A Comparison Framework for Product Matching Algorithms

by
Jeremy Foxcroft

A Thesis
presented to
The University of Guelph

In partial fulfilment of requirements
for the degree of
Master of Science
in
Computer Science

Guelph, Ontario, Canada
© Jeremy Foxcroft, September, 2021
ABSTRACT

A COMPARISON FRAMEWORK FOR PRODUCT MATCHING ALGORITHMS

Jeremy Foxcroft  Advisor:  Dr. Luiza Antonie
University of Guelph, 2021

Product matching is a specific application of record linkage where different digital records that refer to the same product are identified. In this thesis, we design a framework to compare state-of-the-art product matching systems including traditional machine learning models and more recent deep learning approaches. We then employ this system to perform comparisons on both open source product matching benchmarks and real-world modern day industrial product data, measuring performance with both F1 measure and precision-recall curves. We find that traditional machine learning techniques remain superior for clean, structured data and that this superior performance translates seamlessly from the open source product matching benchmarks to the real-world data. We also propose a new application for product matching: forecasting demand for products that are new to market.
ACKNOWLEDGMENTS

First and foremost, I am grateful to my supervisor Dr. Luiza Antonie for her invaluable advice and continuous support during my study. She showed patience when I was stressed, provided direction when I got stuck, and met all my doubts with the unwavering belief that I was up to the challenge.

I am deeply grateful to Dr. Gary Grewal. He offered me my first research opportunity in 2017, which went on to shape and direct my entire university experience. His kindness, wisdom, and mentorship have meant more to me than I can say.

I would like to thank my parents, Maja and John Foxcroft, for their unwavering support regardless of what I choose to pursue. I owe so much of who I am to you. I love you both very much.

Finally, I would like to thank Ian, Trish, Ethan, and Seth Fogarty. Their hospitality made my time studying, writing, and living in New Brunswick a wonderful experience.
# Table of Contents

Abstract ii

Acknowledgements iii

List of Tables vi

List of Figures vii

1 Introduction 1
   1.1 Motivation .................................................. 2
   1.2 Thesis Statement ........................................... 3
   1.3 Contributions ............................................... 3
   1.4 Overview of Thesis ......................................... 3

2 Background 5
   2.1 Record Linkage Background ................................. 5
   2.2 Record Linkage Stages ..................................... 6
      2.2.1 Data Preprocessing ................................. 7
      2.2.2 Blocking ............................................. 7
      2.2.3 Comparison ........................................... 8
      2.2.4 Classification ....................................... 9
      2.2.5 Evaluation .......................................... 10
   2.3 Product Matching .......................................... 12
      2.3.1 Challenges ........................................... 13
      2.3.2 Prevalence ........................................... 13
      2.3.3 Components .......................................... 14
      2.3.4 Contributors ......................................... 16
      2.3.5 Machine Learning Techniques ....................... 16
      2.3.6 State-of-the-Art ..................................... 18
      2.3.7 Looking Forward ..................................... 18
3 Data
3.1 Public Data ................................................. 20
3.2 Proprietary Data ........................................... 21
3.2.1 Data Labeling ........................................... 23

4 Methodology and Implementation 25
4.1 Correspondence Set Generation ............................. 25
  4.1.1 Blocking on Product Name Similarity .................. 28
  4.1.2 Blocking on Category .................................. 30
4.2 Data Profiling .............................................. 31
4.3 Classifiers ................................................ 33
  4.3.1 Traditional Machine Learning Models .................. 34
  4.3.2 DeepMatcher Models .................................... 36
  4.3.3 Entity Matching Transformer Models ................... 38
4.4 Evaluation Framework .................................... 39
  4.4.1 Experimental Setup .................................... 40
  4.4.2 Precision Recall Curves ................................. 42

5 Results 44
5.1 Classifier Performance .................................... 44
5.2 Simulating Deployment .................................... 47
5.3 Use Cases ................................................ 49

6 Conclusions 52
6.1 Future Work .............................................. 52

Bibliography 54

A Appendix 60
  A.1 Levenshtein Similarity .................................... 60
  A.2 Jaccard Similarity ........................................ 60
  A.3 Jaccard Similarity with Inner Levenshtein ............... 61
  A.4 Exact Similarity ........................................... 61
  A.5 Containment Similarity .................................... 61
  A.6 Absolute Difference ...................................... 61
  A.7 TF-IDF Cosine Similarity .................................. 62

B Appendix 63
List of Tables

3.1 An overview of the public and proprietary data. . . . . . . . . . . . . 22
3.2 Examples of products from Retailers A/B. . . . . . . . . . . . . . . 22
3.3 Corresponding categorization examples. . . . . . . . . . . . . . . . 23

4.1 Correspondence set sizes and distributions . . . . . . . . . . . . . . . 27
4.2 Profiling results . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 33
4.3 Hyperparameter values . . . . . . . . . . . . . . . . . . . . . . . . . . 40

5.1 Average precision and recall . . . . . . . . . . . . . . . . . . . . . . . 44
5.2 Average training time and $F_1$ score . . . . . . . . . . . . . . . . . 45
5.3 Examples of unseen pairs classified as “matches” . . . . . . . . . . . 48
5.4 Examples of unseen pairs classified as “non-matches” . . . . . . . . 48
## List of Figures

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.1</td>
<td>Overview of record linkage stages</td>
<td>6</td>
</tr>
<tr>
<td>4.1</td>
<td>A feature vector example</td>
<td>35</td>
</tr>
<tr>
<td>4.2</td>
<td>Cross validation folds</td>
<td>41</td>
</tr>
<tr>
<td>4.3</td>
<td>Precision recall curve creation</td>
<td>42</td>
</tr>
<tr>
<td>5.1</td>
<td>Precision recall curves for the best models</td>
<td>46</td>
</tr>
<tr>
<td>B.1</td>
<td>Precision recall curves for all ML models</td>
<td>63</td>
</tr>
<tr>
<td>B.2</td>
<td>Precision recall curves for all DeepMatcher models</td>
<td>64</td>
</tr>
<tr>
<td>B.3</td>
<td>Precision recall curves for all EMT models</td>
<td>65</td>
</tr>
</tbody>
</table>
Chapter 1

Introduction

Record linkage is the process by which two data records, often from different data sources, are identified as referring to the same entity (e.g., product in a store). When records can be uniquely identified by a common attribute this task is easy. When records do not have a unique common attribute, other common attributes (e.g., name, price, brand) can be used to identify similar records and classify them as matching or non-matching. In the retail space, there are a number of applications for this type of product matching.

Online price comparison tools perform record matching on products (i.e., product matching) to find advertisements for the same products from different merchants. Detailed product listings and productcatalogues can be created by accumulating product data from many different sources.

As businesses increasingly look for ways to use data and machine learning to improve profitability, one application that has proven valuable is demand series forecasting. This is the process of using historical demand for products to predict future demand. For products that are new to market or otherwise lack a sales history, product matching can be used to identify similar products whose history can be substituted for demand series forecasting.

This thesis explores state-of-the-art techniques for record linkage in the product matching domain. Using both open public matching benchmarks and current industrial data, we propose a framework to evaluate and compare different approaches. We also make recommendations regarding different approach’s applicability to product matching problems.
1.1 Motivation

The work in this thesis is motivated by trends in data, academia, and industry. With the release of new products and the growing online presence of retail, product records are more numerous and have a higher potential complexity than ever before. The open-source datasets [13, 26] used for product matching contain at least one order of magnitude fewer product records than the inventory of most modern retailers. While some retailers may keep very detailed product records, others may have product records that are far more terse than the open-source product datasets. These quantitative and qualitative differences between open-source and industrial product data motivate the exploration into which linkage techniques are most suitable for a given matching task.

In academia, many different linkage approaches have been proposed [3, 26, 39]. Recent interest in product linkage has motivated the exploration of multiple deep learning model architectures [6, 28, 32, 41, 46, 49]. This fast paced exploration has led to the lack of thorough comparisons between new techniques and existing techniques, which motivates the need for a consistent framework within which to perform these comparisons (i.e., a consistent methodology for creating training/testing sets and approach to measuring model performance). Many of the newer deep learning techniques have only been applied to open-source product datasets up until this point, thus this work also seeks to explore to what extent the improved performance of deep learning approaches on open-source product data will translate to industrial product data.

