Università degli Studi di Roma Tre

Scuola Dottorale in Ingegneria Informatica

XXVII Ciclo

Crowdsourcing Large scale Data Extraction from the Web: Bridging Automatic and Supervised Approaches

Dottorando:
Disheng Qiu

Tutor:
Prof. Paolo Merialdo

Gruppo di Basi di Dati
Dipartimento di Ingegneria

May 2015

Declaration of Authorship

I, Disheng Qiu, declare that this thesis titled, 'Crowdsourcing Large scale Data Extraction from the Web: Bridging Automatic and Supervised Approaches' and the work presented in it are my own. I confirm that:

- This work was done wholly or mainly while in candidature for a research degree at this University.
- Where any part of this thesis has previously been submitted for a degree or any other qualification at this University or any other institution, this has been clearly stated.
- Where I have consulted the published work of others, this is always clearly attributed.
- Where I have quoted from the work of others, the source is always given. With the exception of such quotations, this thesis is entirely my own work.
- I have acknowledged all main sources of help.
- Where the thesis is based on work done by myself jointly with others, I have made clear exactly what was done by others and what I have contributed myself.

Signed: _____

Date: _____

Università degli Studi di Roma Tre

Abstract

Dipartimento di Ingegneria

Crowdsourcing Large scale Data Extraction from the Web: Bridging Automatic and Supervised Approaches

by Disheng QIU

The Web is a rich source of data that represents a valuable resource for many organizations. Data in the Web is usually encoded in HTML pages, thus they are not processable; a data extraction process, which is made by software modules called wrappers, is required to use these data. Several attempts have been conducted to reduce the efforts of generating wrappers. Supervised approaches, based on annotated pages, achieve high accuracy; but the costs of the training data, i.e. annotations, limit their scalability. Unsupervised approaches have been developed to achieve high scalability, but the diversity of the data sources can drastically limit the accuracy of the results. Overall, obtaining high accuracy and high scalability is challenging because of the scale of the Web and the heterogeneity of the published information.

In this dissertation we describe a solution to address these challenges: to scale to the Web we define an unsupervised approach that is built considering several wrapper inference techniques; to control the quality we define a quality model that understands at runtime if human feedback is required; feedback is provided by workers enrolled from a crowdsourcing platform. Crowdsourcing represents an effective way to reduce the costs for the annotation process, but previous proposals are designed for experts and they are not suitable for the crowd, in fact, workers from crowdsourcing platforms are typically non-expert.
An open issue to scale the generation of wrappers is the collection of the pages to wrap, we describe an end-to-end pipeline that discovers and crawls relevant websites in a case study for product specifications.

An extensive evaluation with real data confirms that: (i) we can generate accurate wrappers with few simple interactions from the crowd; (ii) we can accurately estimate workers' error rate and select at runtime the number of workers to enroll for a task; (iii) we can effectively start by considering unsupervised approaches and switch to the crowd to increase the quality; and (iv) we can discover thousands of websites from a small initial seed.

Acknowledgements

If I look back to these three years, I have many people to thank.
Starting from the beginning. I want to thank Paolo and Lorenzo for their initial push that inspired me to begin my PhD program and a big thank to Paolo and Valter for their passion and guidance that continued to help me during these years and that led me to complete this thesis.
I want to thank Divesh and Srinivas that guided me during my internship in AT&T, those months gave me the confidence that I was looking for my work and (a good side effect) made me appreciate Indian food.
A thank to Francesca and Luca, colleagues in our Lab but also good friends that accompanied me during the first years of my program.
With Luca I want to thank all the team behind Wanderio (Matteo, Giorgio, Matteo, Giovanni, Costanza, Nicola, Simone), working with them was as a breath of fresh air to my research, I was always in touch with real problems.
A thank to other colleagues and professors in my department, Paolo, Luca, Riccardo, Emanuel, Roberto.
I want to thank all the students that I met and guided during these years, in particular Lorenzo and Andrea that I have guided during their thesis but now they guide me with their energy. Another big thank to Xinjie that brought art to my last years and made everything more colourful.
A special thank to my family: my Mother that with her sacrifices thought me the importance of the family; my Brother and my nieces, Diana and Livia, that were able to make me smile when I was down; my cousins, Roberto, Luca, Wangwei, Alessandro, that were friends during all these years.
I want to conclude thanking Kelly that made me think of other things other than my work and Ivan that always reminded me of the importance of having someone to look up to.
A thank to Francesca, Kelly, Jiajia and Roberto for reading through the thesis.

Contents

List of Figures

List of Tables

Notation

p	an HTML page
U	all the input templated pages
I	sample pages with $I \subset U$
r	extraction rule
v	extracted value
$r(p)$	rule r applied to p
A	an attribute to extract
$\mathcal{R}$	set of all the candidate rules
$\mathcal{R}_A$	set of all the candidate rules for the attribute A
$\mathcal{R}(U)$	candidate rules applied to U
V	the set of all the extracted values by $\mathcal{R}$ on U
MQ	membership query
l	a label so that $l \in \{+, -\}$
v^l	a labeled sample value, value from p_v
L	set of labeled sample values
$L(r)$	r admissible to L
R_L	the set rules in $\mathcal{R}$ that are admissible wrt L
L^k	training sequence (t.s.), an ordered sequence of K labeled sample
$P(r)$	probability of the rule r to be the correct rule
$P(r\|L^k)$	probability of the rule r to be the correct rule observed a training sequence L^k
$P(v^l\|r, L)$	the likelihood of acquiring the labeled value v^l_A conditioned to r and L
I	representative sample set
$D^P(r_i, r_j)$	disagreement set, set of pages that make observable differences among r_i and r_j
L^w	the t.s. generated by a worker w
λ_r	target probability of correctness

λ_{MQ}	maximum budget for each worker per attribute
t	a task submitted to the crowd
N	number of attributes per task on the crowd
K	number of redundant attributes per task
$\mathcal{G}$	the tasks attribute bipartite graph
$\mathcal{G}$t	the connected component of $\mathcal{G}$ that includes t
$\mathcal{W}_A$	the set of workers enrolled for an attribute A
η	workers' error rate
η^*	expected workers' error rate
T	a text value type
Γ	the *Disjunctive Types Hierarchy* with $\Gamma = \{T_0, \ldots, T_n\}$
$\Lambda(U)$	set of LFEQs generated from U so that $\Lambda(U) = \{L_0, \ldots, L_m\}$
$\lambda(v)$	set LFEQs associated to the value v
t	a triple made of $< s, p, o >$, subject, predicate, object

To my family

Chapter 1

Introduction

The Web is a valuable source of information. Its scale, the variety of the information and the possibility of adopting tools for big data analytics, inspired many organizations to integrate this information in their internal processes. In fact, automatically process this information could provide a tremendous competitive advantage. Unfortunately, most of the Web contents are embedded in HTML documents; they are designed to be browsed and consumed by humans and not to be automatically processed by machines. Data Extraction studies techniques to collect structured data from the Web, making this information accessible. These techniques are adopted by many organizations to enrich their internal databases with data published on the Web. To extract the structured data from these pages, a data extraction system relies on a software module called wrapper: it filters non relevant information inside the HTML pages and provides a structure to relevant information. The structured data, extracted by wrappers, are then automatically processable and can be integrated into an existing pipeline to augment the capability of the entire system.

Nowadays, many organizations extract data from the Web for different purposes: to filter non relevant information in news applications, to compare product prices in different online shopping websites, to collect data for marketing analysis, to generate an interface for the mobile version of a website, etc.

While defining a wrapper for a single website is not complex, the process of generating wrappers for many websites is challenging. Each website is characterized by its own peculiarities, thus a different wrapper should be defined for each website making the process costly and error prone. The possibility of harvesting structured data from a large number of sources opens interesting opportunities for new applications and improvements to the existing solutions.

This dissertation faces these challenges. We present a large scale data extraction system supervised by humans enrolled from a crowdsourcing platform, the goal is to "scale out" the process of generating wrappers by shifting from few expert humans to a vast number of non-expert workers. To scale the generation process to the Web we bridge the gap between automatic wrapper generation approaches with supervised approaches by adopting a *Quality Model* that understands when we can completely rely on unsupervised approaches and when human feedback is required. If feedback is required, we adopt a crowd based wrapper generation solution, the system controls the quality of the output wrappers and aims to minimize the work required to the crowd to generate the output wrappers.

In this chapter: we first describe the opportunities that scaling the generation of wrappers on the Web could provide in real use cases (Section 1.1); we introduce the wrapper generation problem and open issues related to the process of scaling the extraction task to the Web (Section 1.2); we briefly describe the state of the art and solutions to scale the generation of wrappers (Section 1.3); we introduce Crowdsourcing (Section 1.4) as an opportunity to scale existing systems by plugging in a human component to control and guide the extraction process. We conclude describing an overview of the approach and highlighting the contributions of this thesis (Section 1.5) and the outline of the dissertation (Section 1.6).

1.1 Big Data Opportunities and Challenges

The advent of the Big Data era, and the possibility of augmenting existing solutions with information collected from the Web, opens new opportunities to intriguing applications. For some domains there are large websites that publish information about many instances in that domain (the *head* of the distribution), e.g. IMDB[1] for movies, reducing in this way the need of scaling the generation of wrappers to many sources, i.e. the extraction could be done only in few big websites. In the majority of the cases, domains are characterized by a high degree of fragmentation (the *tail* of the distribution). Many domains are characterized by a *long tail*, i.e. many small websites provide relevant information that is not present in large websites (the head of the distribution [12]). Examples are domains with a geographic focus, e.g. restaurants or real estates, with niches, e.g. products or with subjective information, e.g. reviews. Even for well established domains, where big websites, e.g. IMDB, have a good coverage of all the instances published in

[1] www.imdb.com

that domain, the coverage is not complete. For instance, in IMDB many local movies or movies released only in specific countries are still missing.

To understand the opportunities that a large scale data extraction pipeline could provide, consider 3 interesting applications; they represent different existing use cases that would greatly benefit from scaling the data extraction to the Web.

Transportation Websites related to transportation that expose information of local shuttles, coaches or private drivers are examples of a fragmented domain. Each website is characterized by a local coverage of the services that they directly provide, e.g. shuttles from an airport to the closest city center. There are websites that provide an international coverage of these services[2], but the number of shuttles that are found in these websites is limited compared to the web presence of these services. Scaling the extraction of the data published by transportation websites could create an interesting application for many customers. In fact, by providing a comparison of the prices and the estimated duration time of all kind of transportations, the coverage could be global. The service could provide all the alternatives found on the web. Customers could select the best option for their needs, without extensive search sessions in several websites.
Another interesting use case for the transportation domain are the flight related websites. In fact, even when there are big aggregators[3] or OTA (Online Travel Agency)[4] that provide a comparison platform for many airlines, some flight companies, e.g. low costs airlines, are missing from these platforms. This reduces the possible alternatives returned to the end users. Adopting a Data Extraction pipeline at web scale for flights companies would provide a detailed comparison of all the available companies.

Products There are many price comparison websites that aggregate information about products. The product domain is characterized by the presence of multiple websites that publish information about the same product, e.g. several stores sell a specific camera model. Many of those online stores let users comment on the sold products, thus users' reviews of the same product are spread in many online shopping websites. Collecting reviews for sentiment or opinion mining means extracting reviews from many sources. Even if we consider big web portals, e.g. Amazon, where many products are available, the coverage of niche products (products for specialists) is often low and reviews from other websites on the same products still provide a great value [12]. If we consider an comprehensive database of online products and reviews collected from all the web, users could easily compare in a single website, prices, reviews and product

[2] www.getyourguide.com
[3] www.kayak.com
[4] www.expedia.com

specifications. Users could avoid multiple extensive search sessions in different shopping websites.

For Metadata: The definition of common standards to semantically annotate HTML pages, allowed an increasing presence of websites with metadata. An example is schema.org, a standard de facto that is adopted by many websites to provide a semantic meaning to information published by their pages. To increase the adoption of schema.org, modern search engines reward websites that correctly uses these tags with *rich snippets*. Rich snippets are returned by the search engines in their results, they provide partial previews of the structured data that is embedded in the indexed pages. These previews provide more visibility to the websites and the desired effect is an increased click rate wrt the impressions on the results of the search engine. Semantic tags are adopted by the search engines to understand the content of the HTML pages, thus improving the results of the search query. While there is an increasing usage of these tags, websites that use correctly these metadata are just a fraction of all the websites in Web. There are multiple reasons for these limitations: (i) the adoption is often limited to those tags that are highly rewarded by the search engine, e.g. reviews (ii) webmaster often "forces" a misleading usage of some tags to get better previews (iii) annotating all the values in every HTML pages is time consuming, costly and does not provide an advantage wrt a partial annotation. The possibility of extracting data from HTML pages could provide a powerful tool to annotate values in HTML pages, thus reducing the work required by the webmaster.

1.2 Wrappers at Web Scale

The Web is a rich source of information; wrappers are tools to collect structured data from it. Ideally, if we suppose that each web page is an independent page, a wrapper should be defined for each web page. This is obviously not a practicable solution, thus, to increase the efficiency of the extraction process, we work with a subset of all the pages on the web, the *template based* HTML pages. Example of these pages are script generated pages; a single website could contain millions of pages that are generated from the same script (they share a common template), the script embeds the values from an underlying database into the HTML pages. In this way, a single wrapper generated for this kind of pages can extract the content in all the millions of pages by exploiting the regularity inside the template.
A wrapper can be expressed as an extraction pattern, or extraction rule, that applied to one or many target pages (from the same template) extracts the content of interest and

provides a structure to the extracted information. The extraction rule exploits the fact that an HTML page can be represented as a DOM tree, thus the extraction process apply the rule on the DOM tree to select the target values. Different approaches represent wrappers in different ways, from regular expressions to XPaths, from queries similar to SQL to new languages defined explicitly for the extraction task. All these approaches rely on the fact that an HTML page can be parsed, filtered and the content embedded in the HTML tags can be extracted.
Thanks to the presence of regularities on template-based pages, a new wrapper is not required for each new HTML page, but the process of generating a wrapper for many pages is not trivial. In fact, the script that generates these pages often contains some irregularities, thus to accurately extract all the content the wrapper has to internally "manage" these irregularities. A wrapper is crafted to extract the content by handling the irregularity of a script, hence the wrapper can be applied only on the pages generated by that script, i.e. a wrapper is template specific.

For a given domain there are thousands of web sites that publish information about instances of that domain [12]. As we observed, there is value in the long tail, but the number of possible sites and the diversity among the published information from different sources make the extraction process challenging. To build a comprehensive catalogue of the information published by many websites of a domain, three issues have to be addressed: (1) a wrapper has to be generated to extract the structured information from the relevant pages (2) relevant websites have to be discovered, (3) a crawling strategy has to be adopted to navigate the websites to collect relevant pages. If we consider a manual solution, and we write a software that crawls and wraps structured content from an input website, and we repeat this work for many websites, this is a costly solution, but still a practical if we do not consider the fact that websites are often updated. In fact, websites are subject to continuous updates and these updates make the defined extraction rules to not work correctly for the existing wrappers or crawlers. Nevertheless, increasing the number of wrapped websites increases the probability of changes that can "break" a wrapper for a website. If we consider this *mutability* of the Web and the high cost of the manual work, these two issues make manual solutions not practicable for real settings.

For reference about the discovery and crawling consider the Chapter 6. Scaling the generation of wrappers on the Web for many domains is an open challenge and solutions to this challenge have been studied for many years, by both academia and industry.

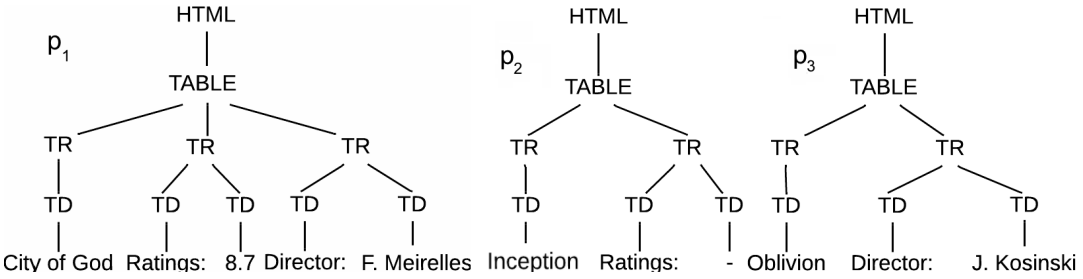


FIGURE 1.1: Running example with the DOM tree of three sample pages

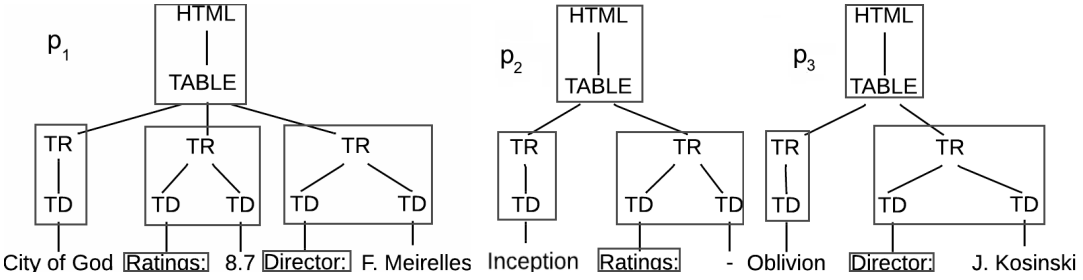


FIGURE 1.2: Running example with the template nodes highlighted

1.3 Wrapper Generation

In literature there are many approaches that try to "scale up" the process of generating wrappers. In general we can divide these approaches in 3 categories: (1) unsupervised approaches, (2) supervised approaches and (3) automatic annotations to supervise the generation.

1.3.1 Unsupervised Approaches

Unsupervised approaches rely on the fact that pages from a common template have a similar structure. The structure is not identical, but it is similar enough to define extraction rules that can separate the content from the HTML template. In Figure 1.1 we have a DOM representation of three HTML pages from the same template that contains information about movies.

Unsupervised approaches analyze the HTML content of multiple pages to understand which potion in the page is a template node and which portion is not. Template texts are filtered while non template values are aligned and extracted in a structured form.

In Figure 1.2 we can observe the HTML template recognized by a wrapper inference system. In Figure 1.3 a wrapper that extracts the values from our previous example is described by means of a regular expression. The tag #DATA models the values to extract and the tag (*)? models optional fragments.

```
<html>
    <table>
        <tr><td>#DATA</td></tr>
        (<tr><td>Ratings:</td><td>#DATA</td></tr>)?
        (<tr><td>Director:</td><td>#DATA</td></tr>)?
    </table>
</html>
```

FIGURE 1.3: Regular expression that extracts target values from the running example

The advantage of unsupervised approaches is that they do not require human supervision for the generation of wrappers. They obtain high scalability; in fact, for the extraction task the human factor is minimized. A well known issue is that unsupervised solutions are not reliable, they extract non relevant data if part of the content embedded in HTML templates is non relevant, e.g. the generation time of the HTML page, advertisements, etc. Another issue is related to the semantic of the extracted data; in fact, a label is required to support the automatic elaboration of the extracted data. For instance, if we suppose to extract the values City of God, Inception and Oblivion from pages describing movies, a label is required to assign the meaning Movie Title to the extracted values. These issues motivate a manual post-processing of the extracted content that limits the overall scalability of these solutions.

1.3.2 Supervised Approaches

As for the unsupervised ones, supervised approaches rely on the presence of an HTML structure, but they exploit also the possibility of asking users to provide feedbacks. In fact, humans can help the inference algorithm by providing annotations on the values to be extracted. From these annotations the inference algorithm can understand the values to be extracted and accurately select the target values.

Suppose that we want to extract all the directors from the pages in Figure 1.1. A user could select from the p_1 the value F.Meirelles, thus the inference system generates an XPath rule r_1=/html/table/tr[3]/td[2] that extracts the target value from the p_1. The wrapper inference system applies the extraction rule to p_2, p_3 and from the feedback of the user it refines r_1. In fact, when r_1 is applied to p_3 no value is extracted, thus the user can provide an annotation on the new value J. Kosinski and the inference algorithm refines the previous rule accordingly defining r_2=//*[contains(text(),"Director:")]/../../td[2].

Common issues for traditional supervised approaches are the cost required by the annotation process and the validation process, i.e. to make sure that the extraction rule works correctly in all the input pages. In fact some technical skills are required to incrementally fix the extraction rule and validate it on the templated pages. Considering

our previous example, a solution based on a human that checks the extracted values on every templated page is not efficient. The typical approach is to provide a visualization of the extracted content on all the HTML pages, as a vector of extracted values; the user selects the values to fix and the system updates the visualization for each iteration.

1.3.3 Automatic Annotations

The manual annotation process is expensive, because it requires continuous feedbacks from human experts. A possible way to address this problem is by defining automatic annotators that recognize portions of the HTML page. Annotations are meant to replace the human work required by the supervised approaches.

instance	title	director
m_1	City of God	F. Meirelles
m_2	Oblivion	J. Kosinski

TABLE 1.1: A database with values that match the running example

Suppose that we have a database of movies represented by Table 1.1, the values inside the database, title and director of movies, can be used to supervise the generation of wrappers for these attributes. The rules are then applied to the entire website, to discover new movies with new directors that are not present in the database. Obviously the values could not match perfectly and non relevant information could be labelled, thus these solutions often provide techniques to deal with the expected noise introduced by the annotation process.

These solutions are scalable, because no human feedback is required. But they still require an expensive initial set up. For instance, the discovery of a seed database and the additional domain knowledge required to reduce the noise introduced by the annotation process have to be manually set up. The inference algorithm could consider domain knowledge to discard erroneous annotations, e.g. the fact that a movie has a single title, or that normally a movie has one director but there are also movies with two directors. This knowledge is processed by the inference algorithm to improve the extraction process. A Knowledge Base can be complex, thus it requires an initial set up from a domain expert.

1.3.4 Open Challenges

Other than the previous issues related to each technique, all the state of the art approaches do not address the following issues:

- Sampling: the inference algorithm is a learning process where we have a training set and a test set, i.e. the training is the set of HTML pages adopted to infer the extract rule and the test set is where the rules are actually used to extract the data. Since the inference step is often more costly than the extraction step, state of the art approaches rely on a fixed training set. The training set is randomly collected or it is biased by the crawling strategy that is adopted to collect the pages. The web is characterized by a high degree of variability, thus if a small number of pages is enough for a website, the same number of pages is not enough for other websites. Successfully learning a wrapper requires a training set that is representative of the entire set of pages, so that the learned wrapper can work correctly on all the pages.

- Experts: all the state of the art approaches rely on expert users, i.e. users that have knowledge of the extraction problem and the underlying solution. For unsupervised approaches the expert has to understand the output of the system and correctly set up the values to be fed in the next step of the data extraction pipeline; for supervised approaches the supervision requires a knowledge of the inference algorithm and the possible inputs required by the system; for automatic annotators the expert has to find a database and/or a knowledge base that represents the data to be extracted.

All these issues and the one described in the previous sections make the generation of wrappers challenging at Web scale.

1.4 Crowdsourcing

The advent of the Crowdsourcing with the possibility of accessing a huge amount of human labour opens the opportunity to scale human work for many open challenges. It has been successfully applied to digitalize legacy textbooks, to tag images and to semantically tag sentences. The principle behind these systems is that many humans can be involved in simple tasks, e.g. labelling data or images, exploiting an IT infrastructure. Humans can be paid to complete these tasks or are self-motivated by some subjective gain, for example the possibility of obtaining a service, e.g. solving a captcha lets the users access the services in the websites, or for personal satisfaction, e.g. playing a game. In general, users can be motivated by different goals, but all crowdsourcing platforms are characterized by few principles: (1) users are voluntarily hired, they accept a job so there is not a selection of the individual to complete the task; (2) users can make mistakes completing their tasks, the mistakes depends on many factors, the kind of the

task, the skills of the works; (3) the more complex is the task required to complete a work the more the worker has to be specialized.

As observed, Data Extraction at web scale is a challenging goal. Automatic approaches can not be applied to the entire web, the quality is often not predictable and most of the times they still require some human interactions. Supervised approaches generally obtain a better (and controllable) quality, but they are hard to scale to all the Web for the human factor. Crowdsourcing represents an opportunity to scale the human factor by "scaling out" the wrapper generation task, i.e. by moving from a small set of expert users to many non expert users. The possibility of learning wrappers from non expert users reduces, on one hand, the costs required by the human intervention and, from the other hand, it increases the scalability of the solution thanks to the high availability of non expert humans.

Crowdsourcing for Data Extraction represents an opportunity, but it is also a challenge. In fact, we have to deal with: **(non experts)** the wrapper generation technique should be designed to be easily used by non experts; **(the cost)** the cost is still an issue and it should be controlled and minimized; **(the quality)** users make mistakes, the wrapper generation system should be tolerant to errors.

1.5 Overview and Contributions

Overview In this dissertation we propose a large scale data extraction system supervised by workers enrolled from a crowdsourcing platform.
We defined an original model that describes the process of obtaining a wrapper from templated HTML pages. The model is based on works from the statistical learning community [13] and adopts supervised learning to infer wrappers with training data generated by mini-tasks submitted to the crowdsourcing platform.
The learning process starts from a first annotation on one templated page that selects the value to extract. From this initial annotation, our inference algorithm generates a pool of possible extraction rules by exploiting the presence of an underlying HTML template. The rules extract all the same value on the first page, i.e., the annotated value, but when they are applied to other pages they behave differently. The variation represents an uncertainty of the correct value to extract on that page.
In our model, the generation of a wrapper is reduced to the selection of the best extraction rule among the pool of the candidate rules. This process is supported by simple mini-tasks that consist of membership queries (*MQ*) [14], e.g., "Observe this page: is the string "Dean Martin" a correct value to extract?". Membership queries admit only a yes/no answer, thus simplifying the interactions required by the enrolled workers. The

learning process interacts with a human worker until it is certain of an extraction rule or it finds that no rule reflects the feedbacks provided by the worker. Our system takes into account two aspects of the learning process: the *Cost*, in terms of number of *MQ* required to terminate the computation; the *Quality*, in terms of the expected probability of correctness of the output wrapper.

Cost	Quality
Active Learning	Quality Model
Distant Supervision	Redundancy
Dynamic Recruitment	Sampling

TABLE 1.2: A summary of the presented features related to cost and quality

In Table 1.2 we summarize the features of our system:

- **Quality Model**: We defined a *Quality Model*, based on the Bayesian Model, to control the quality of the output wrapper. The *Quality Model* computes the probability of correctness of each candidate rule and takes into account the expected error rate of the worker. The system requires more feedback until a termination condition is not met.

- **Active Learning**: The Bayesian Model computes the probability of correctness of the candidate rules and models the uncertainty of the candidate extraction rules. To reduce the costs, i.e. the number of MQ required to infer the correct wrapper, we adopt Active Learning. The most uncertain queries are selected to minimize the number of *MQ* required to select the correct extraction rule.

- **Redundancy**: To deal with mistakes introduced by workers, we adopt redundancy, i.e. we enroll multiple workers on the same task and we compare their answers. We adopt a technique based on Expectation Maximization (EM) to estimate at runtime the workers' error rate and the quality of the output wrapper from multiple workers. We exploit the mutual dependency between the answers provided by the workers with their expected error rate and the expected quality of the output wrapper. Based on the previous *Quality Model* we can understand at runtime the number of redundant workers to engage for a single task, i.e. we engage other workers if the termination condition is not met.

- **Distant Supervision**: We provide an alternative to a solution based only on humans. We bridge the gap between automatic approaches and supervised approaches by adopting Distant Supervision. We defined several automatic responders inspired by state of the art wrapper inference systems and we adopt them to provide answers to the posed *MQ*. We collect the answers from these responders

and we adopt EM to estimate the responders' error rate and the quality of the best extraction rule.

- **Dynamic Recruitment** The number of workers to engage for a single task can be statically set, but if there is a complex task then it is likely that that there is a quality loss. To address this issue, the *Quality Model* selects at runtime whether many real workers are needed, i.e. if the system is certain of the results, no human intervention is required.

- **Sampling**: To make sure that the output wrapper is going to work correctly, we define a sampling algorithm to select the training set that can guarantee the quality of the wrapper on the test set. The right training set is the set of pages that "shows" all the uncertainty present in all the templated pages.

We developed a running prototype that relies on workers enrolled from a crowdsourcing platform to infer wrappers, we provide an extensive evaluation with a real dataset that confirms that: we can generate accurate wrappers with just a few *MQ*; the inference algorithm deals with erroneous workers by estimating the workers' error rate and selecting at runtime the right number of workers to enroll for a single task; we can further reduce the human effort by considering automatic responders, the number of *MQ* on average is just 2.38 with the F of the output wrappers close to 1.

Contribution Part of this thesis has been previously published in conferences and journals. For instance the Active Learning model, the Quality Model and the sampling algorithm are described in [15]. A demonstration of the system has been proposed in [16]. An extended journal version with the noisy tolerance and the *EM* technique to estimate at runtime workers' error rate is described in [17].
Some works [18, 19] are not included in the dissertation, but they provided precious experiences to the results obtained with this thesis.
Other parts of the dissertation are under submission, for instance: Chapter 5 with the description of the Distant Supervision technique and Chapter 6 with the discovery, crawling and extraction of product specifications [20]. The main contribution described in Chapter 6 has completed during an internship at AT&T Labs in New Jersey.

1.6 Outline

In Chapter 2 we describe the state of the art for wrapper generation, we present a subset of the techniques organized in Unsupervised, Supervised and Annotations based

approaches. We also provide a brief introduction to some concepts required for the understanding of the thesis, e.g. Active Learning, Crowdsourcing, Crawling and Discovery of relevant sources on the Web.

In Chapter 3 we introduce the simplest wrapper inference model with a single noisy worker. We describe the Active Learning algorithm and the Quality Model. We describe the sampling algorithm that finds a representative sample set. We provide evaluations with real data, we show that our system can learn wrapper with few MQ, around 5 MQ for each attribute.

In Chapter 4 we extend the previous chapter by considering multiple noisy workers. The chapter describes a technique, ALFRED, that dynamically enrolls several workers on the same task and estimates their error rate by adopting an Expectation Maximization (EM) approach. Furthermore, the chapter describes a scheduling technique that further reduces the cost by optimizing the redundancy for a given set of tasks. We provide evaluations with real workers, we show that our system can effectively infer a wrapper with multiple noisy workers. Erroneous workers and spammers do not affect the quality of the output wrapper, ALFRED deals with the noise by enrolling additional workers on uncertain tasks.

Chapter 5 adopts ALFRED to combine unsupervised approaches with human supervision. We describe a Distant Supervision approach that further reduces the human work required to generate wrappers. The chapter describes several automatic responders. ALFRED combines answers from multiple automatic responders to estimate the responders' error rate and to check the expected quality of the output wrapper. we combine multiple responders to increase the average F and to understand when the system is not confident of the results. We provide experimental evaluation with real data: the average F of wrappers generated by each automatic responder is around 0.9; for 70% of the cases our technique returns wrappers with an F of 0.99; for the other 30% of the cases the system engages human workers to reduce the uncertainty.

Chapter 6 describes a case study related to the extraction of product specifications from the Web. We describe an end-to-end pipeline that discover, crawl and extract product specifications from the Web. We adopt: a search API and backlinks to discover relevant websites; a domain specific crawler to collect target pages on the relevant website; classifiers to recognize specifications inside the target pages. In our evaluation we discovered and crawled 2719 websites for a total of $1M$ product specification pages. We describe two techniques to extract specifications and propose a hybrid approach with F greater then 0.9.

Chapter 7 concludes the thesis with discussions about open problems and possible future directions.

Chapter 2

Related Work

This dissertation describes a large scale data extraction system, the considered techniques are from different fields of research. In this chapter we provide deeper description of the state of the art and an overview related to: (1) **Wrapper Generation** we describe previous techniques for wrapper induction; (2) **Active Learning** we provide an overview of the state of the art and the techniques adopted in this dissertation; (3) **Discovery and Crawling** we provide a description of previous techniques and state of the art; (4) **Crowdsourcing** we describe other crowdsourced systems in literature and we provide a description of common coordinates among different systems.

2.1 Wrapper Generation

A lot of work has been done to study techniques to effectively extract content from the Web.

The initial works in Data Extraction studied formalism and languages to manually define wrappers. Those solutions simplified the process of writing a wrapper so that the parsing of HTML content was done by extraction rules. These approaches provided formalisms to select fields inside the pages and to filter portion of the content [1, 21, 22]. They relied on the fact that the end users of the wrapper system were developers, which are able to interact with the system writing code. This research branch led to the adoption of well known and affirmed standards, such as XPath or Regular Expressions, providing a common formalism to define wrappers.

