**Wrapper generation**

Wrapper generation is a process of automatically creating software components that can extract information from unstructured or semi-structured data sources, such as web pages, PDFs, or databases, and transform it into structured data. Wrappers are software components that encapsulate the functionality of data extraction from a specific data source and present the extracted data in a structured format.

The process of wrapper generation typically involves the following steps:

Identifying the data source: The first step is to identify the data source from which data needs to be extracted. This could be a website, a PDF document, or a database.

Analysing the data source: The next step is to analyse the structure of the data source to determine how the data is organized and how it can be extracted. This may involve examining the HTML code of a website, or the schema of a database.

Creating a wrapper template: A wrapper template is a blueprint for the wrapper that defines the structure of the data to be extracted and how it should be transformed into a structured format. This template is typically created using a domain-specific language (DSL) or a visual interface.

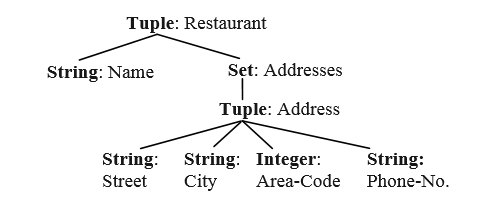
Generating the wrapper code: Once the wrapper template is created, the wrapper code is generated automatically using a wrapper generation tool. This code is tailored to the specific data source and the structure of the data to be extracted.

Testing and refinement: The final step is to test the wrapper code and refine it as necessary. This may involve debugging errors, tweaking the template, or fine-tuning the extraction parameters.

Wrapper generation can save a significant amount of time and effort in the data extraction process, as it automates the creation of the code necessary to extract data from various sources. It is especially useful in cases where the data source structure may change frequently or where large amounts of data need to be extracted quickly and efficiently.

**Wrapper Induction**

We are now ready to study the first approach to data extraction, namely wrapper induction, which is based on supervised learning. A wrapper induction system learns data extraction rules from a set of labeled training examples. Labeling is usually done manually, which simply involves marking the data items in the training pages/examples that the user wants to extract. The learned rules are then applied to extract target data from other pages with the same mark-up encoding or the same template.



Wrapper induction is a technique used in machine learning for automating the process of feature engineering. It involves training a machine learning model on a given dataset and then using the model's performance to select a subset of features that are most relevant for the task at hand.

Here are the step-by-step process of wrapper induction:

* Choose a machine learning algorithm: The first step in wrapper induction is to select the algorithm you want to improve. The wrapper method can be applied to any algorithm, including decision trees, logistic regression, random forests, and support vector machines.
* Choose a subset of features: Next, you need to select a subset of features that you want to use for the wrapper. The subset can be selected based on prior domain knowledge or using feature selection techniques.
* Build a wrapper: After selecting the features, you need to build a wrapper around the machine learning algorithm. The wrapper takes the selected features and trains the model on the training data, and evaluates it on the validation data.
* Evaluate the wrapper: You need to evaluate the performance of the wrapper on the validation data. You can use metrics like accuracy, precision, recall, and F1-score to evaluate the performance.
* Optimize hyperparameters: The next step is to optimize the hyperparameters of the wrapper. You can use techniques like grid search, random search, or Bayesian optimization to find the best hyperparameters.
* Test the wrapper: Once you have optimized the hyperparameters, you need to test the wrapper on the test data. You can use the same metrics to evaluate the performance of the wrapper on the test data.
* Deploy the wrapper: Finally, you can deploy the wrapper in a production environment to make predictions on new data.

Overall, wrapper induction is an iterative process that involves selecting a subset of features, building a wrapper around the algorithm, evaluating the wrapper's performance, optimizing hyper parameters, and testing the wrapper on new data.

For example, you might start with just the customer's age as a feature and train your model on that. Then you can add the customer's income and see if the model's performance improves. If it does, you keep the income feature and move on to the next one. If it doesn't, you remove the income feature and try a different one.

You continue this process until you have a set of features that maximizes the model's performance on a validation set. This set of features is then used to train a final model, which can be used to make predictions on new data.

