

Machine Learning Approaches for Event Web Data Extraction



Julia Wiedmann
Kellogg College
University of Oxford

A thesis submitted for the degree of
Doctor of Philosophy in Computer Science

Michaelmas 2018

To my mother.

“Und so wie’s wird, ist’s gut.”

Acknowledgements

First and foremost, I would like to thank my supervisors, Professor Georg Gottlob and Dr. Tim Furche, for giving me the chance to pursue a doctorate in the field I wanted to learn so passionately. Moving from an undergraduate in Business to Computer Science was a difficult transition, and I appreciate that my supervisors gave me the chance to pursue this doctorate with such an unusual background. I must admit this transition was one of the most challenging times of my life, yet the best decision I could have made.

I would like to thank Dr. Emanuel Sallinger for his support as the VADA lab director. I am grateful to VADA postdocs Dr. Ruslan Fayzrakhmanov, Dr. Evgeny Sherkonov, Dr. Yavor Nenov, and Dr. Stéphane Reissenfelder and DPhil colleagues Dr. Stefano Ortona, Dr. Jinsong Guo, Alina Petrova and Lianlong Wu for their companionship. Special thanks to Ruslan for helping me setup and wrangle through the Amazon clusters and machines for Chapter 4.

A big, heartfelt thank you goes out to my colleagues at Diffbot, for being such supportive, though-provoking and interesting collaborators. A very special thank you to Mike Tung, for reaching out to me, bringing me on board and supporting me through the last six months of this journey. I would also like to thank Dru Wynings, for the encouragement, support and ridiculous jokes. I was very fortunate to have the chance to work with such humble people and I look forward to future projects.

On a more personal note, I owe eternal gratitude to my parents. My mother did everything to make me feel safe and loved, no matter what sacrifices she had to make. For that I am forever in her debt. I will always be grateful for my fathers support and encouragement, teaching me discipline and determination. Special thanks go out to my friends here in Oxford, especially Sarah Griffin, Florian Klimm and Tammo Rukat, for the countless hours spent together, making these four years in Oxford probably the best time of my life. I would also like to mention my friends Léa Louvat and Christina Bartosch for supporting me from afar.

Lastly and most importantly, thank you Max. I would not be here today if it wasn't for your support, love, and patience. I look forward to a lifetime of adventures with you.

Julia Wiedmann

Oxford, January 2019

Abstract

Web data is so ubiquitous and yet it remains so hard to get a quick answer to a simple question like “What’s the closest pub with a pub quizz event going on tonight?” or “What meetups on data science are taking place this week in London?”. Event search engines like Yahoo’s Upcoming tried to address this challenge but failed at the long-tail problem, only integrating with few large-scale ticketing and event announcement sites. Thus, automatically extracting upcoming events (or more precisely “event announcements”) from websites including the long-tail of local pubs, reading groups, or neighbourhood associations remains a crucial, but largely unsolved problem.

Most general web data extraction approaches rely on the template hypothesis, that states that websites contain many pages that are generated from a background database using the same template. Thus these approaches try to re-discover the structure of the template and often induce a wrapper (or “scraper”) automatically that encodes the learned patterns about the template. Unfortunately, in the event (announcement) domain the template hypothesis does not hold: Many websites only have a few active events at any moment in time and delete event announcements after the event has taken place. For such a small number of active events, often not even a consistent template is used and where it is used often the number of samples is small enough to challenge most existing web data extraction techniques.

Therefore, in this thesis we focus on the problem of *template-independent* event data extraction, where not only no per-site supervision is necessary but also no restrictions on the template structure of the sites is imposed. Rather, the approach investigates each *event page* — that is a page dedicated to the announcement of a particular event — in isolation. This also allows the approach to scale out massively.

More specifically, we introduce two complementary, but quite different approaches to event extraction: TIDE leverages existing semantic annotations following the `schema.org` vocabulary to bootstrap a training corpus. Though such annotations are only available on a small subset of event pages, they are sufficiently common to serve as a rich and diverse training set. Intensive cleaning and filtering of those annotations is unfortunately necessary to eliminate the large amount of noisy, incorrect, or abused annotations present on the web. After this filtering step, the training data is used to first classify individual web elements as potential event attributes and then assemble multiple candidates into likely events, leveraging unique features interrelating the

candidates for individual event attributes, such as alignment or position on the page. The latter step is necessary as event pages often contain additional event attributes from other events, shown as “related events” or other such ancillary information. Based on an extensive empirical evaluation, TIDE extracts such fields with 93% accuracy.

Unfortunately, TIDE is limited by the availability of decent semantic annotations and those are only available for few, coarse granular event attributes, namely title, date, and location. Therefore, we introduce a second approach, TIME, that uses more sophisticated feature engineering to reduce the needed training data and thus can afford the manual creation of training data for a larger and more fine granular set of attributes. We thus refer to it as multi-attribute event extraction. As the first approach, it first identifies candidates for each attribute separately, but this time with features configured for the specific attribute, e.g., through a set of gazetteers for related labels or regular expressions for detecting instance patterns, such as a date. Our empirical evaluation demonstrates that the unique characteristics of the events domain can be leveraged by encoding them into machine readable features to achieve high accuracy without massive amount of training data. We report extraction results in the range of 82-95% across a variety of attributes. This approach has been implemented at the commercial data extraction company Diffbot and is currently available as beta. It is already being used to add event information to their Knowledge Graph and is planned to be released as a public API in the next few months.

Contents

1	Introduction	1
2	Preliminaries and Problem Definition	8
2.1	Preliminaries	8
2.1.1	The Structure of Websites	8
2.1.2	Tree-Based Machine Learning Models	9
2.1.2.1	Decision Tree Learning	10
2.1.2.2	Random Forest	12
2.2	Problem Definition	13
3	Related Work	16
3.1	Web Data Extraction	16
3.1.1	Web Wrappers	17
3.1.1.1	Web Wrapper Languages	17
3.1.1.2	Web Wrapper Induction	18
3.1.2	Template-Independent Techniques: Machine Learning and Vision-Based Systems for Data Extraction	20
3.2	Event Extraction	22
3.2.1	Event Extraction from Editorial News Text	23
3.2.1.1	Data-driven Event Extraction	23
3.2.1.2	Knowledge-based Event Extraction	24
3.2.1.3	Hybrid Event Extraction	24
3.2.2	Event Extraction from Social Media	25
3.3	Event Extraction from Single-Entity Pages	26
4	Bootstrapping an Event Corpus from Noisy Schema.org Annotations	27
4.1	Preliminaries	28
4.1.1	Common Crawl	28
4.1.2	Semantic Annotations on the Web	28

4.2	Approach	29
4.3	Results	31
4.3.1	Domain Understanding of Events on the Web	31
4.3.1.1	Distribution of Pages	32
4.3.1.2	Distribution of Annotations	32
4.3.1.3	Distribution of Relevant Event Fields	36
4.3.2	The Resulting Dataset	39
5	TIDE: Distantly-Supervised Event Extraction	41
5.1	A Template-Independent Approach	44
5.2	Element Level Classification	45
5.2.1	Textual Features	45
5.2.2	Structural Features	46
5.2.3	Visual Features	47
5.2.3.1	Attribute Level Classification.	48
5.3	Candidate Selection for Event Announcements	48
5.4	Event Assembly	49
5.5	Implementation	52
5.6	Experimental Evaluation	52
5.6.1	Attribute Level Accuracy	53
5.6.1.1	Error Classification	54
5.6.2	Event Assembly Accuracy	55
5.6.2.1	Threshold Evaluation	56
5.6.2.2	Number of Negative Examples	57
6	TIME: Template-Independent, Multi-Attribute Extraction for Events	59
6.1	System Architecture	60
6.1.1	General Features	60
6.1.1.1	Textual: Character Statistics	60
6.1.1.2	Textual: Token Statistics	61
6.1.1.3	Structural: Basic Tree Statistics	61
6.1.1.4	Structural: Parent Statistics	62
6.1.1.5	Visual: Basic Rendering	62
6.1.1.6	Visual: Font and Style	63
6.2	Event Title Extraction	63
6.2.1	Candidate Selection for Title Extraction	63
6.2.2	Title Specific Features	64

6.2.2.1	General Features for the Title Attribute	65
6.2.2.2	HTML Title Specific Features	65
6.2.2.3	Dictionary-based Features	65
6.2.2.4	Ancestry-Based Features	66
6.2.3	Title Classification and Ranking	66
6.3	Event Date and Time Extraction	66
6.3.1	Our Approach	68
6.3.2	Date-Time Candidate Generation	70
6.3.3	Classifying <date,time> Pairs	70
6.3.3.1	Textual Date and Time Features	71
6.3.3.2	Structural Date and Time Features	72
6.3.3.3	Visual Date and Time Features	72
6.3.4	Classification of <date, time> Pairs	73
6.3.5	Generation of <start,end> Pairs	73
6.3.5.1	Textual Start and End Date Features	74
6.3.5.2	Structural start and end date features	75
6.3.5.3	Visual Start and End Date Features	75
6.3.6	Overall Classification of Temporal Attributes	76
6.4	Event Location Extraction	76
6.4.1	Location Specific Features	77
6.4.1.1	Textual Location Features	77
6.4.1.2	Structural Location Features	77
6.4.1.3	Visual Location Features	78
6.4.2	Location Classification	78
6.5	Event Description Extraction	79
6.5.1	Description Candidate Generation	79
6.5.2	Description Specific Features	79
6.5.2.1	Textual Description Features	79
6.5.2.2	Structural Description Features	80
6.5.2.3	Visual Description Features	80
6.5.3	Description Classification	81
6.6	Experimental Evaluation	81
6.6.1	Dataset	81
6.6.1.1	Distribution of Webpages on Websites	82
6.6.1.2	Website Type Distribution	82
6.6.1.3	Language Distribution	82

6.6.2	Overall Performance	82
6.6.2.1	Temporal Attribute Assembly	82
6.6.2.2	Languages	83
6.6.2.3	Comparing Aggregator Sites with the Long Tail	84
6.6.3	Feature Ablation Study	85
6.6.3.1	Using the Same Features for All Event Attributes	85
6.6.3.2	Multilingual features	85
6.6.3.3	Textual Features	86
6.6.3.4	Structural Features	87
6.6.3.5	Visual Features	87
6.6.3.6	Experimental Summary	88
6.7	Industrial Application of TIME: The Diffbot Knowledge Graph	90
7	Conclusion and Future Work	93
7.1	Future Work	94
	Bibliography	96

Chapter 1

Introduction

Data is spread all over the web, with anybody able to contribute. The web is the major platform for information propagation, news outlets, online sales and advertisements, where users have to sift through an overwhelming amount of data to find answers to their queries. Search engines make a great effort to structure and index this information, however traditional search does not provide readily available structured *answers* to a users questions. Rather a ranked list of relevant documents is returned instead and the user needs to further investigate these documents. Imagine the query “When is the next comedy night in Oxford?”. A user does not want to sift through many different web pages, identifying comedy nights and comparing their dates. When a user asks a question, they expect a concise response.

To address such questions asking for concise answers, search engines have shifted to using knowledge graphs [42] and question answering [73], allowing users to “*search for things, not for strings*”¹. The idea is for users to search for places, people and organisations, rather than just matching strings in a set of documents. That way, when a user queries keywords or asks a question, they receive a structured answer as a response, ideally the entity or piece of information they are looking for.

In this work, we focus on knowledge graphs as a form of structure information about entities in a specific domain. A knowledge graph contains *entities* (e.g., the entity representing the city Vienna or an entity representing the 2019 installation of the Jazz Fest) together with their *attributes* (e.g., that the Jazz Fest 2019 runs from June 24th-July 10th, 2019) and *relations* between the entities (e.g., that the Jazz Fest is located in Vienna). Each entity (or “node”) represents an object in the domain of interest and the relations (or “edges”) between such entities model the relationships between the two entities it connects, while attributes further characterise the entities. For example, Figure 1.1 depicts an entity of type event that represents the Jazz Fest 2019 and is related by a “located in” relation to the entity representing the city Vienna.

Initially, many knowledge graphs have been largely concerned themselves with common-sense knowledge that is fairly static (e.g., that Vienna is the capital of Austria). They also include some temporal information typically in the form of validity intervals, e.g., that Obama has been president of the US from 2009-2017.

¹<https://www.blog.google/products/search/introducing-knowledge-graph-things-not/>

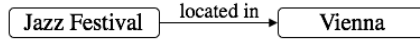


Figure 1.1: Example: Relation structure in knowledge graphs.

Event announcements. For this thesis, we focus on a knowledge graph about online **event announcements**, where events are announced (often well in advance of the actual event taking place) such that users can determine which events to participate in. This is a particularly time-sensitive domain, with numerous updates to the information available online and often also with time-restricted validity. E.g., an event announcement from a week ago may no longer reflect the current facts if the venue has been moved or the price of attendance been reduced. We are focused on event announcements on the web, as they appear, e.g., on ticketing websites, on event venue websites, on schedules of a University course, or on a pub’s website. This is a very different problem from detecting and extracting events that are ongoing or have already ended, whether from articles, social media, or other texts, collectively referred to as **event detection** in this thesis. While event detection has been investigated extensively (see Section 3), the extraction of event announcements for the use in personal assistants or other planning purposes has received much less attention. Where it has been investigated, the focus has been on how to deal with updates and the fast changing nature of the information attached to event announcements on the web [51].

However, previous approaches including [51] have largely been using supervised methods where for each website (or “source”) such as a ticketing website or a local’s pub event listings a separate wrapper is constructed manually or in a semi-supervised fashion. Such approaches are feasible, where the target is the extraction of events from a few large ticketing and event announcement websites, but not realistic for the long tail of small event listings spread all over the web. They are particularly unattractive in the event announcement domain thanks to the local and therefore highly distributed nature of event announcements — I care about my local pub’s events, but not about the one 200 km away, while a book shop 200km away may be perfectly serviceable if they deliver. Furthermore, event listings in the long tail are often managed by hand and thus follow only vaguely a common template even within the same website. That further complicates traditional wrapper induction approaches and even approaches such as [49] or [32] that require no supervision for each individual website, but still assume that a common template is used within each site.

Event announcements are largely organised similar to actual events, in that they carry typical event attributes *title*, *location*, *date* (often a highly structured date and time range), and a *description*. They may in addition have information about *organisers*, *announcement date*, *speakers/performers*, and be related to a *event series*. In this thesis, we focus on the typical event attributes, but point out where the presented methods could be easily extended to cover more attributes. We also focus on the extraction of raw values for each of these attributes and leave the linking of, e.g., the location information to spatial coordinates or known location entities to standard record linking approaches.

Event announcement attribute extraction. Thus we focus on **event announcement attribute extraction** from (single-entity) **event announcement pages**. Event (announcement) pages are web pages dedicated to announcing or describing a specific event, though often with additional related events or other information displayed in sidebars, footers, or other ancillary locations on the page. We focus on such pages, in contrast to search engine result pages (SERPs) or other listings of events, as they contain typically more information. We also observe that in the event announcement domain in particular long-tail websites often contain no search for events specifically and often only a list of links to single-entity event pages with no more than the abbreviated event title as link anchor. Such listings are thus largely useless for extraction. By focusing on event pages, we assume that a crawler (such as Gigablast ² or Common Crawl ³) discovers the targeted pages. We further assume that we can distinguish event announcement pages reasonably well from other pages through a standard web page classifier (see details in Section 6).

Phenomenology of event announcement pages. Figure 1.2 shows the main content area of an event announcement page (from Eventbrite, <https://www.eventbrite.com>). We observe a number of characteristics typical for the event announcement pages, that we capture in this thesis as the *phenomenology* of event announcement pages and use to inform our machine learning approaches through appropriate feature engineering or pre- and postprocessing.

Event announcement pages are typically split into structured event information, commonly the core event attributes title, date, and location, and a separate, longer textual description of the event. There is also typically a fair amount of boilerplate for subscription, sharing, maps, and other functional elements of the web site. While that boilerplate often follows a common template for a single website, unfortunately there are also cases where it does not, not even within the same website. Nevertheless, we can observe certain common patterns in how event announcement pages are styled and organised: The event title is usually prominently highlighted, with the attributes *date* and *location* in relative proximity and visual alignment. As seen in Figure 1.2, the title is in a bigger and bolder font, than the rest of the page, and the date and location of the event are in clear vertical alignment.

We further observe that **date information** tends to be more complex than for simple publishing dates in articles or other timestamp-like information. For event announcements, we not only have to deal with the difference between the event date and the announcement date, we also observe that event dates are often not actually specific points in time, but ranges specified in a variety of formats with any combination of *start date* and *time*, *end date* and *time*, including combinations like a *start date* and a time range as in Figure 1.2.

Location information is often a simple address, but very often also highly context dependent, e.g., just a *venue name* (that needs to be disambiguated) or a *room name* that needs to be placed within a relevant building.

²<https://www.gigablast.com>

³<http://commoncrawl.org>

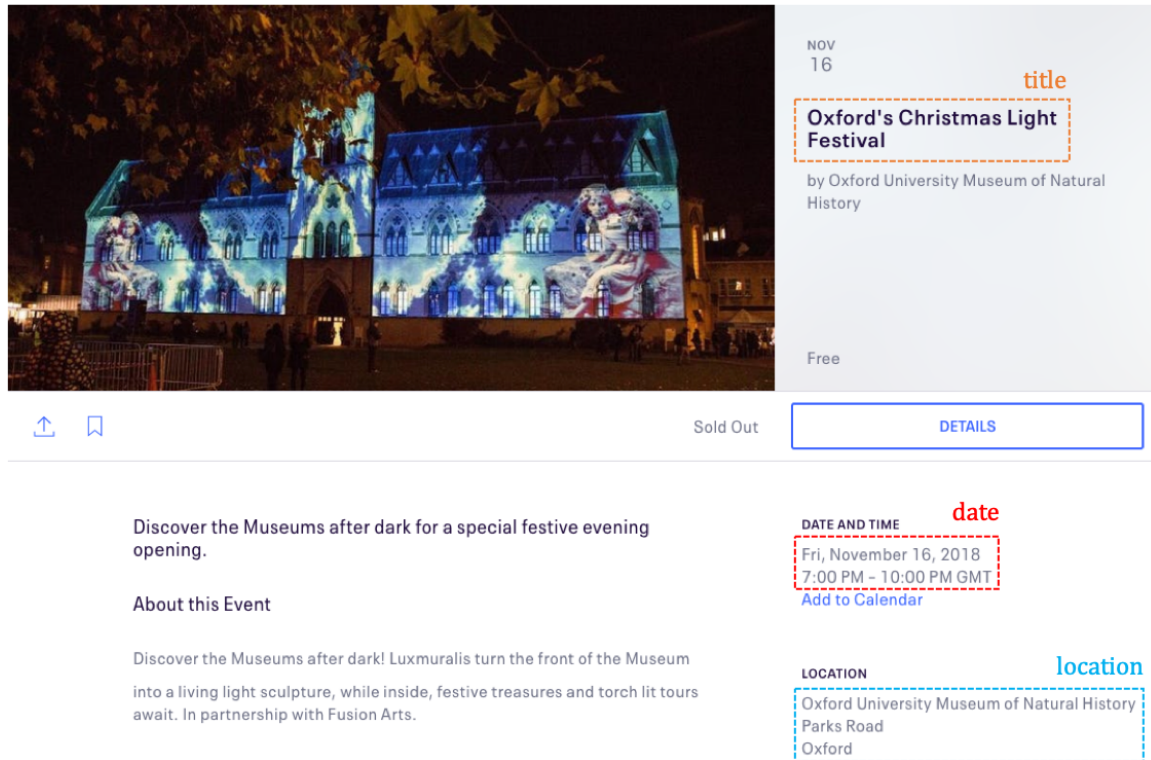


Figure 1.2: Main content area of event announcement page on Eventbrite

As in many other extraction tasks, event announcement pages tend to contain a lot of **noise**, including often listing of other events, as shown in Figure 1.3 which depicts the bottom quadrant of the Eventbrite event announcement page from Figure 1.2 including a list of related events from the same organizer.

Limitations of template-based web data extraction. Event announcement extraction identifies and extracts the event announcement attributes from an event announcement page. This can be seen as an instance of the general web data extraction problem. Traditional web data extraction techniques have made significant progress in extracting data from large websites where the underlying structure can be exploited [49, 37]. However, most approaches for web data extraction have assumed a fundamental assumption, namely that pages on a single site are generate according to one or few templates such that there are many example pages for each template available. This assumption is then used to create so called **wrappers**, which express access patterns in the webpage's HTML to extract the relevant information.

Such wrappers can be used within a single website with the same template, however for every unseen template, a new wrapper has to be generated either manually or automatically. Template-based methods have therefore two main shortcomings making them unsuitable for the events domain:

- (1) Per-site supervision is high: a new wrapper has to be created for every unseen template, as mosts web-sites have a different template structure. While useful with large websites that use the same template

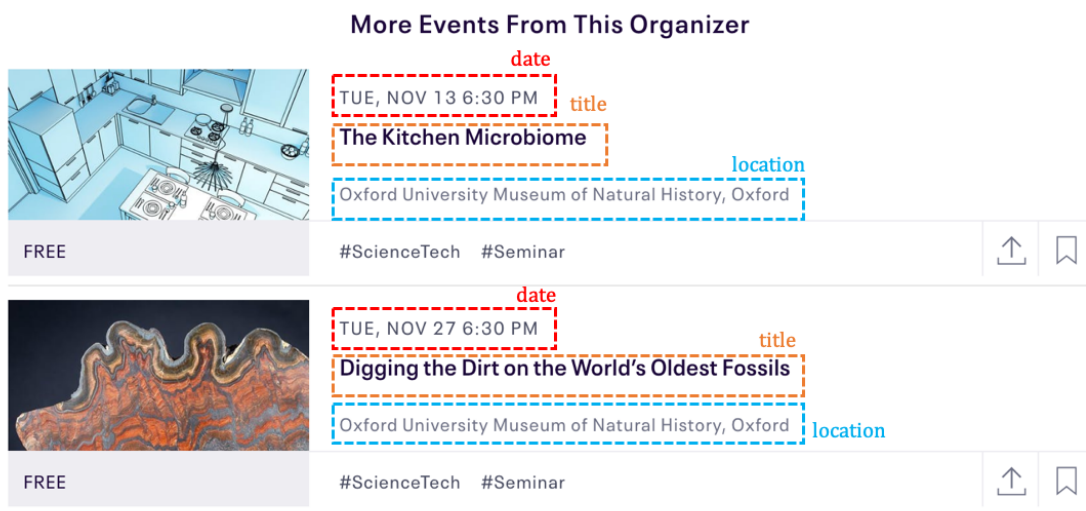


Figure 1.3: More events block on Figure 1.2

for many pages, template-based wrappers are not well suited for the small, long-tail websites, where the page per-site ratio is too small to justify any site level supervision.

- (2) Even automatic wrapper induction systems such as WEIR [32] or DIADEM [49] rely on multiple examples for the same template which often does not hold in the event announcement domain, either because of manually created pages or because of the general sparsity of event announcements in a site, in particular as old event announcements tend to be deleted rather than archived.
- (3) The high turn over of event pages where only a small window of event announcement pages is active at any time also makes wrapper induction particularly brittle: If at a certain point only pages without an end time are active, the induced wrapper is not going to be able to extract end times if later pages become active that contain such a time. Wrapper maintenance techniques can somewhat alleviate this issue, but generally are not suited to deal with small, ever changing example sets either.

Limitations of microdata annotations. Event announcement data has been identified as interesting data for search engines for a long time, even though event aggregators such as Yahoo’s *Upcoming* generally did not succeed. As a result, search engine providers including Google, Bing and Yahoo! launched the **schema.org** initiative to standardise the vocabulary of annotations that web masters can use to provide structured summaries of their web pages to a search engine. This vocabulary contains an event announcement ontology and can be used for annotating websites with HTML 5 Microdata, RDFa, JSON-LD, and other semantic annotation formats as seen in Figure 1.4. The `item-`prefixed attributes highlighted in red are Microdata annotations that can be used by search engines to extract information from webpages with more ease. Website owners are incentivised to provide such information as it boots search engine ranking and is convenient for the user.

```

1 <div itemscope itemtype="http://schema.org/Event">
2 <a itemprop="url" href="nba-miami-philidelphia-game3.html">
  NBA Eastern Conference First Round Playoff Tickets:
4 <span itemprop="name"> Miami Heat at Philadelphia 76ers - Game 3 (Home Game 1) </span>
  </a>
6 <meta itemprop="startDate" content="2016-04-21T20:00">
  Thu, 04/21/16
8   8:00 p.m.
  <div itemprop="location" itemscope itemtype="http://schema.org/Place">
10    <a itemprop="url" href="wells-fargo-center.html">
      Wells Fargo Center </a>
12    <div itemprop="address" itemscope itemtype="http://schema.org/PostalAddress">
        <span itemprop="addressLocality">Philadelphia</span>,
14        <span itemprop="addressRegion">PA</span>
      </div>
16    </div>
      <div itemprop="offers" itemscope itemtype="http://schema.org/AggregateOffer">
18        Priced from: <span itemprop="lowPrice">35 USD</span>
          <span itemprop="offerCount">1938</span> tickets left
20      </div>
    </div>

```

Figure 1.4: HTML example with schema.org annotations (schema.org/Event).

Unfortunately, only a small percentage of the web is actually annotated with such annotations and within those schema.org annotated websites, a large proportion are using the annotations in an erroneous way, whether with the intent of misuse or unintentional. Misuse is somewhat exacerbated due to the influence Microdata annotations have on the ranking of a page in a search engine and to the click through rate. Typical misuse is, e.g., marking up wrong price information omitting additional fees or taxes that make the offer much less attractive. Furthermore, owners of smaller websites might not have the required expertise in order to correctly annotate their web pages. We therefore cannot expect semantic annotations to provide us with a even remotely complete picture of event announcements on the web, yet we show that even the limited annotations that are available can be exploited to bootstrap a corpus of annotated pages that allows us to learn classifiers for core event announcement attributes.

Template-independent event announcement extraction. To conclude, we believe and show that none of the existing techniques or annotation schemes is sufficient to provide a high coverage, high precision extraction of event announcements and their attributes. We therefore propose in this thesis the **first fully automated, template-independent extraction approaches**: TIDE and TIME. The main novelty of our approaches is that they are not relying on identifying template information in a site at all, but rather extract from each page independently. Rather, the first approach learns common patterns from the small, but still significant subset of pages already annotated with semantic annotations using the `schema.org` event vocabulary. The second approach shows how to generalise that to cases where `schema.org` annotations are not available in sufficient numbers, without requiring massive amounts of training data. In particular, the second approach makes extensive use of our analysis of the phenomenology of the events announcement domain to guide feature extraction and candidate generation for classifying the individual event announcement attributes. Here,

visual cues play an important role, as the event information is usually displayed in a way that is optimised for humans to detect.

Contribution In this thesis, we therefore introduce two event announcement extraction approaches that are independent from the underlying web page template and therefore require **no per-site supervision**.

- (1) We bootstrap our methods for the core attributes of an event announcement by extracting semantic annotations for `schema.org` events from the Common Crawl corpus. Extensive filtering and validation of the training samples is necessary to remove noise and abusive semantic annotations. This data set is made available online and hopefully will serve as a much needed dataset for benchmarking in the event extraction community as a whole. We also plan to publish the scripts to rebuild the dataset from a new Common Crawl dump, thus enabling continuous refreshes of the annotation set.
- (2) We then introduce TIDE, the template-independent data extraction for event announcement: Without any further supervision and using a simple set of common features, TIDE successfully learns to recognise event announcement attributes from single entity pages for a huge set of pages. It combines visual, textual, and (HTML) structural features for the classification of individual event attributes, yet also learns how those attributes should be correlated to form a coherent event. The latter is critical as many event pages contain additional noise, as shown in Figure 1.3.
- (3) Finally, we introduce TIME, the template-independent, multi-attribute event announcement extraction approach. In a typical trade-off between annotation corpus size and feature engineering, this method uses a larger set of more specific features including features that are highly specific to a certain event attribute (e.g., date patterns). The additional engineering effort is paid off by the need for only a relatively small annotated corpus and thus by being able to be quickly applied to a large set of attributes, given an extensive library of prebuilt features. This also allows us to extract attributes at a much finer granularity allowing the extractor to be expanded beyond the minimal *what?*, *when?* and *where?* fields. We exploit field specific characteristics to precisely narrow down a set of potential candidates and train random forest models to distinguish them from the large amount of misleading noise on pages.

Thesis Organisation This thesis is structured as follows: we briefly describe related concepts in the preliminaries and further define the problem formally in Chapter 2. Chapter 3 highlights related work. Chapter 4 presents our work in bootstrapping annotations from `schema.org` in the events domain. In Chapter 5, we introduce TIDE, the distantly-supervised event extraction approach, and Chapter 6 introduces TIME, the supervised approach to multi-attribute extraction. Finally, we conclude and discuss future work in Chapter 7.

Chapter 2

Preliminaries and Problem Definition

2.1 Preliminaries

2.1.1 The Structure of Websites

In this section, we introduce the necessary basic concepts related to web pages and their representation which we will use in this thesis. For more complete definitions, we refer to the respective W3C standards [6]. We assume familiarity with the following basic notions of the world wide web: The world wide web is organised into **web sites**, typically identified through a mid-level domain. Web sites are then often referred to as `microsoft.com` or `facebook.com`, that is the mid-level domain (`microsoft`) and the top-level domain (`com` or `co.uk`). Conceptually, we treat a web site as a set of **web pages**, each identified through a HTTP URL and typically with the same mid- and top-level domain.

For the purpose of this thesis, **web pages** are HTML documents possibly containing or referring to additional external resources such as stylesheets, scripts, images, etc. We assume that there is a mapping $\mathcal{R}$ that renders each web page P into a structure (D, C) such that D is the document object model (DOM) representation of that web page as created by a browser when rendering the web page including any referenced external resources and C is a mapping from nodes in D to the CSS properties of that node (as retrieved by the DOM function `getComputedStyle()`).

- A DOM tree D is a tree over the nodes in the HTML page as defined in [6]. Intuitively and ignoring some finer points, there are four types of nodes in the DOM:
 - (1) A virtual *document root node*, representing the entire document and typically with a single child, the element node for the `html` element.
 - (2) A *text node* is piece of textual information enclosed by HTML tags. Text nodes have neither children nor attributes and thus only appear as leaf nodes in the DOM tree.
 - (3) An element node for each element in the HTML. Element nodes are often differentiated by their *tag* corresponding to the tag used in the HTML markup. Element nodes may have other elements or text nodes as children and may additionally have associated an associated map from attribute

names to attribute nodes. The order of the children of an element node is preserved, while the order of the attributes is arbitrary.

- (4) An attribute node is associated with an element and must have a unique *tag* among all attributes of that element. It may not have children or attributes itself.
- CSS properties are associated with an element node from D and are in turn a map from CSS property names to values as defined in CSS 2 [2]. We do not consider complex properties in this thesis and thus all values are plain Strings. We highlight in particular the CSS box model that provides access to “bounding rectangles” around the elements in a DOM (as retrieved by the DOM function `getBoundingClientRect()`). As illustrated in Figure 2.1, the bounding rectangle of a DOM element provides height and width as well as coordinates of the element corners. In the following, when referring to an element’s X and Y coordinates, we refer to the X and Y coordinate of the *left top corner* of the bounding rectangle.

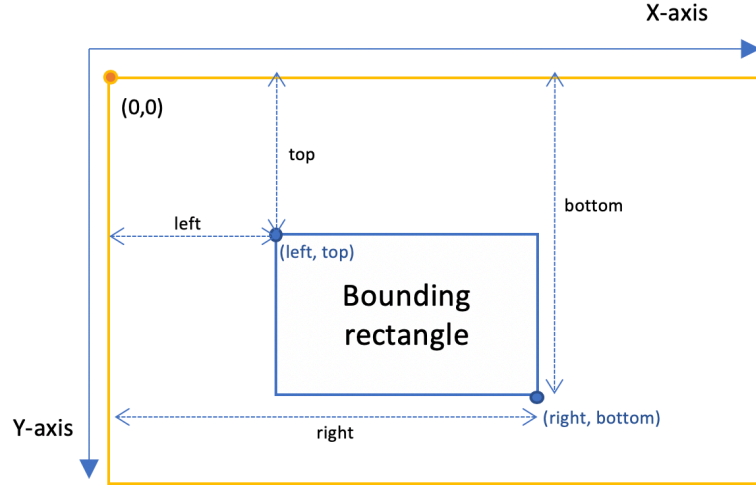


Figure 2.1: Bounding box of a DOM element

There are many other CSS properties used in this thesis, for details we refer to the CSS 2 [2] specification.

