Effective Unsupervised Matching of Product Titles with k-Combinations and Permutations

Leonidas Akritidis, Panayiotis Bozanis
Department of Electrical and Computer Engineering
University of Thessaly, Greece
The problem (1)

• We are given a set of $F = \{f_1, f_2, \ldots, f_N\}$ product feeds (usually in XML format).
• Each feed $f_i$ originates from an electronic store $e_i$ and contains product records.
• Each product record $p$ may contain multiple fields (title, description, price, brand, category, etc).
• A product cannot appear more than once in the same feed.
• But it may appear in multiple feeds.
The problem (2)

• A product may be described differently in these feeds (i.e. it appears under different titles).
• E.g. “Apple iPhone 7” and “iPhone 7” are different titles which refer to the same product.
• The problem: Match the product titles and identify if they describe the same product.
• Useful for:
  – Price comparison applications & platforms.
  – Reviews merging & aggregation.
  – Users who desire to compare characteristics & prices.
Similarity/Distance Metrics

• “Apple iPhone 7” and “iPhone 7” are different titles which refer to the same product.
  – Even though a whole word is missing from the second title (small similarity/distance).

• “Apple iPhone 7” and “Apple iPhone 6” are titles which DO NOT refer to the same product.
  – Even though they only differ by a single character (higher similarity/distance).

• Similarity/Distance metrics (cosine, Jaccard, edit distance, etc.) do not work well in this problem.
Supervised Clustering

• For the same reason, the supervised machine learning clustering approaches (kNN, naïve Bayes, linear/logistic regression) also do not work well.
  – Smaller distances/higher probabilities should not necessarily be clustered to the same entity.
  – Higher distances/smaller probabilities should not necessarily be clustered in different entities.
State-of-the-art (1)


State-of-the-art (2)

• These approaches are similar:
  – They enrich each product title by injecting several missing words.
  – They treat each word in the products’ titles differently, i.e. each word is assigned an importance score.
  – After these two preprocessing phases, they apply the cosine similarity measure (with an over simplistic blocking method).
  – They create clusters which consist of the same products.
State-of-the-art - Disadvantages

• One query submitted to a SE per product:
  – this approach is infeasible for large-scale datasets.

• In their experiments they use only 2 feeds.
  – Most platforms include thousands of electronic stores
    (i.e. product feeds).

• They employ the cosine similarity metric.
  – which does not perform well in this problem.
Our approach is...

- Standalone: It does not rely on external data sources (i.e. Web search engines, Web sites, etc).
- Unsupervised: No requirement to manually train a classifier, or split the dataset in training and testing data subsets.
- Efficient: Faster than the adversary approach; it makes use of in-memory data structures.
- Flexible: It facilitates product classification into multiple clusters.
Overview (1)

• Our proposed method operates in 2 phases:
• Phase 1: construction of two primary data structures:
  – A lexicon which consists of all the \( k \)-combinations of the titles’ words, along with a frequency value and some statistics.
    • Each \( k \)-combination is a candidate product cluster.
  – A forward index: An array which stores for each product, a list of pointers to the respective title \( k \)-combinations (we use pointers to avoid saving the same data twice).
Phase 2: We employ these two data structures to assign scores to each $k$-combination of each product.

The $k$-combinations are then sorted by decreasing score value and the highest scoring combination represents the cluster.
$k$-combinations

- $k$-combinations are combinations of the words of the product title.
- Length (number of words) = $k$.
- Without repetition.
- Without care for word ordering.
- We compute the $K$-combinations of each product title, $K \in [2,6]$
- Number of combinations for a title which consists of $n$ words: $$\sum_{k=2}^{K, k \leq n} \frac{n!}{k!(n-k)!}$$
Phase 1 (1)

3 for each product p do
4     extract the title t;
5     perform linguistic processing of t;
6     for each $k \in [2, K]$ do
7         compute all $k$-combinations $C_{(k)}$ of $t$;
Data Structures - Lexicon

• We employ a lexicon structure $L$ to store the combinations. We also store two statistics:
  • A frequency value which represents the number of documents which contain this combination.
    – Frequent combinations are more likely to be declared cluster labels.
  • A distance value which stores the average distance of the combination from the beginning of the titles.
    – The most important terms in a product description appear early in the titles.
Data Structures – Forward Index

• We also employ a forward index $I$ which for each product $p$, stores a pointer to each combination.
• We assign a score value to each combination in $I$. 
Distance

• Some frequent terms in the titles have no informational value (i.e. they do not describe the product, but they contain offers, specs, etc).
  – E.g. many products have in their titles the terms “EU”, “OEM”, “Retail”, etc.
  – Therefore, in some cases we get wrong cluster labels, e.g. “Apple iPhone EU”.
  – Similar problems can also be caused by other words: colors (black, white, red, etc), sizes (large, small, etc) and others.

