my parents who consistently gave me financial and emotional support. I am also grateful to my two brothers, the role-models from which I learned many valuable lessons.
# Contents

1 Introduction 1

2 Background 4

2.1 Web Crawling and Scraping ................................................. 4

2.1.1 The Web, in a Nutshell .................................................. 4

2.1.2 Web Scraping ............................................................ 5

2.1.3 Web Crawling ............................................................. 6

2.1.4 Literature Review ........................................................ 8

2.2 Classification ............................................................... 11

2.2.1 Evaluating Classifiers .................................................. 11

2.2.2 Random Forests .......................................................... 13

3 Requirements Specification 16

4 Design and Implementation 17

4.1 Web Crawler and Scraper .................................................. 17

4.1.1 The Scraper ............................................................... 18

4.1.2 The Crawler .............................................................. 20

4.1.3 Adding News Sources .................................................. 23

4.1.4 Technology Enablers .................................................... 24

4.2 Classifier ................................................................. 26

4.2.1 First Algorithm: Keyword-based Classification .................... 26

4.2.2 Second Algorithm: Learning Models ................................ 31

4.2.3 Technology Enablers .................................................... 38

4.2.4 Conclusion ............................................................... 38

4.3 Information Extraction .................................................... 39
5 STEEPELE Analysis

5.1 Societal ................................................................. 40
5.2 Political ................................................................. 40
5.3 Technical ............................................................... 40
5.4 Ethical ................................................................. 41
5.5 Legal ................................................................. 41
5.6 Economic ............................................................. 42
5.7 Environmental ....................................................... 42

6 Final Remarks ......................................................... 43

6.1 Challenges and Limitations ........................................ 43
6.2 Future Work .......................................................... 43
6.3 Conclusion ............................................................. 43
List of Figures

1  Number of reported strikes under Bush’s and Obama’s tenure. . . . . . . . 3
2  Client requesting a web page from server. . . . . . . . . . . . . . . . . . . 4
3  Synchronous vs. Asynchronous requests. . . . . . . . . . . . . . . . . . . . 5
4  The phases of scraping. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 5
5  The phases of crawling. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 7
6  From HTML to DOM tree. . . . . . . . . . . . . . . . . . . . . . . . . . . . 9
7  Simple decision tree. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 13
8  Data-flow diagram of Drone-Watch. . . . . . . . . . . . . . . . . . . . . . . 17
9  Drone-Watch scraping logic. . . . . . . . . . . . . . . . . . . . . . . . . . . 18
10 Drone-Watch crawling logic. . . . . . . . . . . . . . . . . . . . . . . . . . . 20
11 Website class holding information about the structure of websites. . . . . . 23
12 Plotting accuracy, recall and precision for first attempt. . . . . . . . . . . . 27
13 Confusion matrix of first attempt. . . . . . . . . . . . . . . . . . . . . . . . 28
14 Accuracy vs. length considered. . . . . . . . . . . . . . . . . . . . . . . . . 29
15 Plotting accuracy, recall and precision for second iteration. . . . . . . . . . 30
16 Confusion matrix for keyword-search, 2nd iteration. . . . . . . . . . . . . . 31
17 Scatter plot showing 2d projection of a sample of our training data. . . . . . 33
18 Mean accuracy of 8-fold cross validations on different models. . . . . . . . 34
19 10 most important words as shown by our initial random forest model. . . . 35
20 10 most important words as shown by our 2nd random forest model. . . . . 36
21 Cross validation result for 2nd model. . . . . . . . . . . . . . . . . . . . . . 37
22 Confusion matrix for 2nd random forest model. . . . . . . . . . . . . . . . 37
23 UI of the labeler. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 39
Abstract

Drone-Watch is a software that facilitates the collection of data on drone airstrikes from newspaper articles. This was achieved by building 3 main components: a generic web crawler/scraper, a classifier and a labeler.

The web crawler is generic as it allows scraping news articles from a variety of news sources. Adding new sources requires little to no programming knowledge from the operator and can be done in only a few minutes. Additionally, the crawler allows scraping historical data by traversing the archives. Drone-Watch also has a built-in classifier that automatically discards irrelevant news articles keeping only those reporting on drone airstrikes. Finally, a built-in labeler UI allows the user to quickly mark data of interest from articles and save them into a database, in structured format.

The crawler and scraper were tested on 5 different news sources with significantly different layouts, achieving an average of 0.5 articles scraped per second. The classifier was tested on a labeled data set of 200 articles, achieving a maximum accuracy of 94%. The aforementioned articles were all labeled using Drone-Watch built-in labeler UI which allowed to store date, location and casualties (injury and death) of strikes in a database.

Drone-Watch design is generic enough that it can be easily reused to build trackers of other topics in the news such as journalist imprisonment, human-rights infringements and other issues that need consistent tracking.
1 Introduction

In 2000, the US Airforce and the CIA became the first to build drones with missile-launching capabilities [1]. It was first used in Afghanistan in 2002 to kill a “tall man to whom other were acting in reverence”, suspected to be Osama Bin Laden [20]. It turned out to be innocents gathering metal scraps. This sort of incidents became characteristic of the drone warfare era. Reporters and organizations concerned about the ethical issues of such strikes found and still find it harder to investigate as these operations became more and more covert and moved away from the eyes of the media and the public. Although 9/11 certainly increased US will to engage in drone warfare, it is during Obama’s presidency that the number of strikes spiked, as shown in figure 1. According to the Bureau of Investigative Journalism, 769 to 1725 civilians were killed from 2004 to now [1]. That averages 1 civilian killed every 4 strikes. Another issue that stems from the total lack of transparency in these strikes is how victim are later identified. The margin between civilian and militant is often thinner than it seems.