2.1.2 Tree-Based Machine Learning Models

In this section, we overview tree-based machine learning models, and in particular random forests [30, 59, 27], which is the class of supervised machine learning models that is used throughout the thesis.

A supervised machine learning model is a function $f(\mathbf{x})$ with parameters θ . The parameters are learned over a training dataset $\mathbf{D}$ with n variables and one label. For $i \in 1, \dots, n$, let X_i denote a variable (or feature), and x_i denote a value in the domain of X_i . Each data point $(\mathbf{x}, y) \in \mathbf{D}$ then contains a vector of feature values

$\mathbf{x} = (x_1, \dots, x_n)$ and a label y . For classification scenarios, the label has a discrete domain $y \in \{1, \dots, d\}$. Once learned, the model $f(\mathbf{x})$ can be used to predict the label y for unlabelled data points. We often represent a model alternatively or additionally through a prediction function $p(\mathbf{x}, y)$ that returns a score on how confident the model is that y is the correct label for $\mathbf{x}$. $f(\mathbf{x})$ is then simply the maximum $p(\mathbf{x}, y)$ over all possible labels y with some tie breaker in case of conflicts.

Tree-based machine learning models are popular classification models for several reasons: (1) they do not require setting and tuning hyperparameters, which makes them easy to learn, (2) they are invariant to irrelevant features, (3) they are scalable, and (4) they tend to be easier to interpret than other models [27]. In this thesis, we use random forests for the classification models, which is one of the most popular classifications models according to the latest Kaggle “*State of Data Science and Machine Learning*” survey [15]. A random forest is an *ensemble* model that is composed of several smaller tree-based models, in particular *decision trees* [30]. Thus, we first overview the algorithm for learning a single decision tree, and then show how it is used to construct a random forests.

2.1.2.1 Decision Tree Learning

A decision tree forms a tree data structure where each inner node represents a decision and each leaf gives a prediction for the label y . A root to leaf path represents a *sequential decision making process* [31, 27].

The decisions at inner nodes are represented by unary threshold functions over one feature X_i , for $i \in \{1, \dots, n\}$, that are parameterised by some parameter θ . These threshold functions partition the data space into regions, and each region is then represented by another decision tree that forms a subtree of the inner node. The threshold functions typically represent binary decisions, such that the resulting decision tree forms a binary tree [31]. This restriction is used to restrict the search space for finding the best threshold function for each inner node.

The type of the threshold function depends on the type of its input variable. For a continuous variable X_i , the threshold functions are typically represented in the form of: $X_i \leq \theta$, where $\theta \in \mathbb{R}$ is a constant parameter. If X_i is a categorical variable, then the threshold function typically has form $X_i = \theta$, where θ is one of the values in the domain of the categorical variable. The functions divide the partition of the input dataset into two regions, based on whether the threshold is satisfied or not.

Each leaf node of the decision tree gives a prediction for the label y , which is derived from the partition of the training dataset that satisfies all thresholding conditions on the path from the root to this leaf. In this thesis, the leaf nodes provide a probabilistic distribution for the label, which is given by the frequency of each category for the label in the dataset partition. This distribution can be directly used as scoring function $p(\mathbf{x})$ as introduced above.

Example Figure 2.2 represents a decision tree that is used for classification. The label y has discrete domain $y \in \{1, 2, 3, 4\}$ and the features are $X_1, \dots, X_5$. Features X_2 and X_5 are categorical variables and all others are continuous. Each inner node defines a threshold function that partitions the data, e.g., the root node partitions

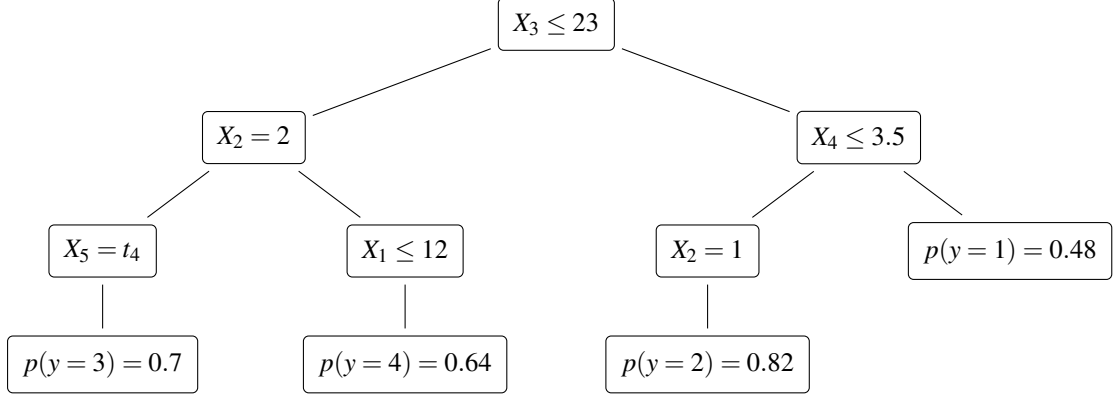


Figure 2.2: Example of a classification tree. The label y has discrete domain $y \in \{1, 2, 3, 4\}$ and the features are $X_1, \dots, X_5$. The leaf nodes show the probability for the most likely category of the label, the other probabilities are omitted for simplicity.

the data based on whether feature X_3 is less than or greater than 23. The first child of the root, i.e. the node with threshold function $X_2 = 2$, further partitions the dataset for which all data points satisfy $X_3 \leq 23$ along the variable X_2 . Leaf nodes provide a categorical probability distribution for the label over the partition of the dataset that satisfies all of the conditions on the root to leaf path.

Learning a decision tree that accurately classifies the input data involves finding the structure of the tree, as well as, the variable and parameter for each threshold function at inner nodes. Finding the optimal decision tree has been shown to be NP-hard [31]. For this reason, the state-of-the-art approach to learning decision trees is to use a greedy algorithm. In this thesis, the classification models are based on the seminal CART (Classification And Regression Trees) algorithm [31], which we overview next.

The core computation of the CART algorithm is to evaluate the cost of choosing a given threshold functions for a new inner node in the decision tree. For classification trees, this cost is given by the gini index, which is defined over a set of categorical labels $\{y_1, \dots, y_d\}$:

$$gini = \sum_{k=1}^d (\pi_k(1 - \pi_k)) = 1 - \sum_{k=1}^d \pi_k^2 \quad (2.1)$$

where π_k denotes the probability that the label y is equal to y_k .

The gini index is a measure of statistical dispersion, i.e. it is high when the probabilities for each category are uniformly distributed, and it is zero when all data points fall into one category.

The CART algorithm constructs each node in the decision tree recursively [31]. Let N denote an inner node in the decision tree, and ϕ be the conjunction of all threshold functions of all ancestor nodes of N , i.e. all nodes between N and the root of the tree. The CART algorithm chooses as thresholding function for N the function that minimizes the gini index for the partition of the training dataset that satisfies all ancestor constraints in ϕ . This is done with an exhaustive search for all possible threshold functions, i.e. for all variables and all possible threshold parameters for each variable.

In theory, the decision tree is grown until any potential new split does not decrease the gini index. In practice, however, this strategy leads to very large trees that are prone to overfit the training data. In this case, the model does not perform well for new and unseen data points. Therefore, it is common to restrict the size of the tree with one or several of the following user defined input parameters:

- *Maximum depth:* Fixes the maximum depth of the decision tree. If the depth of a node equals the maximum depth, this node will not be split further.
- *Minimum number of data points:* A split of a node is only attempted if the partition of the dataset that satisfies all the parent conditions has at least a minimum number of data points.
- *Minimum cost reduction:* A split of a node is attempted only if it decreases the cost given by the gini index by a factor of ϵ , where $\epsilon \in (0, 1)$ is a constant.

Decision trees are popular models for real world applications because they are scalable to large datasets, invariant to scaling or other transformations of the input, robust to irrelevant features, and easy to interpret. A single decision tree, however, is typically not very accurate, and tends to either overfit (if it is too large) or to underfit the training data (if it is kept too small). This results in low accuracy when predicting the label for unseen data points. Therefore, it is common to construct an ensemble model that consists of a variety of small and slightly different decision trees, in order to average out the prediction that each tree makes. This is the underlying characteristic of a random forest, which we will overview next.

2.1.2.2 Random Forest

Random forests are an extension to the idea of bootstrap aggregation [59], or bagging for short. Let β be a predefined constant, then bagging learns β decision trees over random samples of the training dataset. As above, we can represent each decision tree $b \in \{1, \dots, \beta\}$ as a function $p_b(\mathbf{x}, y)$, which for a given feature vector $\mathbf{x}$ returns a prediction for the label y . The prediction of the bagging model is the average over all B decision trees:

$$p(\mathbf{x}, y) = \frac{1}{B} \sum_{b=1}^{\beta} p_b(\mathbf{x}, y).$$

Random forest extend the idea of bagging by enforcing that each decision tree is learned over a random subset of the features [30]. This restriction avoids that the decision trees in the ensemble are highly correlated, since for a bagging model it is possible that there are a few dominant features that would appear in all of the learned decision trees.

In addition to the input parameters for decision trees, a random forest model takes as input additional user defined input parameters, which determine the structure of the model:

- *Number of trees:* the number of individual decision trees that make up the ensemble model.
- *Number of attributes:* the number of attributes to randomly select for each tree in the ensemble.
- *Size of dataset sample:* the size of the random sample of the training dataset that is used to learn each tree, the parameter is usually given as a percentage of the overall dataset.

2.2 Problem Definition

This thesis is concerned with events and event announcements that advertise the fact that a certain event is going to take place. Informally, an *event* is a thing of relevance occurring at a specific time or time span. Typical examples are concerts, conferences, lectures, flight arrivals, and many others, see also [46, 109]. An *event announcement* advertises that a certain event is planned to take place.

Previous approaches on event extraction and detection have focused on detecting largely past events that are being reported on in news or other long- or short-form text, or that can be derived from observations or measurements on streams of data. Typical examples involve the extraction of events from news articles reporting about a terrorist attack or the detection of a wildfire by observing and analysing a stream of tweets.

This work, however, focuses entirely on event announcements where the event is explicitly planned and advertised by the event organiser or some other entity. In the rest of this thesis, we drop the distinction between event and event announcement where it is clear from the context. Only in cases where we need to make the distinction particularly evident are we using event announcement and event as defined here.

Events and event attributes Recall, that we further focus on the extraction of *raw attribute values* for events and leave the linking of certain attributes to existing entities to a separate record linking step (see Chapter 7).

We therefore adopt a very basic model of an event:

Definition 1 Let Σ , the **event attribute schema**, be the set of valid event attributes and D_Σ a mapping such that for all event attributes $a \in \Sigma$, $D_\Sigma(a)$ is the set of permissible values. Then an **event** $E : \Sigma \rightarrow \bigcup_{a \in \Sigma} D_\Sigma(a)$ is a partial function mapping **attributes** from Σ to their concrete value.

We sometimes refer to $E(a)$ where $a \in \Sigma$ as E_a . Unless otherwise stated the permissible values of an attribute is the set of all strings. Note that E is partial and thus does not need to provide a mapping for all event attributes in Σ . However, we also distinguish a subset $\Gamma \subset \Sigma$ of the event attributes, called the **core event attributes** and a set of event constraints C over the attributes.

Definition 2 An (event) constraint is a negation- and quantifier-free first order formula where (1) the variables range over Σ , (2) the constants over $\bigcup_{a \in \Sigma} D_\Sigma(a)$, and (3) the atomic literals are comparisons of the form $t_1 \theta t_2$ with t_i ($i \in \{1, 2\}$) a term (thus either an event attribute variable or a constant from the permissible values of all event attributes) and $\theta \in \{\leq, \geq, =, <, >, \neq\}$. Comparisons between values that are not convertible to a common type always return false. An event E is then **valid** if and only if $E(a)$ is defined for each attribute $a \in \Gamma$ and all constraints in C evaluate to true when substituting each occurrence of a variable with $E(a)$ or $\perp$ if $E(a)$ is undefined. $\perp$ is a new constant that is not convertible to any type.

In the rest of this work, we use $\Sigma_0 = \{\text{title, location, date}\}$ as the basic set of event attributes answering the questions what? where? and when? [9, 46, 109]:

- An event typically has a title or name solving as a short description and identifier of *what* the event is: *Oxford's Christmas Light Festival*.
- The date of an event specifies the day and potentially the times *when* the event takes place: *Fri, November 16, 2018 7:00 PM – 10:00 PM GMT*.
- The location of the event specifies *where* the event takes place: *“Oxford University Museum of Natural History Parks Road Oxford OX1 3PW”*.

Where we use Σ_0 all attributes in Σ_0 are core event attributes, i.e., $\Gamma_0 = \Sigma_0$. Note that in Σ_0 both date and location are fairly unstructured and likely need to be processed further for linking and data integration. We do not use any constraints with Σ_0 . For more details, see Chapter 5.

Therefore, we also use $\Sigma_1 = \{\text{title, location, description, start date, end date, start time, end time}\}$ as the extended event attribute set, with Γ_1 only title, location, and start date. With Σ_1 we typically associate two constraints

- if defined, start date $\leq$ end date and
- if defined, start date $\neq$ end date $\vee$ start time $\leq$ end time

For more details, see Chapter 6.

This still leaves a number of common event attributes uncovered, see Chapter 7 for details.

Event pages We focus in this thesis on the extraction of event (announcements) from (single-entity) event (announcement) pages.

Definition 3 A web page W is called a **single-entity page** if its main content is describing primarily a single entity, e.g., a product, an event, or a person. This term is specifically used to distinguish such pages from search engine result pages (SERPs) or other listing pages, as well as administrative or navigation pages such as contact forms.

Single-entity pages are a common focus of many previous data extraction approaches, e.g., [32], as they tend to contain more information about a specific entity than other pages on a website.

Definition 4 A single-entity page whose primary entity is an event announcement is called a **event page** in the rest of this thesis.

Note that neither definition precludes the presence of information from other entities of the same or other type on the page. In particular, we often observe “related” entities of the same type, in our case other event announcements, in sidebars, footers, or other ancillary locations of the page.

Figure 1.2 shows the main content of an event page¹. The main purpose of such a page is display event related information, in form of the values for event attributes such title, date, location and description, and information about actions the user can take to buy tickets, find the location, or otherwise interact with the event. Figure 1.3 shows the events by the same organiser from the same page, a typical example of a noise “related” events block.

Event identification and extraction This observation leads us to the definition of the problem(s) investigated in this thesis. We focus on the problem of *identifying* an event (announcement) on an event page and *extracting* its attributes from that page. We observe that due to the large variety and significant long tail in the event domain any approach that requires significant supervision for each web site is infeasible and thus require that the problem is solved without supervision for each web site.

Definition 5 *Let W be an event page and Σ be an event attribute schema with associated core attribute set Γ and constraint set C . Then the problem of **event (announcement) extraction** is the problem of identifying the primary E on the page and extract, for each attribute $a \in \Sigma$ that is present on W the concrete values such that E is valid. If the primary event on W is not valid, no event should be returned.*

We sometimes colloquially refer to the extraction of a specific event attribute as “date extraction” or “location extraction” but understand that in the context of the previous definition.

Chapter 5 introduces an approach for an instance of this problem with Σ_0 and no supervision at all. Chapter 6 presents the second approach that uses the larger, more complex Σ_1 and is more easily extended at the cost of requiring some supervision. The supervision is, however, provided once for all sites and we show that only a small sample of sites already suffices to learn classifiers that are broadly applicable in particular when using well tailored features.

¹www.eventbrite.co.uk/e/oxfords-christmas-light-festival-tickets-48590273817

Chapter 3

Related Work

The task of learning to extract events from the web lies at the intersection of multiple research areas. While the problem stems from web data extraction, it is inspired by work in web classification, natural language processing and applied machine learning. In this chapter, we provide a summary of the most relevant research related to our work: Section 3.1 provides an overview of the state of the art in traditional web data extraction. Section 3.2 introduces the field of event extraction, a subfield of web data extraction, which, deserves attention in its own right as the majority of the existing event extraction work focuses on extracting from editorial news text and social media.

3.1 Web Data Extraction

A number of surveys in the area of web data extraction have been published over the last two decades [20, 43, 71, 74, 65, 45, 35]. Traditional approaches to web data extraction can be split into two main categories: template-based and template-independent techniques. *Template-based* techniques [37, 48, 32] use small extraction programs, called “wrappers”. These programs are based on the underlying HTML structure using XPath [14] or in Diadem’s [48] case OXPath [50], which describe how to navigate through a given webpage and extract information at specified position within the DOM tree.

These techniques are useful when handling big websites which have many webpages of the same underlying structure, where a single wrapper is often sufficient to extract data from one website. This effectively requires individual wrappers be written for each website. The drawback to this approach is evident when data is needed from many different websites, as individual wrappers are needed for each and every website. Furthermore, this complexity is compounded when taking into account that many websites use more than one web template and are A/B testing and modifying the underlying structure. In more recent years, *template-independent* techniques try to overcome the short coming of the above mentioned techniques, by exploiting domain specific characteristics of webpages, rather than the underlying structure of candidate web pages. Such approaches have been used in the news domain [109], where the sought attributes are usually limited to the news title and main news article text. However, the main shortcoming of such techniques is the lack of a

large, annotated dataset necessary for training purposes, as it's hard to come by and cost prohibitive to create. In the remainder of this chapter, we discuss both types of approaches, first introducing template-based methods in Section 3.1.1 followed by template-independent approaches in Section 3.1.2.

3.1.1 Web Wrappers

Small programs to extract information from webpages are referred to as “*wrappers*”. In the context of web data extraction, Ferrara et al. [43] provide the following definition:

Definition 6 *A procedure, that might implement one or many different classes of algorithms, which seeks and finds data required by a human user, extracting them from unstructured (or semi-structured) web sources, and transforming them into structured data, merging and unifying this information for further processing, in a semi-automatic or fully automatic way [43].*

We first introduce web wrapper languages in Section 3.1.1.1 and programs to induce such wrappers in Section 3.1.1.2.

3.1.1.1 Web Wrapper Languages

Regular expressions. Writing regular expressions for HTML websites generally focus on table structures, HTML tags, and word boundaries. Regular expressions are typically the underlying technology used by point and click solutions. These systems offer users a graphical user interface (GUI) where they can manually select what fields to extract from the webpage. A user selects one or multiple elements on the page, and the system generates one or many regular expressions that can later be applied to other pages with the same template [100]. The major drawback to this approach is the significant amount of supervision required and a lack of flexibility, as human intervention is required to create a new wrapper for every unique template, both at an induction and maintenance level.

Programming languages. Web specific wrapper programming languages consider web documents as a semi-structured tree document, rather than a long text in regular expression-based techniques. The DOM describes the structure of the page, where each node is an element with corresponding attributes and content. Web wrapper programming languages extend regular expression-based techniques, in that both the semi-structure of the DOM tree and the textual content of the elements are fully exploited. Gottlob and Koch [57] introduced and formalised the first wrapping language. Their language relies on monadic Datalog over trees [58], demonstrating that they are very expressive due to their equivalence to monadic second order logic. The algorithm relabels nodes while truncating irrelevant ones, returning a subset of nodes that represent the selected data. Baumgartner, et.al [18, 19] implemented this technique with some restrictions in their *Elog* wrapper language. OXPath [50] is a scalable data extraction language that extends XPath by enabling data to be extracted from interfaces that are not built for the purpose of being understood by a machine. Therefore, OXPath can simulate user behaviour by identifying visual features a human user would use to navigate their

way through the page and identify the relevant information based on that. In order to simulate the behaviour, OXPath interacts with rich multi-way interfaces of scripted web pages. Actions are declaratively specified through action types with context elements, such as deciding which form field to fill and which buttons to click to navigate through the site. Furthermore, a novel multipage model for web applications, capturing both modifications to a page and navigation within it, allows actions to have formal semantics. Similar to XPath, OXPath has precise selection capabilities, as opposed to heuristics for selection, which has been used in other data extraction techniques [29]. OXPath extends selection capabilities in multiple ways. Firstly, it allows the selection of elements based on their visual features, including CSS properties. Secondly, it can navigate through sequences of pages, including following multiple links from the same page and unbounded or bounded iterations through pages, following next links in a series of result pages, by including the use of the *Kleene Star* to do so. Moreover, OXPath identifies information for extraction which it assembles in to records (sometimes hierarchically), regardless of how it was structured on the page.

The major weakness of XPath based techniques are their lack of flexibility, as each XPath expression is based on the structural properties of the webpage for which it was defined. Changes in the structure of the DOM tree could render XPath expressions obsolete. The respective XPath based extraction technique needs to be updated accordingly. *Wrapper robustness* aims to find the one expression out of all possible XPath expressions, that is the most resistant to structural variations in the webpage template [52].

3.1.1.2 Web Wrapper Induction

Wrappers can be implemented manually, by writing path statements or regular expressions over HTML documents. This requires human expertise in the extraction domain and programming skills in the language in which the wrappers are scripted. Alternatively, wrappers can be defined through an inductive approach, which is the main aim of automated web data extraction systems. In this section, we review some of these approaches.

Roadrunner [37, 36] is a hybrid system performing template-based matching. The approach is tailored to data-intensive websites which share regular structure, commonly found with websites whose pages are generated by the same script. Roadrunner runs with both manually labelled and automatically labelled web pages (as seen in [70]). Furthermore, a priori knowledge of the schema of previously learned pages can be taken into account. The system works with two pages simultaneously to determine similarities in the structure. Roadrunner identifies pages generated from the same script. The problem definition is then reduced to to extract the sought data from the common structure, taking into consideration missing values and small differences in the page.

Rapier [87, 34] is a system which learns simple rules from documents without having to pre- or post process the document. Furthermore, these rules use very little syntactic and semantic information. This makes the system a lot more flexible, however it is harder to determine which extracted rules are actually relevant. The authors therefore introduce a learning algorithm that identifies such rules based on inductive

logic programming. WIEN [70] is based on an inductive learning technique, which automatically labels pages for training purposes. This high level of automation enables less human involvement in the process, however means that the system cannot handle missing values, which is important in web data extraction. SoftMealy [61] was one of the first system built specifically for data extraction from the web. The system uses bottom up inductive learning to learn extraction rules, based on non deterministic finite state transducers. Extraction rules are in the form of state transitions of where states are the extracted data. The training phase uses pages representative of all possible permutations of input pages, and creates an internal representation of HTML documents. The advantage over WIEN [70] is that it is fed separators rather than the entire HTML string so tokens can be matched with rules to determine the right state transition to be applied. STALKER [88] is a supervised learning system that induces wrappers. The system considers documents as a tree of tokens on which extraction rules are applied and inferred. It differs from SoftMealy in that the position of tokens in the documents are manually labelled as to ensure the capability to deal with missing values and hierarchy.

Partial tree alignment strategies aim to identify data record regions within the web page, as formalised by Zhai and Liu [115, 116]. These algorithms segment the page before applying partial tree alignment. The segmentation is a preprocessing phase that analysing the webpage based on the DOM tree and visual cues, recognising gaps between data records. Subsequently, the partial tree alignment algorithm is applied to the previously identified data records. Each identified data records represents a new subtree. The technique excludes nodes that cannot be aligned, applying only node removal, insertion and replacement to transform nodes. As stated above, this method is very sensitive to changes in the HTML structure, as the algorithm is based on the alignment of the tree at the time of the application.

WEIR [32] leverages overlapping data in web page sources within a website to generate extraction rules. The system first compares underlying templates by identifying the textual nodes that do not belong to it and generates XPath rules for such elements, and thirdly removes the rules that are unlikely to be relevant attributes to be extracted. DIADEM [48, 49] is a wrapper induction system, that explores a website for repetitive data records (ie. listings) linking to result pages using extensive domain knowledge. Based on the redundancy of the underlying template of the discovered pages, XPath wrappers are automatically generate to extract from all result pages for that particular website.

Alternatively, Visual Box Model [69, 53, 28] uses spatial reasoning for web data extraction: instead of using cues from standard HTML information such as `<table>` elements, the authors present a technique based on the application of an X-Y cut OCR algorithm on the rendered web page. A visual grid is generated to determine visual cues. Information is stored in an X-Y tree, which is built for parent nodes of leaves nodes not to be empty tables. A further operation assures that the content inserted into the X-Y tree contains relevant information, as some web pages misuse tables for structural and formatting purposes.

3.1.2 Template-Independent Techniques: Machine Learning and Vision-Based Systems for Data Extraction

Early approaches to web page classification used a bag-of-words representation of a web document, considering all parts of a web page as equally significant. However, web documents, unlike other documents, have characteristics which can further enhance the processing of such pages. Such characteristics include hyperlinks, metadata, structural information and visual information. Golub et al. [56] study the significance of different web page elements; namely title, headings, metadata and main text, in automatic web page classification. Three different methods were used to derive the significance indicator of the web page elements: semantic difference, (total and partial) precision and recall, and multiple regression. It was shown that for the best result, all the elements should be included in the classification process. Zhi et al. [119] propose a probabilistic model named *hierarchical conditional random fields*, to extract data records and label attributes simultaneously. Record detection is supported by the availability of semantics required in attribute labelling, and vice versa, the attribute labelling task is improved as the data records are labelled in a collective process.

Wang et al. [109] define the template independent wrapper learning problem for news article extraction as a classification problem, using only webpages from one website as a training input and defining classifiers for the title and body of the news page. For this purpose, both content and spatial features are used for both items to be identified. In terms of content of the news article body, two important features are used. Firstly, the *size of the collection of formatting elements*, which are given in the child nodes of a given subtree T , is normalised by the number of formatting elements in the DOM tree. Secondly, the *formatted content length* is the total length of text contained in the collection of formatting elements. The value is normalised by the sum of the length of all formatting elements in the DOM tree. Furthermore, in terms of spatial features of the news body of the website, the browser is leveraged to determine the bounding rectangle of a subtree T denoted as R . Four spatial features of subtree T are included in the model to learn how to find the body of a news article: Let (R_{left}, R_{top}) denote the coordinate of the upper left corner of the bounding rectangle R , and R_{width} and R_{height} are the width and height of R . Similarly, S_{width} and S_{height} are the width and height of the screen. These features are transformed in relative spatial features, by using the width and height of the whole page to normalise the absolute features. This allows a comparison of features from different websites. However, the normalisation of these features leads to a problem when pages are particularly long because of some user comments at the bottom of the page. Hence, it is assumed that the content in the first two screens of a website is the most important, and the height of the whole page should not be used to normalise. According to this assumption, the four absolute spatial features are normalised in the following way:

$$\begin{aligned}
 R_{left} &= \frac{R_{left}}{S_{width}} & R_{width} &= \begin{cases} R_{width}/S_{width} & \text{if } R_{right} < S_{width} \\ (S_{width} - R_{left})/S_{width} & \text{otherwise} \end{cases} \\
 R_{top} &= \frac{R_{top}}{2 \cdot S_{height}} & R_{height} &= \begin{cases} R_{height}/2 \cdot S_{height} & \text{if } R_{bottom} < 2 \cdot S_{height} \\ (2 \cdot S_{height} - R_{top})/2 \cdot S_{height} & \text{otherwise} \end{cases}
 \end{aligned}$$

The body subtree identification model is a function that maps the feature values of a subtree to the estimated probability a subtree is the news body. The algorithm first generates candidate subtrees, and then applies the model to every candidate, selecting the candidate with the highest posterior probability. The title of the news article is identified in a similar fashion. There are three content features: the font size of the text, whether the text ends with a full stop, and the number of words in the text. The spatial features include the position and size of the title. Again, the whole page size is not taken into consideration, as the authors assume that the title will be in the first screen of the web page. Another interesting feature is *overlap*, which encodes the overlap between the title and the subtree of the news body. This is due to the observation that the title and body of the news article are usually horizontally close to each other. Similarly, the *distance* between the title and the body and the shape of the title are used as a features.

Visual Analysis Zheng et al. [118] perform template independent news extraction based on visual consistency. For that, a web page is represented as a visual block tree. Visual features are extracted and a composite visual feature set for the news domain is derived. The following four basic features are used: position features (left, top and nested depth), size features (width and height), rich format features (font size, boldness, italic) and statistical features (image number, hyperlink number, text length, paragraphs, table number, etc.). HTML tags are not used as features, as the authors argue that the tags are too diverse in the news page domain. Since the topological structures of DOM trees vary at domain level, they are not used for matching (template independency). A “parent-child” visual relation between two blocks on a given page are used. Two blocks are parent-child blocks if one covers the other and there exists no third block in the same page that covers one, and is covered by the other. Hence, the relative width and location of a block, in relation to its parent block can be used. This is a better approach than traditional topological structure based wrappers as the two blocks can be compared despite their nested depth. The Adaboost algorithm [47] is used to learn the composite visual features. Initially, the main block is roughly located, then the main block is analysed more carefully to identify the headline and news body. Liu et al. [80] introduce ViDE. The system transforms a web page into a visual box tree, which is a segmentation of the website into rectangular regions containing text or images. The leaf nodes of such a visual tree are blocks that cannot be segmented any further and therefore correspond to continuous text or images. Position, layout, appearance, and content features are used to develop a clustering method at a data record and data item level to finally generate visual wrappers to improve the efficiency of both data record extraction and data item extraction. The main shortcoming is that the method can only process deep webpages which contain one data region. More recent work [81], introduces an extraction framework based on distantly supervised annotations generated through the alignment of an existing seed set from a knowledge base with websites.

Deep learning approaches. Many information retrieval and classification tasks have been advanced with the progress in deep learning methods. Interestingly, very little work has been done in the field of web data extraction. Cai et.al. [33] and Nagarajan et.al. [89] use convolutional neural networks to process screenshots of webpages to predict the position of relevant data regions. [55] use convolutional neural networks and present a spatial text encoding to encode visual and textual content of web pages into neural networks for data extraction. The sparsity of deep learning approaches in the web data extraction research field could be due to the lack of readily available datasets of webpages which include visual information. We present a method to automatically produce a dataset for the purpose of training web data extraction methods in Chapter 4.

3.2 Event Extraction

Web data extraction techniques heavily depend on the target domain from which they extract data. Not only does the extraction schema vary by domain, i.e. the attributes to be extracted, but also the underlying source and stylistic features. As websites are designed with optical cues optimised for the human eye, both visual and structural features vary accordingly. For example, the events domain has different characteristics than the online news domain. Online news articles characteristically have a larger title, which is visually emphasised by its size, colours and boldness. Such a title is typically followed by a date and a big blob of text, which represents the main content of the news article. The events domain, however, is very distinctive, as it has many distinguishable traits. The main peculiarity of the domain remains the elusive definition of what an event is. Most authors agree that an event consists at least of a temporal attribute, a location, and some form of name or title of the event. However, further event attributes mostly vary on the underlying event source. In events extracted from news, source and target actors are usually important event attributes, whereas events extracted from the open web often have a selling price. Interestingly, depending on the underlying source, sought events either lie in the past or the future: news articles usually report about events in the past, while events extracted from the open web or social media are often in the future.

Traditional research in event extraction focuses on the extraction of political, cultural, or other general interest events from unstructured data. Such text is usually editorial news content like in [72, 107, 94] or social media such as Twitter [99, 98, 101, 111, 24, 23, 110]. There has been increasing interest in extracting events from “single event pages” such as local sites. We shall therefore, in the following section, discuss the different underlying sources from which events can be extracted, as the problem definition, and therefore respective techniques to resolve the problem, intrinsically vary with the source.

Sections 3.2.1 and 3.2.2 overview event extraction from editorial news text and social media respectively. We provide a review of extraction techniques from these sources; however it is important to note that their problem definition differs as they extract from text and microblogging data sources. In Section 3.3, we highlight the most relevant work to extracting events from webpages in the scope of this thesis.

3.2.1 Event Extraction from Editorial News Text

Text-based event extraction has many applications: firstly, extracted events can be used in personalised news aggregators, which optimise the news summarisation based on personal preferences. Event extraction from news is further used in risk and threat analysis, as well as surveillance [94, 96]. Moreover, the financial market is highly sensitive to breaking news, and therefore news surveillance is particularly interesting in such a domain. Lastly, event extraction from text has been exploited in the medical and scientific fields, mostly to extract medical or biological events. In the scope of this section, we will mainly focus on event extraction from news editorial text.

