HANOI UNIVERSITY OF SCIENCE AND TECHNOLOGY

Master's Thesis in Data Science and Artificial Intelligence

Weak Supervision Learning for Information Extraction

NGUYEN HOANG LONG

Long.NH202189M@sis.hust.edu.vn

Supervisor: Dr. Tran Viet Trung _____

Department: Information System

Ha Noi, 07/2022

Declaration of Authorship and Topic Sentences

1. **Personal information**
   Full name: Nguyen Hoang Long
   Phone number: 096 320 7903
   Email: Long.NH202189M@sis.hust.edu.vn
   Major: Data Science and Artificial Intelligence

2. **Topic**

   Weak Supervision Learning for
   Information Extraction

3. **Contributions**

   - We introduce a new data collection system that uses machine learning to automatically extract information from websites without knowing the structure.
   - We apply the Weak Supervision learning method in building training data set for Information Extraction problem to reduce costs labeling.

4. **Declaration of Authorship**
   I hereby declare that my thesis, titled "Weak Supervision for Information Extraction", is the work of myself and my supervisor Dr. Tran Viet Trung. All papers, sources, tables, ... used in this thesis have been thoroughly cited.

5. **Supervisor confirmation**
   .....
   .....
   .....

Ha Noi, July 2022
Supervisor

Dr. Tran Viet Trung

Acknowledgments

I would like to thank my supervisor, Dr. Tran Viet Trung for guiding and helping me during this research. I would also like to thank the professors at the School of Information and Communication Technology, Hanoi University of Science and Technology, especially the instructors who guided me throughout this master's course.

I would like to thank Cengroup for creating favorable conditions and working environment for me to carry out this research.

I am grateful for my family, friends, and colleagues who have always supported me to complete my Master's program.

Abstract

Data collection system is a system that plays an important role in helping businesses and data laboratories to actively exploit data from websites on the Internet. With the rapid development of the Internet today, data collection systems based on information extraction rules for each website have revealed weaknesses in expanding information exploitation on more websites. To solve this problem, we introduce a new design for our data collection system called AI Crawler, which allows us to automatically detect and extract information from multiple websites without defining the structure in advance. In this study, we also apply weak supervision in data labeling for the Information Extraction problem, thereby significantly reducing the cost of data labeling.

Keywords: Information Extraction, Weak Supervise Learning.

Author

Nguyen Hoang Long

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Problem overview

As a leading real estate service business company, Cengroup follows the real estate market closely, which helps us decide on proper business strategies for any situation that arises. Working in the data technology department, our responsibility is to make sure the real estate market information is up-to-date. To do so, we developed a Scrapy Crawler, crawling data from several large Websites. However, when it comes to scaling it up to the increasing number of Websites, we can no longer program to extract data from every Website as before. Therefore, we upgraded the Scrapy Crawler system to a smarter version called the AI Crawler. In this update, we integrated several machine learning models, allowing the crawler to automatically classify web pages and extract information.

In the process of building machine learning models for the AI Crawler System, especially the Information Extraction Model, we ran into a training data hurdle: The data is unlabeled, while manual labeling is very expensive. We have reviewed and considered several methods for reducing labeling costs, such as Active Learning, Transform Learning, and Weak Supervision Learning. Ultimately, we chose Weak Supervision because of its suitability for this particular problem.

1.2 Goals of the thesis

The goals of the thesis are:

- Building an AI Crawler that automatically detects and extracts information from many different Websites.
- Applying Weak Supervision in data labeling to effectively reduce labeling costs.

1.3 Thesis contributions

The work in this thesis proposes improvements to the existing system in information extraction. Generally, the research approach can be applied to similar systems and implemented in other Domains.

1.4 Main content and Structure of the thesis

In this thesis, we introduce the problems of the Scrapy Crawler system that reduce system scalability with an increasing number of Websites and our proposal for improvement in Chapter 2. In Chapter 3, Chapter 4 and Chapter 5, we present the machine learning models, which we have built to improve the system, in detail. Especially in Chapter 4 and Chapter 5, we provide an in-depth description of the Weak Supervision method in building the training data set for the machine learning model. Finally, Chapter 7 gives the conclusions about the achieved results and future development directions.

Chapter 2

Problems and Solutions

In this chapter, we present the problems associated with the Scrapy Crawler and solutions to them.

2.1 Problems with the Scrapy Crawler

The Scrapy Crawler is designed based on Scrapy [1] platform (Fig 2.1). Within this architecture, each Website uses a separate data extraction code called Spider, in which we develop programs to retrieve data from a page. The source code at Listing 2.1 is an example. In the parser function of WebSpider, we defined the rules to check if a page has the required data and then extract data by using css-selector [2]. There are three problems with this design:

- When we needed to crawl a new Website, we had to write a new Spider to execute it.
- When a Website changes its appearance, we have to reprogram the Spider for it.
- When more information is needed, we have to update all the Spiders.

These problems make the system difficult to scale to more Websites. Furthermore, scaling in this approach makes maintenance more expensive.

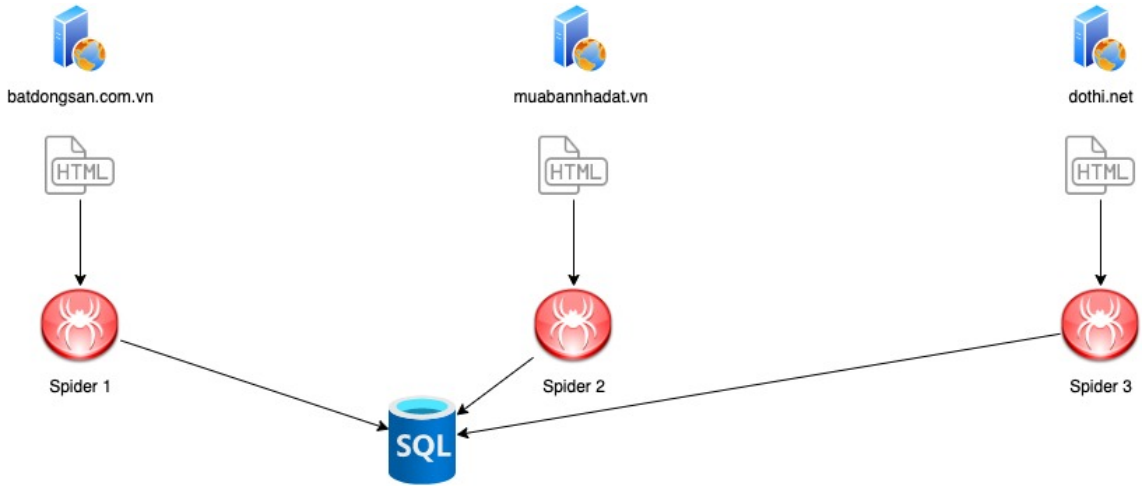

Figure 2.1: Scrapy Crawler Architecture

```
1  import scrapy
2  class WebSpider(scrapy.Spider):
3      name = 'web_spider'
4      start_urls = [
5          'https://example.com/',
6      ]
7
8      def parse(self, response):
9          // Check this page has needed data
10         if response.css('div.real-estate').get() is not None:
11             yield {
12                 'address': response.css('div.real-estate > div.address::text').get(),
13                 'price': response.css('div.real-estate > div.price::text').get(),
14                 'acreage': response.css('div.real-estate > div.acreage::text').get()
15             }
16
17         next_pages = response.css('li.next a::attr("href")').all()
18         for next_page in next_pages:
19             yield response.follow(next_page, self.parse)
```

Listing 2.1: WebSpider example code

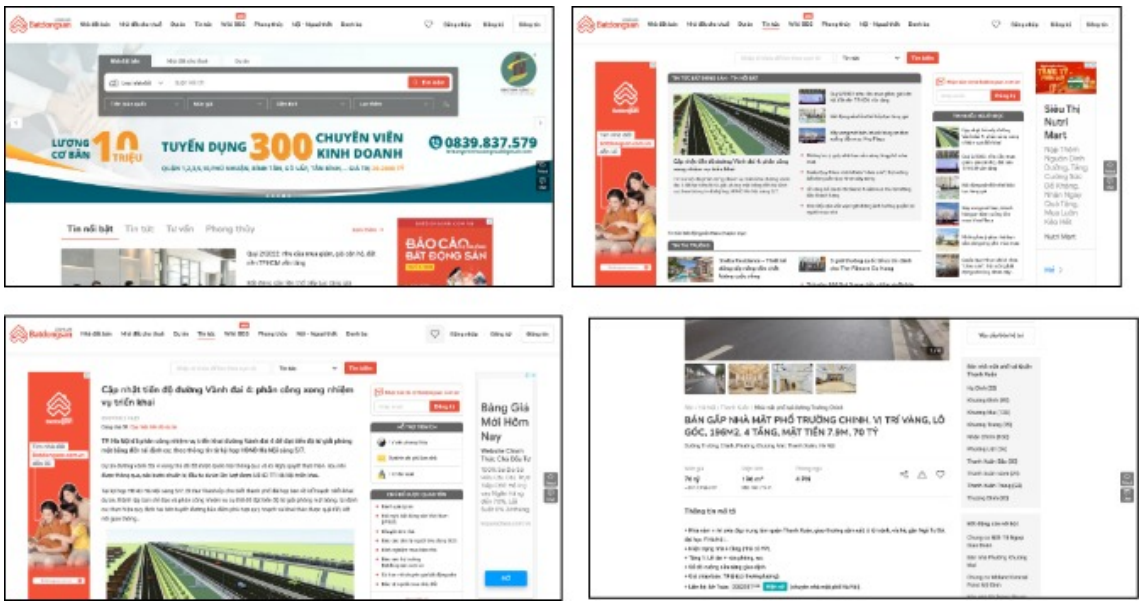

Figure 2.2: A website has many pages, hence many categories

2.2 Requirements for the AI Crawler

In the AI Crawler, we set up the crawler to be able to classify Pages and extract information automatically, with the following requirements:

- The AI Crawler needs to know which Pages needed to get data from since there are many different types (Categories) of Pages on a Website. For instance, in this system the AI Crawler only needs to get data from real estate classified advertising sites (REAL_ESTATE_PAGE Category), hence, we need to program so that the crawler can distinguish between classified advertising sites and others (Fig 2.2). For better optimization, the crawler needs to distinguish the Page's URL before downloading that Page. Therefore, the Page Classification has to be based on the URL of the Page.
- The AI Crawler should be able to automatically extract information from the Raw HTML without the predefined rules as in Scrapy Crawler.

2.3 Solutions analysis

To fulfill the above requirements, our approach is to use machine learning models for the AI Crawler. In this section, we present the solutions we

have experimented with and analyze the advantages and drawbacks of these solutions.

2.3.1 Page Classification

For Page Classification requirements, we have experimented with building a text classifier model based on the Websites' URLs. The results show that in the trained Websites, the model was fairly good; but when it was tested on other websites, the result was poor. Although preprocessing and normalizing were done on URLs before training, the results were not improved. To explain this, we suppose that the URLs of different websites are 1) various and 2) too short for the model to have enough features to distinguish between them.

One upside to this is that the model was doing well on the trained Websites, so if we use one model for each Website separately, the result could be better. But what about new Websites? We continued experimenting with Page Classification by using Title and Description. This time, the results were good even with new Websites. However, two problems remained:

- Data needs to be retrieved from the Page before we can classify it.
- Some Websites do not have Title and Description.

Regarding the first problem, we have an idea is to train a Page Classification model for all known Websites using Title and Description. For new Websites, we will use this model to build a training dataset with URLs, then train a URL Classification Model by using this dataset for the new Website.

In terms of the second problem, we found a way to get the Main Text of the Page to replace the Title and Description. We will present this solution in more detail in Chapter 3

To summarize, the solution to the Page Classification problem is as follows:

- We build a Content Classification Model based on Title, Description, or Main Text on each Page of all known Websites.
- With this model, we will build a training dataset with URL for each new Website then train URL Classification Model using this dataset for them.

2.3.2 Extract information from web page

Regarding the second requirement, AI Crawler must extract information from the Raw HTML automatically without the predefined rules. To do this, the crawler needs to use an Information Extraction Model for HTML data. When building this model, we ran into a training data hurdle where the data was unlabeled.

To solve this data labeling problem, we used Weak Supervision, a training method that allows different labeling methods to be combined to form a better-labeled dataset. We present the details for building this model in Chapter 4.

When experimenting with this model for data of real estate detail Pages, we saw that in the same Page, the model extracted many Values of the same Label. For example, in a real estate detail Page, the model returned many different results for Addresses, Prices, and Acreages – the true information was uncertain. After further examination, we learned that the reason was that the model extracted information from multiple different Parts of the Page.

An HTML page generally has many Parts such as header, footer, sidebar, main... However, true information only appears in the main Part. Therefore, to ensure model accuracy, instead of including the Raw HTML for extraction, we only included the Main HTML. We will present this solution in more detail in Chapter 3.

2.4 Overall solution

In summary, to solve the problems of the Scrapy Crawler, the AI Crawler needs to add the following machine learning models:

- Main Content Detection Model:
  - Input: Raw HTML
  - Output: Main Text and Main HTML
- Content Classification Model:

  - Input: Title and Description or Main Text.
  - Output: Category
- URL Classification Model
  - Input: URL of one Website
  - Output: Category
- Information Extraction Model
  - Input: Main HTML
  - Output: Extracted Data

2.4.1 System Architecture

The AI Crawler includes two components (Fig 2.3) as follows:

- Website Explorer: Explores new Website and trains URL Classification Model for each new Website.
- Parser Crawler: Crawls data from known Website, gets Extracted Data from the Page, then saves them to the database.

