AraProdMatch: A Machine Learning Approach for Product Matching in E-Commerce

Aisha Alabdullatif and Monira Aloud
aisha.alabdullatif@gmail.com mealoud@ksu.edu.sa
Department of Management Information Systems, College of Business Administration
King Saud University
Riyadh, KSA

Abstract
Recently, the growth of e-commerce in Saudi Arabia has been exponential, bringing new remarkable challenges. A naive approach for product matching and categorization is needed to help consumers choose the right store to purchase a product. This paper presents a machine learning approach for product matching that combines deep learning techniques with standard artificial neural networks (ANNs). Existing methods focused on product matching, whereas our model compares products based on unstructured descriptions. We evaluated our electronics dataset model from three business-to-consumer (B2C) online stores by putting the match products collectively in one dataset. The performance evaluation based on k-mean classifier prediction from three real-world online stores demonstrates that the proposed algorithm outperforms the benchmarked approach by 80% on average F1-measure.

Keywords: *product matching; artificial neural network; consumer decision-making; deep learning; e-commerce.*

1. Introduction

Recently, e-commerce in the Saudi market has grown significantly [1]. According to the Saudi Communication and Information Technology Commission (CITC), the rapid growth of online shopping has resulted in global business-to-consumer (B2C) e-commerce spending exceeding $3.8 trillion [1]. The e-commerce industry in Saudi Arabia reached $7.92b in 2016 and is expected to grow over the next years [1]. Furthermore, the number of internet users in Saudi Arabia is continuously growing, reaching 20.8 million users in 2017; 58% of them are online buyers [1].
The proliferation of the e-commerce market creates challenges associated with buyer decision-making from different online stores. Thus, consumers have the choice to purchase the product from different online stores, which made the purchasing decision of the products more complex. Moreover, product descriptions and labels vary across different stores, indicating a lack of standardization in their schema. For example, when a consumer should buy a specific product, they can navigate through different online stores that sell the same product to decide where to buy that product. This issue raised the need for a product-matching scheme to effortlessly look for a product and find the most reliable provider.

To improve consumer shopping experience, an approach for matching e-commerce products based on their description is required. If online stores adopted semantic markup language to interpret the products, the product-matching method can be efficient [2].

The problem of product matching is crucial for e-commerce platforms, their users, and the e-commerce industry in general. In the literature, the issue of e-commerce product matching has been studied systematically and draws a considerable number of researchers. Several prior studies used the standard string similarity method [3, 4]. More recent methods have focused on machine learning techniques such as fuzzy matching, which is based on extending standard token-based similarity functions [5, 6]. Another study by [7] used an unsupervised algorithm for matching products based on their titles.

This paper provides a machine learning method for matching e-commerce products in one platform. The aim is to enhance product lookup and comparison.
Given that we are working with unstructured data, we use unsupervised deep learning ANN techniques to locate matched products. The method starts by detecting and grouping similar products allocated on different online stores based on their different textual descriptions and other available product-specific properties.
The remainder of this paper is organized as follows: Section 2 covers the related works in the field. Section 3 presents the product-matching approach. Section 4 presents the results. We conclude with a discussion and directions for future work in section 5.

Manuscript received April 5, 2021
Manuscript revised April 20, 2021
https://doi.org/10.22937/IJCSNS.2021.21.4.26

2. RELATED WORK

Several product-matching techniques find similar products that different brands can produce, whereas others find identical products of the same brand from various online stores. Although finding duplicate products among different online stores is not yet a mature framework, many proposed algorithms in the literature aim to find duplicate texts. Several algorithms for text duplicate detection can be found in the literature for databases and information networks.

Bakker *et al*. [8] present two methods for detecting duplicated products on the web, specifically for pair-wise matching. The core step in the two methods is to determine whether the two products under consideration are duplicates or not based on the product descriptions. The first approach uses a model-words algorithm to find similar product names. If the product matches, the distance measure is used to determine the similarity of their attributes. The second method is the extended model-words, where the model-words algorithm is used to measure the similarity for product names and their attributes simultaneously. The performance measures used for evaluation are precision, recall, and F1-measure with an average of 63.7%, 59.7%, and a value of 0.607, respectively. The *p*-value was used to compute the significance level when comparing the extended method with each algorithm separately (once with model-words algorithm and once with attribute distance algorithm). Although the extended model-words approach performed satisfactorily, it can be improved by applying optimization and features extraction procedures to the descriptions before the process of finding the duplicate products.