Event data extraction from news can be defined as the problem of retrieving structured data from unstructured text. Unlike news data extraction [109], news *event* extraction algorithms have to retrieve the relevant event features from text, rather than from the DOM structure of the page. Thus, event extraction in the news domain can usually be considered a natural language processing (NLP) problem rather than a web page classification or traditional web data extraction task. Hogenboom et al.[60] distinguish between approaches used in event extraction from text, based on the standard classification of techniques in modelling. Firstly, *data-driven approaches* (section 3.2.1.1) use statistical methods to retrieve information from text. Secondly, *knowledge-based methods* (section 3.2.1.2) uses expert knowledge through patterns and background information, such as gazetteers, to extract information from text. Thirdly, discussed in section 3.2.1.3, a number of techniques use a combination of ideas from both groups. In the following three subsections, we adopt this categorisation of techniques to give an outline of the work in the field.

3.2.1.1 Data-driven Event Extraction

Data driven methods purely rely on quantitative methods to retrieve information from free text, hence a large corpus is usually required. Such methods include statistical and reasoning methods, information theory and probabilistic techniques. The main aim is to find statistical relations that can be exploited for retrieval. This includes word frequencies, TF-IDF (Term Frequency – Inverse Document Frequency metric) methods, clustering and n-grams. The main downside, however, is that such methods do not consider semantic meaning of the derived evidence.

Ahn [17] defines the stages of event extraction as follows:

- (1) Anchor identification: identifying event mentions in text.
- (2) Argument identification: identifying the mentions of entities such as actors, location and time references.
- (3) Attribute assignment: determining modality and generality of the event.
- (4) Event coreference: determining which extracted event instances refer to the same real-life event.

The author argues that the above stated steps are highly interdependent, however studies the steps individually. For the first three tasks, TiMBL2 [38], a memory-based (nearest neighbour) learner and MegaM [40] is used to train classification algorithms. The latter learning framework is used for the coreference step, as it provides probability measures.

Zhang et al.[117] introduced a dynamic indexing tree based framework for the detection of news events by extending two TF-IDF models. The first extension adjusts the term-weights on the term distribution between the clusters of stories and the entire corpus in a dynamic fashion. The second extension enhances the use of shallow NLP terms, such as Part-of-Speech (POS) and named entities. Rather than using a sequential pipeline seen in many information extraction frameworks, Li et.al [79] propose a joint framework for structured prediction and extraction of triggers and attributes of events to mutually improve prediction.

3.2.1.2 Knowledge-based Event Extraction

Petrarch [102, 103, 91] is a series of dictionary-based event extraction systems, mainly used to extract political events from news text.. The approach looks at verb in the input text and maps it to a set of “action” verbs in their dictionary. They locate the “*source actor*” and “*target actor*” of the verb and map it to an event code exploiting grammatical structures. Similarly, Piskorski et al. [107, 95] present a real-time event extraction system, extracting natural and violent disaster events from news on the web. A preprocessing step harvests thousands of news web pages on a daily basis, further classifying them into roughly 700 categories. The articles are parsed for known entities (such as country names or organisations). The articles are clustered by content similarity and geo-tagged for location purposes. Further shallow NLP techniques such as tokenisation, sentence splitting, domain-specific gazetteers, using CORLEONE [93] as a linguistic tool are used, to extract the main attributes from text, based on machine learning and knowledge-based techniques. Extrativism [96] is an activism event specific approach, which automatically extracts event mentions from news sources that could complement, contradict or verify each other, using off-the-shelf NLP tools. The approach is based on what the authors refer to as the *inverted pyramid* style of writing news articles, which states the most pertinent facts at the beginning of the article and leaves the less important details for the end. The major drawbacks with the above approaches are the limitation of the dictionaries used, the sub-domain dependence, and the lack of contextual awareness of such techniques.

3.2.1.3 Hybrid Event Extraction

Frontex [94] is a news event extraction system performing cross article validation from different aggregated articles. Articles are geo-located and clustered based on meta information to determine which of the articles refer to security-related events, using keyword-based techniques. The articles are further clustered into 4 hour windows, the language in which they were published, and according to content similarity. The relevant article clusters proceed to the event extraction engine which is based on 2 systems, namely NEXUS [94, 108] and PULS[112]. NEXUS has a shallow cluster-centric approach, that is more useful for extraction information

from individual clusters of related articles. PULS follows a non-clustered approach which performs a more in-depth of the text of each individual article. A combination of these two systems is used to provide a platform with different approaches to event extraction and to gain a wider coverage in terms of language and event types. The output of the event extraction mechanism is a stream of instantiated event templates for geographical visualisation. To bridge between in-depth analysis of articles and automatically extracted events, the database of automatically extracted events can be accessed to clean, update, compare, group, and filter events.

Naughton et al. [90] merge events by grouping sentences from different news sources that refer to the same event to provide a concise and complete picture about the event. The authors cluster the sentences using regularities in the sequence of events within a news document and a TF-IDF-like weighting scheme, capturing word frequencies. Learning techniques are exploited to determine whether an article refers to single or multiple events. Kuzey et al. [72] populate knowledge bases with events extracted from online news articles. The approach clusters online news based on coreferences of events and sub events, such as the UEFA champions league event and the games of the league as sub events. The clustering of events is performed with a text-based content similarity measure, resulting in a cluster of news sources for each event. A similar graph-like clustering approach is introduced by Argarwal et al. [16]. Julinda et al.[64] work similarly with clusters of news articles that capture temporal expressions referring to the same time and that are likely to be referring to the same event. The authors extract textual references from each source referring to the event using open-domain Information Extraction (OpenIE) [84].

3.2.2 Event Extraction from Social Media

Social media serves as a platform for individual users and organisations to propagate and disseminate event information at a global scale. Depending on the social media platform, such events are either presented in a template provided by the platform, e.g. *Facebook Events*, or small messages such as on microblogging streams such as Twitter. However, social media based event extraction is most commonly based on extracting events from Twitter streams [99, 98, 101, 111, 24, 22, 25, 26, 21, 110, 114, 63, 113, 92], where traditional NLP techniques are not applicable and new Twitter-specific NLP tools are required. With the restricted number of characters to be used for a single microblogging message, users tend to shorten their messages, by omitting stop words and abbreviating words. This renders traditional named-entity recognisers (NER), part-of-speech (POS) taggers, and other traditional shallow NLP tools that are based on the grammatical structure of sentences, obsolete. A number of microblogging specific NLP tools such as NER taggers [98], POS [41], entity relations [66], and dependency parsers [67] are vital for event extraction from social media streams. Event extraction from social media streams is particularly relevant in applications such as disaster management: Imran et al. [62] and Krstajic et al. [68] investigate the extraction of events after mass emergencies, Sakaki et al. [101] analyse Twitter streams specifically related to earthquake responses.

Leveraging multiple sources for extraction. More recent research in extracting events from social media focuses on extracting events from diverse social media outlets for the purpose of cross-referencing and reinforcement. Li et al. [77] merge the output of event extraction mechanisms from Twitter [99] and news [79], considering the challenges of microblogging source data and editorial news text respectively, to reinforce the quality of extracted events. Lit et al. [78] similarly reinforce their findings from multiple sources, further using external knowledge bases to detect fine-grained locations. Becker et al. [24, 23] associate social media events across many platforms through a site-specific scoring method overlapping entity mentions.

3.3 Event Extraction from Single-Entity Pages

There has been increasing interest in industry in extracting events from “single event pages”, such as small, local sites, however, very little has been done in this subfield. Research conducted at Google [46] collects events in the form of retrievable calendar entries, assigning scores to individual event attributes and introduces an algorithm to group such attributes into event records. However, the results only achieving precision in the range of 70-80%. Wang et al. [109] use social media posts, such as tweets that contain links to websites, to find social event web pages and reinforce the extraction from the social media content and the unstructured text on the web page. This technique, however, relies on the overlapping information from multiple sources and cannot be applied to single entity pages directly.

Summary. To summarise, traditional template-based data extraction techniques cannot be applied to the event extraction task as the per-site supervision is not applicable to the domain. Existing template dependent methods cannot handle the complexity of the events domain, as their attributes are far more intricate than attributes in the news domain. The little existing work in extracting events from web pages is either restricting in terms of results or the source pages it can be applied to. This thesis addresses all the shortcomings above, by presenting two different approaches to event extraction that achieve high results over a heterogeneous set of sources.

Chapter 4

Bootstrapping an Event Corpus from Noisy Schema.org Annotations

As with many machine learning problems, the task of learning how to recognise events across the web requires a large corpus of annotated examples. Such a corpus consists of example event webpages, with machine readable annotations for relevant event attributes. However, such a dataset is hard to come by and very expensive to manually create, as human annotators have to sift through thousands of websites to distinguish event pages and annotate specific event attributes on such pages.

In this chapter, we therefore investigate the problem of harvesting such an event corpus from noisy schema.org annotations. The aim of this chapter is twofold: We developed a bootstrapping framework which yields (a) a large annotated dataset, tailored to the purpose and needs of the subsequent learning phase, and (b) provides insight into the events domain on the web. This is particularly useful to understand common characteristics of events for the event extraction learning problem, and additionally provides an understanding of the use of semantic web annotations within the events domain.

The process is a bootstrapping technique, using semantic web annotations as seed values for structured event attributes. We use the Common Crawl web corpus [4], an open repository of crawled web data, to identify single event pages, based on Microdata and RDFa annotations, such as defined by schema.org [9]. We established a set of prerequisites and filtering criteria, to determine which of the pages in the Common Crawl correspond to single entity type pages in the events domain. We leverage the existing annotations of type *event* as a seed set to generate labelled training data for the training and evaluation purposes of the event data extraction task.

This technique yields around 2 Million webpages, from which we extracted event attributes. This corpus not only presents a basis for our event extraction work presented in Chapter 5, but also a much needed dataset for benchmarking in the event extraction community as a whole. Furthermore, it allows us to provide insight into the events domain on the web and usage of semantic annotation.

4.1 Preliminaries

We briefly introduce Common Crawl and Schema.org event vocabulary, as the work introduced in this chapter is based on these two projects.

4.1.1 Common Crawl

The Common Crawl Project [4] maintains an open repository of web crawled data, containing billions of archived web pages, which is hosted on Amazon S3 as part of the Amazon Public Dataset program ¹. The crawls of the data are stored in Web ARChive (WARC) format. Each WARC file stores the HTTP request, response, content and metadata about the crawl. Each crawl contains a snapshot of the crawled web in a given month. For this project, we first used the crawl from September 2016 ² and then at a later stage from April 2018³, as web pages from the first crawl were no longer live after a short period of time.

4.1.2 Semantic Annotations on the Web

The Schema.org project provides a shared collection of schemas for annotating domain knowledge within web pages. Standardised ontologies are available for multiple domains, and can be embedded in webpages through many encodings including RDFa, Microdata and JSON-LD. The aim for the search companies behind the project, namely Google, Microsoft, Yahoo and Yandex, is to make the content on webpages more machine readable, which simplifies automated extraction, structured search and indexing mechanisms. The benefit for the website owners, who add the annotations to their site, is primarily that their attributes can shown in search results which improves clickthrough rates. Such semantic web annotations are embedded in web elements containing event attributes such as *title*, *location* and *start date*, reflecting the schematic structure of an event scope defined by schema.org. However, such annotations are not scalable to the entire web, in particular because not all website owners have the required technical expertise to use them. Another problem is that there is a lot of misuse of annotations. Our bootstrapping mechanism takes this into consideration in a cleaning step which attempts to identify annotations that have been misused or are otherwise inadequate for bootstrapping an event corpus.

Structure. Every schema in Schema.org has *itemscopes*, *itemprops*, and *itemtypes* to represent the hierarchical tree structure of an event:

- **Itemscope.** Itemscope are annotations to a node declaring that this node and its children belong to a particular schema.org domain.
- **Itemtype.** For each itemscope the itemtype annotation declares the domain for the particular scope, such as *events*, *organisations*, or *products*.

¹aws.amazon.com/opendata/public-datasets/

²<http://commoncrawl.org/2016/10/september-2016-crawl-archive-now-available/>

³<http://commoncrawl.org/2018/05/april-2018-crawl-archive-now-available/>

```

<div class="event-wrapper" itemscope itemtype="http://schema.org/Event">
  <div class="event-date" itemprop="startDate" content="2013-09-14T21:30">Sat Sep 14</div>
  <div class="event-title" itemprop="name">SOLD OUT! Typhoon with Radiation City</div>
  <div class="event-venue" itemprop="location" itemscope itemtype="http://schema.org/Place">
    <span itemprop="name">The Hi-Dive</span>
    <div class="address" itemprop="address" itemscope itemtype="http://schema.org/PostalAddress">
      <span itemprop="streetAddress">7 S. Broadway</span><br>
      <span itemprop="addressLocality">Denver</span>,
      <span itemprop="addressRegion">CO</span>
      <span itemprop="postalCode">80209</span>
    </div>
  </div>
  <div class="event-time">9:30 PM</div>
  <span itemprop="offers" itemscope itemtype="http://schema.org/Offer">
    <meta itemprop="availability" content="http://schema.org/SoldOut">
    <div class="event-price" itemprop="price" content="$13.00">$13.00</div>
    <span itemprop="priceCurrency" content="USD" />
    <a itemprop="url" href="http://www.ticketfly.com/purchase/309433">Tickets</a>
  </span>
</div>

```

Figure 4.1: Example event microdata from <https://schema.org/Event>.

- **Itemprops.** Each schema.org domain has a set of properties such as the name and location of an event. Itemprops annotate the nodes containing the value of such properties.

The value of the annotation can either be in the text content or the “*content*” attribute of the node. Itemscopes can be nested, for instance an itemscope of type event can contain an itemscope of type location. The schema.org annotations “*name*”, “*startDate*”, and “*location*” attributes map directly to “*title*”, “*date*”, and “*location*” respectively, from the definition of an event in Section 2.2.

Example Figure 4.1 depicts HTML code with schema.org annotations. The first `div` element with class `event-wrapper` is an *itemscope* of type event. The children of that node contain additional itemscopes and itemprops which describe the event. For instance, the node with class `event-venue` gives the location of the event, and defines a scope of type location, which contains additional nodes that define the properties of this location.

4.2 Approach

In this section, we present our approach to automatically creating an annotated event web data corpus, bootstrapping schema.org annotations. Our approach navigates through each WARC file in the Common Crawl repository, and performs a number of steps in order to identify relevant event related semantic web data. Figure 4.2 depicts the pipeline of the approach, we next overview the intermediate steps.

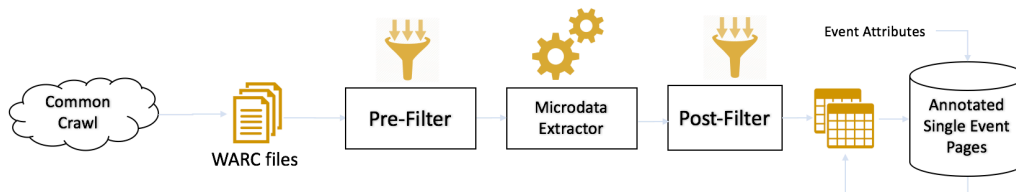


Figure 4.2: Pipeline for bootstrapping an event corpus from noisy schema.org annotations.

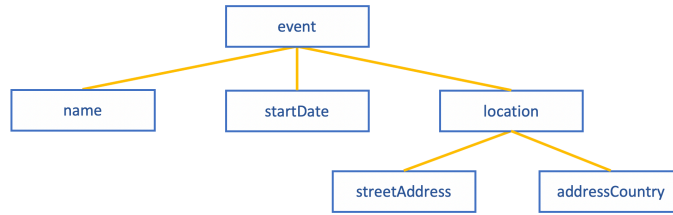


Figure 4.3: Example of nested itemscopes. The location scope is a child of the event scope.

Pre-filter We consider only HTML pages which contain an event scope and can be parsed correctly. We further restrict the minimum and maximum page length to filter out small web pages that are too small to contain relevant information and web pages with a lot of text or long lists, which are unlikely to be single event pages.

Microdata Extractor We traverse the HTML documents to identify the itemscopes and itemprops that describe an event. These annotations are represented by a tree, which we construct recursively. We first traverse the HTML tree to find an itemscope of type event, which is the root of the tree. We then recursively search for nested itemscopes and itemprops, which describe that particular event and add those annotations to the tree. Figure 4.3 shows an example of such a tree. We use the Apache Any23 [1] library for parsing the microdata annotations.

We only consider pages that contain one *and only one* event scope with attributes. For such pages, we map the schema.org annotations belonging to the event item scope into relational records, which stores the relevant information about this node and the nesting structure of the tree. For each annotation, we store the text content, the content attribute, the absolute XPath for this node, as well as, identifiers for the parent scope and the top event itemscope.

Post-processing The pages and annotations are very noisy due to annotation misuse and mistakes. The post processing step filters out those annotations which are likely irrelevant for training purposes. We navigate through nested microdata annotations and perform the following steps to post-process the extracted data:

- We remove all annotations that do not exist in the schema.org ontology.
- We remove all annotations that are irrelevant in the event context, i.e. that belong to a different ontology within schema.org and should not be annotated within an event scope.
- We remove all annotations that annotate a node that does not have a text content.
- We remove all annotations where the text content has less than two characters.
- We remove all annotations where the annotated node is not visible.
- We remove all duplicates.

- We load each page in a headless browser to verify that the page still exists, and the *title*, *date* and *location* related annotations are still included, as sometimes they have been removed or are not available after the Common Crawl archived the page. We remove all pages that no longer exist.

In effort to identify misuse and mistakes, we apply multiple pattern matching techniques to attribute specific patterns (e.g. number of digits in “*startDate*” annotated attributes), to identify misuse and mistakes. From the remaining pages, we select the subset of pages that contain *at least one title* (i.e. “*name*”), date (i.e. “*startDate*”), and one location (i.e. “*location*”) annotated attribute, to yield a dataset of single entity event pages that contain an event as defined in Section 2.2.

4.3 Results

Previous work [39, 86, 54] has provided useful insight into the adoption of semantic web annotations across many domains, including the restaurants, products and automotive domains. However, to the best of our knowledge, no previous study has provided insight into the events domain. In this section, we first give some general insight into the state of semantic web annotations in the events domain as of 2018 and further present the resulting dataset produced by our approach described above.

4.3.1 Domain Understanding of Events on the Web

In total, 64,320 WARC files were processed, which contain 9.3 billion records, 3.1 billion of which are HTTP responses, and 2.9 billion of which were classified in our preprocessing step as “relevant pages” based on their content size and format. We automatically extracted 4.4 million event-related data records from relevant web pages, containing single event microdata formatted in accordance with the event domain on schema.org. The average size of pages with event scopes is 75,979 characters. We also identified 2.5 million pages containing multiple events. Such pages are mostly search result pages in list form, rather than single event pages. They are filtered out due to their irrelevance to the scope of this thesis. Although some of those pages may be single

Number of pages per website	Number of websites	Percentage of Common Crawl event pages
1	35940	32.11%
2	9608	8.58 %
3	6760	6.03 %
4	5298	4.73%
5	4514	4.03%
6	3752	3.35 %
7	3099	2.76 %
8	2604	2.32 %
9	2333	2.08 %
10	2190	1.95 %

Table 4.1: Distribution of the number of pages per website

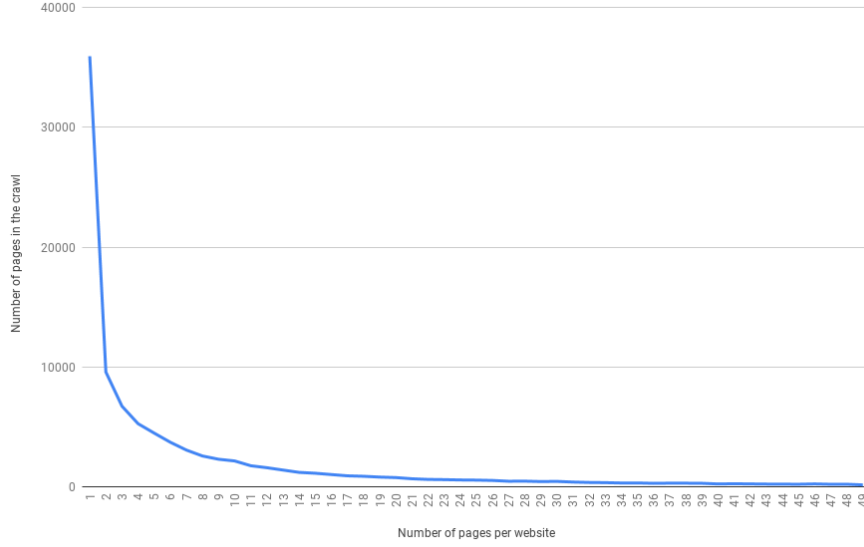


Figure 4.4: Distribution of webpages per websites. The x-axis represents the number of webpages per website and the y-axis represents the number of such websites in our dataset.

event pages and not listings, this step is necessary to ensure the quality of the resulting dataset. Moreover, we discovered that the quality of annotations on these pages is considerably lower than those of single-event pages. In the remainder of this section, we provide general statistics about the state of semantically annotated event pages on the web. We analyse the distribution of pages within a website, along with their annotations, misannotations and types.

4.3.1.1 Distribution of Pages

We consider all webpages from the Common Crawl Corpus of April 2018 that our bootstrapping approach identified as single event pages. For the event scopes on those pages, 96% are microdata annotations and only 4% are in RDFa format. As seen in Table 4.2, almost 95% of these pages belong to a website that has 100 or fewer pages archived in the crawl, and 67.99% have fewer than 10 pages. Figure 4.4 shows heterogeneity of event pages, and therefore underlying templates in the crawl, as the majority of webpages have fewer pages per website. Table 4.1 shows that websites with only 1 webpage example in the crawl are the most common with 32.11% of all the pages in the crawl.

4.3.1.2 Distribution of Annotations

Schema.org defines a range of subtypes of events. 18 out of the 19 defined event types are being used in the Common Crawl, where *CourseInstance* is the only event type that is not being used. Table 4.3 presents the distribution of event types in the extracted data is presented. 87% of the event scopes do not have a specific event type specified and are annotated as an *event*. It can be assumed that this is either the case because i)

website owners are uncertain about determining the event type, or ii) that the available event types are not suitable for the types of events that are being presented and annotated on the web.

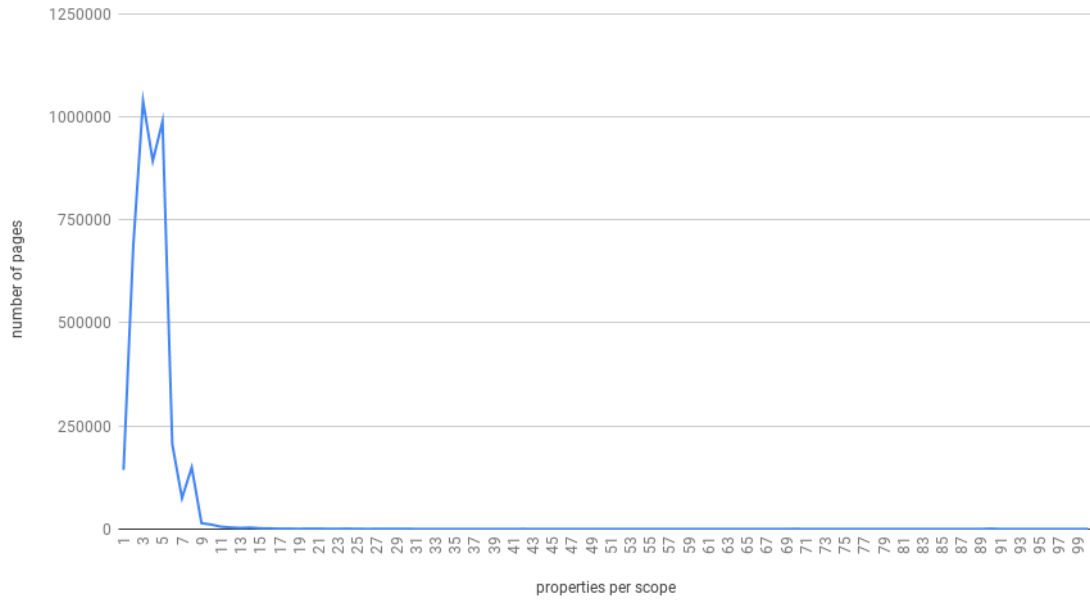


Figure 4.5: Distribution of properties per scope for the webpages in our dataset.

Range: number of pages per website	Number of websites	Percentage of Common Crawl event pages
1-10	76098	67.99%
11-20	11825	10.57%
21-30	5720	5.11%
31-40	3515	3.14%
41-50	2500	2.23%
51-100	5806	5.19%
101-200	3600	3.22%
200+	2860	2.56%

Table 4.2: Distribution ranges of the number of pages per website

Event type	Count	Percentage
event	3819627	87.00%
publicationevent	156401	3.56 %
sportsevent	138868	3.16 %
educationevent	67115	1.53 %
musicevent	66811	1.52 %
saleevent	25904	0.59 %
theaterevent	24410	0.56 %
exhibitionevent	19176	0.44 %
businessevent	16049	0.37 %
danceevent	10870	0.25 %
literaryevent	10522	0.24 %
socialevent	8322	0.19 %
festival	7852	0.18 %
foodevent	6799	0.15 %
childrensevent	5178	0.12 %
visualartsevent	4084	0.09 %
comedyevent	1252	0.03 %
screeningevent	582	0.01 %
deliveryevent	325	0.01 %

Table 4.3: Event types

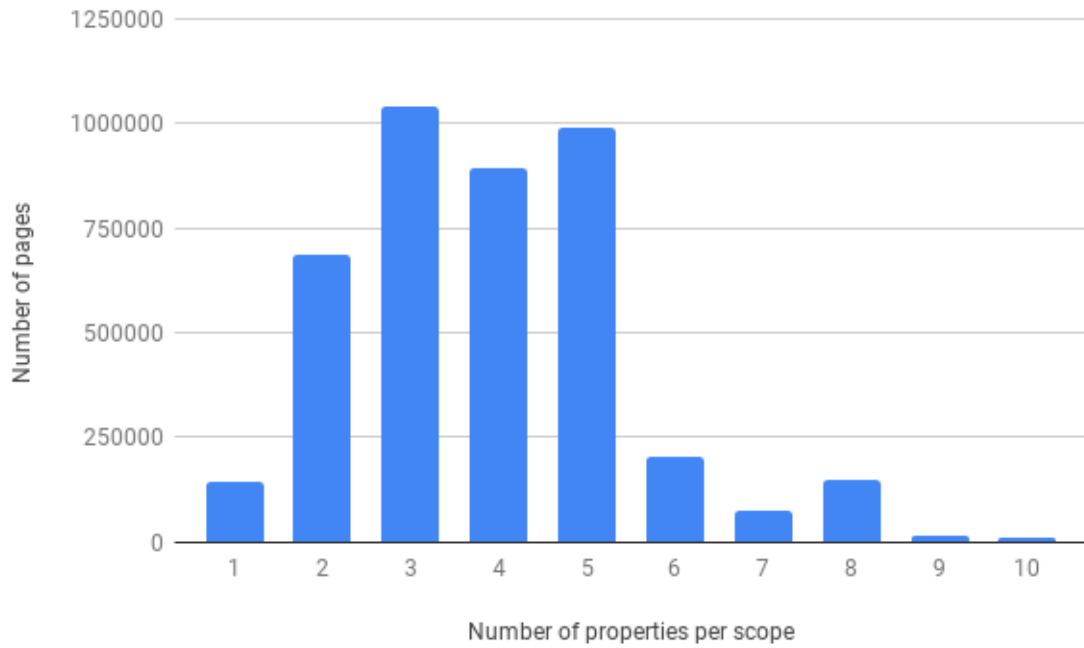


Figure 4.6: Distribution of properties per scope for those scopes that have 1-10 properties.

Properties per scope. Furthermore, we show the distribution of event properties per event scope. In Figure 4.5, one can see the number of properties per scope, where the horizontal axis represents the number of properties per scope, and the vertical axis represents the number of pages that have an event scope with such a number of scopes. 99% of scopes in the extracted Common Crawl pages have under 10 properties per scope. Figure 4.6 depicts the distribution of event properties per scope for the event scopes that have one to ten properties, which shows that the majority of scopes have 3 properties. If the number of properties per scope is significantly higher than the number of properties defined for an event by schema.org, there is an increasing likelihood we can assume misannotations or event types that do not conform to our definition of an *event* as seen in Section 2.2.

Missannotations Figure 4.7 depicts the most common properties by frequency before data cleaning. From all the event pages we extracted from the Common Crawl, we found 727 *distinct* event property names. According to the schema.org ontology, an event has only 45 different properties (including the ones inherited from “*thing*”). Thus 682 property names are misannotations or not supported for the event domain. We disregard such properties for the purpose of this method.

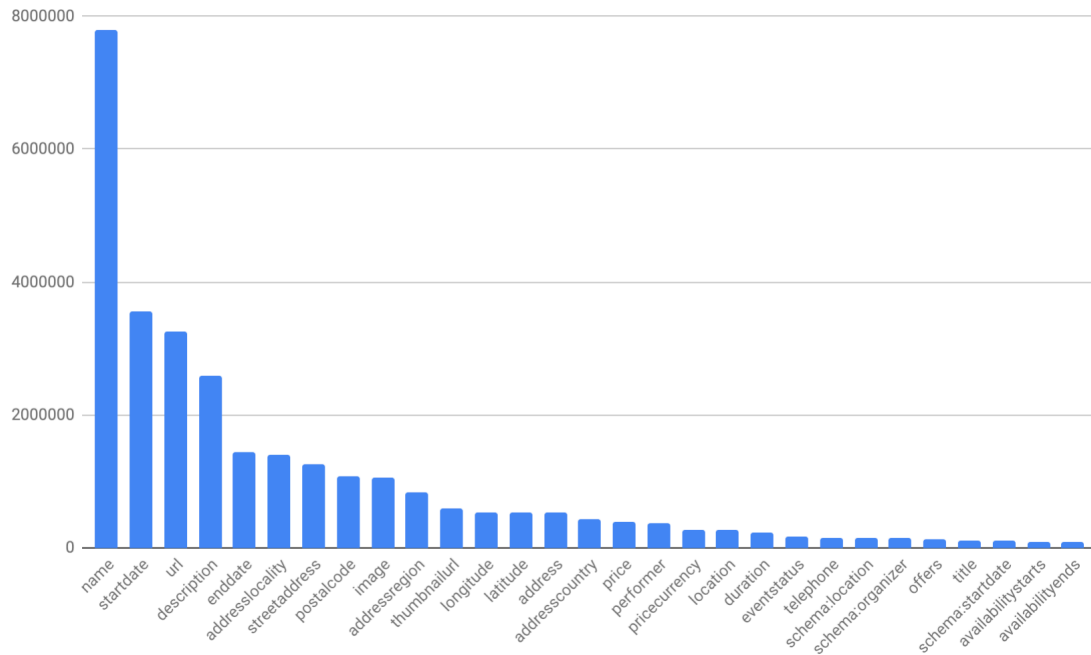


Figure 4.7: The 30 top most used property types in event pages, before data cleaning. Note that a number of them are **not** defined in the schema.org ontology.

4.3.1.3 Distribution of Relevant Event Fields

In this section, we go into more detail regarding the three event attributes that we defined as the minimum attributes for an event: *title*, *start date* and *location*.

Title. Among the three most important event attributes is the title or “*name*” of the event. As seen in Table 4.4, with the retrieved Common Crawl data, merely 70% of such annotated nodes actually contain text. However, within those non-empty nodes, 42% (2.3 million nodes) have a meta node in their ancestry, meaning that they are not visible to the user and cannot be used as a bootstrapped annotation, as we only consider event information visible to the user. Interestingly, “*name*” is the attribute with the most annotated tags with a meta tag in its ancestry.