The data flow in AI Crawler system (Fig 2.4) is as follows:

1. When a user needs to collect data from a new Website, the Website's URL will be sent to Website Explorer.
2. Website Explorer starts to crawl data from the new Website, and saves the Raw HTML of each Page to its database. When the number of Pages on the Website is enough, it will halt.
3. With the collected data, the Website Explorer will use the Main Content Detection Model to determine the Main Text from the Raw HTML of each Page, then use the Content Classification Model to classify them into two Categories: real_estate_site and outlier_site.
4. The Website Explorer uses this dataset to train the URL Classification Model for a new domain by using URL Classification Trainer.

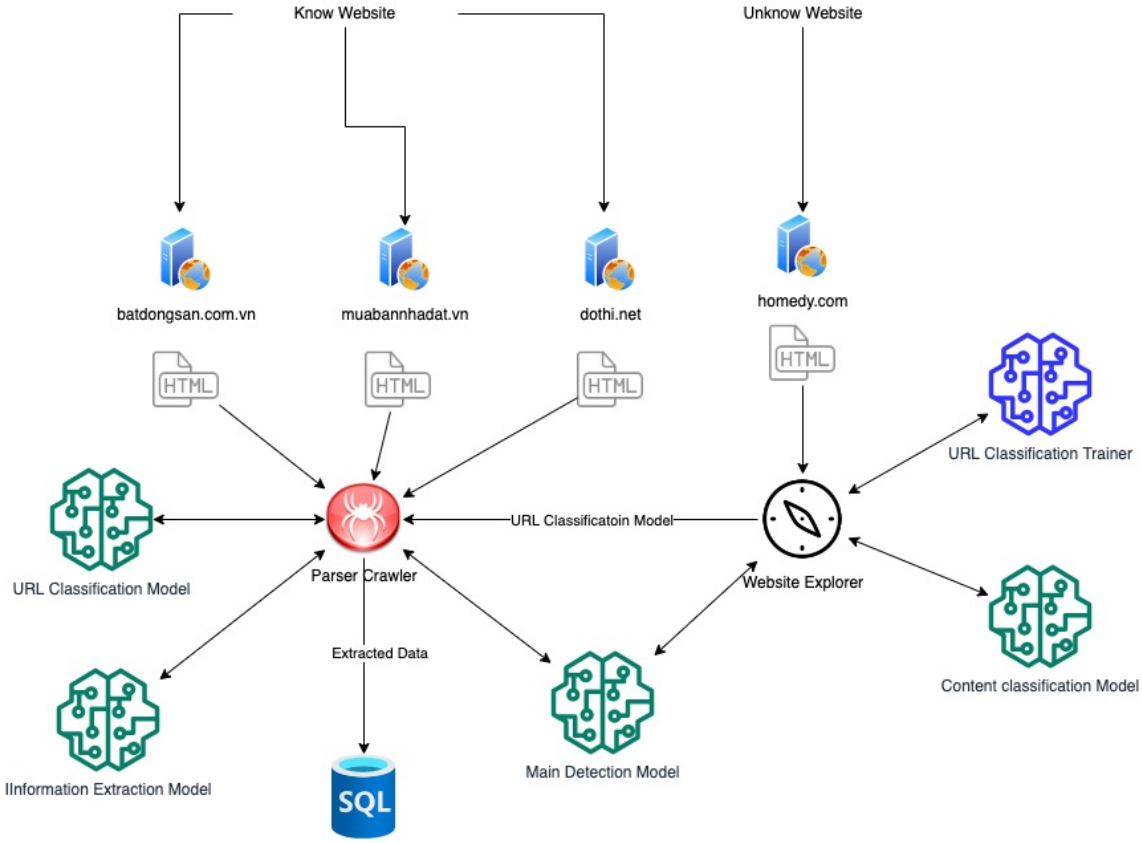

Figure 2.3: AI Crawler Architecture

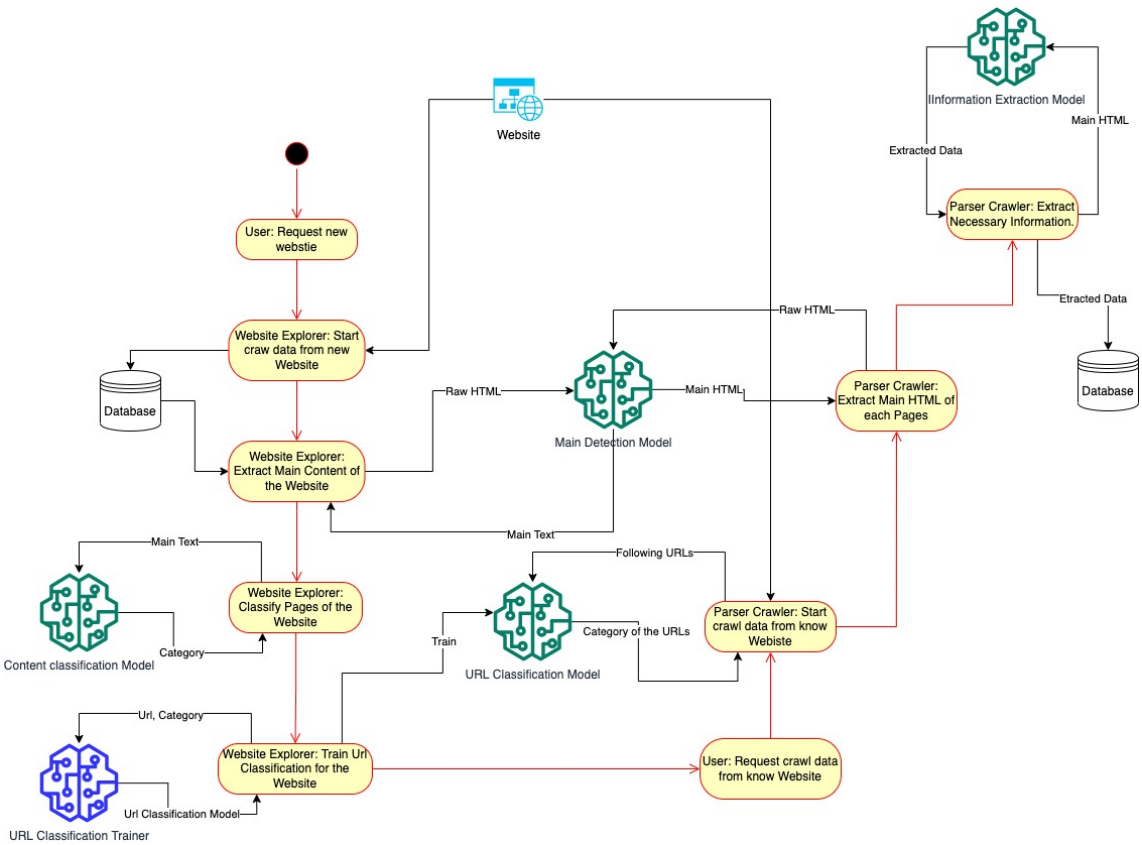

Figure 2.4: AI Crawler Flow

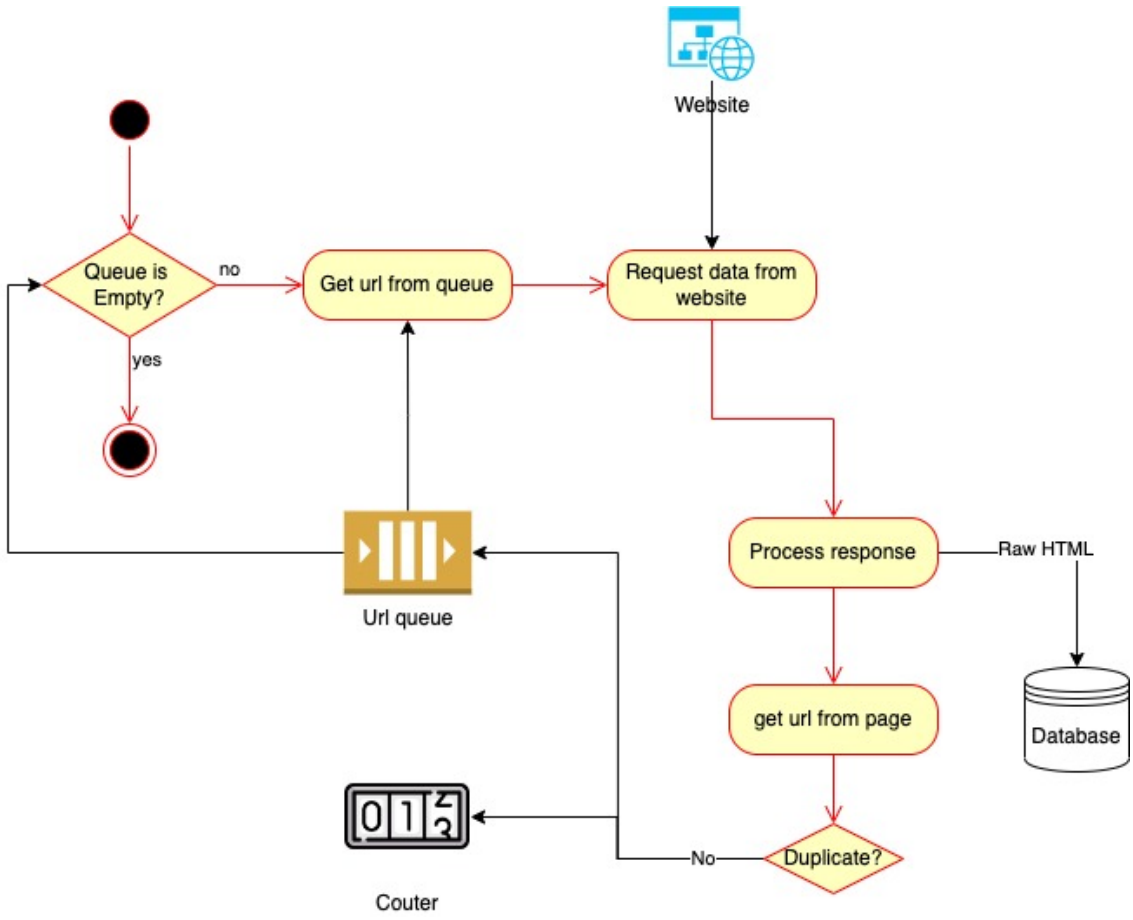

Figure 2.5: Website Explorer survey new Website

5. The user checks the list of explored Websites on the AI Crawler, and starts the Parser Crawler with the Website they want. Parser Crawler will launch a new process to collect data from the requested Website.

6. Parser Crawler uses the URL Classification Model to classify the following URLs and places them in a priority queue, where the URLs of real estate detail Page have higher priorities.

7. At each real estate detail page, the Parser Crawler uses the Main Content Detection Model to extract the Main HTML from the Raw HTML for the Page then uses Information Extraction Model to get the Extracted Data and stores it in a relational database.

2.4.2 Website Explorer

Website Explorer uses a Queue to store the URLs to be processed. When the response is returned from the Website, the Raw HTML of the Page will be stored together with its URL. The URLs that appear in the Page will

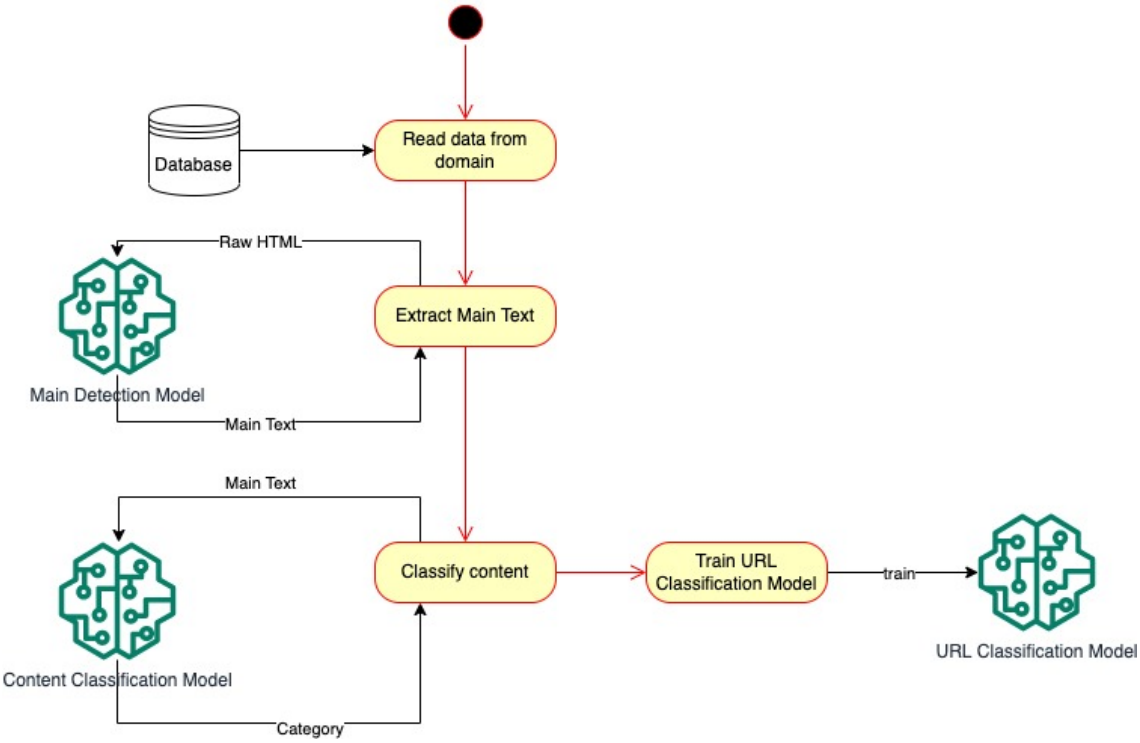

Figure 2.6: Website Explorer train URL Classification Model

be retrieved and checked for duplicates; if not, they will be pushed to the Queue. A counter will count the number of URLs that have been pushed to the Queue; when the counter reaches a certain number, the Queue will stop receiving URLs (Fig 2.5).