A method for duplicate detection proposed by [9] used hierarchical clustering based on K-Means (top-down divisive approach and bottom-up agglomerative approach) to start classifying the products [9]. Following that, the multi-component similarity is used to find similar products by calculating the q-gram string similarity measure and determining whether the similarity was found within the same store or on different online stores. The method by [9] depends on the extracted attributes from product descriptions for its method. The contribution of this approach is noticed when using the *diffBrand(i,j)* python function. The role of this function is to compare the product's brand name. The evaluation datasets were obtained from four online stores with an average of 29 key-value pairs. The evaluation results were 0.475 F1-measure values, the average precision value of 44.5%, and the average recall of 51.2%.

Another matching method proposed in [10] is based on unstructured product descriptions. The proposed matching offers method aims to match identical product offers from various online sources. The process starts by extracting product features from the textual offers. The similarity is then calculated using distance measures for the extracted features suitable to match pair-wise of the product names. The distance measures used are Levenshtein, the python-based class called *difflib.SequenceMatcher()*, Sorensen, and Jaccard distances. Following that, the suitable similarity is chosen based on the mean of all distance measures for each product. The next step is to measure the following similarity values:

- Similarity of Strings.
- Intersection of Words.
- Similarity of Number Words.
- Maximum N-Gram Similarity.
- Average N-Gram Similarity.

Defining the optimal similarity value combination depends on the online store scope. The datasets used in the developing phase come from three categories, while datasets of two categories were used in the evaluation phase. The evaluation proves the matching other method's competence with an average precision of 78.5% and average recall of 91%.

Ristoski *et al*. [2] used ANNs for matching and classifying products. The attributes used in their work are the product's short name and description. The product's features are extracted using the following algorithms:

a. *Dictionary-Based*: a dictionary of the product attributes and values presented in the structured product descriptions.
b. *Conditional Random Field (CRF):* the CRF model with a set of discrete features.
c. *CRF with Text Embeddings*: an improved CRF model integrated with text embedding features and is used to handle the different forms of a word, like the synonyms, found in product descriptions.
d. *Image Feature Extraction Model*: In addition to the textual features, the authors in [2] built an image embedding model using convolutional neural networks (CNNs).

The performance of matching products for the above four models was measured using random forest, support vector machines (SVM), Naïve Bayes, and logistic regression

models. An evaluation of electronics datasets showed promising results. However, the approach proposed by [2] can integrate unstructured product descriptions only if the product name appears in the text.

The basic word2vec word embedding algorithm is used to map each word in the text into a multi-dimensional sequence of numbers. In a paper presented by Biswas *et al*. [11], word embedding has been shown comprehensively to accommodate the context of online product titles. This paper presented the MRNet-Product2Vec approach to achieve extreme performance for product embeddings. Biswas *et al.* suggested developing a technique for learning product embeddings directly from a training set rather than using pre-trained word embeddings. The proposed embeddings were evaluated qualitatively and quantitatively, and the results were effective. The authors of this paper also proposed a multi-mode encoder for comparing products from different countries. Thus, the MRNet-Product2Vec provides initial results using the various factors resulting from different sources that offer products in different languages. The evaluation of this approach provides the flexibility to learn product embeddings with any other list of features or perfectly tune the pre-learned embeddings with additional features.

We focus on unstructured textual descriptions of the products offered by online stores in this paper. We demonstrate the generality of using the reported methods in this section, despite the language of the product description. We use deep learning algorithms to extract product features from product descriptions using text vectors. Following that, we used clustering algorithms on the features vectors to find the matching product in relevant clusters -categories- rather than matching the product to the entire datasets. This paper is based on Arabic language detection, unlike other works, which are based on the English language [2] [8] [10] [11]. We show the potential of applying the above-reported approaches [2] [8] [10] [11] on product descriptions written in Arabic.

3. ARAPRODMATCH: PRODUCT MATCHING APROACH

This section presents our framework that addresses the product-matching issues in e-commerce that were described in the previous section. The framework is called *Arabic E-Commerce Products Matching* (AraProdMatch) and consists of three phases:

- Data collection phase.
- Feature extraction phase.
- Product matching based on related clusters.