Date. Even though dates can be easily standardised according to the ISO 8601 Standard [7], *startDate* attributes in the retrieved event pages from the Common Crawl account for the most misannotations among the three main attributes. Almost 70% of the event “*startDate*” attributes have a content attribute, far more than other attributes, most likely because dates can in theory be annotated in standardised ways. However, as seen in Figure 4.6, out of those startDates that have a non-empty content, 70% are not represented in the ISO 8601 Standard parseable format. Vice versa, 51% of all *startDate* nodes in the crawl do not have any textual content, far more than any of the other attributes. It can be assumed this is mainly due to the fact that standardised, machine readable date formats can be included in the attribute content.

“ <i>name</i> ” annotated properties	Number of properties	Percentage of total number of “ <i>name</i> ” properties
Total number	7,788,337	100.00%
Properties with content attribute empty	5,410,662	69.47%
Properties with node text content empty	2,361,658	30.32%
Properties within an invisible node	2,323,020	29.83%
Properties within node containing both content attribute and text content	16,017	0.21%

Table 4.4: Information about properties annotated as “*name*”.

“ <i>startDate</i> ” annotated properties	Number of properties	Percentage of total number of “ <i>startDate</i> ” properties
Total number	3554642	100.00 %
Properties with content attribute empty	1,070,890	30.13 %
Properties with node text content empty	1,724,058	48.50 %
Properties within an invisible node	1,637,235	46.06 %
Properties within node that contains both content attribute and text content	759,325	21.36 %

Table 4.5: Information about properties annotated as “*startDate*”.

	count of non-empty contents	count of ISO compatibles	% of non empty content
textual content	1,830,584	69,585	3.8%
attribute content	2,483,752	765,035	30.80%

Table 4.6: ISO compatibility of “*startDate*” annotated attributes.

“ <i>location</i> ” annotated properties	Number of properties	Percentage of total number of “ <i>location</i> ” properties
Total number	279514	100.00 %
Properties with content attribute empty	231,596	82.86 %
Properties with text content empty	4,5026	16.11 %
Properties within an invisible node	43,510	15.57 %
Properties within a node contains both content attribute and text content	2504	0.90 %

Table 4.7: Information about properties annotated as “*location*”.

Location. Location annotations are much less common than “*name*” and “*startDate*”. They have the smallest percentage of annotated content attributes, and vice versa the biggest proportion of non-empty text content. Apart from numeric values such as longitude and latitude, which are not typically used on event pages in a visible manner, locations are hard to standardise.

Pages containing all three relevant event attributes. Only 49,988 web pages in the Common Crawl contain an event scope with a title, date and location that is visible to a user. For those webpages, Figure 4.8, depicts the distribution of the number of pages per website. In addition, Table 4.8 shows the distribution of the event types for those pages.

event type	count	percentage
event	38,752	78.30%
businessevent	5,004	10.11%
sportsevent	3,708	7.49%
musicevent	661	1.34%
theaterevent	570	1.15%
educationevent	314	0.63%
visualartsevent	225	0.45%
childrensevent	171	0.35%
festival	37	0.07%
socialevent	25	0.05%
saleevent	14	0.03%
exhibitionevent	5	0.01%
literaryevent	5	0.01%
foodevent	1	0.00%

Table 4.8: Event types for pages that contain all three relevant event attributes.

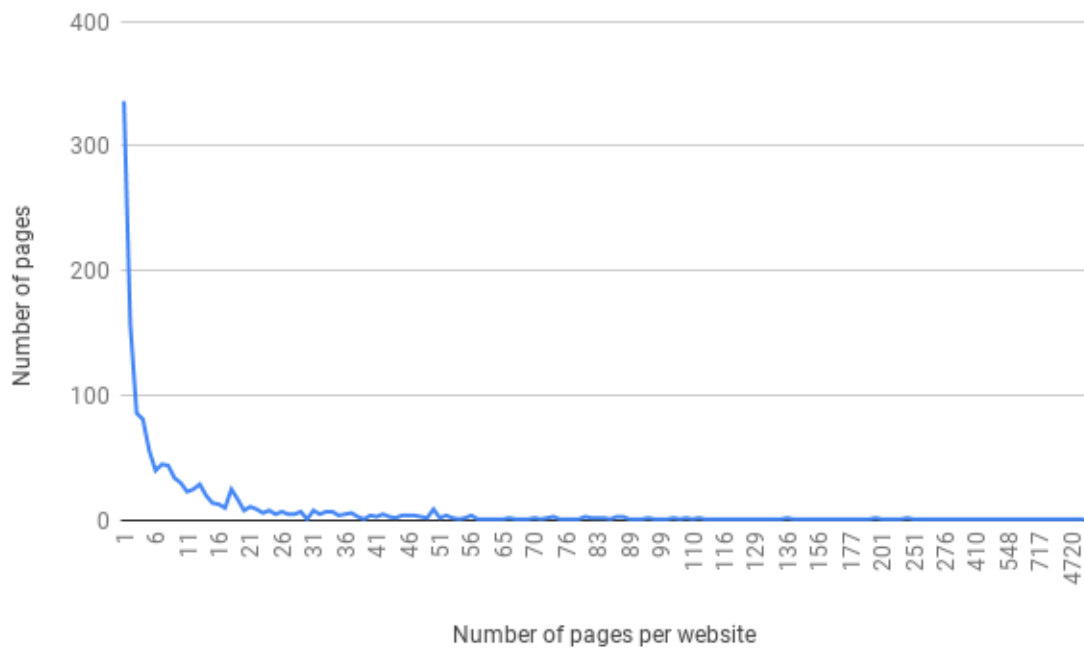


Figure 4.8: Distribution of the number of pages per websites for pages in the corpus that have all three event attributes.

4.3.2 The Resulting Dataset

Extracting Microdata attributes from the Common Crawl in the steps above result in 31,576,135 properties from 4,390,148 unique web pages, which belong to 111,924 unique domains. This initially generates a very large and noisy dataset. As explained in Section 4.2, we create a filtering mechanism to account for duplications, misannotations, annotation misuse, outdated URLs and removed annotations. We found 496 pages that would fit all criteria, but the text did not fulfil the pattern matching process, e.g. the annotated date was only a digit. We further prune the dataset to have a reasonable distribution over domain hosts, as the distribution is highly skewed due to some larger websites. The purpose here is to not bias any model or subsequent evaluations introduced thereafter towards underlying templates of pages that occur often in the dataset. We generate a dataset that randomly sampled six single event pages (that passed our filters) per website. Figure 4.9 shows the distribution of the number of pages per website after sampling. Furthermore, we filtered out pages that had more than one event scope, as this is a strong indicator that this page is a listing page on event aggregators, rather than a single entity event page as described in Section 2.2.

Further web pages, even though still online, had to be removed due to a change or removal of semantic web annotations since they were archived by the Common Crawl. For those pages, which had all three attributes when extracting from the Common Crawl, but are now missing at most one, we nevertheless use them to train the event attribute classifier, but discard them for the event assembly step. On pages we could no longer find a *title*, *date* or *location*, even though we had extracted them from the Common crawl on that basis in the first place, and at least two were remaining, we used those to train event attribute classifiers, however discarded them for event assembly. From the webpages that have all three attributes, we generate negative examples from elements that are not annotated as event attributes. This rigorous filtering step left 49,988 event pages after filtering, and 5,409 pages after normalising the highly skewed distribution by randomly sampling 6 pages per website. Table 4.9 provides an overview of the filtering steps.

Criteria	number of pages
Common Crawl (total)	3.1 Billion
Initially extracted events	4,390,148
Title missing	1,723,757
Date missing	2,945,797
Location missing	4,247,489
At least one event attribute missing	4,340,260
All three annotations present	49,988
After sampling	5,409

Table 4.9: Overview of the different filtering steps.

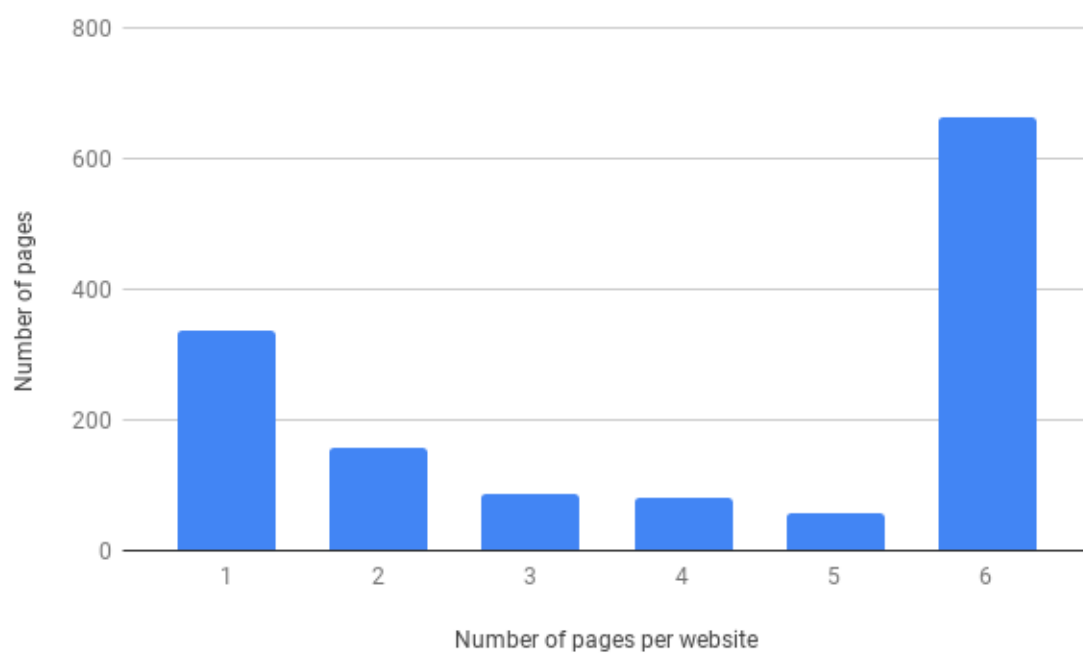


Figure 4.9: Final distribution of pages per website after sampling 6 pages per website.

Chapter 5

TIDE: Distantly-Supervised Event Extraction

This chapter investigates the problem of extracting event information from the open web without per-site supervision. Our main contribution is the novel system TIDE (“Template-Independent Data Extraction”), a multi-stepped learning and extraction approach for **event data extraction**. The system is built on the bootstrapping mechanism to minimise the cost of acquiring labelled event training data described in Chapter 4.

TIDE differs from traditional data extraction methods, as surveyed in Chapter 3. Such classical data extraction methods rely on identifying common structures in the template within single entity pages of the same website. Our novel event data extraction framework, TIDE, overcomes this shortcoming, as it learns to recognise event information on single entity pages using visual, textual and structural features. These features are used for two classification steps: Given a webpage URL, we generate candidate elements, for which the event attribute classifier predicts whether this candidate is an event attribute. The event assembly step then combines classified candidate attributes into candidate event objects, each containing a title, date and location, as formally defined as Σ_0 in Section 2.2. Finally, the event classifier returns the object with the highest likelihood of being the event on the single entity page. Figure 5.1 present an overview of the system.

Our main contributions are:

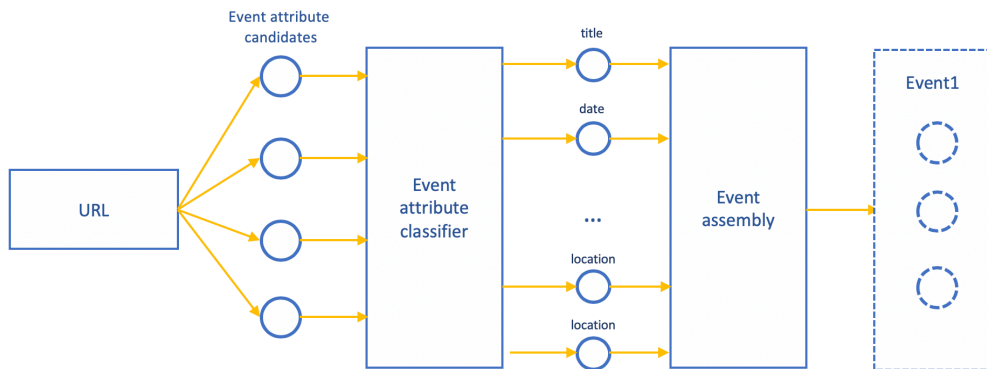


Figure 5.1: High level overview of TIDE.

- An end-to-end event extraction framework called TIDE, which is only distantly supervised and built on the data acquisition framework introduced in Chapter 4.
- TIDE leverages common characteristics (phenomenology) of the event webspace and translates these into visual, textual and structural features.
- A two step classification mechanism, which first exploits information of event attributes in isolation, and then in combination.
- An empirical evaluation of the results and demonstration of how TIDE extracts events `<title, date, location>` with 93% accuracy. We further evaluate the performance of the different steps.

Chapter organisation. We begin by introducing the systems’ overall architecture in Section 5.1. We then describe in more depth the three main aspects of the system, namely the single entity level classification in Section 5.2, candidate selection in Section 5.3 and event assembly in Section 5.4. We present the experimental evaluation in Section 5.6.

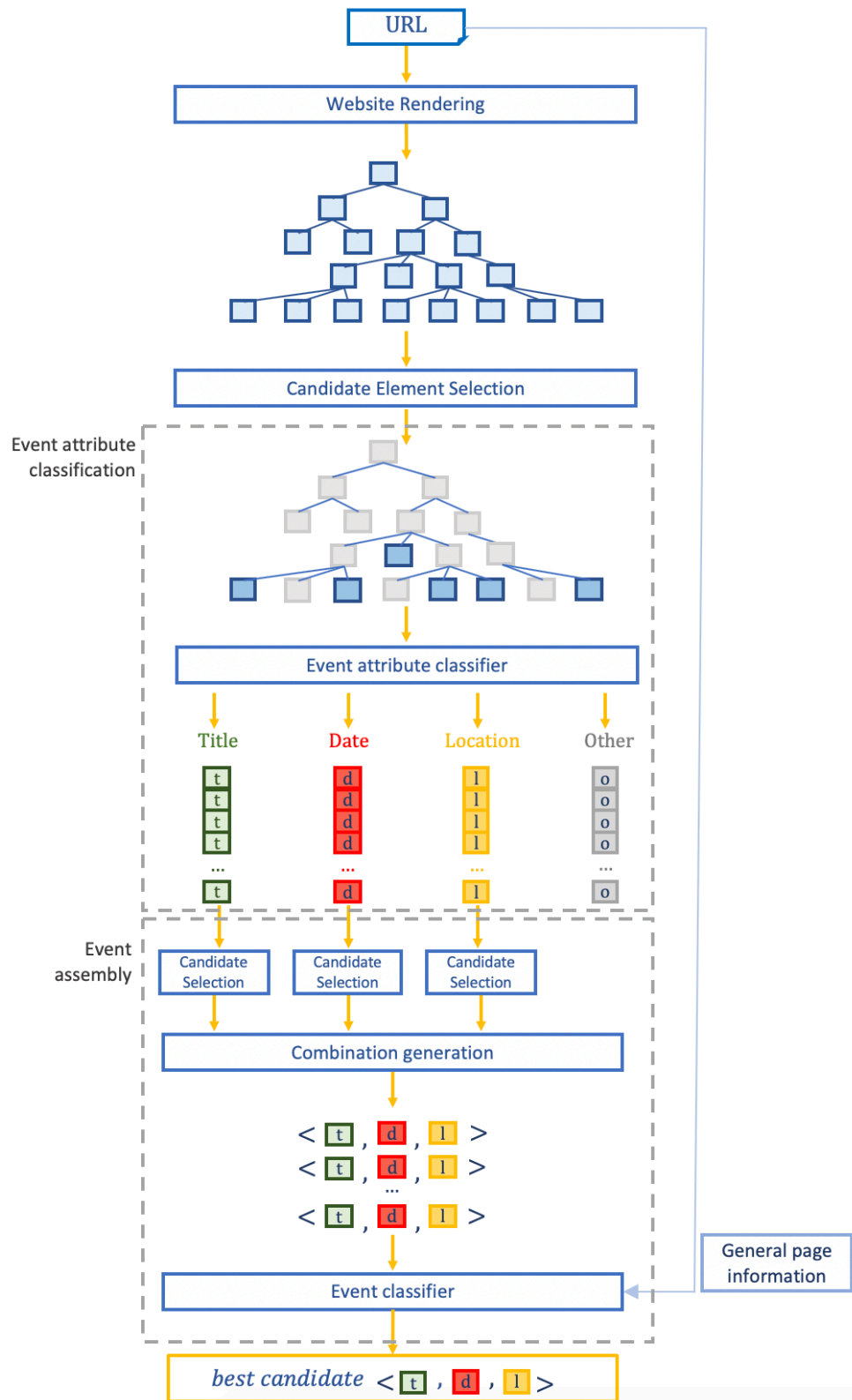


Figure 5.2: TIDE: Overall system architecture.

5.1 A Template-Independent Approach

TIDE’s pipeline consists of five components, including two explicit classification steps, as depicted in Figure 5.2: The first classification step predicts the likelihood of a DOM candidate element being an event attribute. The second classification steps takes a candidate event as an input. This candidate contains elements given a high probability in the first classification round. The intermediate components are preprocessing and feature generation steps for these two classification steps. We next give a high level overview into each component herein and then describe each component in more detail in the following sections.

The first component is a **preprocessing** step that generates features for the first classification step. It takes as input a URL which is rendered. It then selects candidate text nodes based on a number of textual and visual heuristics: A candidate event attribute node is a DOM element X , such that

- it is a leaf node in the DOM tree (i.e. `[count(child:*)=0]`).
- its text content character length is between 1 and 100 (i.e. `[string-length(text(.))>0 and string-length(text(.))<100]`).
- it is a visible element, defined as an element that is visible to the human when the page is rendered. Specifically, it means that the node does not have meta tags in its ancestry or CSS attributes that render it invisible.

Each candidate element is mapped into a feature vector, which subsequently is an input to the **element level classification step**. For each element e , the classifier outputs for each event attribute a , the probability of e being of type a . In the **candidate selection** component each classified element is added to a set of elements that corresponds to the attributes with the highest probability. The **combination generation** component then takes the title, date and location sets and generates all possible triples with an element from each set. We then compute pairwise dependent features. Such features encode characteristics of relations between the elements in the triple, for example the distance in pixels between a title candidate and a date candidate. The pairwise dependent features, in combination with the candidates probabilities and page features, are the input into the **event assembly step**. We return the event object with the highest probability of being of type `<title, date, location>` for that particular single event page. The final output of the system are structured event records with the respective attributes `<title, date, location>`, as seen in the example in Table 5.1.

url	title	date	location
www.ticketmaster.at/event/classic-ensemble-vienna-tickets/12111	Classic Ensemble Vienna	13 May 2018 20:00	Perterskirche Vienna
https://www.eventbrite.co.uk/e/oxfords-christmas-light-festival-tickets-48590273817	Oxford’s Christmas Light Festival	Fri, November 16, 2018 7:00 PM – 10:00 PM GMT	Oxford University Museum of Natural History Parks Road Oxford OX1 3PW
https://events.stanford.edu/events/802/80218/	The Chemistry of Art	Friday, October 19, 2018	Sapp Center Auditorium, Room 111

Table 5.1: Example output of the TIDE system

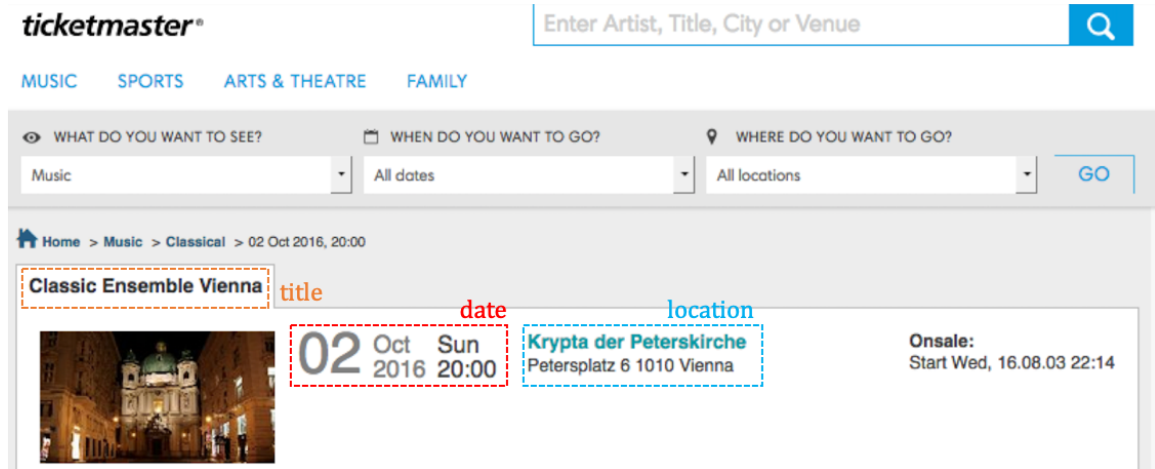


Figure 5.3: Example event page.

5.2 Element Level Classification

This section addresses the element level classification step, which follows the preprocessing step, as depicted in Figure 5.2. Each candidate element previously selected in the preprocessing stage is evaluated in a first classification step. In this process, TIDE looks at elements individually and determines their probability of being an event attribute. Each candidate element is mapped into a feature vector, which captures textual, structural and visual properties of this element. We proceed with a detailed explanation of the feature categories, and then describe the classification process.

5.2.1 Textual Features

Textual features encode linguistic characteristics that can be exploited through shallow NLP techniques. For this purpose, we use features describing lexical syntax patterns. TIDE computes the following textual features for the text content of a given candidate DOM element:

- **T1-T5 - Textual statistics.** Numeric. Textual features 1-5 encode the following basic textual statistics based on the unicode standard [11]: the number of characters, words, lines, white spaces and numeric characters. These features are based on the observation that the text for event attributes usually has similar characteristics. For instance, titles are relatively short, whereas dates and locations typically have numeric characters, such as house numbers, post codes for locations and *dd/mm/yy* patterns for dates.
- **T6 - Title casing.** Boolean. This feature encodes whether the element text is in title case. Event locations are often title cased, as they are proper nouns, and titles are usually emphasised through capital or title casing.

- **T7 - Date parser.** Numeric. We use a date and time tagger [105, 106] to use this feature. The value is generated using the proportion of the element text that the tagger tagged as date or time. We normalise this value over the length of the element text content.
- **T8 - NER Tagger.** Categorical. We use Stanford Named Entity Recogniser (NER) [44] to tag text sequences with the labels *person*, *organisation* and *locations*. We encode whether the candidate text content is tagged with the respective label or “none”. This is particularly useful with location attributes that will be annotated as a “location” entity. Furthermore, if a candidate text has several non-event related NER labels, this is a strong indicator that this element is not an event attribute.
- **T9 -T14 - Keywords.** Boolean. The features encode whether specific keywords for date and location occur in the candidate text, parent text and grandparent text. For date, we consider keywords such as weekdays and months, e.g. “Monday” or “October”. We also take into consideration, abbreviations such as “Sun” for *Sunday* and “Oct” for *October*. For location we consider keywords such as “street” and “avenue”. Titles do not have common keywords as event titles usually only contain keywords of their subdomains (i.e. a sporting event might have words such as “run” or “hike”).

Example 5.2.1 *As seen in Figure 5.3, the title and location are completely title cased, whereas the description is not. The date contains significantly more numbers than other elements on the page, and contain keywords “Oct” and “Sun”.*

5.2.2 Structural Features

Structural features represent information extracted from the underlying template of the website. More specifically, these features relate to the candidate element, their ancestry and siblings. Insightful information can be generated from the element tags and attributes for different event attributes. We consider the following structural features for a given candidate DOM element:

- **S1 - DOM element name.** Categorical. Element tag names denote the type of the element. The element names often have a semantic meaning that imply the role of the specific element within the page. We therefore use the tag name of a candidate element as a categorical feature.

Example 5.2.2 *For instance, if an event page contains a <h1> element node, it is more likely to be a title, whereas a linkable element <a> is rarely used for any event attribute (but tends to be used for the navigation bar, path, and side bar of the page).*

- **S2 - Element attributes.** Boolean. Web elements contain attributes that provide additional information about the element and define a desired behaviour. The value of such attributes can be a strong indicator of the semantics of the element content. We therefore encode as features whether event keywords are present in the attributes of the element, their parents, and their siblings.

Example 5.2.3 The element in Figure 5.4 containing the event date information has the `<time>` tag name, and the attribute `data-automation` contains the event keywords “event-details-time”.

```
▼<time class="clrfix" data-automation="event-details-time">  
  <p>Tue 6 November 2018</p>  
  <p>17:45 – 18:45 GMT</p>
```

Figure 5.4: Example of event date and time.

- **S3 - Tree depth.** Numeric. The number of ancestor nodes in the DOM tree. The position of a candidate element within the DOM tree can be a good indicator of the approximate location with the page.

5.2.3 Visual Features

Visual features are based on the visual representation of the elements to be classified with regards to the rendered web page. Such information can be very powerful since event attributes have distinct characteristics that can be exploited: websites are designed to be read by a human, not a machine, therefore their design is optimised for the human eye. For instance, the title of the event is usually emphasised visually, using cues such as size, boldness and colour. It tends to be in certain distance of the top left corner of the web page and of relative bigger appearance to the rest of the information displayed. The remaining event attributes are typically within proximity and often aligned with the title. Similar to Wang et al. [109] in the news domain, we exploit such features to encode particular characteristics of the presentation of attributes in the events domain.

Visual information is accessed through CSS properties, which, unless otherwise specified, is following the CSS2 Standard [2]. We compute the following visual features for a given candidate attribute:

- **V1 - Font size.** Numeric. The font size of the textual content of the candidate element.
- **Font weight.** Numeric. The boldness of the textual content of the candidate element.
- **V2 - Colour.** Categorical. The colour of the textural content of the candidate element.
- **Height.** Numeric. The height of the bounding box of a candidate element.
- **V3 - Width.** Numeric. The width of the bounding box of a candidate element.
- **V4 - X value.** Numeric. Given the bounding rectangle of a candidate element, the value x measures the distance between the left border of the page to the left edge of the box in terms of pixels.
- **V5 - Y value.** Numeric. Given the bounding rectangle of a candidate element, the value y measures the distance between the top of the web page to the top edge of the box in terms of pixels.

Algorithm 1: Element Level Candidate Selection

Input: Set of candidate elements C , set of attributes Σ_0 , and constant k .

Result: Candidate set for each event attribute $\alpha \in \Sigma_0$.

```
1  $pageLevelMax = 0$ ;  $\lambda = other$ ;  $\gamma = null$ ;  
2 for  $c \in C$  do  
3    $classify(c)$ ;  
4    $elementLevelMax = \max_{\alpha' \in \Sigma_0} P_{\alpha'}(c)$ ;  
5   for  $\alpha \in \Sigma_0$  do  
6     if  $P_{\alpha}(c) > pageLevelMax$  then  
7        $pageLevelMax = P_{\alpha}(c)$ ;  
8        $\lambda = \alpha$ ;  
9        $\gamma = c$ ;  
10    end  
11    if  $P_{\alpha}(c) \geq elementLevelMax - k$  then  
12       $candidates_{\alpha}.add(c)$ ;  
13    end  
14  end  
15 end  
16  $candidates_{\lambda}.add(\gamma)$ ;
```

5.2.3.1 Attribute Level Classification.

In order to classify candidate elements, we train a multi-class random forest model [30] with 100 trees with a bag size of 100% of the training data, to predict the likelihood that an element is a specific event attribute. Our choice for multi-class classification, as opposed to a binary classifier for each event attribute, is based on the observation by Foley et al. [46] that an attribute's best negative examples are other attributes.

As several web elements could be tagged for the same attribute, it is important at this stage to use a probabilistic output to determine the likelihood an element is of a specific attribute type (e.g. *title*, *date* or *location*), as opposed to other attributes. For example, there could be several dates on a website: the event date, the date when tickets go on sale, and the copyright date at the bottom of the website. While all three elements are dates, we are only interested in finding the event date.

Hence, when an event attribute is classified, we store its probability distribution, and how their likelihood for each event attributes compares to i) the likelihood of being a different attribute, and ii) the likelihood other candidates on the page are of that type of event attribute. If an element has the highest probability are a certain attribute out of all the elements on the page, it is annotated as such. For example, if an element has the highest probability of being the event date attribute out of all elements on the page, it is the *page level maximum* for attribute date.

5.3 Candidate Selection for Event Announcements

Based on the probabilities created by the element level classifier, we create a set of elements for each event attribute. These sets are the input to the event assembly step and they are computed with Algorithm 1.

Let $\Sigma_0 = \{title, date, location, other\}$ be the set of attribute types and $P_{\alpha}(e) \in [0, 1]$ be the probability of element $e \in DOM$ being of type $\alpha \in \Sigma_0$ that is returned by the element classifier. Then, for a constant

$k \in (0, 1)$, we define a set $candidates_\alpha$ for each $\alpha \in \Sigma_0$:

$$candidates_\alpha = \{(e, \alpha) : P_\alpha(e) \geq \max_{\alpha' \in \Sigma_0} (P_{\alpha'}(e)) - k \vee P_\alpha(e) = \max_{e' \in DOM} (P_\alpha(e'))\}$$

More specifically, an element e is added to the set $candidates_\alpha$, if it is a **element level maximum**, **runner-up**, or **page level maximum** for α :

- (1) An element e is an **element level maximum** for attribute α , if $P_\alpha(e) = \max_{\alpha' \in \Sigma_0} P_{\alpha'}(e)$, i.e. e is most likely of type α .
- (2) An element e is a **runner-up** for attribute α if the difference between $P_\alpha(e)$ and the **element level maximum** is at most a constant k . We use **runner-ups** to account for potentially false classifications. We empirically evaluated different values for k in Section 5.6.2.1, which exemplifies the trade off between not losing potential event attributes early on in the event assembly and introducing too much noise.
- (3) An element e is a **page level maximum** for α , if $P_\alpha(e) = \max_{e' \in DOM} P_\alpha(e')$, i.e. $P_\alpha(e)$ is the highest probability for α for all elements $e \in DOM$. Note that this element might not be the **element level maximum** for α .

Example Consider the element e and assume that the element level classifier returns the following probabilities: $P_{title}(e) = 0.4$, $P_{date}(e) = 0.02$, $P_{location}(e) = 0.38$, and $P_{other}(e) = 0.2$. Let $k = 0.2$. Element e is an element level maximum for title, because $P_{title}(e)$ is the highest among all 4 attributes. In addition, e is a runner up for location, because $P_{title}(e) - P_{location}(e) \leq k$. We therefore add e to both the title and location candidate sets.

All tags in $candidates_{other}$ or that have not been selected as a **elementLevelMax**, **pageLevelMax** or **runner-up** are discarded at this point and not considered for the event assembly step.

Combination Generation Once the candidate sets for each attribute are populated, the system generates one candidate event object for all possible combinations of one element from each candidates set. If is possible that the same element occurs in two different candidate sets, then it may appear only once in a given event object. For example, if an element is an **element level maximum** for title but also a **runner up** for location, then the element can only be considered for either the title or location attribute in a single event object, but not both.