After having collected a sufficient number of Pages of the new Website. Website Explorer will read all the Pages stored in the database, use the Main Content Detection Model to get the Main Text, and then use the Content Classification Model to classify the Pages. Website Explorer uses this dataset to train the URL Classification Model for the new Website. (Fig 2.6)

2.4.3 Parser Crawler

Parser Crawler uses a priority queue to store URLs to be processed, while the real estate detail URLs will have higher priority. In particular, when Parser Crawler receives the response, it will use the URL Classification Model taken from the Website Explorer to check the URL, if it is a real estate detail Page, it uses Main Content Detection Model to get the Main HTML from Raw HTML of the Page, then uses Information Extraction Model to get Extracted Data from Main HTML, and finally saves it to the database. Parser Crawler continues to get the URLs in the Raw HTML; if there are no duplicates, it

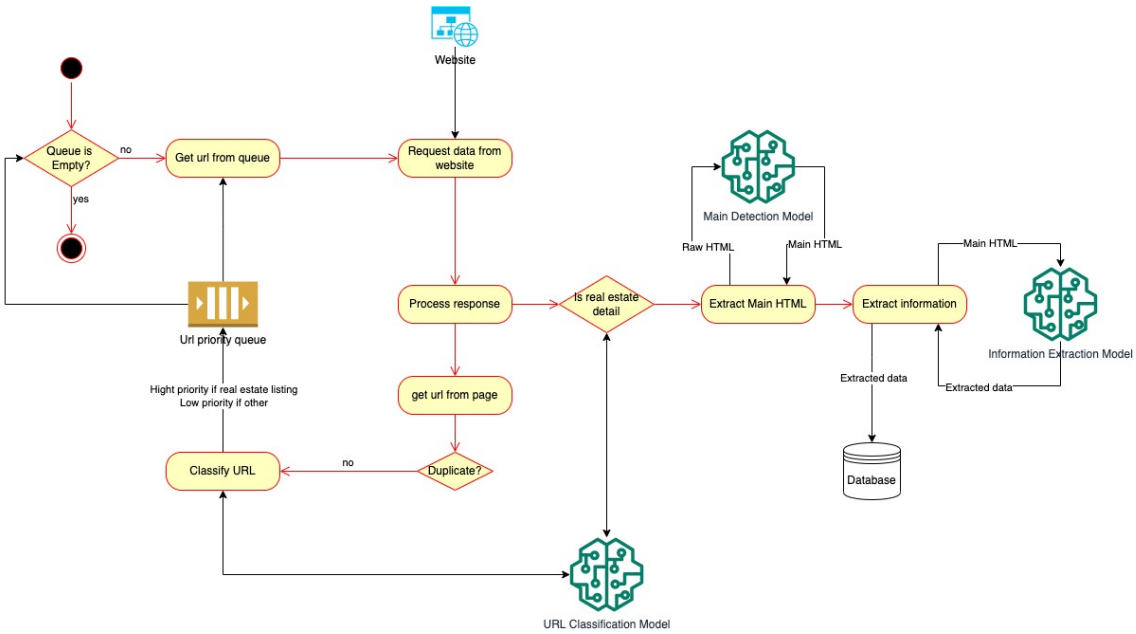

Figure 2.7: Parser Crawler flow

will use the URL Classification Model to classify the URL, and push it to the Queue with the URL of the real estate detail with higher priority.

Chapter 3

Page Classification

This chapter will present the first three models of the system, including the introduction of related research, training and evaluation of the model.

3.1 Main Content Detection Model

3.1.1 Requirements

As mentioned in Chapter 1, the Main Content Detection Model is employed to detect the Main Text and the Main HTML of the Raw HTML, in both of the components of the AI Crawler. The Website Explorer utilizes it to extract the Main Text in the Raw HTML without both the Title and Description. In other cases, the Parser Crawler uses it to extract the Main HTML from the Raw HTML. It is important to mention that the Main Content Detection Model must be able to work on many different Websites, including the new Websites.

3.1.2 Problem analysis and Solution direction

Fig 3.1 presents a common example of a Page, in which we can see that the Page has many Parts such as header, footer, sidebar, navigation bar, main... With this observation, we can easily distinguish the Part in the Page, which leads us to the first approach: Detect Main is based on display factor.

Following this approach, we have surveyed several studies, and finally,

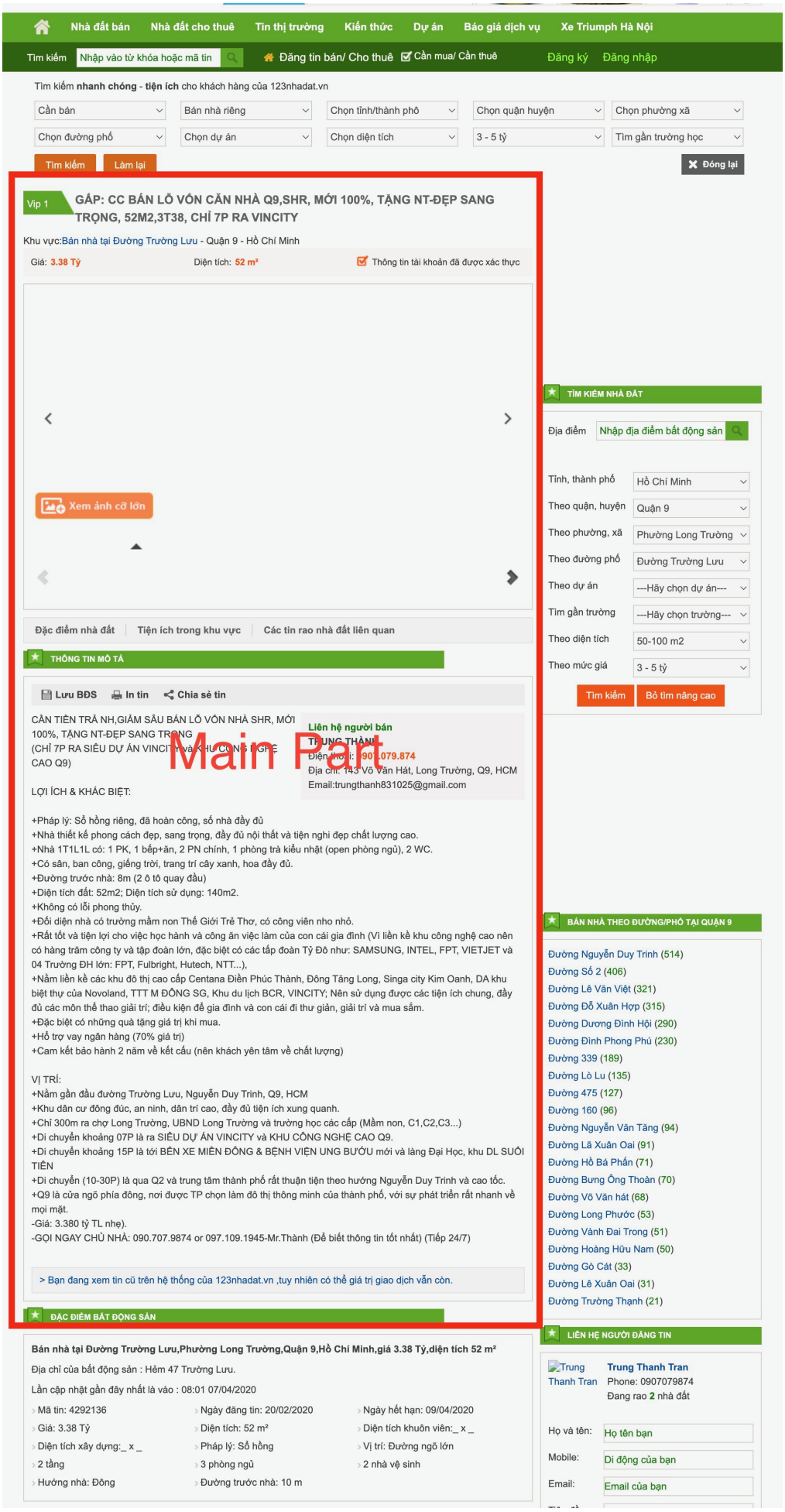

Figure 3.1: A Page Example.

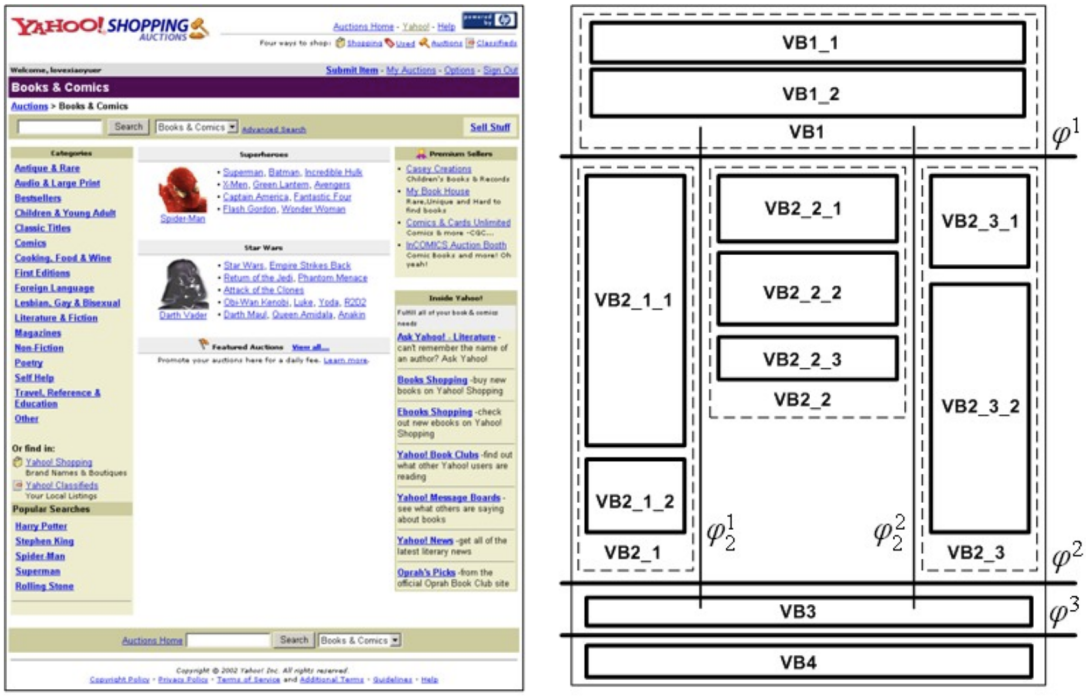

Figure 3.2: VIPS Extractor [4]

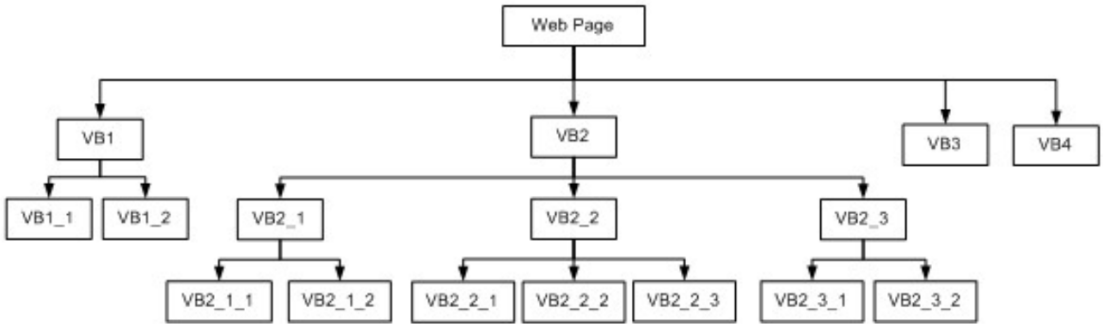

Figure 3.3: VIPS Semantic Structure [4]

selected VIPS [4] for testing. VIPS is an algorithm that extracts the semantic structure of a web page based on its visual presentation (Fig 3.2). VIPS divides a web page into blocks and then calculates the visible distance between them to calculate the Degree of Coherence of each block. This number indicates how distinct one block is from another. Finally, the blocks will be organized into a semantic tree that represents the structure of the page.

The advantage of this algorithm is that there is no need to train the model; we can simply put the required data on the Page to obtain the hierarchy structure of the Page. But this algorithm has a weakness that makes it difficult to apply: the processing time is too long. To acquire input for this algorithm, the system must render the entire page in the browser. Then the system will

have to calculate the position of each element on the page to be input data for the VIPS algorithm. Through actual benchmarks, we measured the average time to process a page from 8 to 12 seconds, depending on the network speed, of which about 60% of the time is to render the Page. For comparison, the Page rendering time will be three times longer than solely getting the Raw HTML.

Thus, the display factor-based approach is generally unsuitable for our problem because it requires rendering the web page on the browser, which significantly increases the crawl time. From here, we looked into another approach based on structure and text factors. In this direction, the input is simply Raw HTML data. After considering several studies in this direction, we have selected two candidates for testing: Dragnet [9], and Web2text [13].

3.1.3 Related Work

Dragnet

Dragnet uses an approach based on combining a single block definition and embedding the entire CETR [14] algorithm inside a machine learning model. Dragnet uses a set of features heuristically designed to capture semantic information in the HTML code left behind by programmers. In this paper, they note that many of the id and class attributes in modern HTML tags include tokens such as "comment", "header" and "nav". These descriptive names are used by programmers when writing CSS and Javascript, and since they are chosen to be meaningful to the programmer, they embed some semantic information about the block's content.

In the preprocessor stage, the HTML Document was turned into a DOM (Document Object Model) and then into a block tree structure, with each block being an element containing text (ignoring non-text elements). Use Cleaneval [3] to mark the main part for data.

In the extract feature stage, a set of features were utilized, including the two most predictive shallow text (ST) described in [7]. Next, the CETR algorithm is used to smooth the ratio of text length to tag count of each block; the results returned from this algorithm are used as features in Dragnet's model. Finally, the semantic features from the id and class of the elements in

the block were also used. Dragnet's network architecture and model, however, were not mentioned in their paper.

