Product Matching Lessons and Recommendations from a Real World Application

Jeremy Foxcroft[†,*], Tianle Chen[‡], Kanchana Padmanabhan [‡], Brian Keng [‡], Luiza Antonie [†]
[†] School of Computer Science, University of Guelph
[‡] Kinaxis

Abstract

Retailers rely heavily on product matching to better serve their customers, and to improve their modelling and forecasting. Product matching refers to the process of identifying similar or identical products across different data sources. This is a challenging problem as standardized unique identifiers are not used consistently across retailers, and product descriptions and characteristics vary across data collections. In this paper we present and discuss lessons learned from product matching in a real world application. We propose an evaluation framework where we investigate and compare the use of traditional machine learning methods and deep learning methods on public and proprietary datasets for product matching. Our findings show that traditional machine learning methods perform well on this task and a practitioner should investigate these methods first.

Keywords: record linkage, product matching, machine learning, deep learning

1. Introduction

The number of products sold by modern retailers has experienced tremendous growth, with marketplaces such as Amazon carrying millions of distinct products. With this ever increasing number of products, the methods to search and compare products have by necessity become increasingly sophisticated. Product matching, the process by which differing product records are identified as referring to the same real-world product, is at the heart of many of these tools. Product matching has enabled consumers to easily browse offers from different merchants for the same product through a single interface. It has also been used to create high-quality product catalogues, usable by merchants to easily provide detailed product listings. There has been a lot of work done to explore different machine learning and deep learning model architectures used for product matching [1, 2], different methods of creating product embeddings [3], crawling and parsing web pages to retrieve product listings [4], and parsing unstructured records into standardized product fields [5].

In this paper we use proprietary data from a pharmaceutical retail chain, and a health and beauty retail chain. The goal is to integrate these two datasets through product matching. One potential application of this work is to improve demand forecasting, especially for products that are new to market. Time series models that predict future demand for a product often use historical sales data as input. For certain products this data may be unavailable for a number of reasons: perhaps a retailer didn't keep sufficiently detailed sales records, perhaps a retailer has only just started carrying the product, or perhaps the product itself is new to market. In these cases, one approach to predicting demand using an existing model is to substitute in the sales data of a similar product. Product matching can be used to surface such products.

The main contributions of the paper are as follows:

- We formulate a hypothesis about the expected performance of the product matching systems based on data profiling and we investigate if that holds through our experiments. This allows us to make recommendations for other companies that are interested in using these methods for their product matching needs.

[*] jfoxcrof@uoguelph.ca

- We propose an evaluation framework for comparing product matching techniques. The novelty of the framework is that it consistently compares all methods over a stratified 10-fold cross validation set and it provides precision/recall curves, which allow performance evaluation over the entire range of precision/recall values.
- We investigate and compare the use of traditional machine learning methods and state of the art deep learning methods for product matching.

2. Background and Related Work

Record linkage is the task of finding records from two or more sources that correspond to the same entity. The problem of record linkage has been studied in the statistics community since the 60's [6]. Advances in databases, machine learning, and data mining communities have led to new and sophisticated solutions to this problem. Winkler [7], Elmagarmid et al.[8] and Christen [9] provide a good review of the field. Many other names are used to refer to this process, including data matching[9], entity resolution [10], and deduplication [11]. A record linkage pipeline can be broken down into a number of discrete phases, as shown in Figure 1.

The goal is to link records from one data collection $\mathcal{A}$ to another, $\mathcal{B}$. A record a in $\mathcal{A}$ (viz. b in $\mathcal{B}$) consists of all the information available for a particular entity; in our case the entity is a product, thus the information will be product related, e.g., brand and product name. Through the record linkage process we aim to find all pairs (a, b), $a \in \mathcal{A}$, $b \in \mathcal{B}$ such that a matches b (we say that a and b refer to the same product). In this case we write $a \simeq b$.

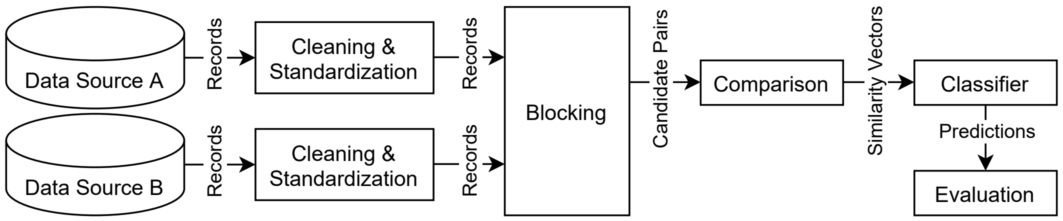


Figure 1. The phases in the record linkage pipeline.

Record linkage usually begins with two datasets that are believed to overlap to some degree in the entities to which they refer. For example, two different retailers in similar domains likely carry some of the same products. Ideally all product records would have a shared unique identifier, and two records would refer to the same product if and only if this identifier matches. In practice however, data is often characterized by errors, missing values, differing schemas, and more confounding factors that make a simple database join operation insufficient or impossible. To address this problem, record linkage employs a range of techniques to measure similarity between pairs of records, which can be used to train a classifier to distinguish pairs of matching records from pairs of non-matching records.