Finally, modern retailers are increasingly looking to apply machine learning and deep learning techniques to customer retention through targeted offers, product demand forecasting, and more. Product matching can be used to both clean and augment product data, making it a valuable precursor to any of the above work. This increased scope of the potential benefits to product matching also serves to motivate
this assessment of the current state-of-the-art techniques.

1.2 Thesis Statement

This thesis proposes a consistent framework for comparing product matching algorithms, and demonstrates that the performance of traditional machine learning approaches remains state-of-the-art and transfers seamlessly from open source data to industrial data.

1.3 Contributions

The main contributions of this thesis are as follows:

- A consistent comparison framework to compare and evaluate record linkage models.
- The evaluation of state-of-the-art record linkage models on both public and industrial product data.
- A general approach for identifying whether traditional machine learning or deep learning techniques are suitable for a given set of data.

Publications arising from this work include [17].

1.4 Overview of Thesis

The remainder of this thesis is structured as follows: Chapter 2 will provide a background for record linkage, and presents a literature review for work in the area of product matching. In Chapter 3, the data used in this thesis is introduced. In Chapter 4, the different approaches to product matching compared in this thesis are introduced. Then in Chapter 5 we analyze the results to perform a comparison of
the different model types and discuss potential use cases. The thesis concludes in Chapter 6 with a summary and directions for future work.
Chapter 2

Background

This chapter presents a background of record linkage in Section 2.1, a description of the stages of record linkage in Section 2.2, and a literature review for product matching in Section 2.3.

2.1 Record Linkage Background

The study of record linkage dates back decades. It is referred to by many names including data matching [9], entity resolution [26], and deduplication [44]. The exact term used to describe this task varies between different research groups, research communities, and applications [10].

Record linkage was first proposed by Dunn in 1946 [14]. He suggested that each person’s life could be captured within a book that starts with their birth, ends with the death, and encloses all important records within. He defined record linkage as the process of assembling such a book.

When records are not consistently identified through a shared unique identifier, it is not a straightforward task to perform record linkage. Research has focused on using the similarity of shared schema attributes to identify matching pairs of records. This probabilistic approach to record linkage was first described in 1959 by Newcombe et al. [33] and later formalized by Fellegi and Sunter in 1969 [16].

In more recent years, machine learning algorithms have slowly replaced probabilistic linkage. These newer techniques often yield superior accuracy provided with sufficient labeled training data [53].
While record linkage can be performed by hand, it is typically performed algorithmically on computers for modern problems. This has led to the ability to process larger data sets, the elimination of some forms of human error in the linkage process, and easier reproducibility of linkage results.

Recent surveys of record linkage have been performed by Winkler [54], Elmagarmid et al. [15] and Christen [9].

2.2 Record Linkage Stages

Record linkage identifies records that refer to the same entity, often across different data sources. When records share a unique identifier, this process can be as simple as performing a database join. With many real-world data, it is common for this process to be more complicated. When records lack a shared unique identifier, linkage systems use other attributes like name, brand, and price to identify matching records.

Figure 2.1: Overview of record linkage stages.

Record linkage usually involves finding links between two datasets, but can also be performed within a single dataset (commonly referred to as deduplication) or between more than two datasets. Linkage is performed in the five stages shown in
2.2.1 Data Preprocessing

Pre-processing is the first stage of the record linkage process, and is typically composed of two parts: cleaning and standardization. Often, the values in data being linked are not perfect. This could take the form of missing values, incorrect values, incomplete values, or values that are categorized under the wrong attribute. To perform meaningful comparisons between records, these errors should be identified and corrected where possible.

It is easier to compare two records when the datasets share the same schema. Standardization is the process of enforcing this. This may include discarding attributes from one of the datasets that does not have a counterpart in the other schema. Sometimes multiple attributes have been composed in one of the schemas, but not in the other. For example, the price, brand, and name of a product might be written together as a single text description. Separating this text description into three discrete attributes allows for more nuanced comparisons to be made later in the linkage process.

2.2.2 Blocking

A naive approach to record linkage is to compare every record in the one dataset to every record in the other dataset. For two record sets $\mathcal{A}$ and $\mathcal{B}$, the number of potential matches grows with the size of the Cartesian product $|\mathcal{A} \times \mathcal{B}|$. These potential matches can be referred to as the correspondence set [39].

The size of the correspondence set grows quadratically with the number of records in $\mathcal{A}$ and $\mathcal{B}$. If the correspondence set becomes too large, it is possible that the record pairs can no longer be stored in memory, or that fully comparing all record pairs in the next stage is too computationally expensive. One solution to this problem is blocking, which reduces the number of potential matches in the correspondence set [9].
Traditionally, blocking is performed by using an existing attribute or an attribute that can be easily precomputed for each record (referred to as a blocking key) and only inserting record pairs into the correspondence set if they match on this attribute. For example, the first letter of a product brand might make a good blocking key. A good blocking key is easy to compute, and reliably matches for records that refer to the same entity.

Blocking is always a trade-off, as it simultaneously seeks to maximally reduce the candidate set size and minimally discard matching records. Sometimes, multiple blocking keys are used in combination to more aggressively reduce the candidate set size (e.g., record pairs in the candidate set must have brands starting with the same letter and have the same price). Other times, blocking keys may be used separately to better capture all potential matches in the candidate set (e.g., record pairs in the candidate set must have brands starting with the same letter or have the same price).

Some blocking strategies may also be performed without a blocking key. This is also referred to as indexing. For example, this might involve adding pairs of records to the correspondence set if their prices do not differ by more than five dollars. There is no blocking key for this approach, but it is still a computationally efficient approach to reducing the number of record pairs that need to be fully compared in the next stage.

Antonie et al. [1] provides good examples of blocking using both blocking keys and computed properties. Febrl [8], an open-source record linkage framework, provides implementations for a variety of common blocking strategies.

2.2.3 Comparison

After blocking, remaining record pairs in the correspondence set are subjected to a broader set of comparison functions. For all pairs of records \((a, b)\), a feature vector \(\phi_{(a,b)} = (x_0, x_1, \ldots, x_n)\) is computed.

Each axis in the resulting vector space represents a specific comparison made
between two product records. For example, two textual product names might be compared using Levenshtein similarity [27] (a heuristic for string similarity) to provide a value for $x_0$. The numeric difference between two prices might be used to provide a value for $x_1$. There is no constraint on $n$, the dimensionality of the feature vector.

Difference comparisons seek to capture different information about how two records are similar or different. Each comparison is usually biased towards measuring records as similar if they differ in a specific way. Levenshtein similarity will measure two strings as very similar if one string can be transformed into another with a small number of character level insertions, deletions, and replacements. Jaccard index [21] applied at the word level to two strings will measure two strings as similar if they contain mostly the same words, even if the words appear in a different order.

Selection of these comparison functions will depend on the attributes and characteristics in the data. Careful selection of the dimensions in the feature vector for a specific dataset is called feature engineering, and is often employed to improve classifier performance. There is no rule that says a single axis $x_i$ has to compare only one attribute value from each record, sometimes it may be helpful to compare multiple attributes to compute a single numeric similarity value.

### 2.2.4 Classification

In the classification stage, each record pair is predicted to match or not match using its feature vector $\phi$ from the previous stage. Before machine learning, this might have been achieved by a domain expert writing a set of rules to try and capture the set of conditions that must be met for two records to match. Modern classifiers learn this decision boundary automatically.

Supervised classifiers use labeled record pairs (that have been identified as matching or non-matching) to divide the vector space of the feature vectors into different regions, with each region corresponding to a “match” or “non-match” prediction. Support Vector Machines [11] and Random Forests [20] are two examples of such
classifiers that have seen significant use in record linkage [9, 15, 54].

Unsupervised methods can infer a model (from the feature vectors alone) which can be used to predict whether a given pair of records is a match or a non-match. Clustering approaches, such as K-means clustering [30], fall into this category. In general, supervised record linkage techniques are preferred when labeled data is available.

For deep learning a similarity feature vector is still created prior to classification, however the similarity feature vector is more of a black box in that it is not always possible to understand what an individual component $x_i$ is measuring.

2.2.5 Evaluation

Without labeled data, measuring the performance of a classifier is a real problem. A human (often a domain expert) can manually review the links to get some feel for the performance. This can prove time-consuming if there are many predicted matches, and also would fail to identify matching records that are not caught by the classifier. Some common sense checks can be performed, such as looking at the number of links relative to the size of the datasets, or looking for records in one dataset that match an unexpectedly large or small number of records in the other dataset. Unfortunately, this can only tell practitioners so much.