The AraProdMatch framework is depicted in Figure 1. The workflow runs in two phases: *training* and *testing*. The training phase launches through preprocessing the unstructured product descriptions. In the application phase, the model generates a set *N* of all feasible candidate-matching product pairs, which guides a more extensive set of candidates.

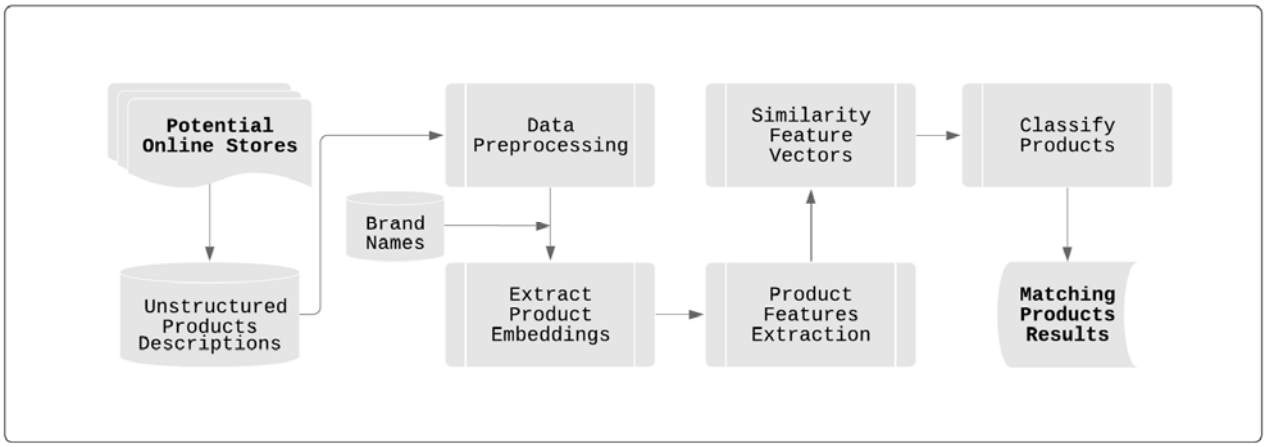

Fig. 1. AraProdMatch Framework Architecture.

A. Data Collection

a) Scrapy

Scrapy is an open-source and joint framework for crawling websites and extracting data. It is a data extraction framework that is fast, simple, and extendable. It can be used for various useful applications, such as data mining, information processing, and historical archival.

Scrapy is written in python, and it has a robust architecture, as illustrated in Figure 2. The architecture of the framework consists of seven components [12]:

- *Scrapy Engine*: it is responsible for controlling the data flow between all its components.
- *Scheduler*: it receives requests from the engine and places them in queues to feed the engine when required.
- *Downloader*: it is responsible for capturing web pages and feeding them to the engine, which in turn feeds them to the spiders.
- *Spiders*: they are custom classes coded by the developers to parse websites' responses and extract data from them or get additional URLs to parse.
- *Item Pipeline*: it is responsible for processing items. In our case, items are the description, price, image URL, and direct URL for each product.
- *Downloader*: it is a middleware that sits between the engine and downloader to process requests and responses when they pass between them.

The Scrapy components provide a convenient mechanism to extend Scrapy functionality by plugging in customized codes. For the spiders used in the AraProdMatch framework, we update the settings of the original spiders as follows:

- FEED_EXPORT_ENCODING = 'utf-8', this variable allows us to extract correctly encoded Arabic texts.
- DOWNLOAD_DELAY = 3. This specifies the time interval between each crawling step in seconds.