The first step in the record linkage process is data cleaning and standardization. This commonly involves discarding information from one dataset that is not present in the other, stripping extraneous characters (e.g., dollar signs) from numeric fields, and converting text fields to a uniform casing.

To determine if two records are similar or not (they refer to the same entity or not), one must calculate a similarity vector $\phi_{(a,b)}$ over all the common attributes. Computing the similarity vectors for all pairs of records in the Cartesian product of the two datasets is often prohibitively computationally expensive. To address this, a blocking phase [12] is used to eliminate as many candidate pairs (two records being considered as a potential match) as possible. Once these similarity vectors are computed, a classification system is employed to determine if the pair should be classified as a match or as a non-match. The training and evaluation of the classification model requires access to some quantity of labeled data. The characteristics of known matches are learned by the classifier during training, and some known matches are withheld until evaluation to measure the classifiers performance.

Product matching has been heavily influenced by trends in machine learning. Traditional machine learning techniques have seen usage in both academia and industry [1, 5, 13]. There have been several notable record linkage frameworks over the years, including Febrl [14] and Magellan [15]. Most linkage problems involve records with text attributes, so record linkage techniques have prospered from advances in natural language processing. This has included TF-IDF based classification systems [16, 17], different word and record embedding techniques [3], and a recent acceleration in the use of deep learning [2, 17–20]. Scraping and parsing structured records from web product listings which can be used in linkage tasks has also been a recent area of focus [4, 16].

3. Data

This section describes the data used in this paper. We first introduce publicly available data that is commonly used to evaluate product matching methods and then we describe our proprietary data. We detail how we generate candidate pairs in Section 3.2.

3.1. Datasets Description

There are a number of open source e-commerce product datasets that are commonly used to benchmark record linkage approaches. Our paper makes use of the Abt/Buy and Amazon/Google datasets[1], and the Amazon/Walmart datasets[2]. These datasets are similar to the proprietary data we wish to explore, as they are not only in the product matching domain, but are also all explicitly comparing the catalogues of two different retailers. While open source datasets are good for comparing different approaches, the focus in this work is to see how the performance of different product matching techniques translates when applied to our proprietary data and how a practitioner can translate knowledge learned on open source datasets to their particular application. To this end, we leverage two product datasets from major retailers. RetailerA is a pharmaceutical retail chain, and RetailerB is a health and beauty retail chain.

The product schemas from each of these two retailers contain a limited number of overlapping attributes. RetailerA's products are uniquely identified by a Global Trade Item Number (GTIN), and RetailerB's products are uniquely identified with a Universal Product Code (UPC). While carrying different names, these fields can act as a shared unique identifier (i.e., when a product carried by RetailerA has a UPC that perfectly matches the GTIN of a product carried by RetailerB, these two products are unambiguously identical). However, only a very small percentage of records can be matched using UPC-GTIN correspondence in our datasets. Thus the need for product matching techniques to integrate these two datasets. We use the UPC-GTIN correspondence to label 1,324 products as true matches, as has been done before in [13]. This approach to identifying true matches is reliable, and does not require large amounts of manual human labor.

The attributes common to both schemas include product name, brand, and 3 levels of categorization for each product. All of these attributes are plain text, with some examples shown in Table 1. Both retailers followed a structured 3-tier hierarchy to categorize products, where the three category attributes define each product's place in a hierarchy with increasing levels of specificity. Unfortunately the hierarchies are different and can not be mapped to

[1]https://dbs.uni-leipzig.de/research/projects/object_matching/benchmark_datasets_for_entity_resolution
[2]https://sites.google.com/site/anhaidgroup/useful-stuff/data

reduce the matching space within categories. We concatenate the three category fields in each dataset into a single textual category attribute prior to matching.

Table 1. Examples of products from Retailers A/B.

UPC/GTIN	Product Name	Brand	Category 1	Category 2	Category 3
4001638098595	weleda skin food 75ml	weleda	skincare	skin - facial	facial moisturisers
85805558420	red door 100ml edt	elizabeth arden	cosmetics	fragrances	fragrance ladies
3600521650042	l/perf conc t/mat deep be	l'oreal	cosmetics	mass color cosmetics	mass face make up

While matching UPCs/GTINs unambiguously communicate a match, differing UPCs/ GTINs are not necessarily sufficient to say two products should be labeled as non-matching. A product might have its UPC/GTIN updated at any time because of some minor ingredient change, a change in the manufacturing process, because it is packaged differently (e.g., Christmas or Easter packaging), or because it is sold in a slightly different quantity. This last option in particular is significant in that stores will sometimes strike agreements with manufacturers to buy a uniquely sized version of their product. This has the effect of preventing consumer price matching; if for example a certain grocery chain is alone in carrying 3.6L of a certain brand of laundry detergent, then their competitor having a 50% off sale on 4L of the same detergent can't be used by consumers under the terms of many price matching policies. As such, we can't just stop at the results of the UPC/GTIN matching and confidently label all other product pairs as non-matches.