Web2text

Web2text [13] investigates the problem of Boilerplate removal for HTML data. Using the textual and structural features of the HTML format, use a CNN network to train the model.

The task of separating the main text in a Web page from the remaining content is known in the literature as "boilerplate removal", "Web page segmentation", or "content extraction". Established popular methods for this problem use rule-based or machine learning algorithms. The most successful approaches first perform splitting of an input Web page into text blocks, followed by binary labeling of each block as either main content or boilerplate.

Web2text proposes a hidden Markov model on top of neural potentials for the task of boilerplate removal. Leverage the representational power of convolutional neural networks (CNNs) to learn unary and pairwise potentials over blocks in a page-based on complex non-linear combinations of DOM-based traditional features. At prediction time, web2text use the Viterbi algorithm [5] to find the most likely block labeling by maximizing the joint probability of a label sequence.

The approach that Web2text uses is to refer to the sequence segmentation problem, whereby an HTML page is converted into a sequence of Blocks, with each block representing an element containing text in the HTML file. Thus, the problem becomes sub-classing whether a block belongs to the main content or not. The data processing pipeline in Web2text can be summarized as follows (Figure 3.4):

1. Preprocessing: HTML data will be converted to DOM (Document Object Model) tree. From there, each element containing text will be represented by a block (Figure 3.5). The entire HTML page becomes an ordered block sequence. The leaves of the Collapsed DOM tree of a Web page form an ordered sequence of blocks to be labeled by using Cleaneval [3].

2. Feature extraction: For each block, web2text extract a number of DOM tree-based features. Two separate convolutional networks operating on this sequence of features yield two respective sets of potentials: unary

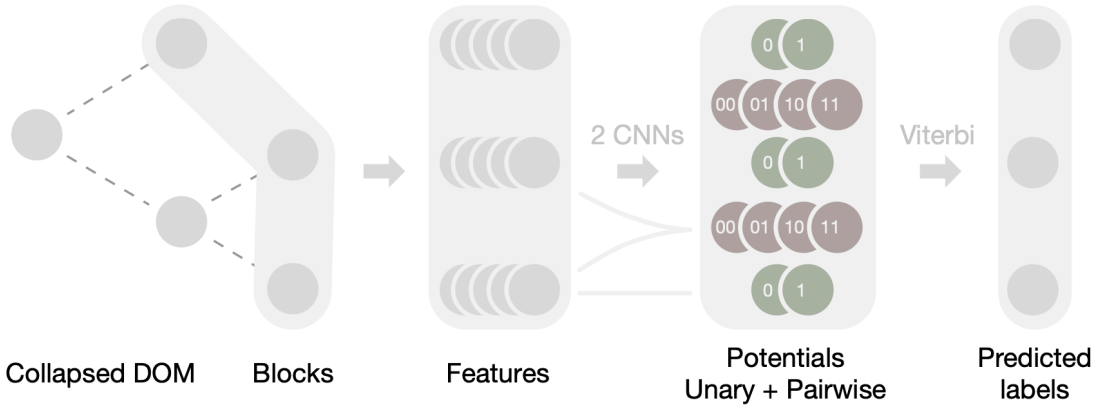

Figure 3.4: Web2text pipeline [13]

potentials for each block and pairwise potentials for each pair of neighboring blocks. These define a hidden Markov model. Web2text uses two features, including Block Feature and Edge Feature

- Block Feature includes text features, block tags, block position on the page, and stop word rate on the page. Block features capture information on each block of text on a page. They are statistics collected based on the block's CDOM node, parent node, grandparent node, and the root of the CDOM tree, e.g., "the node is a <p> element", "average word length", "relative position in the source code", "the parent node's text contains an email address".
- Edge Feature includes the features between a pair of 2 blocks close to each other, including spacing in the page, having the same ancestor or not. Edge features capture information on each pair of neighboring text blocks. Define the tree distance of two nodes as the sum of the number of hops from both nodes to their first common ancestor. The first edge features we use are binary features corresponding to a tree distance of 2, 3, 4, and > 4. Another feature signifies if there is a line break between the nodes in an unstyled HTML page.

3. Training: Web2text uses CNN network model with Block and Edge features, with Unary CNN for Block Feature and Pairwise Potentials for Edge Feature. After that, using the Viterbi algorithm to find optimal labeling that maximizes the total sequence probability as predicted by the neural networks.

4. Post-processing: Using the Web2text model, we were able to extract

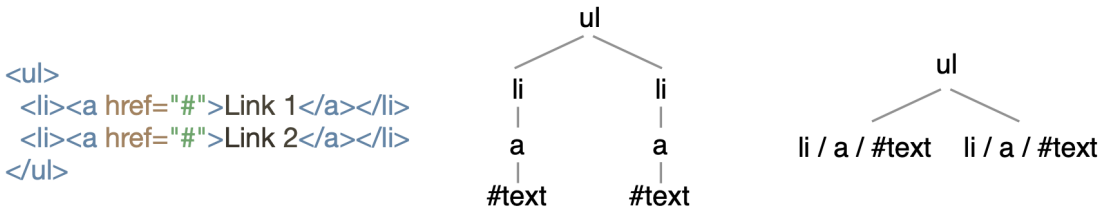

Figure 3.5: Collapsed DOM procedure example [13]

Main Text of the Raw HTML. However, we still need to extract the Main HTML from Raw HTML, which requires us to modify the source code of Web2text. Essentially, the result from the web2text model depends on whether the label of each block belongs to the main part or not: If it is the main part, the model will return the text of that block. We interfered with this process to make the model return the CSS Selector of each block. With the CSS Selector of the blocks, we can easily get the main Part and generate the Main HTML for the page.

Cleaneval

Cleaneval [3] is a shared task and competitive evaluation on the topic of cleaning arbitrary web pages, with the goal of preparing web data for use as a corpus for linguistic and language technology research and development. The first exercise took place in 2007. This section describes how it was set up, the results, and the lessons learned.

In Dragnet, Web2text, and this thesis, Cleaneval was used as a preprocessing stage for marking the Main Text in a Raw HTML

3.1.4 Dataset

By crawling data from many different real estate Websites we had a Page dataset of about 80,000 Pages from 17 different Websites in real estate Domain (see table 3.2), in which each Page contains Raw HTML and URL.

The processing steps to get training data for Dragnet and Web2text are as follows:

1. First, on each Website, we relied on their URL characteristics or their Raw HTML layout characteristics, to classify the Page into two cate-

Table 3.1: Initial data.

Website	Raw	Usable	REAL_ESTATE_PAGE	OTHER_PAGE
www.nhadatviet247.net	8433	7991	3153	4838
ancu.me	7776	7604	3598	4006
alonhadat.com.vn	6620	6168	2495	3673
mogi.vn	6507	6237	3554	2683
timdat.net	6506	6044	3956	2088
bds123.vn	6004	5898	3835	2063
123nhadat.vn	5498	5051	3348	1703
nhadatdongnai.com	5379	5055	2533	2522
nhadatbacninh.vn	5164	4886	2695	2191
timmuanhadat.com.vn	3693	3461	2574	887
muabannhadat.tv	3212	2826	1566	1260
batdongsancantho.vn	3105	2779	1329	1450
batdongsandanang.net	3073	3000	894	2106
chothuenha.me	2599	2405	1124	1281
homedy.com	2108	2016	1549	467
nhadat24h.net	1653	1408	1198	210
muabanchinhchu.net	1613	1123	208	915
Sum	78943	73952	39609	34343

gories include REAL_ESTATE_PAGE and OTHER_PAGE (view Fig 3.1).

2. Second, we converted Raw HTML of the Page to the DOM tree, in the REAL_ESTATE_PAGE Category of each Website we used Start Selector and End Selector to mark the main Part of the Raw HTML, then got text of the main Part to had the Main Text. In fact, we retrieved a clean evaluation dataset as input for Dragnet and Web2text.

3. To train the models, we split the dataset into two sets, including train and test, with each set containing Pages from different Websites. Each time we tested, we used different combinations of Websites. Because the number of Pages in a Website was not the same, when training the model, we got the same number of Page for each Website, the details are in Table 3.2

3.1.5 Results and Discussion

Evaluation Method

To evaluate the results, we compare the main text results obtained from the model when the raw HTML of a page is passed with the correct main

Table 3.2: Data use for train Main Content Detection Model

	Train set		Test set	
Set	Number Page	Number Website	Number Page	Number Website
1	1000	10	490	7
2	960	12	500	5
3	999	9	496	8
4	1000	5	492	12
5	999	3	490	14

text. We use two metrics to evaluate the model:

- Similarity between results and expectations: Coincidence. We took the average value across the data set.
- Percentage of pages with Coincidence greater than 85% in the test set as Coverage.

Results and Discussion

Based on the test results in table 3.3 and table 3.4, we have the following findings:

- Both models have a similar Average of Coincidence when the training set has many websites. Web2text has a higher Coverage which proves that it works more steadily than Dragnet.
- When gradually reducing the number of Websites in the training set (still keeping the number of Pages used for training). Dragnet showed a decrease in Coincidence and Coverage, but Web2text did not experience a significant reduction in these parameters. This indicates that Web2text can work better on new Websites.
- Based on this result, we decided to use the Web2text model as the Main Content Detection Model.

Table 3.3: Result of the Dragnet Model

Set	Average of Coincidence	Coverage
1	0,85	0,65
2	0,88	0,67
3	0,79	0,61
4	0,72	0,59
5	0,75	0,56

Table 3.4: Result of the Web2text Model

Set	Average of Coincidence	Coverage
1	0,84	0,76
2	0,83	0,78
3	0,9	0,74
4	0,84	0,79
5	0,86	0,75

3.2 Content Classification Model

3.2.1 Solution analysis

Content Classification Model is used in Website Explorer to classify text of the Page on the Website in to one of the Categories of the Domain. Text of a Page may be Title and Description or Main Text of the Page. In this system, the Content Classification Model needs to return REAL_ESTATE_PAGE or OTHER_PAGE.

This problem belongs to the Text Classification Problem, which is the most fundamental and essential task in Natural Language Processing (NLP) [8]. It solves many problems in NLP, such as sentiment analysis, topic labeling, question answering, and dialog act classification. In this thesis, we use Fasttext to build text classification models.

3.2.2 Brief about Fasttext

Fasttext [6] is a machine learning library for text data developed by Facebook. With Fasttext we can build machine learning models for text data at a fast speed. The two most prominent functions of Fasttext are building word vectors (unsupervised) and text classification models (supervised).

Table 3.5: Data for Content Classification Model

Dataset	Sum	Number Website	REAL_ESTATE_PAGE	OTHER_PAGE
Train	44993	7	23939	21054
Test	16228	4	9368	6860
Validate	12731	6	6302	6429

Text classification is an important task in Natural Language Processing with many applications such as web search, information retrieval, ranking, and document classification. With the advantage of learning speed, fasttext trains models on large data sets.

Instead of using Bag of words, fasttext uses a bag of n-grams as additional features to capture some partial information about the local word order. This is very efficient in practice while achieving comparable results to methods that explicitly use the order.

3.2.3 Dataset

To build the Content Classification Model, we need to prepare training data including text of Pages and Category for each Page. At this stage, we had a dataset of about 80,000 Pages from 17 different Websites in real estate Domain (see table 3.2). Each Page contains Raw HTML and URL, Main Text and Category. The preparation of data for Fasttext includes the following steps:

- For each Page in dataset, we get Category and text of the Page. If the Raw HTML has the Title and Description, the text of the Page is the concatenation of them. If Raw HTML does not have Title and Description, we will use Main Text to replace.
- We split the data set to three smaller sets: Train, Test, Validate with different Websites (as in table 3.5)).

3.2.4 Results and Discussion

The results in table 3.6 demonstrate that the content classification model has been trained fairly well. When we examined the results in more detail, we found common mistakes including:

Table 3.6: Result of Content Classification Model

	REAL_ESTATE_PAGE	OTHER_PAGE
Prediction	0,93	0,90
Recall	0,92	0,91
F1	0,93	0,90

- Some pages of OTHER_PAGE Category had no Title and Description, the Main Text could not be extracted as well. The Main Content Detection Model was only trained on the REAL_ESTATE_PAGE Category; therefore, it is difficult for the model itself to extract the Main Content from pages that belong to OTHER_PAGE Category. We are improving further to alleviate this issue by adding other Categories into the Main Content Detection Model.
- Some news sites have a similar way of writing Title and Description to REAL_ESTATE_PAGE, thus the model might be confused.

3.3 URL Classification Model

3.3.1 Solution analysis

URL Classification Model is employed in Parser Crawler to classify the URL of a Website in to the Categories of the Domain. These models are trained by the Website Explorer. After having the results of the page classifier of a new Website, we use this result to train an individual URL Classification Model for that website.

Similar to the Content Classification Model, the URL classification problem also belongs to the class of text classification. In the thesis, we also utilize Fasttext to train this model.

3.3.2 Dataset

Because in the system design, the URL Classification Model is trained using the data from the results of the Content Classification Model, we also employ this flow to train and evaluate the model results as follows:

Table 3.7: Result of URL Classification Model

Domain	Sum	Train	Validate	Test	TRUE	F1
nhadatdongnai.com	5055	3033	1011	1011	932	0,92
nhadatbacninh.vn	4886	2932	978	976	857	0,88
timmuanhadat.com.vn	3461	2077	693	691	641	0,93
muabannhadat.tv	2826	1696	566	564	502	0,89

- Use Content Classification Model to classify the Page of the Website in the test set. For each Website, we obtain the URL and Category (that were returned by Content Classification Model).
- For each Website, we split the dataset into three smaller sets: train, test, validate, and then train URL Classification Model using the train set and validate set.
- To validate the result, we compare the result with the expected Category in the test set.