On March 6th 2019, Donald Trump signed an executive order to drop the Obama-era rule that required the CIA to publish annual reports on civilian casualties from drone strikes [15]. This was the latest hit to transparency in reporting about US-led drone strikes. Official reports on drone strikes from the US military have for a long time been considered unreliable [21]. They were now becoming inexistent. Journalists and watch-dogs now have to rely on other sources, the most important of which are arguably newspapers. It is in this context that Drone-Watch, a software that aims to facilitate the tracking and extraction of data on drone strikes from newspaper articles was developed.

Automatic extraction of drone strike data is not a novel idea. Josh Begley’s iPhone app Metadata+, which he created in 2012, achieved the same goal. It delivered notifications about strikes to its users' iPhones and even summarized the data in a map [2]. However, we feel that the app presented many weaknesses. Firstly, its reliance on the Apple App Store
made it fragile and dependent on Apple’s whims. The app was banned for a total of 12 times from the platform for various questionable reasons such as its crude content and its lack of meaningful features [10]. Secondly, the app got its data exclusively from the Bureau of Investigate Journalism’s manually curated database which offers structured, easy-to-read data. As such, it was heavily dependent on the availability of such data. If, for any reason, the BIJ stopped publishing such data, the app would become completely obsolete.

It is in this respect that our solution’s novelty resides. Drone-Watch is meant to be self-contained. It extracts articles from publicly available news websites, filters them to only keep those relevant to the task and then offers them to the user for labeling using the app’s built-in labeler. Drone-Watch aims is to achieve minimal dependency on manual operators. Our software’s strength resides in the ease with which it allows adding new sources, increasing its robustness and decreasing its reliance on a small number of references. This report will explore the technical details of the project, challenges faced as well as future work. Drone Watch’s codebase is hosted at https://github.com/ayoubkachkach/drone-watch.
Figure 1: Number of reported strikes under Bush’s and Obama’s tenure.
2 Background

2.1 Web Crawling and Scraping

Although seemingly synonymous, web crawling and web scraping are different techniques. This section will aim to differentiate between these two crucial steps in our program.

2.1.1 The Web, in a Nutshell

To understand either one of those techniques, one should have basic knowledge of how the web works. Figure 2 shows a simple scenario of client-server communication: the client sends a GET HTTP request to the server which responds back with the HTML of the page requested. Usually, this request is performed synchronously: communication ping-pongs between the client and the web server. As such, the client’s browser blocks after sending the request and only continues execution after a response is received.

<table>
<thead>
<tr>
<th>Client</th>
<th>HTTP Request</th>
</tr>
</thead>
<tbody>
<tr>
<td>http://</td>
<td></td>
</tr>
<tr>
<td>Web Server</td>
<td>HTML</td>
</tr>
</tbody>
</table>

Figure 2: Client requesting a web page from server.

An asynchronous request is one where the client does not block waiting for the server’s response. It continues execution and uses the server’s response only once received. The difference between synchronous and asynchronous requests is illustrated in figure 3. AJAX, short
for asynchronous JavaScript and XML, is a technique that takes advantage of asynchronous requests to update a web page without reloading it. It consists of sending a request behind the scenes to which the server responds by usually sending XML or JSON data. The data is then added to the HTML at the client-side, that is using the client’s machine and not the server. The page is not reloaded during the process. Understanding this concept will be crucial to understanding some of the finer details of the scraping strategy we followed.

Figure 3: Synchronous vs. Asynchronous requests.

2.1.2 Web Scraping

Figure 4: The phases of scraping.

Ryan Mitchell’s Web ScrapeIng with Python [18] has been an extremely useful resource
for us throughout the crawling and scraping phases of the project.

According to [18, p.10]

Web scraping is the practice of gathering data through any means other than a program interacting with an API (or, obviously, through a human using a web browser).

The difficulty of scraping comes with the unstructured nature of the data it deals with. In practice, it is most often used to extract specific data from an HTML document (i.e. Webpage) after querying a server. This involves parsing the HTML document for specific elements of interest. As opposed to structured data which are sent by APIs, HTML documents structure data visually, and not semantically. It is for this reason that web scrapers are known to be vulnerable to website layout changes. The details of scraping will be explained in a subsequent chapter. Figure 4 gives an overview of the process.

An HTML web page is given as input to our scraper. The HTML web page contains a lot more than the information we need: it might contain styling and irrelevant information from the website or ads services. It is the scraper’s task to parse the HTML file to extract only the information relevant to our task to eventually turn it into structured data that could be stored in a database.

2.1.3 Web Crawling

According to [18, p.33]

Web crawlers are called such because they crawl across the web. [...] They must retrieve page contents for a URL, examine that page for another URL, and retrieve that page, ad infinitum.

Hence, a web crawler’s task is to gather a list of URLs for web pages of interest. For example to scrape news articles, a crawler need to first search the website for the web pages
of those articles. To achieve that, we might start our spider, the web crawler, from the website’s homepage. The crawler will then get all the URLs in the homepage and crawl those recursively while saving the URL of any news article it comes across. The output of this operation is a list of URLs that can be individually fed into a scraper afterwards. This is one of the main building blocks of search engines, who crawl the whole web, and not only one website. Figure 5 gives an overview of crawling.

![Image: The phases of crawling.](image)

**Figure 5:** The phases of crawling.

When crawling, it is necessary to take into account many things. Firstly, the spider should not visit the same page more than once as this might result in an infinite recursion. Therefore, de-duplication should take place before crawling every URL. Secondly, the crawler should regulate the speed at which it traverses websites. This problem comes from the bottleneck that occurs between the speed at which an unregulated crawler requests pages and the
speed at which a server sends back responses. The machine running the scraper can make thousands of requests in a second. However, doing it like this will, in the best case, get the crawler’s IP banned from the domain and in the worst case, crash the domain’s server. As detailed in the ethical part of the STEEPLE analysis, the crawler should respect the domain’s server and only request a certain number of pages per second. [18] recommends a threshold of at least 1 request per second. Some websites include this threshold in their /robot.txt file. For our purposes, we will take the maximum between those two values as bigger intervals put less weight on the servers.