Table 2. An overview of the datasets used in this paper, their sizes, number of labeled matches, and overlapping schema attributes used for matching. We used code from three prior publications, all of which also used at least two of these matching tasks (cited in the first column).

	#Records			Overlapping Schema attributes		
Matching Task [Used In]	L	R	#Matches	Short Str.	Long Str.	Numeric
Abt/Buy [2, 19, 21]	1,081	1,092	1,095		name description	price
Amazon/Google [2, 21]	1,363	1,298	1,298	manufacturer	name description	price
Amazon/Walmart [2, 19, 21]	22,074	2,554	1,154	brand modelno	title longdescr	price
RetailerA/RetailerB	188,864	187,970	1,324	brand	name category	

Table 2 outlines the size, number of known matches, and matching schema attributes for each of the datasets we use in this paper. Text attributes are classified as short ("Short Str." in Table 2) if the attribute values contain six or fewer words on average.

3.2. **Training Data Generation and Blocking**

The problem of product matching is a classification task, where pairs of products are classified by a model as a match or a non-match. In order to learn or train a classification system, one needs pairs of products that are labelled as matches and non-matches. It is these matching and non-matching pairs together which form the **correspondence set** we use to train and evaluate our classification systems. We follow the same approach to generating the correspondence sets as used in [21], shown in Algorithm 1. Non-matching pairs are drawn from the Cartesian product of the two datasets being matched, excluding the pairs already labeled as known matches. To include all such pairs in the correspondence set would not only introduce a massive class imbalance, but would also prohibitively slow down the feature vector creation and training of the matching models. The goal is to include enough

Algorithm 1 Correspondence Set Generation

```
procedure GET_CORRESPONDENCE_SET($A, B$)                ▷ $A$ and $B$ are product sets
    $cs \leftarrow \{\}$                                 ▷ initialize empty correspondence set
    $cr \leftarrow \{\}$                                 ▷ initialize empty category rules set
    for $a, b \in A \times B$ do                         ▷ Cartesian product of A and B
        if $known\_match(a, b)$ then
            $cs.add((a, b, True))$
            $cr.add((a.category, b.category))$
    $hnq \leftarrow 10 * len(correspondence\_set)$       ▷ hard negative quota
    $rnq \leftarrow hnq/4$                               ▷ random negative quota
    $appearances \leftarrow \{\}$
    for $p \in A \cup B$ do
        $appearances[p] \leftarrow 0$                    ▷ count record appearances in negative pairs
    for $a, b \in A \times B$ do
        if $known\_match(a, b)$ then
            continue
        else if $appearances[a] = 10$ or $appearances[b] = 10$ then
            continue
        else if $cr.contains((a.category, b.category)) = False$ then
            continue
        else if $relaxed\_jaccard(a, b) > 0.2$ then      ▷ compute on primary attribute(s)
            $cs.add((a, b, False))$
            $appearances[a] \leftarrow appearances[a] + 1$
            $appearances[b] \leftarrow appearances[b] + 1$
            $hnq \leftarrow hnq - 1$
            if $hnq = 0$ then
                break
    for $a, b \in A \times B$ do
        if $known\_match(a, b)$ then
            continue
        else if $appearances[a] = 10$ or $appearances[b] = 10$ then
            continue
        else if $cs.contains((a, b, False)) = False$ then
            $cs.add((a, b, False))$
            $appearances[a] \leftarrow appearances[a] + 1$
            $appearances[b] \leftarrow appearances[b] + 1$
            $rnq \leftarrow rnq - 1$
            if $rnq = 0$ then
                break
    return $cs$
```

non-matching pairs for the matching model to learn what distinguishes a match from a non-match, and including many obvious non-matches does not further this objective.

To sample from potential non-matches at random is also problematic, as the average randomly chosen pair of products is very easy to identify as a non-match. The goal is to include "similar" non-matches (i.e., close to the decision boundary) in the correspondence set, which allows the matching model to learn a good class boundary.

We use two levels of blocking to discard obvious non-matches. The first blocking pass is performed on the primary text attribute(s) using relaxed Jaccard index with inner Levenshtein distance, as performed in [21]. Jaccard index is a set similarity metric, computed for two sets A and B as $\frac{|A \cap B|}{|A \cup B|}$. Text attributes are tokenized at the word level. Instead of using exact equality to compute word equivalency, Levenshtein similarity is used.[3] The second blocking pass, performed only on the RetailerA/RetailerB problem, discards product pairs whose category attribute values pair does not arise in the known matches.

The total number of non-matches selected through the blocking process is capped at ten times the number of labeled matches. To avoid sampling bias, one random candidate pair is included for every four candidate pairs selected through the blocking process. No record from either dataset is featured in more than ten non-matching correspondence set pairs. Table 3 shows the total size, number of matches, and number of non-matches for each correspondence set.