• Key observation: These terms usually appear late in the title (i.e. in high position).
Algorithm 1: Product titles’ processing and data structures construction

1. initialize the lexicon \( L \);
2. initialize the forward index \( F \);
3. for each product \( p \) do
   4. extract the title \( t \);
   5. perform linguistic processing of \( t \);
   6. for each \( k \in [2, K] \) do
      7. compute all \( k \)-combinations \( C_{(k)} \) of \( t \);
      8. for each \( k \)-combination \( c \in C_{(k)} \) do
         9. Set \( d(c, t) \leftarrow \text{distance}(c, t) \);
         10. \( F \).insert\((p, c)\);
         11. Set \( \text{found} \leftarrow L.\text{search}(c) \);
         12. if \( \text{found} = \text{true} \) then
            13. Set \( c.\text{freq} \leftarrow c.\text{freq} + 1 \);
            14. Set \( c.\text{dist} \leftarrow c.\text{dist} + d(c, t) \);
         15. else
Permutations (3)

• In case a combination is not found in the lexicon, we compute all its permutations.
• We search for each permutation in the lexicon.
• In case it is found, we increase the frequency of the corresponding combination and we stop searching.
• In case it is not found, we do not insert it
• We shall insert the corresponding combination instead, after all the permutations have been examined.
Algorithm 1: Product titles’ processing and data structures construction

1. initialize the lexicon $L$;
2. initialize the forward index $F$;
3. for each product $p$ do
   4. extract the title $t$;
   5. perform linguistic processing of $t$;
   6. for each $k \in [2, K]$ do
      7. compute all $k$-combinations $C_{(k)}$ of $t$;
      8. for each $k$-combination $c \in C_{(k)}$ do
         9. Set $d(c, t) \leftarrow $ distance($c, t$);
         10. $F$.insert($p, c$);
         11. Set $found \leftarrow L$.search($c$);
         12. if $found = true$ then
            13. Set $c$.freq $\leftarrow c$.freq + 1;
            14. Set $c$.dist $\leftarrow c$.dist + $d(c, t)$;
         15. else
            16. compute all permutations $M$ of $c$;
            17. for each permutation $m \in M$ do
               18. Set $found \leftarrow L$.search($m$);
               19. if $found = true$ then
                  20. Set $c$.freq $\leftarrow c$.freq + 1;
                  21. Set $c$.dist $\leftarrow c$.dist + $d(c, t)$;
                  22. break;
               23. end
            24. end
         12. end
      17. if $found = false$ then
         19. $L$.insert($c$);
         20. Set $c$.freq $\leftarrow 1$;
         21. Set $c$.dist $\leftarrow d(c, t)$;
      18. end
   7. end
end
Phase 2

- In phase 2 we compute the scores of each $k$-combination of each product.
- To achieve this goal we use the forward index.
- We sort the forward list in decreasing score order.
- The first element of the sorted list is the cluster.

**Algorithm 2: Scores computation and cluster selection**

```plaintext
1 for each product $p$ in $F$ do
2     retrieve the forward list $f_p$;
3     for each $c \in f_p$ do
4         Set $c$.adist $\leftarrow c$.dist / $c$.freq;
5         Set $c$.score $\leftarrow$ ComputeScore($c$);
6     end
7     sort $f_p$ in decreasing score order;
8     Set cluster $\leftarrow f_p[0]$;
9 end
```
An indicative score function

• Score function

\[ S(c) = \frac{l(c)}{a + d(c, t)} \log N(c) \]

where \( l(c) \) is the length of the combination/label, \( N(c) \) is the frequency, and \( d(c, t) \) is the average distance of the combination from the beginning of the string.
Results

• We deployed a focused crawler on skroutz.gr and we collected 16208 products (mobile phones) classified in 922 clusters.

