CS784: Data Models and Languages

Project Stage II
Brand Name Extraction

Date: March 14, 2016

Group Member Information
Artsiom Hovarau: hovarau@cs.wisc.edu
Apul Jain: apul@cs.wisc.edu
Zhuowei Cai: zhuoweic@cs.wisc.edu

Website: http://pages.cs.wisc.edu/~zhuoweic/index.html
In this project stage we analyzed electronic product names and developed an extractor to extract brand names from the product names. For this task, we used dictionary-based approach. We randomly sampled 350 product item from a total of 10K products and split this set of items (Set S) into two sets: Set I and Set J. Set I consists of 200 product items while Set I consists of 120. Set I was used as a development set to develop our extractor while Set J was used to test how well our extractor behaves.

Extractor Algorithm:

In this stage, we used dictionary-based approach. Initially we are given a dictionary of 8442 brand names. We first expanded this dictionary with brand names extracted from the development set I. Our extraction algorithm consists of following steps:

**Step 1: Data Acquisition:**

**Sample data to form set S.** This data will be split into two sets - development set I and test set J. We sampled 350 records to form set S and then split it into 200 and 150 for Set I and Set J, respectively. We then manually extracted the brand names for each of the product names. After this step, we had records in the following format:

Format of product records:

```
prod.id?prod_name?brand_name
```

Note that for some records we couldn’t figure out the brand names from the product name, these were labeled as records having missing brand names (represented by an empty string).

We also used the dictionary D of plausible brand names given to us.

**Step 2: Analysis of brand names from the development set I:**

**Dictionary Expansion:**

Parse the development set (Set I) and pull out the brand names. Merge these extracted brand names with our dictionary D.

**Step 3: Brand name extraction:**

**Dictionary lookup:**

Given a product name, we do a substring matching for each plausible brand name in the product name. If we find no match, then the brand name extracted is empty. If we have more than one collisions in the brand name, we infer its brand name using the rules in the precedence corresponding to their order. We will elaborate the rules that are used in the section below.
**Step 4: Calculation of Precision and Recall**

To calculate precision and recall, we defined False Positive (FP), True Positive (TP), True Negative (TN), False Negative (FN) as follows:

<table>
<thead>
<tr>
<th>Actual brand name</th>
<th>Extracted brand name</th>
<th>Classification</th>
</tr>
</thead>
<tbody>
<tr>
<td>ABC</td>
<td>ABC</td>
<td>True Positive (non-empty match with non-empty brand name)</td>
</tr>
<tr>
<td>ABC</td>
<td>XYZ</td>
<td>False Positive (non-empty mismatch with non-empty brand name)</td>
</tr>
<tr>
<td>&quot;&quot;</td>
<td>ABC</td>
<td>False Positive (non-empty mismatch with empty brand name)</td>
</tr>
<tr>
<td>ABC</td>
<td>&quot;&quot;</td>
<td>False Negative (empty mismatch with non-empty brand name)</td>
</tr>
<tr>
<td>&quot;&quot;</td>
<td>&quot;&quot;</td>
<td>True Negative (empty match with empty brand name)</td>
</tr>
</tbody>
</table>

Precision = \( \frac{TP}{TP + FP} \)

Recall = \( \frac{TP}{TP + FN} \)

**Results:**

Based on our algorithm, we saw following results:

**Precision:** 96.06%
**Recall:** 92.42%
Rules to handle multiple plausible brand names match:

**Assumption:** Brand name is a consecutive sequence of words. We made this assumption so that we can do substring matching between the brand name and the product name.

**Rule 1:** Brand name closer to the beginning first
The basic idea behind this rule is that we prefer brand name match which appears closer to the beginning of the product name.

**Example:**

ASUS T100TAM Transformer Book Intel Atom 2GB Memory 64GB SSD 10.1
2-in-1 Brushed Aluminum Notebook Windows 8.1 + Micro?ASUS

In this case as you can see, there are two plausible brand names based on dictionary lookup: ASUS and Intel. In order to correctly extract brand name, we must prefer ASUS and not Intel. Based on this rule, we see that ASUS appears before Intel so our algorithm would pick ASUS rather than Intel. This also follows our general observation that usually product names start with brand name when we go for shopping on e-commerce websites like Amazon or Walmart (since they are the most important attribute of a product).

\[
\text{match_index("ASUS") < match_index("Intel") \Rightarrow brand_name = "ASUS"}
\]

**Rule 2:** Longest matching brand name first:

This rule handles cases where brand names itself consists of multiple parts and each part individually can refer to a brand name in itself, which is complementary to Rule 1.

**Example:**

11038058?Belkin Mobile Retractable USB Mouse - Black?Belkin Mobile

For this example, we see that both Belkin and Belkin Mobile are valid brand names based on the dictionary lookup. Our algorithm would extract Belkin Mobile because its length is larger than length of Belkin alone.

\[
\text{len("Belkin Mobile") > len("Belkin") \Rightarrow brand_name = "Belkin Mobile"}
\]

**Rule 3:** Suffix rule:

We maintain a suffix array:

\[
\text{SUFFIXES } = ["Inc.", "Corp.", "Incorporation", "Corporation", "Technology", "Ltd.", "Limited"]
\]

This rule says that for each plausible brand name, search for \( \text{brand_name} + \text{<suffix>} \) in the product name, and if we find a match we return \( \text{brand_name} + \text{<suffix>} \) as the extracted brand name. This is essentially an extension of Rule 2.