Overall, wrapper induction is a powerful technique for automating the process of feature selection and engineering, which can save a lot of time and effort compared to manual feature selection.

**Wrapper Maintenance**

Once a wrapper is generated, it is applied to other Web pages that contain similar data and are formatted in the same ways as the training examples. This introduces new problems.

1. If the site changes, does the wrapper know the change? This is called the wrapper verification problem.

2. If the change is correctly detected, how to automatically repair the wrapper? This is called the wrapper repair problem.

One way to deal with both problems is to learn the characteristic patterns of the target items, which are then used to monitor the extraction to check whether the extracted items are correct. If they are incorrect, the same patterns can be used to locate the correct items assuming that the page changes are minor formatting changes. This is called re-labelling. After relabeling, re-learning is performed to produce a new wrapper. These two tasks are very difficult because contextual and/or semantic information is often needed to detect changes and to find the new locations of the target items. Wrapper maintenance is still an active research area.

**Instance-based wrapper learning**

Instance-based wrapper learning is a type of machine learning technique that uses an algorithm to select a subset of features from the dataset, and then trains a model on the selected features. The goal is to improve the accuracy and efficiency of the model by using only the most relevant features.

A simple example of instance-based wrapper learning can be illustrated as follows:

Suppose we have a dataset that contains information about a group of students, including their age, gender, grade point average (GPA), and test scores in math, reading, and writing. Our task is to build a model that can predict whether a student will pass or fail the final exam.

To use instance-based wrapper learning, we would first select a subset of features that are most relevant to the task. In this case, we might choose age, GPA, math score, and writing score, as they are likely to be strong predictors of a student's performance on the final exam.

Next, we would train a model on this subset of features, using a suitable algorithm such as logistic regression, decision trees, or support vector machines. We would then evaluate the performance of the model using a validation set or cross-validation.

If the performance of the model is satisfactory, we would use it to make predictions on new data. If not, we would adjust the subset of features and/or the model algorithm until we achieve a satisfactory level of accuracy.

Overall, instance-based wrapper learning is a powerful technique for feature selection and model building, as it enables us to identify the most relevant features and build models that are both accurate and efficient.

<html>

<head>

<title>Movie Title</title>

</head>

<body>

<h1>Movie Title</h1>

<p>Directed by Director Name</p>

<p>Release Year: 2022</p>

</body>

</html>

wrapper = {

"title": {

"tag": "title",

"type": "text"

},

"director": {

"tag": "p",

"contains": "Directed by",

"type": "text"

},

"release\_year": {

"tag": "p",

"contains": "Release Year",

"type": "text",

"regex": r"\d{4}"

}

}

Instance-based wrapper learning is a technique used to extract structured information from unstructured data such as HTML code. The goal is to automatically identify patterns in the data that correspond to relevant information and use those patterns to extract that information.

Here is an example of how instance-based wrapper learning can be used to extract information from HTML code:

Suppose you have a webpage that contains a list of products with their names, prices, and descriptions. The HTML code for this webpage might look like this:

<div class="product">

<h3 class="product-name">Product 1</h3>

<div class="product-price">$10.99</div>

<div class="product-description">This is a description of product </div>

</div>

<div class="product">

<h3 class="product-name">Product 2</h3>

<div class="product-price">$12.99</div>

<div class="product-description">This is a description of product </div>

</div>

<div class="product">

<h3 class="product-name">Product 3</h3>

<div class="product-price">$9.99</div>

<div class="product-description">This is a description of product </div>

</div>

To extract the relevant information from this HTML code, you could use instance-based wrapper learning to identify the patterns that correspond to each piece of information. For example, you might identify the following patterns:

Product name: the text inside the <h3> element with class "product-name"

Product price: the text inside the <div> element with class "product-price"

Product description: the text inside the <div> element with class "product-description"

Once you have identified these patterns, you can use them to extract the information from the HTML code. For example, you might use a regular expression to extract the product name:

import re

html = # the HTML code for the webpage

product\_name\_pattern = r'<h3 class="product-name">(.+)</h3>'

product\_names = re.findall(product\_name\_pattern, html)

This will extract the product names from the HTML code and store them in a list.