[3]The Levenstein threshold is set to 0.7, and the Jaccard index threshold is set to 0.2 as in [21]

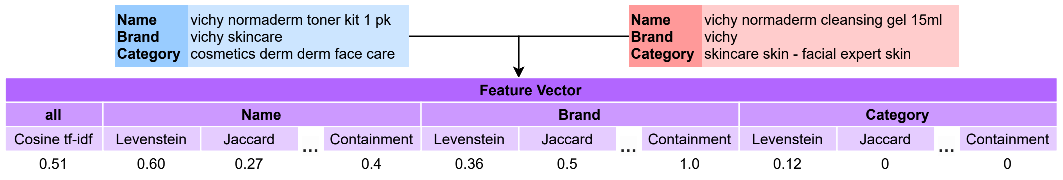
4. Methodology

In this section we describe the methods employed to investigate product matching on open source and proprietary data. With the advances in deep learning, we are interested to explore the use of these methods on the problem of product matching and to evaluate how they compare to more traditional machine learning methods. We generate classification models for product matching using three established methodologies: traditional machine leaning models [22], DeepMatcher hybrid models [2], and transformer architecture models [19]. We chose these three approaches for their recent state of the art performance on product matching tasks, and the availability of code from recent publications [2, 19, 21].

We aim to provide an overview of each model architecture and how the data is represented, however the reader should refer to the original papers for comprehensive and detailed explanations for how each model is implemented. The input for all the classification models will be a pair of products $(a, b),\ a \in \mathcal{A},\ b \in \mathcal{B}$ and the output is one of two labels (i.e., "match" or "non-match"). Although the input datasets are the same for all methods, the modelling of the data will be method dependant as described below.

4.1. Traditional Machine Learning Models

We train four classification models in this category: a **Decision Tree**, a **Random Forest**, a **Logistic Regression** classifier, and a Support Vector Machine (**SVM**) with a radial basis function (RBF) kernel [22]. These are classification systems that have been successfully employed in other record linkage problems [7–9].


Name
Brand
Category
vichy normaderm toner kit 1 pk
vichy skincare
cosmetics derm derm face care
Name
Brand
Category
vichy normaderm cleansing gel 15ml
vichy
skincare skin - facial expert skin
Feature Vector

all	Name				Brand				Category			
Cosine tf-idf	Levenstein	Jaccard	...	Containment	Levenstein	Jaccard	...	Containment	Levenstein	Jaccard	...	Containment
0.51	0.60	0.27		0.4	0.36	0.5		1.0	0.12	0		0

Figure 2. A feature vector example.

We compute feature vectors $(\phi_{(a,b)})$ for all (a, b) pairs of records in the correspondence set. Each (a, b) pair is represented by a feature vector $\phi_{(a,b)} = (x_1, x_2, ...x_n)$ where n corresponds to the number of common attributes of $\mathcal{A}$ and $\mathcal{B}$. x_i shows the level of similarity for the records a and b on attribute i. For text attributes, the similarity measures used are Levenshtein, Jaccard, relaxed Jaccard with inner Levenshtein, exact similarity, containment similarity, and TF-IDF cosine similarity. For numeric attributes, we compute absolute difference and exact similarity. These features are very similar to those generated by Magellan [15]. All the similarity measures return a value between 0 and 1, with 0 being least similar and 1 being most similar. In the event of missing attribute values, similarity values are set to -1. Figure 2 shows an example of one of these vectors. All classification models in this section take as input features vectors $\phi_{(a,b)} = (x_1, x_2, ...x_n)$ along with a label (i.e., "match" or "non-match") for the entire correspondence set.

4.2. DeepMatcher Models

The DeepMatcher [2] hybrid model combines a sequence aware model (a bidirectional RNN) with an attention mechanism. To represent candidate pairs to the classifier, individual product attribute values are converted to sequences of character or word level embeddings. Each embedding has a consistent dimensionality and each sequence of embeddings can be of a variable length. Pre-trained embeddings are used: **fasttext.en.bin** uses character level embeddings released by fastText, **glove.6B.300d** uses word level embeddings trained on Wiki + Gigaword with uncased GloVe, and **glove.42B.300d** uses word level embeddings trained on Common Crawl with uncased GloVe.

Pairs of attribute embedding sequences are then used to compute attribute similarity vectors. The DeepMatcher hybrid model distinguishes itself from the more rudimentary similarity metrics used by the machine learning models through more nuanced and expressive text field comparison. Attribute similarity vector computation is sequence aware, allowing the model to learn complex relationships between the order and semantic significance of an single attribute value's embedding sequence. It is also context aware, allowing it to compute a soft alignment between two embedding sequences and then perform token by token comparisons. These attribute similarity vectors are merged into a single entity similarity vector, which is fed into a final neural network used to make the matching prediction.

4.3. **Entity Matching Transformer Models**

Transformer architectures have become popular recently for NLP tasks, and have started seeing use in the record linkage domain [19, 23]. The higher inherent capacity for parallelism they offer over RNNs can lower training times. At a high level, the flow of this approach is very similar to DeepMatcher. The primary difference is that the RNN used to compute attribute similarity is replaced by a modern transformer architecture for entity matching. Prior work indicates that transformer models excel compared to other approaches when applied to dirty and/or unstructured data. As such, we include this model to test the hypothesis that traditional ML techniques outperform this model in both performance and training time on clean, structured data.