3.3.3 Result and Discussion

The results obtained from the URL Classification Model are closely equivalent to that of Content Classification Model. After reviewing, we found that errors often occur due to the followings:

- The error of the model originates mainly from the URLs being too short or too similar between different Categories.
- The incorrect labels from the Main Content Detection Model also attribute to the overall errors.

Chapter 4

Information Extraction Model: Background

4.1 Requirements

Information Extraction Model is used by Parser Crawler to extract necessary information from the Main HTML of the Page. Extracted Data are returned by model include:

- Address: The address of the real estate being listed has four sub-types:
  - Province
  - District
  - Ward
  - Street
  - Project
- Price
- Acreage
- Surface Width
- Surface Length
- Street Width
- Number of Rooms

- Number of Toilets
- Number of Floors
- Phone
- Submission Date

4.2 Problems analysis and Solutions direction

The requirements of this problem lead us to the Name Entity Recognition (NER) problem. This is a sub-problem of Information Extraction (IE).

Information Extraction (IE) is the extraction of structured information from unstructured or semi-structured text data. IE includes two sub-problems: Name Entity Recognition (NER) and Relation Extraction (RE). NER is to locate and classify unit elements in text into predefined categories such as proper names (names of people, organizations, and places), times,... RE is to extract relationships between entities.

Regarding the problem in this thesis, with the NER problem, we have two approaches: rule-based and machine learning-based. With the rule-based approach, we will observe the characteristics of the data from which the rule is generated. For example: To extract price information on a real estate detail page, the rule given will be a number that is followed by the price unit, preceded by the word “sell”, “rental” or “priced”. The rule-based approach has the advantage of high accuracy, but the disadvantage is that it cannot extract information not already defined in the rule. The second approach with machine learning has the advantage of handling new data but simultaneously proposes the disadvantage that it requires labeled training data. Meanwhile, the cost of data labeling is expensive.

In this thesis, we employed the second approach of machine learning, with the modification that instead of manually labeling data, we use the Weak Supervision method to label data. Weak Supervision allows combining many data sets and different labeling methods, called weak data sources, to get better, new datasets.

At our company, we have some very useful models such as an address ex-

traction model, an information extraction model from real estate description, a language model trained in real estate data, and some data sets such as address tree, address dictionary only, and project list name. Instead of having to label data entirely by hand, we want to take advantage of what we have to reduce the cost of labeling while still ensuring quality for the final model.

One more problem is introduced to the situation: the input data for this model is in the form of HTML, a structured data form. Instead of converting the HTML data into text and handling the NER problem on the text data in the usual way, we want to take advantage of the structural features of the HTML file format to add more characters to the model. To fulfill this requirement, we used the Fontuer framework with modifications in the appropriate source code to solve this problem.

4.3 Weak Supervision

4.3.1 Overview

Labeled training data is an integral component when building a supervised machine learning model. However, a labeled dataset is not always available or can be easily constructed. To solve this bottleneck, there have been many studies with different methods [16] such as:

- Active Learning: This method allows to select the examples with the highest value for the model to train, thereby minimizing the overall cost of labeling.
- Semi-Supervised Learning: The idea of this approach is to use unlabeled data as leverage to improve the performance of the model.
- Transfer Learning: Find a way to transform an existing model into a new one that can be applied to another domain.

Also, with the goal of reducing the cost of data labeling, Weak Supervision takes a different approach. There are many ways to label a dataset, such as heuristics, pre-train models, existing knowledge bases, and 3rd party services. Each of these labeling methods will produce a differently labeled data set with

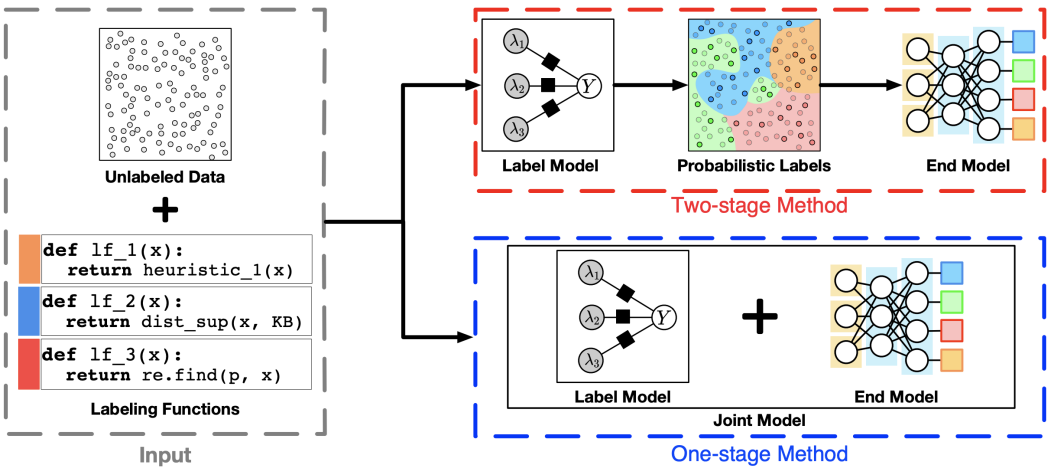

Figure 4.1: Weak Supervision Flow [16]

different accuracy. If we can combine these labeled sets in such a way that they augment each other, we get a new dataset with improved accuracy. This is the idea of weak supervision.

To implement this idea, Weak supervision uses label functions to represent each labeling method, where users can write their own label functions. This is called Programmatic Weak Supervision (PWS) [16]. Each label function takes a sample of data as input and returns the label of that sample. The result of labeling from the label functions will give us an n $\times$ m label matrix where n is the number of examples to be labeled, and m is the number of label functions or the number of weak data sources or label functions. In general, there are two categories of PWS methods, as shown in Fig 4.1:

- One-stage Method: Also known as the joint model, this method builds the final model directly on the label matrix, so when using the final model, one will still have to construct the label matrix from weak data sources before entering the last model.

- Two-stage Method: The method consists of 2 stages. Stage one uses the Label Model to combine the label matrix into one hot hard training label or probabilistic soft training label; this labeled result is then used in stage 2 to train a final model. This method only uses weak data sources in the labeling process and then uses the labeling results to train the final model; weak data sources are entirely unnecessary when using the final model.

4.3.2 Labeling Functions

Using a user-defined label function allows a wide variety of labeling types to be used; some of the more common types of labeling include:

- Heuristic rules: This type of labeling allows the user to define labeling rules based on experience in observing data. This method allows the user to easily adjust the label, thereby affecting the overall result of the labeling result in the desired direction.
- Existing knowledge: Using existing knowledge allows taking advantage of existing knowledge bases, trained models, or provided 3rd party tools for labeling purposes. This is an important resource that greatly supports the labeling process and helps reduce costs.
- Crowd-sourced Labels: Including other weak label sources that can be leveraged on a case-by-case basis, they may be inaccurate, incomplete, and noisy but can still contribute positively to the overall results.

To improve efficiency when designing weak supervision sources, there are three main directions in the generation of LFs [16]:

- Automatic Generation: Start from a small labeled dataset to automatic initialization of label functions simple.
- Interactive Generation: Build an iterative process that allows for gradual improvement of labeling results.
- Guided Generation: Select a small dataset for development with the aim of reducing the effort when building the label function. Active Learning can be a suitable method for picking out dev set examples.

4.3.3 Label Model

Results from label functions give us a set of labels that often either overlap (when multiple LBs return the same result on a sample) or conflict (when LBs return different results on a sample). In the weakly supervised model, the Label Model will be used to resolve these overlaps and conflicts to produce the prediction results in probabilistic labels.

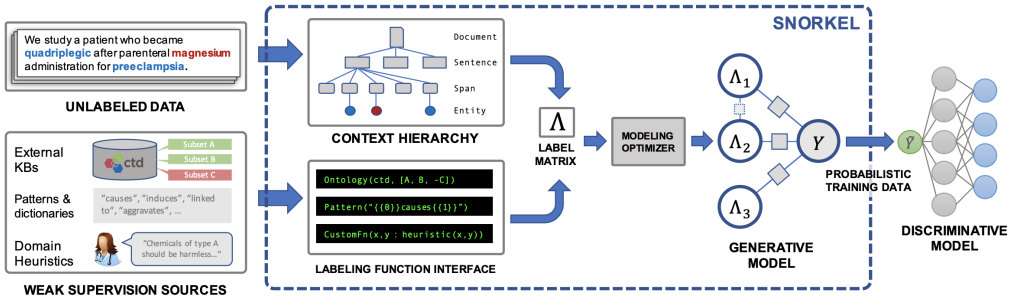

Figure 4.2: Overview of the Snorkel system [10]

The first step in Label Model processing is to find out the dependencies between the label functions. This dependence includes relationships such as similar, reinforcing, fixing, or exclusive. To automate this relationship learning, the Label Model constructs a weighted factor graph to represent the relationships between the LFs, then selects the non-zero weighted relationships. The second processing step is to determine the correct label for each example by learning the reliability of each label function. This can be done in different ways depending on the particular algorithm. And finally, in each example, we calculate the ratio of each label by the total confidence of the label functions that return that label.

4.4 Framework and library

4.4.1 Snorkel

Snorkel [10] is an open source machine learning library that provides a number of programming tools that allow rapid construction of training datasets using weak supervised learning method (view Fig 4.2). The main functions include:

- Labeling: The function allows the user to define label functions to label the data, then Snorkel uses the Generative Model to model and integrates the weak labels returned from the label functions. The goal of the Generative Model is to estimate the accuracy of each label function for each label while not knowing the correct label for each example in advance. The idea here is to evaluate the consensus on the results returned by the label functions on each example; when the number of

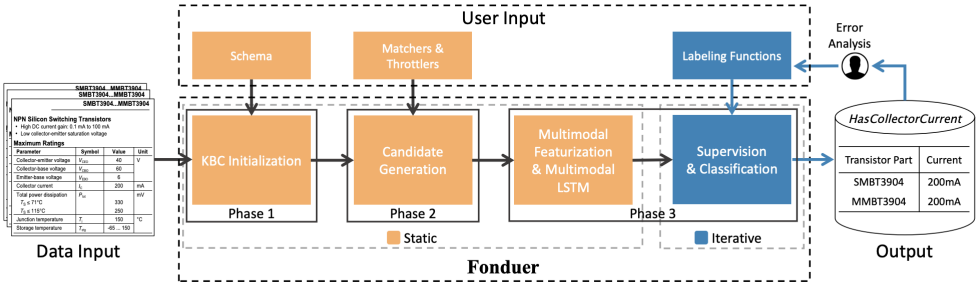

Figure 4.3: Overview of Fonduer [15]

examples is large enough, the variance of the label function for each label will approximate the accuracy of the label function for that label. By computing the label covariance matrix for the label functions, the Generative Model can construct a graph of the correlation of the label functions with each other and with the unknown correct label, thereby estimating the accuracy of the label functions.

- Transforming: This function allows more labeled examples from an original example. Users only need to write transformation functions to define these transformations.

- Slicing: Users write slicing functions that define arguments to subdivide data sets so that they can perform labeling or transformations on smaller data sets to increase labeling accuracy for some specific data especially.

4.4.2 Fonduer

Fonduer [15] is a framework for Knowledge Bases Construction (KBC) written in python and applied in processing rich format data. Fonduer could work with many other formats of data such as Text, CSV, HTML, or PDF, allowing different types of features such as text, structures, tables, and images to be combined, making better use of these data formats. View Fig 4.3.

In building the knowledge base, there are four object types that play an important role: Entity, Relation, Mention of Entity, and Relation of Mention.

- Entity: The entity in the Knowledge Base represents an object that can be a distinct person, thing, or place in the real world

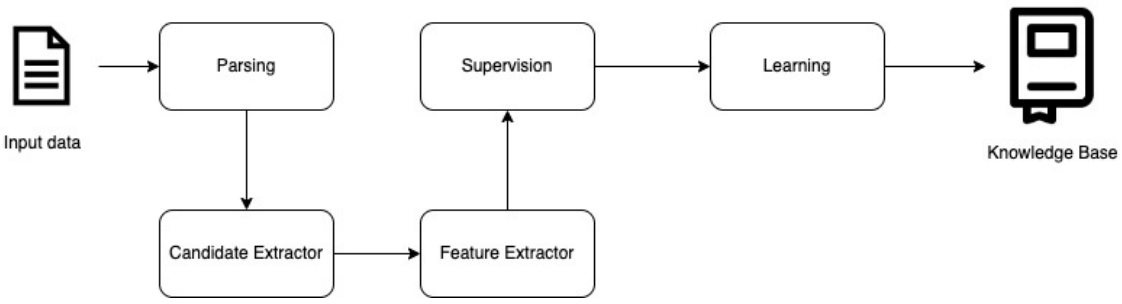

Figure 4.4: Component and Pipeline Process in Fonduer

- Relation: Entities related to each other will be represented through Relation between them. A Relation can have one or more different entities.
- Mention: is a span of text in the data that references (reminds) the entity.
- Relation of Mention: are the relationships between the mentions in the data

The fonduer framework includes many components with different functions to assist users in building the knowledge base (checkout Fig 4.4):

- Parsing: Is the component that plays the role of transforming input data in a variety of formats into Fonduer's data model (Figures 4.5, 4.6). Through the Preprocessor Paser of fonduer can handle many different data formats, then the data goes through the Parser to become Data Model.
- Candidate Extractor: The second stage of the process is to extract the candidates through the Candidate Extractor. As mentioned above, to determine the relationship between entities, it is necessary to determine the relationship between the mentions that refer to that entity. The candidate is a candidate for a Relation of mention that will later be determined to be true or not. Users can define Mention and Candidate by themselves and write Mention Space, Matcher, and Throttler to extract them from the data model. Fonduer also has a number of built-in methods and functions to support users.
- FeatureExtractor: This function allows the user to define the characteristics of a candidate. Fonduer provides a number of built-in methods