A second trend in Data Extraction focused on scaling the generation of new wrappers. Previous works studied languages to define a wrapper, but an open issue was to scale this process. Most of these solutions exploited the presence of HTML templated pages,

thus the wrapper can be applied to many target pages. These solutions described inference algorithms that generate wrappers from a set of examples pages, the training set. Among them we can distinguish two trends: unsupervised approaches, that automatically extracted the content from non-relevant data by exploiting the presence of an underlying HTML template [4–6]; supervised approaches, that refined the extraction rule for pages of the same template by asking feedback to an expert human [2, 3].

Unsupervised approaches can be easily adopted to extract data from many websites without human intervention. The ambition was to automatically wrap the entire Web with no human intervention. Many issues made this goal unrealistic, in fact human intervention was always required: to provide the right semantic to the extracted data, to fix errors in the extracted data. The quality of the output was not stable, for instance, in some websites the output was perfect but in others the quality could dropped. Feedback could not be easily integrated to fix erroneous wrappers. Supervised approaches were the most accurate solutions. Feedback is required to train the inference algorithm to recognize the values to extract; the task required several (even complex) interactions so that the systems could effectively integrate feedback provided by human experts. It was obvious that automatic approaches could not reach the quality of supervised approaches.

The latest trend in wrapper generation is to scale the process to the Web [8–11]. Most of these techniques rely on Knowledge Bases or existing Databases. The principle behind is that supervised techniques can be guided by a Domain Knowledge instead of a human expert. Those solutions can easily scale over the number of the sources, but it is hard to define a good Domain Knowledge and they do not scale well for multiple domains, i.e. a Domain Knowledge has to be defined for each new domain.
An interesting alternative to the Domain Knowledge is to adopt the redundancy of the Web to generate wrappers [10, 11]. Instead of relying on a crafted Domain Knowledge, these solutions exploit the fact that instances on the Web are published by multiple sources, thus it is possible to infer wrappers for different websites simultaneously exploiting common fields. While no experts are required, the quality of these solutions often is still not controllable, i.e. the quality depends on the quality of the redundancy and the considered domain.

2.1.1 Supervised Wrapper Generation

Stalker [1]: Stalker is a wrapper inference system supervised by feedback provided by human experts. A wrapper is described by two patters, one *prefix* that matches the HTML content before the extracted content, and one *suffix* that matches the HTML content after the extracted content. The inference algorithm finds the correct wrapper

with the right *prefix* and *suffix* that applied to the templated pages it extracts the correct value in all the pages. The system requires a training set with the extracted content annotated by human users. The inference algorithm infers an initial wrapper that is iteratively refined considering the provided annotations. In each step the wrapper is refined and scored considering the annotations in the training set.

Muslea et al [2]: The authors described an Active Learning approach based on Strong and Weak Views for wrapper induction. Strong views are features that can be adopted to find the correct extraction rule, while weak views do not have enough information to find correct extraction rule. Examples of strong views are the *prefix* and the suffix described in Stalker [1]. Given multiple strong views on a training set, it is possible to infer different extraction rules that are then applied on the test set. When the extraction rules from different strong views differ, the system actively requires feedback from users. The authors exploit the possibility of using weak views to reduce the human intervention when rules generated from the strong views differ. The system starts considering a set of initial annotations over the training data and in a second moment it requires new annotates to refine the extraction rules.

Lixto [3]: The authors described a visual tool for generating wrappers. The expressiveness and the formalism of the wrapping program are based on Elog, an extension to the *datalog* language. Users can: visualize the tree representation of the HTML pages; select nodes to extract by using a mouse; visualize the inferred extraction patters; add filters and test the patterns on some example pages; change and delete existing extraction patters; specify a crawling program to collect the target pages and so on. The system has been used in real use cases, to collect data on flights, on news and for business intelligence. The tool is extremely powerful, thanks to the expressiveness of the extraction language and to many technical features. The tool was designed so that the end user does not have to posses the knowledge of the extraction language adopted for the wrapping. Even so, the tool was really complex and required many interactions to visually define a wrapping program, thus limiting the overall adoption of the tool.

2.1.2 Automatic Wrapper Generation

RoadRunner [4]: RoadRunner is an automatic wrapper inference system; it infers a regular expression from a set of templated HTML pages by reverse engineering the generation of the HTML pages. The regular expression is refined iteratively, and iteration after iteration it matches the regular expression (the extraction rule) with a new HTML page. The inference algorithm starts from a first HTML page, it sets the content of the

first page as the initial wrapper, it matches the wrapper to a new HTML page from the same template. On mismatch between the wrapper and the HTML page it adopts operators to solve them. There are different kind of operators for different mismatches: (i) on the text content of an HTML tag, this leads to the detection of a tag that denotes a value to extract, (ii) on a portion of the page that is not present in the wrapper (or the opposite), this leads to the detection of an optional portion of the HTML page, (iii) on a portion of the page that is repeated with a number of times different from the wrapper, this leads to the detection of lists. Based on these operators the system infers a wrapper that matches all the training pages, thus leading to the automatic extraction of the content embedded in the HTML template.

ExAlg [5]: ExAlg follows the same principles as RoadRunner. The approach is an unsupervised system that can separate the content from the template. The main difference between RoadRunner and ExAlg is that ExAlg adds the concept of equivalence class. Portions of the HTML pages are grouped together in as equivalence class when they occur exactly the same amount of times (e.g. the html, body tags). ExAlg finds equivalence classes in templated pages and retains only those classes that are frequently present LFEQs (Large Frequently occurring EQuivalence classes). The intuition is that HTML templates generated from scripts are characterized by portions of the page that are equivalently frequent, these portions identify templated tags. In a second step, the system extracts the content separating the values embedded inside the LFEQs. ExAlg takes into account the frequency of the tags inside the templated pages, while RoadRunner does not provide a frequency analysis on the HTML pages.

Zhai and Liu [6]: While RoadRunner and ExAlg consider as input multiple pages generated from the same script, Zhai and Liu proposed an approach that extracts content from a single HTML page with multiple instances embedded in the page. On the Web, there are many index pages that are characterized by the presence of a list of instances that are published into a single HTML page. The authors described a tree alignment technique, the intuition is that the region where instances are embedded in a list is more regular and it is generated by a iterative portion of the script. This motivates the possibility of adopting a partial tree alignment to find sub-portions of the HTML page that are similar. The algorithm aligns the records and extracts the text content embedded in the HTML tags. To discover regions inside the HTML page to run the alignment algorithm, the authors exploited some constraints related to the process of generating index pages (considering the DOM representation, the regions are all under the father node).

2.1.3 Scaling Wrapper Generation

Senellart et al [7] A possible technique to scale traditional supervised wrapper inference system is to automatize the annotation process, i.e. the process of generating the training data. The authors proposed an approach that exploits a Knowledge Base to automatically fill forms, to cluster the result pages and to extract the data from the result pages discarding erroneous pages. The Knowledge Base provides two kinds of information: the schema that describes the domain and a set of sample instances. The fields in the forms are matched with the schema and the instances are used to fill the form and cluster the result pages. The authors observe that with an erroneous input, forms are likely to lead to erroneous result pages. With this intuition the result pages generated from a set of sample instances are clustered together. Instances are used to annotate the result pages and fields are automatically extracted from the template. A domain expert is required to define the schema and to provide the sample instances.

Dalvi et al [8]: They proposed another approach that relied on automatic annotators. The authors described a system that annotates values in the page by matching values on a database or by adopting simple regular expressions (e.g. strings that matches "*Ltd" are likely to be related to names of companies). The annotation process is noisy but previous supervised wrapper inference systems are not noisy tolerant. To increase the robustness of the wrapper inference systems to mistakes during the annotation process, the authors exploited different properties: (i) with some domain knowledge it is possible to understand when a generalization is a good generalization and when it is not, (ii) correct annotations lead to good generalizations while wrong annotations lead to erroneous generalizations, (iii) a subset of the annotations generated by the noisy annotator is correct. The system takes into account all the annotations generated by the annotators, it considers different subsets of them and it generalizes to different extraction rules. Based on the expected precision and expected recall of the annotation process, the system ranks the extraction rules and selects the best rule.
The main limitations are: the Domain Knowledge is crafted by experts, i.e. requiring a cost to build the annotators; the annotation process often is characterized by a high standard deviation, i.e. the quality drops if the expected quality of the annotator is different from the real one.

DIADEM [9]: DIADEM or Domain-Centric Intelligent automated Data Extraction Methodology is a Data Extraction system that adopts Domain Knowledge to guide the extraction process. The Domain Knowledge is manually crafted by a Domain Expert and it describes how entities on that domain are described on the Web. The Domain

Knowledge provides the expected attributes or fields that are found on the Web, the type associated to the attributes and the presence of optional or mandatory attributes. Based on this Domain Knowledge and given a website to explore, DIADEM automatically explores the website by filling forms and navigating the links. To find detail pages that describe instances it matches attributes and records embedded in the HTML pages with the lists and gazetteers. To recognize attributes in the HTML pages, it exploits the types and the constraints defined by the Domain Knowledge. The system exploits the presence of a common template, it infers a wrapper over a small sample set and the wrapper is adopted to extract all the records published by the website. DIADEM adopts a specific extraction language OXPath [23], that lets the inference algorithm define extraction rules and navigation paths on dynamic pages. The solution easily scales over the number of websites but it is hard to adapt for multiple domains, in their evaluation the authors were able to extract content from thousands of websites for few domains (real estates from UK and US websites and used cars in US). The sources were manually added or collected from online lists.

Chuang et al [10]: The authors describe a context aware wrapping algorithm. The process starts from a set of existing wrappers and exploits the presence of peer sources on common domains. Traditional wrapper inference systems generate wrappers one site at a time, thus not exploiting the context where they are applied, i.e. the presence of multiple sites of the same domain. The domain knowledge is used to improve the accuracy of existing wrappers and to provide a help to match the same attributes in different sources. Given a set of sources where wrappers are already defined and given a new set of sources with no wrappers, the system generates some base extractors for the new sources and adopts a turbo decoding paradigm to sync the wrappers on the same fields. Data collected from each source are used to define a model; a model is represented as the set of fields that compose the model and the statistical model to generate the instances. Models defined by existing wrappers are considered as correct models while models from the base extractors on new sources are to be refined considering *decoders*. The authors adopt an EM approach, exploiting a mutual dependency: models are used to refine the *decoders* of the base extractors, refined *decoders* generate new models. The approach exploits the fact that sources publish data independently, thus multiple *decoders* tends to make different mistakes and an EM approach can be applied to converge the system to some majority, leading to an improvement of the results.

WEIR [11]: WEIR or Web-Extraction and Integration of Redundant adopts the redundancy of the same information published by multiple websites to infer wrappers and integrate the data, by aligning the same attributes from multiple websites. The

system exploits the fact that for many domains there are multiple websites that publish information about the same instances with the attributes, i.e. among multiple websites there is an overlap at schema level, same attributes (e.g. code, name, volume of a stock quotes), and at instance level, same entities (e.g. information about GOOG, APPL). The inference algorithm generates multiple extraction rules on all the sources and it selects the best rules for an attribute in each source by matching the extracted content from different sources. The intuition is that correct extraction rules of the same attribute (from different websites) are more similar to each other than erroneous rules. Exploiting this natural constraint, the authors were able to automatically extract and integrate data from multiple websites in different domains. The main limitation in this approach is that the quality of the output is still limited by the quality of the overlap (how much the attributes are similar, how many instances are in common).

2.2 Learning Model and Active Learning

Our inference algorithm finds its root in the statistical learning community. In machine learning, the number of labeled samples needed by a supervised learning algorithm to infer a *good* hypothesis is called *sample complexity* [24], and has been studied from several perspectives. For instance, the author of [14] discusses the problem of *exactly* inferring a *concept*, i.e., a set of elements, by means of different kinds of queries. The simplest form are *membership queries* (MQ), question of the type "*is this an element of the target concept?*", i.e. the system asks if an element is of a target concept and the answers are binary (yes/no). Another kind of queries, *equivalence queries* (EQ) let user provide a counterexample to the original question ("*no, because the element is of another concept c*"). An EQ can be expressed as multiple MQs, thus a learning model based on EQs can be translated to a learning model based on MQs.
If answers provided by users can be erroneous, a more complex setting has to be considered. In [13] the authors defined a statistical learning model, the PAC learning model, in which two *loss* functions are given in order to characterize the quality of the produced hypothesis for the learning model when mistakes are taken into account. One of the *loss* function models the probability that the learning algorithm learned a wrong concept, thus the goal of the learning process is to reduce this probability by asking labels/feedbacks in a sufficient number of examples so that it is lower than a given threshold, that results in a minimal number of queries. The second *loss* function models the possibility of not learning the exact concept, thus learning another concept that can be approximated to the right concept. In this work, the authors theoretically described the learning process and the boundaries required to reduce the *loss* functions.

Previous works described the learning models that are the base of our generative model, but they do not describe policies for the selection of the samples to request feedback. To reduce the number of queries required to learn the right hypothesis, a possible technique is to adopt Active Learning. Active Learning studies techniques to minimize the training required to infer a hypothesis [25], by selecting the best sequence of questions. An Oracle provides answers to the posed questions. Many researchers have proposed several variations of the learning paradigm to make it practically feasible in different applicative contexts: the learning approaches in which the inference algorithm is free to choose which sample to label next are usually defined *active*. These have recently gained interest, since, as clarified in [24], they might produce exponential improvements over the number of samples wrt traditional supervised approaches. Different query strategies are proposed in literature [25]: Uncertainty Sampling selects a query on the least certain sample; Query-By-Committee selects a query on the value where the committee (trained from the same training set) disagree the most; Expected Model Change selects the query on the value that provides the greatest change if we knew its label; Estimated Error Reduction finds the value that that minimizes the expected error rate if it is labeled with a certain label and added training. Most of these approaches are designed for complex classification tasks where different kinds of feedback are provided. In our case, for a binary classification task, many of these approaches collapse. For instance, selecting the most uncertain value is equivalent to selecting the value that reduces the most the uncertainty. Most of the works in Active Learning [24] consider perfect oracles; with the advent of the crowdsourcing phenomena a new trend related to noisy oracles has been considered [26, 27]. The biggest challenge is to reliably estimate the workers error rate by considering a golden standard or by considering multiple workers on the same task.

2.3 Crowdsourcing for Data Management

Recently there is an increasing interest in the crowdsourcing phenomenon. Crowdsourcing is an emerging area of research and is studied by different communities on different research areas. Involving the crowd is considered as a possible way to address many long standing challenges. In fact, the crowd can be integrated in automated tasks, thus for complex operations where algorithms or machines can not achieve sufficient results, humans can be involved. In data management, proposals have been made for crowdsourcing to support different stages of a data management lifecycle, such as data collection (e.g. [17, 30, 31]), integration (e.g. [32–34]), entity resolution (e.g. [35–37]) and querying (e.g. [38–40]).

A common aspect on all these works are the considered optimization coordinates: to minimize the cost required to complete the task, to increase the quality of the output

considering mistakes introduced by workers and to reduce the latency, the time from the submission of a task and its completion.
Some of the proposals address all three coordinates simultaneously [40], others consider just two of them leaving the third coordinate as an orthogonal concern. One of the main challenges for these crowdsourcing systems is the noise management; workers are prone to mistakes and among them spammers can voluntarily provide misleading feedbacks.
To address this challenge, workers' error rate should be estimated and spammers should be detected. Common techniques to manage the noise are: *golden standard* a training set with the correct answers for the posed questions are adopted to score workers and filter spammers; *redundancy* multiple workers are asked to answer the same tasks, thus it is possible to estimate the workers' error rate; *gamification* or *reputations* workers are somehow self motivated to complete the tasks in a correct way for a personal gain, e.g. a ranking in a game or some social reputations.
In [30] the crowd is modelled as an extension of existing database systems. Special annotations are added to table to identify special crowdsourced attributes. From these annotations forms and tasks are automatically created and submitted to a crowdsourcing platform. The crowd is also involved to solve queries on missing values.
Binary Query, Active Learning and Crowdsourcing are adopted by [32] to address schema matching. This work aims to reduce the uncertainty resulting from automated schema matching systems by actively posing query to workers enrolled from Amazon Mechanical Turk. The authors described primarily how to optimize the costs with the crowd without penalizing the quality of the output, but workers' error rate estimation and spammers identification are not considered.
In [30] the crowd is adopted to address the entity resolution problem. The system defines a complete pipeline for entity resolution based on the crowd, *hand-off crowdsourcing*. Blocking Rules are adopted to discard obviously wrong resolutions, Match is used to predict probable resolutions with an Active Learning approach, Accuracy Estimator computes the accuracy of the predictions and Difficult Pairs' Locator is used to resolve and identify errors in Match and to improve the matching rules. All these steps involve the crowd to improve the quality of the output.

2.4 Discovery and Crawling

Data Extraction systems face the problem of finding the input pages, i.e. the templated pages from the same website where it is possible to infer the wrapper. There are two broad approaches to get templated web pages from the web. The first is to start from a complete crawl of the web, and identify the HTML tables and lists that contain the structured data of interest by adopting a clustering technique [41]. The second is to

perform a focused crawl of the web, looking only for target templated pages. While the first approach has been investigated in the literature (see, e.g., [12, 41]), it has its limitations: very few groups have access to up-to-date web crawls, and initiating such crawls is extremely resource intensive. We focus our attention on the second approach. In fact collecting target templated pages is a challenging task, existing generic snapshots of the Web, e.g. Common Crawl [1], do not solve the problem. In fact, target pages are often deep inside the website, thus hard to be collected by a generic crawler, pages are not up-to-date and many small websites with relevant pages are not actually indexed.

In [12] the authors analyzed the distribution of the templated pages on several domains considering all the pages indexed by Yahoo!. The authors adopted search on a certain representation of the instances (e.g. key values, ISBN or restaurants' telephone number) to discover websites that publish information on a target domain. They adopted an automatic wrapper inference technique to extract all the representation of the instances in the discovered websites and iteratively discovered a comprehensive list of target websites related to that domain. For a single domain there are thousands of websites that publish relevant information. If we model the sites discovering process as a bipartite instance-sites graph, thus an edge represents the containment of an instance in that website. The authors found that for many domains the graph is highly connected and that the diameter is small, ranging from 6 to 8. The authors considered the extraction only on few "core" attributes, e.g. identifiers and so on, thus simplifying the extraction task. Many challenges related to focus crawling the target pages from the website and the availability of a huge index of the web were not addressed by the authors.

The previous observation on the connectivity of the Web, motivated many iterative search approaches that, querying an indexed copy of the Web (API from a search engine), they incrementally discovered new websites with a set expansion paradigm [42, 43]. Many issues related to this approach are not addressed.
A first issue is the so called "Semantic Drift", in fact from a small set of reliable websites, adopting search the set expansion paradigm can end-up on a website that publish the same representation used to query the search engine, but with a different semantic (searching for the name of an actor can lead to pages related to a musician with the same name).
A second more technical issue is related to the limitations imposed by the search engines on their APIs. In fact, often these interfaces are limited in number of overall calls per month, number of calls in a small interval of time, and number of results returned in each call. This issue makes the previous bipartite navigation graph, which is highly connected with a small diameter, really challenging to navigate for groups that do not have an open access to a comprehensive index of the entire Web.

[1] commoncrawl.org

Techniques to automatically crawl templated pages have been proposed [44–47].
In [44] the authors defined a combination of online learning and bandit-based approach to discover relevant websites and collect templated pages, balancing exploration and exploitation. The approach uses the semantic annotations on the Web to guide the crawler; semantic annotations provide the feedback required to recognize target pages. The authors evaluated their approach on the dump of Common Crawl. [44] is a generic crawler that can be adopted on different domains.
Another generic approach is to reverse engineer the navigation structure of the website and infer a site map structure of the website. In [46] the system recursively navigates from the target pages to the root of the website. Given a sample page, the crawler navigates all the links from the sample page, from the navigated pages it searches for *link collections* that leads to the original page. This kind of pages are commonly index pages, recursively adopting this strategy finds the index structure of the website that lead to pages similar to the sample page.
An alternative way to collect relevant pages is by considering some categories of pages. [45] describes a forum specific crawler. The approach exploits the fact that forums are organized following a common structure: an entry page of the forum, threads, topics, discussions with user's comments and responses and so on. Based on this structure, classifiers and heuristics are combined together to craft a focus crawler specialized for forums.
In [47] navigation paths are used to extract entities and relationship among the entities. The authors adopt an existing database to guide the crawler and the extraction process and exploit lists and parallel navigation paths to infer relationships among entities. The database, with its instances and the schema associated to the entities, is used to recognize the same entities in webpages: a list groups entities under a common semantic and a parallel navigation describes hierarchies and relationships between instances of different entities.

Chapter 3

Single Noisy Worker

Although many research efforts concentrated on the development of methods and tools to generate web wrappers, large scale data extraction is still a challenging issue. Early proposals to infer web wrappers for data intensive websites were based on supervised approaches. Wrappers were generated starting from a set of training data, typically provided as labeled values, i.e., annotated pages. To overcome the need of human intervention in the production of training data, unsupervised approaches have been investigated. They exploit the local regularities of script-generated web pages to infer a wrapper. Unsupervised approaches adopt sophisticated algorithms to generate the wrappers, and represent an attempt to "scale-up" the wrapper generation process. Unfortunately, although they eliminate the costs of training data, they have a limited applicability because of the low precision of the produced wrappers.

The recent advent of crowdsourcing platforms (such as, for example, Amazon Mechanical Turk) can open new opportunities for supervised approaches. These platforms provide support for managing and assigning mini-tasks to people. In the wrapper production process, crowdsourcing platforms can be used to produce massive training data for supervised wrapper inference systems. As they facilitate the involvement of a large number of persons to produce the training data, we may say that they represent a solution to "scale-out" the wrapper generation process. However, to obtain an efficient and effective process, two main issues need to be addressed. First, since mini-tasks are performed by non-expert people, they should be extremely simple. Second, since the costs of producing wrappers become proportional to the number of mini-tasks, the number of training data produced by the crowd to infer a wrapper should be minimized.

In this chapter we present ALF$_\eta$ a system that relies on crowdsourcing platforms to create accurate web wrappers. Our system adopts a supervised approach to infer wrappers with training data generated by means of a crowd computing platform. The mini-tasks

submitted to the platform consist of membership queries (MQ), which are the simplest form of queries, since they admit only a yes/no answer (e.g., "Observe this page: is the string 'Dean Martin' a correct value to extract?"). To address the costs issue, our system is able to select the queries that more quickly bring to infer an accurate wrapper, thus minimizing the number of mini-tasks assigned to the crowd platform. In this chapter we limit the discussion on dealing with a single worker at the time and leave for the next chapter the discussion about the schedule of tasks and the estimation of the workers' error rate.

The chapter is organized as follows: Section 3.1 presents an overview of ALF_η; Section 3.2 formalizes our setting; Section 3.3 describes the generation process of the candidate rules; Section 3.4 develops our probabilistic model to characterize the correctness of extraction rules; based on the model, Section 3.5 presents the active learning algorithm to infer extraction rules; Section 3.6 introduces the sampling algorithm; Section 3.7 discusses experiments with a set of sources from the Web; finally, Section 3.8 concludes the chapter.

3.1 Overview

We propose a logical framework based on original solutions for exploiting crowd platforms to infer wrappers around large web sources. Since we aim at demanding to a crowd platform the burden of generating labeled examples, our approach considers a cost that takes into account the number of membership queries submitted to a crowd platform.

Our framework includes a supervised active learning algorithm that aims at minimizing the number of membership queries to infer a wrapper: it selects a value and poses a membership query to obtain a confirmation about the correctness of the extracted value. By accurately choosing this query, the user interaction is minimized. Our experiments prove that our learning algorithm can infer high quality wrappers with a fraction of the queries required by a traditional approach.

Our algorithm infers the wrapper on a set of labeled values, and the quality model evaluates the wrapper on a larger set, ideally on the whole set of target pages. However, in many practical cases the evaluation on the whole set is unrealistic because of its size. To overcome this issue, our framework also includes an algorithm to compute a small set of representative pages: the extraction rules inferred and evaluated against our representative set also work on the larger set of target pages. Our experiments show that our algorithm is able to select a representative set several orders of magnitude smaller

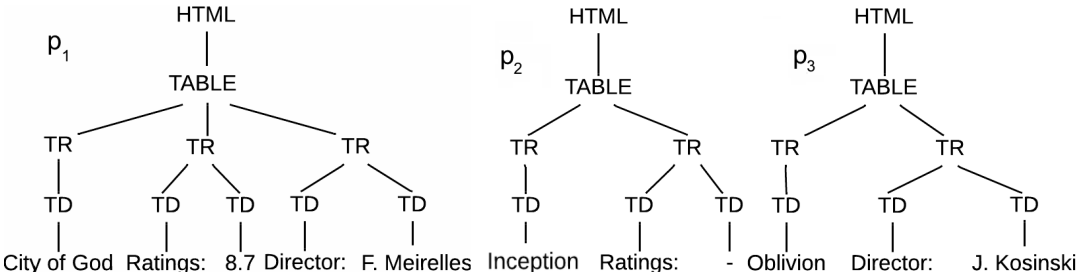


FIGURE 3.1: Running example with the DOM tree of three sample pages

pages / *rules*		U		
		p_1	p_2	p_3
$\mathcal{R}_{\mathsf{Title}}$	r_1	City of God	Inception	Oblivion
	r_2	City of God	Inception	*nil*
	r_3	City of God	*nil*	Oblivion

r_1 =/html/table/tr[1]/td
r_2 =//td[contains(.,"Rating:")]/../../tr[1]/td
r_3 =//td[contains(.,"Director:")]/../../tr[1]/td

FIGURE 3.2: Extraction rules and the extracted values for the running example on the attribute Title .

than the whole set of target pages, and that wrappers inferred from our representative sample outperforms (in term of precision and recall) wrappers generated from much larger randomly selected sets.

In summary in this chapter, we make the following contributions:

- a framework that exploits crowd platforms to infer wrappers around large web sources;
- a cost model that takes into account both the processing costs and the human intervention costs needed to feed the crowd platform;
- a probabilistic quality model for computing correctness of a wrapper over the whole set of pages even if it is inferred on a smaller set of pages chosen by a sampling algorithm;
- an active learning algorithm for generating high quality wrappers in a cost-effective manner;
- a sampling algorithm for selecting small yet representative sets of pages;

3.2 Preliminaries

Let $U = \{p_1, p_2 \dots p_n\}$ be a set of pages. Every page publishes several attributes of interest (e.g., in our running example, movies' Title, Director, etc.). For simplicity we assume that its values are either a textual leaf of the DOM tree representation of the pages, or a distinguished *nil* value. We write $v \in p$ to denote that v is a value of the page p, and p_v to denote the page in which the value v is located.

We refer to a generic *extraction rule* (or simply *rule*) r over the set of pages U as a concrete tool to build a vector of values indexed by the pages in U such that $r(p) \in p \cup \{nil_p\}$. Every rule extracts one vector of values from U denoted $r(U)$. Figure 3.2 shows the vectors extracted by the rules r_1, r_2, r_3. We denote by $R(U)$ the set of vectors obtained by applying a set of rules R over U, and blur the distinction between a rule and the vector it extracts from U. Note that $|R(U)| \leq |R|$, with the strict inequality holding whenever a vector is extracted by different rules.

We introduce the concept of *labeled sample value* (or simply *labeled value*) v^l where $v \in p_v$ is a value from a page p_v, and $l \in \{+, -\}$ is either a positive or a negative label. In the following v^+ and v^- denote a positively labeled value (or annotation) and a negative labeled value, respectively, i.e., the two possible answers to a *MQ*. We denote by R_A and v_A the set of rules and a value extracted by rules related to the attribute A, in the following we omit the notation when a single attribute is considered.

A rule r is *admissible* wrt a set of labelled values L (denoted $L(r)$) iff:

$$L(r) \;\Leftrightarrow\; \forall v^l \in L, \begin{array}{l} l = + \rightarrow r(p_v) = v \\ l = - \rightarrow r(p_v) \neq v \end{array}$$

that is, it is compliant with the labels in the set.

The concept can be trivially extended to a set of rules R. We denote by $R_L = \{r \in R : L(r)\}$ the subset of admissible rules in R wrt L, and by $\widehat{V}_L^R(U)$ all the values they extract from U: $\widehat{V}_L^R(U) = \{v : v = r(p), r \in R_L, p \in U\}$.

Example 3.1. *Let p_1, p_2 and p_3 be the pages in Figure 3.1 and let $U = \{p_1, p_2, p_3\}$. The attribute* Title *is extracted by the rule r_1: two positive annotations are $v_0^+ =$'City of God' in p_1 and $v_1^+ =$'Inception' in p_2; a negative labelled value in p_2 is $v_2^- = nil_{p_2}$. Observe that r_2 is admissible wrt $L = \{v_0^+, v_1^+, v_2^-\}$. Now consider another rule r_3, the rule is not admissible wrt L since $r_1(p_2) = nil_{p_2}$ which is the negatively labelled value v_2^-. Hence, $R_L = \{r_1\}$ and $\widehat{V}_L^R(U) = \{v_0, v_1, v_3\}$ where $v_3 =$'Oblivion' in p_3.*

We denote with V_A the set of all the values extracted by $\mathcal{R}_A$ in U for the attribute A, i.e. we do not consider the admissibility of the *rules* wrt L.

In the following, given a set of rules R, we will only consider special ordered sets of labeled values, called *training sequences*, which are formed by an initial set of positive annotations, and then by adding only new values which are still admissible with respect to those already seen. Intuitively, a training sequence lists the answers to the *MQ* posed to learn a extraction rule.

A Training Sequence *(t.s.) L wrt a set of rules R and a set of pages U is specified by a sequence of labeled values that defines a sequence of (observed) sets L^k with $L^{k+1} = L^k \cup \{v_k\} = \{v_0^+, \ldots, v_{a-1}^+, v_a \ldots, v_k\}$ such that: (i) it begins with sequence of $a \geq 1$ annotations $v_0^+, \ldots, v_{a-1}^+ \neq nil$ with positive labels, and (ii) $\forall k \geq a$, $v_k \in V_{L^k}^R(U) = \widehat{V}_{L^k}^R(U) \setminus L^k$.*

The constraint (i) on the first annotations of the sequence is useful to generate a finite set of admissible rules R_{L^a}, whereas the constraint (ii) on the remaining values entails that the *new* value v_k that forms L^{k+1} from L^k leads to smaller and smaller admissible sets: $R_{L^{k+1}} \subseteq R_{L^k}$. It is worth noting that $R_{L^{k+1}}$ plays the role of what the learning communities call the *version-space* [25], i.e. the set of hypotheses still plausible after having considered an input set of labeled values.

Example 3.2. *Consider again the above Example 3.1 and our running example in Figure 1.1. Then a possible t.s. is $L^2 = \{v_0^+, v_1^+\}$ and $R_{L^2} = \{r_1, r_2\}$. Possible candidate values are $V_{L^2}^R(U) = \widehat{V}_{L^2}^R(U) \setminus L^2 = \{\text{'Oblivion'}, nil_{p_2}, nil_{p_3}\}$. A new* MQ *can be formed by choosing a new value v_2 to query from the elements in $V_{L^2}^R(U)$. E.g., "* is 'Oblivion' a correct value? *".*

In the following we will uniformly refer to both L and one of its observed subsets L^k blurring the differences between the two concepts whenever the context clarifies which one is actually involved.

It can always be decided whether a rule extracting the desired vector exists. However, since it is not known in advance whether that rule was in the set of all candidate rules, the only certain way to be sure of its presence is by checking every single page [14].

3.3 Rules Generation

We propose a wrapper induction process that starts with only one annotated page, that is, one page where the value of the target attribute has been marked.The input

annotated page may be supplied either manually or automatically by looking up in the page a golden value from an available database.

From the input annotated page, we generate a space of hypothesis, i.e., a set of *candidate rules*, denoted $\mathcal{R}$, that extract the given initial annotation for the attribute A. Notice that distinct rules might be equivalent, i.e., they produce the same results on U. In the following, we assume that in $\mathcal{R}$ we save only one representative rule for every class of equivalent rules. We consider extraction rules defined by means of expressions belonging to a simple fragment of XPath. Namely, we use rules that specify paths that start from a pivot node and lead to the annotated value. We adopt several types of node as pivot: nodes that occur exactly once in every input pages, such as, the document root and nodes having an 'id' attribute; textual leaves that occur at most once in a large percentage of the input pages. The rationale is that it is unlikely that these nodes appear once in almost all pages by chance, but rather they are likely to be part of the underlying template [5].

Although the candidate rules are correct for the page containing the initial annotation, they might not work correctly for other pages in U.