You can use similar techniques to extract the product prices and descriptions. Once you have extracted all of the relevant information, you can store it in a structured format such as a database or a spreadsheet.

**Automatic wrapper generation**

Automatic wrapper generation is a technique used in software engineering to automatically generate wrapper code around an existing software component. While this technique can provide many benefits such as reducing the amount of manual work required to integrate components, there are also some problems that can arise. Here are some of the most common problems associated with automatic wrapper generation:

**Loss of Control**: When using automatic wrapper generation, developers may lose control over the wrapper code that is generated. This can make it difficult to debug issues or make changes to the wrapper code.

**Performance Overhead:** Wrapper code can add an additional performance overhead to the system, as the wrapper code needs to be executed alongside the original code. This can result in slower performance for the system as a whole.

**Integration Challenges**: Automatic wrapper generation can lead to integration challenges, as the wrapper code needs to be integrated with the existing system. This can be difficult if the original code was not designed to work with wrappers.

**Compatibility Issues:** Automatic wrapper generation can also lead to compatibility issues, as the wrapper code may not be compatible with the original code or with other components in the system.

**Maintenance Overhead:** Wrapper code generated by automatic wrapper generation can be difficult to maintain, as it is often highly specific to the original code and may require specialized knowledge to modify or update.

Overall, while automatic wrapper generation can be a useful technique for integrating software components, it is important to consider the potential problems that can arise and to carefully evaluate whether this technique is appropriate for a given system.

**Problem 1: Extraction Based on a Single List Page**

Input: A single HTML string S, which contains k non-overlapping substrings s1, s2, …, sk with each si encoding an instance of a set type. That is, each si contains a collection Wi of mi (≥ 2) non-overlapping substrings encoding mi instances of a tuple type.

Output: k tuple types σ1, σ2, …, σk, and k collections C1, C2, …, Ck of instances of the tuple types such that for each collection Ci there is a HTML encoding function enci such that enci: Ci → Wi is a bijection

Given an HTML string S that contains k non-overlapping substrings s1, s2, …, sk, where each si represents a set of instances of a tuple type, you want to extract k tuple types σ1, σ2, …, σk and k collections C1, C2, …, Ck of instances of the tuple types such that there is a bijection between each collection Ci and the corresponding set Wi.

To solve this problem, you can use a combination of parsing techniques and regular expressions. The first step is to parse the HTML string S and extract each of the substrings s1, s2, …, sk. You can use an HTML parser such as BeautifulSoup or lxml for this task.

Next, you can use regular expressions to extract the tuples from each substring si. Each tuple type σi can be represented by a dictionary or a named tuple with keys or fields corresponding to the sub-substrings that encode the tuple. For example, if a sub-substring encodes a tuple with fields "name" and "age", then the corresponding dictionary or named tuple would have keys or fields "name" and "age".

Once you have extracted the tuples for each substring si, you can create a collection Ci of instances of the tuple type σi. The bijection between Ci and the corresponding set Wi can be achieved by defining a function enci that maps each instance of σi in Ci to the corresponding sub-substrings in Wi.

Overall, the steps to solve this problem are as follows:

1. Parse the HTML string S and extract each of the substrings s1, s2, …, sk using an HTML parser such as BeautifulSoup or lxml.
2. For each substring si, use regular expressions to extract the tuples and represent them as dictionaries or named tuples with keys or fields corresponding to the sub-substrings that encode the tuple.
3. Create a collection Ci of instances of the tuple type σi from the extracted tuples.
4. Define a function enci that maps each instance of σi in Ci to the corresponding sub-substrings in Wi.
5. Repeat steps 2-4 for each substring si.
6. Return the tuple types σ1, σ2, …, σk, and collections C1, C2, …, Ck of instances of the tuple types such that there is a bijection between each collection Ci and the corresponding set Wi.

**Problem 2: Extraction Based on Multiple Pages**

Input: A collection W of k HTML strings, which encodes k instances of the same type.

