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)

- V. Gopalakrishnan, SP. Iyengar, A. Madaan, R. Rastogi, S. Sengamedu. Matching product titles using web-based enrichment. In Proceedings of the 21st ACM international conference on Information and knowledge management, pp. 605-614, 2012.
- N. Londhe, V. Gopalakrishnan, A. Zhang, HQ Ngo, R. Srihari. Matching titles with cross title web-search enrichment and community detection. In Proceedings of the VLDB Endowment, pp. 1167-1178, 2014.

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…

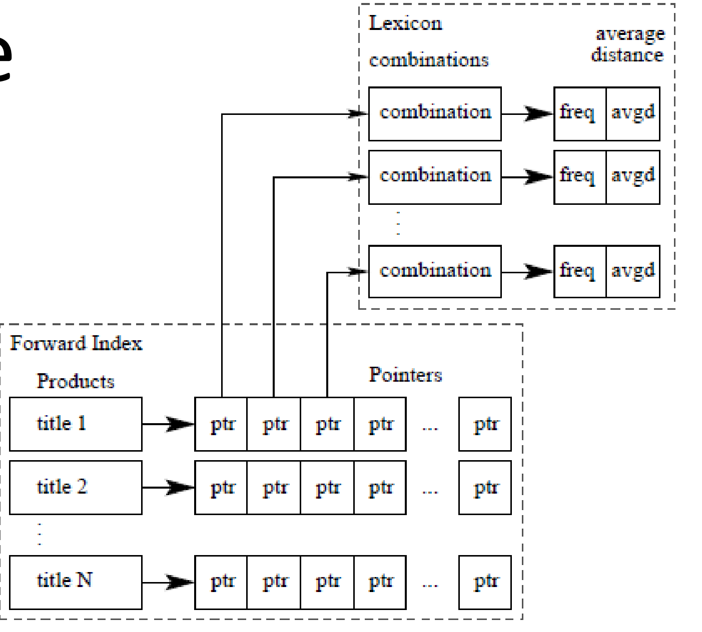
- Standalone: It does not rely on external data sources (i.e. Web search engines, Web sites, ect).
- Unsupervised: No requirement to manually train a classifier, or split the dataset in training and testing data subsets.
- Efficient: Faster than the adversary approach; it makes use of in-memory data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

Overview (2)

- 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)

for *each product* p **do**
- extract the title t;
- perform linguistic processing of t;
- **for** *each* $k \in [2, K]$ **do**
  - compute all k-combinations $C_{(k)}$ of t;

```
C:\Users\leo\Documents\cpp_algorithms\s
apple iphone 7 32gb jet black
1. 2-Combination: apple iphone
2. 2-Combination: apple 7
3. 2-Combination: apple 32gb
4. 2-Combination: apple jet
5. 2-Combination: apple black
6. 2-Combination: iphone 7
7. 2-Combination: iphone 32gb
8. 2-Combination: iphone jet
9. 2-Combination: iphone black
10. 2-Combination: 7 32gb
11. 2-Combination: 7 jet
12. 2-Combination: 7 black
13. 2-Combination: 32gb jet
14. 2-Combination: 32gb black
15. 2-Combination: jet black
1. 3-Combination: apple iphone 7
2. 3-Combination: apple iphone 32gb
3. 3-Combination: apple iphone jet
4. 3-Combination: apple iphone black
5. 3-Combination: apple 7 32gb
6. 3-Combination: apple 7 jet
7. 3-Combination: apple 7 black
8. 3-Combination: apple 32gb jet
9. 3-Combination: apple 32gb black
10. 3-Combination: apple jet black
11. 3-Combination: iphone 7 32gb
12. 3-Combination: iphone 7 jet
13. 3-Combination: iphone 7 black
14. 3-Combination: iphone 32gb jet
15. 3-Combination: iphone 32gb black
16. 3-Combination: iphone jet black
17. 3-Combination: 7 32gb jet
18. 3-Combination: 7 32gb black
19. 3-Combination: 7 jet black
20. 3-Combination: 32gb jet black
1. 4-Combination: apple iphone 7 32gb
2. 4-Combination: apple iphone 7 jet
3. 4-Combination: apple iphone 7 black
4. 4-Combination: apple iphone 32gb jet
5. 4-Combination: apple iphone 32gb black
6. 4-Combination: apple iphone jet black
7. 4-Combination: apple 7 32gb jet
8. 4-Combination: apple 7 32gb black
9. 4-Combination: apple 7 jet black
10. 4-Combination: apple 32gb jet black
11. 4-Combination: iphone 7 32gb jet
12. 4-Combination: iphone 7 32gb black
13. 4-Combination: iphone 7 jet black
14. 4-Combination: iphone 32gb jet black
15. 4-Combination: 7 32gb jet black
1. 5-Combination: apple iphone 7 32gb jet
2. 5-Combination: apple iphone 7 32gb black
3. 5-Combination: apple iphone 7 jet black
4. 5-Combination: apple iphone 32gb jet black
5. 5-Combination: apple 7 32gb jet black
6. 5-Combination: iphone 7 32gb jet black
```