5.4 Event Assembly

At the event assembly stage, TIDE looks at combinations of candidate event objects and computes features that capture the relation between the attributes of this object. These characteristics include features such as proximity, alignment, and relative position between candidates in the page. For instance, the example of the

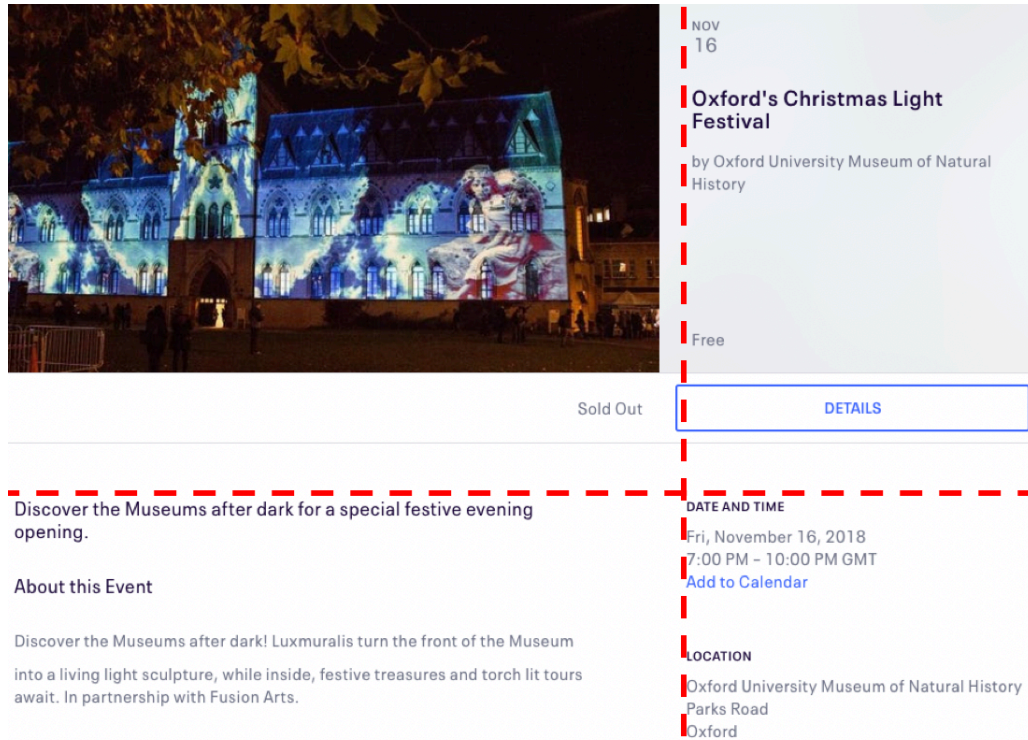


Figure 5.5: Example of a single event page with gridlines to highlight the alignment of event attributes. The title is vertically aligned to the date and location and the date is horizontally aligned with the description.

```

▼<div class="event-details hide-small">
  <h3 class="label-primary l-mar-bot-2" data-automation="listing-info-language">Date and Time</h3>
  ▼<div class="event-details__data">
    <meta content="2018-11-16T19:00:00+00:00">
    <meta content="2018-11-16T22:00:00+00:00">
    ▼<time class="clrfix" data-automation="event-details-time">
      <p>Fri, November 16, 2018</p>
      <p>7:00 PM - 10:00 PM GMT</p>

```

Figure 5.6: Part of the source HTML code of the page in Figure 5.5.

single event page in Figure 5.5 shows that event information is usually displayed in relation to each other. The title is usually found to be more prominent than other attributes, whereas the attributes overall are usually found within a certain alignment and proximity to each other in the first screen of the page.

TIDE computes the following 16 relational features for a given event object that consists of a title, date, and location attribute:

- **P 1-6 - Horizontal & vertical alignment.** The features are based on the horizontal and vertical alignment of attributes nodes, their parent and grandparent within a candidate event. Vertical and horizontal alignment play an important role in determining how elements relate to each other in a web page. Elements containing event attributes are usually aligned. While the event title is often more prominently displayed on the page, the remaining event attributes are usually horizontally or vertically aligned to the title. For instance, in Figure 5.5, the title of the event *Oxford Christmas Light Festival* is vertically

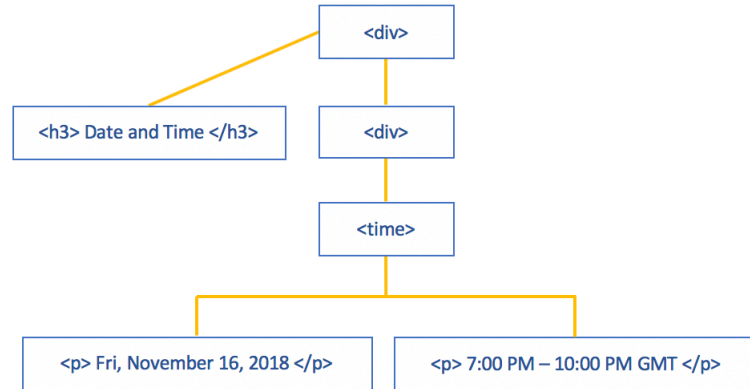


Figure 5.7: The structure of the HTML source code containing the date attribute from Figure 5.5.

aligned with the date and location of the event. In addition, it is common to have event information displayed as key-value pairs. In this case, it is possible that the key labels are aligned, rather than the event attribute values. This can be seen in Figure 5.5, where the event description is horizontally aligned with the the label “*DATE AND TIME*”. The key labels are typically stored in close proximity to the actual value within the DOM tree, i.e. in the parent and grandparent nodes, or any siblings of those nodes. Figures 5.6 displays the part of the HTML source of the page seen in Figure 5.5, that encodes the date and time information. Similarly, Figure 5.7 depicts the structure of this HTML code. The key label “*DATE AND TIME*” is in the sibling of the grandparent node. Based on this observation, TIDE not only takes the alignment of candidates into consideration by comparing their X and Y coordinates for vertical and horizontal alignment respectively, but also looks at the alignment of their parent and grandparent nodes.

- **P 7-9 - Proximity.** The features encode the visual aggregated distances between the elements in the candidate event. Proximity and position are an important factor in recognising related event attributes. Event information is often in relatively close proximity with itself in single entity pages, as the information is usually displayed so the user finds it at first glance. The distance between two candidate elements is the euclidian distance in pixels between top left corners.
- **P 10-12 - Proportion of attributes belonging to candidate sets.** Each of the attributes in the candidate event can be a **page level maximum** and either **element level maximum** or **runner-up** for their respective attribute type *alpha*. The three features compute how many of the attribute in the candidate event are **page level maximum**, **element level maximum** and **runner-up** respectively, normalised by the number of attributes in the event.
- **P 13-15 - Textual similarity of elements.** The features encode the Levenshtein distance [76] between the text of any two combinations of event attributes. These features are based on the intuition that event attributes are typically textually dissimilar.

- **P 16 - Nearest common ancestor.** We compute the distance to the **nearest common ancestor node** within the DOM tree for all attributes in the candidate event. This is based on the observation that related event attributes are often within the a surrounding container element.

Event Assembly Classification and Ranking TIDE uses the above stated features to classify candidate event objects. We trained a random forest model, following the definition in Chapter 2, with 100 trees with a bag size of 100% of the training data. The event announcement with the highest probability is returned.

5.5 Implementation

TIDE is fully implemented in Java. We use Selenium [10] as a headless browser and Chromium Driver [3] for control and rendering.

Heideltime. We use Heideltime [105, 106] to detect temporal expressions within the element text. Heideltime is a multilingual temporal tagger that tags temporal expressions according to the TIMEX3 annotation standard [13], part of the markup language, TimeML [12]. Heideltime considers four types of temporal expressions: date, time, duration and set. For the purpose of our feature computation we only consider annotations of type date and time. The parsed results from Heideltime can sometimes be misleading due to misannotations and only partial annotations. For example, “7.30” could be annotated both as a date and as a time. We therefore use Heideltime only as a feature and not as a date extractor for events. More precisely, we use the ratio of the span length of annotated text to the element content as a numeric feature.

Stanford NER Tagger. We use the Stanford NER Tagger [44] to tag relevant entities in our candidate text. The entity is used as a feature for the element level classifier. We use the three class model, which includes *location*, *person* and *organisation*.

5.6 Experimental Evaluation

In this section, we evaluate the performance of TIDE at several stages of its pipeline and with different configurations. The overall performance of TIDE is presented in Figure 5.2., TIDE extracts event announcements from single entity pages with a 93% accuracy. The intent of evaluating with different configurations is to empirically underpin the design choices made in this chapter. We first evaluate the event attribute classifier presented in Section 5.2 by demonstrating the performance of the step in isolation and the contribution of the pre-classification filtering mechanisms to the performance. Then, we evaluate the event assembly step individually and for different thresholds at the candidate selection step.

Event attribute	precision
Overall	93.91%
Title	93.10%
Date	93.55%
Location	95.48%

Table 5.2: Overall performance of TIDE.

Event attribute	precision
Overall	68.64%
Title	66.81%
Date	79.48%
Location	59.63%

Table 5.3: Event attribute classifier performance in isolation.

Attribute	Without filter	With filter
Overall	60.88%	68.64%
Title	61.64%	66.81%
Date	70.46%	79.48%
Location	52.91%	59.63%

Table 5.4: Performance of event attribute classification with and without prefiltering

5.6.1 Attribute Level Accuracy

We initially evaluate our event attribute classifier in isolation. This step consists of the preprocessing stage and the element classification step. After all candidate elements have been processed by the first classification step, the candidates with the maximum score for each event attribute are selected, and we report the precision in Table 5.3. For this purpose, we perform a 5-fold cross validation. We cluster the event pages by website first, before allocating the website clusters randomly to the 5 different folds, while assuring an even distribution amongst the folds in terms of number of instances. This is to emphasise the template independence of this approach. It is important to ensure that the pages are not randomly allocated to different folds, but to assure that the pages are clustered by underlying template first. If pages from the same website, and therefore possibly from the same underlying template, are placed in both a training and validation set during cross validation, the results are highly biased as the model trains and evaluates on the same template.

Initial Filtering Mechanism. All elements in a DOM tree are filtered by the preprocessing step, which uses a set of heuristics that are derived from the inherent characteristics of the events domain. We demonstrate its effectiveness by computing the precision of the base step with and without a filter, as seen in Table 5.4. The filter increases the precision of the individual attributes by 7 percentage points on average. This is evident because such a filter removes a lot of noise and potential false positives with very simple heuristics.



Figure 5.8: Example of miss-classified event title. The page title is a common false positive for the event title attribute.

5.6.1.1 Error Classification

We compute the confusion matrix for the results in Table 5.5. Noticeably, the majority of falsely labeled event attributes are misclassified as **other** attributes rather than **title**, **date** or **location**. This contradicts the observation from Foley et al. [46], where the choice of a multinomial classifier implied that an event attributes best negative examples are positive examples from other attributes.

The misclassifications are due to noise on the single event pages. We will now consider in more depth common false positives for each event attribute, as depicted in Figure 5.10.

Common false positives for event title. The page logo, header or navigation bar can be mistaken for an event title due to its visual properties of being relatively bigger and more emphasised than the rest of the page. An example is shown in Figure 5.8, where the heading in dark blue (1) is a common false positive for title. Event titles also do not have strong textual features that can be exploited to distinguish them from such false positives. Exploiting the interrelated features of event attributes can therefore overcome the shortcomings of the first classification step. In Figure 5.8, the true positive (2) is clearly more aligned and closer to other event attributes.

classified as →	title	date	location	other
title	66.81%	0.00%	1.62 %	31.57%
date	0.00%	79.48%	0.09%	20.43%
location	0.00%	0.31%	59.63%	40.06%

Table 5.5: Confusion Matrix, Respective Percentage of Evaluated Dataset

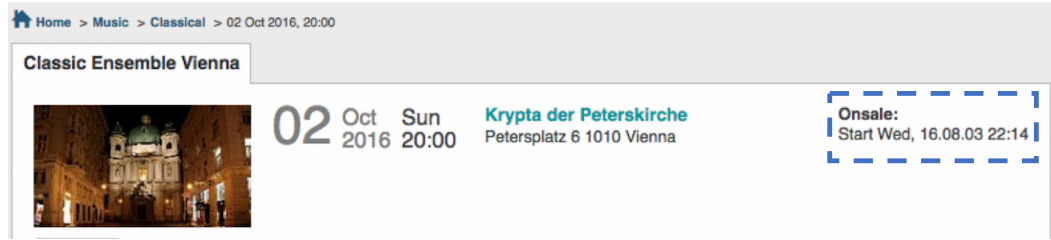


Figure 5.9: Example of typical false positive for event date.

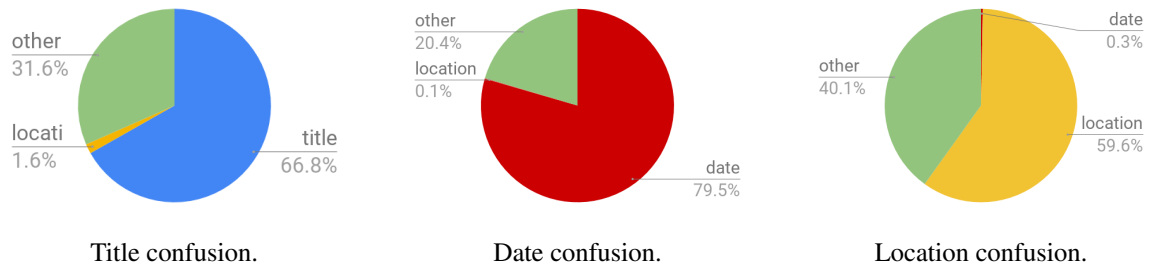


Figure 5.10: Confusion distribution for event title, date and location.

Common false positives for event date. Dates have strong inherent syntax and shape that can be exploited at the element level stage. As opposed to event titles, the event dates have much stronger textual features and less visual features to be exploited at the element level stage. This is because dates usually have a certain textual pattern that can be used, such as their higher digits to character ratio. On the other hand, their visual representation is not as striking as the event title, as they usually are the same size and style as the page average. Another complication is that single entity event pages often have related events advertised on the sides and bottom of them page, other dates can be ranked highly. Furthermore, noise, such as the `ticket sales date` in Figure 5.9, can be mistaken for the actual event date.

Common false positives for event location. The types of false positives for event location are multifold as the location attribute can vary across the event domain. For instance, event locations can vary from “*Lecture Theatre B*” to a full postal address; thus learning to recognise event locations is a hard problem, which is reflected in the relative lower precision of the location attribute in comparison to the other attributes.

The common error categories for each attribute support the case for the event assembly classification step; some of these shortcomings can be overcome by looking at interdependent features, in particular the distance and alignment of candidates.

5.6.2 Event Assembly Accuracy

In this section , we demonstrate the performance of the event assembly classification step and emphasise its value in increasing the precision of the event attribute classifier. As seen in Figure 5.11, the second classifica-

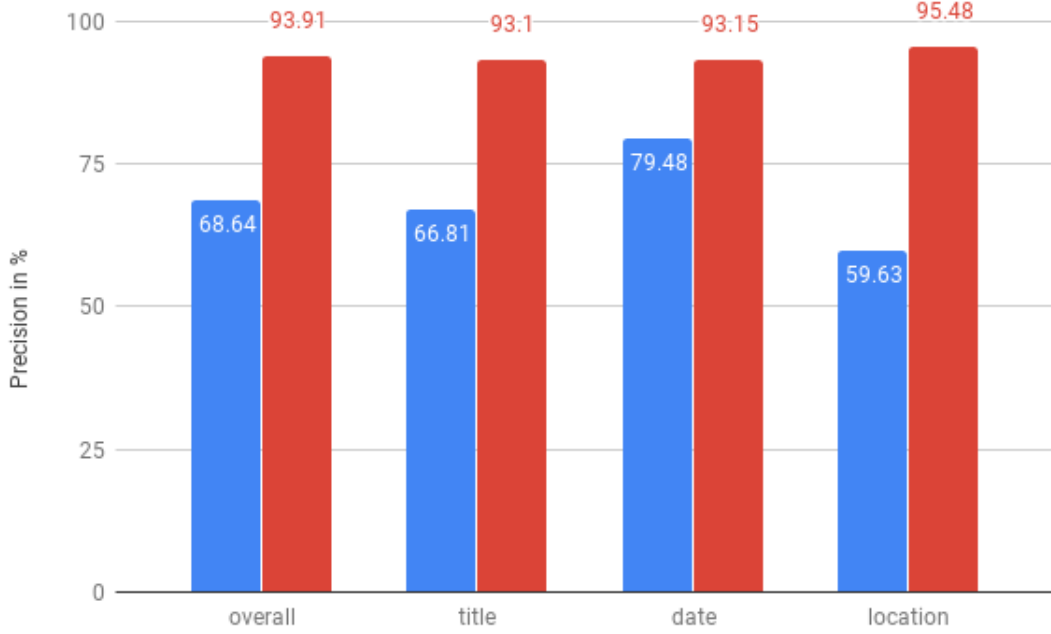


Figure 5.11: Precision before and after event assembly.

tion step significantly improves the precision of each individual event attribute by 13 to 35 percentage points and by 25 percentage points overall.

In the following two evaluations, we focus specifically on two dimensions of event assembly step: (1) the role of the threshold k as described in Section 5.3, and (2) the role of the number of negative instances used at training time.

5.6.2.1 Threshold Evaluation

In the candidate selection phase described in Section 5.3, classified elements are selected based on their performance in the element level classifier. For each element, we compute a distribution of the likelihood of an element being a title, date, location or other. Local maxima and global maxima are being selected for the different attributes. In order to not lose information at this stage, we select local runner-ups, based on a distance threshold k to proceed to the event assembly classification step. This runs the risk of introducing too much noise to the event assembly classification step and to cause the opposite effect and decrease precision. To avoid this, we evaluate the threshold k for four different values, namely 0, 0.1, 0.15 and 0.2 and selecting the top two elements. The performance for this experiment is shown in Figure 5.12 and Table 5.6. We determine that a threshold k of 0.15 is best for title and location, yet the best results for date is achieved when selecting the top 2 labels per element.

Value of k	title	date	location
0	93.06%	93.47%	95.35%
0.1	92.95%	93.32%	95.42%
0.15	93.10%	93.15%	95.48%
0.2	92.44%	92.72%	94.91%
top 2	92.87%	93.55%	94.99%

Table 5.6: Overall Performance for different thresholds k , using all negative instances on the web pages for training purposes.

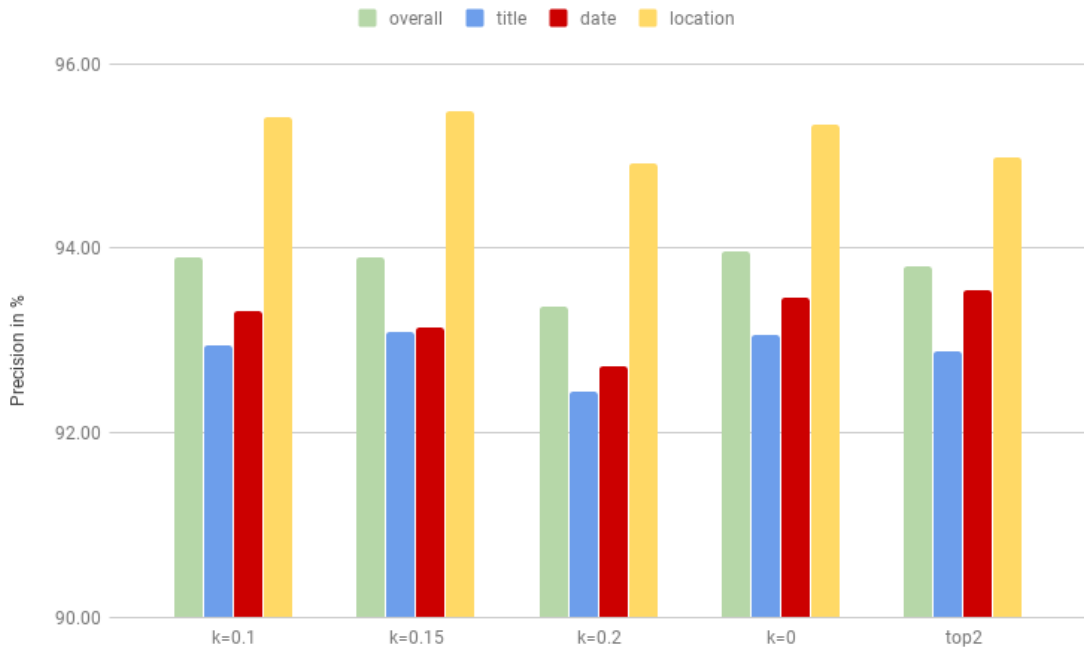


Figure 5.12: Overall Performance with different values thresholds k . Here, we use all negative examples on the web pages for training.

5.6.2.2 Number of Negative Examples

The number of negative examples used for training play a significant role in the precision of the event assembly. When we randomly sample a smaller number of negative samples, as seen in Figure 5.13 and Table 5.7, the precision stays the same or decreases from the event attribute classifier. However we observe a significant improvement if all elements annotated with a different type are considered as negative examples. While this generally does not come as a surprise, it is particularly important in the classification tasks where interdependent features of the attributes for a single event are being computed. It is vital to compute all possible combinations of attribute assemblies to achieve high precision.

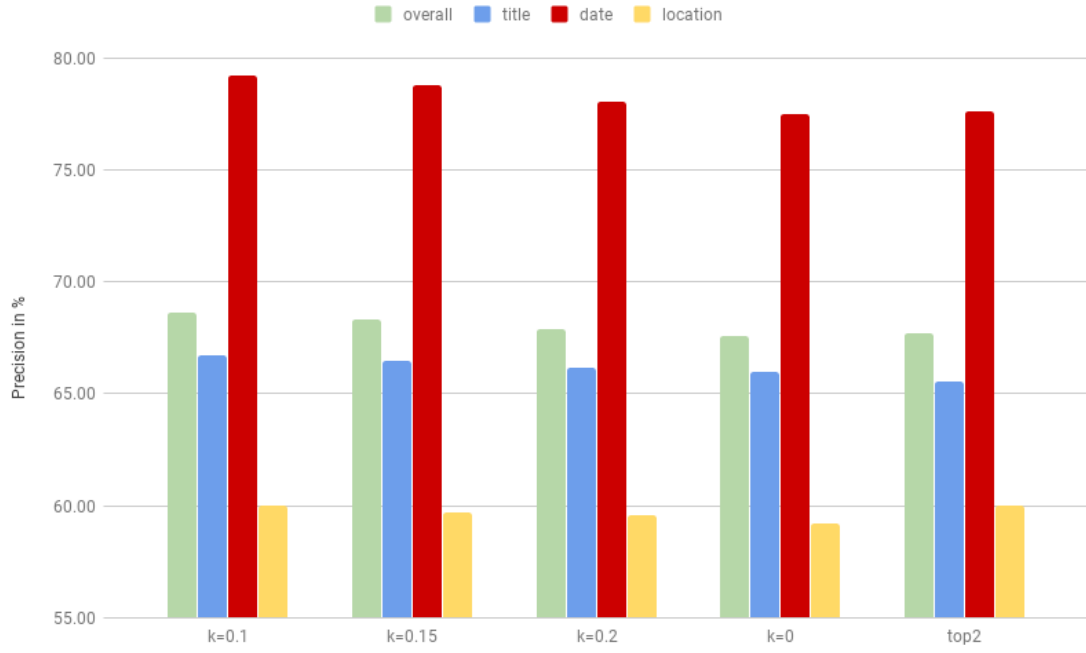


Figure 5.13: Threshold study, with same number of randomly sampled negative instances.

Value of k	title	date	location
0	66.00%	77.53%	59.17%
0.1	66.70%	79.22%	60.00%
0.15	66.47%	78.79%	59.70%
0.2	66.15%	78.04%	59.56%
top2	65.56%	77.62%	60.03%

Table 5.7: Overall Performance with different values thresholds k , using a reduced number of negative examples for training, randomly selecting the same number of negative examples as positive examples used.

Chapter 6

TIME: Template-Independent, Multi-Attribute Extraction for Events

In this chapter, we introduce TIME, the template-independent, multi-attribute event extraction framework, which differs from TIDE introduced in Chapter 5 in multiple ways. TIME uses a wide variety of more specific features which are tailored to the respective event attributes. The additional engineering effort enables the quick addition of new attributes given the extensive library of features. Thus, TIME supports an event schema beyond *title*, *date* and *location*, and extracts more granular attributes, in particular, for temporal event attributes, such as start date, start time, end date and end time. TIME has been prototyped as a standalone API and is part of the Diffbot Knowledge Graph population pipeline¹. For the purpose of this thesis, we focus on attributes formally defined as Σ_1 in Section 2.2, and depicted in Figure 6.1, suggesting further attributes for future work in Chapter 7.

Chapter organisation. We present the overall architecture in Section 6.1 and go into further details for the different event attributes in Sections 6.2-6.5. We present the experimental results in Section 6.6 and an application of TIME in industry in Section 6.7.

¹<https://www.diffbot.com/knowledge-graph/>

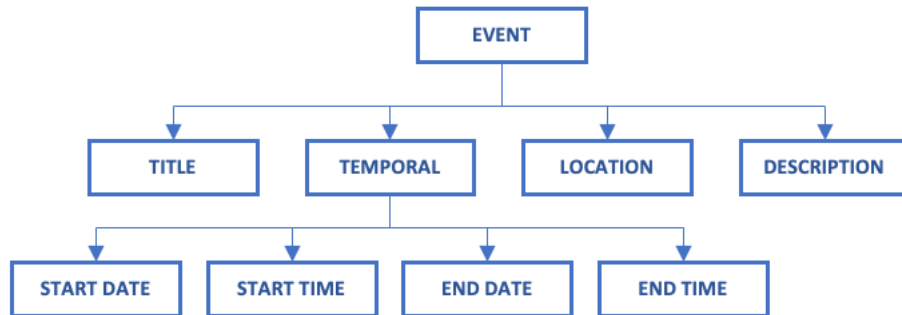


Figure 6.1: Event attributes in TIME.

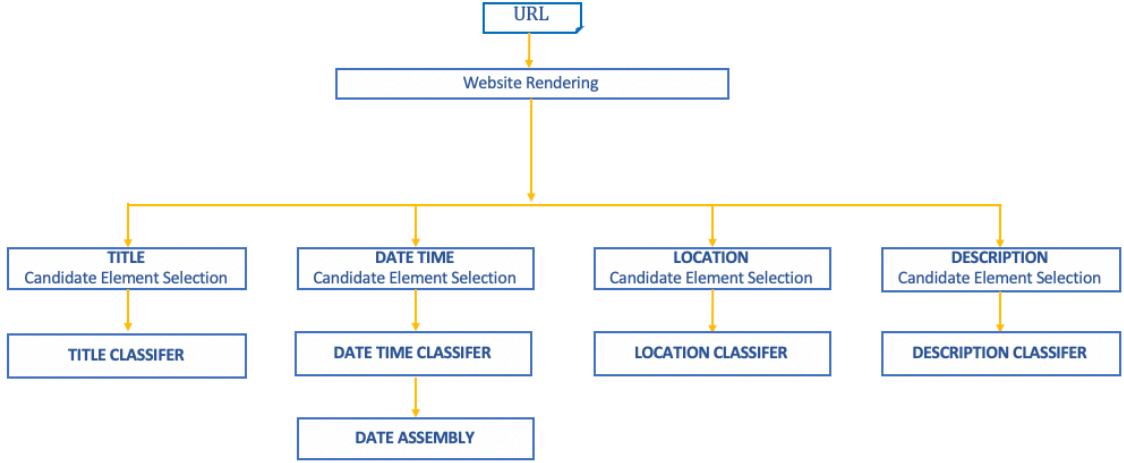


Figure 6.2: TIME pipeline overview.

6.1 System Architecture

As explained in the problem definition in Section 2.2, TIME uses one extractor for each attribute. This allows TIME to compute attribute-specific candidate elements, extract attributes in finer granularity, and exploit a larger number of attribute-specific features.

Figure 6.2 depicts the architecture of the system. Given a URL as input, TIME renders the page to access the visual features of the page. The rendered DOM tree is the input to the attribute level candidate selection processes. Each attribute extractor computes a number of candidate elements based on common characteristics of the respective event attribute. We learn to recognise each attribute based on a number of domain-wide and attribute-specific features. For temporal attributes *start date*, *start time*, *end date*, and *end time* we designed an event specific temporal extractor, which is based on the unique interrelated features of the relevant temporal attributes. After introducing event domain-wide, attribute-agnostic features in Section 6.1.1, we introduce the frameworks for event attributes in the following Sections 6.2-6.5.

6.1.1 General Features

In this section we present general, domain independent features used for all of the event attributes. We categorised these features into six main groups based on their semantics: character statistics, basic rendering, font and style, basic tree statistics, parent statistics and token statistics. All character classes are unicode [11].

6.1.1.1 Textual: Character Statistics

We capture basic character statistics of the candidate element text. The choice of such characteristics is based on the observation that event attributes vary in terms of character length, number of digits and punctuation.

- **Text 1-5 - Character frequencies.** Numeric. Features capturing frequencies of characters in the node text content. The number of characters, punctuation (" [] , . ? ! : () - / = + & % # "), capital letters, digits, and whitespace characters (considering all unicode whitespaces).

6.1.1.2 Textual: Token Statistics

The following features capture token related statistics of the node text content and use the Stanford NLP Tokenizer [83]. The count and capitalisation of tokens in candidate text vary by event attributes: event titles are usually capitalised and relatively short, whereas dates have a high proportion of numeric tokens.

- **Token 1-5 - Token count.** Numeric. We encode the number of tokens, tokens with digits, capitalised, and all capitalised tokens in the element text content as features.
- **Token 6 -9 - Token statistics.** Numeric. We encode the percentage of tokens in the element text content with specific character groups: digits, capitalised, fully capitalised.
- **Token 10 - Count of token meta description tag overlap.** Numeric. We encode the number of tokens in the candidate element text content that overlap with the content attribute of the `description` meta element of the page. Metadata contains semantic information about the page content. An overlap with such metadata suggests that the elements considered are relevant to the subject matter of the page.
- **Token 11-14 - Vocabulary based token frequencies.** Numeric. We encode the count and frequencies of stop words and “*share*” words as numeric features. We use stop words defined by Mallet [85], and we define *share* words to be the following : “share”, “facebook”, “fb”, “tweet”, “twitter”, “pin”, “follow”.
- **Token 15 - 20 - Pronoun frequencies.** Numeric. Count and percentage of first (“i”, “my”, “me”, “our”, “we”, “us”), second (“you”, “yours”), and third persons pronouns (“he”, “she”, “her”, “him”, “his”, “hers”).

6.1.1.3 Structural: Basic Tree Statistics

The following features encode statistics about the structure of the DOM tree around the candidate event attribute element. The surrounding elements of a candidate event attribute can indicate the density of information around the particular node. Based on our observation, event attributes are usually closer to the leaf nodes, containing a small number of children nodes.

- **Tree 1 - Total number of nodes in the tree.** Numeric. Number of nodes in the subtree of the element.
- **Tree 2 - Total number of text nodes in tree.** Numeric. We encode the number of text elements in the subtree of the candidate element. Event attributes are usually close to the leaf nodes of the DOM tree, thus when a candidate element contains many child nodes, it is less likely to be an event attribute.

- **Tree 3 - Total number of non-whitespace text nodes.** Numeric. Number of nodes that contain characters other than whitespaces. Textual nodes with only whitespaces are visually empty and therefore irrelevant nodes in the context of data extraction. We count the number of non-whitespace nodes in the elements subtree.
- **Tree 4-6 - Subtree.** Numeric. We encode characteristics of the underlying subtree of the candidate node. We encode the maximum subtree height, average subtree height and standard deviation of all possible paths from top to bottom of the subtree of the candidate node.

6.1.1.4 Structural: Parent Statistics

The following features encode basic rendered and structural statistics about the parent node of the candidate node.

- **Parent 1 - Number of siblings.** Numeric. We encode the number of siblings a candidate element has in the DOM tree as a feature.
- **Parent 2 - Position among siblings.** Numeric. Positioning of the candidate element among its siblings, starting from 0. For example, if a candidate has 3 siblings, and the candidate is the right-most node, then its position is 2 (counting from 0).
- **Parent 3-4 - X and Y distance of parent.** Numeric. Vertical and horizontal distance between the candidate element and the parent element's top left position, in pixels.

6.1.1.5 Visual: Basic Rendering

The following features encode characteristics regarding the geometric positioning of a candidate node that are gained from the rendered information of the page. Visual features are particularly important, as web pages are designed for humans to find event attributes at a glance. Thus, event attributes are typically relatively close and aligned with the top left corner of the page. Unless otherwise specified, the coordinates of nodes are the the top left of the bounding rectangle of the node.

- **Render 1 - X value.** Numeric. We encode the location of the candidate element along the X axis. This value is the number of pixels between the left margin of the page and the bounding rectangle.
- **Render 2 - Y value.** Numeric. We encode the location of the candidate element along the Y axis. This value is the number of pixels between the top margin of the page and the bounding rectangle top left border.
- **Render 3-4 - Dimensions.** Numeric. This feature encodes height and width in terms of pixels of the bounding rectangle.

- **Render 5-6 - Distance from right and bottom.** Numeric. The distance in terms of pixels between the right margin of the page and the candidate element and the bottom margin of the page and the candidate element.
- **Render 7-8 - Rendered size.** Numeric. We encode an approximation of the surface area of the candidate element, which is computed by the product of height and width in pixels. We use this value in absolute and relative ratio by normalising it by the surface area of the rendered page.