2.1.4 Literature Review

Scraping a web page can be done in a multitude of ways. In this section, we review some of the methods we found in the literature.

Newspaper Python package  Newspaper is Python package that offers ready-made features for scraping articles [4]. It allows retrieving the title, body, author name and various other information through simple function calls. It is renowned to have good performance. However, we deemed that it would be better, from a pedagogical perspective and in accordance to the spirit of capstones, to build our own news scraper instead of relying on a black-box.

Parsing the DOM tree

The DOM  One of the main building blocks of an HTML is its tags. The Document Object Model (DOM) tree considers every tag as an object where its attributes are the HTML tag’s attribute. The id attribute, for example, uniquely identifies an tag object from all others. A class attribute binds a tag’s styling to an CSS class.

Each one of those objects is a node in a tree. A nested tag in the HTML is a children to the node representing the tag it’s enclosed by. To demonstrate this, figure 6 shows a
representation of a DOM tree from an HTML file. The DOM tree usually includes other metadata nodes that we have omitted for the sake of simplicity.

```
<div id="main-body">
  <p>This is the <b>first paragraph</b> of this document.</p>
  <p>Paragraph 2.</p>
  <p>Paragraph 3.</p>
</div>
```

![DOM tree example](image)

Figure 6: From HTML to DOM tree.

Trees are very convenient to represent relationships between entities. Extracting individual elements from this HTML file is much easier this way as we can run many tree algorithms, such as traversal.

**Querying the DOM** Scraping was earlier defined as extracting specific information from the page. As shown in [18], we can query the DOM tree to easily achieve that. One simple way we can get elements is by ID. `getElementById("main-body")` would return the root div in the tree modeled above. Searching depth-first on the left branch will yield the root node holding the text “This is the”. Note that recovering the text “first paragraph” will require us to search one depth further because of the boldening b tag. Therefore, hard-coding the depth at which to search for is not a good idea. Therefore, if we want to recollect the text contained by the div, we will have to traverse the tree inorder, and concatenate the text making up the leaves in the order we find them at.

It is clear that querying the DOM tree in this fashion requires some knowledge about
the website’s layout. For example, knowing the ID of the element to query is a crucial step towards scraping. This might be problematic in building the generic scraper we are aiming to build. In further sections, we describe how we adapted this solution to meet our needs.

**Using machine learning**  This approach, as described in [22], stems from the observation that humans find no problems extracting the information they need from a website visually, given that the latter is somewhat well-designed. The assumption is that related content is organized in lumps of text. Therefore, [22] devised an approach where the website’s HTML is rendered into the visual website it represents. Then, a clustering algorithm identifies blocks of content on the website. They then use an SVM classifier to get a probability that indicates the model’s certainty that the block it’s classifying is relevant to the content to extract. They then pick the cluster with the highest certainty value to represent as positive label.

This method’s strength is the fact that it is language-dependent. Furthermore, it can be easily made into a scraping software to be used by non-programmers. For example, users could highlight (label) a few occurrences of content they’re interested in scraping to build a data set for the model. This data set is then used to train the classifier to extract even more data. However, this model requires the scraper to render the whole HTML page which might require loading JavaScript and images as well. This makes it significantly slower. Furthermore, the method’s reliance on data makes it unusable in a context such as ours, where data on drone strike reporting is very scarce.
2.2 Classification

Classification is a task that involves assigning a label to an object. Binary classification is one that deals with assigning one of two possible labels to the object. Multi-class classification is one that chooses from more than two labels. They each usually require different techniques. A classifier is a model that can perform such a task. Must note that classifiers are not restricted to only machine learning models. Even a naive method like assigning one label to all occurrences can and should be considered a classifier, albeit a very useless one.

To build good classifiers, we usually need a labeled data set. That is, one that has been labeled by a human beforehand. The occurrences in that data set are considered as ground truth. Depending on the method we use, we might also need to split that data set into training and testing data sets. The training data set can be used to train supervised statistical models. The testing data set can be used to measure the model’s performance on unseen data. Whether we use machine learning or not, having a testing data set is always a way to validate a model.

2.2.1 Evaluating Classifiers

Once a model defined, it can be run against our testing data set. This produces object labeled by our classifier. Comparing the output of our classifier against the ground truth data of our labeled training set, we can compute a variety of metrics. Each example labeled by our classifier can be either a true positive (TP), false positive (FP), true negative (TN) or false negative (FN). A true positive is a positive example that was labeled as such. Conversely, a false positive is a negative example that was labeled as positive. A true negative (TN) is a negative example that was labeled as such. Conversely, a false negative is a positive example that was labeled as negative.

- **Accuracy**: Calculates the ratio of accurately labeled objects over the total number of objects.
Formula: \[ acc = \frac{TN + TP}{FN + TN + FP + FP} \]

- **Precision**: Calculates the ratio of accurately positively-labeled objects over the total number of positively-labeled objects by our classifier.

  Formula: \[ prec = \frac{TP}{TP + FP} \]

- **Recall**: Calculates the ratio of accurately positively-labeled objects over the total number of positively-labeled objects in the data set.

  Formula: \[ rec = \frac{TP}{TP + FN} \]

It is worth noting that contrary to popular view, accuracy on its own is not indicative of the model’s performance. It can be misleading in many situations. Imagine a data set that contains 80 negatively labeled data points and 20 positively labeled data points. A classifier that always classifies the occurrences it gets as being false would achieve 80% accuracy. This number hugely inflates the performance of a model that has no predictive power whatsoever. It is clear that accuracy didn’t reflect the imbalance present in our testing data set. The recall for such a model would give 0% and would be a good indicator of the model’s bias. This phenomenon, as described in [13], is called the accuracy paradox.