Example 3.3. *Suppose that we are interested to generate a wrapper that extracts the* Title *from the fictional set of movie pages* $U = \{p_1, p_2, p_3\}$ *whose DOM trees are sketched in Figure 1.1. Assume that the initial annotation '*City of God*' is supplied on the sample page* p_1*. Figure 3.2 shows the set* $\mathcal{R}_{\mathsf{Title}} = \{r_1, r_2, r_3\}$ *of candidate rules generated from this initial annotation. Rule* r_1 *is pivoted in the document root, rule* r_2 *and* r_3 *are pivoted in the template nodes '*Rating:*' and '*Director:*', respectively, which occur once in two out of three input pages. Note that* r_1 *is the only correct rule, as it extracts the* Title *from all input pages, whereas* r_2 *does not work on* p_3*, and* r_3 *does not work on* p_2*.*

In order to select the correct rule, our inference process evaluates the candidate rules by posing a sequence of queries to a human *worker* recruited from a crowdsourcing platform. The worker is shown a page p and is asked whether a given value $v_A = r(p), r \in \mathcal{R}$, extracted from the candidate rule r is the correct one for the target attribute A in the page p. The binary answer l, with $l \in \{-, +\}$, provided by the worker adorns the queried value v_A with either a positive or a negative label, producing a *labeled value*, denoted by v_A^l.

Example 3.4. *Continuing the previous example, the inference process may build a query with the value* $r_1(p_2) = r_2(p_2) =$ Inception*: the worker is shown page* p_2*, and is asked to confirm whether* Inception *is the* Title *of the movie in that page. A confirmation corresponds to produce the labeled value* $\mathsf{Inception}^+$*.*

The labeled value produced by the worker is appended into a *training sequence* (t.s.) for A, denoted L. A Bayesian model is used to compute the probability of correctness of the candidate rules, given the labeled value just acquired, and the t.s. collected so far. As new queries are posed, the t.s. is expanded with the returned labeled values, and the probabilities are updated until a termination condition is satisfied.

3.4 Bayesian Model

In this section, we develop our Bayesian model for estimating the probability $P(r|v_A^l, L)$ of each candidate rule $r \in \mathcal{R}_A$ of being a correct extraction rule of A for the whole set of input pages U, given a new labeled value v_A^l and the t.s. L acquired so far. Our model considers noisy workers making independent and random mistakes, that is, providing erroneous labels with a certain error rate η.

The probability of correctness of an extraction rule is computed whenever the worker provides a new labelled value v_A^l, which will expand the current t.s. L. The posterior probability $P(r|v_A^l, L)$ can be obtained starting from the probability $P(r|L)$ by means of a Bayesian update.

The whole process is triggered by a prior p.d.f. $\mathcal{P}(r)$ over the candidate extraction rules $r \in \mathcal{R}$ extracting the initial annotated value of A from the input pages U. We assume that a correct rule exists in $\mathcal{R}$ and we use a simple uniform prior: $\mathcal{P}(r) = \frac{1}{|\mathcal{R}(U)|}$.

By applying Bayes' theorem:

$$P(r|v_A^l, L) = \frac{P(v_A^l|r, L)P(r|L)}{P(v_A^l|L)} \tag{3.1}$$

where $P(v_A^l|r, L)$ is the likelihood of acquiring the labeled value v_A^l conditioned to the correctness of r, once a t.s. L has been observed, and $P(v_A^l|L)$ is a *normalization factor* that can be expressed as:

$$\sum_{r_i \in \mathcal{R}} P(v_A^l|r_i, L)P(r_i|L) \tag{3.2}$$

to sum up all the probabilities to one.

The p.d.f. $P(v_A^l|r, L)$ can be obtained by introducing a probabilistic *generative model* to abstract the actual process leading to the generation of every possible t.s. in presence of a correct rule r. Notice that the labeled values forming the t.s. L will be labeled as either positive or negative based on the values of A, assumed correctly extracted by r, but these values are not known in advance.

We adopt a simple generative model of the labelling process that randomly chooses, without replacement, the next queried value among the set of candidate values, i.e., the next value is chosen from the set $V_A \setminus L$.

To take into account the errors of workers, we assume that a worker makes independent random mistakes, as for example in the *Classification Noise Process* [48], with an expected error rate η. It follows:

$$P(v_A^l|r,L) = \begin{cases} \frac{1-\eta}{|V_A \setminus L|} & \text{, iff } v_k \in V_A^l(r) \\ \frac{\eta}{|V_A \setminus L|} & \text{, iff } v_k \in V_A^{-l}(r) \\ 0 & \text{, otherwise} \end{cases} \tag{3.3}$$

where, given a correct rule r, $V_A^l(r)$ denotes the set of values that can form new values labeled l after having observed the t.s. L; $-l$ is the opposite label of l.

3.5 Active Learning for Wrapper Generation

In the previous Section 3.4, we defined the probabilistic model that computes the probability of the candidate rules. In the remaining of this section we discuss a simple Active Learning algorithm to select the best extraction rule and we discuss the subprograms CHOOSEQUESTION(), which implements the query selection strategy by choosing the best value to be queried, and HALT(), which implements the stopping condition for the algorithm.

Listing 1 ALF$_\eta$: Active Learning Algorithm for Wrapper Inference

Input: a set of pages U
Input: the set of candidate rules $\mathcal{R}$

Parameter η: worker error rate
Parameter λ_r: target probability of correctness
Parameter λ_{MQ}: maximum budget

Output: a p.d.f. describing the probability of correctness of the rules in $\mathcal{R}$

1: **let** $L = \varnothing$;
2: **while** (**not** HALT$(L, \lambda_r, \lambda_{MQ})$) **do**
3: $v_A \leftarrow$ CHOOSEQUESTION(L);
4: $l \leftarrow$ GETANSWER(w, v_A);
5: compute $P(r|v_A, L)$, $\forall r \in \mathcal{R}$ with Eq. 3.1 and Eq. 3.3 ;
6: $L \leftarrow L \cup \{v_A^l\}$;
7: **end while**
8: **return** $P(r|L)$, $\forall r \in \mathcal{R}$;

The above approach is detailed in the ALF$_\eta$ algorithm, whose pseudo-code is illustrated in Listing 1: it takes as input a set of pages U and a set of candidate rules $\mathcal{R}$ for the

attribute A (computed from an initial annotated page), it poses queries to the worker and collects its answers into a t.s., and it returns a p.d.f. describing the probability of correctness of the rules in $\mathcal{R}$.

ALF$_\eta$ progressively builds a t.s. L by posing queries to a worker. In every iteration (lines 2–7), the worker is asked to label a new value v_A (lines 3–4). Then, for all the rules $r \in \mathcal{R}$, ALF$_\eta$ computes the probability distribution function $P(r|v_A, L)$, that is, the probability that each $r \in \mathcal{R}$ is correct, given the last labeled value and the t.s. L acquired so far (line 5). Finally, the t.s. L is expanded by adding v_A^l (line 6).

We can customize the algorithm by modifying the function to select the query (CHOOSEQUESTION()) and by setting the termination condition (HALT()).

3.5.1 Asking the Right Questions

rules \ *pages*		p_1	p_2	p_3	
		U			
$\mathcal{R}_{\text{Title}}$	r_1	City of God	Inception	Oblivion	$P(r_1\|L) = 0.4$
	r_2	City of God	Inception	nil_{p_3}	$P(r_2\|L) = 0.2$
	r_3	City of God	nil_{p_2}	Ratings:	$P(r_3\|L) = 0.2$
	r_4	City of God	-	Director:	$P(r_4\|L) = 0.2$

$V_A \setminus L = \{\text{“Inception”}, nil_{p_2}, \text{“-”}, \text{“Oblivion”}, nil_{p_3}, \text{“Ratings:”}, \text{“Director:”}\}$

TABLE 3.1: Running example for asking the right questions

The CHOOSEQUESTION() procedure chooses the next membership query: it decides the next value to be labeled. We propose three alternative strategies: ENTROPY, GREEDY, and LUCKY, plus a baseline algorithm RANDOM. The value is picked up from the *set of candidate values* for attribute A, denoted V_A, which contains all the values extracted from pages in U by the candidate rules in $\mathcal{R}$.

RANDOM: It chooses a random value from V_A:

$$\text{CHOOSEQUESTION}(R,L)\ \{\ \textbf{return}\ \ \text{a random } v \in V_A \setminus L;\ \}$$

and it serves as a baseline against other strategies.

ENTROPY: $V_A = \{r(p), r \in \mathcal{R}, p \in U\}$. ENTROPY strategy consists in choosing the value on which rules most disagree, appropriately weighted according to their probability. This is equivalent to compute the *vote entropy* [25] for each $v \in V_A \setminus L$:

$$H(v) = -[P(v^+|L) \log P(v^+|L) + P(v^-|L) \log P(v^-|L)] \tag{3.4}$$

Equation 3.4 computes the *vote entropy*. i.e. the uncertainty of a given value. Equations 3.5 and 3.6 are the probabilities that v is respectively either a value to extract or an incorrect value: $\mathcal{R}^v$ is the set composed of rules in $\mathcal{R}$ that extract v.

$$P(v^+|L) = \sum_{r \in \mathcal{R}^v} P(r|L) \tag{3.5}$$

$$P(v^-|L) = \sum_{r \in \mathcal{R} \setminus \mathcal{R}^v} P(r|L) \tag{3.6}$$

Intuitively, the entropy measures the uncertainty of a value and querying the value with the highest entropy removes the most uncertain value:

$$\text{CHOOSEQUESTION}(L)\ \{\ \textbf{return}\ \ \text{argmax}_{v \in V_A}\, H(v);\ \}$$

Example 3.5. *Reconsider the running example in Table 3.1, and the t.s.* $L^1 = \{\mathsf{City\ of\ God}^+\}$. $P(v_1^+|L^1)$ *and* $P(v_1^-|L^1)$ *can be computed as follow:*

v_1	$P(v_1^+\|L^1)$	$P(v_1^-\|L^1)$
Inception	$0.4 + 0.2$	$0.2 + 0.2$
nil_{p_2}	0.2	$0.4 + 0.2 + 0.2$
Oblivion	0.2	$0.4 + 0.2 + 0.2$
Director:	0.2	$0.4 + 0.2 + 0.2$
...	...	...

From $P(v_1^l|L^1)$ *by using Eq. 3.4 the entropy* $H(v)$ *can be obtained as follows:*

v_1	$H(v_1)$
Inception	$-0.6 \cdot log(0.6) - 0.4 \cdot log(0.4) = 0.292$
nil_{p_2}	$-0.2 \cdot log(0.2) - 0.8 \cdot log(0.8) = 0.216$
Oblivion	$-0.2 \cdot log(0.2) - 0.8 \cdot log(0.8) = 0.216$
Director:	$-0.2 \cdot log(0.2) - 0.8 \cdot log(0.8) = 0.216$
...	...

Hence, ENTROPY *chooses* $v_1 =$ Inception *as the next value to query to get* $L^2 = L^1 \cup \{v_1\}$.

GREEDY: The construction of the whole version-space is inefficient, since it requires to enumerate all possible t.s.. However, the version-space can be exploited to find the quickest t.s. confirming that a given rule is a solution. Let us call such a kind of sequences *confirming* t.s.: they aim more at deciding as quickly as possible that a given rule is a solution, rather than at finding which is the solution.

In every search step, GREEDY "elects" the most likely rule to play the role of the solution, and then it *greedily* builds a confirming t.s. wrt that conjecture. If, after a few labeled

values, that rule is confuted and removed from the version-space, the whole process is repeated by formulating another conjecture around the most likely rule in the remaining version-space.

In this setting, the query is selected by greedily taking the value extracted by the supposedly "correct" rule from the page on which most other rules behaves differently: if that value is labeled positive as expected, the largest number of rules is removed from the version-space.

$$\begin{aligned} &\textsc{chooseQuestion}(R,L) \;\; \{ \textbf{return } r^*(p^*) \; \} \\ &\text{where:} \quad r^* = \; \text{argmax}_{r \in R_L(U)} P(r|L); \\ &\qquad\quad p^* = \; \text{argmax}_{p \in U} |\{r(p) : r(p) \neq r^*(p)\}|. \end{aligned}$$

As the cost of this approach depends on the size of the version-space, it can be relevant in the early stages of the searching. The next variant delays its construction until the best rule emerges as significantly more likely than other candidates.

Example 3.6. *Reconsider the running example in Table 3.1, and the t.s.* $L^1 = \{\mathsf{City\ of\ God}^+\}$.

The most likely rule is r_1 *because* $P(r_1|L) = 0.4$, p^* *is selected considering the following scores:*

p	*score*
p_1	0
p_2	2
p_3	3

Hence $p^* = p_3$ *and* Greedy *chooses* $v_1 = r^*(p^*) = \mathsf{Oblivion}$ *as the next value to query to get* $L^2 = L^1 \cup \{v_1\}$.

Lucky: It is a hybrid of the former two approaches, and it works in two phases: first, it accumulates enough evidence of the correctness of a rule by using Entropy; then, it switches to Greedy modality to confirm it. The switch is triggered by a fixed threshold λ_{r^*} on the probability of the most likely rule r^*.

This approach can be seen as a generalization of Greedy: at the beginning it waits to observe enough evidence before allocating all its trust on the most likely rule.

3.5.2 Termination Condition

The most appropriate termination policy might well depend both on budget constraints and on the quality targets. We propose a simple implementation of halt() that takes into account both aspects:

$$\text{HALT}(L, \lambda_r, \lambda_{MQ}) \ \{ \ \textbf{return} \ (\max_{r \in \mathcal{R}} P(r|L) > \lambda_r) \ \textbf{or} \ (|L| > \lambda_{MQ}); \ \}$$

According to this policy, we stop when the probability of the best rule overcomes a threshold λ_r or just run out of a "budget" of λ_{MQ} membership queries allocated for learning the rule from this worker.

3.6 Sampling

So far we considered feasible the application of our algorithm ALF_η to the whole set of input pages. However, in many practical cases this assumption is unrealistic because of the number of pages (e.g., consider www.imdb.com, which provides more than $6 \cdot 10^6$ pages about actors). Finding a sampling set that is "cheaper" to work on, and yet it represents a larger population, is a traditional statistic problem. In this section we contextualize this issue in our setting and move to the related problem of *sampling* the input pages into a much smaller set of sample pages. The extraction rules can be evaluated on the sample set much more efficiently than on the whole set of pages; at the same time, a *representative* sample set must preserve the power of differentiating the rules by showing all their differences. However, the sample pages need to be carefully selected to be representative while, at the same time, minimizing their number.

These aspects are often neglected in the literature. Typically, sample pages are selected randomly, or they are collected following straightforward crawling strategies. While random samples could end up not representing the whole set of pages, crawling strategies can lead to the composition of biased samples. As an example, www.imdb.com exposes its content mainly in the form of *top-lists*, such as top-list movies, top-list actors and so on. A crawler following the links in these lists will inherently collect biased samples concentrated around "famous" instances.

We formulate the problem of finding a set $I \subset U$ such that $|I| \ll |U|$ yet I is *representative* (with respect to a given class of extraction rules R) of all the pages in U. The representativeness of a set of pages $I \subset U$ wrt a set of rules R can be formalized by introducing the *disagreement set* of two extraction rules.

Given a set of pages P, and a set of rules R, the disagreement set, denoted as $D^P(r_i, r_j)$, between two rules $r_i, r_j \in R$, is the set of pages in P making observable their differences: $D^P(r_i, r_j) = \{p \in P : r_i(p) \neq r_j(p)\}$, i.e., the subset of pages in P on which r_i and r_j extract different values. Two rules r_i, r_j extract from P the same vector of values, and hence are indistinguishable for our purposes, if and only if $D^P(r_i, r_j) = \varnothing$.

We say that a subset $I \subseteq U$ is *representative* of U wrt a set of rules R if and only if:

$$\forall r_i, r_j \in R, [D^I(r_i, r_j) = \varnothing \iff D^U(r_i, r_j) = \varnothing].$$

In other terms, I is representative of U wrt to R if all the differences amongst the rules in R are also observable on I.

Example 3.7. *Consider again our running example in Figure 1.1 and suppose that* $I = \{p_1, p_2\}$*, while* $U = \{p_1, p_2, p_3\}$*. I does not represent U since* $D^U(r_1, r_2) = \{p_3\}$ *whereas* $D^I(r_1, r_2) = \varnothing$*.*

Given the set of input pages U, and the class of rules R, there exist many representative subsets, including U itself. As discussed above, our goal is to find a small sample set. Finding the smallest one is an instance of the well-known Set Cover problem: a page differentiates the set of rules that extract distinct values from it.[1] Set covering is an NP-complete problem but actually we do not need to compute the optimal sample set: it suffices to estimate it by considering a small but not necessary minimal set of pages.

Listing 2 proposes PageSampler, a greedy sampling algorithm to extract a representative set of pages I wrt a class of rules R from a large set of input pages U in $O(|U| \cdot |R_{L^a}|)$ time and $O(|R_{L^a}|)$ space.

Listing 2 PageSampler: A greedy sampling strategy

Input: a set of pages U;
Input: a class of rules R;
Input: a set of initial annotations L^a;

Output: a set $I \subseteq U$ that is *representative* of U; wrt R

```
1: let  I = ∅;
2: let  n = 0;
3: for p ∈ U do
4:    if (|R_{L^a}(I ∪ {p})| > n) then
5:       I ← I ∪ {p};
6:       n ← |R_{L^a}(I)|;
7:    end if
8: end for
9: return  I;
```

PageSampler processes the whole set of pages U (lines 3-8). It maintains a set of pages I, initially empty, that is representative wrt the subset of pages already processed. It selects as representative only those pages that increase the number of different vectors extracted by the set of admissible rules R_{L^a} (line 4). The pages selected according to this

[1] The problem reduces to finding the smallest set of pages such that the union of the sets of rules differentiated from them equals the set of rules differentiated directly by U.

criterion make observable *new* differences between at least two rules that were otherwise indistinguishable in the subset of pages processed until the previous iteration.

Example 3.8. *Consider the running example and suppose that* PageSampler *has already processed* p_1 *and* p_2*, producing* $I = \{p_1, p_2\}$*. Let* $R_{L^1} = \{r_1, r_2, r_3\}$ *be the set of admissible rules wrt to* $L^1 = v_0^+ = \{\mathsf{City\ of\ God}\}$*. The pages in* I *do not differentiate* r_1 *from* r_2*:* $r_1(I) = r_2(I)$*. However, when processing the next page* p_3*,* PageSampler *detects the different behaviour of* r_2 *wrt other rules:* $r_2(p_3) \neq r_1(p_3)$*, and then adds it to* I*.*

To clarify how PageSampler is related to the disagreement sets, consider that if $|R_{L^a}(I \cup \{p\})| > |R_{L^a}(I)|$ it follows that there exist at least two rules $r_i, r_j \in R_{L^a}$ such that $r_i(p) \neq r_j(p)$ and $D^{I \cup \{p\}}(r_i, r_j) \setminus D^I(r_i, r_j) = \{p\}$. Conversely, if $|R_{L^a}(I \cup \{p\})| = |R_{L^a}(I)|$ then it follows that $D^{I \cup \{p\}}(r_i, r_j) \setminus D^I(r_i, r_j) = \varnothing, \forall r_i, r_j \in R_{L^a}$. Therefore, PageSampler maintains the representativeness of I for the subset of U already processed by adding a page p to I if and only if p changes the disagreement sets of the rules.

3.7 Experiments

We have developed a working prototype that has been used to conduct experiments for evaluating the proposed approach [16]. The prototype takes as input a collection of pages containing data of interest. The attributes to be extracted are specified by annotating their value over a single page. Based on the input annotations, the system produces the initial sets of candidate rules, then it generates multiple tasks, and submits them to a crowdsourcing platform. The workers recruited on the crowdsourcing platform are redirected to an interactive web application. Each worker is asked to accomplish a task, consisting of a set of membership queries actively chosen by ALF$_\eta$ and posed to the worker through the web application. In particular, the application shows a page and asks the worker whether a proposed string occurrence, extracted by a candidate extraction rule, represents a correct value for the target attribute. Presenting HTML pages downloaded from an external server into a web application is not trivial: client side scripts and dependencies on remote resources (e.g., images) could prevent the pages to be rendered outside the server originally publishing them. To overcome these issues, the web application shows pre-computed images of the pages, generated during the downloading by our application. When a worker has completed the task, the web application returns a code that the worker has to insert into the crowdsourcing platform to prove the task fulfillment.

wesite	domain	#pages	$\lvert I_C \rvert$
www.imdb.com	Actor	$5 \cdot 10^5$	30
www.imdb.com	Movie	$5 \cdot 10^5$	42
www.allmusic.com	Band	$5 \cdot 10^5$	36
www.allmusic.com	Album	$5 \cdot 10^5$	29
www.nasdaq.com	Stock quote	$7 \cdot 10^3$	15

TABLE 3.2: Dataset for Sampling and the average representative sample set

With our prototype we conducted both experiments with real workers engaged by CrowdFlower (a popular meta-platform that offers services to recruit workers on AMT) and experiments with synthetic workers whose behavior was simulated following the *CNP* probabilistic model [48], i.e., they make random and independent errors with a fixed error rate η.

In this section we describe the experiments conducted to evaluate our approach. Section 3.7.1 presents our dataset and the evaluation metrics, we describe two dataset, one to evaluate the sampling algorithm and one to evaluate the ALF_η. Section 3.7.2 presents the results of the learning algorithm ALF considering perfect workers with $\eta = 0$. We compare the query selection policies: selecting the right query to pose drastically reduces the costs wrt the RANDOM baseline. Section 3.7.3 illustrates the results of experiments to evaluate the effectiveness of the sampling strategy implemented by the PAGESAMPLER algorithm. Our experimental results show that a few dozens of pages selected by PAGESAMPLER are sufficient to represent large collections of 10^5 pages from real-life websites. In Section 3.7.4, we present an experiment conducted with a population of real workers recruited on the crowdsourcing market. The goal of this experiment was to study how real workers behave with our particular tasks composed of a sequence of membership queries over pages from data-intensive websites. In Section 3.7.5, we present experiments for evaluating ALF_η with a single noisy worker on tasks related to a single attribute. We show the impact of workers error rate on ALF_η, the algorithm driving the interaction with the worker: the results motivate the introduction of our technique to estimate the workers error rate by submitting redundant tasks.

3.7.1 Datasets

We considered two distinct datasets to evaluate the learning algorithm ALF_η.

The first dataset has been obtained by downloading pages from large websites related to specific domain entities, as shown in Table 3.2. We wrote ad-hoc crawling programs, and let them collect around $5 \cdot 10^5$ pages for each entity from www.imdb.com and www.allmusic.com, and all the available pages about stock quotes from www.nasdaq.com

(around $7 \cdot 10^3$). For each entity we selected about 10 attributes, for a total of 40 attributes.

website	domain	#pages
imdb.com	Movie	10,000
imdb.com	Actor	10,000
allmusic.com	Album	10,000
allmusic.com	Band	10,000
nasdaq.com	Stock	6,461
allgames.com	Game	10,000
allmovies.com	Movie	10,000
espnfc.com	Player	10,000
espnfc.com	Teams	10,000
espon.go.com	Player	10,000
goodreads.com	Author	10,000
goodreads.com	Books	10,000
picclick.com	Monitor	100
picclick.com	Camera	100
alibaba.com	Monitor	7,185
alibaba.com	Camera	4,957
alibaba.com	Headphone	2,096
alibaba.com	Notebook	4,850
alibaba.com	Tv	4,734
dealtime.com	Monitor	358
dealtime.com	Camera	798
dealtime.com	Headphone	745
dealtime.com	Notebook	272
dealtime.com	Tv	213
colnect.com	Stamp	10,000

TABLE 3.3: Dataset for the evaluations.

The second dataset (Table 3.3) consists of 25 collections of pages from 12 websites and 16 different domains as detailed. For every website we downloaded a number of pages ranging from hundreds to 10 thousands. We considered the test set of 10,000 pages from each collection and around 4 attributes, for a total of 110 attributes. We run the PAGE-SAMPLER algorithm over these sample sets of pages to derive a representative sample for every domain (the sizes of the input sets, #pages, and of representative samples, $|I_C|$).

The first dataset will be used to evaluate the sampling algorithm, while the second dataset will be used to evaluate our wrapper inference algorithm.

We manually crafted a golden XPath rule for every attribute to extract its values. The (non-null) values extracted by the golden rules over the whole sets of pages were then used to compute and evaluate the precision and recall of the best rule inferred by our learning algorithm ALF_η. For each rule r generated by our algorithm wrt a

Strategy	P	R	F	average MQ	MQ
RANDOM	0.99	0.97	0.98	9.43	1037
GREEDY	1.00	0.99	0.99	4.33	476
LUCKY	1.00	0.99	0.99	4.14	455
ENTROPY	1.00	0.99	0.99	4.17	459

TABLE 3.4: Total number of *MQ* for Dataset 2 and average quality of the output

golden rule r_g, we used the standard metrics of precision (P), and recall (R), as follows: $P = \frac{|r_g(U) \cap r(U)|}{|r(U)|}$; $R = \frac{|r_g(U) \cap r(U)|}{|r_g(U)|}$.

3.7.2 Learning with ALF

Over the set of attributes in Table 3.3 we run the ALF_η algorithm to infer the extraction rules on the target attributes. In this experiment we set the probability threshold that governs the halt condition to 0.9 and λ_{MQ} to unlimited, and consider perfect workers with $\eta = 0$, i.e. we do not consider the budget limitation and we deal with ideal workers.

We were mainly interested to evaluate the impact of the different strategies to choose the next membership query. Table 3.4 summarizes the results of the experiment. We report the number of membership queries ($\#MQ$) for all the CHOOSEQUESTION() strategies. The most efficient strategies are ENTROPY and LUCKY, which significantly outperforms the baseline, represented by RANDOM. Results obtained with GREEDY are comparable with ENTROPY with an average waste of 0.16 *MQ*.

Another experiment aimed at considering the behavior of ALF_η by using different CHOOSEQUESTION() strategies, wrt the size of the hypothesis space, which in our context corresponds to the number of admissible vectors after the initial annotations, i.e., $|R_{L^a}(I)|$. Intuitively, the size of the hypothesis space is a measure of the cost that any learning algorithm needs to pay to infer a rule.[2]

The plots in Figure 3.3 show the average number of membership queries vs size of the hypothesis space. Note that when $|R_{L^a}(I)|$ is low, the differences in terms of $\#MQ$ are not apparent.

On the contrary, when $|R_{L^a}(I)| \gg 5$, RANDOM performs worse than other strategies. ENTROPY and LUCKY outperform the other approaches and, as expected from an active learning algorithm [25], $\#MQ$ follows a logarithmic trend with respect to the size of the hypothesis space.

[2]This is strictly related to the *sample complexity* commonly used by the machine learning community, as the amount of training data to learn a concept [24].

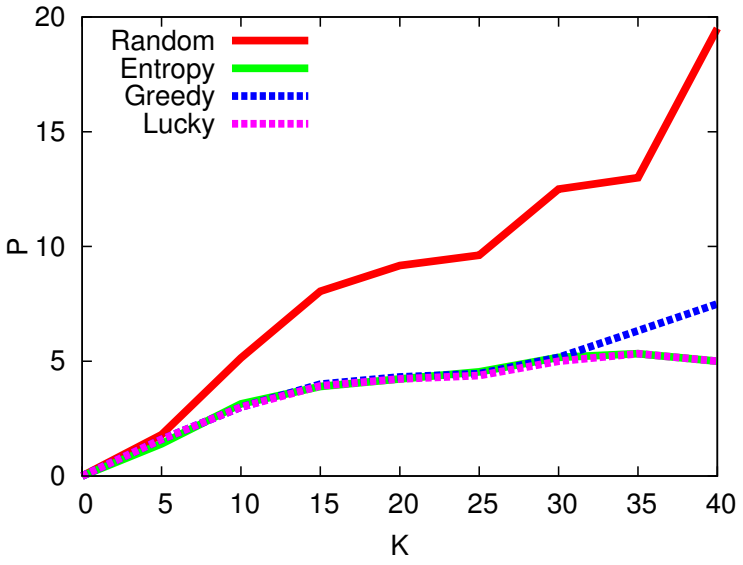


FIGURE 3.3: $\#MQ$ vs size of the hypothesis space

It is interesting to observe that GREEDY exhibits very good performances with smaller hypothesis space, while the number of MQ increases with larger hypothesis space.

We observe that between ENTROPY and LUCKY the performances are comparable, thus the following we will consider ENTROPY as our query selection strategy.

3.7.3 Sampling with PAGESAMPLER

We now discuss the experiments to evaluate the sampling algorithm PAGESAMPLER. For this evaluation we used the pages of the first dataset (Table 3.2). We collected three sample sets I according to different strategies, as follows:

- I_B represents a "biased sample": many large websites propose navigation paths to facilitate the browsing towards lists of relevant objects (e.g. famous actors, top-stocks, etc.). In our experiments, for each entity we downloaded the pages from the first list proposed by the sites. Therefore the size $|I_B|$ corresponds to the dimension of the proposed list.
- I_R is a set of pages randomly selected from the whole set U of pages with $|I_R|$ equals $|I_B|$.
- I_C is the representative sample set as computed by our sampling algorithm starting from the pages collected in our data set. $|I_C|$ is determined by the algorithm.

The first strategy does not pick up pages from the whole set of input pages U, while the second one chooses the sample pages in an uninformed way. These sampling strategies

Domain	Sampling	$\|I\|$	P	R
Movies	Crawler I_B	250	0.98	0.71
	Random I_R	250	0.99	0.99
	Representative I_C	42	1.00	1.00
Actors	Crawler I_B	250	1.00	1.00
	Random I_R	250	1.00	0.96
	Representative I_C	30	1.00	1.00
Stocks	Crawler I_B	86	1.00	0.98
	Random I_R	86	1.00	0.99
	Representative I_C	15	1.00	1.00
Albums	Crawler I_B	258	1.00	0.99
	Random I_R	258	1.00	1.00
	Representative I_C	29	1.00	1.00
Bands	Crawler I_B	289	1.00	0.68
	Random I_R	289	1.00	1.00
	Representative I_C	36	1.00	1.00

TABLE 3.5: Precision and recall with different sampling strategies

are used by many wrapper inference approaches more focused on the inference phase rather than on the sampling.

To evaluate the role of the three sampling strategies, Table 3.5 reports the average precision and recall computed over the attributes of all the entities of each domain. We inferred the extraction rules by running ALF$_\eta$ (without SRM) on the three samples obtained. We obtained perfect rules when the inference was performed on the representative sample I_C; conversely, both the random set I_R and the biased set I_B loose precision and recall for a majority of cases, with a significant lost of recall with I_B for bands and movies.

Table 3.5 also reports the size of the samples: it is worth observing that the representative sample set I_C is always much smaller than the random sample set I_R. This is an important point as it affects the running times of the learning algorithm, which performs better when working on small samples. Figure 3.4 illustrates this issue: the graphic plots the average wrapper learning times (in logarithmic scale) vs the size of the random sample $|I_R|$ (number of pages). The curve associated to I_R describes the learning times to compute the wrapper using a random sample of increasing size. As an example, for a random sample of 50 pages, it runs in about 15 secs; for a random sample of 450 pages, it runs in 100 secs. The curve associated to the representative sample I_C reports the learning times to infer the wrapper over a representative sample I_C whose pages have been selected from a random sample I_R with that number of pages. For example, from a random sample of $|I_R| = 450$ pages, PAGESAMPLER selected a representative sample composed of $|I_C| = 25$ pages, and on this sample ALF$_\eta$ inferred the wrapper in about 7 secs. Computing the representative sample has its own costs. However, as we can observe from the curves on Figure 3.4, even counting these costs the overall

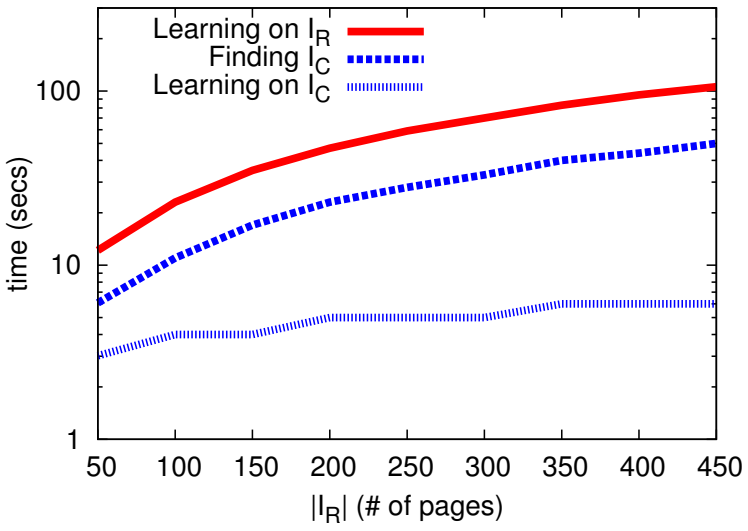


FIGURE 3.4: Wrapper learning times vs sample size

computation cost (sampling I_R to compute I_C + learning on I_C) is lower than the time required by learning without sampling (learning on I_R).