When labeled data is available, there are many measures that can be used to evaluate performance. When training supervised models, labeled data is split into training, validation, and testing sets. The training and validation data are used to create the classifier, and then predictions are made on the testing set to compute performance.

Comparing the labels of record pairs with a classifier’s predictions can be used to categorize the results. True Positives (TP) are the record pairs labeled and predicted as matches. True Negatives (TN) are the record pairs labeled and predicted as non-matches. False Positives are the record pairs which are labeled as non-matches but
predicted as matches. False Negatives are the record pairs that are labeled as matches but are predicted as non-matches.

Looking at any one of these four numbers by itself does not communicate much. TPs and TNs tend to imply good performance, whereas FPs and FNs tend to imply that the classifier is making mistakes. Depending on where the decision boundary is, the number of records predicted as matches might increase or decrease (which generally increases or decreases both the TPs and the FPs).

Precision and recall [38] are two measures that capture how this affects performance. Precision measures how many of the predicted matches are correct, and recall measures how many of the matching records are correctly identified by the classifier.

\[
\text{Precision} = \frac{TP}{TP + FP} \quad \text{Recall} = \frac{TP}{TP + FN}
\]

Classifying all pairs as matching increases recall at the expense of precision. Classifying only the most similar pairs as matching increases precision at the expense of recall. \( F_1 \) measure [38], defined as the harmonic mean of precision and recall, is often used in record linkage to condense the performance of a classifier down to a single number. \( F_1 \) measure rewards high values for precision and recall while simultaneously penalizing divergence between the values of precision and recall.

\[
F_1 = \frac{2 \times \text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}}
\]

\( F_1 \) measure has been shown to be affected by class imbalance [52]. As such, it is important to keep the class ratio of matches to non-matches consistent when measuring performance. \( F_1 \) measure has also been criticized for weighing precision and recall as equally important. Inspecting the precision/recall curves (PR curves) addresses this. A PR curve visualizes how adjusting the prediction threshold for a given model (which moves recall between 0 and 1) affects precision.

In some cases, the time (and/or computational resources) it takes to train a model
is pertinent. Depending on the availability of computational resources, a model that is significantly slower to train may need to offer a significant increase in performance to be worth it.

This thesis uses $F_1$ measure, PR curves, and training time to evaluate and compare classifiers.

### 2.3 Product Matching

A common phenomenon in the retail world is for references to products to provide limited information. This is most often observed in various forms of print and electronic advertisements (such as flyers or banner ads) where featured products are frequently unaccompanied by a *universal product code* (UPC), *global trade item number* (GTIN), or other unique name/model number. UPC was originally a North American standard developed for supermarkets in the 1970s [45]. GTIN came later, and was designed to be an international standard for product identification and supply chain logistics. GTIN is a numbering superset of UPC [5].

Being able to identify products from advertisements or other listings is of great potential value to a number of stakeholders. On the consumer side, easier identification of products from listings allows products in niche markets to more easily be found. It also enables systems that aggregate products from different sources to provide reliable information, saving consumers time and money. On the retailer side this information can contribute toward improving the consumer experience through relevant advertising, more complete product listings, better product recommendations, and more.

Consumers, advertisers, and aggregators of products all benefit directly when products from different sources can be clearly identified, despite the ill-specified nature of the products from many sources. These benefits are more completely explored in [22].
2.3.1 Challenges

Challenges in identifying products fall into a number of categories. Product information can be incomplete, lacking values for some attributes. Product information can be dirty, where a structured product record does not consistently contain the correct values in the correct fields (e.g., the brand of the product is listed as part of its name, but is missing in the brand field). Product listings are sometime unstructured, where all attributes are mashed together in a single string without separation or explicit labeling of name, brand, price, etc. (e.g., “Kingston 133x high-speed 4GB compact flash card ts4gef133, 21.5 MB per sec data transfer rate, dual-channel support, multi-platform compatibility”).

Beyond the product data itself, challenges also lie in that retailers often organize products into differing, non-interoperable taxonomies. Is a Chromebook a laptop? Should unisex clothing be listed amongst both “male” and “female” clothing, or separated into its own category? Different retailers make different decisions, and these often unclear or inconsistent categorizations and distinctions make it hard to derive information from the meta-structure the product data lies within. Challenges that face modern systems are further elaborated on in [22, 32].

Major areas of focus in this domain include comparing/converting unstructured records to structured records [3, 23], identifying which attributes are most indicative of a true match (and what types of errors are common amongst these attributes) [26], keeping methodologies scalable as the size of the datasets increase [23, 40], and dealing with all the ways in which data can be incomplete, incorrect, and otherwise less than perfect [23, 37].

2.3.2 Prevalence

Product matching is an area of increasing interest, with a large number of new publications since 2018. Where previously publications were spaced further apart allowing new work to reference and build upon previous work, new papers on relatively
similar techniques have been emerging mere months apart [6, 25, 28, 35, 39]. This increase in overlapping work in the field has led to fewer clear comparisons between more recently published techniques and approaches.

Growing interest can be attributed to a number of factors. The passing of time since the dawn of computers has more or less monotonically increased human access to data, both in respect to the data tracked and recorded by companies, and the data freely available for academics to work with [7]. This can be plainly seen in the form of datasets on websites like Kaggle\(^1\), wide-reaching projects like the Web Data Commons\(^2\), and an ever increasing number of product datasets from two different retailers that have been annotated with known matches (e.g., Database Group Leipzig’s Benchmark datasets for entity resolution\(^3\)). This prevalence of relevant and informative data alongside the development of strategies and tools to process big data [7] has made product matching significantly more accessible to academics and corporations alike.

In tandem with the increase in data and data accessibility is an increase in consumer online shopping [48]. Amazon continues to grow year after year, and traditional brick and mortar stores such as Walmart are growing their online foothold. As consumer spending increasingly shifts into the online marketplace, corporate investment into improving online platforms steadily grows. Product matching, with its ability to improve everything from product pages to product recommendations is of ever increasing value [22].

2.3.3 Components

As is true for many modern problems, proposed solutions draw from many different domains of research. Product matching is no exception, drawing upon a number of more established domains.

\(^1\)https://www.kaggle.com/data
\(^2\)http://webdatacommons.org/
\(^3\)https://dbs.uni-leipzig.de/research/projects/object_matching/benchmark_datasets_for_entity_resolution
Record Linkage as a field has established many domain independent flows. Concepts like blocking are important to make many classification techniques computationally viable [1, 8], and a variety of classification techniques have been proposed and tested over what is now more than half a century [15, 16, 32]. The more traditional pair-wise classification approach fails to address transitive closure issues (which arise from multiple conflicting matches arising during linkage) [19, 51]. This problem is addressed by newer techniques such as collective classification [2], or re-framing the problem to make a clustering algorithm applicable [18].

Natural Language Processing (NLP) is a field dedicated to parsing and understanding text. NLP techniques can be applied to multiple subproblems encountered during product matching. One major area of application is parsing unstructured records into structured records, as done in [23, 40]. Very similar to this is resolving conflicting category or attribute names (a linkage task unto itself) from different product listings, which is employed in [40]. NLP techniques such as Term Frequency - Inverse Document Frequency (TF-IDF) have also been applied to represent textual attributes [47], and to generate features for classifiers [37, 41].

Scraping is the process of retrieving data from websites. Data from search engines and e-commerce sites can be valuable both during the linkage process, and to build datasets on which further experimentation can be performed. DEXTER [40] is a recent state of the art web scraping framework built specifically to retrieve product records. Older papers such as [3] often scraped their own datasets to evaluate linkage frameworks. Product aggregation tools such as Google Products and the Bing Product Catalog [23] also need to scour the web in a similar manner in order to function.

Classification is a field that has been dominated by traditional machine learning models over the past several decades. This can be seen in this field, with many established techniques (not just in academia [3, 25, 26] but also running in production in industry [23]) using well-established traditional machine learning techniques. More
recently, *deep learning* has emerged as a technique that, given enough data, can outperform previously state of the art traditional machine learning models. Deep learning techniques offer current state of the art performance, as can be seen in [28, 32, 41, 46]. A review of the design trade-offs that are made during deep learning model design are summarized especially eloquently by Mudgal et al. [32].

