## Web Object Extraction

The goal of this project was to extract title, image and price (TIP) of a product from a given webpage, and draw bounding boxes around it. The motivation behind this was to create a large dataset of web page with such bounding boxes around TIP elements, which can be later fed into an ML model for further learning.

*Programming Language*: Python 3, Selenium

### Acquiring List of Product Web Pages

First task is to search for a keyword on "Google Shopping" page, and then get the top 10 pages from the result's page. Each page displays a list of products which when clicked shows the availability of the product by different vendors (for instance, websites like Amazon, eBay) along with their price. Now, we scrape all of these vendor sites for all the products displayed on the top 10 pages. For each vendor page, we perform web object extraction module.

### TIP Extraction Module

For this module, multiple techniques were used, and required intensive testing to check if the algorithm works for various kinds of websites. This module is pretty hard to implement since the websites are coded in a very different manner, and figuring out where the TIP elements are located requires a smart algorithm.

The smart way would be to use the information mentioned on the Google Shopping page to identify the product's TIP on the vendor page. Thus, while acquiring the list of vendor pages, we also store the TIP for that vendor page. Please note that the image changes from Google Shopping page to vendor page, so mapping the image elements is most difficult compared to price and title element.

Initially, I used the technique of "Template Matching", where you match two images to each other and check if the two images are the same or not. If they are the same, then the element has been found. This was not the best way to find the element because this technique is absolute in nature and doesn't use the fact that "tip, price and title maintain locality towards each other, that is all three elements appear close to each other in a webpage". Thus, we used the "Confidence based element detection".

### Finding confidence of an element

Find at least one label from TIP (Title, Image and Price) which has "High" Probability and then use that element to find other elements. "High" is given by following threshold values for each case.

#### 1. Price:

*Text Similarity Algorithm*: SequenceMatcher from diff library (Python 3 lib)

```
if sim(price)>0.8:
        High
elif sim(price)>0.5:
        Medium
else:
        Low
```

#### 2. Title:

*Text Similarity Algorithm*: SequenceMatcher from diff library (Python 3 lib)

```
if sim(heading)>0.7:
        High
elif sim(heading)>0.5:
        Medium
else:
        Low
```

Which HTML tags to consider for finding the title and price element?

1. <h*> Heading tags

2.  tags

3. Any tags which has "price"/"title" string present in their class/id/name attribute for Price and Heading Labelling respectively.

Performed in order of preference: One only goes down the following priority, if current tag didn't give atleast 'Medium' similarity.

Eg: If the heading <h*> tag's element gave a similarity of Medium/High, then you have found the right element and you don't go further, or else you will have to go to search for your element in next priority order tag which is  and so on.

### 3. Images:

*Image Similarity Algorithm*: AverageHashing from ImageHash library (Python 3 lib)

```
if sim(image)>0.7:
        High
elif sim(image)>0.5:
        Medium
else:
        Low
```

Other features to consider while finding prospective images for aforementioned similarity task:

1. The top part of the image should be in the fourth quadrant of the page.

2. If more than one image produces a 'High' similarity then choose the element with the biggest size.

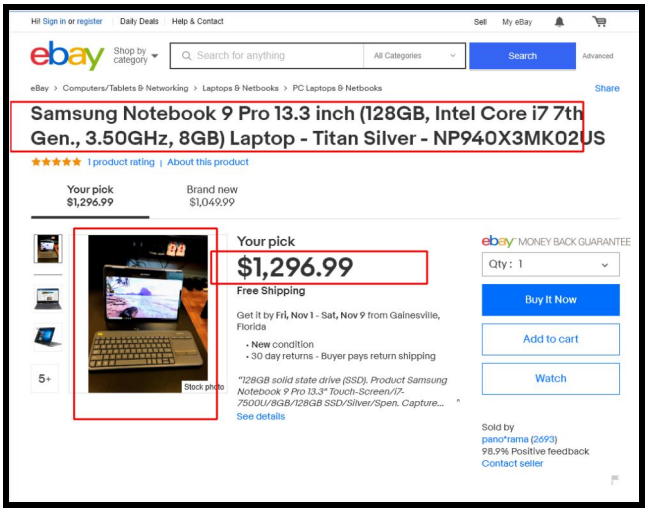


*Figure 6: Bounded Boxes drawn around TIP on an eBay page when "Samsung Notebook" was searched on Google Shopping page.*

The "Confidence based element detection" gave around 70% accuracy, so if one needs to implement it then human supervision is also required in the end to filter the false positives and negatives. The future work would be to combine template matching with the confidence-based detection algorithm, because there were few edge cases where template matching performed better.