Output: A type σ, and a collection C of instances of type σ, such that there is a HTML encoding enc such that enc: C → W is a bijection.

Given a collection W of k HTML strings, each of which encodes an instance of the same type, you want to infer the schema of the type and extract a collection C of instances of the type such that there is a bijection between the instances in C and the corresponding HTML strings in W.

To solve this problem, you can use a combination of clustering, parsing, and schema inference techniques. The first step is to cluster the HTML strings in W based on their structural similarities. You can use clustering algorithms such as k-means or hierarchical clustering for this task.

Next, for each cluster of HTML strings, you can use an HTML parser such as BeautifulSoup or lxml to parse the strings and extract the structured data. You can then use schema inference techniques such as frequency analysis, association rule mining, or machine learning to infer the schema of the type.

Once you have inferred the schema of the type, you can create a collection C of instances of the type from the extracted structured data. The bijection between C and W can be achieved by defining a function enc that maps each instance in C to the corresponding HTML string in W.

Overall, the steps to solve this problem are as follows:

1. Cluster the HTML strings in W based on their structural similarities using clustering algorithms such as k-means or hierarchical clustering.
2. For each cluster of HTML strings, use an HTML parser such as BeautifulSoup or lxml to parse the strings and extract the structured data.
3. Use schema inference techniques such as frequency analysis, association rule mining, or machine learning to infer the schema of the type.
4. Create a collection C of instances of the type from the extracted structured data.
5. Define a function enc that maps each instance in C to the corresponding HTML string in W.
6. Return the inferred type σ and collection C of instances of the type such that there is a bijection between C and W.

**Building DOM Trees**

The Document Object Model (DOM) is a programming interface for HTML and XML documents. It represents the page so that programs can change the document structure, style, and content.

To build a DOM tree, we start with the HTML code, which is parsed by the web browser into a tree structure. Each element in the tree is represented by a node. There are different types of nodes, such as element nodes, attribute nodes, and text nodes.

Here's an example of HTML code and its corresponding DOM tree:

<html>

<head>

<title>My Website</title>

</head>

<body>

<header>

<h1>Welcome to My Website</h1>

<nav>

<ul>

<li><a href="#">Home</a></li>

<li><a href="#">About</a></li>

<li><a href="#">Contact</a></li>

</ul>

</nav>

</header>

<main>

<article>

<h2>My First Article</h2>

<p>Lorem ipsum dolor sit amet, consectetur adipiscing elit. Sed ac lectus eget nunc dictum lobortis sed sit amet neque. Nullam luctus est eu libero venenatis pharetra. </p>

</article>

<article>

<h2>My Second Article</h2>

<p>Etiam convallis tellus id ipsum congue, ut bibendum quam molestie. Maecenas faucibus ipsum sed tellus imperdiet, in cursus ex bibendum. </p>

</article>

</main>

<footer>

<p>&copy; 2023 My Website. All rights reserved.</p>

</footer>

</body>

</html>

- Document

- Doctype: html

- Element: html

- Element: head

- Element: title

- Text: "My Website"

- Element: body

- Element: header

- Element: h1

- Text: "Welcome to My Website"

- Element: nav

- Element: ul

- Element: li

- Element: a

- Text: "Home"

- Element: li

- Element: a

- Text: "About"

- Element: li

- Element: a

- Text: "Contact"

- Element: main

- Element: article

- Element: h2

- Text: "My First Article"

- Element: p

- Text: "Lorem ipsum dolor sit amet, consectetur adipiscing elit. Sed ac lectus eget nunc dictum lobortis sed sit amet neque. Nullam luctus est eu libero venenatis pharetra. "

- Element: article

- Element: h2

- Text: "My Second Article"

- Element: p

- Text: "Etiam convallis tellus id ipsum congue, ut bibendum quam molestie. Maecenas faucibus ipsum sed tellus imperdiet, in cursus ex bibendum. "

- Element: footer

- Element: p

- Text: "&copy; 2023 My Website. All rights reserved."