### 2.3.4 Contributors

Publications in this area come from groups with a variety of motivations:

**Companies** have used techniques developed in this area to deploy real products running at scale [23]. Multiple companies have also put out publications where they make advances to the state of the art in the field [46, 47]. Occasionally, a company partners with an academic institution to provide otherwise inaccessible data to knowledgeable practitioners [17, 28].

**Academia** plays not only the role of advancing state of the art techniques, but also of making these techniques accessible. An especially notable example of the latter is Magellan [24]: an easy point of entry and extensible framework for record linkage.

### 2.3.5 Machine Learning Techniques

Machine learning is an umbrella term referring to computer algorithms that improve at a task without human input by exploiting data.

Machine learning approaches can be broken into supervised learning and unsupervised learning methods. Supervised learning relies on data containing both inputs and desired outputs of the model being trained. Unsupervised learning uses only inputs, and finds/uses structure in the data to make predictions. In general supervised learning yields superior results, but its requirement for labeled data means it can not be employed for all problems. Recent state-of-the-art product matching techniques use supervised learning [6, 32, 39] due to the availability of labeled product matching data.
Deep learning is a specialized subset of machine learning. Deep learning models are based on artificial neural networks [31] inspired by the biological neural network of the human brain. Deep learning models are more complex than traditional machine learning models. They contain many more parameters to tune during training, and are significantly more expressive. Deep learning models tend to combine feature extraction and classification into a single step, eliminating the need for manual feature engineering [46] and reducing the workload of the practitioner. In contrast, traditional machine learning techniques typically require feature extraction to be performed before classification occurs. This often requires the assistance of a domain expert.

Deep learning models are particularly effective for certain types of complex inputs. Images are a good example of this; deep learning is far better suited to answering the question of whether two images refer to the same product than traditional machine learning techniques [41]. Deep learning is also well suited for problems with data containing large quantities of unstructured text [6, 32]. Deep learning, as compared to traditional machine learning, yields models whose performance plateaus more slowly but at a higher peak if given enough training data.

For all its advantages, there are a number of trade-offs to consider before using deep learning. First, deep learning requires significantly more data than traditional machine learning approaches. When working with a big data, deep learning is likely able to outperform traditional machine learning. For problems with limited data, deep learning might struggle to perform as well as traditional machine learning.

Deep learning models can also take a prohibitively long time to train. To an extent, training times can be sped up using specialized hardware. Graphics processing units (GPUs) are commonly employed, as deep learning algorithms can be implemented to take advantage of massively parallel processing. Even with this hardware, large deep learning models have been known to take hours, days, or longer to train.
2.3.6 State-of-the-Art

State-of-the-art techniques in this area are still quite dependent on the specific characteristics of the data. Recent deep learning models have been primarily targeted at problems with unstructured and/or dirty textual data. The DeepMatcher architecture from the University of Wisconsin-Madison [32] and the more recent entity matching transformer model architectures published by Brunner and Stockinger [6] are both leading examples of this. Both of these groups have made their code open-source, serving as an excellent starting point for practitioners interested in exploring the effectiveness of deep learning on their problem.

Traditional machine learning approaches still achieve state-of-the-art performance on some linkage tasks, especially when the data is cleaner with a simple schema and no lengthy text attributes. The various challenges traditional machine learning techniques face in this area have been recently defined and categorized by Primpeli and Bizer [39]. For the modern practitioner, machine learning models can be easily implemented using powerful user-friendly libraries like Scikit-Learn [34].

2.3.7 Looking Forward

In this field driven by data, it is reasonable to expect data to point to the future. The era of deep learning, in many senses, has just begun. It has proven effective in this field as it has in many others, but there is still seemingly endless potential to apply it in new, interesting, and creative ways. Future product linkage systems will likely start to incorporate data that was previously unusable. Deep learning has opened the door to using image data, but that is likely just the tip of the iceberg. Future systems will likely start to exploit knowledge of consumer behavior: from their interests, to their purchases, to their demographics.

As the state of the art improves, corporate adoption of these techniques will also continue to rise. This will place greater emphasis on engineering concerns, such as using scalable frameworks and measuring performance accurately in a shifting product...
Finally, there are many comparisons that need to be made on the performance of existing systems across diverse sets of data. Recent interest in this area has led to diverging approaches, many of which have not yet been compared/combined. Trying to nail down the strengths and weaknesses of competing approaches remains a major area of focus. In this thesis, we explore the performance of applying individual product linkage methods to industrial data.
Chapter 3

Data

The chapter introduces the product datasets used in this thesis. Publicly available data is presented in Section 3.1. This research also took advantage of some proprietary industrial data, which is presented in Section 3.2.

3.1 Public Data

There are a number of publicly accessible datasets for product matching. These public datasets are valuable to researchers, as they lower the barrier to entry for testing new methodologies and linkage strategies. Using public data also keeps results reproducible, especially for projects with open-source code. The consistent use of the same set of public datasets by different researchers also makes it easier to compare different linkage approaches.

We used three public datasets: Abt/Buy and Amazon/Google (made available by the database group at the University of Leipzig [26]), and Amazon/Walmart (made available by the the University of Wisconsin-Madison’s Magellan Data Repository [13]). We selected these datasets both due to their prevalence in existing research, and their resemblance to the proprietary data presented in Section 3.2.

Attributes that arise in one or more of the public datasets include:

1. name: A concise text snippet describing the product.

2. description/longdesc: A longer written description of the product.
3. **brand/manufacturer**: The name of the company responsible for creating the product.

4. **modelno**: An alpha-numeric sequence usually used to uniquely identify a product from other products created by the same manufacturer.

5. **price**: The cost of the product. This is the only numeric attribute that arises in any of the datasets.

There are a number of additional attributes in the Amazon/Walmart dataset that are not discussed in this thesis. We focus on a subset of the attributes, identified by Primpeli and Bizer [39] as the most relevant for record linkage.

Each of the public datasets comes with a labeled set of known matches, enabling supervised learning techniques. Researchers typically select non-matches for supervised learning by sampling from the remaining product pairs in the Cartesian product. Selection of these non-matches can significantly influence the performance of the classifier, making it challenging to compare different linkage techniques when researchers do not explicitly make available their non-matches. Primpeli and Bizer [39] address this issue by providing standardized sets of non-matches for each dataset, which we use in this work.

An overview of the datasets is provided in Table 3.1. This overview includes the number of records from each retailer (Left/Right in column 2), the number of labeled matches provided (column 3), and the overlapping schema attributes used for matching. A text attribute is listed as a short string (Short Str.) if the average attribute value contains six or fewer words, or a long string (Long Str.) otherwise.

### 3.2 Proprietary Data

One focus of this thesis is to explore how the performance of state-of-the-art methods for product linkage translates to real-world data. Our findings [17] serve to
inform practitioners looking to leverage findings from public data to improve their specific application.

Kinaxis\(^1\) is a company that specializes in supply chain management, promotional planning, and loyalty driven marketing. Kinaxis provided two large product datasets from modern retailers to facilitate this research, shown in the last row of Table 3.1. The agreement with Kinaxis prevents us from disclosing the identities of these two retailers.

The first of the two retailers, henceforth referred to as RetailerA, is a pharmaceutical chain. The second, RetailerB, is a health and beauty retail chain.

The overlapping attributes in the product schemas of these two retailers include UPC/GTIN, product name, brand, and category. All of these attributes are textual. Sample product records are shown in Table 3.2.

<table>
<thead>
<tr>
<th>UPC/GTIN</th>
<th>Product Name</th>
<th>Brand</th>
<th>Category 1</th>
<th>Category 2</th>
<th>Category 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>4001638098595</td>
<td>weleda skin food 75ml</td>
<td>weleda</td>
<td>skincare</td>
<td>skin - facial</td>
<td>facial moisturisers</td>
</tr>
<tr>
<td>85805558420</td>
<td>red door 100ml edt</td>
<td>elizabeth arden</td>
<td>cosmetics</td>
<td>fragrances</td>
<td>fragrance ladies</td>
</tr>
<tr>
<td>3600521650042</td>
<td>l/perf conc t/mat deep be</td>
<td>loreal</td>
<td>cosmetics</td>
<td>mass color cosmetics</td>
<td>mass face make up</td>
</tr>
</tbody>
</table>

Table 3.2: Examples of products from Retailers A/B.