3.7.4 Modeling Real Workers

We report on a set of experiments that we conducted on a population of real workers engaged from CrowdFlower. The goal of the experiment was to obtain statistics about real workers behavior on our kind of tasks. Namely, we measures their error rates as they work through our web application. The goal of this session of experiments is to gain insights for setting realistic configurations of the synthetic workers used in other experiments.

We posted on CrowdFlower 485 tasks to generate with ALF$_\eta$ the extraction rules for 125 attributes, randomly selected from our datasets. Each task was paid 10¢, and posed queries to infer the rules for 5 attributes. We collected all the answers and we checked that the tasks were assigned to distinct workers. We evaluated the correctness of each answer by means of the golden rules of our datasets, and then computed the error rate of each worker as the ratio between the number of erroneous answers and the number of answers.

The observed average error rate of a real worker was $\overline{\eta} = 10\%$, with a standard deviation $\sigma_\eta = 14\%$. Interestingly, about 27% of the workers responded correctly to all the queries.

The information gathered in this experiment has been used to set up the other experiments: the average error rate empirically observed $\overline{\eta}$ is used to set the parameter $\eta = \overline{\eta}$

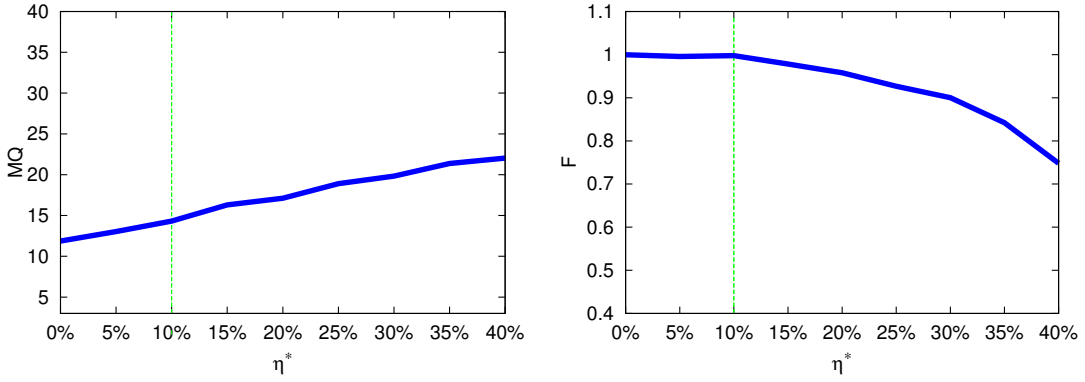


FIGURE 3.5: ALF$_\eta$, with $\eta = 10\%$, and variable worker error rate η^*: (Left) cost and (right) quality.

in ALF$_\eta$, and the error rate distribution observed on real workers is used to create population of synthetic workers with the same average error rate. To create populations of synthetic workers with a greater average error rate, we just scaled the error rates of the distribution on real workers.

3.7.5 ALF$_\eta$ Evaluation

We evaluated ALF$_\eta$ by conducting two experiments on attributes taken from our dataset. The main goal of these experiments was to evaluate the sensitivity of ALF$_\eta$ to its parameter η, i.e., the estimation of the worker error rate. We set $\lambda_r = 90\%$, and $\lambda_{MQ} = +\infty$, respectively: ALF$_\eta$ tries to reach the target probability of the best rule without any bound on the number of queries.

The first experiment aims at studying the effects produced by an inaccurate worker: we set $\eta = \overline{\eta} = 10\%$, and we run ALF$_\eta$ with synthetic workers with an increasing error rate η^* ranging from 0 to 40%.

Figure 3.5 reports the results of this experiment averaged over 20 executions, and it shows that as the actual error rate η^* of the workers increases, the results degrade: ALF$_\eta$ poses a larger number of queries, but the quality of the results, F-measure, decreases.

The second experiment aims at empirically evaluating how an incorrect setting of the parameter η, i.e., the expected worker error rate, influences ALF$_\eta$ performances. We used a single worker with $\eta^* = \overline{\eta} = 10\%$, and repeated several inference processes, configuring ALF$_\eta$ with η ranging again from 0 to 40%.

Figure 3.6 reports the results of this experiment (averaged over 20 executions): when the system overestimates the accuracy of worker ($\eta < \eta^*$) we observe a reduction of the number of MQ, but the quality of the output wrapper drops. The system trusts the

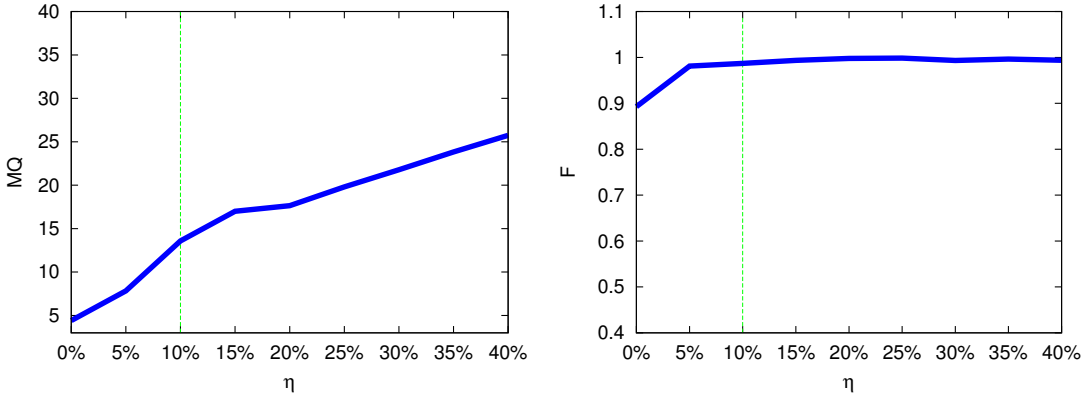


FIGURE 3.6: ALF$_\eta$, with a noisy worker $\eta^* = 10\%$, with variable η: (Left) cost and (Right) quality.

workers and terminates quickly, thus posing less queries than actually needed. When the system underestimates the worker accuracy ($\eta > \eta^*$), some queries are wasted since the system does not trust the worker, however there is no loss in the quality of the result. With an η larger than η^* by $+30\%$, ALF$_\eta$ requires more than 25 MQ, i.e., 2 times those required when $\eta = \eta^*$. Observe that many queries are wasted since the F-measure gain is less than 5%.

3.8 Conclusions

We propose a framework that allows supervised inference with simple membership queries suitable for the non-expert workers of a crowd platform. An original algorithm, ALF$_\eta$, applies active learning techniques to infer a wrapper, while minimizing the number of queries. Noisy answers are adopted to update a Bayesian Model that computes the probability of correctness of the candidate rules. We developed a complimentary sampling algorithm, PAGESAMPLER, to select for the learning phase a small yet representative set of sample pages from a much larger set of pages to wrap.

Experimental results prove the effectiveness of the approach. We can learn accurate wrappers with few membership queries. The sampling strategy leads to the selection of a small number of samples that effectively represents a much larger set of pages.

We observe that a correct estimation of the parameter η that models a worker's error rate highly improve the performance of ALF$_\eta$.

Chapter 4

Noisy Crowd

In the previous chapter we described ALF$_\eta$, an active learning algorithm that is able to infer a wrapper by selecting the queries that more quickly bring to the generation of an accurate wrapper, thus reducing the costs. ALF$_\eta$ relies on a probabilistic quality model that considers the presence of errors in the answers returned by a worker.

The algorithm ALF$_\eta$ computes the quality of an extraction rule by using answers provided by an inaccurate worker. However, the performances of the algorithm are strongly affected by the estimation of the worker's error rate. If the worker is overestimated, i.e. the algorithm expects that the worker performs better than she really does, the quality of the results is compromised. Conversely, if the worker is underestimated, i.e. it assumes that the worker performs worse than she actually does, the cost augments: since the algorithm does not trust the worker, it does not weight her answers enough, and it ends up posing more questions than actually needed.

A simple technique for estimating the workers error rate is to rely on the availability of ground truth information. To check the workers performance, they are asked a number of control queries, which have been already pre-labeled with the correct answers. However, this solution is expensive, because of the costs of preparing the ground truth, and because of the costs paid to the workers for answering the control queries rather than the real ones [49].

An alternative solution to evaluate workers performance without making use of any ground truth information is based on redundancy: the same tasks are assigned to several workers, and their error rate estimation relies only on their agreement with other workers. This approach is based on the assumption that independent workers make independent errors, which is indeed a realistic assumption with workers recruited on a crowdsourcing

platform.[1] On the one hand, redundancy offers the advantage of eliminating the costs of the ground truth information; on the other hand, it originates the additional costs of employing more workers on the same task.

In general, the use of redundancy raises the question of how much redundancy is actually needed to reach an optimal trade-off between costs and quality. In many situations it has been proved as good as those based on ground truth (e.g., [50]) for estimating the workers performance, with lower costs. Some theoretical bounds have been developed for specific (and simple) models (e.g., [51]), but usually the redundancy is established statically, i.e, before the tasks are assigned, and in a non-adaptive way, i.e., independently from their difficulty and from the provided answers.[2]

The chapter is organized as follows: Section 4.1 presents an overview of ALFRED; Section 4.2 describes the technique adopted to estimate η; Section 4.3 presents the a schedule of the tasks submitted in the crowdsourcing platform; Section 4.4 presents the evaluation of the scheduling algorithm; we conclude the chapter in Section 4.5.

4.1 Overview

ALFRED (ALF$_\eta$ with redundancy) builds on ALF$_\eta$, it improves the resilience to errors of ALF$_\eta$ by recruiting several workers on the same task. Unlike traditional approaches, which statically set the number of workers to employ on the same task, ALFRED decides the number of workers during the learning process, at runtime, thus minimizing the costs: it engages only the workers actually needed to achieve the desired quality. Also, ALFRED exploits the agreement among multiple workers to jointly estimate their error rates and the quality of the wrappers inferred: the larger the amount of redundancy, the better the estimation of the workers error rates, but the fewer resources are left for the inference process. To further optimize the costs, we consider a practical setting in which ALFRED assembles the tasks submitted to the crowd with queries to infer the wrappers of several attributes, and study the optimal amount of redundant attributes that should be allocated in each task. Our experimental evaluation shows that ALFRED can adaptively control the learning process to compensate the noise introduced by workers: it jointly estimates their error rates and allocates the right amount of redundancy to consistently achieve the target quality.

ALFRED represents the first proposal that exploits crowdsourcing to address wrapper inference. This Chapter makes the following contributions: (i) we develop a solution to

[1] Notice that this approach can be seen as a special case of the previous one, as a task with ground truth information can be seen as a redundant task solved by a *perfect* worker.

[2] For example, Marcus et al. [52] engage 5 workers per task.

decide at runtime how many workers should be recruited to deal with the presence of noisy answers; (ii) we present techniques to leverage redundant tasks to estimate the workers? error rates during the learning process; (iv) we report the results of an extensive experimental activity conducted with both synthetic and real workers recruited from a crowdsourcing platform.

4.2 Error Rates Estimation

To combine the efforts done by multiple workers on the same attribute, we rely on our probabilistic quality model that is easily extended to consider training data produced by several workers: it suffices to concatenate the t.s. obtained by several workers into a single training sequence.

We now denote L^w the t.s. produced by a worker w, L^w_A the t.s. produced by the worker w for the attribute A.[3] Also, we generalize the notation L_A to indicate the t.s. obtained by concatenating the t.s. of several workers for the same attribute A, that is, $L_A = \uplus_{w \in \mathcal{W}_A} L^w_A$, where $\uplus$ denotes the concatenation over several sequences, and $\mathcal{W}_A$ indicates the set of workers that provided labelled values for attribute A.

Given a t.s. L_A, we compute the p.d.f. $P(r|L_A)$ by using Eq. 3.1 (and Eq. 3.3). Such a p.d.f. can then be used to compute the error rate of a worker w, as the average number of incorrect answers provided in the t.s. L^w, weighted by the probability of each answer, as follows:

$$\eta_w = \frac{\sum_{v^l_A \in L^w} \begin{cases} 1 - P(v_A|L_A) & \text{, iff } l = + \\ P(v_A|L_A) & \text{, iff } l = - \end{cases}}{|L^w|} \tag{4.1}$$

where $P(v_A|L_A)$ is the probability that v_A is a correct value to extract for the attribute A, and, as it has been done before, it is reduced to the sum of the probabilities of the rules that extract v_A; that is: $P(v_A|L_A) = \sum_{r \in \mathcal{R}^v} P(r|L_A)$.

Observe the mutual dependency between Eq. 3.1 and Eq. 4.1: the former computes a p.d.f. for the correctness of the rules, given a training sequence and the error rates of the workers who labeled its values; the latter computes the error rates of each worker, given the p.d.f. associated with the rules.

[3] In the following, we consider workers producing t.s. for several attributes in a single task.

We exploit such mutual dependency by triggering an iterative process that interleaves the estimation of the error rates and the computation of the p.d.f. until a convergence criteria is satisfied, i.e., until these values do not significantly change anymore.[4]

4.3 Schedule

The approach described in the previous section allows us to dynamically choose the number of workers to be recruited for inferring the extraction rule of a single attribute. For every attribute we can submit a pair of tasks, each posing queries to infer the rules for the same target attribute. With the t.s. generated from these tasks, we can trigger the iterative process to jointly estimate the workers error rates and the p.d.f. over the candidate rules, as described in the previous section. If, at the end of the process, a quality criterion is not satisfied (e.g., the most likely correct rule has a low probability of correctness), other workers can be recruited on the same attribute, until they produce t.s. that allows the selection of the correct extraction rule.

However, we observe that on a real crowdsourcing platform it is not convenient to submit too short tasks composed of just a few membership queries related to a single attribute. As reported by Ipeirotis in his study on the Amazon Mechanical Turk (AMT) marketplace [53], 90% of the AMT tasks give a reward of 10¢, and the estimated hourly wage is approximately $5: the typical 10¢ task requires about 75 seconds to be fulfilled. On average, for every attribute ALF$_\eta$ poses a number of queries that are answered by an ordinary worker in around 15 seconds. Therefore, by following those guidelines, it is fairly reasonable to submit 10¢ tasks composed of queries related to 5 attributes.

For the sake of generality, we now present our solution for the composition of task assuming that each task includes N attributes. In practice, according to the considerations discussed about, in our experiments we set $N = 5$.

To guarantee a reliable estimation of the error rate (and thus of the probability of correctness of the rules) a straightforward solution is that of assembling tasks containing N attributes, and submit each task (at least) twice. However, we have experimentally observed that, due to the simplicity of our membership queries, the average error rate of real workers is rather low (10%), and thus for a significant percentage of attributes even one worker suffices to select the correct rule (we report details on these experiments in Section and 4.4.2).

[4] For our experiments we stop when all the error rates do not change, in absolute value, more than $\Delta_\eta = 10^{-4}$.

Therefore, instead of redundantly submitting a whole task, we prepare tasks with a limited overlapping: each task is composed of N attributes, and only a subset of K *redundant* attributes is included also in another task. With the results of the redundant portion of each task, we estimate the error rate of the worker that accomplished the task, and then we use such error rate estimation to compute the p.d.f. of the extraction rules for the remaining *non-redundant* attributes of the task.

Example 4.1. *Suppose given the set of attributes* $\mathcal{A} = \{A_1, A_2, \ldots, A_{10}\}$. *Assuming* $N = 3$ *and* $K = 1$, *the following tasks would be created:* $t_1 = \{A_1, A_2, A_3\}$, $t_2 = \{A_1, A_4, A_5\}$, $t_3 = \{A_6, A_7, A_8\}$, $t_4 = \{A_6, A_9, A_{10}\}$. *Note that* A_1 *and* A_6 *are submitted twice, in different tasks. A worker produces a sequence covering* N *attributes, e.g.,* w_1 *produces* $L^{w_1} = L^{w_1}_{A_1} \uplus L^{w_1}_{A_2} \uplus L^{w_1}_{A_3}$ *that we also denote* $L^{w_1} = [L^{w_1}_{A_1}, L^{w_1}_{A_2}, L^{w_1}_{A_3}]$.

Given $L_{A_1} = L^{w_1}_{A_1} \uplus L^{w_2}_{A_1} = [L^{w_1}_{A_1}, L^{w_2}_{A_1}]$, *we compute the p.d.f.* $P(r|L_{A_1})$ *of the redundant attribute* A_1 *and the error rates* η_{w_1} *and* η_{w_2} *of the involved workers. Similarly, we compute* $P(r|L_{A_6})$, η_{w_3} *and* η_{w_4} *and any* $P(r|L_{A_i})$ *and* $\eta_{w_{k+1}}$ *for* $k = 0, 1, 2, \ldots$ *and* $i = 1 + k(2N - 1)$.

These error rates are then used to recompute the p.d.f. of any redundant attributes in the same tasks, such as $P(r|L_{A_2})$, $P(r|L_{A_3})$, $P(r|L_{A_4})$, $P(r|L_{A_5})$, $P(r|L_{A_7})$, ... *and any* $P(r|L_{A_i})$ *such that* $i \neq 1 + k(2N - 1)$.

Those attributes whose inferred extraction rules do not satisfy a quality criterion are used to compose new tasks, which are submitted again to the crowdsourcing platform. The new sequences returned by these tasks feed a new estimation of the error rates of the workers that elaborated the same attributes, as well as the updating of the p.d.f. of all the involved attributes.

Example 4.2. *Continuing the previous example, suppose that the probability attributes* A_2, A_3 *and* A_7 *do not reach the target quality. Then, in order to acquire more labeled values for these attributes, their candidate values will be used to formulate the queries of a new task* $t_5 = \{A_2, A_3, A_7\}$ *to be submitted to the crowdsourcing platform.*

The produced t.s. $L_{A_2} = [L^{w_1}_{A_2}, L^{w_5}_{A_2}]$ *is used to recompute all the error rates of the workers directly involved on the attributes of the new task* t_5, *e.g.,* η_{w_5} *has to be computed and* η_{w_1} *has to recomputed. Notice also that* w_2, *that worked on* A_2, *is indirectly involved since its error rate* η_{w_2} *depends on* $P(r|L_{A_2})$ *that depends on* η_{w_1}. *Transitively following these dependencies, it turns out that the error rates of all the workers need to be recomputed, and hence all the p.d.f. of all attributes in these five tasks.*

Listing 3 illustrates the pseudo-code of the ALFRED algorithm, which implements the above approach. Here the crowdsourcing platform is modeled as a queue q abstracting a

Listing 3 ALFRED

Input: a set of attributes $\{A_1, A_2, \ldots, A_n\}$;

Parameter λ_r: target probability of correctness
Parameter λ_{MQ}: maximum budget for each worker per attribute
Parameter N: number of attributes per task
Parameter K: number of redundant attributes per task

Output: the set of the most probable extraction rules $r \in \mathcal{R}$, $\forall A \in \{A_1, A_2, \ldots, A_n\}$;

1: **let** $\mathcal{C} = \varnothing$; *// set of completed tasks*
2: $q.submitAll($ CREATETASKS$(\{A_1, \ldots, A_n\}, N, K, \lambda_r, \lambda_{MQ}))$; *// create initial tasks*
3: **while** (**not** $q.isEmpty()$) **do**
4: **let** $\mathcal{U} = \varnothing$; *// set of unsolved attributes (already with a t.s. but need more...)*
5: **let** $t = q.take()$; *// removes a fulfilled task from the completion queue*
6: $\mathcal{C} \leftarrow \mathcal{C} \cup \{t\}$; *// save as completed*
7: **let** $\mathcal{G}$ **be** << *the bipartite assignment graph of the submitted tasks* >>;
8: **let** $\mathcal{G}(t)$ **be** << *its connected component including* t >>;
9: **let** $\mathcal{T} \leftarrow$ *all the tasks in* $\mathcal{G}(t)$;
10: **let** $\mathcal{A} \leftarrow$ *all the attributes in* $\mathcal{G}(t)$;
11: **if** $(\mathcal{T} \subseteq \mathcal{C})$ **then**
12: *// all the tasks of the connected component that includes t have been completed*
13: **let** $R \leftarrow \{A \in \mathcal{A}$ such that $|\mathcal{W}_A| \geq 2\}$; *// redundant attributes*
14: compute $P(r|L_A)$ and η_w with Eq. 3.1 and Eq. 4.1, resp., $\forall r \in \mathcal{R}, \forall w \in \mathcal{W}_A, \forall A \in R$;
15: **let** $N \leftarrow \{A \in \mathcal{A}$ such that $|\mathcal{W}_A| = 1\}$; *// non-redundant attributes*
16: compute $P(r|L_A)$ with Eq. 3.1 with η_w as computed on line 14, $\forall r \in \mathcal{R}, \forall A \in N$;
17: $\mathcal{U} \leftarrow \mathcal{U} \cup \{A \in \mathcal{A}$ such that **not** HALT$(L_A, \lambda_r, +\infty)\}$;
18: **end if**
19: $q.submitAll($ CREATETASKS$(\mathcal{U}, N, K, \lambda_r, \lambda_{MQ}))$;
20: **end while**
21: **return** $\{r \in \mathcal{R}$ such that $r = \text{argmax}_{r \in \mathcal{R}} P(r|L_A)$, $A \in \{A_1, A_2, \ldots, A_n\}$ $\}$;

completion service to which task submissions are performed by means of a non-blocking operation $q.submitAll()$; completion notifications are provided via a blocking operation $q.take()$ that removes the next completed task from the queue, or just wait if none of the submitted task has been fulfilled yet.

Starting from the input set of attributes, a set of tasks are initially submitted to the crowd (line 2) following the redundancy scheme illustrated above: Each task is composed of queries related to N attributes, with K attributes per task assigned also to another task. We model the composition of the submitted tasks in attributes by means of a bipartite graph, denoted $\mathcal{G}$ (line 7), whose nodes are either attributes or tasks, and there is an edge between a task and an attribute if and only if the task includes queries on the attribute. Figure 4.1 (right) shows the bipartite graph for the tasks described in Example 4.1. Observe that $\mathcal{G}$ is composed by several connected components. Each connected component includes tasks that are related by some shared redundant attribute:

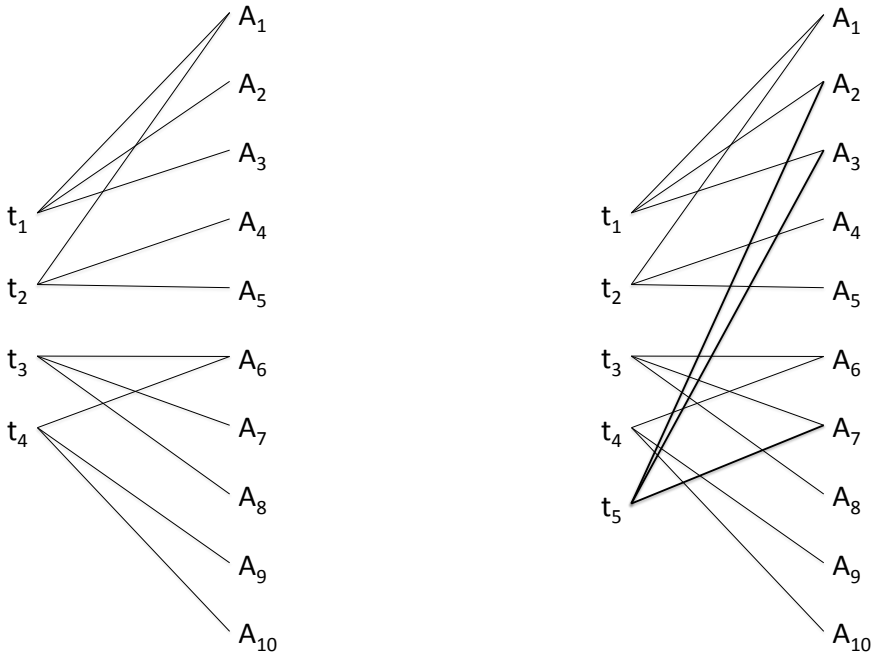


FIGURE 4.1: The bipartite graph for the task allocation of Example 4.1 (left) and Example 4.2 (right).

all the p.d.f. associated with the attributes, and all the error rates of the workers in the component, are mutually dependent. Given a task t, we denote $\mathcal{G}(t)$ the connected component of $\mathcal{G}$ that includes t (line 8).

While the queue is not empty (lines 3–20), i.e., there are tasks yet be completed by the crowd, the first ready task t is taken from the queue, and it is added to a set of completed tasks, $\mathcal{C}$ (line 6). If every task of the connected component $\mathcal{G}(t)$ have been already completed (line 11), the t.s. produced for the attributes of these tasks, $\mathcal{A}$, can be processed. First, the workers error rates and the p.d.f. of the associated extraction rules are computed (line 14) for the redundant attributes (i.e., attributes A such that $|\mathcal{W}_A| \geq 2$). The estimated error rates are then used to recompute the p.d.f. of the extraction rules for the remaining (non redundant) attributes of the same connected component (line 16).

All the attributes that do no reach the quality target[5] are added to a set of *unsolved* attributes (line 17): for these attributes the collected t.s. did not lead to the production of a satisfactory extraction rule, and thus they will be added in a new task, which is submitted to the crowdsourcing platform (line 19).[6] It is worth noting that all the unsolved attributes have been already processed at least by one worker and they become

[5]We neglect any budget issue at ALFRED's level where the goal is to reach the quality target λ_r; however ALF_η bounds to λ_{MQ} the budget per attribute spent for each worker.

[6]For the sake of simplicity we are assuming that $|\mathcal{U}|$ is a multiple of N. Otherwise, the tasks can be completed by inserting control attributes with known answers to better estimate the workers error rate.

redundant: they might trigger the merging of several connected components of the graph thus creating a component with a larger diameter that includes a larger number of attributes.

Figure 4.1 (right) illustrates the graph for the tasks of Example 4.2: observe that task t_5, which is composed by the unsolved attributes A_2, A_3, A_7, creates a component that includes all the attributes. Before t_5 had been submitted, the same attributes were spread in two separate components.

4.4 Experiments

In this section we evaluate ALFRED, our system built from ALF$_\eta$. For instance: in Section 4.4.1 we describe a third dataset based on a previous work; in Section 4.4.2 we focus on the impact of redundancy by considering tasks related to a single attribute and in Section 4.4.3, we report the results with tasks covering $N > 1$ attributes each, and then study how our algorithm performs in presence of redundancy confined to only a subset of K (with $0 \leq K \leq N$) attributes. Section 4.4.4 reports our final experiments on ALFRED with real workers.

4.4.1 Datasets

In the previous chapter we presented two datasets, one to evaluate the sampling algorithm 3.2 and one generic 3.3. In this chapter we will evaluate our techniques on 3.3 and on a third one. The new dataset is the public SWDE dataset [54],[7] which includes about 124,000 pages from 80 websites related to 8 different domains (10 sites per domain, 200–2,000 pages per website). From the SWDE dataset, we used 34 websites: those whose HTML pages were correctly rendered at the time of the submission to the crowdsourcing platform (pages from the discarded websites were not correctly rendered because their images or CSS files were not available anymore). For the collections of pages of this dataset, we selected a total of 127 attributes, and we manually crafted a golden extraction rule for each of them.

The learning algorithms used in the evaluation were run on a small yet *representative* sample set of pages (e.g., a few dozens of pages), selected by a suitable sampling strategy described in Chapter 3 from the whole set of pages. The pages in these samples are carefully chosen to guarantee that a wrapper inferred on the sample also works on the whole set of pages. The extraction rules obtained on the representative sample were then evaluated on the test set.

[7] http://swde.codeplex.com

	average				max		
	$\#w$	F	$\#MQ$	$\lvert\eta_w - \eta^*\rvert$	$\#w$	$\#MQ$	σ_F
ALF$_\eta$	1	0.96	9.15	—	1	11	15%
ALFRED$_{no}$	2.37	1	23.35	—	8	80	0.29%
ALFRED	2.13	1	20.7	0.8%	4	40	0.18%
ALFRED*	2.11	1	20.27	0%	4	40	0.18%

TABLE 4.1: ALFRED vs ALF$_\eta$ with a population of synthetic noisy workers; average and max total number of workers engaged per attribute ($\#w$); average F-measure of the output wrapper; average and max total number of queries ($\#MQ$); average difference between actual and estimated worker error rate ($|\eta_w - \eta^*|$); standard deviation of the output wrapper F-measure (σ_F).

In the following, we measure the cost in terms of the number of membership queries, and rather than focusing on the F-measure value of the output wrapper, which is always very close to 1 in the settings that we consider, we report its average standard deviation (σ_F) over several executions. The latter is a simple measure of the predictability of the output quality of the learning process.

4.4.2 Impact of Redundancy

As discussed in Section 4.2, ALFRED submits the inference of the same attribute to several workers. It lazily recruits additional workers, at runtime, to estimate their error rate while minimizing the costs.

Although in practice it is convenient to assemble tasks composed of queries related to several attributes (as we discussed in Section 4.3), in this experiment we focus the study on the solely role of redundancy: we run ALFRED with tasks composed by a single redundant attribute, i.e., $K = N = 1$.

Table 4.1 reports the results of the experiment averaged over 20 executions in which ALFRED recruits workers from a population of synthetic workers with the same error rates distribution observed over real workers. We compare the algorithm against a baseline (ALFRED$_{no}$) in which the error rate estimation is disabled (we just set $\eta_w = \overline{\eta}$ without using Eq. 4.1), and against a bound (ALFRED*) in which an oracle sets $\eta_w = \eta^*$ (since the workers are synthetic, their actual error rates are known). Also, in order to emphasize the impact of the redundancy, we report the performance of ALF$_\eta$, which does not rely on redundant tasks. Observe that the workers error rate estimation is precise ($|\eta_w - \eta^*| = 0.8\%$ when the learning terminates), and it allows the system to save queries (20.7 vs 23.35 on average). The average number of queries posed by ALFRED to learn the correct rule is very close to the lower bound set by ALFRED*. Compared to ALF$_\eta$, which employs a single worker, the number of queries is more than twice (20.7 vs. 9.15).

However, notice that ALFRED always concluded the tasks with an almost perfect result, and therefore it is much more robust to variations in the workers error rates as shown by the standard deviation of the F-measure (ALFRED's $\sigma_F = 0.18\%$ vs ALF$_\eta$'s $\sigma_F = 15\%$).

	#workers		
	2	3	4
$\%attr.$	89%	10%	1%
$\#MQ$	19.62	30	40

TABLE 4.2: ALFRED: percentage of attributes ($\%attr.$) that reach the target quality with 2, 3, and 4 workers; their average cost as total number of membership queries posed ($\#MQ$).

Overall, ALFRED was able to recruit more workers, thus paying their answers, only when it is necessary to achieve the target quality of the output wrapper. However, consider Table 4.2 that aggregates the same results in terms of the number of recruited workers (#workers) and the number of MQ posed ($\#MQ$) per attribute: in a large majority of cases (89%), ALFRED terminates recruiting only 2 workers, and seldom 3 and 4 workers (10% and 1%, respectively) with an average of 2.13 workers engaged and 20.7 queries posed per attribute.

So it is reasonable to conjecture that for a significant portion of these attributes requiring only 2 workers, similar results can be achieved even with less redundancy, and then with lower costs.

The experiments presented in the next section investigate and confirm such a conjecture, thus motivating the study of a more complex scheme of redundancy than that discussed here.

4.4.3 ALFRED Evaluation

We now present our experimental evaluation of ALFRED in the most general setting: we consider tasks composed of N attributes, each containing K redundant attributes. The goal of the experiment is to study how ALFRED is affected by the amount of redundancy initially introduced in the tasks, expressed as the value of K. Therefore we run several experiments with K ranging from 0 to N.

We analyze the cost per attribute versus the ability of ALFRED to reach the target quality independently from the noise introduced by the workers. We run the system with different populations of synthetic workers, and for each experiment we averaged the results over 100 executions. In the next section we present experiments with a

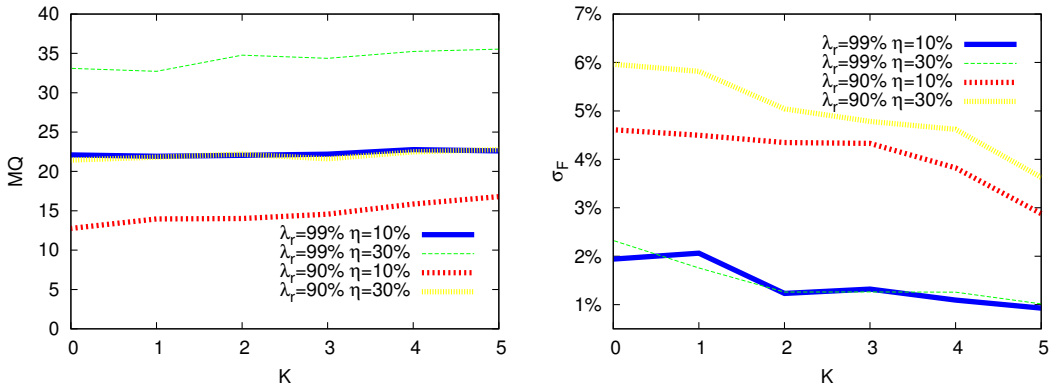


FIGURE 4.2: The effects of the initial redundancy K: (left) average cost: $\#MQ$; (right) standard deviation of output F-measure: σ_F.

population of real workers recruited on CrowdFlower, selecting the number of redundant attributes K according to the experiments with synthetic workers.