6.1.1.6 Visual: Font and Style

The following features encode a candidate text content’s formatting. These features are accessed through the rendered information of a page.

- **Style 1-9 - Basic font measures.** Font weight, size, color, line height, decoration, style, display property and float property as defined by the CSS2.1 standard [2].

6.2 Event Title Extraction

The first attribute to be extracted on an event page is the *event title*. This is because subsequent event attributes use features that are relative to the extracted event title. The title of an event is usually prominently displayed, as it describes the main entity of the page. Visual cues therefore play a substantial role in the identification of the event title, as it is being visually emphasised for users to notice at a glance. Such visual cues are mainly exploited through either textual emphasis, i.e. capitalisation, or features that can be accessed through rendered properties of the page such as size, colour and position. Event titles come in many variations, and are often not the biggest title of the page. A *general* page title classifier as seen in the news domain [109] is therefore not applicable.

6.2.1 Candidate Selection for Title Extraction

The goal of the candidate selection process is to find suitable DOM tree elements that could contain the title attribute. To do so, we first consider text nodes, which satisfy basic heuristics that are inherent to title attributes. For each such node, we select as a candidate element the container node, which contains this text node and potentially other non-textual nodes. We use the container as the candidate, as opposed to, the node itself, because the surrounding non-textual nodes often contain important structural characteristics that we exploit in the feature computation step for this title candidate.

More specifically, given a rendered DOM tree, we compute the following set S of textual nodes:

$$S = \{n \mid n \in DOM \wedge n \text{ is text} \wedge n \text{ is visible} \wedge minLength < n.textLength < maxLength\}$$

In words, S contains a node $n \in DOM$ if n has a text content of a certain length and is visible to the user. In practice, we observed that a common title attribute has a text length that ranges between 4 and 50 characters.

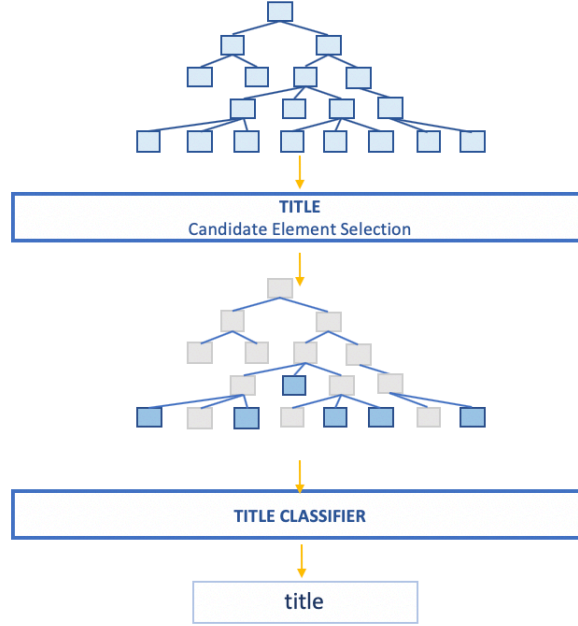


Figure 6.3: TIME Title architecture.

For each $n \in S$, we then find the container element $n' \in DOM$ that is the highest ancestor of n , such that n' has the same text content as n . We denote this step **greatest-same-content-ancestor** and define it using standard XPath [14] notation:

$$\begin{aligned} &\text{greatest-same-content-ancestor}(n) \\ &:= \$n/\text{ancestor}::*[\text{normalize-text}(\cdot) = \text{normalize-text}(\$n)] [1] \end{aligned} \quad (6.1)$$

For a given container element n' , we verify if the subtree rooted by n' contains any of the following negative element tags: `input`, `iframe`, `script`, `noscript`, `head`, `button` and `style`. If any such negative tag is found in the subtree, we can confidently assume that the subtree does not contain relevant information. If no such negative tag is found, we add n' as a candidate node for the subsequent title classification step.

6.2.2 Title Specific Features

In order to correctly identify the event title in a given page, the candidate elements are transformed into vectors for classification. We use event attribute independent features as described in Section 6.1.1. In addition, we designed a number of *event title* specific features that represent the unique **structural** characteristics of event titles.

6.2.2.1 General Features for the Title Attribute

- **T 1 - URL similarity.** Numeric. We further compare the candidate event title with the URL of the page and encode the Levenshtein distance [76] as a numeric feature. Often, the website URL path contains the event name.
- **T 2 - Candidate contains website host name.** Boolean. Binary encoding of whether the candidate element text contains the website's host name. This is a particularly relevant feature for event titles, as a common false positive is the page title or menu item in the page. As shown in a previous example in Figure 5.8, the false positive "*WKU Events*" contains the host name of the website, *wku.edu*.
- **T 3-4 - Element Tag.** Categorical. We consider the element tag of the candidate and parent as a categorical feature. Common tags for event titles are title tags `<h1>`, . . . `<h9>`.
- **T 5 - Link.** Boolean. Webpages by nature contain many links. Event information usually is not hyperlinked, as the main information about the event is on the event page itself. We therefore encode whether the attribute tag is a link, additionally to the element tag category in the previous feature.

6.2.2.2 HTML Title Specific Features

The following features relate to the HTML title of the document.

- **T 6-10 - Similarity to HTML title.** Numeric. Overlap of candidate text with HTML title, in terms of Levenshtein distance, tokens, punctuation count, stop word count and character length. HTML pages contain a special `<title>` element in the header, often used as the table and window title for the page. We compare the candidate element text with the HTML title of the page. Since single entity event pages are dedicated to the single event we are trying to extract, there is a high correlation between the HTML title and the event title.

6.2.2.3 Dictionary-based Features

The following features relate to the appearance of keywords with the element text, node and attributes.

- **T 11-15 - Dictionary based structural features.** Boolean. Binary features that captures whether the words "*title*" or "*name*" appear in the element's, the parent's and the grandparent's attributes or text content. Sometimes, event information is presented in form of key value pair, visually similar to a tabular structure. We therefore encode the presence of the words *name* or *title* in the parent and grandparent node of the text, as the key value is typically in close proximity within the tree.

6.2.2.4 Ancestry-Based Features

The following features relate to the ancestry of the candidate element, namely the elements between the root node of the DOM tree and the candidate element.

- **T 16-17 - Negative Element in ancestry.** Boolean. Event titles are usually visually emphasised for the user to find at a glance. However, website headings, menu bars and navigation bars are equally emphasised, and can therefore be mistaken as the event title. We perform a binary encoding as whether the `<nav>` or `<menu>` element tag is contained in a candidate’s direct ancestry, as this is a strong indicator that the candidate element is actually part of the page rather than the event information.

6.2.3 Title Classification and Ranking

We train a binary classification model based on random forests as described in Chapter 2, to predict the likelihood of a given candidate element being the event title. We configured the model to build 200 trees over random subsets of the title features and the entire training dataset. We select the candidate with the highest probability of being the event title attribute. We predict the correct title with an accuracy of 92%, as further detailed in Section 6.6.2.3.

6.3 Event Date and Time Extraction

Extracting temporal information from text such as news has received significant attention in literature [106, 105, 82, 97, 75], however such techniques are not applicable for event dates. Previous approaches to extracting dates focused on extracting dates from natural text such as news and blogs. However, temporal information in the events domain differs in two main ways: (i) dates are more complex and granular, as they could express time ranges, and (ii) such temporal information is often split into multiple DOM elements.

Semantic web annotations such as schema.org have a simplistic representation of temporal attributes in the events domain, as they consider it represented and annotated in a single node. This is reflected in TIDE where we bootstrap these annotation to extract single node date attributes. However, many event pages on the web have more granular temporal information regarding online events that is neither in a single dedicated node, nor easy to parse.

While asking the question “*when* does an event take place” seems relatively straight forward, finding the complete answer in a DOM tree turns out to be non-trivial. This is because an event date can be further segmented into four attributes:

`<start date, start time, end date, end time >`

Each attribute might be located in different DOM elements, and the context can often be inferred only when they are parsed together. The temporal event information is given by any combination of `start date` with any subset of `{end date, start time, end time }`, subject to the constraints defined in Definition 2 in Section 2.2:

- (1) `start date < end date`
- (2) `start time < end time`
- (3) `start time < end date`

The combinations above however do not take into consideration the different orders, variations and formats one could express the combinations. Months can be expressed in language such as “January”, “Janvier”, “Januar” or as a number. Similarly, the order plays a significant role: “12/06” might refer to the “twelfth of June” or the “sixth of December”. Taking years into consideration, “12/06/18” might refer to the “twelfth of June 2018”, the “sixth of December 2018” or the “eighteenth June 2012”. Unfortunately, the variation in time notations is just as multifold: While most Europeans count in a 24h system, many North Americans use a 12 hour system with notations such as “AM” and “PM” to denote the time of the day. Similarly, the letters denoting the times differ: *seven thirty in the evening* can be denoted in the following ways: 7.30, 7.30PM, 7:30, 7:30 PM, 7h30, 7h30PM, 19.30, 19:30, 19h30, 1930.

While some people denote 7.30 as a time, namely “seven thirty”, somebody else might use the same sequence of characters to denote “the thirtieth of July”. When a user sees this information on a page, they subconsciously use contextual information to determine whether “7.30” refers to a date or a time. Such information could simply be a key value “`start time: 7.30`”, in which case it is obvious to a user that “7.30” corresponds to the `start time`. Similarly, if another attribute is present amongst the event information that looks like a time, let’s say “17h00”, a user automatically rules out the possibility that “7.30” is the time and infers that the event takes place on the “thirtieth of June at five o’clock in the afternoon”. An event date and time could also be as simple as “Every Monday at 4”. A human user automatically assumes that the organiser refers to “four in the afternoon”. While this kind of reasoning is intuitive for a human, it is a non-trivial problem to infer such implicit domain knowledge automatically.

Underlying structure. An additional challenge for the problem of date-time extraction is that the underlying structure that captures combinations of temporal attributes can differ vastly. For instance, a single temporal attribute, such as `start date`, can be stored in a single node or in multiple ones. On the other hand, it is possible that a combination of multiple temporal attributes is stored in a single node. For example, Figure 6.4 shows an event page and a part of its source code. The temporal attributes consist of three different nodes, with the weekday “*Samstag*” and `start time` in a respective child node of the parent’s sibling.

While our approach is template-independent and we do not base our extraction method on the repetitive structure or pages, the location within the underlying structure is important for the candidate generation step. In the case of Figure 6.4, the node encapsulating the `start date` would be `<div class="datebox">` and the node encapsulating the time would be `<div class="event__card__body__when">` with the substring “20:00”.



NOV

10

2018

SAMSTAG, 20:00

Classic Ensemble Vienna

Peterskirche, Wien

BESTPLATZBUCHUNG

*Gebühren-Informationen

Preis inkl. Gebühren. Kostenloser Versand. Weitere Infos bei Klick auf "i".



```

<div class="eventcard">
  <div class="datebox">
    <div class="datebox__month">Nov</div>
    <div class="datebox__day">10</div>
    <div class="datebox__year">2018</div>
  </div>
  <div class="eventcard__body">
    <div class="eventcard__body__when">Samstag, 20:00</div>
    <h1 class="eventcard__body__name">Classic Ensemble Vienna</h1>
    <div class="eventcard__body__venue">Peterskirche, Wien</div>
  </div>
  <i class="eventcard__icon">_</i>

```

Event page on Ticketmaster.at

Source code for the temporal information.

Figure 6.4: Example of a single event page (left) and a part of its source code (right). The start date attribute for the event is spread across four web elements.

6.3.1 Our Approach

The problem of extracting temporal attributes could have been approached from many angles. Since there are potentially four attributes, we could have created four separate extractors for each of the four temporal attributes. If, for example, the **end date** classifier would not have returned any results with a significant confidence, we could have assumed that this event did not have an **end date** on the page. However, this would not have taken into consideration interrelated features, such as the proximity of a **start date** and **start time**, which are fundamentally important cues in determining the correct combination of temporal attributes to overcome the challenges outlined above.

We argue that such interrelated features need to be taken into consideration at an early stage to be able to extract attributes that are interrelated by either being expressed as ranges or by sharing nodes in the tree.

For that purpose, we split the task into multiple steps, which considers different combinations of event attributes at different stages. As depicted in Figure 6.5, we first use linguistic heuristics to determine possible date and time candidates. It is important to consider the same string as both a time and a date. We add empty nodes to represent the scenario of a missing value, ie that the page does not contain some of the temporal attributes. We proceed to combine the date and time candidates for a multinomial classification into start, end and other. We create combinations of start and end **<date, time>** pairs with a number of hard constraints, as per definition in Section 2 in Chapter 2.2: e.g. combinations cannot have an **end time** without a **start time**, and any other temporal information without a **start date**. Where the candidates can be parsed by the date parser, we verify that the **end date** is not before the **start date** and the **end time** is not before the **start date**.

In the following sections, we present the different steps depicted in Figure 6.5 in more depth.

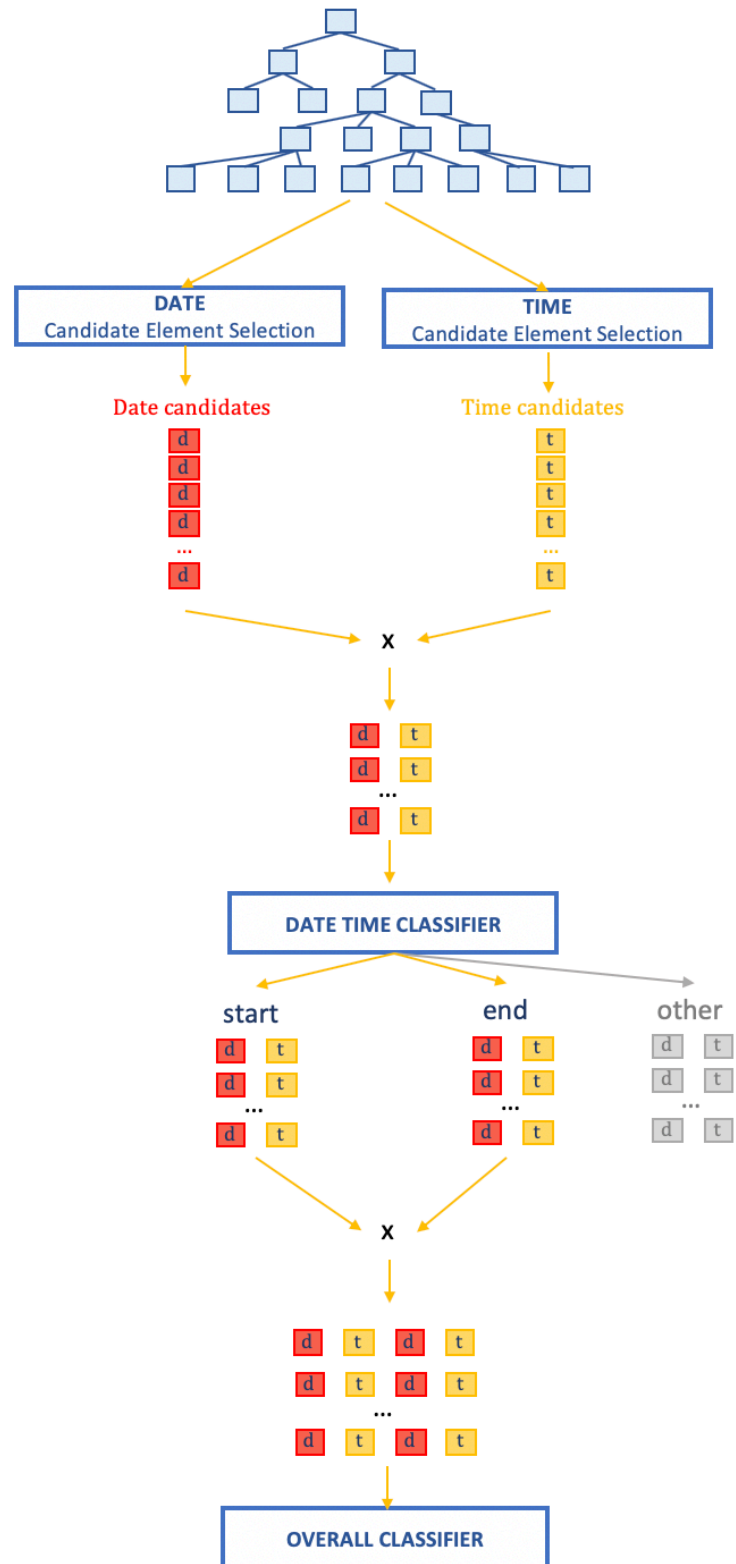


Figure 6.5: TIME Temporal attributes extraction architecture.

6.3.2 Date-Time Candidate Generation

The candidate generation phase for $\langle \text{date}, \text{time} \rangle$ pairs is designed to identify candidate temporal attributes, where the temporal information is either in a single node or spread across multiple nodes. The system identifies a DOM element as a date-time candidate, by first finding possible candidates that are date or time attributes in isolation. Let $\text{dateNodes} = \emptyset$ and $\text{timeNodes} = \emptyset$ be two empty sets of DOM element nodes. We populate dateNodes and timeNodes with candidate elements for date and respectively time attributes with the following heuristics. For a given node $n \in \text{DOM}$:

- (1) If n has text content, which includes a date keyword, such such as “*Tomorrow*” or “*Monday*”, then we add n to dateNodes .
- (2) If n has text content, which contains only digits, it could be a fragment of a date or time attribute. We use the `greatest-same-content-ancestor` function from Equation (6.1) to identify the highest ancestor container node n' that has the same text content. We then check that the subtree rooted by n' does not contain any negative tags, as was done the title candidate generator. By definition, the parent of n' contains additional text content. We use the concatenation of the text nodes in parent of n' and run a date and a time parser to tag substrings of this text as either date or time patterns. The parent of n' is added to dateNodes and timeNodes for each substring that is tag as a date or respectively time pattern. Therefore, variations of the same node can be added to the dateNodes and timeNodes multiple times.
- (3) If n has text content, which contains both digits and alphabetic characters, we run the time and date parsers over the text content of n . We add a node for each substring of the text that has been tagged as either a date or time pattern to dateNodes and respectively timeNodes .

A final date-time candidate element is a $\langle \text{date}, \text{time} \rangle$ pair, where date is a date candidate in dateNodes , and time is time candidate in timeNodes . The set of date-time candidates contains one elements for each possible pairwise combination of the nodes in dateNodes with the nodes in timeNodes . This set is subsequently used as an input for the date-time classifier.

A more fine-tuned heuristic to generate candidate date-time pairs to restrict the search space would not be applicable as event dates and times are generally very noisy and non standardised. As TIME supports multilingual event extraction, the variability of potential date patterns is rendered significantly more complex. Furthermore, compiling such a set of patterns would be a significant manual effort. We therefore argue that the approach above is the best tradeoff to limit the search space while achieving a maximum candidate recall.

6.3.3 Classifying $\langle \text{date}, \text{time} \rangle$ Pairs

Given the date and time candidate selection above, we encode the candidate pairs into feature vectors. The aim of this is to determine the likelihood of (i) being a date with corresponding time and (ii) being a start or

end <date, time> pair. For this purpose we build a three-class classifier with the class values *start*, *end* and *other*.

In addition to the general purpose features defined in Section 6.1.1, we define event date and time specific features. In this initial classification step, the most important features are the ones relating the elements in the pair themselves, in terms of position, appearance, and textual coherence. We group the features in textual, structural and visual features as listed in detail below:

6.3.3.1 Textual Date and Time Features

The following features encode textual characteristics of date and time candidates. Such textual features are based on the text content of candidate nodes.

Language model-based features. The following features relate to the language model we trained on event dates and times.

- **DT-T 1-2 - Language model.** Numeric. We train a language model [104] on date elements and time elements separately. The language is a probability distribution over the sequence of words, and provides the relative likelihood that of sequence of text.

Parser-based features. The following features relate to the use of a standard date and time parser.

- **DT-T 3 - Date candidate is parsable.** Binary. A standard date parser is used to determine whether the candidate date element text contains a parsable date. We encode the existence of such a parsable date as a binary feature. If the parser recognises a date, we store which parts it has identified: i.e. the date, time and year. We encode the existence of such patterns in the subsequent features 4-7 below.
- **DT-T4- 7 - Date parsed fragments.** Boolean. The parser detects the following fragments: date, time and year. Features DT4-7 encode whether these have been identified, first individually, and whether all three have been found.
- **DT-T 8 - Time candidate is parsable.** Boolean. Whether the candidate time text content is parsable.
- **DT-T 9-12 - Time parsed fragments.** Boolean. Similarly as with features DT-T4-7, the features DT 9-12 encode whether the individual and all fragments have been identified.
- **DT-T 13-14 - Remaining character counts.** We further compute the remaining character count of the node text, excluding the matched pattern. The intuition is that if a date pattern is found, but the candidate element contains a very long text, it is likely that a date is mentioned in a longer text such as the description.

Textual similarity-based features. The following two features relate to the textual similarity between the date and time candidate.

- **DT-T 15-16 - Text similarity.** Boolean. Whether the text of the two elements is the same. Sometimes the same text and node appears multiple time on the page. In other cases the same candidate is in both the date and time candidate sets, especially for cases with ambiguous date and time notations such as “7.30”, which could refer to both a time and a date. Dates and times, however are always in different patterns within the same page. We therefore verify the candidate text is not the same, as this is usually a strong indicator that the candidate `date` and the candidate `time` in this pair are the same.

6.3.3.2 Structural Date and Time Features

Tree related features. The following features relate to the depth and position of the candidate element within the DOM tree.

- **DT-S 1-2 Node depth.** Numeric. The number of edges from the root of the DOM tree to the date and time candidate node.

Dictionary-based features. The following features relate to the appearance of keywords in the node of the candidate.

- **DT-S 3-8 - Keywords in underlying structure.** Boolean. We encode whether date specific keywords appear in the text of the node, its parent and grandparent, as well as the node’s content attribute. We consider the keywords “*time*”, “*when*”, “*date*” in multiple languages.

Metadata-based features. The following features relate to the meta data in the header of the page with regards to the candidate node.

- **DT-S 9-12 - Metadata.** Boolean. Overlap of candidate text and meta data content in the header of the page. The four features capture whether the meta data contains the candidate text and vice versa.

Node similarity based features. The following features relate to the similarity of the date and time candidate node.

- **DT-S 13-15 - Node similarity.** Boolean. We encode whether the time candidate and the date candidate are actually in the same node have the same parent and grandparent. This is the case if they are either the same node, siblings, or their parent node are siblings.

6.3.3.3 Visual Date and Time Features

In this section we discuss visual features for detecting date and time elements.

If not otherwise specified, a rectangle’s coordinates relate to the *top left* corner of the bounding rectangle of an element.

Geometry-based features. The following features relate to the geometric appearance of the candidates elements within the page.

- **DT-V 1-4 - X and Y values.** Numeric. The X value, also referred to as *left*, is the distance between the left border of the page and the element rectangle's top left corner. The y value, also referred to as *top*, is the distance between the top border of the page and the element rectangle's top left corner. We encode the X and Y coordinates of the date and time candidate.
- **DT-V 5-8 - Width and Height** Numeric. The height and width of the date and time element rectangle.
- **DT-V 9 - Screen.** Boolean. We encode whether the candidate date and time candidate rectangles are in the first screen of the webpage, i.e. a user would not have to scroll in order to see it.

Pairwise geometry-based features. The following features relate to the geometric appearance of the candidates elements within the page.

- **DT-V 10-12 - Distance and alignment of date and time.** Numeric. The absolute horizontal, vertical and euclidian distance of the candidate date and time rectangles.

Title-based features. The following features relate to the previously extracted title of the event, and leverage characteristics of the event phenomenology. The event title attribute is often visually related to temporal information.

- **DT-V 12-19 - Geometric relation to the title.** We compute the horizontal and vertical alignment and distance between the previously extracted event title and the date and time candidate element rectangle.

6.3.4 Classification of <date, time> Pairs

We train a random forest model to perform a multi-label classification task of <date, time> pairs, represented in vector form with the features listed above. The model classifies the <date, time> pairs into $\{start, end, other\}$ sets, ranked by the likelihood of the candidate belonging to that set. We use 89 features with 200 trees and 100% of the training data for each tree. The precision of detecting the **start date**, **start time**, **end date**, **end time** is 84%, 78.2%, 80.2% and 83.4% respectively. The performance is bound by the performance of the date parser, which is not built for noisy event dates. It was originally built for standard article publishing and company founding dates, which do not contain any times.

6.3.5 Generation of <start, end> Pairs

The classifier in Section 6.3.4 outputs a distribution over the three categories $\{start, end, other\}$ of a given <date, time> candidate pair. Each <date, time> candidate pair is added to a start or end set if the previous classifier classified it as such. This implies that the same <date, time> candidate pair can be

added to both the `start` and the `end` set. We proceed to create point-wise products of `<date, time>` pairs from the `start` and `end` set, given a number of hard constraints. As defined in Section 2 in Chapter 2.2, a `start` candidate and an `end` candidate are not combined if one of the following is true:

- The `start` candidate and `end` candidate are the same `<date, time>` pair. This happens when the same `<date, time>` pair has a score above threshold j for both `start` and `end`.
- There is an `end date` but no `start date` or an `end time` but not a `start time`. As we introduce empty candidates to represent a missing value on the page (e.g. when there is no `end date`), we need to filter out those combinations where the start dates is empty.
- When the `start date` and `end date` can be parsed, we remove instances where the `end date` occurs before the `start date`. Similarly, if the end date is empty, we remove combinations where the `end time` is before the `start date`. When an `end date` is known, the `end time` can actually be earlier in the day, as the start day as the end date could be on a different day.

The combined `start` and `end`, namely quadruples in the form of `< start date, start time, end date, end time`, are input into the second classification step.

In the following section, we present the pairwise dependent features used in the second classification step. These features most importantly focus on the textual, structural, and visual relations between the `start` and `end` `<date, time>` pair.

6.3.5.1 Textual Start and End Date Features

The following features encode the textual relationships between candidate event date and time attributes.

Textual containment. Features encoding whether the four candidate text content **contain** one of the others text contents.

- **O-T 1-12 Textual overlap.** Boolean. Whether the four different candidate element text contain each other. For example, if the `start date` candidate text contains the `end date` candidate text, it is unlikely to be the right combination of temporal attributes, as they typically complement each other.

Text matching. The following features determine whether the textual content of nodes within a candidate equal each other.

- **O-T 13-18 - Text similarity.** Boolean. We encode whether the text content of the candidate temporal attributes **equal** one of the others text contents.

Parser related features. The following features determine whether the parsed dates make sense within the context of the events domain: e.g. an event cannot end before it has started.

- **O-T 19-20 - Start before end.** Boolean. If both **start date** and **end date** are parsable and can therefore be normalised, we encode whether the start date occurs before the end date and the **start time** before the **end time**. If the attributes are not parsable the value is false.
- **O-T 21-22 - Format similarity.** Boolean. If the candidate temporal attributes are parsable, we encode whether the date and time candidates are written in the same format, e.g. “dd-mm-YY”. We compute this feature with standard pattern matching techniques.
- **O-T 23-28 - Pattern similarity.** Boolean. Interrelated pairwise features that encode whether the temporal attributes’ parsed date **equal** one another.

6.3.5.2 Structural start and end date features

The following features capture the structural relationship between temporal event attribute candidates.

- **O-S 1-6 Node similarity.** Boolean. Pairwise features encoding whether the four temporal attribute candidates refer to the same node as one another.

6.3.5.3 Visual Start and End Date Features

The following features encode the visual similarity and relation between event temporal candidates.

Geometric pairwise features. The following features encode the geometric relationships between candidate temporal attributes. Where coordinates are used, we refer to an element rectangles’ top left corner. Event attributes, especially temporal information, are usually positioned in a systematic juxtaposition. We exploit this fact by encoding alignment and distance between pairs of temporal candidates.

- **O-V 1-6 Interrelated distance.** Numeric. These features encode the distance between two candidate temporal attributes in pixels. The distance between two candidate elements is computed by the euclidian distance between the two rectangles top left corner.
- **O-V 7-12 Interrelated horizontal alignment.** Numeric. These features encode the distance between two candidate temporal attributes rectangles top left corner along the X-axis in pixels.
- **O-V 13-18 Interrelated vertical alignment.** Numeric. These features encode the distance between two candidate temporal attributes rectangles top left corner along the Y-axis in pixels.
- **O-S 19-26 - X and Y coordinates.** Boolean. The X and Y coordinates of each of the temporal attribute candidates rectangle top left corner.

Features based on the output of the first classifier. The following features relate to the output of the first classifier.

- **O-G 1-2 - Attribute confidence.** Numeric. We use the output of the first classifier as a feature for the event assembly. We use each confidence score as a feature for the respective probabilities of
 - the **start** <date, time> pair being the **start** <date, time> pair,
 - the **end** <date, time> pair being the **end** <date, time> pair.

6.3.6 Overall Classification of Temporal Attributes

We train a random forest model [30], to predict the likelihood of a quadruple of the form <start date, start time, end date, end time > to be the respective four temporal attributes for the event presented on the webpage under consideration. We train a random forest with 200 trees, 88 features and bags the size of the full dataset. We return the quadruple that has the highest likelihood of being the respective temporal information for the event page at hand. We achieve a precision of 95.6%, 82.5 %, 91.3% and 89.3% respectively, which is an improvement averaging 8 percentage points from the first classification step. We present an in-depth analysis of the experimental results in Section 6.6.2.1.

6.4 Event Location Extraction

The third of the most important event attributes is the location of an event, answering the third extraction problem *where?*. Locations appear in many variations in the events domain, as they can range from a simple room number to a full address. The format of the location often depends on the website. If the location is simply a room within a facility, the event is usually organised by the website owner. Otherwise, a user would not have the contextual understanding where this room is situated. For example, the location string “*Lecture Theatre B*” is a valid location on the website of a university department, as users automatically assume that the room must be located within the facility of the department. On the other side of the spectrum, a location can appear in the form of a full address, including the venue name, street address and city. The full address is usually more common on websites which aggregate and display events from multiple organisers at different locations. When a location is just a room name or a number, the event is usually organised by the website owner and the user automatically understands the context of where *Lecture Theatre B* is located. In the case of an address, the information is typically spread across multiple DOM elements.

The event location candidate generation, therefore, takes these varying particularities into account and selects a larger amount of candidate nodes: we select visible nodes that are under 900 characters, contain at least two alphabetic characters, and have one of the following element tags: {dd, small, a, strong, blockquote, address, em, h2, h3, h4, h5, p, div, td, abbr, li, span}.

6.4.1 Location Specific Features

In this section we introduce the selected features to extract event location attributes from single event pages. As for previous attributes, we structured these into textual, structural and visual features.

6.4.1.1 Textual Location Features

Dictionary-based features. The following features relate to the candidate node text containing location related keywords.

- **LOC-T 1-2 - Dictionary based features.** Boolean. We encode whether the candidate element text contains location specific keywords. We consider the street suffices “*street*”, “*st*”, “*lane*”, “*ln*”, “*ave*”, “*avenue*”, “*boulevard*”, “*blvd*”, “*alley*”, “*road*”, “*rd*”, “*drive*”, “*dr*”, “*rue*”, “*parkway*”, “*pkwy*”, as they are usually a strong indicator for an address, and cardinal words “*n.*”, “*n*”, “*north*”, “*south*”, “*s*”, “*s.*”, “*west*”, “*w*”, “*w.*”, “*east*”, “*e*”, “*e.*”. Cardinal directions are especially used in addresses and location names in North America.
- **LOC-T 3-4 - Longest consecutive character.** Numeric. We consider the longest consecutive number of tokens and non-tokens in the candidate text. Shorter sequences of digits often appear in addresses, such as house number and postal codes, however longer sequences of digits are usually phone numbers or other pieces of information.
- **LOC-T 5-7 - Location-based patterns.** Boolean. We use a set of address specific patterns, namely US and UK post code patterns, street number patterns and lenient phone number patterns and encode whether such patterns occur within the candidate location element text in binary features.

6.4.1.2 Structural Location Features

In this subsection we present the structural features that encode the structure of the underlying vicinity of the candidate node in the DOM tree.

Tree related features. The following features relate to the candidate node within the DOM tree.