The original product data from these two retailers contained three distinct category attributes. Both retailers followed a structured 3-tier hierarchy, where the three category attributes define each product’s place in a hierarchy with increasing levels of specificity.

\(^1\)https://www.kinaxis.com/
Unfortunately the hierarchies are different, as shown in Table 3.3. Each row in the table shows how a specific product that is sold by both RetailerA and RetailerB is categorized by each retailer. Only five such category pairs are shown in the table, but there are a total of 86 unique category pairs across the 1,324 known matches.

As part of the data pre-processing we concatenated the three category attributes into a single category attribute for both retailers, as the distinctions made at each category level were not consistent between the two schemas.

<table>
<thead>
<tr>
<th>RetailerA Category 1/Category 2/Category 3</th>
<th>RetailerB Category 1/Category 2/Category 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>cosmetics/fragrances/fragrance gift sets mens</td>
<td>fragrance/male gifts/male gifts</td>
</tr>
<tr>
<td>health &amp; beauty care/hair care/shampoo</td>
<td>hair (brushes-combs)/hair mass/shampoo</td>
</tr>
<tr>
<td>cosmetics/derm/derm face care</td>
<td>skincare/skin - premium/premium skin</td>
</tr>
<tr>
<td>other/acquisition - otc/acquisition otc</td>
<td>medicines &amp; vitamins/vitamins &amp; diet supps/energy</td>
</tr>
<tr>
<td>grocery/fresh food/produce nut meats</td>
<td>gift/gifts/girls</td>
</tr>
</tbody>
</table>

Table 3.3: Corresponding categorization examples.

3.2.1 Data Labeling

Unlike the public datasets, the proprietary data did not come with any pre-identified product pairs that are known to be matching. To address this, we performed a database join on the UPC/GTIN attributes and found 1,324 product pairs that shared a value for these attributes.

Products that share a UPC/GTIN value are unambiguously perfect matches. Using this approach to identify perfect matches has been done before [3]. It has the advantages of being both reliable, and requiring little human labour.

Values for UPC/GTIN that differ are unfortunately not enough to definitively classify two products as non-matching. A change in the manufacturing process, a minor ingredient change, or a change in packaging can all be enough to cause a manufacturer to assign a different UPC/GTIN to a product. These differences can be indiscernible to a consumer, and serve to motivate the use of more advanced record linkage techniques despite the presence of this seemingly shared, unique identifier.
Despite differing UPC/GTIN values not proving that two product records do not match, sampling at random from these record pairs is highly unlikely to turn up two matching products. As such, the set of labeled non-matches can safely be constructed from the records not labeled as matching. Even if a true match were to be accidentally labeled as a non-match, the set of non-matches as a whole would still be representative of what non-matching product records typically look like.

By the same token, even though the set of true matches is not exhaustive, it is representative of what matching product records typically look like. Having these two classes in the labeled data, even with minor imperfections, is sufficient for most classification techniques to learn a good decision boundary.

The UPC/GTIN attributes are not made available to any of the classifiers to help predict whether two product records match, as shown in Table 3.1.
Chapter 4

Methodology and Implementation

In this chapter, the methodology followed for linkage and evaluation is presented. Section 4.1 discusses correspondence set generation and blocking. Section 4.2 introduces the profiling of the data. Section 4.3 presents the architecture of the classification models used, and Section 4.4 discusses the evaluation framework used to measure and compare the results.

4.1 Correspondence Set Generation

To train and test a classifier with supervised learning, labeled data is required. The previous Chapter discussed each of the datasets and the labeled true matches that we already have.

As discussed in Section 3.1, the specific selection of non-matches for the correspondence set will affect the classifier’s ability to learn a good decision boundary, and also the measured performance of the classifier. In the interest of consistency, we use the pre-generated correspondence sets published by Primpeli and Bizer [39] for the public datasets.

For the proprietary data, we use the same approach to generate the correspondence set. The one area we deviate on is blocking; an additional blocking strategy utilizing the proprietary data category information is used to discard obvious non-matches. This blocking pass was motivated by the significantly higher number of records in the RetailerA/RetailerB datasets (the Cartesian product of RetailerA/RetailerB contains more than 35 billion product pairs, the largest open source datasets
Amazon/Walmart have a Cartesian product containing fewer than 57 million product pairs). For the public datasets lacking category information, this blocking pass was not possible.

The first step in generating the correspondence set is to select known matches. All known matches are included in the correspondence set, as the total number of known matches is small for all the datasets.

To select non-matching pairs, we turn to the set of all pairs of records in the Cartesian product of the two datasets, less the pairs that are already included as matching. To include all of these pairs in the correspondence set would not be constructive for a number of reasons. First, it would introduce an extremely large class imbalance. For the proprietary data there are roughly 35.5 billion records in the Cartesian product but only 1,324 known matches.

Including all pairs would also immensely increase training time, which would render some of the deep learning methods unusably slow. Including a sufficient number of non-matches to characterize the class is the goal; including more non-matches (especially very obvious non-matches) does not add further value.

As previously discussed, differing UPC/GTIN values is not sufficient to confidently label two records in the proprietary data as non-matches. In the Cartesian product there may be many unidentified pairs of matching records, possibly even more than the 1,324 known matches that are labeled. To feed all of these incorrectly labeled but matching pairs to the classifier would likely obscure where the classification boundary truly should be.

The naive approach to selecting non-matches is to sample from the set of all potential non-matches at random. The main flaw with this approach is that it will overwhelmingly select for non-matches with very little in common. For a machine learning method to learn a good decision boundary, it needs to have many examples from both classes near the decision boundary. Selecting non-matches at random does not achieve this.
To address the issues posed by both class imbalance and randomly sampling for non-matches, the following strategy is used by Primpeli and Bizer [39]:

- To keep class imbalance manageable, limit the ratio of matches to non-matches to 10:1
- To ensure there are non-matches near the decision boundary, deliberately include non-matches that exceed a similarity threshold
- To avoid sampling bias from using this similarity threshold, include one non-match chosen completely at random for every four non-matches selected for their similarity
- To promote diversity in the correspondence set, prevent any one record from either source dataset from occurring in more than ten correspondence set non-matches

It is important to clarify that the 10:1 ratio proposed does not include the randomly sampled non-matches. It is also possible that the target 10:1 ratio may not be achieved if an insufficient number of non-matches pass through the blocking process. This provides context for the correspondence set sizes of each dataset as reported in Table 4.1.

<table>
<thead>
<tr>
<th>Matching Task</th>
<th>Correspondence Set</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Size</td>
<td>#Matches</td>
</tr>
<tr>
<td>Abt/Buy</td>
<td>7,154</td>
<td>1,095</td>
</tr>
<tr>
<td>Amazon/Google</td>
<td>8,440</td>
<td>1,298</td>
</tr>
<tr>
<td>Amazon/Walmart</td>
<td>15,579</td>
<td>1,154</td>
</tr>
<tr>
<td>RetailerA/RetailerB</td>
<td>17,874</td>
<td>1,324</td>
</tr>
</tbody>
</table>

Table 4.1: Correspondence set sizes and distributions

The full correspondence set generation process is described in Algorithm 1. The bulk of this process was originally used in [39], and this particular formulation of the algorithm with category level blocking was originally published in [17]. The remainder
of this section goes into greater detail on the two blocking strategies that are used to select similar non-matches.

4.1.1 Blocking on Product Name Similarity

As previously discussed, two blocking passes are performed. The first pass focuses on the product name attributes.

The specific similarity measurement used in this process is Jaccard index with inner Levenshtein similarity. Jaccard index is a similarity heuristic for sets, and for two sets $A$ and $B$ is defined as the size of the intersection of $A$ and $B$ divided by the size of the union of $A$ and $B$:

\[
\frac{|A \cap B|}{|A \cup B|}
\]

Levenshtein similarity is defined as the number of character level insertions, deletions, and replacements required to transform one string into another. We normalize this to the range $[0, 1]$ by dividing it by the length of the longer string. We then ensure that higher values indicate greater similarity by subtracting this value from one and taking the result.