Figure 4.2 reports the results obtained with synthetic workers from two distinct populations: the first one is composed of workers with an average error rate $\eta^* = 10\%$, while the second one is more noisy, with $\eta^* = 30\%$. We also consider two different quality targets by considering $\lambda_r = 90\%$ and $\lambda_r = 99\%$.

Overall ALFRED achieves high quality ($F > 99\%$) with a low standard deviation ($\sigma_F < 6\%$) in all the configurations considered. As shown in Figure 4.2 (left), ALFRED recognizes and compensates a large amount of errors (when $\eta^* = 30\%$) by augmenting the number of queries posed with respect to the other population (with $\eta^* = 10\%$).

As regards the behavior of ALFRED versus the initial amount of redundancy as K grows, Figure 4.2 (left) shows that for the population of less noisy workers ($\eta^* = 10\%$), the cost increases from an average of around 13 ($K = 0$) to 17 ($K = 5$) queries per attribute, while σ_F, in Figure 4.2 (right), decreases from 6% to 3.5%. Both trends are less visible when $\eta^* = 30\%$ than when $\eta^* = 10\%$: With a population of noisy workers, ALFRED quickly detects that a larger amount of redundancy is needed to achieve the quality targets, and therefore the initial amount of redundancy is augmented towards the level reached at the end of the simulation, that depends on the initial values of K only loosely.

As a particular example of this behavior, consider Figure 4.2 (left) when $K = 0$ and $\eta^* = 30\%$: this corresponds to an "optimistic" approach, in which the initial tasks are not redundant at all, and the workers error rates estimations are set with the initial fixed parameter $\eta = 10\%$. Even if ALF_η is overestimating the workers (the actual average error rate of the workers in this population is $\eta^* = 30\%$), thus giving them more trust than they deserve, the redundancy is introduced as soon as ALFRED detects that the quality targets are not reached, and it ends up with almost the same amount of redundancy of

a "pessimistic" approach in which all the attributes of the tasks are already redundant at the beginning of the simulation ($K = 5$).

It is worth observing that the quality target, i.e., the threshold λ_r, has an impact on ALFRED's effectiveness. As the plots in Figure 4.2 show, increasing $\lambda_r = 90\%$ to $\lambda_r = 99\%$ pushes ALFRED to quickly increase the initial redundancy to reach the stricter quality target, and the initial redundancy K does not have a strong impact on the results. In settings with really high value of λ_r, ALFRED's sub-task redundancy does not lead to any saving in the costs.

	#workers			
	1	2	3	4
$\%attr.$	58%	31%	8%	3%
$\#MQ$	7.2	16.5	26.68	37.21

TABLE 4.3: ALFRED (with $N = 5$, $K = 0$, $\lambda_r = 90\%$, $\eta = 10\%$): percentage of attributes ($\%attr.$) that reach the target quality with 1, 2, 3, and 4 workers; their average cost as total number of membership queries posed ($\#MQ$).

Conversely, Table 4.3 has been obtained with $K = 0$, $N = 5$, $\lambda_r = 90\%$, and $\eta = 10\%$, and reports the percentage of attributes grouped by number of distinct workers employed on them. The majority of attributes (about 58%) were assigned to only 1 worker (i.e., without redundancy), 31% of the attributes required 2 workers, and just 11% of the attributes needed to be assigned to more than 2 workers. The number of workers needed for an attribute was 1.57 on average, with 12.7 queries, with a significant saving compared to the simplistic redundancy scheme that allocates at least two workers for each attribute.

4.4.4 ALFRED on the Crowd

We evaluated ALFRED with real workers recruited on the CrowdFlower crowdsourcing platform. We chose a configuration for which ALFRED produces good results in our simulations with synthetic workers, while producing a significant saving in the costs: we set $N = 5$, $K = 2$ and $\lambda_r = 90\%$.

These experiments have been conducted by randomly selecting 100 attributes from 35 websites within our two datasets. To generate the extraction rules for the 100 attributes, a total of 34 tasks were submitted, and executed by the same number of (distinct) workers. After the first submission of 25 tasks, only other 8 tasks have been created, with a single attribute requiring 5 workers.

The total cost for inferring the extraction rules of 100 attributes was \$3.4 with an average cost per attribute of 3.4¢. The tasks were completed in 6 hours with an average $F = 99.7\%$ and standard deviation $\sigma_F = 1.8\%$. Workers answered to $1,286$ queries

Overall	Instructions Clear	Test Questions Fair	Ease of Job	Pay
3.8 / 5	4.1 / 5	3.8 / 5	3.7 / 5	3.6 /5

TABLE 4.4: Evaluation of our tasks by the CrowdFlower workers.

with an average number of around 37 queries per task. The average error rate observed during this session was $\overline{\eta} = 10.5\%$.

CrowdFlower also provides feedbacks from the workers about the requester and the submitted jobs. Table 4.4 reports the scores that we obtained from the workers that fulfilled our tasks.

4.5 Conclusions

We presented wrapper inference algorithms specifically tailored to exploit crowdsourcing solutions. Our approach allows the generation of wrappers by means of training data obtained by posing simple queries to workers recruited on a crowdsourcing platform. In this chapter we proposed ALFRED that is built on ALF$_\eta$. ALF$_\eta$ can infer a wrapper with the labeled data produced by a single worker, but open issues are: the estimation of the worker's error rate η and the schedule of the tasks for the crowd. ALFRED addresses these open issues by estimating in runtime η, ALFRED adopts an *EM* approach by exploiting the mutual dependency between the workers' error rate and the quality of the inferred wrapper. To reduce the cost, ALFRED adopts a quality model to dynamically set the number of workers to recruit in the same task. To optimize ALFRED we consider the schedule of the tasks in a real crowdsourcing platform. Workers are enrolled to generate wrappers with ALF$_\eta$ for several attributes; ALFRED optimizes the presence of several attributes by using only a part of them to estimate η.

An extensive evaluation shows that ALFRED can produce high quality wrappers at reasonable costs, and that the quality of the output wrapper is highly predictable. We show that with an accurate scheduling of the tasks, we can achieve accurate wrappers with an average of 1.57 tasks per attribute.

Chapter 5

Automatic Responders

In the previous chapters we defined a crowd based solution and we minimized its cost by estimating workers' error rate and selecting at run time the correct number of redundant tasks. In our evaluation we observed that for the majority of the cases redundancy is not required and this motivated the work in Chapter 4 where we introduced a scheduling technique so that we can estimate worker's error rate based only on a portion of the tasks. We adopted ALFRED to estimate η on a subset of the tasks, while the other part of the tasks were completed by a single worker adopting the estimated worker' η. With a combination of the schedule algorithm and ALFRED we were able to accurately infer a wrapper with an average number of 1.57 workers per attribute and on average 12.7 MQ per attribute. In this chapter our goal is to further reduce the average number of required workers and the number of MQ without penalizing the quality of the output. To achieve this goal we observe that there are many automatic wrapper inference systems in literature that can achieve a reasonable quality in output if some conditions of the input are met [4, 5, 8–11]. For instance, they consider the presence of: a template without ambiguity [4, 5], a database or annotators [8], a domain knowledge [9], redundancy from the Web [10, 11]. All these approaches rely on some features that can be used to guide the extraction process to infer wrappers, but these solutions suffer of the same limitations: (i) at Web scale there are many cases where these systems are not applicable, (ii) the quality can drop if the conditions are not perfectly met, (iii) there is not a reliable quality control of the output.

From one hand we have several proposals that consider different features to generate wrappers, from the other hand we have the scale and the variety of the Web. All these approaches considered singularly can not scale to all the Web, but there is a subset of the Web that can be wrapped by one or some of them. This motivates an approach that is in part automatic but when the system finds out that the automatic

approach is not reliable enough, humans are involved to reduce the uncertainty left by the automatic approach. This approach opens the possibility to further reduce the human work required to complete a task. For instance, we could go from 1.57 workers per task to less than 1 worker. To understand when a solution is not reliable we implement several automatic wrapper inference systems and we combine them with ALFRED to estimate the expected wrapper quality and decide at runtime if a crowd work is required.

A similar approach has been adopted by [55, 56]. The intuition is that for several domains it is possible to define *independent processes* extractors (or learners) to extract triples from the Web [55] (or match two different schemas [56]). In one case [56] learners were trained from an initial supervised approach, in the other case [56] extractors are simple extraction heuristics. All these works follow the intuition that mistakes can be automatically discovered by the system considering the incongruent responses of different *processes*. In fact, it is unlikely that all the *processes* make the same mistakes, thus incongruences in the responses provides an evidence of the presence of mistakes. In this way a further supervision is submitted, but this is required only for cases when the automatic solutions end up with an uncertain solution.

Similar to them, we define four independent wrapper inference systems, we combine the responses provided by all our *processes* with ALFRED and we select at runtime if real workers are required or not. The described wrapper inference systems are: (1) *Types* that observe the types of the extract values, (2) *LFEQs* that exploits the regularity of templated pages, (3) *Knowledge Base* that supervises the system with an existing Knowledge Base and (4) *PMI* that exploits the correlation between the extracted values considering the Web. To combine the answers provided by these automatic wrapper inference systems we design them to generate distinct training sequences, each following their own strategy. Training sequences are then combined with ALFRED as described in the previous chapters, the system converges to the most likely rule by exploiting the mutual dependency between the error rate of the responder and the expected quality of the extraction rules. We present two new scenarios: (i) An automatic approach that combines several automatic wrapper inference systems, (ii) An hybrid approach that starts considering an automatic solution and on uncertainty it involves the crowd. We compare these two scenarios with the crowd based solution described in the previous chapters.

We present an extensive evaluation that shows: the automatic approach based on the combination of several systems is the most accurate ($F = 0.94$) and that it is able to find cases where the uncertainty is lower ($F = 0.99$ in 71% of the attributes); the hybrid approach can achieve almost the same quality of the human based solution with just a

-0.01 in F but requiring only a fraction of the MQ required by the completely human solution.

This Chapter is organized as follow: Section 5.1 presents preliminaries about the creation of a responder from a wrapper, in the following Sections we describe four different wrapper inference approaches, based on Types (Section 5.2), LFEQs (Section 5.3), Knowledge Base (Section 5.4), PMI (Section 5.5), Section 5.6 presents an extensive evaluation with automatic responders and a comparison with a solution humans-based and a hybrid approach.

5.1 Preliminaries

In this section we provide some preliminaries for this chapter. First we define a wrapper as a responder for ALF$_\eta$ 5.1.1, then we describe a wrapper as a scoring function, this approach is adopted by the wrapper inference systems 5.1.2 considered later.

5.1.1 Automatic Responder

Listing 4 ALF$_\eta$: Active Learning Algorithm With Automatic Responders

Input: a set of pages U
Input: an extraction rule generated by a wrapper inference system $r* U$
Input: the set of candidate rules $\mathcal{R}$

Parameter η $r*$ error rate
Parameter λ_r: target probability of correctness
Parameter λ_{MQ}: maximum budget

Output: a p.d.f. describing the probability of correctness of the rules in $\mathcal{R}$ and the t.s L

1: **let** $L = \varnothing$;
2: **while** (**not** HALT$(L, \lambda_r, \lambda_{MQ})$) **do**
3: $v_A \leftarrow$ CHOOSEQUESTION(L);
4: $l \leftarrow$ GETANSWER(r^*, v_A);
5: compute $P(r|v_A, L)$, $\forall r \in \mathcal{R}$ with Eq. 3.1 and Eq. 3.3 ;
6: $L \leftarrow L \cup \{v_A^l\}$;
7: **end while**
8: **return** $P(r|L)$, $\forall r \in \mathcal{R}$;

A wrapper is expressed as an extraction rule. The wrapper is given in input to ALF$_\eta$ so that we generate the training sequence that corresponds to the considered wrapper. ALF$_\eta$ poses MQ to reduce the uncertainty, but instead of asking to a human Oracle we interact with the input wrapper; the wrapper answers to the posed MQ by matching the query to the extracted values of the wrapper. When ALF$_\eta$ terminates the computation it

returns the training sequence required by ALF_η that satisfies the termination condition. Consider the Listing 4, given the set of pages U, a wrapper r^* generated for the set of pages and a set of candidate rules $\mathcal{R}$, we adopt the extraction rule r^* to provide answers to the function GETANSWER. The GETANSWER takes into account the value extracted by r^* for the considered page and the questioned value. The answer is positive if the questioned value is also extracted by r^*, otherwise the answer is negative.

$$\text{GETANSWER}(r^*, v_A) = \begin{cases} Positive & \text{, iff } r^*(p_{v_A}) = v_A \\ Negative & \text{, otherwise} \end{cases}$$

ALF_η provides questions until a termination condition is met, the system returns the most likely rule and the training sequence L.

5.1.2 Rules Selection

The wrapper inference process can be separated in two components: the first component provides the candidate rules while the second component selects the best candidate. If we give as granted the generation of the candidate rules, by considering the rules generated by ALF_η, then a wrapper inference system can be identified by the policy that is adopted to select the best extraction rule. Following this observation, a wrapper inference system can be described as a scoring function that given a rule r it provides a score for the rule $\mathcal{S}(r)$, thus the selection of the best rule r^* is reduced to find the rule that maximizes the function $\mathcal{S}$.
For instance, given a set of candidate rules $\mathcal{R}$ a wrapper inference system $\mathcal{W}$ is described by a scoring function $\mathcal{S}_\mathcal{W}$ on $\mathcal{R}$. The scoring function provides for each $r \in \mathcal{R}$ a score $\mathcal{S}_w(r)$ and the output wrapper is the rule that maximizes $\mathcal{S}_\mathcal{W}$.

$$\mathcal{W}(\mathcal{R}, U) = \underset{r \in \mathcal{R}}{\text{argmax}}\, \mathcal{S}_w(r)$$

We can define different scoring functions; the scoring functions can be modified to that generate different wrappers.

5.2 Type

A possible way to select the best extraction rule is by selecting the rule where the Types of the extracted values are more similar.

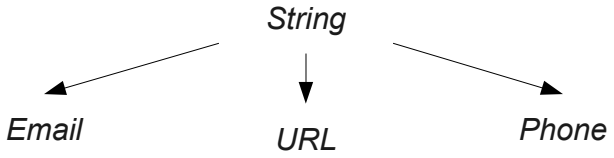


FIGURE 5.1: Types hierarchy for the Running Example

In this section we introduce the concepts of attribute *type* and of *types-hierarchy* (Sec 5.2.1) and propose an automatic responder that considers more likely rules extracting homogeneous values of the same *type* (Sec 5.2.2).

5.2.1 Types Definition

We call *type* a set into which values of pages in U can be organized. For example, the type *String* and one of its possible *sub-type* is *Email*. A set of types can be structured into a *disjunctive types-hierarchy*:

Definition 5.1. A *Disjunctive Types Hierarchy* Γ is a set of types $\Gamma = \{T_0, T_1, T_2 \dots T_n\}$ such that for every pair of its types $T_i, T_j \in \Gamma$: either (i) $T_i \subseteq T_j \vee T_j \subseteq T_i$, or (ii) $T_i \cap T_j = \varnothing$.

Furthermore, there exists a unique maximal type T_0, called the *root* type, wrt the inclusion relationship such that $\forall T \in \Gamma : T \subseteq T_0$ and $\forall v \in p, p \in U : v \in T_0$.

According to this definition, there exists a partial order amongst types in the hierarchy, and it is possible to define an operator $\tau(\cdot)$ returning the *minimal* type $T \in \Gamma$ over any value v of pages in U: $\tau(v) = T \Leftrightarrow v \in T, T \in \Gamma \wedge (v \in T', T' \in \Gamma \Rightarrow T \subseteq T')$. This operator can be trivially overloaded to work on a vector of values, rules, samples and labelled samples. We set by definition that $\tau(nil_p) = \bot, \forall p \in U$ considering the *nil* values as belonging to every type of the hierarchy.

It is worth noting that the operator $\tau(\cdot)$ returning the minimal type induces a partition on every set of non-*nil* values, rules, vectors of values, whereas the types do not.

5.2.2 Scoring with Types

Correct vectors are likely to have a sequence of values of the same types, i.e. it is more likely that a rule extracting all *Email* values is correct rather than a rule mixing *String* and *Email* values. In our case we have a first annotation v_0^l that provides a strong evidence of the main *type* of the correct vector. Based on this observation we define a

scoring function that selects the extraction rule that best match the type of the first annotation $T_0 = \tau(v_0^l)$.

Based on the type of the first annotation we define the scoring function as follows:

$$\mathcal{S}_{Types}(r, T_0, \Gamma, U) = \sum_{p \in U} \begin{cases} 1 & \text{, iff } \tau(r(p)) = T_0 \\ 0 & \text{, otherwise} \end{cases}$$

The function takes in input a rule $r \in \mathcal{R}$, the type associated to the initial value T_0, a disjunctive types hierarchy Γ, and the set of all the pages U. The rule r is scored considering a function that matches the type T_0 with the type of the extracted value $\tau(r(p))$ in each page in U. The function simply scores 1 if there is a perfect match between the initial type T_0 and the type of the extracted value.

5.3 LFEQ

Another possible way to select the best extraction rule is by exploiting the regularity of the HTML template. In this section we describe an automatic responder based on LFEQs (Large and Frequently occurring EQuivalence classes) a state of the art automatic wrapper inference system described in [5].

In this Section, we will introduce the definition of LFEQs and provide an intuition of the automatic wrapper inference described in [5] 5.3.1 and we introduce our automatic responder based on LFEQs 5.3.2.

5.3.1 LFEQs definition

Templated pages are generated by a common script that embeds values in HTML tags; values are returned from an underlying database. Templated pages are characterized by the presence of EQuivalence classes (EQ) that represents portions of the HTML pages that occurs exactly the same amount of time. If we observe the HTML tag and the body tag, it is obvious that a page that contains an HTML tag it contains also the body tag. If this analysis is reproduced in templated pages, we can find different kinds of EQs that describe the template structure of the pages [5]. While some EQs are likely to be present on a subset of the pages, there are several EQs that are likely to be present in many pages and these EQs are the ones that characterize the most the template, the template tags are likely to be frequent in all the templated pages. This subset is described as LFEQs, i.e. EQs that are frequent for that template.

The wrapper generation process described by [5], first infers the LFEQs that describe the template structure of the pages and in a second step it extracts the values embedded in these tags. To extract the contents, the intuition is that values of interest are embedded in the templated tags, thus for each value to be extracted the tag in which the value is embedded belongs to an LFEQ.

We can formally define $\Lambda(U) = \{L_0, L_1, L_2 \dots L_n\}$ as a set of LFEQs generated for a set of pages U. To find the LFEQ associated to a value v we define an operator $\lambda(v)$ that finds the LFEQ of the template tag that embeds the value v. As for the *types*, for each value on the pages we can provide a single label that in this case is described by the template structure.
In some cases the values are embedded in EQs that are not frequent enough, thus there is not an LFEQ associated to the value. In contrast to *Type*, the LFEQ L_{no} is considered different from the other LFEQs in $\Lambda(U)$. We overwrite the operator $\lambda(v)$ to return a L_{no} that marks the absence of an LFEQ associated to v. As for the *Type*, we have to model the presence of *nil*, we define that $\lambda(nil) = \bot \forall\ p \in U$ and we model *nil* as a value that do not belong to any LFEQs in $\Lambda(U)$.

5.3.2 Scoring with LFEQs

Correct vectors are likely to have a sequence of values that belong to the same LFEQ, i.e. it is more likely that a rule extracting all values embedded in a single EQ is correct rather than a rule that extracts some values in a EQ and some values in another EQ. As for the types, we have a first annotation v_0^l that provides a strong evidence of the LFEQ of the correct vector. Based on this observation we define a scoring function that selects the extraction rule that best matches the LFEQ of the first annotation $L_0 = \lambda(v_0^l)$.

Based on the type of the first annotation we define the scoring function as follows:

$$\mathcal{S}_{LFEQs}(r, L_0, \Lambda(U)) = \sum_{p \in U} \begin{cases} 1 & \text{, iff } \lambda(r(p)) = L_0 \\ 0 & \text{, otherwise} \end{cases}$$

As for the *Type*, the function takes in input a rule $r \in \mathcal{R}$, the LFEQ associated to the initial value L_0, the set of all the LFEQs generated on U. The rule r is scored considering a function that matches the initial LFEQ L_0 with the LFEQ of the extracted value $\lambda(r(p))$ in each page in U. The function simply scores 1 if there is a perfect match between the initial LFEQ T_0 and the LFEQ of the extracted value.

5.4 Knowledge Base

Another possible way to select the best extraction rule is by exploiting the semantic meaning of the extracted values. In this section we describe an automatic responder based on the Knowledge Base, the system matches the extracted values with the triples in the Knowledge Base and scores extraction rules that matches better.

In this Section, we will introduce the notation of a Knowledge Base 5.4.1 and we introduce our automatic responder based a Knowledge Base 5.4.2.

5.4.1 Knowledge Base definition

A *Knowledge Base* (KB) is expressed as a set of triples $t =< s, p, o >$, subject, predicate and object. The predicate represents a relationship between the subject entity and the object entity.
In the KB, we add the concept of the *type* of an entity. We extend the previous definition so that each entity e in the KB is composed by a pair (t_e, v_e), with t_e the *type* of the entity and v_e the value of the type.
A subject and an object in a triple represents the relationship between two entities, both subject and object are described by a pairs $s = (t_s, v_s), o = (t_o, v_o)$.
We say that a triple $< s, p, o >$ is related to t if the type of the subject is equal to t, $t_s = t$.
From the initial KB $\mathcal{K}$ we can select the subset of the triples $\mathcal{K}_t$ so that all the triples in $\mathcal{K}$ are related to the *type* t. For instance, examples of triples related to a type *movie* are all the triples that describe the relationships where the subject entity is a *movie*.

The set of all possible *types* in the KB is $T = \{t_1, t_2, \dots\}$ and each entity is associated to a single [1] *Type* $t_e \in T$.

5.4.2 Automatic Responder with Knowledge Base

Correct vectors that extract data published in a KB are likely to extract values that match with subjects and objects of triples of a specific predicate in the KB. In this section we exploit this intuition by scoring extraction rules considering the presence of a KB. The system requires in input the set of templated pages U and two set of rules, $\mathcal{R}_a$ and $\mathcal{R}_b$. $\mathcal{R}_a$ are the candidate rules for the main attribute, the attribute that identifies the entity, e.g. the name of a person or the title of a book, and $\mathcal{R}_b$ are the candidate rules for the secondary attributes, e.g. the age of a person or the rating of a book.

[1] For the sake of the presentation we make the assumption that each entity has a single *type*

Compared to *Types* and *LFEQs* the solution based on *Knowledge Base* infers the best extraction rules considering a pair of attributes. This is motivated by the fact that the system matches triples and it is more reliable to match both subject and object instead of matching a single entity in the triples.

We can divide our approach in two steps: (i) discover the best type t in the KB that contains triples related to the considered pages, and (ii) find the best pair of rules that matches the triples in KB of the type t.

We consider a type discovery step for a performance issues. In fact, the dimension of the KB can be huge, thus searching among all the triples in the knowledge base is an expensive task. Our type discovery exploits the presence of an interface in the KB so that given a string value, the interface returns all the types in the KB associated to the string value, i.e. for all the entities e, it selects t_e if v_e is matched with input string.

Type Discovery The first step of our algorithm is to find the type related to the attributes that we want to extract, our goal is to find a *type* τ and select all the triples of the given type K_τ.

τ is the *type* that maximizes a scoring function; this function takes into account the extracted values and all the *Knowledge Base*.

$$argmax_{\{t_i \in T\}} scoreType(K, R_a(U), R_b(U), t_i)$$

The scoring function is described as follow:

$$scoreType(K, R_a(U), R_b(U), t_i) = \sum_{p \in U} \begin{cases} 1 & \text{iff } \exists < s, p, o > \in K | t_i = t_s, v_s \in R_a(p), v_o \in R_b(p); \\ 0 & \text{, otherwise;} \end{cases}$$

scoreType counts the number of triples that match the type and the values extracted by the rules in $\mathcal{R}_a$ and in $\mathcal{R}_b$ with triples in the KB.

By finding the most likely *type* the system selects a subset of the original K where the *type* is compatible with the target *type* τ.

$$K_\tau = \{< s, p, o > \in K | t_s = \tau\}$$

Rules scoring Discovered the type we can finally define the algorithm to select the pair of rules that best matches the triples in the knowledge base.

Listing 5 Wrapper inference with a Knowledge Base

Input: a set of pages U
Input: a set of rules for the main attribute $\mathcal{R}_a = \{r_{a1}, r_{a_2}, \dots\}$
Input: a set of rules for a secondary attribute $\mathcal{R}_b = \{r_{b_1}, r_{b_2}, \dots\}$
Input: a knowledge base K_τ, set of $< s, p, o >$ where $\tau = t_s$.

Output: a pair $\{r_a, r_b\} | r_a \in \mathcal{R}_a, r_b \in \mathcal{R}_b$

1: **let** $k =< s, p, o > \in K_\tau | v_s \in R_a(U), v_o \in R_b(U)$;
2: **let** $P = \forall p \in < s, p, o > | v_s = v_{a0}^+, v_o = v_{b0}^+$
3: **let** $T = \mathcal{R}_a(U) \times \mathcal{R}_b(U) \times P$;
4: $r_a, r_b, p \leftarrow \text{argmax}_{\{r_i(U), r_j(U), p_z\} \in T}\, scorePair(r_i(U), r_j(U), p_z, k)$;
5: **return** r_a, r_b;

In Listing 5 we describe the algorithm. The set of pages U, two set of candidate rules $\mathcal{R}_a$ and $\mathcal{R}_b$, a set of triples K_τ are the input of the algorithm.

In line 1, the system pre-computes all the triples k that match the values extracted from the main attributes and the secondary attributes. In line 2, it computes all the predicates that match the first annotation of the attributes a and b. In this step we model the presence of different kind of relationships between two objects. In line 3, T is computed as the Cartesian product between the vectors in $\mathcal{R}_a(U)$, the vectors in $\mathcal{R}_b(U)$ and the set of predicates. In line 4, the system finds a r_a, r_b and p that maximizes the function *scorePair*.

scorePair is defined as follow:

$$scorePair(r_i(U), r_j(U), p_z, k) = \sum_{p \in U} \begin{cases} 1 & \text{iff } \exists e =< s_e, p_e, o_e > \in k | v_{s_e} = r_i(p), v_{o_e} = r_j(p), p_z = p_e; \\ 0 & \text{, otherwise;} \end{cases}$$

The function *scorePair* takes in input two vectors of extracted values, one by r_i and the second by r_j, a predicate p_z and scores the pair for each page $p \in U$. If there is a triple in k that contains the subject and the object extracted by the rules, then it is counted as a positive score.

5.5 PMI

Another interesting way to select the best extraction rule is by exploiting the correlation between the extracted values among different attributes. In this section we describe an automatic responder based on the measure of mutual association, the Pointwise Mutual Information (PMI). The system scores the mutual association between pairs of attributes

and finds the best extraction rules based on the frequency of the association of the extracted values on the Web.

In this Section, we will give an intuition of PMI 5.5.1 and we introduce our automatic responder based a Knowledge Base 5.5.2.

5.5.1 PMI intuition

The Pointwise Mutual Information (PMI) measures the association between two distinct events X and Y. On the specific, PMI measures the probability of the coincidence between X and Y with their individual probabilities [57]:

$$PMI(X,Y) = log(\frac{P(X,Y)}{P(X)P(Y)})$$

Higher is the PMI score and higher is the correlation between the two events. Nevertheless if the events X and Y are two independent events, then the joint probability is computed by $P(X)P(Y)$ and the score would be equal to $log(1)$. If the events are correlated and they tend to co-occur together, then the joint probability will be greater than $P(X)P(Y)$. The ratio between $P(X,Y)$ and $P(X)P(Y)$ measures the degree of correlation between the two events.

PMI has been adopted to measure the correlation between keywords in Information Retrieval [58]. The technique can be adopted to measure the correlation between two keywords on the Web. Given two keywords A and B, we can compute the correlation between A and B by first counting the number of documents in the Web that contain them by searching with a Search Engine first with A, then with B and then we search for documents that contain both A and B. We can compute the probabilities by normalizing the number of documents with the number of all the documents indexed by the Search Engine, but this is not required for our setting. In fact, as we will observe later, our goal is to select the rule that maximizes the scoring function, thus we can adopt the following function:

$$scoreMI(X,Y) = \frac{hits(X,Y)}{hits(X)hits(Y)}$$

The score $scoreMI$ is proportional to PMI, thus for our maximization task they are equivalent. $hits$ is the function that computes the number of results returned by the search engine [58].

5.5.2 Automatic Responder with PMI

Correct vectors from different attributes are likely to extract correlated values. For instance, if we are extracting the names of movies and the names of directors, these two values are likely to co-occur in the web, i.e. a document that contains the name of a movie is likely to contain also the name of the director. We exploit this intuition by scoring extraction rules with the function $scoreMI$ and programmatically querying a search engine[2] to count the number of documents in the Web.

Listing 6 Wrapper inference with PMI

Input: a set of pages U
Input: a set of rules for the main attribute $\mathcal{R}_a = \{r_{a1}, r_{a_2}, \dots\}$
Input: a set of rules for a secondary attribute $\mathcal{R}_b = \{r_{b_1}, r_{b_2}, \dots\}$

Output: a pair $\{r_a, r_b\} | r_a \in \mathcal{R}_a, r_b \in \mathcal{R}_b$

1: **let** $T = \mathcal{R}_a(U) \times \mathcal{R}_b(U)$;
2: $r_a, r_b \leftarrow \text{argmax}_{\{r_i(U), r_j(U)\} \in T}\, scorePair(r_i(U), r_j(U))$;
3: **return** r_a, r_b;

In Listing 6 we describe the algorithm. As for the solution based on the KB, the system requires in input the set of templated pages U and two set of rules, $\mathcal{R}_a$ rules for the main attribute and $\mathcal{R}_b$ rules for a secondary attribute.

The system computes T as the Cartesian product between the vectors in $\mathcal{R}_a(U)$ and the vectors in $\mathcal{R}_b(U)$ and then it selects the pair of vectors $r_a(U)$, $r_b(U)$ that maximizes the function $scorePair$.

The function $scorePair$ is described as follow:

$$scorePair(r_i(U), r_j(U)) = \sum_{p \in U} \frac{hits(r_i(p), r_j(p))}{hits(r_i(p)) hists(r_j(p))})$$

For each $p \in U$ the system apply the function $scoreMI$. Each call of $scoreMI$ executes three queries on a search engine: the value extracted by the candidate of the main attribute, the value extracted by the candidate of the secondary attribute and the values extracted by both candidate rules.

5.6 Experiments

In the previous sections we have described several automatic wrapper inference approaches that exploit different kind of information. But some questions arise: Can they

[2] Bing Search API

be combined together effectively to increase the quality, wrt, the approaches that consider each responder separately? Can they be guided with a small human supervision when the result is still uncertainty? Is there a quality loss when we move from human supervision to automatic responders? In this section we address these open questions. We compare the quality of the output wrappers by considering the standard metrics of P, R and F. We consider the standard deviation of the F measure σ_F to understand the reliability of the results.

We first present an outline of the experiments in 5.6.1 and we describe our evaluations in 5.6.2.

5.6.1 Experiments outline

For our evaluation we adopt the dataset 3.3 described in Chapter 3. We first compare the results obtained considering only automatic responders. We denote with: Types (T), LFEQ (L), Freebase (F), PMI (P) the automatic responders described previously in this chapter. Responders are used to answer MQ posed by ALF_η on our dataset. The initial η is statically set to 0.1 and the termination condition is set with $\lambda_p = 0.95$ and $\lambda_{MQ} = 10$. The answers from multiple automatic responders can be combined adopting ALFRED. We adopt ALFRED to estimate the responders' error rate and to evaluate the expected quality of the returned extraction rule. The termination threshold on ALFRED provides a metric to terminate the computation or to search for human workers to reduce the uncertainty, i.e. the system can trust or not the combined results.

In the first part of the evaluation, we compare the results considering only automatic responders. We evaluate responders singularly, combining them together and we compare ALFRED with two simple baselines. In the second part, we compare the results of the automatic responders with synthetic workers; synthetic workers simulate the answers provided by real workers. We denote with H the output provided by a synthetic human worker and with H_i the output of ALFRED when the initial number of workers is set to i.