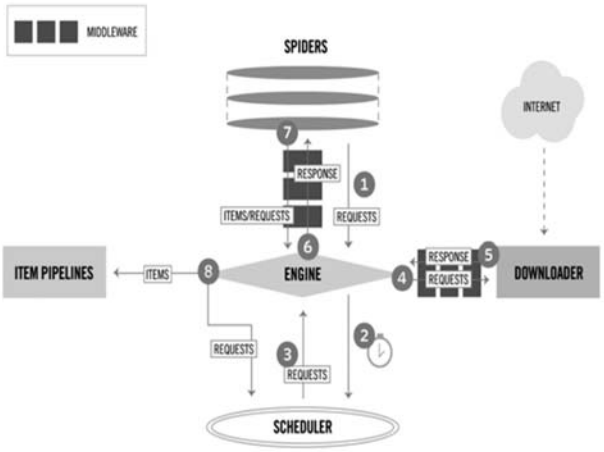

Fig. 2. Scrapy Architecture [7].

b) Dataset

We choose three Saudi online stores to collect product data. Since each online store has its HTML structure for listing products, a spider for each store has been developed. Spiders, also known as web crawlers or spider bots, are internet bots that automatically visit websites and collect data based on their HTML structure [13]. Spiders are mainly used for web indexing, but they are also used for automatically collecting massive data from the web.

Before developing the spiders, we manually navigate through each online store to learn and collect data from the webpage HTML structure. There are numerous ways to access any HTML page content; one of them is using x-path selectors. The x-path selector is a language used to select elements of any XML or HTML files [14]. Learning the HTML structure enables us to move to the next step, where we use the same code of the spider with minor changes to the four following variables: product description, original price, image URL, and product URL. These four variables hold the x-path to the four desired elements in the targeted online store.

One of the most powerful web crawler framework libraries is *Scrapy*. We used Scrapy to develop our spiders for each store.

In our framework, we set four main constraints on the variables that we collect (i) Product should contain textual description. (ii) The description length should be more than three words. (iii) The descriptions should be written in Arabic. (iv) Each product should show the original price in

Manuscript received April 5, 2021
Manuscript revised April 20, 2021
https://doi.org/10.22937/IJCSNS.2021.21.4.26

case the product is discounted. The pseudocode for data extraction spiders is shown in Algorithm 1.

Algorithm 1: Data extraction spider

Require: Scrapy library
Require: start_urls: sets a list of all targeted urls
Require: custom_settings: sets the location of a.CSV file to save the scraped results
Require: total_pages: to easily crawl over paginations
Require: parse: takes the current url to parse

```
titles = list of product titles in the current page from given xPath
prices = list of product original prices in the current page from given xPath
img_urls = list of product images in the current page from given xPath
prod_url = list of product direct URL to the products in the current page from given xPath
For all items in (titles, prices, img_urls, prod_url) Do:
        scraped_info = {
            'title': item[0],
            'price': item[1],
            'image_url': [item[2]],
            'prod_url': item[3]
        }
End For
add scraped_info to CSV file
If visited_pages < total_pages Then:
        next_page = visited_pages
        visited_pages = visited_pages + 1
        new page request (
                url = next_page URL,
                callback = self.parse )
```

Following the filtering process, the dataset contains ~10,000 product data from three online stores with different sizes. The store is considered big or small based on the count of its products. If the product count is less than 2000, it is considered small; otherwise, the online store is considered big.

The dataset is simple and consists of just electronic products: computers, TVs, smartphones, wearables, refrigerators, and washing machines. We divide the dataset into two parts: training and the remaining data for the product-matching approach.

c) Data Preparation

Preprocessing data starts by removing punctuation, specific article characters, stop words, checking, and converting letters that have different representations. We followed the standard steps to preparing Arabic content for deep learning, except removing Latin letters since we are considering the product model number included in descriptions. For example, the letter "*Alef*" can be represented as follows:
آ or ا، أ، إ
Table 1 shows an example of a product description as an output from the data preparation phase.

TABLE I: A DEMONSTRATION OF A PRODUCT DESCRIPTION PREPARATION PROCESS

Step	Example	Illustration of descriptions in English
Original text	سماعة واي فاي، وبلوتوث من جيه بي إل جو–اللون رمادي	Wi-fi headset, and Bluetooth from JBL the Color is Gray
Remove punctuations	سماعة واي فاي وبلوتوث من جيه بي إل جو اللون رمادي	Wifi headset and Bluetooth from JBL the Color is Gray
Remove definite article	سماعة واي فاي وبلوتوث من جيه بي إل جو لون رمادي	Wifi headset and Bluetooth from JBL Color is Gray
Remove stop words	سماعة واي فاي وبلوتوث جيه بي إل جو لون رمادي	Wifi headset Bluetooth JBL Color Gray
Check and convert letter Alef	سماعة واي فاي وبلوتوث جيه بي ال جو لون رمادي	Handle different representation of the Arabic alphabet Alef
Check and convert letter Ha and closed Ta’a	سماعه واي فاي وبلوتوث جيه بي ال جو لون رمادي	Handle different representation of the Arabic alphabet Ta'a

