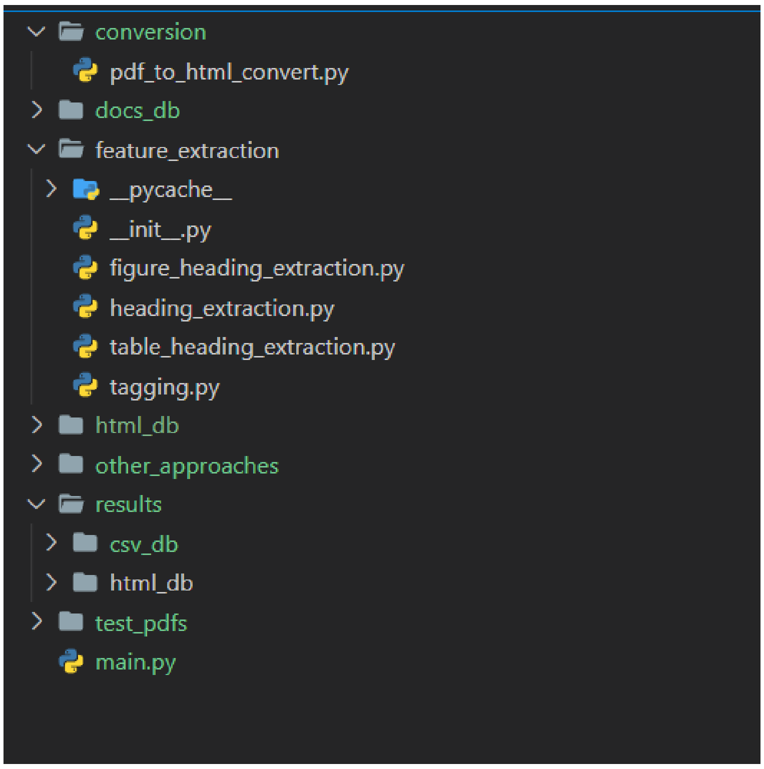
Directory Structure



```
conversion
    pdf_to_html_convert.py
docs_db
feature_extraction
    __pycache__
    __init__.py
    figure_heading_extraction.py
    heading_extraction.py
    table_heading_extraction.py
    tagging.py
html_db
other_approaches
results
    csv_db
    html_db
test_pdfs
main.py
```

The Directory Structure consists of the following folders

1.conversion Directory:

The Conversion directory consists of pdf_to_html_convert.py script.

1.1 pdf_to_html_convert.py: This script converts the pdf document taken as input to HTML document using the cloud convert API.

2. docs_db directory:

The docs_db directory will consist of the word document files which are converted from the input pdf file and used for header footer extraction.

3.feature_extraction directory:

This directory consists of scripts which help in extracting features from the converted HTML document and tagging the document to create a final output. The functionalities of all scripts are given below. Moreover, the code in the files is also commented properly for better understanding.

3.1 __init__.py:

The code in this file will be executed whenever the feature extraction module is called.

3.2 figure_heading_extraction.py:

The Python script that reads in CSV files from a specified directory, adds a new column to each CSV file called "is_figure_heading", and updates the rows in each file based on whether they contain text that is likely a figure heading.

First, the code imports the pandas library as pd and the os library to manage file paths. It then defines a list of potential keywords that might be present in figure headings, called pool.

Next, the code defines the directory where the CSV files are stored, csv_dir, and uses the os library to get a list of all the file names in that directory. For each file in the directory, the code opens the file using pd.read_csv(), reads the contents into a DataFrame called df, and adds a new column to the DataFrame called "is_figure_heading" with a default value of False.

The code then loops through each row in the DataFrame using df.iterrows(). For each row, it extracts the text from the "Text" column, splits it into individual words using .split(), and checks if any of the first three words match a word from the pool list of potential keywords for figure headings. If a match is found, the code sets the "is_figure_heading" value for that row to True, and checks if the row is already marked as a heading in the "is_heading" column. If it is, the code sets the "is_heading" value for that row to False.

After looping through all rows in the DataFrame, the updated DataFrame is saved back to the original CSV file using df.to_csv(), with the index parameter set to False to exclude the row numbers. Finally, the code prints a message to indicate which file was updated with figure headings.

3.3 heading_extraction.py:

First, it defines a list of words called pool which contain words that are usually used as headings. It also defines a function is_heading(text) which returns True if the text matches any of the words in the pool list. It then creates a directory called results/csv_db to store the output CSV files.

Next, it lists all the HTML files in the directory specified in the html_dir variable and then loops through each file. It opens each file and reads its contents, then uses the BeautifulSoup library to parse the HTML. It then defines regular expressions to match specific patterns that are commonly used as headings.

After that, it creates an empty list called data to store the extracted data. It then iterates through each heading in the parsed HTML and extracts the text, checks if the text is all in capital letters, and checks if the text matches the regular expressions defined earlier. It also checks if the text is a heading by calling the is_heading() function. It then appends the extracted data to the data list.

Finally, it creates a Pandas DataFrame from the data list and saves it as a CSV file with the same name as the HTML file in the results/csv_db directory.