This similarity measure is computed for the product name attributes, which are tokenized at the word level to form sets. When computing the intersection and the union of the two sets, two elements would normally need to be exactly the same to be considered the same element. For this calculation however, “inner Levenshtein similarity” means that two words are considered to be the same if their Levenshtein similarity exceeds a given threshold. Here, the threshold for Levenshtein similarity is set to 0.7, and we use a threshold on the Jaccard index of 0.2 to discard non-matches that are significantly dissimilar. These thresholds are reasonable for blocking, as the goal at this stage is simply to discard a large number of the highly dissimilar non-matches.
Algorithm 1 Correspondence Set Generation

1: procedure get_correspondence_set($A, B$) \Comment*{A and $B$ are product sets}
2: \hspace{1em} $cs \leftarrow \{\}$ \Comment*{initialize empty correspondence set}
3: \hspace{1em} $cr \leftarrow \{\}$ \Comment*{initialize empty category rules set}
4: \hspace{1em} for $a, b \in A \times B$ do \Comment*{Cartesian product of $A$ and $B$}
5: \hspace{2em} if known_match($a, b$) then
6: \hspace{3em} $cs.add((a, b, True))$
7: \hspace{3em} $cr.add((a.category, b.category))$
8: \hspace{1em} $hnq \leftarrow 10 \times \text{len(correspondence set)}$ \Comment*{hard negative quota}
9: \hspace{1em} $rnq \leftarrow hnq/4$ \Comment*{random negative quota}
10: \hspace{1em} $appearances \leftarrow \{\}$
11: \hspace{1em} for $p \in A \cup B$ do
12: \hspace{2em} $appearances[p] \leftarrow 0$ \Comment*{count record appearances in negative pairs}
13: \hspace{1em} for $a, b \in A \times B$ do
14: \hspace{2em} if known_match($a, b$) then \Comment*{compute on primary attribute(s)}
15: \hspace{3em} continue
16: \hspace{2em} else if $appearances[a] = 10$ or $appearances[b] = 10$ then
17: \hspace{3em} continue
18: \hspace{2em} else if $cr.contains((a.category, b.category)) = False$ then
19: \hspace{3em} continue
20: \hspace{2em} else if relaxed_jaccard($a, b) > 0.2$ then
21: \hspace{3em} $cs.add((a, b, False))$
22: \hspace{3em} $appearances[a] \leftarrow appearances[a] + 1$
23: \hspace{3em} $appearances[b] \leftarrow appearances[b] + 1$
24: \hspace{3em} $hnq \leftarrow hnq - 1$
25: \hspace{3em} if $hnq = 0$ then \Comment*{break}
26: \hspace{2em} break
27: \hspace{1em} for $a, b \in A \times B$ do
28: \hspace{2em} if known_match($a, b$) then
29: \hspace{3em} continue
30: \hspace{2em} else if $appearances[a] = 10$ or $appearances[b] = 10$ then
31: \hspace{3em} continue
32: \hspace{2em} else if $cs.contains((a, b, False)) = False$ then
33: \hspace{3em} $cs.add((a, b, False))$
34: \hspace{3em} $appearances[a] \leftarrow appearances[a] + 1$
35: \hspace{3em} $appearances[b] \leftarrow appearances[b] + 1$
36: \hspace{3em} $rnq \leftarrow rnq - 1$
37: \hspace{3em} if $rnq = 0$ then \Comment*{break}
38: \hspace{2em} break
39: \hspace{1em} return $cs$
For example, when comparing the product names “large green apples” and “red apple large”, the first product name would be tokenized into the set \{“large”, “green”, “apples”\} and the second product name would be tokenized into the set \{“red”, “apple”, “large”\}. The intersection of these two sets would be \{“large”, “apple/apples”\}, as the normalized Levenshtein similarity of apple and apples is \(1 - \frac{1}{6} = 0.83 > 0.7\). The union of the two sets is \{“large”, “green”, “red”, “apple/apples”\}. The similarity of these two sets, as computed by the Jaccard index with inner Levenshtein similarity measure, is \(\frac{2}{4} = 0.5 > 0.2\).

If inner Levenshtein similarity were not employed, Jaccard index would be computed as \(\frac{1}{5} = 0.2 \not< 0.2\).

### 4.1.2 Blocking on Category

The second similarity requirement that is imposed exclusively on the proprietary data uses the product category values. Table 3.3 provides examples of five pairs of categories that arise in the set of known matches.

There are a total of 86 such category pairs that arise across the 1,324 known matches in the proprietary data. This blocking pass discards product pairs whose categories do not occur in this set of 86 category pairs.

For example, products from RetailerA with the category value “cosmetics derm derm face care” paired with products from RetailerB with category value “skincare skin - facial expert skin” are allowed past this blocking phase. Products from RetailerA with the category value “cosmetics fragrances fragrance ladies” paired with products from RetailerB with category value “hair (brushes-combs) hair mass dry shampoo” are not allowed past this blocking phase.

This pass can be easier to conceptualize when thought of as two steps:

1. Learn category blocking rules from the known matches.

2. Apply these rules during blocking to potential candidate pairs.
For the proprietary data 108,774,763 candidate pairs remain after the product name blocking pass. The category blocking pass reduces the number of candidate pairs to 13,951,157.

4.2 Data Profiling

Primpeli and Bizer [39] proposed a method of profiling data for record linkage across five dimensions. These dimensions help illuminate, for each matching task, what common challenges are likely to arise. By performing this profiling, we are both able to compare the different datasets we are working with, and make predictions about the performance of traditional machine learning and deep learning methods on our data.

The five profiling dimensions are Schema Complexity (SC), Sparsity (SP), Textuality (TX), Development Set Size (DS), and Corner Cases (CC). All dimensions are computed on the correspondence set.

**Schema Complexity** is defined as the number of shared attributes that are valuable for the matching task. A higher schema complexity can imply the need to define a more complex underlying relationship for what constitutes a match. A more complex schema is often still preferable to the same information stored more homogeneously in fewer attributes. A record linkage problem is made easier when there are no composite attributes that could be clearly broken apart into more atomic attributes. So long as that is the case, a lower schema complexity usually indicates that a solution can be reached with a simpler model.

**Sparsity** measured the quality of the data, specifically by examining the number of missing attribute values. It is computed as the percentage of non-missing attribute values across all attributes of all records in the correspondence set. Fewer missing values implies higher quality data, which is usually a predictor of better classifier performance.
Textuality is the average number of white-space delimited words in the primary textual attribute describing a record. Long text strings might imply that the information could have better been broken into multiple, more specific attributes. Long product names also have the issue that the information within might be presented in a different order. One retailer may follow the standard of product size, brand, then name all within the product name attribute, while another retailer might store product brand, name, then size within their version of the product name attribute. Without a significant amount of cleaning the data, and possibly splitting this type of attribute into multiple attributes, any similarity measures used would ideally have the capacity to account for these types of discrepancies.

Development Set Size is defined as the number of labeled matches and non-matches in the training data. If the development set size is too small, it can be hard to learn a good decision boundary due to the underrepresentation of what a pair of matching records can look like.

Corner Cases aims to quantify how many matching record pairs look like non-matching record pairs, and how many non-matching record pairs look like matching record pairs. More corner cases will lead to a greater number of False Positives and False Negatives, which in turn negatively affects precision, recall, and $F_1$ measure. If all of the matches and non-matches can be easily cleanly separated, that would be the best case scenario.

To compute a value for this dimension, we start with the feature vectors computed in Section 4.3.1. For each product pair, we compute a single similarity value by averaging all the elements in its feature vector. We then choose a threshold, which when used to make predictions using this single similarity value, maximizes the resulting $F_1$ score. The predictions generated by this model are then used to compute this dimension as:

\[
\frac{FP + FN}{\text{#matching\_pairs}}
\]
The results of profiling each of the four datasets across these five dimensions are shown in Table 4.2.

<table>
<thead>
<tr>
<th>Matching Task</th>
<th>Profiling Dimensions</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>SC</td>
</tr>
<tr>
<td>Abt/Buy</td>
<td>3</td>
</tr>
<tr>
<td>Amazon/Google</td>
<td>3</td>
</tr>
<tr>
<td>Amazon/Walmart</td>
<td>5</td>
</tr>
<tr>
<td>RetailerA/RetailerB</td>
<td>3</td>
</tr>
</tbody>
</table>

Table 4.2: Profiling results

In Primpeli and Bizer’s work [39], the results of this profiling are shown to be effective at predicting the effectiveness of traditional machine learning techniques. The public datasets all have relatively high textuality, which when combined with many corner cases can pose a challenge for state-of-the-art traditional machine learning methods. Abt/Buy and Amazon/Google are the two datasets that fall most into this category of high textuality and many corner cases, which makes these two datasets the best candidates for deep learning approaches that rely on embedding-based matching methods [6, 32].