B. Feature Extraction

We use several extraction approaches to extract features from product descriptions. These methods are

frequently used algorithms for extracting features designed for unstructured descriptions [2] [8] [9] [10] [15].

a) Dictionary-Based Approach

To increase the speed of our framework, we prepared a dictionary of available electronics brands in Saudi Arabia. To generate a dictionary of brands; we extracted the list of brand names used in the online stores. Since many brands manufacture the product type, we set the product as a key identifier and the brand name as values. The dictionary of brands is used to cluster the dataset. Furthermore, we matched the brands with all possible *n*-grams produced from product descriptions in each online store product list.

b) Word Embeddings

We undertake to address the challenge that the product descriptions in Arabic from online stores contain combinations of numbers, Arabic and Latin alphabets. Despite the characters' complexity in the Arabic product description, we did not exclude the manufacturer model number since it is considered a unique product feature.
In particular, we used ANNs to extract product embeddings from unstructured product descriptions. We followed the proposed product to vector approach (MRNet-Product2Vec) in [11]. We roughly calculate the likelihood of a specific sequence product to appear in the product lists. We assume that the nearest product neighbors in the product vectors are statistically dependent where different product-related indications are explicitly inserted into their embeddings.
Afterward, we followed the rest of the approach from [2] to implement the CRF with product embeddings. The only difference is that we replaced the simple word2vec approach used in [2] with MRNet-Product2Vec to train the CRF model. We also compare the results of using basic word2vec trained using Twitter corpus collected by the authors of [16] against using MRNet-Product2Vec trained using massive descriptions of online products collected roughly from ten Arabic online stores.

c) Calculating Similarity of Feature Vectors

Our method measures the probability of finding matched products by measuring the similarity distance among the collected online stores' products. However, the feature extracted in the previous subsection, we defined product features as $P = \{p_1, p_2, p_3, \dots p_n\}$. *P* here represents all the possible features related to and extracted from a product description. Then, we calculated the similarity between the two products' vectors by measuring the distance between each feature in *product1* from every feature in *product2*.
The used similarity measures are two of the most widely used distance measures: Jaccard and Cosine similarity measures. The Jaccard similarity is computed on character *n*-grams where $n <= 4$, and the Cosine similarity is computed on the words of product descriptions tokens as shown in Equation (1) [2].

$$f(p1, p2) = \begin{cases} \text{JaccardSimilarity}(p1.\text{res(f)};\ p2.\text{res(f)}); & \text{if } f \text{ is string} \\ \text{CosineSimilarity}(p1.\text{res(f)};\ p2.\text{res(f)}); & \text{if } f \text{ is long string} \end{cases} \quad (1)$$

The similarity here is computed between two products at a time, as explained in Algorithm 2. The AraProdMatch framework is not restricted to the comparison between two stores only. Still, it provides a way to compute products' similarity between the unlimited numbers of online stores that the AraProdMatch approach has to check.

Algorithm 2: Computing similarity between products

Require: scikit-learn library

Require: stores data directory

```
starting_store = path to store 1
For all store in stores directory Do:
  next_store = path to the next occurrent store in the directory
  For all products in (starting_store, next_store) Do:
  compute the similarity between every product in
 the starting store to every product in next_store
      If similarity => threshold Then
        matched
      else
        not_matched
  End For
End For
```

d) Classification Approaches

Similarity feature vectors are being generated when the similarity process is complete. Next, is the classification step, where we train two classifiers: K-Means and logistic regression.

4. APROACH EVALUATION

We used the cross-validation method to train and test our models at a rate of 70% of our collected dataset for the training phase and 30% for testing.

A. Experiment Setup

To set up the experiment, we relied on Anaconda, a rich data science platform with an extensive number of python libraries and are widely used by data scientists. In Anaconda, we used Jupyter Notebook[1], an open-source and intuitive application to read code documentation and data visualization easily. Further, the python version in use is python 3.6. Moreover, the python libraries used to achieve our goal are Gensim[2] and Scikit-Learn[3].