5.6.2 Evaluation

Automatic Responders We first evaluate the quality of the output wrapper considering different responders by adopting ALF_η.
In Table 5.1 we provide the results considering a single responder at a time. Overall, responders based on Types and LFEQ are the most accurate, with an F greater than 0.9. Types and LFEQ are domain independent, thus on average the results on several

Resp.	P	R	F	σ_F
T	0.91	0.93	0.91	22%
L	0.93	0.91	0.90	20%
F	0.88	0.88	0.88	26%
P	0.88	0.91	0.89	27%

TABLE 5.1: Average quality based on different responders with a probability threshold of 0.95

domains are better than domain dependent approaches. Freebase and PMI are domain specific and, in fact, Freebase provides the worst results. This is motivated by the fact that some attributes are not found in the Knowledge Base, common examples are: subjective attributes, e.g. the number of the users that provides a review to a movie or a product, site specific attributes, e.g the number of pages of the edition of the book and so on. The PMI responder suffers in part of the issues of Freebase, in fact some attributes are less likely to be found on the web because they are site specific, thus the quality is affected by the presence of unique values on the website. Another interesting result is the standard deviation, in fact, only adopting a single automatic responder we can obtain a high F, but the quality is often not predictable with a high standard deviation ($\sigma_F > 0.2$). The approach based on LFEQ obtains less uncertain, but the standard deviation is still high $\sigma_F = 0.2$.

Resp.	η	σ_η
T	0.13	24%
L	0.17	25%
F	0.17	28%
P	0.22	27%

TABLE 5.2: Average η and σ_η of the responders

Table 5.2 confirms our previous observation. The average η of automatic responders is higher than the η observed with humans. There is a small difference if we consider the average η, it increases between 0.03 and 0.12. A much more interesting comparison is on the standard deviation. With automatic responders we observe a much higher σ_η that increases between 13% and 16% wrt the humans $\sigma_\eta = 11\%$. This is an expected result, in fact automatic approaches tends to work really well in some cases and fail completely in others.

In Table 5.3 we compare the results combining the responders together considering ALFRED and two baselines, the baselines describe different approaches to combine the answers provided by the responders: *Majority Voting* (*MV*) selects the rule that is returned by more responders, ALF_η returns the rule that is more likely considering a training sequence generated by the union of the training sequences of the responders.

	all attributes				threshold				
Alg.	P	R	F	σ_F	P	R	F	σ_F	%
ALF$_\eta$	0.91	0.92	0.91	22%	0.95	0.98	0.96	13%	68%
MV	0.93	0.95	0.94	18%	0.93	0.95	0.94	18%	97%
ALFRED	0.94	0.95	0.95	16%	0.97	0.99	0.98	4%	71%

TABLE 5.3: Comparison between ALFRED and two baselines, Majority Voting and ALF$_\eta$

We say that the system "trusts" the output extraction rule: *MV* the rule is returned at least by half of the responders; ALF$_\eta$ the probability of the most likely rule has to be greater than λ_r. Overall ALFRED achieves the best F and with the lowest uncertainty σ_F. If we compare ALFRED with ALF$_\eta$ we can observe the difference between estimating the responders' error rate and a simple approach that combines the answers, in ALF$_\eta$ erroneous responses are weighted too much, thus the system is not able to combine the answers efficiently. A simple *MV* achieves good results with an F and σ_F close to ALFRED. If we consider the termination threshold, we observe that *MV* obtains the same results as the average and it terminates in 97% of the cases, thus *MV* is less sensitive to erroneous majorities. ALF$_\eta$ achieves better results wrt *MV*, but the standard deviation is still high. ALFRED obtains the best results with the lowest standard deviation of 4% and an $F = 0.98$.

	all attributes				threshold				
Resp.	P	R	F	σ_F	P	R	F	σ_F	%
LP	0.91	0.90	0.89	25%	0.95	0.98	0.96	14%	59%
TF	0.92	0.91	0.91	21%	0.92	0.95	0.92	21%	46%
PF	0.87	0.88	0.87	30%	0.91	0.95	0.92	23%	49%
TL	0.92	0.93	0.92	20%	0.93	0.97	0.95	15%	67%
TP	0.92	0.93	0.91	23%	0.95	0.98	0.96	4%	65%
LF	0.92	0.90	0.91	21%	0.93	0.96	0.93	13%	48%
LPF	0.91	0.91	0.90	24%	0.95	0.98	0.96	13%	68%
TLF	0.93	0.93	0.92	21%	0.94	0.96	0.95	16%	75%
TPF	0.91	0.93	0.93	18%	0.95	0.99	0.97	11%	69%
TLP	0.92	0.93	0.93	20%	0.96	0.99	0.97	8%	73%
TLFP	0.94	0.95	0.95	16%	0.97	0.99	0.98	4%	71%

TABLE 5.4: Average quality based on different combinations of responders on all the attributes and considering a probability threshold of 0.95

In Table 5.4 we compare different combination of responders with ALFRED and we check the quality of the output when a threshold over the combined answers is met. Answers are collected by running ALF$_\eta$ with each responder separately. The answers are then collected and combined by running ALFRED. The configuration of the evaluation is denoted by the symbol of each responder, e.g. TL denotes ALFRED executed on answers

provided by two responders Types and LFEQ. Overall, the combination of multiple responders increases quality of the output. Combining all the responders (TLFP) provides on average a high F measure of 0.94, with a lower standard deviation of 0.17. A really interesting result is when we consider the probability threshold set by ALFRED. In fact, the system is able to understand when we should trust the results and when additional feedback is required to reduce the uncertainty. In Table 5.4 we consider a threshold of $\lambda_p = 0.95$; in 71% of the attributes we obtained an $F = 0.99$ with a standard deviation of 0.04, i.e. almost perfect results without human feedback. In the remaining 29% of the attributes the system is not certain of the results, but if we still consider the most likely rule for these attribute on average we obtain $F = 0.94$.
If we consider configurations with two responders: TP is the best combination in terms of F and σ_F when $\lambda_r > 0.95$, TL obtains the best coverage with the algorithm that terminates in 67%. Considering configurations with three responders: TLP is the best solution with a high $F = 0.97$ and the lowest standard deviation of 8%, TLF terminates on 75% of the attributes but the F is the lowest wrt other combination of three responders. Overall TLFP obtains the best results in terms of average F and has the lowest standard deviation. F is close to 1 and the standard deviation is 4%.

Humans vs Automatic Responders We previously observed that ALFRED understands when additional feedback is required to reduce the uncertainty. In those cases, humans can be involved to answers the *MQ* provided by ALF_η. The collected answers are combined together with answers provided by automatic responders with ALFRED. As described in the previous chapters, ALFRED can dynamically enroll additional workers, until the termination condition is met (λ_p is greater than a threshold).

Resp.	P	R	F	σ_F	#MQ	$\lvert W\rvert$	max $\lvert W\rvert$
TLFP	0.94	0.94	0.94	17%	0	0	0
H_1	0.99	0.98	0.98	11%	9.19	2.06	6
H_2	1.00	0.98	0.99	6%	13.31	2.38	7
TLFP+H_1	0.97	0.99	0.98	7%	2.30	0.36	4

TABLE 5.5: Average quality based on different combinations of responders on all the attributes and considering a probability threshold of 0.95

In Table 5.5 we compare the results considering answers from synthetic human workers and automatic responders. We compare different configurations: TLFP provides results based only on automatic responders where all the responders are combined and the most likely rule is returned anyway, i.e. even when the termination condition is not met; H_1 runs ALFRED with an initial set up of one worker; H_2 runs ALFRED with 2 workers in the initial setup; TLFP+H_1 describes an hybrid approach where automatic responders are adopted to infer the extraction rule, and only when the termination condition is not

reached by ALFRED, humans are involved.
A solution based only on automatic responders (TLPF) that does not consider the quality threshold, obtains the worst F measure with $F = 0.94$ and a high standard deviation of 0.17. No human work is required, but the results are not predictable and the quality can drop.
Enrolling humans to infer wrappers drastically increases the quality of the output, when ALFRED engages a single worker at the beginning of the inference (H_1) the F is 0.98. Enrolling multiple workers at the beginning of the task reduces the uncertainty with an $F = 0.99$ and a -0.05 in standard deviation. The reason for the high standard deviation in H_1 is that workers can make mistakes and enrolling a single worker on a task does not give enough evidence of her mistakes. Engaging multiple workers at the beginning of the task is the best way to address this issue, but the price to pay is on the number of MQ required for each attribute, on average is 13.31. ALFRED dynamically enrolls multiple workers on the same task and on average 2.38 workers are required for each task in H_2. TLFP+H_1 addresses many issues related to a complete human based approach. The quality of the output is comparable to H_2 with a small loss in precision -0.03, but the standard deviation is lower than H_1 and it is close to H_2. The hybrid approach obtains almost perfect results with just a fraction of the MQ required by H_2. In fact, on average the number of tasks required for each attribute is 0.36 and the number of MQ is 2.3. Notice that even with the hybrid approach, ALFRED dynamically engages multiple workers when some attributes are uncertain, in fact for some attributes 4 workers are involved before terminating the algorithm. If we consider the number of workers, 71% of the tasks are completed with no human intervention, 25% with a single worker and 6.07 *MQ*, and only 4% of the tasks require more than 1 worker with an average *MQ* of 20.00.

5.7 Conclusions

In this chapter we presented several independent automatic wrapper inference systems: *Types* exploits the similarity between the types of the extracted values (repeating emails, or URLs); *LFEQs* exploits the equivalence classes in the HTML templates; *Knowledge Base* exploits the presence of a knowledge base that matches extracted values and *PMI* exploits the correlation between the values extracted in different attributes for the same entity. We compared the responders, we evaluated each responder separately and we combined them with ALFRED to define a new automatic wrapper inference system. We show that the combination of all the responders increases the average F and reduces its standard deviation. We also compared a crowd based solution with the automatic approach and we defined a hybrid solution that starts by considering only automatic

responders and when it is required humans are engaged. The results are promising: around 70% of the attributes are automatically wrapped with the quality close to the crowd based solution, ALFRED dynamically engages humans workers only on those cases where automatic responders fail to reduce the uncertainty, the hybrid approach learns accurate wrappers with a fraction (17%) of the *MQ* required by the crowd based solution. The loss is very limited, a drop of 0.01 in F.

Chapter 6

Discovery and Extraction of Product Specifications

In the previous chapters we described a hybrid wrapper generation approach that dynamically understands if human feedback is required. We optimized the solution to reduce the cost related to human work and we controlled the quality of the output by using redundancy. An open issue is the discovery of the input pages to wrap, i.e. the pages that share a common template. In fact, to define an end-to-end pipeline that scales to the Web other challenges have to be addressed: the discovery of relevant websites that are likely to contain templated pages, the crawling inside a website of the templated pages and finally the extraction of the output values.

In this chapter we address these issues. We consider a case study on specifications published on the Web and we describe techniques to discover and collect the attribute-value pairs embedded in the specifications. We focus on product specifications because: (i) specifications are widely available in many websites (ii) specifications are example of templated pages; (iii) they are published in many domains for people (e.g., entertainers, politicians), products (e.g., cameras, computers), organizations (e.g., restaurants, hospitals), etc; (iv) specifications are characterized by a strong regularity and a potentially big number of attribute-value pair in each specifications.

A recent study [59] showed that specifications (set of attribute-value pairs for a single entity) are among the most common forms of structured data available on the web, e.g., infoboxes in wikipedia pages. Figure 6.1 shows example product specifications from different websites. Specifications are typically represented as vertical tables [60] or HTML lists, though only a small fraction of vertical tables and HTML lists are specifications. There has been considerable interest in using specifications that can be collected from Web data sources for a variety of applications, such as data integration,

faceted search and question answering [61]. For example, many organizations today look to the Web to augment their internal databases with specifications in specific domains of interest.

A possible strategy to collect specifications from the Web is to run a general crawler, and then extract the data from the gathered pages. This approach has been investigated in the literature (see, e.g., [62]), and has its limitations: very few groups have access to up-to-date Web crawls, and initiating such crawls is extremely resource intensive. An alternative is to use a publicly-accessible snapshot of the Web, e.g. Common Crawl [1]. We examined this possibility and verified that on Common Crawl many pages are out-of-date and some small websites are not even indexed. To avoid all these issues, we present in this chapter a scalable focused-crawling technique that only looks for specifications in a domain of interest. Domains are represented by categories of products (e.g., cameras, computers). We focus this study on product specifications for three reasons. First, there is much interest in using product specifications for comparison shopping, faceted search, etc. [63, 64]. Second, product specifications are available from a large variety of Web data sources (e.g., e-commerce retailers, local stores with an online presence). Third, despite their availability, efficiently obtaining a large set of high-quality product specifications in a given category comes with many challenges.

Efficiency: The number of data sources (websites) for any given product category can be in the thousands, but these sources are spread out on the web. Similarly, only a small fraction of pages in a relevant website have product specifications.

This problem of sparsity makes it challenging to *efficiently* obtain a large set of product specifications.

Quality: There is considerable variability among product specifications in terms of their attributes, sizes, format and content. Further, even though product specifications are typically represented using HTML tables and lists, only a small fraction of HTML tables and lists are product specifications.

This problem of identifiability makes it challenging to obtain a large set of *high-quality* product specifications.

To effectively address the challenges of efficiency and quality, we propose an end-to-end system, DEXTER, that consistently uses the principles of *vote-filter-iterate*. From a small collection of seed Web pages and product specifications in a given category, DEXTER iteratively obtains a large set of product specifications. More specifically, in each iteration, DEXTER uses voting and filtering to prune potentially irrelevant websites

[1] http://commoncrawl.org

and Web pages in a site, to reduce the noise introduced in the pipeline, and to efficiently obtain a large number of high-quality product specifications.

In this chapter, we make the following contributions: (i) an original approach to efficiently discover websites that contain product specifications; (ii) an adaptation of an existing crawling technique [45] designed for forums to work for product websites; (iii) an original technique to find and extract attribute-value pairs from product specifications; (iv) an end-to-end system to efficiently and accurately build a database of product specifications.

In this chapter we develop a new extraction technique wrt the techniques presented in the previous chapters. The motivation is that specifications are characterized by the presence of a big number of attribute/value pairs, thus making inefficient a per attribute cost model. A possible technique to address this issue is to exploit the regularity that characterize the portion of the HTML that contains the specifications, thus instead of learning an extraction rule for each attribute, we describe a technique to recognize the specifications area.

We have performed an extensive experimental evaluation with five different product categories on the web. Our results show that (1) our website discovery and in-site crawling strategies efficiently identified over 2,719 websites and 1M HTML pages that contain product specification pages in the five product categories; (2) our specification detector obtains a high value of F-measure (close to 0.9) over a large variety of product specifications; and (3) our wrapper (attribute-value extractor) gets very high values of precision and recall.

We compared our dataset with Common Crawl: 68% of the sources discovered by DEXTER are not present in Common Crawl, only in 20% of the websites DEXTER discovered fewer pages, but product pages are just a fraction of all the discovered pages. In fact, on a sample set of 12 websites where Common Crawl indexed more pages, 99.2% of the pages were non-specification pages.

The rest of this chapter is organized as follows. Section 6.1 defines our problem and presents an overview of DEXTER. In Section 6.2, we introduce our strategies for website discovery and in Section 6.2 we describes the in-site crawling to locate product pages. Section 6.4 describes the approach we used in generic specification detection, Section 6.5 presents our approach to extract attribute-value pairs. In Section 6.6, we present our extensive experimental evaluation results. We discuss some related works in Section 6.7, and we summarize in Section 6.8.

Features	
Color	Blue, Gold, White
Metal	White Gold
Stone	Diamond, Gemstone
Diamond Color	White H-I

Color Category	White
Operating System	Apple iOS
Wireless Capability	Wi-Fi
UPC	885909575176

FIGURE 6.1: Examples of specifications of different products: (Left) ring specification (www.overstock.com), (Right) tablet specification (www.bestbuy.com)

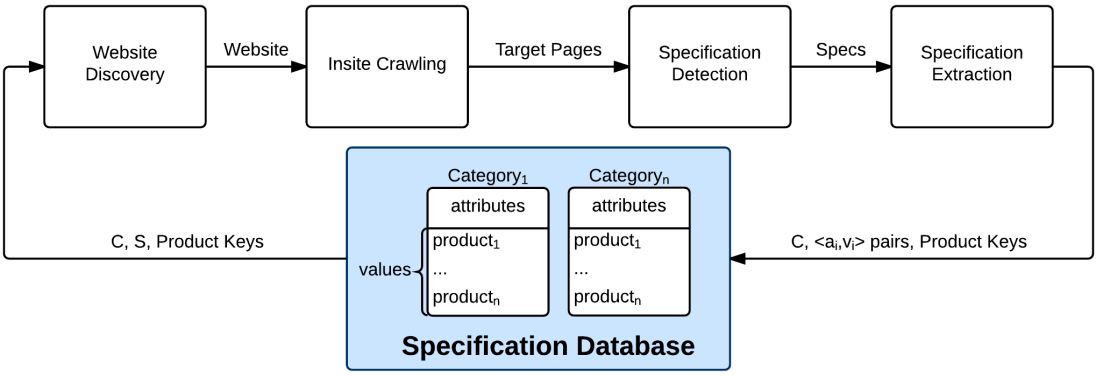


FIGURE 6.2: Architecture of DEXTER, composed by the Sites Discovery, the In Site Crawling, the Specification Detector (detection), and Generic Wrapper (extraction).

6.1 Overview

We define a product specification as follows.

Definition 6.1 (Product Specification)**.** A product specification SP is a set of attribute-value pairs $< a_i, v_i >$. The attributes (partially) form a schema for the product category C which SP belongs to.

The goal of this work is to build a database of products specifications. More formally, we state the problem as:

Definition 6.2 (Problem Definition)**.** Given seed products P and product websites S in a specific category C, we aim to efficiently crawl a comprehensive database of products specifications in C.

To deal with this problem, we propose DEXTER. Initially, from seed products and product websites, DEXTER locates product websites such as shopping websites (Website Discovery). Here we assume specifications are mostly contained in such sites. Second,

DEXTER crawls those websites to collect product specification pages (In-site Crawling). From those pages, DEXTER detects the HTML portion of the pages that contains the specification (Specification Detection) and, finally extracts the attribute-value pairs from the specification (Specification Extraction). Figure 6.2 shows the architecture of DEXTER. In the rest of this section we provide further details about each step and their main challenges.

Website Discovery: The goal of Website Discovery is to locate candidate product websites. Since product websites are sparsely distributed on the Web, DEXTER uses two strategies to deal with that: (i) it queries a search engine with known product keys, and (ii) identifies hubs that contain links to known product websites. Voting is used to generate a ranking of the candidate websites to select websites for further exploration. Since some of the resulting websites might not be relevant, we implemented a product website classifier to quickly filter out irrelevant websites. Iteration is subsequently used to obtain a large number of relevant websites.

In-site Crawling: Within a potentially relevant product website, we aim to efficiently locate the product specification pages. For that, we use a number of classifiers to discover product category entry pages and index pages, filtering out Web pages on the website that are unlikely to lead to product specification pages. Voting is used to score the links from product specification pages in a website, discovered using a search engine, to aggregate the classifier scores for identifying promising category entry pages.

Specification Detection: Once product specification pages are identified, the goal is to detect the HTML fragments on those pages that correspond to specifications. For that, we propose a product category detector, to avoid labeling data for each considered category. The specification detection classifier looks at HTML fragments (e.g., tables and lists) in these pages, and makes use of various structural features such as the number of links, items, text size, etc., that distinguish high-quality specifications from non-specifications.

Specification Extraction: The final step in the DEXTER pipeline is to extract attribute-value pairs in the detected specifications. Since a large number of specifications can be detected, it is important for attribute-value pairs to be extracted efficiently. For that, we implemented two different strategies: (1) a heuristic lightweight approach that takes into consideration HTML structural commonalities between specifications across sites, and (2) a hybrid method that combines the approach of [65] of inferring extraction patterns based on noisy annotations with our heuristic approach.

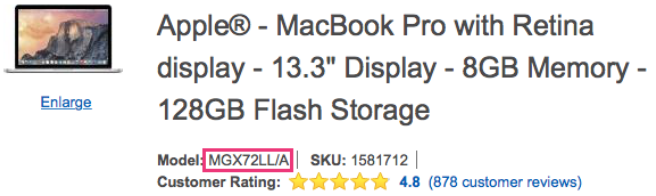


FIGURE 6.3: A product detail page with the product key.

6.2 Discovery

Product pages are usually located in online stores. Under this assumption, the first task of DEXTER is to find online store sites to collect product pages from them. Since these websites are sparsely distributed on the Web, the main challenges are: to efficiently find online stores while avoiding visiting unproductive regions on the Web, and to discover a comprehensive catalog of them with a high quality. To achieve these goals, we implemented four different strategies: (1) Search: the crawler issues queries based on known products to a search engine in order to discover other websites that publish information about these products; (2) Backlink: from known relevant sites, the crawler explores their backlinks to find other relevant sites; (3-4) Merge: the system defines two rankings strategies made by a combination of the discovered websites adopting Union and Intersection. We give further details about these strategies in the rest of this section.

Search In our domain, we expect that multiple websites publish specifications of the same product. Taking advantage of this high redundancy, searching for known products on a search engine would return pages of these products in different sites. Notice that the search engine can return non-product-specification websites, e.g. a forum with a discussion of a product. To efficiently discover relevant websites without penalizing the overall recall we exploit the previous observations, DEXTER searches for multiple products and provides an ordered ranking of the returned websites from the search engine considering the number of times a website is present in the results, i.e., a forum or other non specification websites are less likely to have pages in the results for many products. Based on the ranking, for each iteration, DEXTER selects the top K websites.

The idea of using search engines to help collect pages has been previously explored [42, 43, 62]. For instance [62] analyzed the distribution of entities for a set of domains by searching for entities on the Web using some unique identifiers, e.g. restaurant phone numbers. Similarly, we extract a representation of given products to discover new product sites that contain the products.

Examples of the used representation are the product identifiers like the model name that often is published in the page that contains the product specifications. In Figure 6.3 we show the product key extracted and used for the search step.

Our method works as follows. Given a set of product keys $\mathcal{K}$ obtained from seed websites S, the crawler uses a search engine API to discover new websites S' that publish information for products in $\mathcal{K}$. More specifically, S' is built considering all the websites returned by the search engine over all queries in $\mathcal{K}$. S' is then ranked considering the score according to the following equation:

$$sSearch(\mathcal{K}, s_j) = \frac{\sum_{k_i \in \mathcal{K}} search(k_i, s_j)}{|\mathcal{K}|} \tag{6.1}$$

The *search* function is a binary function that scores a website $s_j = 1$ if and only if the website is returned by the search engine using as query the keyword k_i:

$$search(k_i, s_j) = \begin{cases} 1 & \text{if } s_j \text{ returned searching for } k_i \\ 0 & \text{otherwise} \end{cases} \tag{6.2}$$

Using the function $sSearch$ for all the websites $s_j \in S'$, we generate an ordered ranking R'.

Based on the ranked list of websites we can adopt a threshold to select the top websites and consider them for further steps in the pipeline.

The main limitations of the search approach are the restrictions usually imposed by search engine APIs as e.g. number of results per query, number of results per user and total number of queries in an interval of time. This is a significant issue if the goal is to collect an unbounded number of product websites. Our next approach tries to deal with this problem of scalability since it is less dependent of search engines.

Backlink An interesting source for finding relevant websites are pages that point to multiple sites, so-called hub pages. As an example, previous work [66] uses backlinks of multilingual websites to find hub pages that point to other multilingual sites, and then restricts the crawler to the Web region defined by the bipartite graph composed by the pages pointed by the backlinks of relevant sites and the outlinks of these hub pages. A backlink is just a reverse link so that from the initial website we can discover hubs that point to it.

Efficiency, recall and quality are challenging goals. In fact, backlinks can lead to non-relevant hubs, thus leading to non-relevant websites. Common examples are generic

hubs that point to multiple websites, like popular websites in a country, websites where the name of the domain starts with an 'A' and so on.

We adopted a similar *vote-filter-iterative* strategy to locate product websites and to address our challenges. Non-relevant hubs are less likely to point to many relevant websites, while relevant websites are more likely to be pointed by many relevant hubs. Based on this intuition using backlinks, we score hubs that point to many relevant websites. From these hubs, we compute an ordered ranking of the new websites pointed by the hubs. As with search, we generate an ordered ranking and for each iteration we select the top K websites.

The approach works as follows. Given the initial set of websites S, we want to discover a new set of websites S'' and an ordered ranking R'' of S'' so that S'' are all pointed by hubs discovered from S. To provide a ranking and prune the enormous number of websites and hubs discovered by backlinks we first search for the hubs H using backlinks for each website $s_i \in S$. Each hub $h_j \in H$ is scored following this formulation:

$$sHub(S, h_j) = \frac{\sum_{s_i \in S} hub(s_i, h_j)}{|S|} \tag{6.3}$$

The *hub* function is a binary function that scores a hub h_j with 1 if s_i has a backlink to h_j.

$$hub(s_i, h_j) = \begin{cases} 1 & \text{if } h_j \text{ in backlink for } s_i \\ 0 & \text{otherwise} \end{cases} \tag{6.4}$$

To improve the performance, we prune those hubs that are pointed only by a single website. We expect that if S is big enough, hubs discovered only by one website are not likely to lead to interesting websites. From the hubs H, we follow the forward links to discover new websites S'', for each new website s_j we score the website as the weighted average of all the hubs that point to s_j:

$$sForward(H, s_j) = \frac{\sum_{h_i \in H} forward(h_i, s_j) * sHub(S, h_i)}{\sum_{h \in H} sHub(S, h_j)} \tag{6.5}$$

The $forward$ function describes whenever the website s_j is pointed by the hub h_i.

$$forward(h_i, s_j) = \begin{cases} 1 & \text{if } s_j \text{ in forward links of } h_i \\ 0 & \text{otherwise} \end{cases} \tag{6.6}$$

Using the function $sForward$ for all the websites $s_j \in S''$ we generate a second ordered ranking R''. We adopt the same threshold of the search approach to select the top K websites for further processing in the pipeline.

Merge An interesting observation is that the backlink and the search are two independent approaches and they can be combined to define a new ranking algorithm.

In this work, we consider two merging algorithms: the first one based on the union and the second based on the intersection of the two generated rankings. The intuition behind the union is that if the two rankings generate two disjoint but good ordered sets, considering the union of the top results from each ranking can provide better results. For the intersection we expect that if there is an overlap among the results from the two distinct rankings, a good website is likely to be in the overlap and ranked better by the new ordering.

Given two rankings R' and R'', each value in R^x is made of a pair (s_i, p_i^x) where $s_i \in S' \cup S''$ and p_i^x is the position assigned to s_i by the ranking R^x.

We score a site s_i for a ranking R^x considering the following formulation:

$$rScore(s_i, R^x) = \frac{1}{|R^x|}||R^x| + 1 - p_i^x|$$

i.e, we score a source s_i for a ranking R^x considering the position of the source inside the ranking.

We define the score of the union as follow:

$$sUnion(s_i) = max(rScore(s_i, R'), rScore(s_i, R''))$$

From this function, we generate a new ranking R_{union} so that the score of the website s_i is defined by $sUnion$.

We define the intersection as the harmonic mean among the rankings as follow:

$$sIntersection(s_i) = 2 * \frac{rScore(s_i, R') * rScore(s_i, R'')}{rScore(s_i, R') + rScore(s_i, R'')}$$

From $sIntersection$ we define $R_{intersection}$.

Notice that we do not merge the rankings considering the scores provided by the source discovery strategies. An issue that we observed is that often the scores are not balanced because one ranking often leads to scores close to 1 while the second has lower scores. A naive merging could penalize one of the rankings.

Since there are many non-relevant sites returned by the voting and iterating steps, the final step in the Website Discovery is to efficiently detect product websites. It is based on a classifier trained to recognize if a website is a product website considering the features in the root home page (Home Page Classifier). More specifically, we trained the

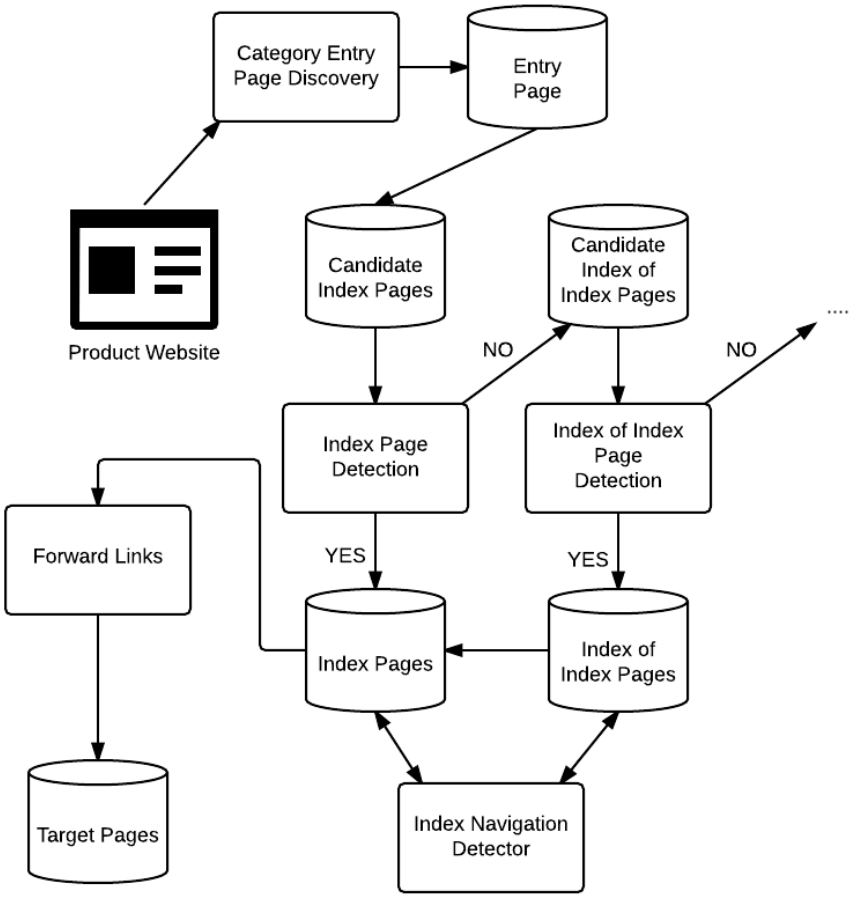


FIGURE 6.4: The pipeline to crawl a new website for target pages.

classifier to recognize websites that publish specifications of products by just looking at their home page. We trained the classifier with the anchor text of the links in the home page[2]. The words inside the anchor text are good features because the home page, as the root page, provides links to multiple categories, and links that leads to the same category are likely to be the same, e.g. for TV common anchor texts are TV, Television, Home, Living Room etc.

6.3 Crawling

Having discovered a new product website, the following step in the pipeline is to crawl the website and discover product specification pages. To avoid visiting unproductive regions of a website, it is important to have a strategy that collects as many product pages as possible, visiting as few pages as possible.

[2]We use 50 relevant product websites and 50 non-relevant websites as training data for all the categories.

DEXTER's Insite Crawling is inspired by [45], which was focused on forum crawling. We use a similar approach to crawl generic shopping websites. The main assumption is that, similar to forum sites, product-based sites have a well-defined structure consisting of an entry page to the product content in a given category, index pages that point to product pages and, finally, the product pages themselves. Based on that, we implemented the following strategy. Figure 6.4 shows an overview of our approach. First, the Insite Crawling discovers the entry page related to a category (Entry Page Discovery). Normally, shopping websites organize their catalogs in categories and subcategories. This is expected because the website administrator wants to improve the navigation experience of the customer inside the website. Next, the crawler performs the Index Page Detection. The category entry page leads to index pages or nested index pages. An index page is a structured page inside the website that lets the customer search, filter and select the product that she wants to purchase. The Index Navigation Detector discovers the pagination structure inside an index or a nested index and finally, a classifier is trained to detect pages of a given product. The category that we assign to the crawled pages is inferred considering multiple steps: the keywords used to find the Web site are category specific, the entry page is category specific, the features that we adopt to recognize the target pages are specific to the category. The category is automatically assigned to the target pages based on the category of the crawling step. In the rest of this section, we provide further details of the Entry Page Discovery and the Index Page Detection.

Entry Page Discovery To discover the entry page of a given category, we defined three distinct strategies:

- **From the home page**: the first approach starts from the site's home page and uses the Entry Page Classifier to detect links to the entry page of a given category. For that, it uses as features words in the anchor texts. In this strategy we crawl the website from the home page and we score candidate entry pages as the product of the score returned by the Entry Page Classifier on the anchor text of each crawled link.
- **From the target pages**: The second approach follows the intuition that often in the product page we have references to the category that the product belongs to. We use the confidence score from the Entry Page Classifier to score the links of the given page. This is repeated for every candidate target page and the final score is the average score obtained from each page.
- **By search**: The last approach uses a search engine to directly find the entry page. We search within the website using the category name as query terms. We score the candidate results by considering the ranking of the search engine.