Amazon/Walmart has fewer corner cases, making it an easier problem to solve with traditional machine learning. The proprietary data looks to be the easiest to address with traditional machine learning, as it combines a simple schema with low textuality and very few corner cases.

4.3 Classifiers

To perform classification, we use both traditional machine learning and deep learning model architectures. We selected specific model architectures based on their recent state-of-the-art performance on product matching tasks, coupled with the availability of open-source model implementations [6, 32, 39].

For every architecture, pairwise classification is performed. Each model takes as
input a pair of product records \((a, b)\), \(a \in A\), \(b \in B\), and predicts as output one of two labels: “match” or “non-match”. While the input to the models is consistent, different architectures represent the products and model their similarity differently during the classification process.

4.3.1 Traditional Machine Learning Models

Traditional machine learning is characterized by a relatively simple model structure, and a lesser need for large amounts of training data when compared to deep learning approaches. In the field of record linkage, especially for datasets that are clean and structured, traditional machine learning approaches can still yield state-of-the-art results.

We consider four classification models in this category: Decision Trees [42], Random Forests [20], Logistic Regression classifiers [12], and Support Vector Machines (SVM) with a radial basis function (RBF) kernel [11]. A random forest is an ensemble model built using multiple decision trees, and as such random forest models generally outperform decision trees. Each of these four architectures has a history of being successfully used in other record linkage tasks [9, 15, 54].

To represent a product pair \((a, b)\) to these models, we compute its feature vector \(\phi_{(a,b)}\). Each dimension in \(\phi_{(a,b)}\) corresponds to a specific comparison made using a specific attribute or set of attributes.

For short string attributes (as shown in Table 3.1), we compute Levenshtein similarity, Jaccard similarity at the word level, Jaccard similarity at the word level with inner Levenshtein, Jaccard similarity at the token level, exact similarity, and containment similarity. For long string attributes, we compute Levenshtein similarity, Jaccard similarity at the word level, Jaccard similarity at the word level with inner Levenshtein, exact similarity, and containment similarity, and TF-IDF cosine similarity. For numeric attributes, we compute exact similarity and absolute difference. These similarity measures are defined in Appendix A.
<table>
<thead>
<tr>
<th>RetailerA Record (A₁)</th>
<th>RetailerB Record (B₁)</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>name</strong></td>
<td><strong>name</strong></td>
</tr>
<tr>
<td>nyx butter lipstick</td>
<td>nyx butter gloss -</td>
</tr>
<tr>
<td>hubba bubba 1 un</td>
<td>eclair</td>
</tr>
<tr>
<td>cosmetics mass color</td>
<td>cosmetics &amp; bronzing</td>
</tr>
<tr>
<td>cosmetics mass lip</td>
<td>cosmetics lips</td>
</tr>
<tr>
<td>color</td>
<td>nyx</td>
</tr>
</tbody>
</table>

| **category**           |                         |
| cosmetics mass color   |                         |
| cosmetics mass lip     |                         |
| color                  |                         |

| **brand**              |                         |
| nyx                    |                         |

<table>
<thead>
<tr>
<th>Feature Vector for A₁/B₁</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>Levenshtein</td>
</tr>
<tr>
<td>Jaccard</td>
<td>0.200</td>
</tr>
<tr>
<td>Jaccard (inner Levenshtein)</td>
<td>0.200</td>
</tr>
<tr>
<td>Exact Match</td>
<td>0.000</td>
</tr>
<tr>
<td>Containment</td>
<td>0.400</td>
</tr>
<tr>
<td>Cosine TF-IDF</td>
<td>0.269</td>
</tr>
<tr>
<td>category</td>
<td>Levenshtein</td>
</tr>
<tr>
<td>Jaccard</td>
<td>0.143</td>
</tr>
<tr>
<td>Jaccard (inner Levenshtein)</td>
<td>0.333</td>
</tr>
<tr>
<td>Exact Match</td>
<td>0.000</td>
</tr>
<tr>
<td>Containment</td>
<td>0.250</td>
</tr>
<tr>
<td>Cosine TF-IDF</td>
<td>0.248</td>
</tr>
<tr>
<td>brand</td>
<td>Levenshtein</td>
</tr>
<tr>
<td>Jaccard</td>
<td>1.000</td>
</tr>
<tr>
<td>Jaccard (inner Levenshtein)</td>
<td>1.000</td>
</tr>
<tr>
<td>Jaccard (token level)</td>
<td>1.000</td>
</tr>
<tr>
<td>Exact Match</td>
<td>1.000</td>
</tr>
<tr>
<td>Containment</td>
<td>1.000</td>
</tr>
<tr>
<td>name + category + brand</td>
<td>Cosine TF-IDF</td>
</tr>
</tbody>
</table>

Figure 4.1: A feature vector example
For each record, we also compute an attribute that is all the other attributes concatenated together. We include TF-IDF cosine similarity on this attribute as the final dimension in the feature vector.

All similarity measures return a value between 0 and 1, where a higher value indicates higher measured similarity. For exact similarity, two identical attribute values receive a 1, and two differing values would receive a 0. For all other similarity measures, two identical attribute values receive a 1, and non-matching values receive a score in the range $[0, 1)$.

We use the code published by Primpeli and Bizer\(^1\) to compute these feature vectors. See Figure 4.1 for an example of a feature vector from the RetailerA/RetailerB dataset.

### 4.3.2 DeepMatcher Models

DeepMatcher is a Python package\(^2\) designed and published by Mudgal et al. [32]. This package provides multiple architectures for employing deep learning to a record linkage problem. In this thesis, we specifically focus on the DeepMatcher hybrid model. The hybrid model is the most expressive, the most expensive to train, and yields the best results of all the DeepMatcher model architectures [32].

The hybrid model combines a sequence aware model (a bidirectional RNN) with an attention mechanism. To represent product records to the model, 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. We test three different methods to generate embeddings:

- **fasttext.en.bin** uses character level embeddings released by fastText [4]
- **glove.6B.300d** uses word level embeddings trained on Wiki + Gigaword with uncased GloVe [36]

\(^1\)https://github.com/wbsg-uni-mannheim/EntityMatchingTaskProfiler  
\(^2\)https://github.com/anhaidgroup/deepmatcher
• **glove.42B.300d** uses word level embeddings trained on Common Crawl with uncased GloVe [36]

Converting words to embeddings is a common approach taken by many modern NLP tasks. Pre-trained embeddings are valuable, as they allow complex relationships between words to be exploited in the context of a specific task that lacks a large amount of textual data. In general character level embeddings are better at dealing with unusual words (e.g., words that occur due to misspellings or abbreviations) than word level embeddings [32]. Some product names contain abbreviations (e.g., “mngmt software”) which should benefit from use of character level embeddings.

The hybrid DeepMatcher constructs similarity vectors from the sequences of embeddings using a bidirectional recurrent neural network (RNN) with an attention mechanism. The advantages of this approach are twofold. The RNN provides sequence awareness, which allows the model to learn complex relationships between the order and semantic significance of a single attribute value’s embedding sequence. This is useful for instances where a series of sequential words have a meaning that is very different from any individual word. The attention mechanism allows the model to compute a soft alignment between two embedding sequences and then perform token by token comparisons.

Due to this complex model structure, the DeepMatcher hybrid model is good at dealing with long written descriptions that communicate the same information in a different order, often with different words. An example of where this technique should significantly outperform the traditional machine learning similarity measures can be found when looking at these two product descriptions taken from the Amazon/Google data:

> “peachtree premium accounting 2007 accountants’ edition is the powerful multi-user ready accounting solution accountants need to service their peachtree clients more effectively and efficiently. create open modify and save client files from any member of the peachtree 2007 line plus handle complex tasks such as company consolidation compensation management
and comprehensive budgeting. converts quicken and quickbooks data to peachtree (from 1999-2005) restore backed-up client data plus web transactions and customized forms e-mail forms and reports – get invoices out faster save money on postage and avoid printing hassles quickly review all your clients’ journal entries requires microsoft outlook 2000 2002 or 2003; outlook express 5 or 6; or other mapi compliant e-mail programs included tax forms - 940 941 941b 943 945 w-2 w-3 and over 180 state forms”


These two strings refer to the same product, however they are very different lengths and each description contains multiple words that do not occur in the other. The word embedding comparison technique used by the DeepMatcher hybrid model has significantly more potential to evaluate these two descriptions as similar than the more basic text similarity measures used by the traditional machine learning approaches.