We use the code[4] published by [19] to test this hypothesis. We consider two different models: **RoBERTa** has distinguished itself via state of the art results on a number of NLP tasks, whereas **DistilBERT** offers shorter training times and a reduced model size. The code is structured to accept a single unlabeled text string to represent each product. For Abt/Buy we concatenate the product name and description, for Amazon/Google we use only the product name, for Amazon/Walmart we use only the product title, and for RetailerA/RetailerB we use only the product name.

4.4. **Data Profiling**

Primpeli and Bizer [21] proposed five dimensions that can be used to profile matching-related challenges. We use these dimensions to form a hypothesis about which matching methods are the most appropriate for our dataset. We briefly restate each dimension here, and we use the same methodology to profile our proprietary data to show how it compares to the open source datasets. Schema Complexity (SC) is the number of shared schema attributes that contribute to the matching task. Sparsity (SP) is the percentage of missing values for all attributes in the correspondence set. Textuality (TX) is the average number of words present in the top relevant attributes. For our dataset, this considers only of the product name attribute. Development Set Size (DS) represents available training data. It is computed as the combined size of the training and validation sets[5]. Corner Cases (CC) attempts to capture how many difficult candidate pairs are present in the correspondence set. It is computed by averaging all individual similarity measures present in the machine learning feature vectors (the creation of which is described in Section 4.1), and then identifying the a threshold that maximizes the F_1 score (F_1 is defined in Section 5.1). This dimension is

[4]https://github.com/brunnurs/entity-matching-transformer

[5]Since we are using 10-fold cross validation, we report the rounded average of the 10 folds for this dimension

computed as the percentage of pairs in the correspondence set that are incorrectly classified using this threshold. All dimensions except schema complexity are computed in the context of the correspondence set. Profiling results are shown in Table 3.

Table 3. Correspondence set size, Schema Complexity (SC), Sparsity (SP), Textuality (TX), Development Set Size (DS) and Corner Cases (CC) for each matching task.

	Correspondence Set			Profiling Dimensions				
Matching Task	Size	#Matches	#Non-Matches	SC	SP	TX	DS	CC
Abt/Buy	7,154	1,095	6,059	3	0.23	8.69	6,439	0.74
Amazon/Google	8,440	1,298	7,142	3	0.03	130.22	7,596	0.68
Amazon/Walmart	15,579	1,154	14,425	5	0.08	10.52	14,021	0.27
RetailerA/RetailerB	17,874	1,324	16,550	3	0.03	6.57	16,087	0.04

The three open source datasets are characterized by high textuality and many corner cases. In contrast, our proprietary data has lower schema complexity, lower sparsity, and shorter attribute values. Based upon the groupings presented in [21], we expect our proprietary data to be a good fit for traditional machine learning approaches, while the other datasets may benefit from the deep learning approaches (Note that Primpeli et. al [21] investigated only Random Forest and SVM in their paper).

5. Evaluation Framework and Results

Based on data profiling, we formulate the following hypothesis: traditional machine learning methods will perform well on our proprietary data. We want to investigate this hypothesis in a consistent evaluation framework, to understand what method performs better on what type of data and to observe if there are any trade-offs (e.g., time complexity) a practitioner should consider. This allows us to make recommendations for other practitioners that are interested in using these methods for their product matching needs. To achieve this goal we propose an evaluation framework for comparing product matching techniques. The novelty of the framework is that it consistently compares all methods over a stratified 10-fold cross validation set, and it provides precision recall curves which allows performance evaluation over the entire range of precision/recall values. In addition to testing this hypothesis, we wish to answer the following research questions in this section:

- How do different classification systems perform on product matching tasks?
- Do traditional machine learning methods perform well on the proprietary data as suggested by data profiling?
- Can we draw a general conclusion about which system a practitioner should choose for their product matching task?
- Does the data modeling and training time impact the choice of the system?

5.1. Evaluation Measures

To evaluate the performance of all the systems we use precision and recall. Precision is the fraction of correctly predicted matches out of all the predicted matches and recall is the fraction of correctly predicted matches out of all the matches. F_1 measure combines precision and recall in a single measure by using the harmonic mean.

$$P = \frac{TP}{TP + FP} \qquad R = \frac{TP}{TP + FN} \qquad F_1 = \frac{2 \times P \times R}{P + R}$$

In addition, we compute and plot precision recall curves which show the trade-off between precision and recall over a range of thresholds. These curves offer more insight into model performance than precision, recall, and F_1 score alone [24]. More detail on how these curves are computed when using 10-fold cross validation is provided in Appendix A. In record linkage or entity matching tasks we are not concerned with true negatives (TN), as the majority of record pairs will be non-matches.