For this product, we found only “Biltwell” in our dictionary. But based on the suffix rule, we would search for “Biltwell + <suffix in SUFIXES>” and we will hit a match for “Biltwell Inc.” So our algorithm would return “Biltwell Inc.”

match(“Biltwell” + “ Inc.”) == true ⇒ brand_name = “Biltwell Inc.”
Analysis of Precision/Recall further improvement:

We see that precision and recall mainly depend upon False positives and False negatives. To understand effect of False positives/negatives on precision/recall, we observed two factors affecting our precision:

Reason for no-further-improvement in precision/recall:

1. Brand name mismatch due to multiple similar brand names (for precision):

   Example: False positive due to wrong brand name prediction
   42397735#TigerDirect?Kingston ValueRAM - DDR3 - 16 GB - DIMM 240-pin - 1600 MHz / PC3-12800 - CL11 - 1.5 V - registered with parity - ECC?Kingston ValueRAM

   True brand name: Kingston ValueRAM
   Predicted brand name: Kingston

   In this case Kingston is present in our dictionary but not "Kingston ValueRAM". So this results in false positive by our algorithm.

2. Brand name not found in dictionary (for recall first, then for precision):

   Example: False negative due to missing brand name in dictionary
   20850274?Winslow TV Stand, for TVs up to 46, Espresso?Winslow

   True brand name: Winslow
   Predicted brand name: ""

   In this case no brand name from our dictionary is present in the product name. So our algorithm fails to extract Winslow and results in false negative. In fact, we consider using a non-brand-name dictionary to get rid of the common words from the product name. But the size of the non-brand-name dictionary is huge, and it turns out that it does not work quite well.

Solution:

We finally came up with another idea that will significantly reduce the size of the dictionary we need and could possibly offer better performance (we did not have time to test this idea though). That is, to maintain a product dictionary. For example, this dictionary consists of the name of the product like

   TV Stand
   Case
   Refrigerator
   ...
And usually the brand name will be the substring that immediately precedes the name of the product. So it is possible turn to the product dictionary in order to get ultimate performance and even pull up the above parameters in the future. This also seems to be quite a reasonable heuristics to use.
Appendix:

Code to illustrate our extraction rules: [Please refer to actual code for complete details]

# Extract brand name from product name.
# Expecting the input data (development) and test files
# containing record in the form of
# id?product_name?brand_name
# ...
# The second argument corresponds to the dictionary used.
# Dictionary Expansion: merge brand names in the development set with the given dictionary
merge(dictfile, datafile)

# do the actual testing
extracted_brand = list()
for rec in test:
    match = ""
    index = len(rec[PROD])
    product_name = " " + rec[PROD].lower() + " 
    for brand in dict:
        guess_brand = " " + brand.lower().strip() + " 

        try:
            pos = product_name.index(guess_brand)
        except ValueError:
            pos = index + 1
    id = get_id([rec])

    # Rule 1: Leading brand name match is preferred
    if (pos < index) or ((pos == index) and (len(match) < len(brand))):
        index = pos
        match = brand

    # Rule 2: Longest matching brand name is preferred
    if ((pos == index) and (len(match) < len(brand))):
        index = pos
        match = brand

    # Rule 3: Suffix rule: try finding suffixes
    for suffix in SUFFIXES:
        if (" " + match + " " + suffix + " ").lower() in product_name:
            match = match + " " + suffix
            break

    extracted_brand.append(match)

# calculate precision and recall
compute_pr(get_id(test), get_truth(test), extracted_brand)
tp, brand: V7, extracted: V7 id: 40870909
tp, brand: Seidio, extracted: Seidio id: 41981072
tp, brand: Corsair, extracted: Corsair id: 42508283#TigerDirect
fn, brand: Cocoa Touch, extracted: id: 11089046#Walmart.com
tp, brand: APC, extracted: APC id: 40871410
fp, brand: Dell Latitude, extracted: Dell id: 41177187
tp, brand: Idatalink, extracted: Idatalink id: 42517022#HappEshopper
tp, brand: Innovera, extracted: Innovera id: 14922688
tp, brand: Digi, extracted: Digi id: 41195472
tp, brand: LaCie, extracted: LaCie id: 11016993#Walmart.com
tp, brand: C2G, extracted: C2G id: 40984286#TEKENVY
tp, brand: PNY, extracted: PNY id: 40987111
tp, brand: Xerox, extracted: Xerox id: 42509754
tp, brand: Corlink, extracted: Corlink id: 42462509#TigerDirect
tp, brand: Intel, extracted: Intel id: 42814082#TigerDirect
tp, brand: Pelican, extracted: Pelican id: 41493819#TEKENVY
tp, brand: Belkin, extracted: Belkin id: 8223016
tp, brand: Tripp Lite, extracted: Tripp Lite id: 11077951
tp, brand: Dataproducts, extracted: Dataproducts id: 19311008
tp, brand: roocase, extracted: roocase id: 42398267#TigerDirect
fn, brand: Ambir, extracted: id: 17046055
tp, brand: Amped Wireless, extracted: Amped Wireless id: 40999925#TigerDirect
tp, brand: Acer, extracted: Acer id: 40871293#TEKENVY
fp, brand: Kingston ValueRAM, extracted: Kingston id: 42397735#TigerDirect
tp, brand: MSI, extracted: MSI id: 40871056
tp, brand: V7, extracted: V7 id: 41812441
tp, brand: Middle Atlantic, extracted: Middle Atlantic id: 11462848#Wayfair
tp, brand: Creative Concepts, extracted: Creative Concepts id: 21576697#UnbeatableSale.com
fn, brand: Winslow, extracted: id: 20850274
tp, brand: Memorex, extracted: Memorex id: 11013678#Walmart.com
tp, brand: GN, extracted: GN id: 40869983
precision: 0.96062992126 recall: 0.924242424242
true positive: 122 true negative: 13
false positive: 5 false negative: 10