**K-fold cross-validation**  
K-fold cross-validation is a model evaluation technique. Instead of making one split of the training/testing set, k such splits are made. If the size of our labeled data set is N, the size of each testing set in those splits will be \( \frac{N}{K} \). The k testing sets that are generated should not overlap. This results in the training of k different models. We can then calculate aggregate metrics such as the mean accuracy, precision or recall across the k splits. Some methods even allow for “averaging” the models generated in the different splits.
2.2.2 Random Forests

Random forests is a supervised ensemble machine learning method. [14] defines ensemble methods as

Learning algorithms that construct a set of classifiers and then classify new data points by taking a weighted vote of their predictions.

In the case of random forests, this set of classifiers is made of decision trees. A decision tree gets an unlabeled data point at its root, and “asks” successive questions at each level of the tree. Depending on the answer, it either moves to the left or right sub-tree. A conclusion with regards to the data point’s label is made once a leaf is reached. Figure 7 shows a simple example of a decision tree.

The number of trees that the random forest contains as well as the depth of each tree (i.e. the number of questions asked) are hyper-parameters of the model.
Training  The questions asked at each level are not hard-coded. They are learned from the training data set. To understand how, let us first define information gain. Information gain is a value that measures how discriminative the question asked is. A question that splits our training data set into two “pure” sets (containing data points with the same label), the information gain is at its maximum. (1 in case of binary classification) The information gain we get by moving from parent node to children nodes is expressed in a formula as follows:

$$IG(parent, children) = E(parent, C) - E(children, C)$$

where $E$ is the entropy of the set $D$ given classes $C$, defined as

$$E(D, C) = \sum_{c \in C} -p_c \log_2(p_c)$$

where $p_c$ is the probability that a sample of class $c$ is drawn randomly from the set.

At each decision node, we pick the question that splits our data set so as to maximize the information gain, i.e. reducing the entropy of its children. If a set has an uniform distribution of classes, its entropy high. In the contrary, if the distribution of classes in a set is dominated by a single class, it will approach 0. We want our resulting classes to have as low entropy as possible.

Classifying unseen data  To classify a new sample $x_i$, the random forest will feed it to each one of its decision trees. Each of the trees will yield a positive or negative label. $x_i$ is labeled with the label for which most trees voted.

Advantages  Random forest presents many advantages. It is easily interpretable because of its intuitive visualization. However, the main strength of random forest lies in its robustness which comes from its built-in feature-selection mechanism. Each decision tree only operates on a randomly-chosen subset of the features given to the random forest. This
makes sure that prominent features do not always result in maximum split which lets us discover other less important, but still interesting features. Furthermore, even the data points we use to train each tree are a randomly-chosen subset of the original training data set. Because of this, we can calculate the out-of-bag error. It is the mean of prediction errors on a sample $x_i$ as predicted by trees that were not trained on $x_i$. This is very important because it allows us to use random forest without splitting our data set into training and testing, further enhancing our model's performance.
3 Requirements Specification

Drone-Watch should:

- Collect the title and body of news articles of sources that the user specifies.
- Filter collected articles to discard those that are irrelevant to drone strikes.
- Store the relevant articles in a database.
- Allow the user to easily extract location, date and casualty of the strike from the collected articles in the form of structured data through an intuitive user interface.
- Allow to easily add new sources even by non-programmers.
- Be modular and reusable to enable quick fixes in case a website’s layout changes.
- Respect the websites it scrapes by abiding to the /robot.txt rules of the website.
4 Design and Implementation

The data-flow diagram represented in figure 8 shows the main steps of Drone-Watch.

![Data-flow diagram of Drone-Watch.](image)

The next sections will dive into each part in more details.

4.1 Web Crawler and Scraper

A rule of thumb given in [18] is to implement a separate crawler/scraper for each website to be scraped as scrapers are usually short-lived and need to be refactored frequently. However, opting for this approach will make it harder for non-programmers to add new sources as it will require building a whole scraper from scratch. Therefore, we isolated the main logic of crawling and scraping into modular units described in figures 10 and 9 respectively, which can be used on any news source, given that the user defines an instance of the Website class described in figure 11.
4.1.1 The Scraper

As mentioned in the background section, a scraper takes an article web page as input and returns the title, body and publication date of the article in structured format. To achieve that, we parse the DOM tree of the web page’s HTML using CSS classes that we identify beforehand. The CSS classes each website assigns to the title, body and date published are consistent in all articles in the website. For example, one of the websites we scrape, PBS, always assigns the CSS class “body-text” to the div holding the articles content. Most article structures resembles the example shown in figure 6: a container tag, div for example, holding the paragraphs making the article. Again, the text of the article can be at any variable depth from the root container. Hence, we traverse the DOM tree in-order starting from the root container and concatenate the leaves content as we reach them. The final output is the article body reconstructed. We do the same for the title and date published. The latter needs to be parsed into a Python date object before being stored in the database.

Figure 9: Drone-Watch scraping logic.
Querying with XPath  Parsing HTML DOM trees in this fashion can be efficiently done using XPath. XPath is a querying language that was initially built to query XML files. However, it can also be used to parse HTML documents. Let’s analyze the following XPath query which Drone-Watch uses to retrieve articles body: ‘//*'[contains(@class, 'body-class')]/descendant-or-self::*[/text()]

The ‘//’ at the start positions the search at the root of the DOM tree. The ‘*’ that follows it specifies that we can move to any tag that fulfills the predicate inside the brackets: that its class attributes contains the string ‘body-class’. The value of the string used varies from website to website. In the example shown in figure 6, this would position us at the level of the main div container of the body. After that we apply the function ‘text()’ which extracts the text contains by an HTML tag to the main div container and its descendants. This give us all the text inside the main container. We then concatenate it in the order it was received to reconstruct the article body. We filter the HTML text we get from any non-rendered escape characters because our scraper has only access to unrendered HTML.
4.1.2 The Crawler

The seed URL The first design choice when it comes to crawling is deciding on seed URLs, that is, the web pages from which the crawler will start. We identified two different alternatives:

- Crawling from the homepage: the intuitive solution. It simulates a normal reader’s behavior of starting from the homepage.