The three strategies return independent scores, and DEXTER considers the webpage that gets the best score from all three scores using a harmonic mean formulation.

Index Page To recognize generic index pages, we make the assumption that the anchor text that points to product pages have some regularity. Under this assumption, we trained several classifiers, one for each category, using as features the words in the anchor text of the link that points to the target pages. Since index pages usually contain group of links to product pages, to improve classification accuracy, we group links and then score the groups as the average score of the links that compose the group. To group links, we make the observation that regions in the index pages that points to target pages have the same layout and presentation properties. This motivates the usage of link collections [67], a group of links grouped by an extraction rule (XPath), like lists. The same technique is applied to recognize a nested index structure. We recognize a nested index that points to multiple index pages by scoring each page as index page. We empirically observed that a complex nested structure is unlikely and that most of the cases are managed by a two level structure. We say that a page is an index page if the average score is greater than a given threshold that we set empirically.

6.4 Features for Specification Detection

The process of automatically extracting specifications in many websites for different categories is a challenging task. For each website, the template based HTML pages are characterized by some local variability. This variability is related to the script used to generate them. The script is site specific, thus to accurately extract the data, we have to generate a specific wrapper for each website. A possible way to address this issue is to recognize the variability in a website by using domain knowledge to recognize a specific category, like in [9]. DEXTER adopts an alternate approach, which does not require any domain knowledge and is not specific to few categories. The detection addresses the local variability inside the websites, recognizing product specifications with a Machine Learning based solution. The extraction is then applied to a regular portion of the HTML page, thus simplifying the extraction task.

Previous approaches have been proposed to detect tables/lists that consist of structured data [68, 69]. More specifically, Wang and Hu [69] use machine learning algorithms to detect tables with relational information, and Gupta and Sarawagi [68] proposed some heuristics to remove useless and navigational lists. Similar to [69], we also use machine learning techniques but our task is not to detect relational table on the Web, but to detect specifications.

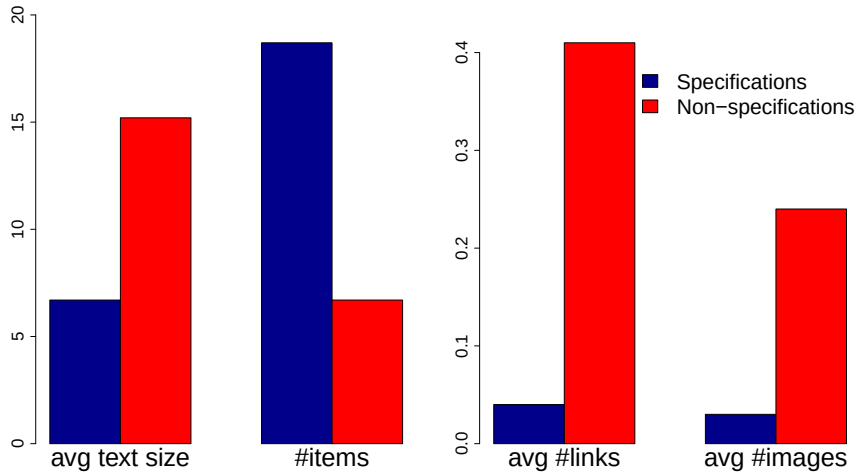


FIGURE 6.5: Average number of links and images, number of items and average text length per item for specifications and non-specifications.

Specifications can be contained in different HTML structures. But they are primarily found within tables and lists (*ul* tag). We empirically verified that by manually inspecting 301 specifications from a great variety of products. Among them, 62% were inside tables whereas 31% inside lists. The remaining 7% were in miscellaneous structures (e.g. HTML div tag). By also annotating tables and lists that are not specifications (304 instances), we observed that some structural characteristics of specifications can be useful as a good indicator of whether a table or list is a specification or not, independent of product or site. To illustrate that, we present in Figure 6.5 features such as average number of links and images, number of items and average text length per item from our sample set containing both specifications and non-specifications. As one can see from these numbers, specifications contain far fewer links and images than non-specifications, and more items and smaller texts. Figure 6.6 presents concrete examples of tables and lists that do not contain product specifications. As one can see, they present many links and larger text size compared with the specifications presented in Figure 6.1. Other features that we use to differentiate specifications from non-specifications include average node depth of the items and its standard deviation, standard deviation of the text size, overall frequency of the word "specification", HTML type and average number of the pattern of two upper cases followed by a number (e.g., "Weight Max 40 pounds").

As we present in Section 6.6, DEXTER not only is effective to detect specifications, but it is also able to pinpoint them on the page because it classifies not the entire page but the tables or lists which contain specifications. As we show in Section 6.5, this is very helpful for the extraction of the specification's set of attribute-value pairs from these HTML fragments.

- USB MIDI Cable Converter PC to Music Keyboard Window Win Vista XP, Mac OS
  3.7 out of 5 stars (406)
  $3.77

- HDE USB MIDI Cable Converter PC to Music Keyboar
  3.9 out of 5 stars (162)
  $5.95

		price
0509703PART/R3	Zipp 900 Clincher Rear Wheel	$1,849.99
0508068PART/R3	Easton EC90 TT 90mm Tubular Wheelset	$1,799.99
0508654/R3	Shimano Dura-Ace WH-7900-C50-TU Tubular Wheelset	$2,799.99
1599957/R3C	Wipperman 10S0 10/speed Chain (Shimano Compatible)	$29.99
GISEC0PART/R3C	Giro Section Helmet '10	$39.99
1729849/R3C	Kryptonite R4 Retractable Combo Cable Lock	$16.99
SHM161GPART/R3C	Shimano SH-M161G MTB Cycling Shoe	$109.99

ASIN	B00QVWJ7QE
Customer Reviews	★★★★☆ 11 reviews 4.1 out of 5 stars
Best Sellers Rank	#293 in Computers & Accessories (See top 100)
Shipping Weight	2.1 pounds (View shipping rates and policies)
Date First Available	December 11, 2014

FIGURE 6.6: Examples of non-specifications.

6.5 Specification Extraction

The final tasks of DEXTER are: the extraction of the attribute-value pairs from the HTML fragments provided by the generic specification detector (see Figure 6.2) and the extraction of the keywords $\mathcal{K}$ to feed the search algorithm.

According to [70], designing an automated procedure to extract Web data remains challenging due to its large volume and variety. To achieve high accurate performance, existing tools [71–73] always ask human experts to design the extraction pattern or prepare labeled data for training, which are labor-intensive. Other platforms such as RoadRunner [74] avoid human engagement by learning patterns from unlabeled examples automatically. However, they suffer in poor resilience by trading off the performance against the power of automation. In our context, we can do better because the data to extract is no longer the raw data on the web, but some "clean" HTML fragments

supplied by our generic specification detector. In other words, the extraction task is simplified by our first-phase processing. We adopted other techniques [74] for our task but we obtained poor performance. Therefore, instead of using an existing automatic wrapper generation system, we implemented two different wrapper strategies: the first completely unsupervised that exploits the regularity of the specification structure and the second that uses automatic annotations to generate the wrapper.

The first strategy is based on the observation that the structure of these fragments containing the specifications are very homogeneous. By inspecting these tables and lists, we came up with the following heuristic for the extraction. For HTML tables, we first assume each attribute-value pair of the specification is contained in a table row tag (tr). Subsequently, we parse the DOM subtree, in which tr is the parent node, and extracts the text nodes in this subtree. The first text node is considered as attribute and the remaining ones as concatenated values. With respect to HTML lists (ul), we consider that each item in the list (li) contains an attribute-value pair, and the token that separates the attribute from the value is the colon character. In addition to its simplicity, this wrapper is domain-independent and does not require any training. We show in Section 6.6 that it obtains very high values of recall and precision over a set of heterogeneous sites with specifications.

The second strategy is based on the technique proposed by [65]. Dalvi et al. defined an approach for inferring extraction patterns by annotations generated by automatic but noisy annotators. For our purpose, we train two simple annotators considering the attribute-values pairs from the extracted websites. The first annotator annotates nodes in an HTML fragment if there is a perfect match between the string contained in the fragment with one of the values in the training, the second annotator is similar to the previous one but it is trained to extract only the attribute names. Notice that our implementation infers two extraction rules, one for all the values and the other for all the attribute names published in the specifications. A straightforward implementation would be to infer an extraction rule for each attribute in the specifications. As observed by [75], the attributes published by multiple websites of the same domain are skewed, in fact 86% of the attributes are published only by a small percentage of the sources. Adopting a per attribute inference would lead to successfully extracting only overlapping attributes that are just a fraction of the total number of published attributes.

Notice that while the previous heuristic is completely domain independent, the technique based on annotators is domain dependent. [65] requires training related to a specific category and an a-priori distribution to improve the generation of the extraction rule.

Extraction of Keywords A similar technique to the previous specification extraction by annotators is adopted to incrementally extract product keys. From an initial seed set of websites S, we manually define the extraction rules to extract product keys $\mathcal{K}$. From $\mathcal{K}$ we define an annotator that annotates nodes that publish information that match any keyword k_j in $\mathcal{K}$. For each new website the system infers a new extraction rule and extracts new product keys that are added to the initial seed set $\mathcal{K}$. The wrapper generation adopts the technique proposed by [65]. These products keys are then used for querying the search engine and discovering new product websites.

We observe that even with an accurate wrapper generation system, the noise, iteration after iteration, can affect the quality of the generated $\mathcal{K}$. To address this issue we follow the intuition that keywords extracted by multiple websites are more likely to be relevant keywords. We provide a ranking of the keywords and we set a threshold τ to select only the keywords that are extracted by multiple websites.

Each keyword $k_i \in \mathcal{K}$ is scored following this formulation:

$$sKey(k_i, S) = \sum_{s_j \in S} key(k_i, s_j) \tag{6.7}$$

The *key* function is a binary function that scores a key k_i with 1 if and only if k_i is extracted from s_j.
Keywords are considered for search only if $sKey(k_i, S) > \tau$.

6.6 Experiments

In this section, we initially assess the site discovery and crawling, then the specification detection and extraction, and we conclude the section with a summary of the evaluation.

6.6.1 Product sites Discovery and Crawling

Data Collection and Description. We discovered and collected $935k$ pages from $2,719$ online shopping websites related to 5 categories (camera, notebook, headphone, tv, monitor). The corpus was collected running our pipeline with different configurations from an initial set of 10 well-known products websites. We manually wrote the wrappers to extract the product keys $\mathcal{K}$ and the set of attribute-value pairs from the specifications for the initial seed set. We ran the pipeline several times with different configurations: we considered all ranking strategies (backlinks only, search only, union and intersection) and different setup of I, K and S. We used the Bing Search API to search new product

websites with a limit of 250 results for each query and a public API[3] to find backlinks from known relevant websites, here also with a limit of 250 backlinks per website.

cat.	# sites	# pages
camera	1,162	248k
headphone	454	141k
monitor	997	171k
notebook	593	241k
tv	882	133k
all	2,719	935k

TABLE 6.1: Number of sites and pages per category in the dataset

# cat.	% pages
1	71%
2	15%
3	7%
4	4%
5	3%

TABLE 6.2: Percentage of websites with multiple categories

Overall, we discovered 152.539 websites. Over those sites, we ran our home page classifier to detect product sites, resulting in $19,673$ detected sites. We then crawled these sites and manually checked the candidate target pages of $2,719$ websites. These sites were used as gold data G for evaluating the site discovery strategies.

For each website on average we have collected 343 product pages for a total of $935k$ pages. The biggest websites contain more than $40k$ pages while the smallest websites have 5 product pages. We show: in Table 6.1 the number of sites per category and in Table 6.2 the percentage of sites with the number of categories. Only 3% of the websites covers all the categories, while the majority of 70% are related to a single category. Camera is the most frequent category with $1,162$ websites while headphone is the least frequent with 454 websites. The number of pages is very uneven across sites, in fact, popular websites provide a huge catalogue with many products. The $2,719$ websites is collected from $4,088$ websites/categories.

Manual Effort and Tuning. The manual effort required to trigger the complete pipeline was limited. For the initial seed set of 10 websites, we manually wrote wrappers and crawlers to collect the pages and extract specifications for a given category. For each category in those websites we also found manually the Entry Page related to the category. For the Discovery of new websites we set the minimal redundancy of $\mathcal{K}$ to 4, we also trained the Home Page classifier with a manually labeled list of 50 relevant and 50 non relevant websites. For the Insite Crawling we trained two Classifiers to discover of the Target Pages (TP) and Entry Pages (EP). The training was limited to the initial seed set, we trained TP considering the anchor text that pointed to the target pages collected from the initial seed set, and EP using the anchor text that pointed to the

[3]Link Metrics from `apiwiki.moz.com`

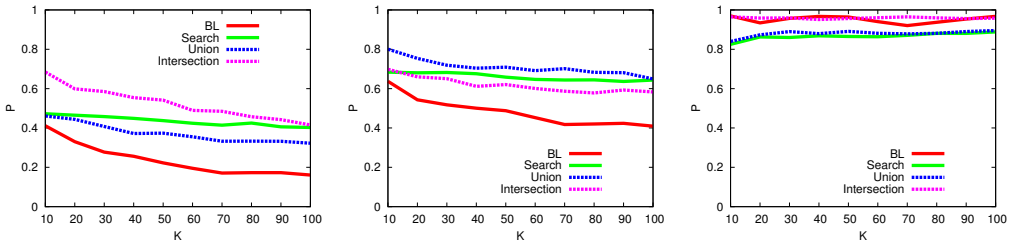


FIGURE 6.7: Precision of the ranking algorithms, $I = 1$: fixed $|S| = 50$ and variable K, (Left) No filter, (Middle) HPF, (Right) ICF.

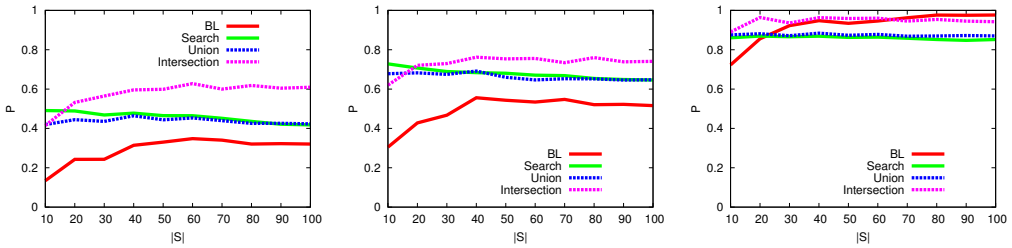


FIGURE 6.8: Precision of the ranking algorithms, $I = 1$: fixed $K = 20$ and a variable $|S|$, (Left) No filter, (Middle) HPF, (Right) ICF.

entry pages from the root of the website. For the specification extraction we trained a classifier based on the features from 37 websites in G.

Strategies. To assess the discovery of new product websites, we evaluated the ranking strategies considering different parameters: the top websites returned from the ranking K, the size of the initial seed set $|S|$, the quality control filters and the number of iterations I for which the pipeline is executed. The quality control filters are based on the home page classifier, Home Page Filter (HPF), which discards evaluation websites that are not recognized by the classifier trained on the home pages of online shopping websites, and on the In-site Crawling Filter (ICF), which discards websites where the crawling returned no product specification pages.

For this evaluation, we take an initial seed set of websites S from our golden set G, from S our system extracts $\mathcal{K}$ and computes different rankings of new websites based on different ranking strategies. It then retrieves the top K sites in the ranking and passes them to the filters. The websites that successfully pass the filters are then considered relevant sites and are added to the initial seed set S. We score the precision considering the intersection between the updated S with our golden set G.

Rankings Results. In Figures 6.7 and 6.8 (Left), we evaluate the ranking strategies when we increase the top K (with a fixed $S = 50$) and the size of the seed set S (with a

fixed $K = 20$). For this experiment, $I = 1$ (number of iterations) and no filter is applied to increase the precision. Overall, increasing K reduces the precision of all the ranking strategies: the Search Only strategy is slightly more resilient. Intersection achieves the best quality obtaining almost a 0.68 in precision when $K = 10$; the precision drops to 0.40 when $K = 100$. Increasing the seed set improves the quality of the ranking strategies: also in this case intersection achieves the best scores, precision goes from 0.41 to 0.61. The quality of the search-only strategy is not affected by the size of seed set. Explanations are (1) we have to consider also the number of product keys we used to search new sites (the average number of pages per site is 343) (2) we expect that the top shopping websites (amazon, bestbuy etc) are easily ranked even with few product keys. From Figure 6.7 (Left), we can also observe that the quality of the combined ranking strategies is affected by the quality of the single ranking strategies: the quality of Intersection drops because the quality of the Backlinks Only drops, the quality of Union is lower than the quality of Search Only because generally Backlinks Only is lower than the Search Only. The loss in precision for the Backlinks Only ranking when the initial seed set is 10 is caused by the presence of many non popular websites.

Filters Results. The previous ranking results were obtained without any filter. But before discussing how our filters affect the ranking results, we present an evaluation of the quality of our filters. The biggest challenge in this evaluation is to feasibly make an estimation of the overall recall. To craft a golden set a manual effort is required but the number of relevant websites is just a small fraction of all the non relevant websites (not conditioned by a ranking strategy), making the manual evaluation difficult. To design an evaluation to estimate the recall, we randomly collected $1,000$ websites from the list of all the websites that are discovered from all the combination of our techniques. We then applied our HPF to discover 145 candidate websites, and our ICF, selecting 40 relevant websites among this smaller filtered set. On these 40 candidate results, we manually checked the quality and estimated the precision for HPF.ICF. To estimate the recall, we further execute the ICF for a random set of 145 websites chosen from the initial $1,000$ websites that are discarded by the HPF. For these websites the ICF returns only 20 websites and manually checking the quality only 5 were relevant. To complete the evaluation, we manually checked the quality of 50 websites from the ones discarded by HPF and ICF, and for those accepted by HPC and discarded by ICF (Table 6.3). From Table 6.3 we can finally estimate the overall P and R of the filters algorithms over a random set of 1000 websites in Table 6.4.

We observe from Table 6.4 that a combination of the two filters HPF and ICF leads to a reasonable solution with $P = 0.87$. The loss in recall is related to multiple factors: non-english websites, index pages with a small list of target pages, dynamic navigation

		HPF	
		yes	no
ICF	yes	35/40	5/20
	no	12.6/105	0/125

TABLE 6.3: # Relevant websites / # non relevant websites, for the HPF and ICF.

	P	R
HPF	0.33	0.62
ICF	0.41	0.84
HPF.ICF	0.87	0.45

TABLE 6.4: Estimated P and R of HPF and ICF.

inside the website, non representative anchor text for links that lead to target pages or to the category entry page. Notice that the obtained results are from a randomly selected 1000 websites, thus the ratio of relevant websites to non relevant websites is not even. Hence, in the next evaluation we consider the effect of the filters on the top K websites returned by the ranking strategies.

In Figures 6.7 and 6.8 (Middle, Right), we also show the impact of our filtering techniques to the ranking strategies, when $I = 1$. In Figures 6.7 and 6.8 (Middle) we can observe that all the ranking strategies are positively affected by the HPF. Overall we have a gain of 0.2 in precision. The strategy that achieves the best boost is Backlinks Only with a gain of 0.3 in precision, when we have a seed set between $70 - 90$ sites. This result is confirmed in Figures 6.7 and 6.8 (Right) where the HPF is combined with the ICF. The precision of Backlinks Only is higher than 0.9 with seed set higher than 60 sites, whereas Search Only achieves only 0.82 in precision. A plausible explanation for that is that if we search for new websites using product keys, we are likely to find websites that provide pages that publish some information about the products with these keys. But it is not guaranteed that these new websites also publish product specifications, e.g., price comparator and review websites. Overall the HPF+ICF combined with the Intersection ranking achieves the best scores, obtaining a precision of 0.95 for K ranging from 10 to 90, and for a seed set greater than 20.

Iterations Results. Figures 6.9 and 6.10 show results of our ranking algorithms running our pipeline for multiple iterations, with initial seed set to $|S| = 20$ and $K = 10$. One may expect that when running multiple iterations the quality of the obtained sites will drop, even as the number of obtained sites increases. However, in our evaluation (Figure 6.9), we can observe that after a slight initial drop in the first 15 iterations, the precision of all the algorithms is almost stable, between iterations 15 and 50. The precision of Intersection is around 0.95 while Search Only has the worst precision, around 0.92. This result supports our statement that a complete iterative pipeline with multiple filtering steps can be adopted to harvest product specifications from all the web. If we consider the absolute number of relevant sites obtained from the ranking algorithms,

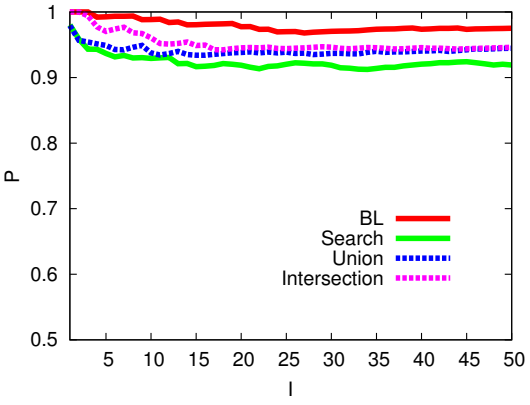


FIGURE 6.9: Precision with an increasing I, $S = 20$ and $K = 10$

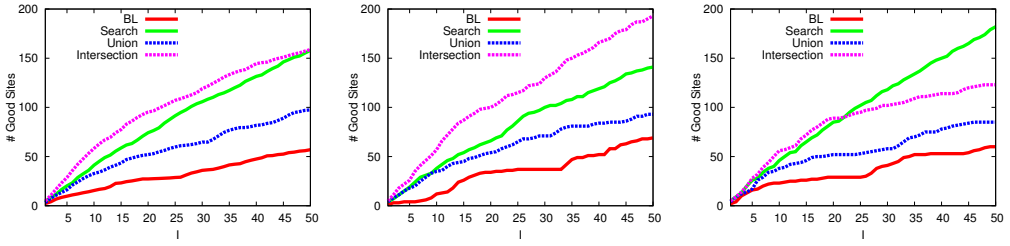


FIGURE 6.10: Number of relevant websites with an increasing I, $S = 20$ and $K = 10$: (Left) average on all domains (Middle) camera (Right) notebook

Figure 6.10 (Left) shows that Intersection and Search Only are the best approaches, discovering overall 160 new websites after 50 iterations. Whereas, in Figure 6.10 (Left), we provide an average value across different categories, we observe that the growth in number of sites related to the ranking algorithm is strongly related to the searched category. Figure 6.10 (Middle) shows that for camera category Intersection is better than Search Only for all the 50 iterations, while Figure 6.10 (Right) for notebook category after 20 iterations Search Only obtains more relevant websites. This is caused by a few factors: the quality of the hubs and the presence of a clear key attribute for the considered category.

6.6.2 Specification Detection and Extraction

Data and setup. To evaluate our specification detection (SD) and specification extraction (SE) steps, we consider a subset of 37 websites from G (30 random websites and 7 well known shopping websites). For each website, we manually crafted wrappers to extract the specifications and their attribute name/value pairs. To train the SD, we take at most 50 pages from each website and from each category (some websites have fewer pages). The positive examples are the specifications extracted by the wrapper and

the negative examples are the tables/lists not considered relevant by the site wrapper. Since in this context, the number of negative examples is overwhelmingly higher than the positive ones, and this can affect the classification performance [76], we restricted the number of negative examples to be the same as the number of positive.

We evaluate the SD and SE in two different scenarios: 1) across sites and same category and 2) across sites and across categories. For the testing we adopt a *leave-one-outside* approach. More specifically: (1) we train a classifier for each website considering all the features from the other websites of the same category; and (2) we train the same classifier considering also features from different categories.

For Specification Extraction, we consider our two alternative solutions: one [65] based on wrapper inference from noisy annotations (WI) and one that follows the heuristic of the table structure (SE|SD).

Features	P	R	P (table)	R (table)
Our	0.84	0.90	0.88	0.92
Wang-Hu	0.66	0.78	0.79	0.94
Both	0.87	0.91	0.92	0.94

TABLE 6.5: Precision and recall for the Specification Detection

Results. Table 6.5 compares precision and recall of a classifier trained with our features and the features defined in [69, 77] for specification detection. It also shows the average precision and recall obtained on our dataset. Our features are more robust to non-table specifications and have a better precision with a small loss in recall for only table specifications. Combining all the features from both approaches increases slightly the average precision by 0.03 and recall by 0.01, the increase is mostly for tables where the combination of our features achieves a 0.92 in precision and 0.94 in recall.

Table 6.6 shows the average precision and recall obtained by all approaches on the 37 sites and, for the sake of space, we present the individual results of only 10 of them, focusing our discussion on those cases where our approach (SE|SD) achieves the worst results. Table 6.6 compares our wrapper conditioned on a perfect SD's output (SE|SD*) i.e., we calculated the performance of the wrapper in isolation, the baseline WI, the quality of the annotations (An.), the wrapper inference conditioned on a perfect SD's output (WI|SD*) and a hybrid approach (WI+(SE|SD)) that chooses between WI and SE|SD when a quality check is passed.

Overall, our approach (SE|SD) obtained very high values of recall and precision over most of the sites (average precision equals to 0.80 and recall 0.90) and the results are

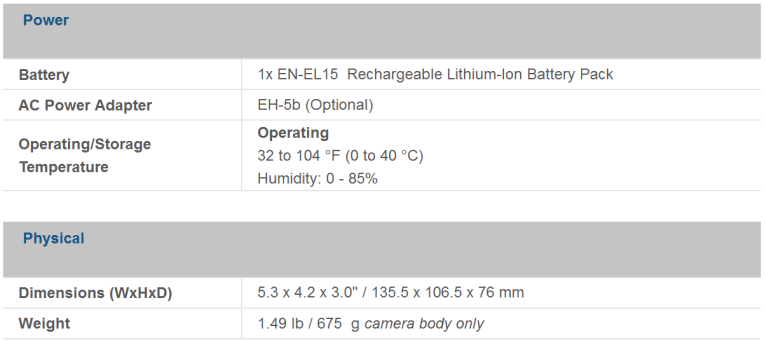
	SE|SD		SE|SD*		WI		An.		WI|SD*		An.|SD*		WI+SE|SD	
website	P	R	P	R	P	R	P	R	P	R	P	R	P	R
shop.lenovo	0.81	0.87	0.87	0.87	1.00	0.87	0.24	0.22	1.00	0.87	0.24	0.22	0.81	0.87
uk.hardware	0.83	0.76	0.83	1.00	0.96	0.82	0.46	0.33	0.95	0.79	0.68	0.23	0.96	0.82
abt	0.46	1.00	1.00	1.00	0.99	1.00	0.30	0.57	1.00	1.00	1.00	0.48	0.99	1.00
alibaba	0.35	0.75	0.90	0.90	0.07	0.07	0.10	0.05	0.99	1.00	0.48	0.05	0.35	0.75
bhphotovideo	0.96	0.78	1.00	1.00	0.97	1.00	0.49	0.43	0.93	1.00	0.99	0.64	0.97	1.00
buzzillions	0.73	1.00	0.95	1.00	0.86	1.00	0.24	0.42	0.92	1.00	0.82	0.32	0.86	1.00
cyberguys	0.66	0.99	1.00	0.99	0.99	0.96	0.49	0.15	0.99	0.96	0.89	0.07	0.99	0.96
netplus	0.60	0.96	0.80	0.96	0.99	1.00	0.44	0.48	0.98	1.00	0.87	0.45	0.99	1.00
newegg	0.92	0.54	1.00	1.00	0.92	1.00	0.29	0.41	0.92	1.00	0.74	0.38	0.92	1.00
pcrichard	0.00	0.00	0.00	0.00	1.00	1.00	0.17	0.14	1.00	1.00	1.00	0.06	1.00	1.00
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
Average	**0.80**	**0.90**	**0.95**	**0.95**	**0.85**	**0.85**	**0.38**	**0.39**	**0.92**	**0.86**	**0.88**	**0.32**	**0.92**	**0.95**

TABLE 6.6: Results for our wrapper and the baselines on 10 websites (the most erroneous among the 37 sites).

comparable with WI with a loss in precision but a higher recall. The only exception was pcrichard, in which WI obtained a perfect score while our wrapper was not able to extract the correct results. We observed that pcrichard does not provide a table-like structure (it is the only site with a *dl* structure) leading to mistakes for the SD and SE. For precision, SE|SD obtained poor results in some websites: newegg, alibaba and abt. The reason for this is that these websites provide in their specification pages other types of information that have similar structure to specifications or might also be considered as part of them. For instance, some of the lists misclassified by the SD on newegg website contained information of some product features, which were not presented in the specification of gold data for this product. Regarding recall, the loss occurs in those websites with specifications consisting of small tables, as one can see in Figure 6.11.

In addition, when one compares SE|SD vs SE|SD*, it is clear that SD is the main reason for the limitations of SE|SD. The number shows that SE performs an almost perfect job: average precision and recall equals to 0.95. The loss in quality is related to sites such as pcrichard, where no table structure is present, shop.lenovo and netplus, where the specifications' table rows sometimes consist of three columns, two dedicated to labels and one to the value.

In WI, the extraction performance is determined by the quality of the set of annotations. The annotator generally performs poorly with an average precision of 0.38 and an average recall of 0.39. For some websites WI achieves perfect scores. The loss in precision and recall is strongly related to the quality of the annotators [65]. Another aspect that strongly affects the extraction error rate is the dependency of the annotations' mistakes. In fact, in [65] the authors adopted a random distribution to control and define an annotator of desired quality, in our setting we observed that there is a strong dependency on mistakes made by our annotator.



Power	
Battery	1x EN-EL15 Rechargeable Lithium-Ion Battery Pack
AC Power Adapter	EH-5b (Optional)
Operating/Storage Temperature	Operating 32 to 104 °F (0 to 40 °C) Humidity: 0 - 85%

Physical	
Dimensions (WxHxD)	5.3 x 4.2 x 3.0" / 135.5 x 106.5 x 76 mm
Weight	1.49 lb / 675 g *camera body only*

FIGURE 6.11: Example of a specification from bhphotovideo.

We observe that inferring the specifications from the right portion of the HTML page boosts both the annotator's precision and the extraction quality. WI|SD* obtains a precision of 0.92 and a recall of 0.86 compared to the previous 0.85 and 0.85 of WI. The most common mistakes are (1) the presence of short feature lists that are not specification but with specifications values and (2) the presence of specification values that are used as attribute names in other websites.

The first issue is addressed by WI|SD*, so that the annotation process is applied only on the portion of the HTML document with a specification. The second issue is addressed by taking into account both specifications values and attribute names during training: matches on values are positive annotations while matches on attribute names are negative annotations.

In WI+(SE|SD), we considered a hybrid approach. We observe that often WI and SE|SD makes mistakes for different reasons and that it is possible to define a criterion to choose one approach over another. The intuition is that we can check the quality of the WI by comparing the attribute names and the values. As for WI, the system learns an extraction rule to extract the values of the specifications by using an annotator that matches the values in the test website with the values in the training websites. We adopt the same technique to learn an extraction rule to extract the attribute names of the test website, and compare the two extraction rules, the one for the values and the one for the attribute names. We observe that rules that extract values and attribute names for the specifications are likely to extract paired nodes. When it successfully learns two paired rules, it uses WI, or uses SE|SD if no paired rules are found. WI+(SE|SD) achieves really high precision and recall. The loss in quality is related to few websites, where WI fails and the heuristic for SE|SD is not perfect as, e.g., alibaba.

We observe that the diversity across websites is the main issue that affects the SD quality. This observation is confirmed by Table 6.7 where we compared the classification quality with a per categories basis. The SD (websites) has the same configuration as Table 6.6

	SD (websites)		**SD (categories)**	
	P	R	P	R
camera	0.88	0.92	0.94	0.96
headphone	0.86	0.89	0.89	0.97
notebook	0.81	0.88	0.98	0.98
monitor	0.83	0.87	0.89	0.98
tv	0.83	0.91	0.96	0.98

TABLE 6.7: Results for the SD, per site and per category

but with the average score for each category. Here we observe the classification quality is not drastically affected by the variability across categories. The next question is: if a classifier is trained to recognize a set of categories, can it be used on other categories? The answer is in Table 6.7, in SD (categories) for each category, the training is made by features from other categories and the testing on websites that contain pages related to the considered category and at least another category, i.e., the classifier has been trained to recognize another category for the same website. The quality is much higher: for notebook and tv, we have almost perfect precision and recall with a small drop only for headphone and monitor precision. This motivates our consideration that after discovering the specifications for a subset of the categories of a website, we can use the same classifier to recognize the specifications for new categories in the website.