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).**

Phase 1 (2)

Algorithm 1: Product titles' processing and data structures construction

initialize the lexicon L;
initialize the forward index F;
for *each product* p **do**
  extract the title t;
  perform linguistic processing of t;
  for *each* $k \in [2, K]$ **do**
    compute all k-combinations $C_{(k)}$ of t;
    for *each* k*-combination* $c \in C_{(k)}$ **do**
      Set $d(c, t) \leftarrow$ distance(c,t);
      F.insert(p,c);
      Set $found \leftarrow L$.search(c);
      if *found* = *true* **then**
        Set c.freq $\leftarrow$ c.freq + 1;
        Set c.dist $\leftarrow$ c.dist + $d(c, t)$;
      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.

Phase 1 (3)

Algorithm 1: Product titles' processing and data structures construction

```
initialize the lexicon L;
initialize the forward index F;
for each product p do
    extract the title t;
    perform linguistic processing of t;
    for each k ∈ [2, K] do
        compute all k-combinations C_(k) of t;
        for each k-combination c ∈ C_(k) do
            Set d(c, t) ← distance(c,t);
            F.insert(p,c);
            Set found ← L.search(c);
            if found = true then
                Set c.freq ← c.freq + 1;
                Set c.dist ← c.dist + d(c, t);
            else
                compute all permutations M of c;
                for each permutation m ∈ M do
                    Set found ← L.search(m);
                    if found = true then
                        Set c.freq ← c.freq + 1;
                        Set c.dist ← c.dist + d(c, t);
                        break;
                    end
                end
            end
            if found = false then
                L.insert(c);
                Set c.freq ← 1;
                Set c.dist ← d(c, t);
            end
        end
    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

for *each product* p *in* F **do**
    retrieve the forward list f_p;
    for *each* $c \in f_p$ **do**
        Set c.adist $\leftarrow$ c.dist / c.freq;
        Set c.score $\leftarrow$ ComputeScore(c);
    end
    sort f_p in decreasing score order;
    Set cluster $\leftarrow$ $f_p[0]$;
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

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

TABLE II
UMaP PERFORMANCE FOR VARIOUS VALUES OF K AND α.

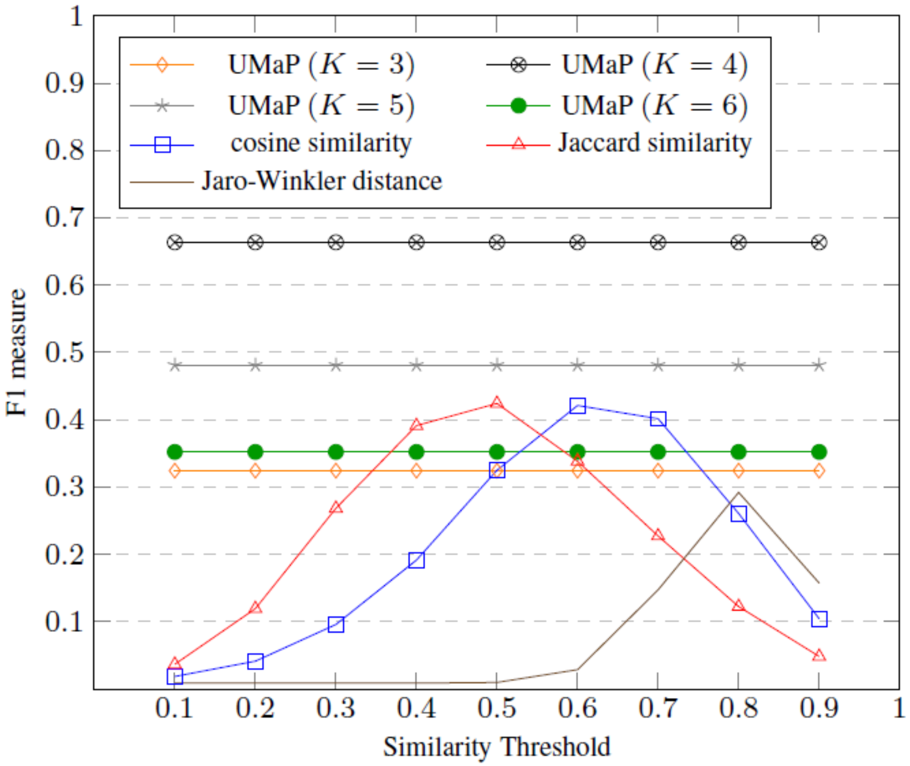


Fig. 2. Comparison of $F1$ scores for UMaP, cosine similarity, Jaccard similarity, and Jaro-Winkler distance

Efficiency

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

TABLE III
EFFICIENCY EVALUATION OF UMAP AGAINST COSINE SIMILARITY, JACCARD SIMILARITY, AND JARO-WINKLER DISTANCE FOR VARIOUS VALUES OF K.