5.2. **Experimental Setup**

All of our experiments are performed using 10-fold cross validation. The correspondence set for each matching task is split into ten stratified folds. Each fold is assigned to the validation set in one of the rounds, to the test set in one of the rounds, and to the train set in the remaining eight rounds. For the open source datasets, we use the published correspondence sets from [21], generated using the process described in Section 3.2. We use this same process with the addition of category blocking to generate the correspondence set for our proprietary data, as described in Algorithm 1. We use scikit-learn [25] for the implementation of the traditional machine learning approaches, DeepMatcher [2], and EMT [19] for the deep learning methods.

Table 4. Parameters tested in the grid search.

Decision Tree		max. depth	min. leaf size
		[1, 3, 5]	[5, 10, None]
Random Forest	estimators	max. depth	min. leaf size
	[10, 100, 500]	[1, 3, 5]	[5, 10, None]
Logistic Regression		C	penalty
		logspace(-2, 5, 10)	[l1, l2]
SVM		C	gamma
		logspace(-2, 5, 10)	[1^{-6}, 1^{-4}, 1^{-2}, 1, 10]

We perform a hyperparameter grid search for the traditional machine learning models. Tested parameters are shown in Table 4. Average F_1 score on the validation set was used to select the best parameters. Deep learning models were trained over 15 epochs, with average F_1 score on the validation set at the end of each epoch used to select the best model. The number of epochs and all the other parameters for the deep learning models were set as described in their respective papers [2, 19]. Average test set performance is reported for each of the best models in Section 5.3. When a hyperparameter grid search was performed, average training time is only reported for the best parameters.

5.3. **Results**

Table 5 shows the performance of all systems on the three open source datasets and one proprietary dataset along with the training times. We use bold font to show the best performance in terms of F_1 score for each dataset. We underline the best performance in terms of F_1 score for each classification system. The results show that the hypothesis we formulated holds and that traditional machine learning methods (i.e., Random Forest) perform well on the proprietary data. The results show that Random Forest outperforms all other methods on three of the four datasets. One of the deep learning methods (RoBERTa) performs best on the Amazon/Google dataset. According to the data profiling Amazon/Google had the highest textuality. The deep learning methodology was able to handle this data characteristic better.

Can we draw a general conclusion about which system a practitioner should choose for their product matching task? At first glance Random Forest seems a good choice. The PR curves in Figure 3 give us a better look at the overall performance of the top performing classification systems in each category (i.e., Random Forest for traditional machine learning, fasttext.en.bin for DeepMatcher and, RoBERTa and DistilBERT for the Entity Matching Transformer models). Random Forest outperforms all the other methods over the entire range of precision recall values on Abt/Buy, Amazon/Walmart and RetailerA/RetailerB datasets. The ranking of the classifiers and their performance is not as clear on Amazon/Google, as the PR curves are overlapping at times. In this instance, a practitioner might choose a classifier depending on the accepted level of precision or recall for their particular application. The complexity of the deep learning approaches require longer training times, while all the traditional machine learning methods train in seconds.

Table 5. Average training time and F_1 score for each matching task and model type.

	Abt/Buy		Amazon/Google		Amazon/Walmart		RetailerA/RetailerB	
Model	Time	F_1	Time	F_1	Time	F_1	Time	F_1
Decision Tree	0.036s	0.78	0.031s	0.72	0.197s	0.92	0.012s	**<u>0.98</u>**
Random Forest	1.620s	**0.82**	1.801s	0.76	0.445s	**0.94**	0.037s	**<u>0.98</u>**
Logistic Regression	0.238s	0.79	1.770s	0.68	4.737s	0.90	18.079s	**<u>0.98</u>**
SVM	0.600s	0.81	4.598s	0.73	0.714s	0.93	5.051s	**<u>0.98</u>**
fasttext.en.bin	8m 45s	0.74	2h 03m	0.81	1h 54m	<u>0.93</u>	24m 06s	0.85
glove.6B.300d	10m 37s	0.55	1h 59m	0.73	1h 54m	0.75	22m 37s	<u>0.83</u>
glove.42B.300d	11m 38s	0.56	2h 00m	0.77	1h 55m	0.82	22m 45s	<u>0.83</u>
RoBERTa	26m 27s	0.65	30m 57s	**0.84**	56m 50s	<u>0.92</u>	1h 05m	0.86
DistilBERT	13m 29s	0.68	15m 37s	0.82	28m 42s	<u>0.89</u>	33m 04s	0.86

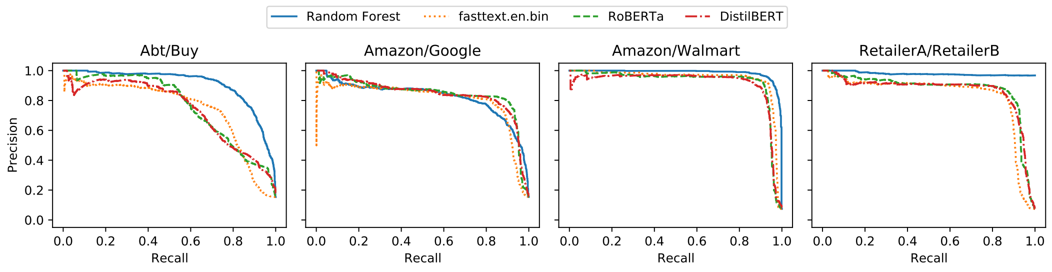


Figure 3. Precision recall curves for the best traditional ML, DeepMatcher, and EMT models.