- **LOC-S 1 - Leaf node.** Boolean. We encode whether the candidate element is a leaf node in the DOM tree.

Meta data related features. Boolean. The following features relate to location metadata.

- **LOC-S 2-4 - Existence of location meta.** Boolean. We encode whether there is location meta data in the header of the page, the siblings or the parent of the candidate element.
- **LOC-S 5-6 - Count location meta.** Numeric. We count the number of location meta nodes in the page header and siblings of the candidate element. If there are none the value is 0.

- **LOC-S 7-8 - Total overlap of location meta.** Numeric. We encode whether the candidate element text *equals* the meta data in the page header and siblings.
- **LOC-S 9-10 - Partial overlap of location meta.** Numeric. We encode whether the location meta in the page header and siblings contain the candidate text and, vice versa, whether the candidate text contains the meta information.

Dictionary based features. The following features relate to key words appearing in the node attributes and ancestry.

- **LOC-S 11-22 - Node attributes contain.** Boolean. We encode whether the candidate elements attributes and attributes of the parent and grandparent element contain location related keywords. We consider the keywords “*location*”, “*where*”, and “*address*”.

6.4.1.3 Visual Location Features

Title related features. The following features concern the relationship between the candidate location and the previously extracted title.

- **LOC-V 1-6 - Vertical and horizontal alignment with title.** Numeric. The distance of the title and the candidate, parent and grandparent along the X-axis and Y-axis.
- **LOC-V 7 - Title distance.** Numeric. We compute the euclidian distance between the title and the candidate location, as the location is typically within the event title’s proximity.
- **LOC-V 8 - Title above location.** Boolean. We encode whether the title is located above the location candidate within the page. In other words, whether the titles’ X-coordinate is smaller than the candidate elements. This feature is based on the intuition that the location attribute is usually located below the event title.
- **LOC-V 9 - Title location common ancestor.** Numeric. We compute how far the closest common ancestor of the title and the location is in the DOM tree.
- **LOC-V 10-11 - Title height and width difference.** Numeric. We generate the difference in height and width between the title element and the candidate location elements bounding rectangles in pixels.

6.4.2 Location Classification

As for previous attributes, we train a random forest model, yielding the distributed binary likelihood of a candidate location element being of type location and being the location attribute of the event we are trying to extract. We train the location classifier with 200 trees with different subsets of features over the full size of the training dataset. We return the candidate with the highest score returned by the model.

6.5 Event Description Extraction

The event description attribute in a single entity event page usually contains additional information about the event that is not expressed in the main key-value pairs. Depending on the event organiser or website designer, such descriptions can range from a single sentence to an almost article length blob of text. Unlike the main event attributes, precise description-based patterns do not exist, and we therefore heavily depend on structural and visual features to recognise event descriptions. As with other event attributes, we can exploit the visual alignment and juxtaposition of event description, as they are usually in some geometric relation with other event attributes and the overall page.

6.5.1 Description Candidate Generation

The candidate generation phase for the extraction of the event description is the least complex out of all the event attributes, as event descriptions come in many shapes and forms. An event description can range from simply just a word or two, to a full article-length string of text. Hence, significantly restricting the number of candidate elements to be featurised and classified is limiting, as to not eliminate promising elements at an early stage. We therefore select all textual nodes that contain at least a whitespace-normalised token as a potential candidate, given it is visible and does not have a number of blacklisted element types in its ancestry. Not pruning the list of candidate elements i) introduces a lot of noise and ii) affects the speed at which our extractors run. The former can be corrected by a number of carefully selected and powerful features described in the following section.

6.5.2 Description Specific Features

In this section, we introduce the selected features for recognising event descriptions on single entity event pages.

6.5.2.1 Textual Description Features

Textual features for recognising event description are relatively sparse compared to other event attributes, as description text do not have particular characteristics or pattern we could exploit.

General textual features.

- **DESC-T 1 - Description language model.** Numeric. We train a language model in the training phase of the event description text and use the output as a numeral feature.
- **DESC-T 2 - Text similarity with event title.** Numeric. The text similarity with the previously predicted event title.

Dictionary-based features. Dictionary-based features encode whether certain relevant keywords appear in the text. As previously mentioned, these dictionaries are relatively small in size as descriptions have very few keywords.

- **DESC-T 3-4 - Keywords Boolean.** We encode whether the candidate element and parent text contains keywords. We use keywords “*description*” in multiple languages, which is sometimes used as a key value before the text.
- **DESC-T 5-9 - Number of keyword tokens in text.** Numeric. Number of dictionary-based tokens in the candidate and parent element text. We compute two features for each of the stop words, positive tokens: “*detail*”, “*desc*”, “*overview*”, and “*description*”, and negative tokens: “*mail*”, “*shipping*”, “*share*”, “*phone*”, “*click*”, “*suggestion*”, “*question*”, “*comment*”, “*welcome*”, “*contact*”.

6.5.2.2 Structural Description Features

The following features encode structural properties of a candidate description and encode characteristics of the underlying adjacency of the candidate node.

- **DESC-S 1 - Text overlap with meta description.** Boolean. We encode whether the text overlaps with the meta description text content in terms the Levenshtein distance [76].
- **DESC-S 2-5 - Element children.** Boolean. Four features encode whether the candidate element contains the following types of elements among its children: <div>, <pre>, <td>, <block>, <quote>, <address>, , and <dl>.
- **DESC-S 6-7 - Element ID - keywords.** Boolean. Two features encode whether the candidate element id contains the following types positive keywords: <detail>, <desc>, <overview> and <description>, and negative keywords: <face>, <call>, <mail>, <send>, <faq>, <gplu>, <contact>, <fb>, <breadcrumb>, <twitter>, <share>, <price>, <pin>, <popup>, <tweet>, <comment>, <review> and <video>.

6.5.2.3 Visual Description Features

The following features encode the visual characteristics of candidate description elements and their appearance, particularly in terms of geometric properties.

Geometric features. The following features relate to the geometric position of the candidate element.

- **DESC-V 1-2 - Horizontal and vertical alignment to title.** Numeric. These two features encode the distance of the candidate rectangle’s top left corner to the previously extracted title along the X-axis and Y-axis respectively.

- **DESC-V 3 - Horizontal distance to title.** Numeric. The distance of the candidate to the title along the Y-axis.
- **DESC-V 4-15 - Layout image checker.** Numeric. Twelve features encoding relative position to the first four images in the page, in terms of euclidian distance, vertical, and horizontal alignment as above.

6.5.3 Description Classification

We trained a random forest model to predict the likelihood a candidate element is the description of the event on the page. We train the random forest with 200 trees using different combinations of the features over the complete training dataset. We return the candidate with the highest likelihood of being the event description of the particular event.

6.6 Experimental Evaluation

In this section we present the extensive experimental evaluation. All evaluations have been performed with a 5 fold cross validation. For this purpose we randomly split the dataset in 5 equal subsets, and rotate through the sets while training the models on 4 sets and evaluating on one.

6.6.1 Dataset

The dataset used in this chapter has been manually curated by two data curators, given precise instructions on how to find and semantically annotate event web pages based on the formal definition in Section 2.2.

In the following subsections we will describe the distribution of the dataset, specifically in terms of websites in Section 6.6.1.1, language in Section 6.6.1.3 and type in section 6.6.1.2 to illustrate the wide range of the data used in this study.

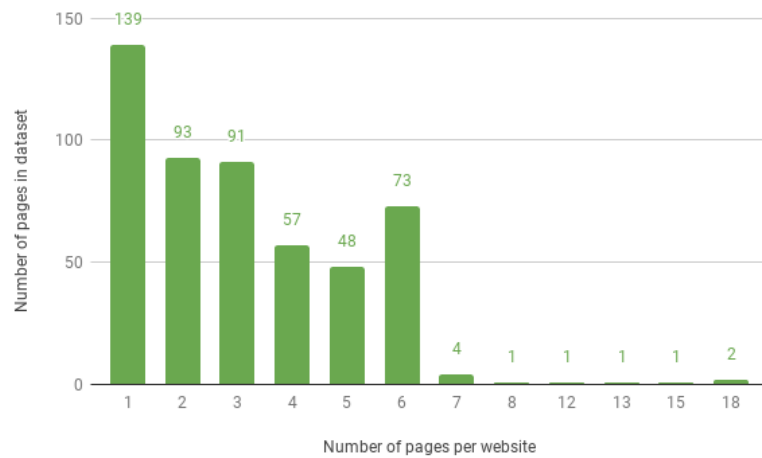


Figure 6.6: Distribution of pages per website in the dataset.

6.6.1.1 Distribution of Webpages on Websites

Our dataset of 1647 manually curated event pages is from 512 unique websites. In Figure 6.6, we show the distribution of event pages amongst unique websites. As depicted, the great majority have 1-6 instances per unique website, ranging from 1 to 18 pages per website. There are an average of 3.15 pages per website with standard deviation of 2.17.

6.6.1.2 Website Type Distribution

We further analysed the dataset to highlight the proportion of aggregator sites and long tail pages. The reason this is important is that aggregator site for events are more standardised and stable. Long tail pages tend to be more ad-hoc, hence more difficult to extract from. However, particularly in the events domain, the long tail pages have relevant information which is often not captured on big aggregator sites. Out of the 1647 pages, only 47 pages belong to 7 aggregator websites (out of 52 unique websites in the dataset).

6.6.1.3 Language Distribution

We further illustrate that our approach is largely agnostic at the surface language of a website. In Table 6.1 we show that our dataset is spread across 24 different languages (not counting any dialects such as Brazilian Portuguese or Canadian French). While the majority of the pages are in English, almost 40% of pages are in non-english languages sampled from across the world.

6.6.2 Overall Performance

We begin by summarising the overall performance of our approach, before delving into different aspects of the system. Table 6.2 illustrates the percentage of correctly identified event attributes when performing a five fold cross validation on extractors for title, date, location and description individually.

6.6.2.1 Temporal Attribute Assembly

In Section 6.3, we introduce the event specific date extractor, which takes into consideration the complex variations of temporal event information with the attributes **start date**, **start time**, **end date**, **end time**. To iterate, in a first classification step, we classify `<date, time>` pairs into `start`, `end` and `other`. In a second classification step, we combine `start` and `end` candidates for a second classification step to determine the best `<start, end>` pair for a given event page. In this second phase, we particularly exploit pairwise-interrelated features amongst the individual temporal attributes.

In Table 6.3, we show that the second classification step boosts precision by 11.6, 4.3, 11.1 and 5.9 percentage points respectively for **start date**, **start time**, **end date** and **end time**.

Language	Code	Count	Percentage
English	en	1002	60.84%
German	de	290	17.60%
Italian	it	74	4.49%
French	fr	48	2.91%
Dutch	nl	48	2.91%
Spanish	es	44	2.67%
Catalan	ca	21	1.27%
Polish	pl	21	1.27%
Russian	ru	15	0.91 %
Czech	cs	13	0.79%
Afrikaans	af	12	0.73%
Portuguese	pt	9	0.55%
Tagalog	tl	7	0.43%
Chinese	zh	7	0.43%
Finnish	fi	6	0.36%
Rumanian	ro	6	0.36%
Croatian	hr	6	0.36%
Norwegian	no	5	0.30%
Greek	el	3	0.18%
Swedish	sv	3	0.18%
Bulgarian	bg	3	0.18%
Danish	da	2	0.12%
Hungarian	hu	1	0.06%
Indonesian	id	1	0.06%

Table 6.1: Distribution of the languages of the web pages in the dataset.

6.6.2.2 Languages

We claim that TIME supports many languages. In this experiment, we first train TIME on non-english pages and evaluate on english pages and then do the opposite. The experiment unfortunately did not have a large enough dataset, as the english pages were more frequent than the non-english ones, which introduce a strong bias (especially when training on only few non-english pages and evaluating on a much larger set of english pages). Nevertheless, we report the findings in Table 6.4 demonstrate that, in particular, when we train only on english pages, and test on non-english pages, the accuracy does not fluctuate more than 8 percentage points.

title	start date	start time	end date	end time	location	descr- iption
92.0%	95.6%	82.5%	91.3%	89.3%	83.2%	82.5%

Table 6.2: Overall performance of TIME.

	start date	end date	start time	end time
1st classification step	84.0%	78.2 %	80.2%	83.4%
2nd classification step	95.6%	82.5%	91.3%	89.3%
Difference in percentage point	+11.6	+2.3	+11.1	+5.8

Table 6.3: Evaluation of the two steps in the temporal attribute extraction of TIME.

	title	start date	start time	end date	end time	location	descr- iption
Trained and evaluated on both english and non-english pages	92.0%	95.6%	82.5%	91.3%	89.3%	83.2%	82.5%
First step where applicable	N/A	69.96%	59.96%	56.69%	67.43%	N/A	N/A
Train on multi-lingual and evaluate on english	91.8%	76.01%	57.18%	64.36%	71.14%	59.86%	81.94%
Difference in percentage point	0.2	+19.59	+25.23	+26.94	+18.16	+6.56	+0.56
First step where applicable	N/A	85.55%	74.53%	69.40%	82.91%	N/A	N/A
Train on english and evaluate on multi-lingual	83.8%	92.7%	75.0%	85.6%	86.3%	76.64%	87.04%
Difference in percentage point	+8.2	+2.9	+7.5	+5.7	+3	+6.56	+4.54

Table 6.4: Evaluation of TIME trained on multilingual pages and evaluate on english pages, and vice-versa.

6.6.2.3 Comparing Aggregator Sites with the Long Tail

The major benefit of the techniques used in TIME over other techniques is that it can be applied to new unseen pages and is therefore applicable to the long tail of small websites. While the webpages in the dataset are carefully selected to represent the long tail in majority, we aimed to underpin the fact that TIME extracts from the long tail templates as well as from the standardised aggregator pages with similar results. For this purpose, we evaluated TIME in a two fold evaluation where the test set is split into aggregator pages and long tail pages. For convenience, we select pages that belong to websites that have more than 8 pages per website in our dataset, namely: *www.eventbrite.com*, *events.stanford.edu*, *paloaltojcc.org*, *www.explorelogan.com*, *www.lincolncenter.org*, and *bampfa.org*. As seen in Table 6.5, we achieve similar results when splitting the dataset, with the exception of some attributes decreasing significantly amongst the aggregators. This is due to a skewed data distribution, as the aggregator pages are by far fewer than the long tail pages.

	title	start date	start time	end date	end time	location	descr- iption
Aggregator	96.4%	82.0%	67.8%	64.5%	76.8%	74.8%	82.4 %
Long tail	89.8%	90.0%	80.2%	73.0%	82.4%	81.7%	80.0%

Table 6.5: Evaluation of the precision of TIME for aggregator pages as well as pages in the longtail.

	title	start date	start time	end date	end time	location	descr- iption
Same features for all	77.6%	95.4%	69.7%	91.7%	84.1 %	82.18%	73%
Individual Features	92.0%	95.6%	82.5%	91.3%	89.3%	83.2%	82.5%
Difference in percentage point	+14.4	+0.2	+12.8	-0.4	+5.2	+1	+9.5

Table 6.6: Evaluation of the precision of TIME using the same features for all event attributes.

6.6.3 Feature Ablation Study

In this section, we highlight the importance of our choice of features through an extensive ablation study. In the following subsections, we present TIME’s performance when removing certain types of features. More specifically, we aim to emphasise the importance of the combination of textual, structural and visual features. We underpin the fact that only with the combination of these three types of features our results are in the 80s and 90s.

6.6.3.1 Using the Same Features for All Event Attributes

In this particular setting, we want to highlight the importance of individual feature sets for each event attribute in our extraction schema. In Section 6.1.1, we introduce a number of features, which are used across all event attributes extracted in our approach. Each attribute then has attribute specific features introduced in the Sections 6.2-6.5. In the first row of Table 6.6, we report the performance of our system if only the standard library of features described in Section 6.1.1 is used without the attribute specific features. In the second row we show the performance of the system with the individual features.

6.6.3.2 Multilingual features

In our second ablation study, we evaluate the importance of language specific features by removing them and evaluating the performance. Multilingual features are dictionary-based features that contain keywords in multiple languages. In Table 6.7, the performance of the system both with and without multilingual features is displayed, and we show that the title, start time and description attribute significantly drops in precision when the multilingual features are removed.

	title	start date	start time	end date	end time	location	description
First step where applicable	N/A	82.2%	72.6 %	75.0%	80.3%	N/A	N/A
Excl. multilingual features	90.4%	94.5%	80.6%	87.7%	89.0%	83.0 %	73.7%
Incl. multilingual features	92.0%	95.6%	82.5%	91.3%	89.3%	83.2%	82.5%
Difference in percentage point	+1.6	+1.1	+1.9	+3.6	+0.3	+0.2	+8.8

Table 6.7: Evaluation of the precision of TIME ablating multilingual features.

	title	start date	start time	end date	end time	location	description
First step where applicable	N/A	81.1 %	74.2 %	75.9 %	79.2 %	N/A	N/A
No Textual Features	82.7%	95.0%	81%	90.7%	89.1 %	82.9 %	56.8%
All Features	92.0%	95.6%	82.5%	91.3%	89.3%	83.2%	82.5%
Difference in percentage point	+9.3	+0.6	+1.5	+0.6	+0.2	+0.3	+25.7

Table 6.8: Evaluation of the precision of TIME ablating textual features.

6.6.3.3 Textual Features

We evaluate the performance of the textual features for the overall performance of our system. For this purpose, we evaluate the system by removing general textual features, namely character statistics **Text 1-5** introduced in Section 6.1.1.1 and token statistics **Token 1-20** introduced in Section 6.1.1.2. Furthermore, we disable features **T 11-15** for title, **DT-T 1-16** and **O-T 1-7** for dates, **LOC 1-7** for locations and **DESC-T 1-9** for descriptions from Sections 6.2-6.5 respectively. Textual features capture all characteristics of the textual node content. We use character, token and keywords statistics as well as language models where applicable. As seen in the last column of Table 6.8, textual features improve the performance of all event attributes in our schema, on average by 5.5 percentage points. While event temporal attributes and the location attribute are less impacted by removing all textual features, especially the event title and event description fluctuate. This is mainly because temporal attributes and locations come in many different textual variations, with temporal attributes often containing a higher average number of digits within the text. However, titles often have different casing than the rest of the page to emphasise the text, fewer tokens and characters. A significant boost of +25.7 percentage points through textual features can be seen for the event description attribute, which is mainly because descriptions are usually longer, natural text, which can therefore be distinguished through its character and token statistical features.

	title	start date	start time	end date	end time	location	description
First step where applicable	N/A	80.7%	69.1%	73%	78.8%	N/A	N/A
Only Textual	87.9%	91.9%	79.2%	83.9%	86.7%	0.1%	68.2%
All Features	92.0%	95.6%	82.5%	91.3%	89.3%	83.2%	82.5%
Difference in percentage point	+4.1	+3.7	+3.3	+7.4	+2.6	+83.1	+14.3

Table 6.9: Evaluation of the precision of TIME using only textual features.

While textual features are very important, our next study emphasises that the textual features alone are not sufficient. In Table 6.9, we present the results of running TIME with *only* textual features, hence removing all structural and visual features. As seen in the last column, textual features alone are not sufficient to achieve the overall results presented in Table 6.2 and repeated in the fourth row below for convenience. Specifically, the system recognises barely any location attributes in our dataset with only textual features. Likewise, the description attribute, while dropping to only 56.8% precision when *removing* textual features, only performs at 68.2% when using *only* textual features.

6.6.3.4 Structural Features

In the following two sections, we evaluate the performance of TIME with regards to structural features used in the learning phase. More specifically, we refer to tree statistics **Tree 1-6** introduced in Section 6.1.1.3, and parent statistics **Parent 1-4** introduced in Section 6.1.1.4. Furthermore we disable features **T 1-10** and **T 16-17** for title, **DT-S 1-15** and **O-S 1-6** for dates, **LOC-S 1-22** for locations and **DESC-S 1-7** for descriptions from Sections 6.2-6.5 respectively. Structural features exploit information gathered from the candidate element itself and its position in the DOM tree.

In Table 6.10, we report the performance of TIME when disabling all structural features in the learning phases. Removing the structural features decreases the performance of extracting event attributes between 0.2 and 9.7 percentage points, on average 2.8 percentage points with a standard deviation of 3.5. Most interestingly however, is that if we invert the study and instead of removing the structural features we *only* use structural features, the precision for some attributes decreases significantly, as seen in Table . The performance for location and description drops to almost zero, which is comprehensible for the description attribute, since it is usually visually quite prominent in terms of size.

6.6.3.5 Visual Features

The performance of visual features is particularly important to establish, as visual features are the main reason our approach needs to render web pages to access such visual information from the renderer. Visual features are accessed through the CSS properties and rendering web pages creates a performance overhead as each page takes time to render. We refer to basic rendering features **Render 1-8** introduced in Section

	title	start date	start time	end date	end time	location	description
First step where applicable	N/A	82.5%	72.8%	75.2%	80.6%	N/A	N/A
No Structural	91.0%	94.4%	80.6 %	87.7%	89.0%	81.9%	72.2%
Individual Features	92.0%	95.6%	82.5%	91.3%	89.3%	83.2%	82.5%
All in percentage point	+1	+1.2	+1.9	+3.6	+0.3	+1.3	+9.7

Table 6.10: Evaluation of the precision of TIME ablating structural features.

	title	start date	start time	end date	end time	location	description
First step where applicable	N/A	70.2%	63.6%	72.3 %	71.5%	N/A	N/A
Only Structural	88.4%	91.1%	76.7%	87.0%	82.1%	0%	1.5%
All Features	92.0%	95.6%	82.5%	91.3%	89.3%	83.2%	82.5%
Difference in percentage point	+3.6	+4.5	+5.8	+4.3	+7.2	+83.2	+81.0

Table 6.11: Evaluation of the precision of TIME using only structural features.

6.1.1.5, and font and style features **Style 1-9** introduced in Section 6.1.1.6. Furthermore we don't disable any title-specific features, as we have not computed any. For the other attributes, we consider **DT-V 1-19** and **O-V 1-26** for dates, **LOC-V 1-11** for locations and **DESC-S 1-15** for descriptions from Sections 6.2-6.5 respectively. In Table 6.12, we report the performance of TIME when removing all visual features during the classification phases. The performance of the classifier drops from 2.3 to 12.7 percentage points, on average by 5.1 percentage point with a standard deviation of 4.1.

6.6.3.6 Experimental Summary

The aim of the above experiments was to underpin our design choices for the event extraction approach introduced in this chapter. By comparing the performance of the approach through an ablation study, we were able to demonstrate the vital contribution of textual, structural and visual features in our approach. As seen in Figure 6.7, only using one type of such features reduces the performance significantly. Likewise, it can be seen in Figure 6.8 that using two out of the three types of features also does not produce the results of all three types of features depicted in green. It is only through the combination of textual, structural and visual features that event extraction can be achieved at a high accuracy. The reasoning behind this finding is relatively intuitive: humans use textual and visual information to subconsciously recognise event information on a given webpage. Additionally, web developers build websites in a systematic way, that provides additional cues for web data extraction. So all in all, we encode the same cues used by humans to both read and build webpages in machine learning features for our extraction methods, imitating any rational reasoning a human would use to identify the relevant information on the page.

	title	start date	start time	end date	end time	location	description
First step where applicable	N/A	81.4%	70%	73.3 %	78.8 %	N/A	N/A
No Visual	87.9%	91.6%	79.0%	83.4%	87%	82.8%	69.8%
All Features	92.0%	95.6%	82.5%	91.3%	89.3%	83.2%	82.5%
Difference in percentage point	+4.1	+4.0	+3.5	+7.9	+2.3	+0.4	+12.7

Table 6.12: Evaluation of the precision of TIME ablating of visual features.

	title	start date	start time	end date	end time	location	description
First step where applicable	N/A	80.2%	73.7%	76.1 %	79.2%	N/A	N/A
Only Visual	87.9%	95.4%	80.6%	89.3%	88.3%	9.8%	9.7%
All Features	92.0%	95.6%	82.5%	91.3%	89.3%	83.2%	82.5%
Difference in percentage point	+4.1	+0.2	+1.9	+2.0	+1.0	+73.4	+72.8

Table 6.13: Evaluation of the precision of TIME using only visual features.

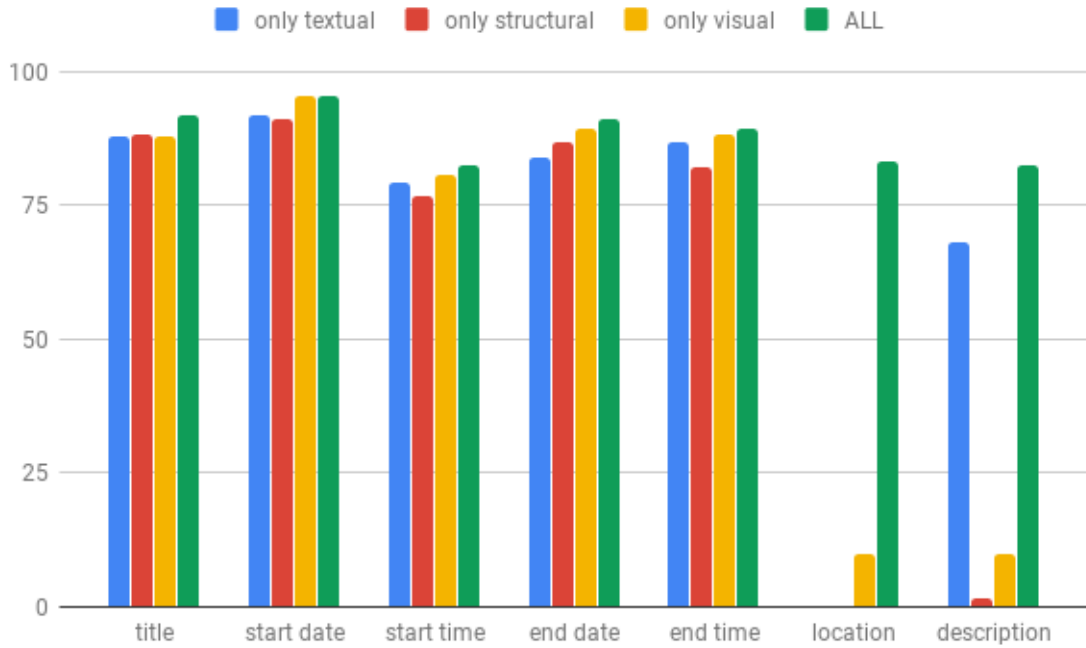


Figure 6.7: Comparison of the performance of four variants of TIME, which use (1) **only** text features, (2) **only** structural features, (3) **only** visual features and (4) all features.

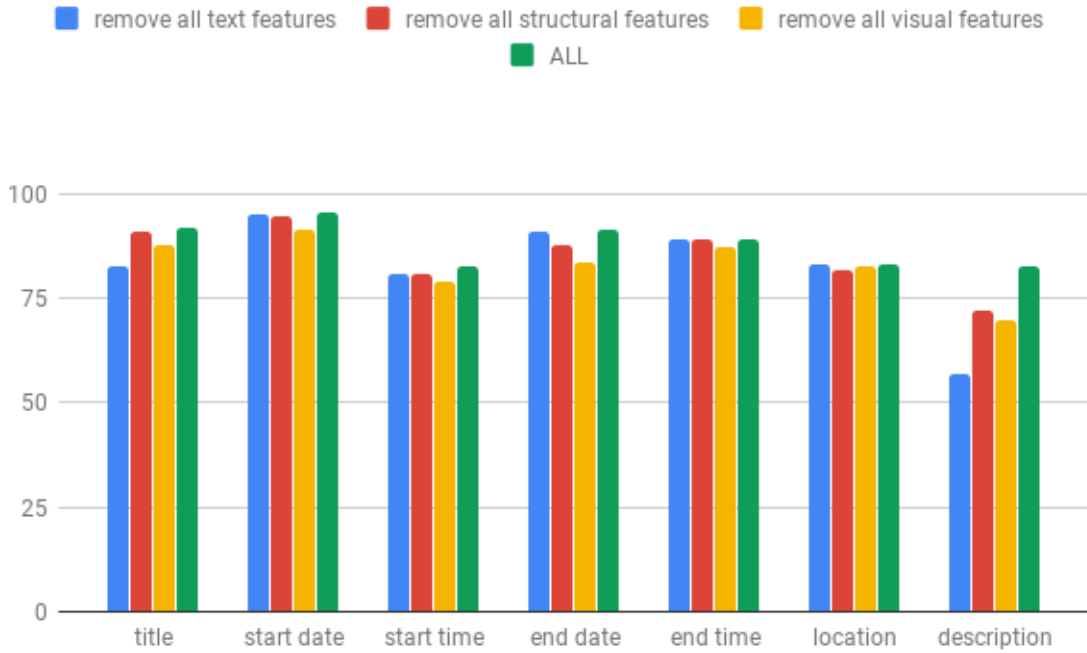


Figure 6.8: Comparison of the performance of four variants of TIME, which **ablate** (1) textual features, (2) structural features, (3) visual features, and (4) which includes all features.

6.7 Industrial Application of TIME: The Diffbot Knowledge Graph

Diffbot² develops and operates a production system for web-scale automatic knowledge acquisition. Figure 6.9 shows the overall information processing pipeline that takes in the world wide web as an input and produces inferred entities in the knowledge graph. We implemented the above introduced TIME approach into the Diffbot knowledge graph pipeline, for the purpose of populating events into the knowledge graph, as seen in Figure 6.9.

Web-crawling Diffbot utilises the Gigablast [5] web crawler, an open-source C++ complete crawling, indexing, and search system developed by Matt Wells, the current VP of Search at Diffbot. It implements its own signals-based architecture implemented over UDP communications over the shards of a Gigablast cluster.

The Gigablast implementation that operates in the Diffbot datacenter has been customised to pass off discovered web pages to Diffbot automatic extraction services via a REST API all, which returns a JSON body that is indexed by Gigablast. Gigablast supports a query language that allows for query both HTML documents as well as JSON documents according to the key-value properties of the JSON. For example, it is possible to query Gigablast for documents about machine learning simply with the query “[machine

²www.diffbot.com

learning]” as well as query for Diffbot JSON objects that are articles written by “[Michael Jordan]” with the query `[type:Article author:“Michael Jordan”]`.

Web Rendering Modern web documents cannot be modelled as simple HTML documents, but a collection of resources that are fetched via HTTP that include HTML, Javascript code, CSS, images, movies, flash and along with external state, such as cookies and server-side storage, combine to form a dynamic movie that is rendered within a web browser. In fact, it even common on the web to find webpages (e.g. single-page applications) that comprise no visible text in the initial HTML, but are fully created client-side using Javascript and dynamic loading of resources.

Therefore, the first step involved in faithfully reproducing the web page that is experience by human visitor, necessarily involves rendering the page in a virtual web browser. Diffbot has developed a proprietary rendering technology that allows for large scale rendering of the web and at much faster rate than a typical consumer web browsing client (such as Selenium used in Chapter 5), by dropping user-facing features of a web browser that are not needed for computing the rendered DOM. For example, UI elements such as bookmark manager, session management, and web resources such as tracking javascript codes are not necessary to render a page for the purposes of extracting visual characteristics of the page. Because rendering sessions are shorter than typical client web browser lifetime, even memory management is not necessary.

Additionally, Diffbot has modified the web page rendering process using machine learning to predict and optimise which web resources are needed in rendering a page. Another prediction is at what cut-off time a render of a page should be considered complete. This is a non-trivial problem since many of the visual elements that appear on the page are generate after the initial load of the HTML and after the “onload” event. There are some webpages that infinitely load content as the user scrolls down. In this way, a webpage can be thought of as a movie, where we want to cut-off or “serialise” the visible document at a point in time.