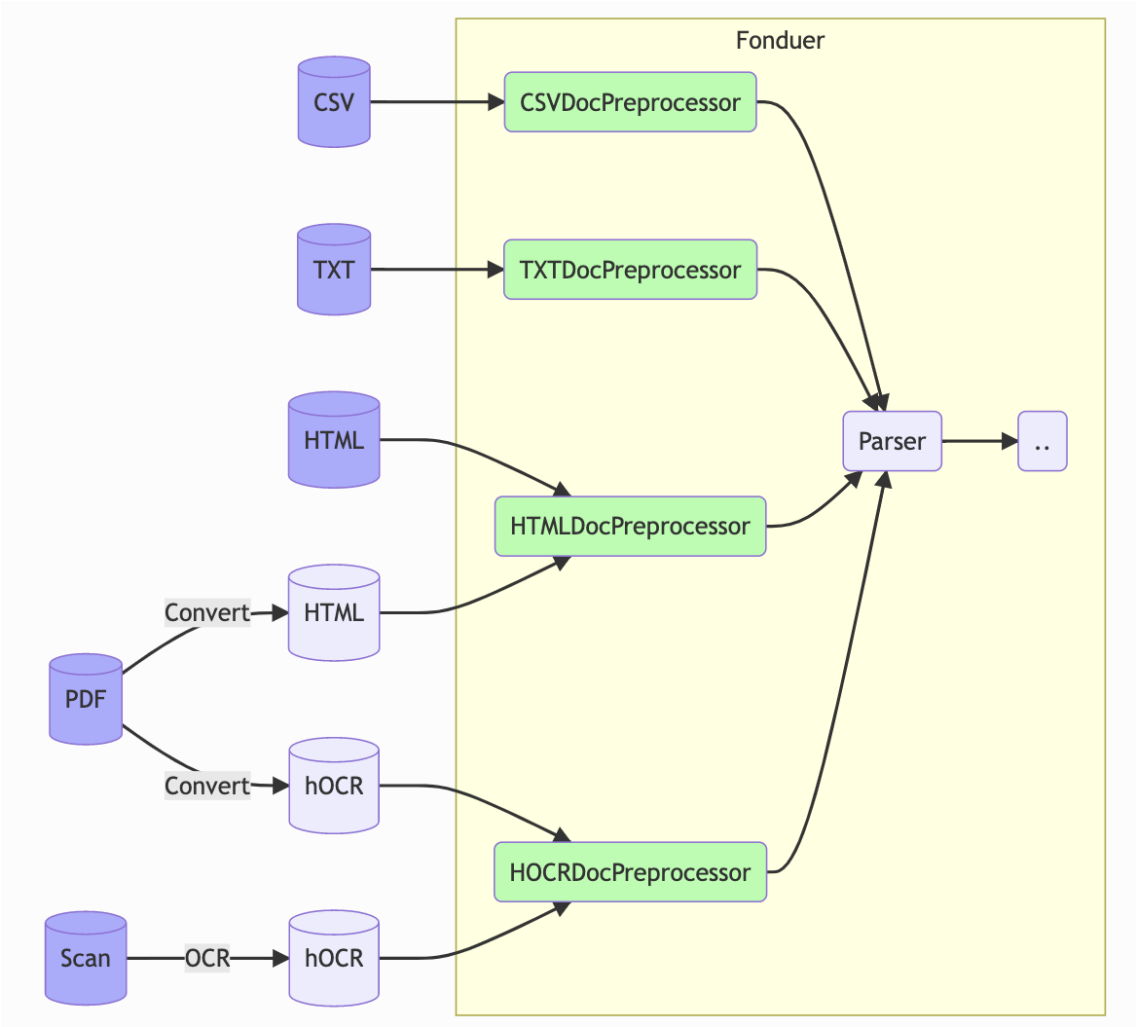

Figure 4.5: Parsing Component in Fonduer [15]

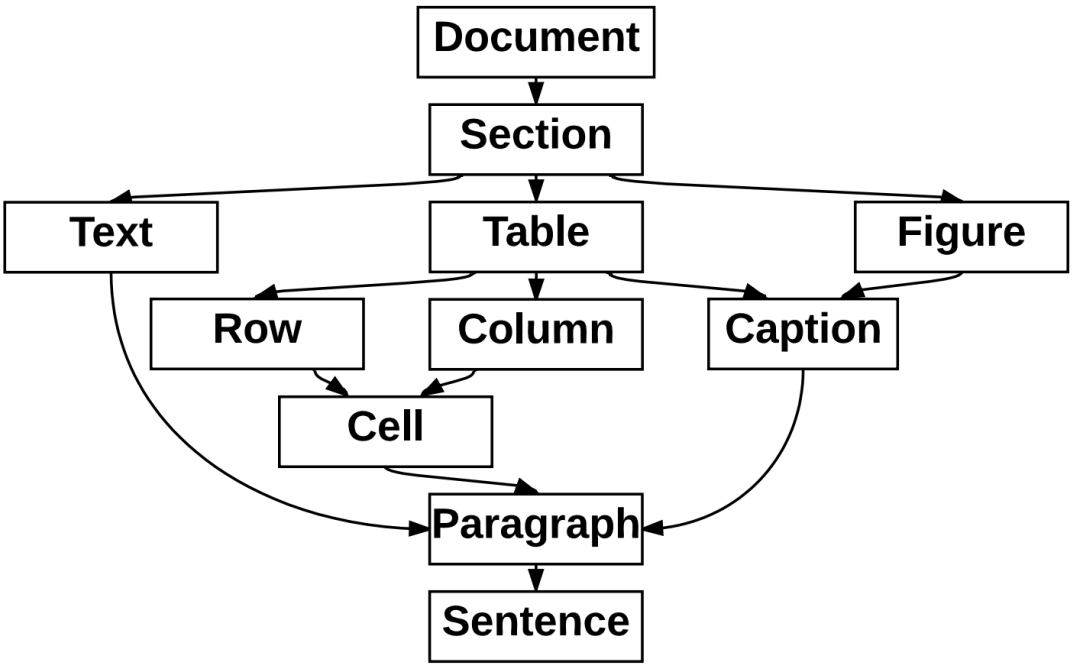

Figure 4.6: Fonduer Data Model [15]

to help users extract common features such as Text Features, Structure Features, and Display Features.

- Supervision: In order to be able to train a model to determine if a candidate is a Relation of Mention or not by a classifier machine learning model, however, because the classification problem belongs to the type of supervised machine learning, training data is required. To do this, Fonduer uses the Snorkel library, which fully supports allowing users to write label functions to label data.

- Learning: This is the final stage in the process; the candidates, after feature extraction and labeling, will be included for training to get the model. Fonduer supports some popular models such as LSTM and Logistic Regression.

- Package and Deploy: Fonduer supports the package and deployment of the entire pipeline model (parsing, extraction, featurization, and classification) and deploys it to a remote place to serve, using the MLflow format.

Chapter 5

Information Extraction Model: Implementation and Results

5.1 Dataset

The data used to build the model is the Main HTML of the Pages labeled as REAL_ESTATE_PAGE. This is the result from the main content model discussed in Chapter 3, without any additional preprocessing.

5.2 Implementation Idea

With the requirement of the information to be extracted, we designed including four candidates as follows: (Fig 5.1):

- Address Candidate: Made up of Address Mention and includes four labels: Province, District, Ward, Street, Project.
- Number Candidate: Made up of Number Mention, are all numbers appear in the document. Includes seven labels: Acreage, Surface width, Surface length, Street width, number of rooms, number of toilets, and number of floors.
- Price Candidate: Is a pair of numbers and units standing side by side and has two labels: Price and False (not the price)
- Date Candidate: Made up of Date, has two labels are Submission Date and False (not the submission date).

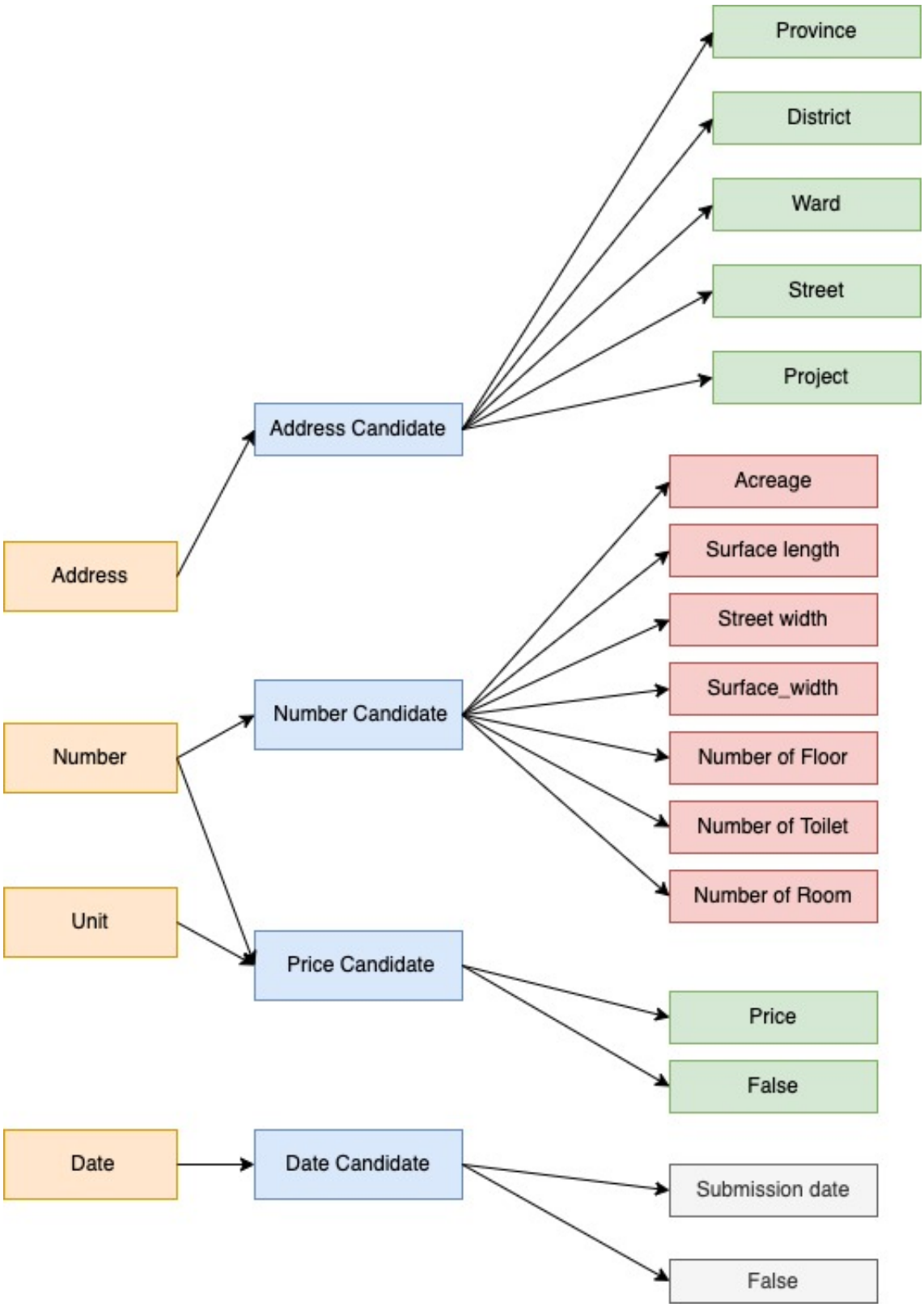

Figure 5.1: Implementation Idea

5.3 Implementation

In this section, we will go through the entire process of data processing, labeling and training the model according to Fonduer's steps presented in Section 4.4.2.

5.3.1 Setup

Firstly, we need to install Fonduer, Jupyster notebook, vi-spacy[11] and Postgresql. Then we create a new database in Postgresdb call : real_estate_fonduer. Open jupyter notebook and config database. See in Appendix A1.

Fonduer uses the Postgresql database to store data between processing steps, which prevents the data from needing to be reprocessed if your code in the previous steps does not change, thereby improving efficiency.

5.3.2 Parser

Our data in this problem is in HTML so we use the HTMLDocPreprocessor built into Fontuer. This preprocessor allows HTML input data to become Fonduer's Data Model (see Figure 4.6).

Because the data we are using is in the Vietnamese language, we need to use an additional language model for Vietnamese to support tokenizer and Pos tagging. Fonduer has built-in support to use Spacy's model [12], so we used the Vietnamese Language Model vi-spacy [11] by Tran Viet Trung for our project. See Appendix A2.

5.3.3 Candidate Extractor

Analyzing the information that needs to be extracted for the requirements of this model, we find that it can be divided into the following four Mentions and four Candidates. See Appendix A3.

Number is a mention that refers to words that are numbers including both real numbers and integers. We use RegexMatcher, which is supported by Fonduer to identify the mention Numbers. See Appendix A4.

AddressMention is a text that refers to the address generated from a sentence containing the address by the N-gram method from 1 to 5 words, with the matcher using the rule to remove span text containing special characters (because the address string does not contain special characters). See Appendix A5.

UnitMention is a mention mentioning price units, e.g., billion, million, million per m2. To catch this mention, we use DictionaryMatcher; this matcher allows us to catch mentions in a given dictionary set. Since the price unit only includes a certain number of keywords (about ten keywords), we use this matcher. DateMention refers to time, which is similar to Number; we use RegexMatcher with a suitable regular expression to catch this mention.

5.3.4 Labeling

As described in the technology introduction, Fonduer fully supports the Weak Supervision method through the Snorkel library. In this section, we build label functions to label the data, presented as follows.

The main idea of our data labeling is to use an iterative cycle of writing label functions, training the model and evaluating and then updating the label functions again. In the first loop we use pre-train models, simple rules for constructing label funcition see appendices A6, A7, A8, A9.