5.4. **Classification System Deployment**

In the previous section we have evaluated and discussed the performance of the classification systems on the testing data. However, the goal of training a classifier is to deploy it in practice where the model will classify unseen instances. In this section, we present results for the deployment of the classifier on RetailerA/RetailerB data.

Table 6. Examples of classified unseen pairs during the deployment.

Pair	Retailer	Product name	Product category	Product brand	label
1	A	katy perry killer queen edp 100 ml	cosmetics fragrances fragrance ladies	katy perry	match
	B	katy perry killer queen 100ml	fragrance female fine frag female fine frag	katy perry	
2	A	batiste dry shampoo blush 200 ml	health beauty care hair care shampoo	batiste	match
	B	batiste dry shampoo blush 200ml	hair (brushes-combs) hair mass dry shampoo	batiste	
3	A	batiste dry shampoo tropical 200 ml	health beauty care hair care shampoo	batiste	non-match
	B	batiste dry shampoo oriental 200ml	hair (brushes-combs) hair mass dry shampoo	batiste	
4	A	avene cold cream 40 ml	cosmetics derm derm face care	avene skincare	non-match
	B	avene cold cream 100ml	skincare skin - facial expert skin	avene	
5	A	la roche posay effaclar h 40 ml	cosmetics derm derm face care	la roche posay skincare	non-match
	B	la roche-posay effaclar k 40ml	skincare skin - facial expert skin	la roche - posay	

We select 10,000 unseen pairs of records and we use Random Forest to classify them as a "match" or a "non-match". None of these pairs are part of the correspondence set used for training/testing. 500 of these pairs were chosen for their similarity and 9,500 pairs were drawn at random. These are respectively analogous to the hard and random negatives in the correspondence set. Out of the 10,000 pairs 6 were predicted as a "match" and the rest as a "non-match" (The reader should note that this is a desirable property, as most pairs in the Cartesian product of two sets will be non-matches). We do not have labels for these pairs, thus we inspect them manually and we qualitatively assess the predicted labels. The manual inspection of these pairs allows us to better understand where the classification system performs well and where it could be improved or changed to address application specific demands. We show in Table 6 some example pairs from the deployment stage. Pair 1 and 2 were predicted as matches and upon manual inspection they seem to be the same product, as product name and brand match almost exactly. Pairs 3, 4, and 5 are predicted as non-matches. We consider pairs 3 and 4 as being predicted correctly (products in Pair 3 look like two different shampoos given "tropical" vs. "oriental"; products in Pair 4 seem to be the same cream, but the size is different). Pair 5 is an interesting example, as product names are almost identical save for a one letter indicator. Pair 5 is a potential false negative. Such false negatives can be addressed by talking with the retailers to better understand the significance of their notation.

6. Conclusions

In this paper we investigated the performance of several product matching systems in a consistent evaluation framework that we proposed. We compared the use of traditional machine learning methods and newer deep learning approaches, allowing us to make informed recommendations to potential practitioners who want to use product matching for data integration. We conducted a detailed experimental study and our findings show that traditional machine learning methods perform well on this task and a practitioner should investigate these methods first. As a next step, we plan to measure the practical usage of these results by incorporating the product matching outcome with some real world use cases (e.g., forecasting demand for new products) to better understand the advantage product matching offers to this use case.

Appendix A. Precision Recall Curve Creation

Precision recall curves are constructed using the predicted match probabilities for the test set. As the threshold value used to classify matches is lowered, recall monotonically increases. Precision recall curves show how this affects model precision. Since we perform 10-fold cross validation, we have 10 curves for each model type. We choose to show a single curve generated by concatenating all of the test set predictions across the 10 folds into a single vector, shown in Figure 4. This one curve should should not be interpreted as the performance of a specific model, but rather as the expected precision/recall of the model building process.

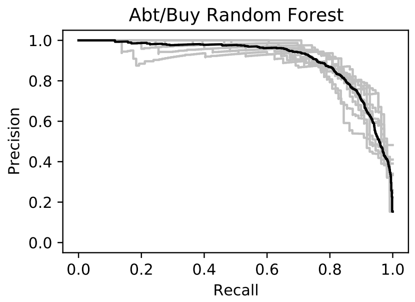


Figure 4. This figure shows how the single representative precision recall curves for each model type shown in Figure 3 relate to the ten precision recall curves from the 10-fold cross validation. The curves for Abt/Buy Random Forest models are used as an example.

References

[1] H. Köpcke, A. Thor, and E. Rahm. "Evaluation of entity resolution approaches on real-world match problems". In: *Proceedings of the VLDB Endowment* 3.1-2 (2010), pp. 484–493.

[2] S. Mudgal, H. Li, T. Rekatsinas, A. Doan, Y. Park, G. Krishnan, R. Deep, E. Arcaute, and V. Raghavendra. "Deep learning for entity matching: A design space exploration". In: *Proceedings of the 2018 International Conference on Management of Data*. 2018, pp. 19–34.

