Documentation

# classes

# technologies, libraries and other names

# methods

My approach for solving the assignment was to first create a crawler that will scrape the furniture names from the first 100 links and label them and then train a NER model using BERT to be able to identify the furniture names from the rest of the links.

Crawler

HIGH-LEVEL APPROACH:
To implement the web crawler, I utilised Selenium. Initially, I created a Python class named **Extraction**, designed to navigate each link from the CSV and retrieve relevant data. My initial approach involved extracting the entire HTML from each webpage in order to identify potential furniture names for labeling. Data processing was done using the **Process_Data** class, where I experimented with Spacy to establish rules for extracting furniture names from the HTML. However, this strategy proved inefficient due to the diverse nature of the text.

I adjusted my approach by extracting "h" tags from the HTML of each page. Upon inspecting several web pages, I came to the conclusion that furniture names were often displayed using "h" tags. Following the extraction of these tags, I encountered a mix of furniture names and other text, such as "Contact Us" and "Sale only today." I implemented methods to preprocess data by eliminating stop words, removing newline characters, and certain punctuation. Stemming and Lemmatization were not applied as they alter the original words within the names. Additionally, I created a function to apply heuristics for extracting furniture names using Spacy. This function filtered out "h" titles containing pronouns, verbs, and adverbs, considering contextual relevance. For instance, in the phrase "Hamar Plant Stand," the algorithm recognised "stand" as a noun, not a verb, preserving its inclusion. To enhance efficiency, I created a list of popular furniture names, and if a word containing a verb lacked clear context, the phrase was retained if it included any of the popular furniture names.

The heuristical approach does not guarantee 100% efficiency, but it still establishes guidelines for the extraction of furniture names, eliminating the need for manual labeling. Following the extraction of names, I proceeded to label and save them in a CSV, manually inspecting for errors and removing non-furniture text.

Upon script execution, I observed that accessing each link and extracting names from the respective pages provided insufficient training data for the model. To address this, I introduced another class, **Exploration**, which recursively traverses links from the initial webpage, generating additional links and corresponding furniture names. The function collects all "a" tags from the starting pages, appends the discovered links to a list, and subsequently accesses each link individually, feeding it as an argument to the recursive function.

I created two run files one that extracts the training data and one that extracts the test data(containing furniture names and also other text).

CLASSES AND METHODS:

1. **Exploration**:
   - *get_links(filename):* gets all the links from the CSV and returns a list
   - *is_valid_https_link(link):* checks using regex if a link is a valid link
   - *extract_after_https(link):* function that checks if the domain of a link is in the list of initial link because some web pages may redirect you to other websites like shopify or google
   - *get_all_links(initial_links, depth, visited):* recursive function that explores links depth of exploration is by default set to 3. The function returns a list with all the links

2. **Extraction**:
   - *get_url_content(url):* it extract "h" elements and raw HTML data. It uses web driver waits to wait for the presence of elements an discard inaccessible links
   - *get_data():* processes extracted data by *get_url_content* and returns a list of strings containing furniture names and other text
   - *close():* closes web driver

3. **Process_Data**:
   - *get_data():* calls *get_data* from the **Extraction** class
   - *preprocess(text):* removes stop words, certain punctuation, newline characters
   - *preprocess_raw_data():* uses *preprocess* and returns a list of all the separate phrases within the HTML
   - *preprocess_h_data():* uses *preprocess* and returns a list of all the preprocessed "h" elements
   - *contains_verb_adverb_pronoun_in_context():* sets heuristics
   - *heuristic_matching_h_data():* applies heuristics
   - *label_training_data():* labels data after preprocessing and applying heuristics

Note

Neither the training data nor the test data were extracted using recursion. I tried running data extraction using recursion but it did not complete running the script on ~100 links not even after 72 hours so data is extracted from just the main page of each link. Being a PoC and recursion being implemented, I hope data extraction lives up to the expectations as it can later be ran with recursion on a more powerful machine.

BERT

HIGH-LEVEL APPROACH:

Steps:

1. Load the data from CSV and transform PRODUCT label to Integer
2. Create Tokenizer
3. Tokenise data
4. Convert labels to PyTorch tensor
5. Create DataSet
6. Split data into training and validation (validation size = 0.1)
7. Define DataLoader and define batch_size
8. Import BertForSequenceClassification model from transformers and set num_labels = 1
9. Define Adam as optimiser
10. BCEWithLogitsLoss as loss function (best for loss function for classification within NNs)
11. Model:
    - create training loop
    - iterate over batches set by loader
    - reset gradient from previous batch
    - compute the loss
    - propagate the loss and update model parameters using optimiser
    - acumulate batch loss to the total loss for the epoch
    - disables gradient computation during validation to reduce memory usage
    - passes the validation data through the model and computes predictions
    - the sigmoid function is applied to the model logits to obtain probabilities
    - calculates and prints the validation accuracy

I explored various combinations of batch sizes, epochs, and learning rates. Choosing lower learning rates proved effective, particularly for smaller datasets. Additionally, smaller batch sizes were found to be optimal for working with these datasets.

As for normalisation I just lowercased everything with the exception of every first letter of each word.

After training the model, display a plot.

After experimenting for a little while I researched more techniques to improve my model and I implemented:

1. Learning Rate Scheduling to adjust the learning rate during training, potentially helping the model converge faster and achieve better performance
2. Early Stopping which based on the validation loss prevents overfitting. If the validation loss starts decreasing or increasing, stop training early to avoid overfitting.

Data is displayed in a separate file inside data/merged_data.csv. This file contains the data from the first ~100 links and also data extracted from the rest of the links that the model recognised as furniture names.

ISSUES:

The main came from the training data exclusively featuring positive labels, consisting of just furniture names without any non-furniture names. Another issue was the huge diversity and proper nouns among the furniture names. As this is a PoC, performance is not optimal, taking into account the issues stated above.