- Crawling from the archive page: news websites have an archive page where all articles are listed in descending order of publication date. Our crawler would start from the first page (the latest articles), get all the articles listed and move to the next page and so on.
We ended up going for the second approach as the first one represented some downsides that might go against the purpose of Drone-Watch. The homepage of news websites contains a selection of articles which reporters make based on various criteria. Articles on drone strikes will most likely never appear on the homepage, and therefore never scraped if the first approach is used. Therefore, the approach of crawling the archive which contains all articles without any human-imposed hierarchy of importance is the one that guarantees the fairest crawling. The only downside is that we might encounter a source that does not have an archive page, something which we have not come across during the development of Drone-Watch.

After the crawler extracts all the links in a page, it will need to only keep those that point to articles in order to be fed to the scraper. To achieve this, we use regular expressions that characterize how article URLs should look like. This parameter is also one that the user should give to the Website object.

The crawling strategy  Starting from the first page in the archive, the crawler extracts all URLs in the page. This list of URL will also contain links to non-article pages within the website and perhaps even links to external websites. It is for this reason that we filter URLs using regular expressions to keep only those that point to articles. This could be something like “x.com/article/*” which would keep all strings starting with the prefix “x.com/article/”.

For every URL kept, we check if it has been already scraped before. It is only passed as input to the scraper if it hasn’t been. After feeding all the URLs to the scraper, the crawler moves to the next page of the archive to continue crawling and gathering URLs. Moving to the next page is straightforward in websites that use traditional pagination methods that involve sending a synchronous request to the server (i.e. loading a new URL). The paging URL usually have simple patterns such as “x.com/page/2”. In that case, the next URL to crawl would be “x.com/page/3”. Therefore, the user of Drone-Watch defines a function that takes the current URL as a string and returns the next one. This is the only programming
the user has to do when adding a new source.

Some websites with pagination that uses AJAX (Asynchronous JavaScript and XML) do not fit within this pattern. As explained in the background section, AJAX loads new content into the page without reloading the page. It makes an API call to the server which sends back data in structured format. Data is then injected into the HTML using JavaScript. Therefore, the only way to crawl these websites using the pattern we have identified is to fake the API call to the server which is difficult to do and varies vastly among websites. It would require an intermediate programmer to achieve that. Therefore, the only option is to rely on web drivers, which are interfaces that allow us to interact programatically with the browser. Hence, we can fake button clicks on the “Load more” button generating the API call programatically to load more articles into the view and extract the URL using the same technique we have described before. If the website requires this kind of crawling, the user must set the “button_id” attribute in the Website class which is the HTML id of the button.

Setting a termination condition is important so as to not crawl infinitely. In our crawler, the program stops once it reaches a page in which all articles have been seen before.
4.1.3 Adding News Sources

Figure 11: Website class holding information about the structure of websites.

```python
@dataclass
class Website():
    name: str
    homepage: str
    seed_urls: List[str]
    url_patterns: Match
    relative_url: bool
    title_class: str
    body_class: str
    date_class: str
    favicon: str
    next_button_id: str
    next_request: Callable[[HtmlResponse], Request]
```

For our logic to work, the operator needs to create an instance of our Website class, as shown in 11, with appropriate attribute values. Here is a rundown of what each attribute does:

- **name**: Name of the website, as it appears in logs and UIs.
- **homepage**: Link to the homepage of the website.
- **seed_urls**: List of seed URLs where our crawler will begin.
- **url_patterns**: Regular expression that describes the format of URLs that hold news articles. For example, “x.com/articles/*” will describe any link that starts with “x.com/articles/” followed by zero to many characters.
- **relative_url**: Boolean indicating whether the website uses relative URLs or absolute. If True, the crawler will need to prepend the website’s domain name to the URLs it finds.
- title_class, body_class, date_class: Strings holding the CSS class of the title, body and published date HTML container respectively.

- favicon: URL to the website's icon.

- next_button_id: HTML ID of the “Load more” button if browser-powered scraping is needed.

- next_request: Function that takes the current page’s URL and returns an HTML request to the next.

An instance of this class only needs to be set once per website. It can be easily modified if any change occurs on the website’s layout. The logic of scraping/crawling is never affected by that. In the 5 sources we added to Drone-Watch, it took us 15 minutes maximum to instantiate such a class for our sources.

To implement this, we have used a new Python feature called dataclasses introduced in [6] which acts as an equivalent to C-style structs and which is useful to implement classes that only hold data and have no behaviors. We also used a new feature in Python called type hinting introduced in [5] which adds readability to Python code. Furthermore, users can easily know what data type they need to pass to the constructor. It also helps detect some type-related errors a run-time.

4.1.4 Technology Enablers

- Scrapy: a web scraping and crawling framework in Python that has a built-in HTML parser, an XPath query engine and a request scheduler. The latter is what enables us to set values for time between requests and comply with the website’s /robot.txt specifications. Furthermore, Scrapy has built-in de-duplication of links: it never visits the same URL twice. Scrapy operates on non-rendered HTML which makes it faster, but also unable to simulate button clicks and other JavaScript actions. This is why we
coupled Scrapy with another scraping framework called Selenium in case asynchronous pagination is implemented.