6.6.3 Summary

Our results show that, with a limited human effort, DEXTER:

- Efficiently **discovered** and **crawled** 935k product specification pages from 2,719 websites for five different product categories. Tables 6.1 and 6.2 present the collected dataset and Figures 6.8, 6.8, 6.9 and 6.10 show that the *vote-filter-iterate* principles applied to our setting can accurately discover websites with product specifications with a high precision.

- Accurately **detected** product specifications. In Tables 6.5 and 6.7 our specification detector achieves on average $F = 0.87$ for specifications in unknown websites and known categories and F that goes from 0.94 to 0.98 for known websites and unknown categories.

- Accurately **extracted** attribute name/value pairs. Table 6.6 shows that a hybrid approach that combines a domain independent with a domain dependent approach achieves a 0.92 in precision and 0.95 in recall, close to settings where the detection is given as perfect.

6.7 Related Works

Webtable. Many previous approaches try to explore the Web to obtain structured data [68, 69, 77, 78]. The WebTable project [77, 78] extracts HTML tables which contain relational data, similar to [69], and applies techniques to search and explore these tables. Similarly, Gupta and Sarawagi [68] propose a system that extracts and integrates tuples from HTML lists. Both approaches target at constructing a corpus of high-quality data, where, to *recover* the semantics of the data content, a huge amount of post-processing effort such as entity resolution and schema matching [79, 80] is needed. In contrast, our approach requires to explore product semantics (e.g., category) during the discovery of their specifications. Namely, the specifications we obtain are automatically categorized.

Wrappers. Along with these systems, various techniques/tools have been proposed to extract structured data from web pages. Strategies exploit the opportunity of web page similarity in both HTML structure and natural language. Usually, a pattern (aka. wrapper) exploring the underlying similarity is obtained and will later be applied to other pages for further extraction. Much work [74] studies how to develop wrappers automatically but the quality of the output, in many cases, is low and not controllable. To control the automatic generation of wrappers, several techniques adopt: *domain knowledge* [9], but a knowledge base has to be crafted for each category; *redundancy* [10, 11], but products are characterized by many "rare" attributes that are present only in few sources; *annotators* [65], but it is hard to define a priori a set of annotators that can annotate all the present attributes.

Source Discovery. An analysis of structured data on the web has been described by [62]. The authors adopted a search paradigm to discover all the websites related to several domains and extracted some attributes from them. The authors found thousands of websites for several domains, but their evaluation was limited to few key attributes (identifiers), making the extraction step much simpler. Many other works adopted the search paradigm [42, 43], but the number of visited websites is one order of magnitude lower than our approach, thus the task of maintaining a high efficiency and quality was much simpler.

Products. Another topic related to our work is product integration and categorization. Existing work follows a supervised approach, which starts from an already well-established product database, and accomplishes integration for new product instances. Nguyen et al. [81] propose a scalable approach to synthesize product offers

from thousand of merchants into their centralized schema. Kannan et al. [63] build a system to perform matching from unstructured product offers to structured specifications. However, none of them has specified how to construct a high-quality product database aforehand.

Crawling. Techniques to automatically crawl target pages have been studied [44, 45, 47]. All these techniques guide the crawler by adopting some kind of knowledge: in [44] authors guide the focused crawler considering the semantic annotations on target pages; in [45], authors exploit the expected structure of a forum to efficiently crawl generic forums; in [47], the system infers relationships among instances present in a database from parallel navigation paths. In our approach the concept of target pages and the domain differ from previous approaches making these previous approaches not directly applicable.

6.8 Conclusions

In this chapter we have presented DEXTER, an end-to-end solution to the task of building specification databases from web pages. For that, we propose techniques to discover, crawl, detect and extract product specifications.

To efficiently discover product websites DEXTER explores different techniques that rely on existing search APIs, for keywords search and navigating backlinks. To collect product pages DEXTER crawls shopping websites. To detect specifications, the Specification Detector identifies the tables and lists that contain product specifications. Finally, to extract the attribute-value pairs from the detected specification fragments, DEXTER adopts two wrapper generation techniques, a domain independent and a domain dependent approach.

A future direction is to use the specification database obtained using our technique to perform entity and attribute matching in order to build a universal specification database. Other interesting directions are: the selection of "good" sources to integrate [82] and the discovery of new categories based on the navigation structure of the product websites.

Chapter 7

Conclusions and Future Works

Data Extraction at Web scale is an open challenge. Previous proposals tried to "scale up" the generation of wrapper by defining inference algorithms: unsupervised approaches have a high scalability but they still require human experts to guide and fix the quality of the output wrapper; supervised approaches have a higher accuracy, but the generation of the training data required for inference process is expensive. Many proposals tried to address the previous issues [8–11], but they are limited by the information adopted as training in each system. Some proposals are domain dependent [8, 9], they are limited by the domain knowledge required for the inference algorithm, others adopt the redundancy on the Web to infer wrappers [10, 11], they scale over several domains, but the quality of the output is not controllable. The Web is characterized by a great variety of websites, thus a single automatic technique can scale the Web.

Contributions This dissertation presented a technique to address these challenges:

- we defined a Quality Model and a learning algorithm ALF$_\eta$ that estimates the quality of the output wrapper considering a sequence of annotations, a training sequence.
- we defined 4 unsupervised wrapper inference techniques, some inspired by previous proposals [5] and we adopted the Quality Model to evaluated them by defining responders that generate training sequences based on MQ posed by ALF$_\eta$.
- We described an algorithm ALFRED inspired by an EM solution that exploits the mutual dependency between the error rate of responders and the expected quality of the output wrapper.

- We designed a hybrid approach that adopts the Quality Model to evaluate the automatic wrapper inference approach based on the combination of the previous 4 unsupervised wrapper inference techniques.
- If the expected quality is not enough, additional training sequences are created by engaging humans enrolled from a crowdsourcing platform.
- We designed an original learning paradigm suitable for non-expert workers.
- To control the quality of the crowd, the Quality Model is combined with ALFRED to enroll multiple workers on the same task and estimate at runtime the their error rates.
- To reduce the cost of the crowd, we adopted Active Learning and we selected at runtime the number of required workers for the task.
- We exploited the schedule of the crowd to further reduce the cost.

An extensive evaluation with real data collected from the Web with workers enrolled from a crowdsourcing platform and synthetic workers show: we learn accurate wrappers with F close to 1 and a low standard deviation; the hybrid approach achieves the quality of a solution based completely on humans with just a fraction of the human supervision, 2.3 *MQ* and on average 0.36 workers for each attribute; ALFRED can effectively estimate workers' error rate.

Parts of this dissertation are the results of several publications in conferences and a journal. In Chapter 3: the Active Learning with the query selection policies and the sampling algorithm have been described in [16]; a demo with the application that we adopted to evaluate the crowd has been described in [15]; the Quality Model and the model with a single noisy worker has been described in [17]. In Chapter 4: the scheduling of the tasks and the dynamic recruitment of workers enrolled from a crowdsourcing platform have been described in [17]. The Chapters 4 and 6 are original and they are under submission. Other works published during the PhD program are not described in this dissertation: in [19] we described a sequence prediction model based on HMM for routes prediction in a noisy environment; [18] shows technique to extract and integrate triples from a knowledge base considering Data Extraction techniques; and several workshop papers [83–85] that describes parts of the system and future works.

Future Directions The focus of this dissertation is to scale the data extraction pipeline to the Web. But to deploy a real end to end system there are many open challenges to address.

A first issue is to organize the extracted data under a common database, this is a well known data integration problem, but the scale of the setting and the noise of the extracted information make this issue non trivial. Which sources do we have to consider? In which order? How can we effectively scale over thousands of sources? These are still open questions.

A second issue is related to the noise in different steps of the pipeline. For instance, the noise and errors: on the websites to consider, on the selected pages to wrap, on the extraction rules adopted to collect structured information, of the integrated data under an uniform schema and the linkage of the instances. Considering the scale of the Web, the solutions for these steps are characterize by a high degree of automation. Automatic approaches are typically not accurate enough, while supervised approaches are costly and require expert users. "Humans in the loop" is required to control the quality; this motivates a hybrid approach deployed in all the data processing lifecycle. With this assumption we developed our approach for generating extraction rules but a similar approach could be deployed to address other steps of the pipeline.

A third issue is related to the motivation of the crowd. The interactions described in this dissertation are based on a "work for pay" model, where workers are paid to complete a defined task. There are several disadvantage on this model: (i) there is a constant cost for each completed task, (ii) workers are motivated by the money and they are to make a great job, (iii) there is no history of the workers' tasks, thus it is challenging to predict their error rates. Self-motivated workers are known to be more accurate wrt paid workers. The possibility of defining a self-motivated community would further improve the quality of the pipeline. Possible triggers would be: benefits of the community, considering the extraction pipeline, if we specialize the task on some domains such as travel, people could be motivated to share their work for the good of the community; personal achievements, in some cases it is more cost effective to motivate workers to do a good job by providing few prizes than paying them for their tasks; gamification, the feedback provided by workers could be encoded inside "games" to self-motivate workers.

Another open issue is the discovery of structured data that belongs to new categories inside a crawled website. In Chapter 6 we proposed an approach that discovers new sources given a small seed set of websites organized in a category. An interesting problem is to discover new categories given a small seed set of categories. The approach exploits the fact that websites often publish data on multiple categories, thus the information learned from the known categories could be used to discover new categories on the Web. With this extension our approach would be able to start from a fixed seed of websites and categories and discover on the Web new sources by increasing both coordinates, number of sources and number of categories.

Bibliography

[1] Dayne Freitag and Nicholas Kushmerick. Boosted wrapper induction. In *AAAI/IAAI*, pages 577–583, 2000.

[2] Ion Muslea, Steven Minton, and Craig A. Knoblock. Active learning with strong and weak views: A case study on wrapper induction. In *IJCAI*, pages 415–420. Morgan Kaufmann, 2003.

[3] Georg Gottlob, Christoph Koch, Robert Baumgartner, Marcus Herzog, and Sergio Flesca. The Lixto data extraction project - back and forth between theory and practice. In *PODS*, pages 1–12. ACM, 2004. ISBN 1-58113-858-X.

[4] V. Crescenzi, G. Mecca, and P. Merialdo. RoadRunner: Towards automatic data extraction from large web sites. In *Int. Conf. on Very Large Data Bases (VLDB'2001), Roma, Italy, September 11-14*, pages 109–118, 2001.

[5] Arvind Arasu and Hector Garcia-Molina. Extracting structured data from web pages. In *SIGMOD Conference*, pages 337–348. ACM, 2003. ISBN 1-58113-634-X.

[6] Yanhong Zhai and Bing Liu. Structured data extraction from the web based on partial tree alignment. *IEEE Trans. Knowl. Data Eng.*, 18(12):1614–1628, 2006.

[7] Pierre Senellart, Avin Mittal, Daniel Muschick, Rémi Gilleron, and Marc Tommasi. Automatic wrapper induction from hidden-web sources with domain knowledge. In *Proceedings of the 10th ACM Workshop on Web Information and Data Management*, WIDM '08, pages 9–16, New York, NY, USA, 2008. ACM. ISBN 978-1-60558-260-3. doi: 10.1145/1458502.1458505. URL http://doi.acm.org/10.1145/1458502.1458505.

[8] Nilesh N. Dalvi, Ravi Kumar, and Mohamed A. Soliman. Automatic wrappers for large scale web extraction. *PVLDB*, 4(4):219–230, 2011.

[9] 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), 2014.

[10] Shui-Lung Chuang, Kevin Chen-Chuan Chang, and ChengXiang Zhai. Context-aware wrapping: synchronized data extraction. In *Proceedings of the 33rd international conference on Very large data bases*, pages 699–710. VLDB Endowment, 2007.

[11] 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.

[12] Nilesh N. Dalvi, Ashwin Machanavajjhala, and Bo Pang. An analysis of structured data on the web. *PVLDB*, 5(7):680–691, 2012.

[13] Vladimir Vapnik. An overview of statistical learning theory. *IEEE Transactions on Neural Networks*, 10(5):988–999, 1999.

[14] Dana Angluin. Queries revisited. *Theor. Comput. Sci.*, 313(2):175–194, 2004.

[15] Valter Crescenzi, Paolo Merialdo, and Disheng Qiu. A framework for learning web wrappers from the crowd. In *Proceedings of the 22nd international conference on World Wide Web*, WWW '13, pages 261–272, Republic and Canton of Geneva, Switzerland, 2013. International World Wide Web Conferences Steering Committee. ISBN 978-1-4503-2035-1. URL http://dl.acm.org/citation.cfm?id=2488388.2488412.

[16] Valter Crescenzi, Paolo Merialdo, and Disheng Qiu. Alfred: Crowd assisted data extraction. In *Proceedings of the 22Nd International Conference on World Wide Web Companion*, WWW '13 Companion, pages 297–300, Republic and Canton of Geneva, Switzerland, 2013. International World Wide Web Conferences Steering Committee. ISBN 978-1-4503-2038-2. URL http://dl.acm.org/citation.cfm?id=2487788.2487927.

[17] Valter Crescenzi, Paolo Merialdo, and Disheng Qiu. Crowdsourcing large scale wrapper inference. *Distributed and Parallel Databases*, 33(1):95–122, 2015. ISSN 0926-8782. doi: 10.1007/s10619-014-7163-9. URL http://dx.doi.org/10.1007/s10619-014-7163-9.

[18] Lorenz Bühmann, Ricardo Usbeck, Axel-Cyrille Ngonga Ngomo, Muhammad Saleem, Andreas Both, Valter Crescenzi, Paolo Merialdo, and Disheng Qiu. Web-scale extension of RDF knowledge bases from templated websites. In Peter Mika, Tania Tudorache, Abraham Bernstein, Chris Welty, Craig Knoblock, Denny Vrandečić, Paul Groth, Natasha Noy, Krzysztof Janowicz, and Carole Goble, editors, *The Semantic Web – ISWC 2014*, volume 8796 of *Lecture Notes in Computer*

Science, pages 66–81. Springer International Publishing, 2014. ISBN 978-3-319-11963-2. doi: 10.1007/978-3-319-11964-9_5. URL http://dx.doi.org/10.1007/978-3-319-11964-9_5.

[19] Disheng Qiu, Paolo Papotti, and Lorenzo Blanco. Future locations prediction with uncertain data. In Hendrik Blockeel, Kristian Kersting, Siegfried Nijssen, and Filip Železný, editors, *Machine Learning and Knowledge Discovery in Databases*, volume 8188 of *Lecture Notes in Computer Science*, pages 417–432. Springer Berlin Heidelberg, 2013. ISBN 978-3-642-40987-5. doi: 10.1007/978-3-642-40988-2_27. URL http://dx.doi.org/10.1007/978-3-642-40988-2_27.

[20] Disheng Qiu, Luciano Barbosa, Xin Luna Dong, Yanyan Shen, and Divesh Srivastava. Dexter: Large-scale discovery and extraction of product specifications on the web. Technical report, 2015. URL http://cl.ly/3s182f0a2M1A/paper.pdf.

[21] Nicholas Kushmerick. Wrapper induction: Efficiency and expressiveness. *Artif. Intell.*, 118(1-2):15–68, 2000.

[22] Chia Hui Chang, Mohammed Kayed, Moheb R Girgis, and Khaled F Shaalan. A survey of web information extraction systems. *Knowledge and Data Engineering, IEEE Transactions on*, 18(10):1411–1428, 2006.

[23] 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 VLDB Journal*, 22(1):47–72, 2013. ISSN 1066-8888. doi: 10.1007/s00778-012-0286-6. URL http://dx.doi.org/10.1007/s00778-012-0286-6.

[24] Maria-Florina Balcan, Steve Hanneke, and Jennifer Wortman Vaughan. The true sample complexity of active learning. *Machine Learning*, 80(2-3):111–139, 2010.

[25] Burr Settles. Active learning literature survey. Computer Sciences Technical Report 1648, University of Wisconsin–Madison, 2009.

[26] Victor S Sheng, Foster Provost, and Panagiotis G Ipeirotis. Get another label? improving data quality and data mining using multiple, noisy labelers. In *Proceedings of the 14th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 614–622. ACM, 2008.

[27] Mike Mintz, Steven Bills, Rion Snow, and Dan Jurafsky. Distant supervision for relation extraction without labeled data. In *Proceedings of the Joint Conference of the 47th Annual Meeting of the ACL and the 4th International Joint Conference on Natural Language Processing of the AFNLP: Volume 2-Volume 2*, pages 1003–1011. Association for Computational Linguistics, 2009.

[28] Utku Irmak and Torsten Suel. Interactive wrapper generation with minimal user effort. In *WWW*, pages 553–563. ACM, 2006. ISBN 1-59593-323-9.

[29] Ion Muslea, Steven Minton, and Craig A. Knoblock. Active learning with multiple views. *J. Artif. Intell. Res. (JAIR)*, 27:203–233, 2006.

[30] Michael J Franklin, Donald Kossmann, Tim Kraska, Sukriti Ramesh, and Reynold Xin. Crowddb: answering queries with crowdsourcing. In *Proceedings of the 2011 ACM SIGMOD International Conference on Management of data*, pages 61–72. ACM, 2011.

[31] Aditya Ganesh Parameswaran, Hyunjung Park, Hector Garcia-Molina, Neoklis Polyzotis, and Jennifer Widom. Deco: declarative crowdsourcing. In *Proceedings of the 21st ACM international conference on Information and knowledge management*, pages 1203–1212. ACM, 2012.

[32] Chen Jason Zhang, Lei Chen, HV Jagadish, and Chen Caleb Cao. Reducing uncertainty of schema matching via crowdsourcing. *Proceedings of the VLDB Endowment*, 6(9):757–768, 2013.

[33] Robert McCann, Warren Shen, and AnHai Doan. Matching schemas in online communities: A web 2.0 approach. In *Data Engineering, 2008. ICDE 2008. IEEE 24th International Conference on*, pages 110–119. IEEE, 2008.

[34] Norman W Paton and Alvaro AA Fernandes. Crowdsourcing feedback for pay-as-you-go data integration. *DBCrowd 2013*, page 32, 2013.

[35] Jiannan Wang, Tim Kraska, Michael J Franklin, and Jianhua Feng. Crowder: Crowdsourcing entity resolution. *Proceedings of the VLDB Endowment*, 5(11):1483–1494, 2012.

[36] Gianluca Demartini, Djellel Eddine Difallah, and Philippe Cudré-Mauroux. Large-scale linked data integration using probabilistic reasoning and crowdsourcing. *The VLDB Journal*, 22(5):665–687, 2013.

[37] Chaitanya Gokhale, Sanjib Das, AnHai Doan, Jeffrey F Naughton, Narasimhan Rampalli, Jude Shavlik, and Xiaojin Zhu. Corleone: Hands-off crowdsourcing for entity matching. In *Proceedings of the 2014 ACM SIGMOD international conference on Management of data*, pages 601–612. ACM, 2014.

[38] Jiannan Wang, Guoliang Li, Tim Kraska, Michael J Franklin, and Jianhua Feng. Leveraging transitive relations for crowdsourced joins. In *Proceedings of the 2013 ACM SIGMOD International Conference on Management of Data*, pages 229–240. ACM, 2013.

[39] Susan B Davidson, Sanjeev Khanna, Tova Milo, and Sudeepa Roy. Using the crowd for top-k and group-by queries. In *Proceedings of the 16th International Conference on Database Theory*, pages 225–236. ACM, 2013.

[40] Aditya G Parameswaran, Hector Garcia-Molina, Hyunjung Park, Neoklis Polyzotis, Aditya Ramesh, and Jennifer Widom. Crowdscreen: Algorithms for filtering data with humans. In *Proceedings of the 2012 ACM SIGMOD International Conference on Management of Data*, pages 361–372. ACM, 2012.

[41] Lorenzo Blanco, Nilesh Dalvi, and Ashwin Machanavajjhala. Highly efficient algorithms for structural clustering of large websites. In *Proceedings of the 20th international conference on World wide web*, pages 437–446. ACM, 2011.

[42] Lorenzo Blanco, Valter Crescenzi, Paolo Merialdo, and Paolo Papotti. Supporting the automatic construction of entity aware search engines. In *Proceedings of the 10th ACM Workshop on Web Information and Data Management*, WIDM '08, pages 149–156, New York, NY, USA, 2008. ACM. ISBN 978-1-60558-260-3. doi: 10.1145/1458502.1458526. URL http://doi.acm.org/10.1145/1458502.1458526.

[43] Oren Etzioni, Michael Cafarella, Doug Downey, Stanley Kok, Ana-Maria Popescu, Tal Shaked, Stephen Soderland, Daniel S. Weld, and Alexander Yates. Web-scale information extraction in knowitall: (preliminary results). In *WWW*, pages 100–110, 2004. ISBN 1-58113-844-X. doi: 10.1145/988672.988687. URL http://doi.acm.org/10.1145/988672.988687.

[44] Robert Meusel, Peter Mika, and Roi Blanco. Focused crawling for structured data. In *Proceedings of the 23rd ACM International Conference on Conference on Information and Knowledge Management*, CIKM '14, pages 1039–1048, New York, NY, USA, 2014. ACM. ISBN 978-1-4503-2598-1. doi: 10.1145/2661829.2661902. URL http://doi.acm.org/10.1145/2661829.2661902.

[45] Jingtian Jiang, Xinying Song, Nenghai Yu, and Chin-Yew Lin. Focus: learning to crawl web forums. *Knowledge and Data Engineering, IEEE Transactions on*, 25(6): 1293–1306, 2013.

[46] Lorenzo Blanco, Valter Crescenzi, and Paolo Merialdo. Efficiently locating collections of web pages to wrap. In *In WEBIST*, 2005.

[47] Tim Weninger, Thomas J. Johnston, and Jiawei Han. The parallel path framework for entity discovery on the web. *ACM Trans. Web*, 7(3):16:1–16:29, September 2013. ISSN 1559-1131. doi: 10.1145/2516633.2516638. URL http://doi.acm.org/10.1145/2516633.2516638.

[48] Dana Angluin and Philip Laird. Learning from noisy examples. *Mach. Learn.*, 2 (4):343–370, April 1988. ISSN 0885-6125. doi: 10.1023/A:1022873112823. URL http://dx.doi.org/10.1023/A:1022873112823.

[49] Qiang Liu, Alexander T. Ihler, and Mark Steyvers. Scoring workers in crowdsourcing: How many control questions are enough? In *Advances in Neural Information Processing Systems 26: 27th Annual Conference on Neural Information Processing Systems 2013. Proceedings of a meeting held December 5-8, 2013, Lake Tahoe, Nevada, United States.*, pages 1914–1922, 2013. URL http://papers.nips.cc/paper/4889-scoring-workers-in-crowdsourcing-how-many-control-questions-are-enough.

[50] Michael D Lee, Mark Steyvers, Mindy De Young, and Brent Miller. Inferring expertise in knowledge and prediction ranking tasks. *Topics in cognitive science*, 4 (1):151–163, 2012.

[51] David R. Karger, Sewoong Oh, and Devavrat Shah. Iterative learning for reliable crowdsourcing systems. In John Shawe-Taylor, Richard S. Zemel, Peter L. Bartlett, Fernando C. N. Pereira, and Kilian Q. Weinberger, editors, *Advances in Neural Information Processing Systems 24: 25th Annual Conference on Neural Information Processing Systems 2011. Proceedings of a meeting held 12-14 December 2011, Granada, Spain.*, pages 1953–1961, 2011. URL http://papers.nips.cc/paper/4396-iterative-learning-for-reliable-crowdsourcing-systems.

[52] Adam Marcus, David R. Karger, Samuel Madden, Rob Miller, and Sewoong Oh. Counting with the crowd. *PVLDB*, 6(2):109–120, 2012. URL http://www.vldb.org/pvldb/vol6/p109-marcus.pdf.

[53] Panagiotis G. Ipeirotis. Analyzing the amazon mechanical turk marketplace. *XRDS*, 17(2):16–21, December 2010. ISSN 1528-4972. doi: 10.1145/1869086.1869094. URL http://doi.acm.org/10.1145/1869086.1869094.

[54] Qiang Hao, Rui Cai, Yanwei Pang, and Lei Zhang. From one tree to a forest: a unified solution for structured web data extraction. In Wei-Ying Ma, Jian-Yun Nie, Ricardo A. Baeza-Yates, Tat-Seng Chua, and W. Bruce Croft, editors, *Proceeding of the 34th International ACM SIGIR Conference on Research and Development in Information Retrieval, SIGIR 2011, Beijing, China, July 25-29, 2011*, pages 775–784. ACM, 2011. doi: 10.1145/2009916.2010020. URL http://doi.acm.org/10.1145/2009916.2010020.

[55] Xin Luna Dong, Evgeniy Gabrilovich, Geremy Heitz, Wilko Horn, Kevin Murphy, Shaohua Sun, and Wei Zhang. From data fusion to knowledge fusion. *Proc. VLDB*

Endow., 7(10):881–892, June 2014. ISSN 2150-8097. URL http://dl.acm.org/citation.cfm?id=2732951.2732962.

[56] AnHai Doan, Pedro Domingos, and Alon Y Halevy. Reconciling schemas of disparate data sources: A machine-learning approach. In *ACM Sigmod Record*, volume 30, pages 509–520. ACM, 2001.

[57] Kenneth Ward Church and Patrick Hanks. Word association norms, mutual information, and lexicography. *Computational linguistics*, 16(1):22–29, 1990.

[58] David A Grossman. *Information retrieval: Algorithms and heuristics*, volume 15. Springer Science & Business Media, 2004.

[59] Eric Crestan and Patrick Pantel. Web-scale table census and classification. In *Proceedings of the fourth ACM international conference on Web search and data mining*, pages 545–554. ACM, 2011.

[60] Larissa R. Lautert, Marcelo M. Scheidt, and Carina F. Dorneles. Web table taxonomy and formalization. *SIGMOD Rec.*, 42(3):28–33, October 2013. ISSN 0163-5808. doi: 10.1145/2536669.2536674. URL http://doi.acm.org/10.1145/2536669.2536674.

[61] M.J. Cafarella, A. Halevy, and J. Madhavan. Structured data on the web. *Communications of the ACM*, 54(2):72–79, 2011.

[62] N. Dalvi, A. Machanavajjhala, and B. Pang. An analysis of structured data on the web. *Proceedings of the VLDB Endowment*, 5(7):680–691, 2012.

[63] Anitha Kannan, Inmar E. Givoni, Rakesh Agrawal, and Ariel Fuxman. Matching unstructured product offers to structured product specifications. In *Proceedings of the 17th ACM SIGKDD international conference on Knowledge discovery and data mining*, KDD '11, pages 404–412, New York, NY, USA, 2011. ACM. ISBN 978-1-4503-0813-7. doi: 10.1145/2020408.2020474. URL http://doi.acm.org/10.1145/2020408.2020474.

[64] H. Nguyen, A. Fuxman, S. Paparizos, J. Freire, and R. Agrawal. Synthesizing products for online catalogs. *Proceedings of the VLDB Endowment*, 4(7):409–418, 2011.

[65] Nilesh Dalvi, Ravi Kumar, and Mohamed Soliman. Automatic wrappers for large scale web extraction. *Proc. VLDB Endow.*, 4(4):219–230, January 2011. ISSN 2150-8097. URL http://dl.acm.org/citation.cfm?id=1938545.1938547.

[66] Luciano Barbosa, Srinivas Bangalore, and Vivek Kumar Rangarajan Sridhar. Crawling back and forth: Using back and out links to locate bilingual sites. In *IJCNLP*, pages 429–437, 2011.

[67] Valter Crescenzi, Paolo Merialdo, and Paolo Missier. Clustering web pages based on their structure. *Data & Knowledge Engineering*, 54(3):279–299, 2005.

[68] R. Gupta and S. Sarawagi. Answering table augmentation queries from unstructured lists on the web. *Proceedings of the VLDB Endowment*, 2(1):289–300, 2009.

[69] Y. Wang and J. Hu. A machine learning based approach for table detection on the web. In *Proceedings of the 11th international conference on World Wide Web*, pages 242–250. ACM, 2002.

[70] Emilio Ferrara, Pasquale De Meo, Giacomo Fiumara, and Robert Baumgartner. Web data extraction, applications and techniques: A survey. *CoRR*, abs/1207.0246, 2012.

[71] Nicholas Kushmerick. Wrapper induction: efficiency and expressiveness. *Artif. Intell.*, 118(1-2):15–68, April 2000. ISSN 0004-3702. doi: 10.1016/S0004-3702(99)00100-9. URL http://dx.doi.org/10.1016/S0004-3702(99)00100-9.

[72] 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. ISBN 1-55860-615-7. URL http://dl.acm.org/citation.cfm?id=645925.671350.

[73] Stephen Soderland. Learning information extraction rules for semi-structured and free text. *Mach. Learn.*, 34(1-3):233–272, February 1999. ISSN 0885-6125. doi: 10.1023/A:1007562322031. URL http://dx.doi.org/10.1023/A:1007562322031.

[74] Valter Crescenzi, Giansalvatore Mecca, and Paolo Merialdo. Roadrunner: Towards automatic data extraction from large web sites. In *VLDB*, pages 109–118, 2001.

[75] Xian Li, Xin Luna Dong, Kenneth Lyon, Weiyi Meng, and Divesh Srivastava. Truth finding on deep web: Is the problem solved. *Proceedings of the VLDB Endowment*, 2013.

[76] G.M. Weiss and F.J. Provost. Learning when training data are costly: The effect of class distribution on tree induction. *J. Artif. Intell. Res. (JAIR)*, 19:315–354, 2003.

[77] Michael J. Cafarella, Alon Y. Halevy, Daisy Zhe Wang, Eugene Wu, and Yang Zhang. Webtables: exploring the power of tables on the web. *PVLDB*, 1(1):538–549, 2008.

[78] Michael J. Cafarella, Alon Y. Halevy, Yang Zhang, Daisy Zhe Wang, and Eugene Wu. Uncovering the relational web. In *WebDB*, 2008.

[79] Anish Das Sarma, Lujun Fang, Nitin Gupta, Alon Halevy, Hongrae Lee, Fei Wu, Reynold Xin, and Cong Yu. Finding related tables. In *Proceedings of the 2012 ACM SIGMOD International Conference on Management of Data*, SIGMOD '12, pages 817–828, New York, NY, USA, 2012. ACM. ISBN 978-1-4503-1247-9. doi: 10.1145/2213836.2213962. URL http://doi.acm.org/10.1145/2213836.2213962.

[80] Michael J. Cafarella, Alon Y. Halevy, and Nodira Khoussainova. Data integration for the relational web. *PVLDB*, 2(1):1090–1101, 2009. URL http://dblp.uni-trier.de/db/journals/pvldb/pvldb2.html#CafarellaHK09.

[81] Hoa Nguyen, Ariel Fuxman, Stelios Paparizos, Juliana Freire, and Rakesh Agrawal. Synthesizing products for online catalogs. *Proc. VLDB Endow.*, 4(7):409–418, April 2011. ISSN 2150-8097. URL http://dl.acm.org/citation.cfm?id=1988776.1988777.

[82] Xin Luna Dong, Barna Saha, and Divesh Srivastava. Less is more: selecting sources wisely for integration. In *Proceedings of the 39th international conference on Very Large Data Bases*, PVLDB'13, pages 37–48. VLDB Endowment, 2013. URL http://dl.acm.org/citation.cfm?id=2448936.2448938.

[83] Disheng Qiu and Lorenzo Luce. Extraction and integration of web sources with humans and domain knowledge. In Chin-Wan Chung, Andrei Z. Broder, Kyuseok Shim, and Torsten Suel, editors, *23rd International World Wide Web Conference, WWW '14, Seoul, Republic of Korea, April 7-11, 2014, Companion Volume*, pages 1295–1298. ACM, 2014. doi: 10.1145/2567948.2579707. URL http://doi.acm.org/10.1145/2567948.2579707.

[84] Valter Crescenzi, Paolo Merialdo, and Disheng Qiu. Wrapper generation supervised by a noisy crowd. In Reynold Cheng, Anish Das Sarma, Silviu Maniu, and Pierre Senellart, editors, *Proceedings of the First VLDB Workshop on Databases and Crowdsourcing, DBCrowd 2013, Riva del Garda, Trento, Italy, August 26, 2013*, volume 1025 of *CEUR Workshop Proceedings*, pages 8–13. CEUR-WS.org, 2013. URL http://ceur-ws.org/Vol-1025/research1.pdf.

[85] Rolando Creo, Valter Crescenzi, Disheng Qiu, and Paolo Merialdo. Minimizing the costs of the training data for learning web wrappers. In Marco Brambilla, Stefano

Ceri, Tim Furche, and Georg Gottlob, editors, *Proceedings of the Second International Workshop on Searching and Integrating New Web Data Sources, Istanbul, Turkey, August 31, 2012*, volume 884 of *CEUR Workshop Proceedings*, pages 35–40. CEUR-WS.org, 2012. URL http://ceur-ws.org/Vol-884/VLDS2012_p35_Creo.pdf.