• Vendors: 320

• Average number of words in a title: 9

• We consider the classification of skroutz.gr as the ground truth and we compare the effectiveness of our algorithm (UMaP) against this.
Effectiveness – F1 measure

<table>
<thead>
<tr>
<th>$K$</th>
<th>$\alpha$</th>
<th>$F1$</th>
<th>Precision</th>
<th>Recall</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>$\alpha = 1$</td>
<td>0.32433</td>
<td>0.20470</td>
<td>0.78032</td>
</tr>
<tr>
<td></td>
<td>$\alpha = 2$</td>
<td>0.35216</td>
<td>0.22748</td>
<td>0.77929</td>
</tr>
<tr>
<td></td>
<td>$\alpha = 3$</td>
<td>0.33412</td>
<td>0.21313</td>
<td>0.77296</td>
</tr>
<tr>
<td></td>
<td>$\alpha = 4$</td>
<td>0.34597</td>
<td>0.22321</td>
<td>0.76880</td>
</tr>
<tr>
<td></td>
<td>$\alpha = 5$</td>
<td>0.33753</td>
<td>0.21637</td>
<td>0.76704</td>
</tr>
<tr>
<td>4</td>
<td>$\alpha = 1$</td>
<td>0.66370</td>
<td>0.64175</td>
<td>0.68721</td>
</tr>
<tr>
<td></td>
<td>$\alpha = 2$</td>
<td>0.62118</td>
<td>0.60239</td>
<td>0.64118</td>
</tr>
<tr>
<td></td>
<td>$\alpha = 3$</td>
<td>0.61290</td>
<td>0.57920</td>
<td>0.65076</td>
</tr>
<tr>
<td></td>
<td>$\alpha = 4$</td>
<td>0.60302</td>
<td>0.56046</td>
<td>0.65258</td>
</tr>
<tr>
<td></td>
<td>$\alpha = 5$</td>
<td>0.58569</td>
<td>0.53552</td>
<td>0.64624</td>
</tr>
<tr>
<td>5</td>
<td>$\alpha = 1$</td>
<td>0.48130</td>
<td>0.61997</td>
<td>0.39333</td>
</tr>
<tr>
<td></td>
<td>$\alpha = 2$</td>
<td>0.45771</td>
<td>0.59741</td>
<td>0.37096</td>
</tr>
<tr>
<td></td>
<td>$\alpha = 3$</td>
<td>0.43544</td>
<td>0.61239</td>
<td>0.33783</td>
</tr>
<tr>
<td></td>
<td>$\alpha = 4$</td>
<td>0.42029</td>
<td>0.57447</td>
<td>0.33136</td>
</tr>
<tr>
<td></td>
<td>$\alpha = 5$</td>
<td>0.40041</td>
<td>0.53044</td>
<td>0.32158</td>
</tr>
<tr>
<td>6</td>
<td>$\alpha = 1$</td>
<td>0.35216</td>
<td>0.71483</td>
<td>0.23363</td>
</tr>
<tr>
<td></td>
<td>$\alpha = 2$</td>
<td>0.31339</td>
<td>0.69577</td>
<td>0.20225</td>
</tr>
<tr>
<td></td>
<td>$\alpha = 3$</td>
<td>0.29679</td>
<td>0.66443</td>
<td>0.19107</td>
</tr>
<tr>
<td></td>
<td>$\alpha = 4$</td>
<td>0.29022</td>
<td>0.62577</td>
<td>0.18892</td>
</tr>
<tr>
<td></td>
<td>$\alpha = 5$</td>
<td>0.27862</td>
<td>0.62301</td>
<td>0.17944</td>
</tr>
</tbody>
</table>

**TABLE II**
UMaP performance for various values of $K$ and $\alpha$.

Fig. 2. Comparison of F1 scores for UMaP, cosine similarity, Jaccard similarity, and Jaro-Winkler distance.
### Efficiency

<table>
<thead>
<tr>
<th></th>
<th>Durations (sec)</th>
<th>Combinations</th>
<th>Permutations</th>
</tr>
</thead>
<tbody>
<tr>
<td>$K = 3$</td>
<td>1.55</td>
<td>2,896,310</td>
<td>3,292,384</td>
</tr>
<tr>
<td>$K = 4$</td>
<td>13.20</td>
<td>8,742,866</td>
<td>46,738,214</td>
</tr>
<tr>
<td>$K = 5$</td>
<td>292.32</td>
<td>21,733,514</td>
<td>1,177,518,713</td>
</tr>
<tr>
<td>$K = 6$</td>
<td>7177.36</td>
<td>46,482,486</td>
<td>$\approx 1.3 \cdot 10^{12}$</td>
</tr>
<tr>
<td>COSim</td>
<td>171.47</td>
<td>–</td>
<td>–</td>
</tr>
<tr>
<td>JSim</td>
<td>244.89</td>
<td>–</td>
<td>–</td>
</tr>
<tr>
<td>JRD</td>
<td>282.30</td>
<td>–</td>
<td>–</td>
</tr>
</tbody>
</table>

**TABLE III**  
Efficiency evaluation of UMaP against cosine similarity, Jaccard similarity, and Jaro-Winkler distance for various values of $K$. 