Diffbot has modified the web page rendering process to serialise detailed information to “augment” the DOM of the page, including detail about every geometric position of the page, the RGBA value of the pixels,

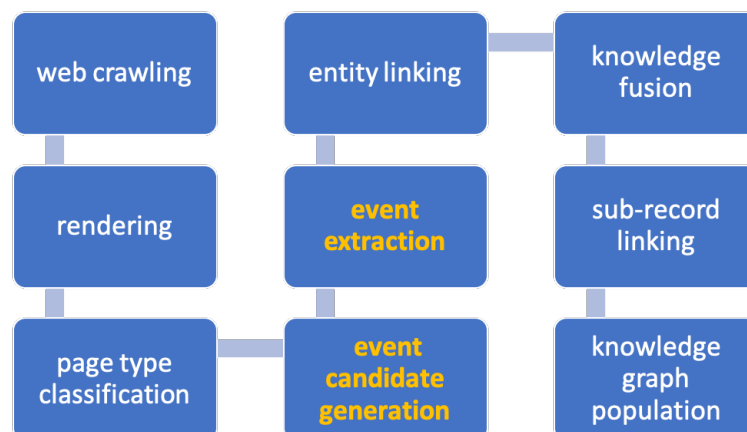


Figure 6.9: Diffbot Knowledge Graph pipeline

the fonts and styles, and positioning properties applied to each element of the page, the internal state of the Javascript VM and layout engines, and the history and timing of the HTTP transactions and headers. This detailed rendering output, which encapsulates the total visual, textual, DOM, and internal state of the rendering process at a point in time (referred to as the “*rendered DOM*”) is supplied to the machine learning algorithms.

Page type classification Given a page, the page classifier retrieves the rendered DOM to classify the page into a top-level page type. From previous surveys of the web, Diffbot has found that 98% of the web can be categorised into one of 20 page types [8]:

article	audio	chart	discussion	document
download	error	event	FAQ	front page
game	image	job	location	product
profile	recipe	form	search engine result	video

Table 6.14: Page types supported by the Page Classifier

Diffbot computes a set of features from the rendered DOM, using an ensemble classifier with semi-supervised training regime to produce a probability distribution along these 20+1 (other) classes. The 2-fold cross validation accuracy of this classifier is PR: 0.98/1.00 (n=30273).

Once the top-level type of the page has been determined, the request is passed to a type-specific automatic extraction algorithm. Different systems (utilising common components) handle each of these 20 types. In this chapter, we introduced the event extraction framework TIME implemented in the Diffbot pipeline as the ***type extraction*** for the events domain. The event extractor is now in beta version and will be launched to customers as an API mid 2019 and be implemented in the full Knowledge Graph pipeline in the second half of 2019.

Chapter 7

Conclusion and Future Work

This thesis focuses on template-independent data extraction from single entity pages on the web. We particularly concentrate on the events domain, as event attributes have characteristics that make event objects particularly challenging to extract. Furthermore, single entity pages in this domain are distributed across a large variety of different sources (the long tail), that render traditional data extraction techniques unsuitable. We show that template-independent event extraction without per-site supervision is feasible at $\geq 90\%$ accuracy for most event attributes.

- (1) We show that though semantic annotations tend to be noisy and often erroneously applied, an extensive automated filtering and cleaning process is effective at removing most of that noise and yields both a dataset that can serve as benchmark for event extraction on the web and training data for our first approach.
- (2) We introduce TIDE, a template-independent, distantly supervised event extraction system that requires no supervision other than the dataset bootstrapped above. TIDE first predicts which category of event attributes an element belongs to, and second assesses which combinations of candidate event attributes form valid events by considering their relative position, alignment, and similar features correlating with a consistent event announcement. We show through an extensive evaluation that TIDE achieves a precision of 93% for extraction event attributes from event pages and demonstrate the role of different steps and thresholds to that overall performance.
- (3) Unfortunately, semantic annotations are only really available for a small set of core event attributes. Therefore, we introduce a second approach, TIME, a supervised event multi-attribute extraction approach that learns to recognise event attributes from heterogeneous sources. We leveraging unique characteristics of the events domain for each attribute, optimising the number of elements to be processed. We demonstrated that encoding similar visual cues human use to recognise information on webpages into machine learning features can achieve high accuracy across a heterogeneous, multilingual dataset. We translate such unique characteristics into textual, structural and visual features. We demonstrate that the combination of these three types of features is necessary, as Section 6.6.3 shows a

clear decrease in precision when only some of those features are used. TIME achieves an accuracy of $\geq 80\%$ for all attributes and $\geq 90\%$ for many of the more important ones. It has been implemented at the commercial data extraction company Diffbot and is currently in beta, powering the population of the Diffbot knowledge graph for event data.

While TIDE and TIME can be applied to the same type of webpages, they have different areas of applicability depending on whether the system is calibrated to be an unsupervised approach or for granularity and precision of extracted event attributes. If annotated training data is not a concern, we can argue that TIME is more sophisticated and applicable in real world applications.

Generality of the approach. The methods applied in TIME and TIDE can be used to build extraction methods for fields of schemas from other domains such as products, articles and job listings. Similarly as in the work in this thesis, the same visual, textual and structural cues humans would use to identify such information would have to be identified and incorporated in the extractors. These unique characteristics of the particular domain need to be considered both at a candidate generation level, and encoded into machine learning features in the learning phase. The temporal event attributes are rendering the event extraction domain particularly challenging, it can therefore be assumed that the extractors for the majority of fields in other domains would have a similar complexity to the event title, location and description extraction presented in Section 6.2, 6.4 and 6.5.

7.1 Future Work

To conclude, we briefly discuss future work addressing some limitations and research challenges raised by the results presented in this thesis:

- (1) **Temporal event attributes.** First, we aim to extend the date grammar to deal even better with the kind of date and time information as seen in the events domain.

We strongly believe that this would improve the accuracy of extracting temporal event information significantly. In Sections 6.3.4 and 6.3.6, we introduce a two step approach to identifying temporal event information, including start date, start time, end date and end time. In the second step, once we have a number of candidates for each field, we leverage domain knowledge to apply heuristics (such as the start date must occur before the end date) and normalisations. However, both in identifying candidates for the first step and applying heuristics before the second classification step, we have to use a date parser library for dates that was (a) not built to recognise times and (b) was built for the article and product domain that have significantly more standardised and less noisy dates than the events domain. Furthermore, article and product domains do not use times and date ranges, such as “9-10th of February”. This shortcoming is reflected in the resulting accuracy of the temporal event fields extraction, and we suggest a substantial improvement could be achieved with more suitable date and

time patterns that can handle (a) times and (b) noisier dates. This will further enable us to normalise all temporal information, allowing the generation of time series and crisp data for knowledge graph population.

(2) **Benchmarking in the events domain.** Event extraction from the web as defined in Section 2.2, has received limited attention in literature. Relevant approaches are either not similar enough to directly compare [109, 48], or hard to replicate [46, 81]. Therefore, our evaluation lacks a comparison to competitor systems. Once we have improved the accuracy of our extraction as suggested above, we plan to bootstrap our technique to create a large dataset to be published online to provide a benchmark dataset in this domain. We aim to release such a dataset in mid 2019, as we will be co-organising a knowledge graph population challenge for event data in May 2019.

(3) **Extend TIME.** We plan to extend the attributes of the TIME, namely *organiser*, *pricing*, *images* and *attendees*. We estimate an effort of 1-2 weeks to add each attribute to the system.

If previously extracted fields follow a similar visual pattern, e.g. as entries in a tabular structure, further values in the same visual structure that have not been extracted could represent further event entities. Exploiting such visual similarities could be a viable approach to extend the schema.

(4) **Augment location data.** As seen in Section 6.4, extracting locations from event pages is a hard problem, as locations range from a single room such as “*Lecture Theatre B*” to a complete combination of venue name and postal address. When an event contains only a room or building name as a location attribute, the attribute cannot be linked to the relevant location entity in the knowledge graph. We therefore plan to augment the information of such location attributes, by finding the relevant information on the page. When only a room or building name is provided, we assume that the relevant location address or building is stated on the web page or website. Such a building name or postal address could for example be located in the description or footer of the page, or in the contact or home page of the website. We plan to investigate this intuition as this would be particularly useful in the context of knowledge graph population, to link the event to the correct address or venue.

(5) **Event entity linking.** We plan to fully integrate events in the Diffbot Knowledge graph in 2019. For this purpose, we are planning to use record linking to deduplicate event announcements that refer to the same event, and to link event attributes to their respective entity, e.g. location or organiser, in the knowledge graph.

Bibliography

- [1] Apache Any23. <https://any23.apache.org/>.
- [2] Cascading Style Sheets Level 2 Revision 1 (CSS 2.1). <https://www.w3.org/TR/CSS2/>.
- [3] Chrome Driver. <http://chromedriver.chromium.org/>.
- [4] Common crawl corpus. <http://commoncrawl.org>.
- [5] Gigablast. <https://https://www.gigablast.com/>.
- [6] HTML5.3. <http://w3c.github.io/html/>.
- [7] ISO/DIS 8601-1 Date and time — Representations for information interchange. <https://www.iso.org/obp/ui#iso:std:iso:8601:-1:dis:ed-1:v1:en>.
- [8] Mashable: Twitter: A Day in the Life [INFOGRAPHIC]. <https://mashable.com/2012/08/16/twitter-day-in-the-life-infographic>.
- [9] Schema.org. <http://schema.org/>.
- [10] Selenium Webdriver. <https://www.seleniumhq.org/projects/webdriver/>.
- [11] The unicode standard. <https://unicode.org/>.
- [12] TIMEML. http://www.timeml.org/publications/timeMLdocs/timeml_1.2.1.html.
- [13] TIMEX3. http://www.timeml.org/publications/timeMLdocs/timeml_1.2.1.html#timex3.
- [14] XML Path Language (XPath). <https://www.w3.org/TR/xpath/>.
- [15] The State of Data Science & Machine Learning. <https://www.kaggle.com/surveys/2017>, 2017.
- [16] Manoj K Agarwal, Krithi Ramamritham, and Manish Bhide. Real time discovery of dense clusters in highly dynamic graphs: identifying real world events in highly dynamic environments. *Proceedings of the VLDB Endowment*, 5(10):980–991, 2012.

- [17] David Ahn. The stages of event extraction. In *Proceedings of the Workshop on Annotating and Reasoning about Time and Events*, pages 1–8. Association for Computational Linguistics, 2006.
- [18] Robert Baumgartner, Sergio Flesca, and Georg Gottlob. The elog web extraction language. In *International Conference on Logic for Programming Artificial Intelligence and Reasoning*, pages 548–560. Springer, 2001.
- [19] Robert Baumgartner, Sergio Flesca, and Georg Gottlob. Visual web information extraction with lixto. Morgan Kaufmann, 2001.
- [20] Robert Baumgartner, Wolfgang Gatterbauer, and Georg Gottlob. Web data extraction system. In *Encyclopedia of Database Systems*, pages 3465–3471. Springer, 2009.
- [21] Hila Becker, Feiyang Chen, and Dan Iter. Automatic identification and presentation of Twitter content for planned events. In *Proceedings of the Fifth ACM International Conference on Web Search and Data Mining*, pages 655–656, 2011.
- [22] Hila Becker and Luis Gravano. Selecting Quality Twitter Content for Events. *International AAAI Conference on Weblogs and Social Media*, pages 442–445, 2010.
- [23] Hila Becker, Dan Iter, Mor Naaman, and Luis Gravano. Identifying content for planned events across social media sites. *Proceedings of the fifth ACM international conference on Web search and data mining, WSDM '12*, (533):533, 2012.
- [24] Hila Becker, Mor Naaman, and Luis Gravano. Event identification in social media. *Twelfth International Workshop on the Web and Databases*, 2009.
- [25] Hila Becker, Mor Naaman, and Luis Gravano. Learning similarity metrics for event identification in social media. *Proceedings of the third ACM international conference on Web search and data mining, WSDM '10*, page 291, 2010.
- [26] Hila Becker, Mor Naaman, and Luis Gravano. Beyond Trending Topics: Real-World Event Identification on Twitter. *International AAAI Conference on Web and Social Media*, pages 1–17, 2011.
- [27] Christopher M. Bishop. *Pattern Recognition and Machine Learning*. Springer-Verlag, Berlin, Heidelberg, 2006.
- [28] Paul Bohunsky and Wolfgang Gatterbauer. Visual structure-based web page clustering and retrieval. In *Proceedings of the 19th international conference on World wide web*, pages 1067–1068. ACM, 2010.
- [29] Michael Bolin, Matthew Webber, Philip Rha, Tom Wilson, and Robert C Miller. Automation and customization of rendered web pages. In *Proceedings of the 18th annual ACM symposium on User interface software and technology*, pages 163–172. ACM, 2005.

- [30] Leo Breiman. Random forests. *Machine Learning*, 45(1):5–32, 2001.
- [31] Leo Breiman, Jerome Friedman, Christopher James Stone, and Richard Olshen. *Classification and Regression Trees*. The Wadsworth and Brooks-Cole statistics-probability series. Taylor & Francis, 1984.
- [32] Mirko Bronzi, Valter Crescenzi, Paolo Merialdo, and Paolo Papotti. Extraction and integration of partially overlapping web sources. *Proceedings of the VLDB Endowment*, 6(10):805–816, 2013.
- [33] Zehuan Cai, Jin Liu, Lamei Xu, Chunyong Yin, and Jin Wang. A vision recognition based method for web data extraction. In *Advanced Science and Technology Letters*, volume 143, pages 193–198, 2017.
- [34] Mary Elaine Califf and Raymond J Mooney. Bottom-up relational learning of pattern matching rules for information extraction. *Journal of Machine Learning Research*, 4(Jun):177–210, 2003.
- [35] Chia-Hui Chang, Mohammed Kayed, Moheb R Girgis, and Khaled F Shaalan. A survey of web information extraction systems. *IEEE transactions on knowledge and data engineering*, 18(10):1411–1428, 2006.
- [36] Valter Crescenzi and Giansalvatore Mecca. Automatic information extraction from large websites. *Journal of the ACM (JACM)*, 51(5):731–779, 2004.
- [37] Valter Crescenzi, Giansalvatore Mecca, and Paolo Merialdo. Roadrunner: Towards automatic data extraction from large web sites. In *Proceedings of the VLDB Endowment*, 2001.
- [38] Walter Daelemans, Jakub Zavrel, Kurt Van Der Sloot, and Antal Van den Bosch. Timbl: Tilburg memory-based learner. *Tilburg University*, 2004.
- [39] Nilesh Dalvi, Ashwin Machanavajjhala, and Bo Pang. An analysis of structured data on the web. *Proceedings VLDB Endowment*, 5(7):680–691, March 2012.
- [40] Hal Daumé III. Notes on CG and LM-BFGS Optimization of Logistic Regression. 2004.
- [41] Leon Derczynski, Alan Ritter, Sam Clark, and Kalina Bontcheva. Twitter part-of-speech tagging for all: Overcoming sparse and noisy data. In *Proceedings of the International Conference Recent Advances in Natural Language Processing RANLP 2013*, pages 198–206, 2013.
- [42] Xin Dong, Evgeniy Gabrilovich, Jeremy Heitz, Wilko Horn, Ni Lao, Kevin Murphy, Thomas Strohmman, Shaohua Sun, and Wei Zhang. Knowledge vault: A web-scale approach to probabilistic knowledge fusion. In *Proceedings of the 20th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, KDD’14*, pages 601–610. ACM, 2014.
- [43] Emilio Ferrara, Pasquale De Meo, Giacomo Fiumara, and Robert Baumgartner. Web data extraction, applications and techniques: A survey. *Knowledge-based systems*, 70:301–323, 2014.

- [44] Jenny Rose Finkel, Trond Grenager, and Christopher Manning. Incorporating non-local information into information extraction systems by gibbs sampling. In *Proceedings of the 43rd Annual Meeting on Association for Computational Linguistics*, ACL '05, pages 363–370. Association for Computational Linguistics, 2005.
- [45] Sergio Flesca, Giuseppe Manco, Elio Masciari, Eugenio Rende, and Andrea Tagarelli. Web wrapper induction: a brief survey. *AI communications*, 17(2):57–61, 2004.
- [46] John Foley, Michael Bendersky, and Vanja Josifovski. Learning to extract local events from the web. In *Proceedings of the 38th International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '15, pages 423–432, New York, NY, USA, 2015. ACM.
- [47] Yoav Freund and Robert E Schapire. A decision-theoretic generalization of on-line learning and an application to boosting. In *European conference on computational learning theory*, pages 23–37. Springer, 1995.
- [48] Tim Furche, Georg Gottlob, Giovanni Grasso, et al. Diadem: Domain-centric, intelligent, automated data extraction methodology. In *Proceedings of the 21st International Conference on World Wide Web*, 2012.
- [49] Tim Furche, Georg Gottlob, Giovanni Grasso, Xiaonan Guo, Giorgio Orsi, Christian Schallhart, and Cheng Wang. Diadem: thousands of websites to a single database. *Proceedings of the VLDB Endowment*, 7(14):1845–1856, 2014.
- [50] Tim Furche, Georg Gottlob, Giovanni Grasso, Christian Schallhart, and Andrew Sellers. Oxpath: A language for scalable data extraction, automation, and crawling on the deep web. *The International Journal on Very Large Databases*, 2013.
- [51] Tim Furche, Giovanni Grasso, Michael Huemer, Christian Schallhart, and Michael Schrefl. Peace-ful web event extraction and processing as bitemporal mutable events. *ACM Transactions on the Web*, 10(3):16:1–16:47, 2016.
- [52] Tim Furche, Jinsong Guo, Sebastian Maneth, and Christian Schallhart. Robust and noise resistant wrapper induction. In *Proceedings of the 2016 ACM SIGMOD International Conference on Management of data*, 2016.
- [53] Wolfgang Gatterbauer and Paul Bohunsky. Table extraction using spatial reasoning on the css2 visual box model. In *Proceedings of the National Conference on Artificial Intelligence*, volume 21, page 1313, 2006.

- [54] Anna Lisa Gentile, Ziqi Zhang, Isabelle Augenstein, and Fabio Ciravegna. Unsupervised wrapper induction using linked data. In *Proceedings of the seventh international conference on Knowledge capture*, pages 41–48. ACM, 2013.
- [55] Tomas Gogar, Ondrej Hubacek, and Jan Sedivy. Deep neural networks for web page information extraction. In Lazaros Iliadis and Ilias Maglogiannis, editors, *Artificial Intelligence Applications and Innovations*, pages 154–163, Cham, 2016. Springer International Publishing.
- [56] Koraljka Golub and Anders Ardö. Importance of html structural elements and metadata in automated subject classification. In *International Conference on Theory and Practice of Digital Libraries*, pages 368–378. Springer, 2005.
- [57] Georg Gottlob and Christoph Koch. Logic-based web information extraction. *SIGMOD Record*, 33(2):87–94, June 2004.
- [58] Georg Gottlob and Christoph Koch. Monadic datalog and the expressive power of languages for web information extraction. *Journal of the ACM (JACM)*, 51(1):74–113, 2004.
- [59] Tin Kam Ho. Random decision forests. In *Proceedings of the Third International Conference on Document Analysis and Recognition - Volume 1, ICDAR '95*, pages 278–. IEEE Computer Society, 1995.
- [60] Frederik Hogenboom, Flavius Frasincar, Uzay Kaymak, and Franciska De Jong. An overview of event extraction from text. *CEUR Workshop Proceedings*, 779:48–57, 2011.
- [61] Chun-Nan Hsu and Ming-Tzung Dung. Generating finite-state transducers for semi-structured data extraction from the web. *Information systems*, 23(8):521–538, 1998.
- [62] Muhammad Imran, Carlos Castillo, Fernando Diaz, and Sarah Vieweg. Processing social media messages in mass emergency: a survey. *ACM Computing Surveys (CSUR)*, 47(4):67, 2015.
- [63] Ander Intxaurreondo, Eneko Agirre, Oier Lopez De Lacalle, and Mihai Surdeanu. Diamonds in the Rough : Event Extraction from Imperfect Microblog Data. *NaacI2015*, pages 641–650, 2015.
- [64] Silvia Julinda, Christoph Boden, and Alan Akbik. Extracting a Repository of Events and Event References from News Clusters. *The 23rd International Conference on Computational Linguistics (COLING 2014)*, pages 14–18, 2014.
- [65] Katharina Kaiser and Silvia Miksch. Information extraction-a survey. *Technical report, Vienna University of Technology, Institute of Software Technology and Interactive Systems*, 2005.

- [66] Gregory Katsios, Svitlana Vakulenko, Anastasia Krithara, and Georgios Paliouras. Towards Open Domain Event Extraction from Twitter: REVEALing Entity Relations. *Proceedings of the 4th International Workshop on Detection, Representation, and Exploitation of Events in the Semantic Web (DeRiVE 2015)*, 1363:35–46, 2015.
- [67] Lingpeng Kong, Nathan Schneider, Swabha Swayamdipta, Archana Bhatia, Chris Dyer, and Noah A Smith. A dependency parser for tweets. *Proceedings of the 2014 Conference on Empirical Methods on Natural Language Processing, EMNLP*, 2014.
- [68] Milos Krstajic, Christian Rohrdantz, Michael Hund, and Andreas Weiler. Getting there first: real-time detection of real-world incidents on Twitter. *2nd Workshop on Interactive Visual Text Analytics: Task-Driven Analysis of Social Media Content with Visweek’12*, pages 2–5, 2012.
- [69] Bernhard Krüpl, Marcus Herzog, and Wolfgang Gatterbauer. Using visual cues for extraction of tabular data from arbitrary html documents. In *Special interest tracks and posters of the 14th international conference on World Wide Web*, pages 1000–1001. ACM, 2005.
- [70] Nicholas Kushmerick. Wrapper induction: Efficiency and expressiveness. *Artificial Intelligence*, 118(1):15 – 68, 2000.
- [71] Nicholas Kushmerick. Finite-state approaches to web information extraction. In *Information Extraction in the Web Era*, pages 77–91. Springer, 2002.
- [72] Erdal Kuzey, Jilles Vreeken, and Gerhard Weikum. A fresh look on knowledge bases: Distilling named events from news. In *Proceedings of the 23rd ACM International Conference on Conference on Information and Knowledge Management*, pages 1689–1698. ACM, 2014.
- [73] Cody Kwok, Oren Etzioni, Oren Etzioni, and Daniel S. Weld. Scaling question answering to the web. *ACM Transactions on Information Systems*, 19(3):242–262, July 2001.
- [74] Alberto HF Laender, Berthier A Ribeiro-Neto, Altigran S Da Silva, and Juliana S Teixeira. A brief survey of web data extraction tools. *ACM Sigmod Record*, 31(2):84–93, 2002.
- [75] Jure Leskovec, Lars Backstrom, and Jon Kleinberg. Meme-tracking and the Dynamics of the News Cycle. *Proceedings of the 15th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 497–506, 2009.
- [76] Vladimir Levenshtein. Binary Codes Capable of Correcting Deletions, Insertions and Reversals. *Soviet Physics Doklady*, 10:707, 1966.
- [77] Hao Li and Heng Ji. Cross-genre event extraction with knowledge enrichment. In *Proceedings of the 2016 Conference of the North American Chapter of the Association for Computational Linguistics:*

- Human Language Technologies*, pages 1158–1162. Association for Computational Linguistics, June 2016.
- [78] Hao Li, Heng Ji, and Lin Zhao. Social event extraction: Task, challenges and techniques. In *Proceedings of the 2015 IEEE/ACM International Conference on Advances in Social Networks Analysis and Mining 2015*, pages 526–532. ACM, 2015.
 - [79] Qi Li, Heng Ji, and Liang Huang. Joint Event Extraction via Structured Prediction with Global Features. *Proceedings of the 51st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 73–82, 2013.
 - [80] Wei Liu, Xiafeng Meng, and Weiyi Meng. Vide: A vision-based approach for deep web data extraction. *IEEE Transactions on Knowledge and Data Engineering*, 22(3):447–460, March 2010.
 - [81] Colin Lockard, Xin Luna Dong, Arash Einolghozati, and Prashant Shiralkar. Ceres: Distantly supervised relation extraction from the semi-structured web. *Proceedings of the VLDB Endowment*, 11(10):1084–1096, June 2018.
 - [82] Inderjeet Mani and George Wilson. Robust temporal processing of news. In *Proceedings of the 38th Annual Meeting on Association for Computational Linguistics, ACL ’00*, pages 69–76. Association for Computational Linguistics, 2000.
 - [83] Christopher D. Manning, Mihai Surdeanu, John Bauer, Jenny Finkel, Steven J. Bethard, and David McClosky. The Stanford CoreNLP natural language processing toolkit. In *Association for Computational Linguistics (ACL) System Demonstrations*, pages 55–60, 2014.
 - [84] Mausam, Michael Schmitz, Robert Bart, Stephen Soderland, and Oren Etzioni. Open language learning for information extraction. In *Proceedings of the 2012 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning, EMNLP-CoNLL ’12*, pages 523–534. Association for Computational Linguistics, 2012.
 - [85] Andrew Kachites McCallum. MALLET: A Machine Learning for Language Toolkit. <http://mallet.cs.umass.edu>, 2002.
 - [86] Robert Meusel, Christian Bizer, and Heiko Paulheim. A web-scale study of the adoption and evolution of the schema.org vocabulary over time. In *Proceedings of the 5th International Conference on Web Intelligence, Mining and Semantics*, page 15. ACM, 2015.
 - [87] R Mooney. Relational learning of pattern-match rules for information extraction. In *Proceedings of the Sixteenth National Conference on Artificial Intelligence*, volume 328, page 334, 1999.
 - [88] Ion Muslea, Steve Minton, and Craig Knoblock. A hierarchical approach to wrapper induction. In *Proceedings of the third annual conference on Autonomous Agents*, pages 190–197. ACM, 1999.

- [89] S. Nagarajan and K. Perumal. A deep neural network for information extraction from web pages. In *2017 IEEE International Conference on Power, Control, Signals and Instrumentation Engineering (ICPCSI)*, pages 918–922, Sep. 2017.
- [90] Martina Naughton, Nicholas Kushmerick, and Joseph Carthy. Event extraction from heterogeneous news sources. In *proceedings of the AAAI workshop event extraction and synthesis*, pages 1–6, 2006.
- [91] Clayton Norris. Petrarch 2 : Petrarcher. *SSRN Electronic Journal*, 02 2016.
- [92] Ruchi Parikh. ET : Events from Tweets. *Proceedings of the 22nd international conference on World Wide Web companion*, pages 613–620, 2013.
- [93] Jakub Piskorski. Corleone-core linguistic entity online extraction. *Technical Report EUR 23393 EN*, 2008.
- [94] Jakub Piskorski and Martin Atkinson. Frontex real-time news event extraction framework. In *Proceedings of the 17th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 749–752. ACM, 2011.
- [95] Jakub Piskorski, Hristo Tanev, Martin Atkinson, and Erik Van Der Goot. Cluster-centric approach to news event extraction. In *Proceedings of the 2008 Conference on New Trends in Multimedia and Network Information Systems*, pages 276–290, Amsterdam, The Netherlands, The Netherlands, 2008. IOS Press.
- [96] Thomas Ploeger, Maxine Kruijt, Lora Aroyo, Frank De Bakker, Iina Hellsten, Antske Fokkens, J Hoeksema, and Serge ter Braake. Extractivism: Extracting activist events from news articles using existing nlp tools and services. *Proceedings of the Workshop on Detection, Representation, and Exploitation of Events in the Semantic Web*, 2013.
- [97] James Pustejovsky, Patrick Hanks, Roser Saurí, Andrew See, Robert Gaizauskas, Andrea Setzer, Dragomir Radev, Beth Sundheim, David Day, and Lisa Ferro. The TIMEBANK Corpus. *Corpus linguistics*, page 40, 2003.
- [98] Alan Ritter, Sam Clark, Mausam, and Oren Etzioni. Named entity recognition in tweets: An experimental study. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*, EMNLP ’11, pages 1524–1534. Association for Computational Linguistics, 2011.
- [99] Alan Ritter, Oren Etzioni, and Sam Clark. Open domain event extraction from twitter. *Proceedings of the 18th ACM SIGKDD international conference on Knowledge discovery and data mining - KDD’12*, page 1104, 2012.

- [100] Arnaud Sahuguet and Fabien Azavant. Building light-weight wrappers for legacy web data-sources using w4f. In *Proceedings of the 25th International Conference on Very Large Data Bases, VLDB'99*, pages 738–741, San Francisco, CA, USA, 1999. Morgan Kaufmann Publishers Inc.
- [101] Takeshi Sakaki, Makoto Okazaki, and Yutaka Matsuo. Earthquake shakes twitter users: Real-time event detection by social sensors. In *Proceedings of the 19th International Conference on World Wide Web, WWW '10*, pages 851–860, New York, NY, USA, 2010. ACM.
- [102] Philip A Schrodtt. Automated coding of international event data using sparse parsing techniques. In *annual meeting of the International Studies Association*, 2001.
- [103] Philip A. Schrodtt. Precedents, progress, and prospects in political event data. *International Interactions*, 38(4):546–569, 2012.
- [104] João M. C. Sousa, João R. Caldas Pinto, C. S. Ribeiro, and João M. Gil. Ancient document recognition using fuzzy methods. In *The 14th IEEE International Conference on Fuzzy Systems, 2005. FUZZ '05.*, pages 833–838, May 2005.
- [105] Jannik Strötgen and Michael Gertz. Heideltime: High quality rule-based extraction and normalization of temporal expressions. In *Proceedings of the 5th International Workshop on Semantic Evaluation, SemEval '10*, pages 321–324. Association for Computational Linguistics, 2010.
- [106] Jannik Strötgen and Michael Gertz. Multilingual and cross-domain temporal tagging. *Language Resources and Evaluation*, 47(2):269–298, 2013.
- [107] Hristo Tanev, Jakub Piskorski, and Martin Atkinson. Real-time news event extraction for global monitoring systems. *Joint Research Center of the European Commission Joint Research Center of the European Commission*.
- [108] Hristo Tanev, Jakub Piskorski, and Martin Atkinson. Real-time news event extraction for global crisis monitoring. In *Proceedings of the 13th International Conference on Natural Language and Information Systems: Applications of Natural Language to Information Systems, NLDB '08*, pages 207–218, Berlin, Heidelberg, 2008. Springer-Verlag.
- [109] Junfeng Wang, Chun Chen, Can Wang, Jian Pei, Jiajun Bu, Ziyu Guan, and Wei Vivian Zhang. Can we learn a template-independent wrapper for news article extraction from a single training site? In *Proceedings of the 15th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 1345–1354. ACM, 2009.
- [110] Yu Wang, David Fink, and Eugene Agichtein. Seeft: Planned social event discovery and attribute extraction by fusing twitter and web content. In *Twenty-Ninth AAAI Conference on Artificial Intelligence*, 2015.

- [111] Andreas Weiler, Michael Grossniklaus, and Marc H Scholl. Event Identification and Tracking in Social Media Streaming Data. *17th International Conference on Extending Database Technology, EDBT*, pages 282–287, 2014.
- [112] Roman Yangarber, Lauri Jokipii, Antti Rauramo, and Silja Huttunen. Extracting information about outbreaks of infectious epidemics. In *Proceedings of HLT/EMNLP on Interactive Demonstrations, HLT-Demo '05*, pages 22–23. Association for Computational Linguistics, 2005.
- [113] Yue You, Guangyan Huang, Jian Cao, Enhong Chen, and Jing He. GEAM : A General and Event-Related Aspects Model for Twitter Event Detection. *Web Information Systems Engineering – WISE 2013*, pages 319–332, 2013.
- [114] Quan Yuan, Gao Cong, and Zongyang Ma. Who, where, when and what: discover spatio-temporal topics for twitter users. *Discovery and Data Mining*, pages 605–613, 2013.
- [115] Yanhong Zhai and Bing Liu. Web data extraction based on partial tree alignment. In *Proceedings of the 14th international conference on World Wide Web*, pages 76–85. ACM, 2005.
- [116] Yanhong Zhai and Bing Liu. Structured data extraction from the web based on partial tree alignment. *IEEE Transactions on Knowledge and Data Engineering*, 18(12):1614–1628, 2006.
- [117] Kuo Zhang, Juan Zi, and Li Gang Wu. New event detection based on indexing-tree and named entity. In *Proceedings of the 30th annual international ACM SIGIR conference on Research and development in information retrieval*, pages 215–222. ACM, 2007.
- [118] Shuyi Zheng, Ruihua Song, and Ji-Rong Wen. Template-independent news extraction based on visual consistency. In *Proceedings of the National Conference on Artificial Intelligence*, volume 22, page 1507, 2007.
- [119] Jun Zhu, Zaiqing Nie, Ji-Rong Wen, Bo Zhang, and Wei-Ying Ma. Simultaneous record detection and attribute labeling in web data extraction. In *Proceedings of the 12th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 494–503. ACM, 2006.