- Selenium: as mentioned in the previous section, Selenium uses a web driver: an interface that allows to pragmatically control the browser to scrape JavaScript-rendered websites.
4.2 Classifier

To build our classifier, we labelled 200 articles: 78 positively labelled (i.e. reporting on drone strike) and 122 negatively labelled. Those 200 articles were extracted using the web crawler and scraper described in the previous section.

4.2.1 First Algorithm: Keyword-based Classification

As a first attempt, we built a classifier that labels articles based on the presence of keywords. The classifier takes as input a tuple of tuple of words in the form

\[ ((w_{11}, w_{12}, \ldots w_{1k}), (w_{21}, w_{22}, \ldots w_{2k}), (w_{31}, w_{32}, \ldots w_{3k}), \ldots) \]

and outputs True if the text we are classifying contains at least one word from every inner tuple.

We first tried it on the following list of words:

\[ (((\text{drone}, \text{uav}, \text{unmanned}), (\text{strike}, \text{missile})).] \]

This means that if an article contains the words “drone” and “strike”, it will be labeled as True. If it only contained the words “drone” and “uav”, it wouldn’t because it doesn’t contain any words from the second tuple.

Any subsequent metrics will be calculated from an artificially balanced testing set. We under-sampled a portion of the negative samples so that we get a balanced data set of 78 data point for each label. The negative samples we kept were randomly chosen from the labelled data set.
**First attempt**  Figure 12 shows the results of the first iteration of our keyword-based classifier. The accuracy score shows 71% which is decent for such a simple solution. The recall scores the maximum possible value with 100%, showing that the classifier correctly labeled all the positive samples. In other words, no articles on drone strikes were misclassified. However, 63% precision gives a red flag that there might be something wrong with our model.

![Scores of keyword-based classifier](image)

Figure 12: Plotting accuracy, recall and precision for first attempt.

**Further analysis**  To further explore our model’s results, we will draw the confusion matrix which displays the number of false and true positives as well as false and true negatives. Figure 13 shows the result.

As expected, we notice that the bottom-right cell shows that 100% of the positive labels in our data set were labeled as such. (true positives) However, we notice that only
31% of the negative labels were correctly labeled which leaves 69% of articles that are not drone strikes but that were still labeled as relevant. In this respect, this classifier classifies irrelevant articles worse than a random classifier that picks either class with a 50% chance.

**Interpretation and conclusion** The aforementioned results show that our model is very likely to classify any article it gets as being relevant. If an automatic data extractor is put in place, this might have the effect of over-inflating the numbers and thus reducing Drone-Watch credibility. Looking at some examples of false positives, we notice that our model positively classifies articles which succinctly mention drones at one point or another. This includes editorial pieces that talk about drone warfare but don’t report on any specific strike. Those should not be labeled as relevant.
Second attempt  The first solution we thought about is to include some measure of frequency in our classifier. However, looking at our labelled data set, we notice that articles which talk about drone strikes do not mention words like “drones” more than irrelevant articles. In fact, it was quite the opposite. Articles reporting on strikes were succinct and straight-to-the-point in nature. This observation made us think about only taking the first X characters of the article into account so as to advantage articles that mention drone strikes early on, which is indicative that they are relevant.

Hyper-parameter Search  We calculated accuracy for all models that look at the first X characters of the article for X ranging between 0 and 800 and increasing with increments of 40. Figure 14 shows the results.

![Figure 14: Accuracy vs. length considered.](image)

A length of 480 characters yields the best accuracy with 86%. Figure 15 shows it plotted alongside precision and recall, each of which gave 78% and 99%. Recall’s high value shows that our classifier still performs well in finding true positives. We also notice a noticeable
improvement in terms of both precision and accuracy compared to our previous results. Since 480 characters and 160 characters give approximately the same performance, we will favor lesser the number for simplicity of the model.

Figure 15: Plotting accuracy, recall and precision for second iteration.

Further analysis  This is confirmed upon looking at the confusion matrix in figure 16. To the contrary of the confusion matrix in figure 13, the number of true negatives is higher than the false positives. This means that only classifying based on a portion of the article did improve the performance of our classifier. However, at 27% false negatives, it still letting in a lot of irrelevant articles.

Interpretation and conclusions  Our model improved with the introduction of this constraint. However, it is still a burden on users of Drone-Watch who would have to go through all the samples that are labelled as true by the classifier. This portion will include 27% of articles that actually belong to a negative class. Those articles are, unfortunately the
most frequent: drone strike articles constitute only a small portion of the number of articles online. Furthermore, our model will not generalize well. If applied on a news site whose reporters adopt a less-direct writing style, it might see its performance decrease. In addition to that, as Drone-Watch runs and gathers more data, the model will not improve from the newly-seen data. In the following section, we try machine learning models and analyze their results.

![Normalized confusion matrix](image)

Figure 16: Confusion matrix for keyword-search, 2nd iteration.

### 4.2.2 Second Algorithm: Learning Models

In order to have a model that is more flexible than a hard-coded list of keywords, we will explore solving this classification task using machine learning models.

**Text representation** Although the title and body of the articles are stored in a machine-readable format, it is still not enough to feed it to a machine learning model. Models are not good with text data. We have to first convert them into numerical format.
**Word2Vec**  A popular representation of text which came out of Google AI is word2vec. [17] describes the process with which a vector representation of words can be learned from a corpus. This learning bases itself on words and its surrounding context. In word2vec representation, each word is represented with regards to the probability that others will occur around it. Therefore, if two words have different contextual probabilities, they probably mean different things semantically. This representation is powerful because it captures the semantic meaning of words while mapping words to a vector space which eases a lot of the maths. It is worth noting that generating the word2vec representation requires a large corpus to train. There are, however, pre-trained corpuses that are available online. [9]

**One-hot encoding**  Possible the simplest representation possible. One-hot encoding a corpus containing N unique words will result in a Nx1 matrix for each document, where each ith element of the vector represents whether the ith word in the corpus exists in the text or not [16]. The high dimensionality of this representation coupled with its sparsity affect models' performance negatively.