3.4 table_heading_extraction.py:

This code reads CSV files from a specified directory and updates them by adding a new column "is_table_heading" to each file. It then loops through each row in the DataFrame and checks if any of the first three words of the "Text" column contain a word from the pool, which contains several variations of the word "Table". If a match is found, the value of the "is_table_heading" column is set to True for that row. If the text is already marked as a heading or figure heading, the code unmarks them by setting their corresponding values to False. Finally, the updated DataFrame is saved to the same CSV file.

Here's a breakdown of the main parts of the code:

- Import the required libraries: pandas and os.
- Define the word pool as a list of variations of the word "Table".
- Set the directory where the CSV files are stored.
- Get a list of all the CSV files in the directory.
- Loop through each file in the directory:
- Read the CSV file into a pandas DataFrame.
- Add a new column "is_table_heading" to the DataFrame and initialize all values to False.
- Loop through each row in the DataFrame:
- Get the text from the "Text" column of the current row.
- Split the text into words.
- Check if any of the first three words of the text contain a word from the pool.
- If a match is found, set the value of the "is_table_heading" column to True for that row.
- If the text is already marked as a heading or figure heading, unmark them by setting their corresponding values to False.
- Save the updated DataFrame to the same CSV file.
- Print a message indicating that the file has been updated with table headings.

3.5 tagging.py

This code reads HTML files and corresponding CSV files, adds classes to certain div elements in the HTML file based on information in the CSV file, and saves the modified HTML file with the added classes to a new file.

The code first imports the necessary modules - pandas and BeautifulSoup - and defines three word pools (fig_pool, tab_pool, and sh_pool) that contain words that are commonly found in figure headings, table headings, and section headings, respectively. It also defines a function is_heading() that checks whether a given text is a section heading.

The code then sets the paths for the CSV and HTML directories, and checks if a results directory exists, creating one if it does not. It reads the filenames in both directories and uses a for loop to iterate over pairs of corresponding HTML and CSV files. Within the for loop, the code loads the CSV file into a pandas DataFrame and the HTML file into a BeautifulSoup object. It then finds all div elements in the HTML file with classes that match certain predefined classes (div_class). For each div, it checks if the corresponding row in the CSV file has the is_heading section marked as True, in which case it adds the s_heading class to the div's class attribute. If the is_figure_heading section is marked as True and any of the first three words in the Text column of the CSV file match words in the fig_pool, it adds the figure_heading class to the div's class attribute. Similarly, if the is_table_heading section is marked as True and any of the first three words in the Text column of the CSV file match words in the tab_pool, it adds the table_heading class to the div's class attribute.

Finally, the code saves the modified HTML to a new file in the results folder with the same filename as the original HTML file but with "_output" added to the filename. The code also prints a message indicating that the HTML file has been tagged and saved.

3.6: header_footer_extraction.py:

The code processes a collection of Microsoft Word DOCX files located in an input directory, and extracts the header and footer from each file. It then writes the extracted header and footer to a corresponding CSV file in an output directory.

The code uses the following libraries:

- os: to work with file paths and directories.
- zipfile: to open the DOCX file as a zip file and access its contents.
- BeautifulSoup: to parse the XML files inside the DOCX file and extract text content from them.
- csv: to write the extracted header and footer to a CSV file.

Here are the main steps of the code:

Define a function extract_header_footer that takes a path to a DOCX file as input and returns the header and footer text content from the file as a tuple. The function first opens the DOCX file as a zip file and checks if the header and footer XML files exist in the zip file. If they exist, the function uses BeautifulSoup to parse the XML files and extract the text content from them. The function then returns the header and footer text content as a tuple. Define a function is_empty that takes a string as input and returns True if the string is empty or contains only whitespace characters, and False otherwise. Define input and output directories as input_dir and output_dir, respectively. If the output directory does not exist, create it using os.makedirs(output_dir). Loop through all files in the input directory using os.listdir(input_dir). For each file, check if it has a .docx extension using file_name.endswith('.docx').

If the file is a DOCX file, extract the header and footer text content from the file using the extract_header_footer function. Construct the output file path by replacing the extension with .csv. Open the output file using open(output_file, 'w', newline='').

Write the header row to the file using csv.writer.writerow(['Text', 'is_header', 'is_footer']).

If the header text is not empty, write it to the CSV file with the is_header flag set to True. If the footer text is not empty, write it to the CSV file with the is_footer flag set to True. If both header and footer text are empty, write an empty row to the CSV file with both is_header and is_footer flags set to False. The row is written using csv.writer.writerow(). The loop continues with the next file in the input directory, until all files have been processed.

4. html_db directory:

This directory stores the converted HTML Document obtained after running the pdf_to_html_convert.py script in the conversion directory.

5. Other approaches (Valid for research purposes only):

This directory consists of the dropped down approaches.

6. result_db directory:

This directory consists of two more directories:

1.html_db: Stores the final output in the form of a tagged HTML document.

2.csv_db: Stores the intermediate csv file which is used for tagging the final output Document. Obtained after running the scripts in the feature extraction directory in the following order:

- heading_extraction()
- figure_heading_extraction()
- table_heading_extraction()
- tagging()

7.main.py :

Executing this file runs the entire framework.