On the dataset, we set two constraints: the name of the online store and the product descriptions to avoid getting similarities from the same online store since we are assuming that one online store will not list a product twice. Matching products are labeled as matched if they contain a defined quantity of information identified as unique features.

The measures for evaluating the AraProdMatch approach are the three well-known measures: precision, recall, and f-measure.

Precision in data mining, as shown in equation (2), is the number of items predicted correctly among all predictions [17].

$$Precesion = \frac{mpPos}{mpPos+npPos} \quad (2)$$

MpPos denote the count of products that are matched correctly, and npPos denotes the count of matched products incorrectly. Hence, the optimum way of improving precision is to decrease the poorly matched products. This explains the reason behind the progress toward ontology mapping, adopting conventional and strict approaches.

The second measure is *recall* – Equation 3-, which is in data mining, and it recalls the number of actual items captured correctly among all items [17].

$$Recall = \frac{mpPos}{mpPos+mpNeg} \quad (3)$$

In Equation (3), *mpPos* is the count of products that are matched correctly, and *npNeg* is the count of matching products that are not captured as matched by our approach.

Robust matching approaches attempt to increase precision as much as possible, regardless of the low recall [18]. This choice is undesirable in this study since we need a highly accurate and trustworthy matching method.

The third and last measure is the *F-measure*. The F-measure is described as a harmonic mean of precision and recall [19]. To compare the algorithm, the same pair of measures should be used for all of them [20]. When viewed as weighted arithmetic mean in the F-measure and the ratio of weights is shown in Equation (4).

$$FMeasure = 2 \times \frac{Precision \times Recall}{Precision + Recall} \quad (4)$$

B. Results

In our method, we compared the use of basic word2vec with product embedding. The results of the similarity measure evaluation are shown in Table 2. The similarity results show that the product's embedding outperforms the basic word embeddings word2vec approach by 13% on average F1-measure.

Furthermore, we compare the performance of the two classifiers: K-Means and logistic regression. Table 3 shows the results of the classifier's performance comparison. As we can see from Table 3, the K-Means classifier delivers the best result for all three performance measures over logistic regression. K-Means classifier exceeds the accuracy of the logistic regression by 22.4% on average F-measure.

TABLE II: SIMILARITY MEASURES EVALUATION RESULTS

	Precision	Recall	f-measure
Word2vec	0.229	0.219	0.224
Product2Vec	0.329	0.384	0.354

TABLE III: ARAPRODMATCH PERFORMANCE

	Precision	Recall	f-measure
K-Means	0.819	0.770	0.794
Logistic Regression	0.490	0.680	0.570

[1] http://jupyter.org
[2] https://radimrehurek.com/gensim/about.html
[3] https://scikit-learn.org

5. CONCLUSION AND FUTURE WORKS

This study builds a product-matching approach for e-commerce, to provide a method for matching products from various online stores. We proposed an AraProdMatch method that uses a combination of the best algorithms used in the literature, starting from the MRNet-Product2Vec to extract features from product descriptions to Jaccard and Cosine's use similarity measures to match products vectors. This study demonstrates that the AraProdMatch method has good quality performance on text extraction designed for Arabic. Out of the two matching classifiers, K-Means and logistic regression, we conclude that using K-Means outperforms logistic regression. Moreover, we adopt the ANN deep learning method using the word embeddings algorithm. We looked at the difference between using the basic word embedding method word2vec versus the advanced product embeddings method MRNet-Product2Vec, and we demonstrated that MRNet-Product2Vec outperforms the basic word2vec algorithm.
The first step in the AraProdMatch method is to collect dynamically collect data from online stores. One limitation in this phase is that we should feed the spiders with the x-path of online store elements. The step is expected to be improved in future work by conducting a standalone study on detecting the product descriptions x-path without human intervention.

In future works, we add a phase for matching products by using the algorithms to build a robust image classifier. The aim is to support the matching process of the product descriptions. Furthermore, in the future, we test the model on a large number of product descriptions from a wider range of online stores and other categories rather than just the electronic dataset.