**Absolute term frequency**  Instead of storing boolean values in our vectors, we can store the frequency of a word in each document [16]. However, this tends to favor longer documents.

**Term-frequency (TF)**  To counter this, we divide the number of occurrences of a word by the size of the text it appears in so as to balance things out when it comes to large articles. However, this stills favor words that are often used in the English language such that ‘the’, ‘a’, ‘an’ etc..
Term-frequency, Inverse Document-Frequency (TF.IDF) To counter TF’s issue, we will multiply this value by in the inverse frequency of the term in the entire corpus yielding this formula:

\[ w(D_i, t_j) = \frac{f_{ij}}{D_i} \cdot -\log\left(\frac{n_j}{N}\right) \]

This counters the common words issue since this penalizes words that are repeated too often across documents. \( n_j \) corresponds to the number of documents that contains the term \( t_j \).

We opted for TF.IDF representation as it is quite simple to implement while still providing good results. The resulting representation was of dimension 1871x1.

Exploring the data Figure 17 shows a 2d projection of our 1871-dimension features. We notice that there is a clear separation between the positive and negative data points. The axes values do not hold any particular meaning as this is a projection of a higher-dimensional space.

![tf-idf feature vector for each article, projected on 2 dimensions.](image)

Figure 17: Scatter plot showing 2d projection of a sample of our training data.
Model selection  In this section, we select a model among the following: random forest, naive bayes, logistic regression. To pick a model, we run 8-fold cross validation on the whole data set and pick the model with the best accuracy. Figure 18 shows the results.

Figure 18 shows a box-plot of the validation results. For each model, the data points show the accuracy measures generated by each split of the 8-fold cross validation. The lines at the extremities of each box plot extend to the maximum and minimum value. The line in the middle denotes the median whereas the length of the box represents the upper and lower quartile. Random forest achieves the best accuracy with 94% accuracy whereas logistic regression comes narrowly behind with 93.5% accuracy. Naive Bayes comes last with 90.6% accuracy. The prior baseline and always true baseline models are dummy classifiers to give us a comparison measures for our models. The prior base line always classifies samples with the majority class in the data set (negative, in this instance). The always true baseline is one that always classifies samples as belonging to the positive class. Note that prior baseline
achieved a comparable accuracy to our first attempt with keyword-based classification.

Training a random forest classifier with 60 trees each with a depth of 5 yields the best accuracy.

**Interpretability** Random forest classifiers include a built-in mechanism to measure feature importance called mean Gini decrease. It is calculated by measuring the decrease in mean of accuracy when leaving out a variable during training. The more accuracy decreases when leaving out a variable, the more it is important to the model. [3]

Measuring the 10 most important features (i.e. words) for articles yields the following:

```
militants: 0.055  
post: 0.045  
drone strike: 0.045  
post views: 0.037  
drone: 0.030  
militants killed: 0.022  
killed: 0.021  
district: 0.021  
eastern: 0.020  
airstrike carried: 0.017
```

Figure 19: 10 most important words as shown by our initial random forest model.

“Militants”, “drone strike”, “drone”, “militants killed”, “killed” all make sense. However, words like “eastern” and “district” show bias in our training data set. Even more striking is the inclusion of “post views”. It turned out that most positively labeled articles came from khaama.com, an Afghan press agency. That particular website always includes the post views as part of its article body. Therefore, we clean the data from post views and present the new results in figures 20, 21 and 22.
Figure 20: 10 most important words as shown by our 2nd random forest model.

militants killed: 0.048
drone: 0.040
province: 0.039
militants: 0.039
drone strike: 0.031
carried: 0.022
afghanistan: 0.019
isis: 0.015
taliban: 0.015
regarding report: 0.015
Random forest model after cleaning up the data  Removing occurrences “post views” from our articles did not have much effect on our classification accuracy as is shown in figure 21. However it did give more meaningful important features in figure 20.

Figure 21: Cross validation result for 2nd model.

Figure 22: Confusion matrix for 2nd random forest model.
Figure 22 shows substantial improvement from figure 16. The ratio of true negatives and true positives are both high with 96% and 100% respectively.

4.2.3 Technology Enablers

The whole classification program has been written in Python. Scikit-learn has been used to train the models, run cross validation and calculate classification metrics. Pandas and NumPy have been used to manipulate the data. Seaborn and matplotlib have been used for data visualization. The code for the various classifiers is hosted in the following colab links:

- ML classification.
- Keyword-based classification.

4.2.4 Conclusion

The final results shown by our machine learning model are satisfactory. It will be interesting to see how it generalizes on real data. One of the advantages of using machine learning models is that they can be adapted as more data is collected. One approach that we can follow is what is called active learning. [19] defines it as a machine learning technique where the model queries the user to label particular data points. Instead of letting the user label hundreds of data points which the classifier can easily label (i.e. with high certainty), the classifier will query the user to labeler specific data points in which it has low certainty. Fortunately, random forest is able to give a probabilistic measure of certainty that can be calculated as the ratio of trees that chose the majority label. It has been shown in [19] that a lot of empirical studies demonstrated the effectiveness of active learning in building stronger classifiers with less data.
4.3 Information Extraction

The filtered articles will then be available in our labeler interface. It’s a web app implemented using Django for the back-end and HTML/CSS/JavaScript for the front-end that allows users of Drone-Watch to quickly label data into a structured format. To do so, we convert some labeled text to a standard format. For example, if the word “Wednesday” is highlighted by the user as date of the strike, our labeler looks at the date of publication of the article as it was scraped in earlier phases and saves the date to be the previous Wednesday from that. The same kind of inference is done for relative dates like “yesterday”. The labeler saves the labels in a SQLite database. Figure 23 shows an image of the labeler’s UI.