To evaluate the results, we use GoldLabel to evaluate Labeling by hand-labeling ten Pages on each Website; we use this label to make GoldLabel. See Appendix A10. Using GoldLabel allows us to observe the progress of the trained model through each iteration, check the label function changes for better or worse results for the final model and then make fit adjustments.

We use the analysis results on each label to update, add more label functions for the next loop. Table 5.1 shows the analysis of the label function of address. In this analysis, we will see important parameters about label functions such as the coverage of the label function on the dataset, the degree of overlap, the conflict of label functions and their accuracy when compared to GoldLabel.

Table 5.1: Analyst Label Function Address

	j	Polarity	Coverage	Overlaps	Conflig	Acc
lf_address_extract_model	0	[0, 1, 2, 3, 4]	0,3357	0,3185	0,0172	0,98
lf_in_dict	1	[0, 1, 2, 3, 4]	0,2435	0,138	0,1055	0,76
lf_pre_key	2	[0, 1, 2, 3, 4]	0,098	0,075	0,023	0,88
lf_pre_thanh_pho_is_tinh	3	[3]	0,0012	0,0008	0,0004	0,36
lf_pre_thanh_pho_is_quan	4	[2]	0,0054	0,0032	0,0022	0,14
lf_pre_is_quan	5	[3]	0,0025	0,00017	0,00233	0,47

5.3.5 Feature Extraction

After assigning label, we get LabelModel for each label, can use this LabelModel directly for final model in One Stage method, but will always have to use label functions, which causes some problems subject, especially when the label functions use a third-party service. In this problem, we use the Two Stage method, that is, train a new model based on the results of the LabelModel and its features. This makes the end model no longer dependent on the label function. First, we divide the existing dataset into train, test and dev. See Appendix A11.

To extract the features for the Candidates we use the support from Fonduer. Frame work Fontuer allows to extract features from data in its Data Model form including display features, structure features and text features. Users only need to configure the specific types to use in the file "fonduer-config.yaml". See Appendix A12. In our problem, we only use structural features and text features because the display feature requires calculating the distance and size of the elements, which is not possible without rendering the page on the browser. On the source code we just need to use the Featurizer to create a feature matrix for the candidates. See Appendix A13.

5.3.6 Train Final Model

At the training stage, since the problem is a typical classification problem, we use the LogisticRegression model supported in Fonduer, with the textual and structural features described above. We then compare the results with GoldLabel to evaluate the model.

Table 5.2: Result of Information Extraction Model.

Label	Gold Label	F1 score	Coverage
Address	740	0,72	0,98
Price	236	0,81	0,88
Area	228	0,83	0,75
Surface width	136	0,88	0,42
Street width	67	0,7	0,37
Phone	239	0,65	0,81
Post date	245	0,74	0,83

5.4 Result evaluation

5.4.1 Evaluation Method

Since the input data is not labeled, we build a labeled dataset by hand called Gold Label. The Gold Label dataset is built by randomly taking from each website about 10 to 15 pages (Total is 250 pages), then determining the information on that page, including seven types of information to be extracted: Address, price, area, surface size, street width, phone number, posting date. This data set is then compared with the results extracted from the model so that the quality of the model can be assessed. The process of building the model is done in a loop: labeling -> train -> evaluation -> labeling, continuously until the model achieves the desired accuracy.

We performed the assessment on 7 fields of information to be extracted (Figure 5.2). Columns in the results table include:

- Gold Label: Total number of labels from 250 pages in Gold Labels dataset. Note with Address in 1 page can get from 1 to 4 addresses corresponding to 4 labels: Province, District, Ward, Street.
- F1 score: Calculated on 250 Pages of the Gold Label set by each information field.
- Coverage: Ratio of pages extracted information to total pages (39609 pages).

5.4.2 Result and Discussion

Based on the results from the table 5.2 we discuss the findings and analysis as follows:

- The accuracy when extracting address information is not high because the labeling uses an address dictionary, so the model cannot handle new addresses in practice. If we look at the Province and District labels separately, the accuracy is higher because there are fewer new addresses than the Ward and the Street. Some other errors due to the similarity of names between labels led to model errors; these errors were reduced when adding label functions based on the relationship between labels.

- The labels returned from NumberCandidate are generally more accurate because the rules can be defined explicitly, with less ambiguity. For phone numbers because there are many cases where phone numbers contain special characters such as spaces or dashes. To solve this problem, we have added a step to normalize the phone number, so it has also improved.

- The coverage ratio on the entire data set is an important criterion to evaluate the model. It helps to avoid creating too tight label functions that make the accuracy high but reduce the coverage of the model.

- Another important thing to mention is that building this model does not take too much time, including the time to install the system to label the data; we only need two weeks to build the model from the unlabelled raw data. This is because we have fully used the existing models to label data for the new model.

Chapter 6

System Implementation Result

Figure 6.1 depicts the deployment architecture of the system in the product environment. Besides 2 main components are AI Crawler and Website Explorer, the system has some other components as follows:

- Tor proxy: a proxy that stands between the system and the Internet that allows to change the IP when collecting data to enhance anonymity.
- AWS S3 Storage: Where Website Explorer stores URL models of Websites, and AI Crawler retrieves these models for use.
- Mongodb: The data collected by AI Crawler will be stored here.
- Scheduler: Allows scheduling to re-collect data from websites to update the latest information.
- Management System: A CMS provides functions that allow administrators to add new websites and manage existing websites. See Figure 6.2.
- Metadata Storage: A relational database that stores metadata of known websites for CMS and Scheduler.

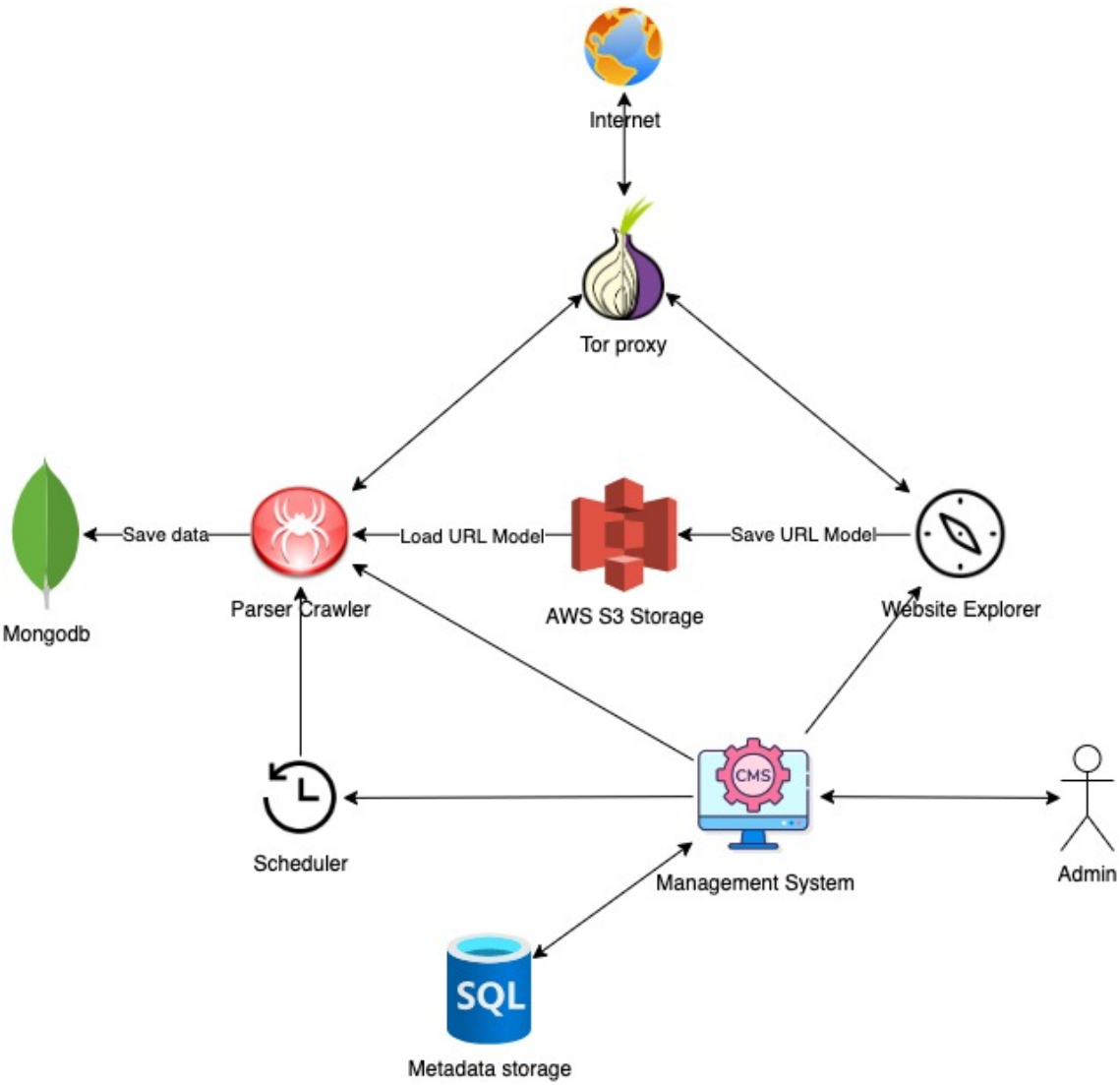

Figure 6.1: Deployment Architecture

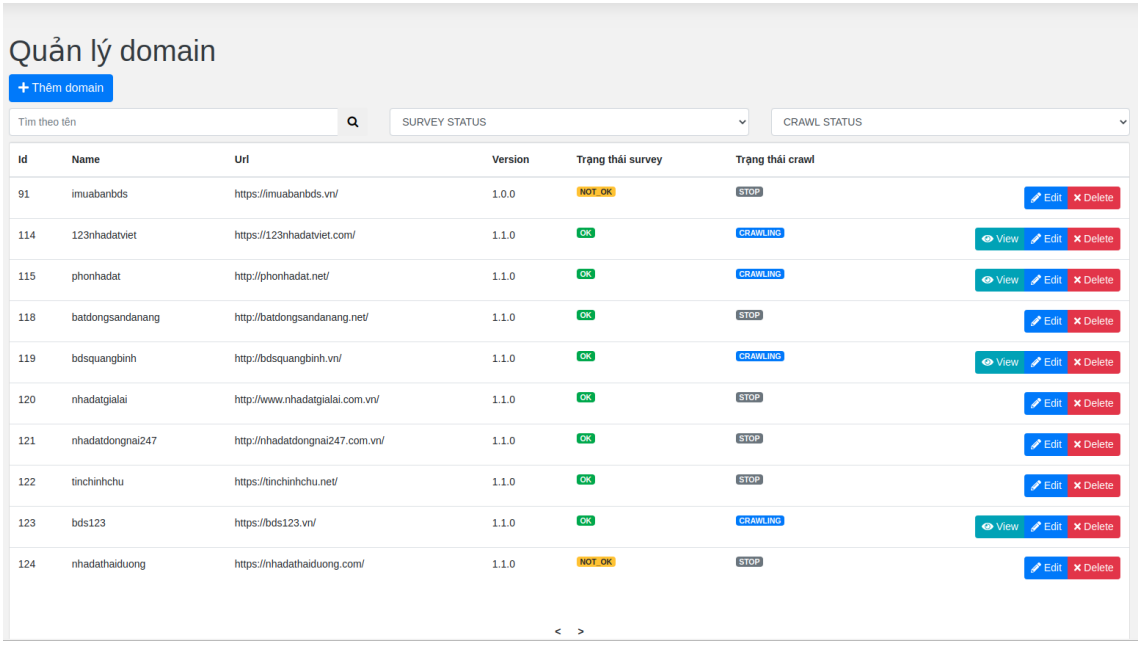

Figure 6.2: Management System

Chapter 7

Conclusion

In the thesis, we have presented the results of our research on Weak Supervision for Information Extraction applied in building AI Crawler system. The problems posed for this research, including the problem of extracting information from HTML data, and the data collection system from many different websites, have been solved and completed.

Regarding the contributions of this study, we introduced a new design for a data collection system that uses machine learning to extract information from web pages without predefined definitions. The system has been successfully deployed and is being used at Cengroup.

Besides, we have also successfully applied the Weak Supervision method in data labeling for the Information Extraction problem, which helps to reduce the cost when building this model.

Regarding the direction of this research development, We will continue to test this approach in other domains such as e-commerce, employment, and NFT. As an outlook for future work, we want to develop an open-source toolkit available to the public to build similar systems for other domains.

Bibliography

[1] https://scrapy.org.

[2] https://www.w3schools.com/cssref/css_selectors.asp.

[3] Marco Baroni, Francis Chantree, Adam Kilgarriff, and Serge Sharoff. Cleaneval: a competition for cleaning web pages. In *Proceedings of the Sixth International Conference on Language Resources and Evaluation (LREC'08)*, Marrakech, Morocco, May 2008. European Language Resources Association (ELRA).

[4] Deng Cai, Shipeng Yu, Ji-Rong Wen, and Wei-Ying Ma. Vips: a vision-based page segmentation algorithm. 2003.

[5] G David Forney. The viterbi algorithm. *Proceedings of the IEEE*, 61(3):268–278, 1973.

[6] Armand Joulin, Edouard Grave, Piotr Bojanowski, and Tomas Mikolov. Bag of tricks for efficient text classification. *arXiv preprint arXiv:1607.01759*, 2016.

[7] Christian Kohlschütter, Peter Fankhauser, and Wolfgang Nejdl. Boilerplate detection using shallow text features. In *Proceedings of the third ACM international conference on Web search and data mining*, pages 441–450, 2010.

[8] Qian Li, Hao Peng, Jianxin Li, Congying Xia, Renyu Yang, Lichao Sun, Philip S Yu, and Lifang He. A survey on text classification: From traditional to deep learning. *ACM Transactions on Intelligent Systems and Technology (TIST)*, 13(2):1–41, 2022.