ACKNOWLEDGMENT

The authors thank the Deanship of Scientific Research and RSSU at King Saud University for their technical support .

REFERENCES

[1] CITC, "ICT Report - ECommerce in Saudi Arabia," 2017.

[2] P. Ristoski, P. Petrovski, P. Mika, and H. Paulheim, "A Machine Learning Approach for Product Matching and Categorization Use case: Enriching Product Ads with Semantic Structured Data," *Semant. Web–Interoperability, Usability, Appl. an IOS Press*, vol. 0, 2016.

[3] L. Akritidis, A. Fevgas, P. Bozanis, and C. Makris. A self-verifying clustering approach to unsupervised matching of product titles. Artificial Intelligence Review. 2020;53(7): pp. 4777–4820. doi:10.1007/s10462-020-09807-8

[4] A. K. Elmagarmid, P. G. Ipeirotis, and V. S. Verykios Knowl Data Eng 19(1): pp. 1–16

[5] W. H. Gomaaand A. A. Fahmy (2013) A survey of text similarity approaches. Int J Comput Appl 68(13):13–18

[6] S. Chaudhuri, K. Ganjam, V. Ganti, and R. Motwani (2003) Robust and efcient fuzzy match for online data cleaning. In: Proceedings of the 2003 ACM international conference on management of data (SIGMOD), pp 313–324

[7] J. Wang, G. Li, and J. Fe (2011) Fast-join: An efcient method for fuzzy token matching based string similarity join. In: Proceedings of the 27th IEEE international conference on data engineering (ICDE), pp 458–469

[8] M. de Bakker, D. Vandic, F. Frasincar, and U. Kaymak, "Model words-driven approaches for duplicate detection on the web," *Proc. 28th Annu. ACM Symp. Appl. Comput. - SAC '13*, p. 717, 2013.

[9] R. Van Bezu, J. Verhagen, and R. Rijkse, "Multi-component Similarity Method for Web Product Duplicate Detection," pp. 761–768, 2015.

[10] A. Horch, H. Kett, and A. Weisbecker, "Matching Product Offers of E-Shops," Springer, Cham, 2016, pp. 248–259.

[11] A. Biswas, M. Bhutani, and S. Sanyal, "MRNet-Product2Vec: A Multi-task Recurrent Neural Network for Product Embeddings," *Lect. Notes Comput. Sci. (including Subser. Lect. Notes Artif. Intell. Lect. Notes Bioinformatics)*, vol. 10536 LNAI, pp. 153–165, 2017.

[12] Scrapy developers, "Architecture overview— Scrapy 1.5.1 documentation," *2017-01-10*. [Online]. Available: https://doc.scrapy.org/en/latest/topics/architecture.html. [Accessed: 31-March-2021].

[13] J. Wang and Y. Guo, "Scrapy-Based Crawling and User-Behavior Characteristics Analysis on Taobao," in *2012 International Conference on Cyber-Enabled Distributed Computing and Knowledge Discovery*, 2012, pp. 44–52.

[14] w3schools, "XML and XPath." [Online]. Available:

https://www.w3schools.com/xml/xml_xpath.asp. [Accessed: 30-March-2021].

[15] H. Lee and Y. Yoon, "Engineering doc2vec for automatic classification of product descriptions on O2O applications," *Electron. Commer. Res.*, vol. 18, no. 3, pp. 433–456, 2018.

[16] A. B. Soliman, K. Eissa, and S. R. El-Beltagy, "AraVec: A set of Arabic Word Embedding Models for use in Arabic NLP," *Procedia Comput. Sci.*, vol. 117, pp. 256–265, 2017.

[17] J. Han and M. Kamber, *Data mining : concepts and techniques*. Elsevier, 2006.

[18] S. Park and W. Kim, "Ontology Mapping Between Heterogeneous Product Taxonomies in an Electronic Commerce Environment," *Int. J. Electron. Commer.*, vol. 12, no. 2, pp. 69–87, 2007.

[19] Y. S.-T. T. mater and undefined 2007, "The truth of the F-measure," *cs.odu.edu*.

[20] D. Hand and P. Christen, "A note on using the F-measure for evaluating record linkage algorithms," *Stat. Comput.*, vol. 28, no. 3, pp. 539–547, May 2018.