[3] A. Y. Sim and A. Borthwick. "Record2Vec: unsupervised representation learning for structured records". In: *International Conference on Data Mining*. IEEE. 2018, pp. 1236–1241.

[4] D. Qiu, L. Barbosa, X. L. Dong, Y. Shen, and D. Srivastava. "Dexter: Large-Scale Discovery and Extraction of Product Specifications on the Web". In: 8.13 (Sept. 2015), pp. 2194–2205.

[5] A. Kannan, I. E. Givoni, R. Agrawal, and A. Fuxman. "Matching Unstructured Product Offers to Structured Product Specifications". In: *Proc. of the 17th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. ACM, 2011, pp. 404–412.

[6] I. P. Fellegi and A. B. Sunter. "A Theory for Record Linkage". In: *Journal of the American Statistical Association* 64 (1969), pp. 1183–1210.

[7] W. E. Winkler. *Overview of Record Linkage and Current Research Directions*. Statistical Research Division Report. 2006.

[8] A. K. Elmagarmid, P. G. Ipeirotis, and V. S. Verykios. "Duplicate Record Detection: A Survey". In: *IEEE Transactions on Knowledge and Data Engineering* 19 (2007), pp. 1–16.

[9] P. Christen. *Data Matching: Concepts and Techniques for Record Linkage, Entity Resolution, and Duplicate Detection*. Springer Publishing Company, 2012.

[10] H. Kang, L. Getoor, B. Shneiderman, M. Bilgic, and L. Licamele. "Interactive Entity Resolution in Relational Data: A Visual Analytic Tool and Its Evaluation". In: *IEEE Transactions on Visualization and Computer Graphics* 14.5 (2008), pp. 999–1014.

[11] M. Bilgic, L. Licamele, L. Getoor, and B. Shneiderman. "D-Dupe: An Interactive Tool for Entity Resolution in Social Networks". In: *Visual Analytics Science and Technology (VAST)*. Baltimore, Oct. 2006.

[12] R. C. Steorts, S. L. Ventura, M. Sadinle, and S. E. Fienberg. "A Comparison of Blocking Methods for Record Linkage". In: *Privacy in Statistical Databases*. Springer International Publishing, 2014, pp. 253–268.

[13] M. Bilenko, S Basil, and M. Sahami. "Adaptive product normalization: Using online learning for record linkage in comparison shopping". In: *Fifth IEEE International Conference on Data Mining (ICDM'05)*. IEEE. 2005, 8–pp.

[14] P. Christen. "Febrl- an open source data cleaning, deduplication and record linkage system with a graphical user interface". In: *Proceedings of the 14th ACM SIGKDD international conference on Knowledge discovery and data mining*. 2008, pp. 1065–1068.

[15] P. Konda, S. Das, P. Suganthan GC, A. Doan, A. Ardalan, J. R. Ballard, H. Li, F. Panahi, H. Zhang, J. Naughton, et al. "Magellan: Toward building entity matching management systems". In: *Proceedings of the VLDB Endowment* 9.12 (2016), pp. 1197–1208.

[16] P. Petrovski and C. Bizer. "Extracting Attribute-Value Pairs from Product Specifications on the Web". In: *Proceedings of the International Conference on Web Intelligence*. Leipzig, Germany: Association for Computing Machinery, 2017, pp. 558–565.

[17] 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". In: *Semantic Web* 9 (Aug. 2018), pp. 1–22.

[18] J. Li, Z. Dou, Y. Zhu, X. Zuo, and J.-R. Wen. "Deep cross-platform product matching in e-commerce". In: *Information Retrieval Journal* 23.2 (2020), pp. 136–158.

[19] U. Brunner and K. Stockinger. "Entity matching with transformer architectures-a step forward in data integration". In: *Proc. of Conf. on Extending Database Technology, Copenhagen*. 2020.

[20] J. Tracz, P. I. Wójcik, K. Jasinska-Kobus, R. Belluzzo, R. Mroczkowski, and I. Gawlik. "BERT-based similarity learning for product matching". In: *Proceedings of Workshop on Natural Language Processing in E-Commerce*. 2020, pp. 66–75.

[21] A. Primpeli and C. Bizer. "Profiling entity matching benchmark tasks". In: *Proc. of the 29th International Conference on Information & Knowledge Management*. 2020, pp. 3101–3108.

[22] T. Hastie, R. Tibshirani, and J. Friedman. *The elements of statistical learning: data mining, inference and prediction*. 2nd ed. Springer, 2009.

[23] R. Peeters, C. Bizer, and G. Glavaš. "Intermediate training of BERT for product matching". In: *small* 745.722 (2020), pp. 2–112.

[24] D. Hand and P. Christen. "A Note on Using the F-Measure for Evaluating Record Linkage Algorithms". In: *Statistics and Computing* 28.3 (May 2018), pp. 539–547. ISSN: 0960-3174.

[25] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. “Scikit-learn: Machine Learning in Python”. In: *Journal of Machine Learning Research* 12 (2011), pp. 2825–2830.