![Labeler UI](image)

**Figure 23:** UI of the labeler.

Labels are stored as database tables. They refer to the article using a foreign key. The classification and labeling code both have access to the same database model. All interactions with the database have been done through a Python interface. Therefore, the DBMS can be changed at any time. At this point we are using SQLite because it is light-weight and portable. The labeler saves entities’ content but also start and ending index. This is done for easy access to the entities surrounding words if users want to further analyze labeled data. Communication between JavaScript and Django is done via JSON.
5 STEEPLE Analysis

The STEEPLE analysis is a widely-used tool to analyze external societal, technical, economic, environmental, political, legal and ethical factors that advantage or limit the proposed product [7]. The following is the STEEPLE analysis for Drone-Watch.

5.1 Societal

Drone warfare is slowly becoming a worrisome issue for many human rights organizations. They have managed to raise awareness on this problem through many communication campaigns. However, these campaigns are still limited in reach. Drone-Watch will aim to raise drone warfare in the hierarchy of social concern.

Furthermore, this new model of automatic news retrieval and filtering is changing the way society consumes news. Individually-tailored content promises to appeal to many readers because of its convenience.

5.2 Political

Trump’s recent executive order stripping CIA from the responsibility of reporting civilian casualties from drone strikes [15] represents a real danger to the transparency in using that technology. Thus, a software like Drone-Watch comes at a convenient time to help collect that data from newspaper articles. Therefore, it will put pressure on political entities by bringing to this warfare the transparency it requires.

5.3 Technical

This project comes at a time when web scraping reached unprecedented popularity. Organizations now use it to achieve multitude of goals which range from measuring customer satisfaction by analyzing comments in discussion forums to collecting data from archives for
social sciences. These efforts come in the context of a large effort to build a semantic web. Semantic web, as defined by Tim Berners-Lee et al., is “an extension of the current web in which information is given a well-defined meaning, better enabling computers and people to work in cooperation” [12]. A consequence of that is the availability of many well-documented tools for web scraping as well as a striving community online which facilitates development.

5.4 Ethical

Any project that involves automation is bound to have ethical implications as it usually comes at the expense of human workers who might be laid-off. However, our project’s aim is not to completely replace humans. Its main objective is to assist humans in the very specific task of gathering data on drone airstrikes in the extraction and filtering of articles. Human intervention in labeling and monitoring the quality of data generated is still needed at the moment. Furthermore, this project is to be used in non-profit organizations who might not be able to afford specialized workers who will work on day on gathering articles on strikes.

5.5 Legal

Web scraping, like many other recent techniques in technology, is still considered a grey area legally. Web scraping, by definition, is the automatic collection of publicly available data. Thus, many organizations use it without thinking about the legal implications it might have as it only concerns publicly available data. However, the content making up websites is considered copyrighted and subject to the terms of service of the website. Using such content in opposition with terms of service of the scraped website is, in fact, illegal [11]. There have been precedents of legal cases against organizations or individuals using crawler bots. [8] is such an example, in which LinkedIn sued more than 100 individuals scraping their website.

To regulate this, websites include a text file called robot.txt in the top directory of the page. (e.g. nytimes.com/robot.txt) This file contains a list of URLs that should not be
scraped by robots as well as robot names that are not allowed access. Drone-Watch scraper respects the will of websites and abides by the rules set in the /robot.txt files.

5.6 Economic

Drone-Watch is an open-source and non-profit software. Its code is publicly available at github.com/drone-watch. Its aim is not to advertise any particular newspaper. Its sole interest is bringing transparency to the issue of drone warfare.

5.7 Environmental

Drone-Watch is not affected by any environmental factors. The servers required to run the code need only to run in short bursts and at periodic intervals. Therefore, it might result in less energetic consumption, if anything, as it represents a reduction in time spent collecting articles by a human operator.
6 Final Remarks

6.1 Challenges and Limitations

The main challenge we faced during the development of Drone Watch was the lack of pre-existing data. The classification and information extraction face rely a lot on the existence of data. Therefore, we had to spend more time than expected on the first phase to scrape enough data to build a classifier. Especially that scraping was new territory to us.

We had initially intended to implement automatic extraction of information from the article but were not able to do that because of time constraints. We have also realized that the problem of automatic information retrieval from documents is a very complex one that might require a separate project solely dedicated to it.

6.2 Future Work

Future work will focus on building the automatic information retriever as this would automate all components of Drone Watch. Drone Watch would also benefit from a friendly user interface that would enable users to have access to its features. Most of the components of Drone Watch, except for the labeler, require running scripts or accessing the database.

6.3 Conclusion

Drone Watch’s initial intent was to fully automatize the process of drone-strike data extraction to display aggregate statistics to end-users. As the information retrieval part was not automatized, we have not managed to do that. However, Drone Watch can still server as a valuable tool to intermediate human data providers such as journalists working at the Bureau of Investigative Journalism who have to manually sift through a lot of articles and then extract the data manually into a Google spreadsheet. Drone Watch would reduce a lot
of time and efforts in the initial article-gathering phase using the crawler/scraper. In that respect, Drone Watch can scrape from a multitude of sources thus making a useful tool to track drone strikes at many fronts at the same time. This, coupled with the classifier will reduce the amount of article a human operator would need to look at.

To conclude, this has been a unique learning opportunity. It was my first time working on a project for such an extended period of time. It certainly requires a different skill set of patience, dedication and long-term vision, which I am eager to further develop in the future. Any contributions to Drone-Watch are welcome at https://github.com/ayoubkachkach/drone-watch.
References