[9] Matthew E Peters and Dan Lecocq. Content extraction using diverse feature sets. In *Proceedings of the 22Nd international conference on world wide web*, pages 89–90, 2013.

[10] Alexander Ratner, Stephen H Bach, Henry Ehrenberg, Jason Fries, Sen Wu, and Christopher Ré. Snorkel: Rapid training data creation with weak supervision. In *Proceedings of the VLDB Endowment. International Conference on Very Large Data Bases*, volume 11, page 269. NIH Public Access, 2017.

[11] Tran Viet Trung. Vietnamese language model for spacy.io. Unpublished paper, 2021.

[12] Yuli Vasiliev. *Natural Language Processing with Python and SpaCy: A Practical Introduction.* No Starch Press, 2020.

[13] Thijs Vogels, Octavian-Eugen Ganea, and Carsten Eickhoff. Web2text: Deep structured boilerplate removal, 2018.

[14] Tim Weninger, William H Hsu, and Jiawei Han. Cetr: content extraction via tag ratios. In *Proceedings of the 19th international conference on World wide web*, pages 971–980, 2010.

[15] Sen Wu, Luke Hsiao, Xiao Cheng, Braden Hancock, Theodoros Rekatsinas, Philip Levis, and Christopher Ré. Fonduer: Knowledge base construction from richly formatted data. In *Proceedings of the 2018 international conference on management of data*, pages 1301–1316, 2018.

[16] Jieyu Zhang, Cheng-Yu Hsieh, Yue Yu, Chao Zhang, and Alexander Ratner. A survey on programmatic weak supervision. *arXiv preprint arXiv:2202.05433*, 2022.

Glossary

AI Crawler The new crawler as an integrated AI model.

Category The type of a Page.

Content Classification Model The classification model to determine what Category a text (such as Title, Description, Main Text) belongs to.

Description The description of a Raw HTML of a Page; the text inside meta['description'] of HTML.

Domain An area of knowledge, such as Real Estate, E-commerce, Education.

End Element The last element of the main Part in a Raw HTML.

End Selector The CSS Selector [2] of an End Element.

Extracted Data The result when Information Extraction Model extracts a Main HTML, containing Label and Value of label in the Main HTML.

Information Extraction Model The model that returns Extracted Data of the Main HTML .

Label A label in Extracted Data, such as Address, Price, Acreage.

Main Content Detection Model The model that returns Main HTML, Main Text of a Raw HTML.

Main HTML Raw HTML after removing all Parts except main. This is the result of the Main Content Detection Model after processing a Raw HTML.

Main Text The main content in text format of a Page; text returned by Main Content Detection Model after processing a Raw HTML.

OTHER_PAGE A Category of the Real Estate Domain that refers to a different type, not to be confused with REAL_ESTATE_PAGE.

Page A specific page within a Website that associates with a unique URL. A website can contain multiple Pages.

Page Classification The act of determining what Category a Page belongs to; may use Content Classification Model or URL Classification Model.

Parser Crawler A component inside AI Crawler to crawl data from known Website.

Part A part of an HTML page, such as header, footer, navigation bar, sidebar, main.

Raw HTML The HTML response of a Page when requested to the URL of this Page.

REAL_ESTATE_PAGE A Category of the Real Estate Domain, including the Real Estate detail Page in the Websites of Real Estate Domain.

Scrapy Crawler The old data crawler based on Scrapy platform, without the use of AI models.

Spider A module used in Scrapy Crawler to extract data from an HTML page.

Start Element The first element of the main Part in a Raw HTML.

Start Selector The CSS Selector [2] of a Start Element.

Title The title of a Raw HTML of a Page; the text inside <title></title> tag of HTML.

URL The URL of a Page.

URL Classification Model The classification model to determine what Category a URL belongs to.

URL Classification Trainer The module used to train URL Classification Model of a new Website.

Value Value of Label in Extracted Data. For example, with the Label "Address", the Value is "Ha Noi".

Weak Supervision As described in Chapter 4.

Website A collection of web pages and related content. For example, batdongsan.com.vn, muabannhadat.tv,... that belong to a particular Domain (Real Estate).

Website Explorer The tool applied to a new Website to generate URL Classification Model for this Website.

Appendix A

Some source code were used in thesis

```
1  PARALLEL = 4  # assuming a quad-core machine
2  ATTRIBUTE = "real_estate_fonduer"
3  conn_string = 'postgresql://localhost:5432/' + ATTRIBUTE
4
5  from fonduer import Meta, init_logging
6
7  # Configure logging for Fonduer
8  init_logging(log_dir="logs")
9
10 session = Meta.init(conn_string).Session()
```

Listing A.1: Config database

```
1  from fonduer.parser.preprocessors import HTMLDocPreprocessor
2  from fonduer.parser import Parser
3
4  docs_path = 'data/html/'
5
6  max_docs = 100
7  doc_preprocessor = HTMLDocPreprocessor(docs_path, max_docs=
       max_docs)
8
9  corpus_parser = Parser(session, structural=True, lingual=True
       , language="vi", tabular=True,)
10 %time corpus_parser.apply(doc_preprocessor, parallelism=
       PARALLEL)
```

Listing A.2: Parse HTML input to Fonduer Model

```
Number = mention_subclass("Number")
Address = mention_subclass("Address")
Unit = mention_subclass("Unit")
Date = mention_subclass("Date")
NumberCandidate = candidate_subclass("NumberCandidate", [Number])
AddressCandidate = candidate_subclass("AddressCandidate", [Address])
PriceCandidate = candidate_subclass("PriceCandidate", [Number, Unit])
DateCandidate = candidate_subclass("DateCandidate", [Date])
```

Listing A.3: Define candidates

```
mention_number = MentionNgrams(n_max=1)
regx_match_number = RegexMatchSpan(rgx='[0-9]', search=True, full_match=False)

mention_unit = MentionNgrams(n_min=1, n_max=3)
match_unit = DictionaryMatch(d=price_unit)
mention_date = MentionNgrams(n_min=5, n_max=5)
match_date = RegexMatchSpan(rgx='
    ^[0-3]?[0-9][\/-][0-3]?[0-9][\/-](?:[0-9]{2})?[0-9]{2}$')
```

Listing A.4: Extract mentions number

```
def check_is_title(m: TemporaryContext):
    words = m.get_attrib_tokens()
    text = ' '.join(words)
    if text.istitle():
        false_list = [',', '|', '-']
        for char in false_list:
            if char in words:
                return False
        return True
    elif text.isdigit():
        return True
    else:
        return False

class MentionAddressNgrams(Ngrams):
    def __init__(
            self, n_min: int = 1, n_max: int = 5,
               split_tokens: Collection[str] = [],
            address_name: Collection[str] = []
    ) -> None:
        self.address_name = address_name
        Ngrams.__init__(self, n_min=n_min, n_max=n_max,
           split_tokens=split_tokens)

    def apply(self, doc: Document) -> Iterator[
       TemporarySpanMention]:
        for sentence in doc.sentences:
            text_sent = sentence.text.lower().replace('_', ' '
               )
            count_address = 0
            for address in self.address_name:
                if address in text_sent:
                    count_address += 1
            if count_address > 0:
                for ts in Ngrams.apply(self, sentence):
                    yield ts
mention_address = MentionAddressNgrams(n_min=1, n_max=3,
   address_name=list_address_name)
match_address = LambdaFunctionMatcher(func=check_is_title)
```

Listing A.5: Extract mentions address

```
ABSTAIN = -1
DUONG = 0
PHUONG = 1
QUAN = 2
TINH = 3
DU_AN = 4
FALSE_ADDRESS = 5
from gmlib.address_tagger.parse import AddressTagger
tagger = AddressTagger()
tagger.load_model('accent')

def lf_address_extract_model(c):
    span_mention = c.address.context
    sent_text = span_mention.sentence.text
    char_start = span_mention.char_start
    char_end = span_mention.char_end
    result = tagger.parser(sent_text)
    for entity in result['entities']:
        if entity['start'] == char_start and entity['end'] ==
            char_end:
            return map_entity[entity['entity']]
    return ABSTAIN

def lf_in_dic(c):
    text_span = c.des[1].lower().replace('_', ' ')
    if text_span in key_address_name['tinh']:
        return TINH
    if text_span in key_address_name['quan']:
        return QUAN
    if text_span in key_address_name['phuong']:
        return PHUONG
    if text_span in key_address_name['duong']:
        return DUONG
    if text_span in key_address_name['du_an']:
        return DU_AN
    return ABSTAIN
```

Listing A.6: Labeling Address

```
ABSTAIN = -1
DIEN_TICH = 0
SO_TANG = 1
SO_PHONG_NGU = 4
SO_PHONG_KHACH = 5
SO_PHONG_VE_SINH = 7
MAT_TIEN = 8
CHIEU_DAI = 9
CHIEU_RONG = 10
DO_RONG_DUONG = 11
SO_DIEN_THOAI = 12
NUMBER_FAILE = 13
map_entity_label = {
    "ABSTAIN": ABSTAIN,
    "surface_size": DIEN_TICH,
    "number_of_floors": SO_TANG,
    "number_of_rooms": SO_PHONG_NGU,
    "number_of_toilets": SO_PHONG_VE_SINH,
    "surface_width": MAT_TIEN,
    "surface_length": CHIEU_DAI,
    "street_width": DO_RONG_DUONG,
    "contact_phone": SO_DIEN_THOAI,
    "NUMBER_FALSE": NUMBER_FAILE
}

from gmlib.content_extract import IE
content_ie = IE()

def lf_number_extract_model(c):
    span_mention = c.address.context
    sent_text = span_mention.sentence.text
    char_start = span_mention.char_start
    char_end = span_mention.char_end
    result = IE.extract(sent_text)
    for entity in result['entities']:
        if entity['start'] == char_start and entity['end'] ==
            char_end:
            if entity['entity'] in map_entity_label:
                return map_entity_label[entity['entity']]
    return ABSTAIN
```

Listing A.7: Labeling Number

```
NGAY_DANG = 0
FALSE_DATE = 1
date_labels = {
    "submission_date": NGAY_DANG,
    "FALSE_DATE": FALSE_DATE,
}
def lf_is_ngay_dang(c):
    pre_words = [word.lower() for word in c.des[0]]
    pre_text = ' '.join(pre_words)
    pre_text = pre_text.replace('_', ' ')
    if 'đăng' in pre_text:
        return NGAY_DANG
    if 'cập nhật' in pre_text:
        return NGAY_DANG
    return FALSE_DATE
```

Listing A.8: Labeling submission date

```
ABSTAIN = -1
GIA = 0
FALSE_GIA = 1
price_labels = {
    "price": GIA,
    "FALSE_GIA": FALSE_GIA,
}


def lf_is_not_false_gia(c):
    (number, unit) = c
    pre_words = [word.lower() for word in c.des[0]]
    pre_text = '␣'.join(pre_words)
    lower_unit = unit.context.get_span().lower()
    if lower_unit in ['triệu', 'tr']:
        if 'lỗ' in pre_text:
            return FALSE_GIA
        if '%' in pre_text:
            return FALSE_GIA
        if 'chiết_khấu' in pre_text:
            return FALSE_GIA
        if 'giảm' in pre_text:
            return FALSE_GIA
        if 'quà' in pre_text:
            return FALSE_GIA
        if 'tặng' in pre_text:
            return FALSE_GIA
        if 'voucher' in pre_text:
            return FALSE_GIA
        if 'trả␣trước' in pre_text:
            return FALSE_GIA
    if lower_unit in ['triệu/tháng', 'triệu/th', 'tr/th']:
        if 'trả_góp' in pre_text:
            return FALSE_GIA
    return GIA
```

Listing A.9: Labeling price

```
def ls_address_gold(c):
    document = c.document
    gold_label_address = document.meta['gold_label']['address
        ']
    span_text = c.address.context.get_span().lower()
    for key in gold_label_address:
        if gold_label_address[key] == span_text:
            return map_label[key]
    return ABSTAIN
```

Listing A.10: Labeling GoldLabel for Address

```
docs = session.query(Document).order_by(Document.name).all()
ld   = len(docs)

train_docs = set()
dev_docs   = set()
test_docs  = set()
splits = (0.8, 0.9)
data = [(doc.name, doc) for doc in docs]
data.sort(key=lambda x: x[0])
for i, (doc_name, doc) in enumerate(data):
    if i < splits[0] * ld:
        train_docs.add(doc)
    elif i < splits[1] * ld:
        dev_docs.add(doc)
    else:
        test_docs.add(doc)
```

Listing A.11: Split data train-test

```
featurization:
  textual:
    window_feature:
      size: 3
      combinations: True
      isolated: True
    word_feature:
      window: 7
  tabular:
    unary_features:
      attrib:
        - words
      get_cell_ngrams:
        max: 2
      get_head_ngrams:
        max: 2
      get_row_ngrams:
        max: 2
      get_col_ngrams:
        max: 2
learning:
  LogisticRegression:
    hidden_dim: 100
    bias: False
```

Listing A.12: Config Feature Extraction

```
for i, docs in enumerate([train_docs, dev_docs, test_docs]):
    candidate_extractor.apply(docs, split=i, parallelism=
       PARALLEL)

train_cands = candidate_extractor.get_candidates(split = 0)
dev_cands = candidate_extractor.get_candidates(split = 1)
test_cands = candidate_extractor.get_candidates(split = 2)

from fonduer.features import Featurizer

featurizer = Featurizer(session, [NumberCandidate,
   AddressCandidate, PriceCandidate, DateCandidate])

%time featurizer.apply(split=0, train=True, parallelism=
   PARALLEL)
%time F_train = featurizer.get_feature_matrices(train_cands)

%time featurizer.apply(split=1, parallelism=PARALLEL)
%time F_dev = featurizer.get_feature_matrices(dev_cands)

%time featurizer.apply(split=2, parallelism=PARALLEL)
%time F_test = featurizer.get_feature_matrices(test_cands)
```

Listing A.13: Extract featuer