The final stage of the DeepMatcher model uses a neural network, which takes the concatenation of the attribute similarity vectors as input, and outputs a matching prediction.

4.3.3 Entity Matching Transformer Models

Transformer model architectures have achieved state-of-the-art results for many NLP problems recently, both with respect to model performance and training time [50]. Transformer models have recently started seeing use in the record linkage domain [6, 35], and are most valuable when tackling record linkage tasks that contain dirty or unstructured data.

Transformer models have a greater inherent capacity for parallelism than RNNs, which leads to lower training times. As with DeepMatcher, product records are converted to embedding sequences prior to the computation of similarity vectors.
Instead of using a bidirectional recurrent neural network (RNN) with an attention mechanism like DeepMatcher, the entity matching transformer (EMT) models use a two different transformer architectures configured for record linkage:

- **RoBERTa** has distinguished itself via state of the art results on a number of NLP tasks [29]

- **DistilBERT** features shorter training times and a reduced model size [43]

We use the code³ published by Brunner and Stockinger [6]. The implementation of these models is structured to accept a single string for each product record as input. Following a similar approach to [6], we concatenate the product name and description for Abt/Buy, we use only the product name for Amazon/Google, we use only the product title for Amazon/Walmart, and we use only the product name for RetailerA/RetailerB.

This framework does not use pre-trained embeddings. It trains sub-word level embeddings for RoBERTa and byte-level embeddings for DistilBERT. This framework does start with a pre-trained RoBERTa and a pre-trained DistilBERT model. This framework is less expressive than the DeepMatcher hybrid model. It is similar to the DeepMatcher attention model, with the advantage that it uses newer transformer architectures.

### 4.4 Evaluation Framework

From the data profiling, we have generated several predictions as to the relative applicability of the various classification techniques to each dataset. A consistent evaluation framework is important, as it allows us to compare the results to validate or invalidate these hypotheses. Measuring the performance of each model is important, but so too is measuring and comparing runtime and computational requirements (i.e.,

³https://github.com/brunnurs/entity-matching-transformer
whether the model can be efficiently trained with just a CPU, or requires a system with a GPU).

In addition to the numeric results (precision, recall, $F_1$ measure, and runtime) we also generate PR curves, allowing the tradeoff between these two measures to be evaluated over the full range of precision/recall.

### 4.4.1 Experimental Setup

Previous experiments on this data using these classifiers have been performed on a single train/validation/test set split [6, 32, 39]. The framework used in this thesis is different from these previous works in that we use ten-fold cross validation. We split the data into 10 stratified folds, and repeat each experiment ten times. For each experiment, we select one of the folds to be the test set, one fold to be the validation set, and merge the remaining eight folds to form the training set. We do this in such a way as to cause each fold to serve as the test set once and the validation set once, as shown in Figure 4.2.

We use the code published by Primpeli and Bizer [39] to generate the feature vectors for traditional machine learning, and Scikit-Learn to implement the traditional machine learning models. We use the published code for DeepMatcher [32] and the Entity Matching Transformer [6] for the two deep learning methods.

<table>
<thead>
<tr>
<th>Model</th>
<th>Parameter</th>
<th>Values</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Decision Tree</strong></td>
<td>max. depth</td>
<td>[1, 3, 5]</td>
</tr>
<tr>
<td></td>
<td>min. leaf size</td>
<td>[5, 10, None]</td>
</tr>
<tr>
<td><strong>Random Forest</strong></td>
<td>estimators</td>
<td>[10, 100, 500]</td>
</tr>
<tr>
<td></td>
<td>max. depth</td>
<td>[1, 3, 5]</td>
</tr>
<tr>
<td></td>
<td>min. leaf size</td>
<td>[5, 10, None]</td>
</tr>
<tr>
<td><strong>Logistic Regression</strong></td>
<td>C penalty</td>
<td>logspace(-2, 5, 10)</td>
</tr>
<tr>
<td><strong>SVM</strong></td>
<td>C gamma</td>
<td>logspace(-2, 5, 10)</td>
</tr>
</tbody>
</table>

Table 4.3: Hyperparameter values

Hyperparameter tuning for traditional machine learning is performed via a grid search. The best hyperparameters for each model are selected using the average $F_1$
score on the validation set. Only the average training time for the best hyperparameters is reported in Chapter 5. The sets of hyperparameter values considered is shown in Table 4.3.

Deep learning models are trained for 15 epochs. The validation set is evaluated on the model at the end of each training epoch. The final model evaluated with the test set is determined by when the highest performance on the validation set is achieved. All deep learning parameters, including the number of training epochs, are set as described in their original implementations [6, 32].

All numeric results in Chapter 5 represent the average test set performance on
4.4.2 Precision Recall Curves

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 4.3. This figure shows how the single representative precision recall curves for each model type shown in Section 5.1 relate to the ten precision recall curves from the 10-fold cross validation.

![Representative PR Curve Creation](image)

Figure 4.3: Precision recall curve creation

The one curve shown as part of the experimental results should not be interpreted as the performance of a specific model, but rather as the expected precision/recall of
the model building process.
Chapter 5

Results

In this chapter, we report all of the experimental results for each classification system paired with each matching problem. Precision, recall, $F_1$ score, training times, and precision recall curves are presented and discussed in Section 5.1. Section 5.2 discusses classification examples when simulating deploying for the best model on the proprietary data. Section 5.3 discusses use cases for these classifiers.

5.1 Classifier Performance

In this section, we report the performance for all classifier/dataset combinations. Precision and recall results are reported in Table 5.1, and $F_1$ score and training time are reported in Table 5.2. For $F_1$ score, bold font highlights the best performance for each dataset and underlined font highlights the best performance for each model architecture.

<table>
<thead>
<tr>
<th>Model</th>
<th>Abt/Buy P</th>
<th>Abt/Buy R</th>
<th>Amazon/Google P</th>
<th>Amazon/Google R</th>
<th>Amazon/Walmart P</th>
<th>Amazon/Walmart R</th>
<th>RetailerA/RetailerB P</th>
<th>RetailerA/RetailerB R</th>
</tr>
</thead>
<tbody>
<tr>
<td>Decision Tree</td>
<td>0.86</td>
<td>0.72</td>
<td>0.78</td>
<td>0.67</td>
<td>0.93</td>
<td>0.90</td>
<td>0.97</td>
<td>1.00</td>
</tr>
<tr>
<td>Random Forest</td>
<td>0.89</td>
<td>0.77</td>
<td>0.81</td>
<td>0.72</td>
<td>0.96</td>
<td>0.93</td>
<td>0.97</td>
<td>1.00</td>
</tr>
<tr>
<td>Logistic Regression</td>
<td>0.86</td>
<td>0.72</td>
<td>0.77</td>
<td>0.61</td>
<td>0.92</td>
<td>0.88</td>
<td>0.97</td>
<td>1.00</td>
</tr>
<tr>
<td>SVM</td>
<td>0.90</td>
<td>0.72</td>
<td>0.81</td>
<td>0.61</td>
<td>0.96</td>
<td>0.88</td>
<td>0.97</td>
<td>1.00</td>
</tr>
<tr>
<td>fasttext.en.bin</td>
<td>0.74</td>
<td>0.75</td>
<td>0.80</td>
<td>0.83</td>
<td>0.92</td>
<td>0.93</td>
<td>0.85</td>
<td>0.85</td>
</tr>
<tr>
<td>glove.6B.300d</td>
<td>0.50</td>
<td>0.63</td>
<td>0.70</td>
<td>0.75</td>
<td>0.74</td>
<td>0.77</td>
<td>0.81</td>
<td>0.85</td>
</tr>
<tr>
<td>glove.42B.300d</td>
<td>0.50</td>
<td>0.64</td>
<td>0.73</td>
<td>0.82</td>
<td>0.79</td>
<td>0.84</td>
<td>0.83</td>
<td>0.84</td>
</tr>
<tr>
<td>RoBERTa</td>
<td>0.75</td>
<td>0.60</td>
<td>0.82</td>
<td>0.87</td>
<td>0.93</td>
<td>0.90</td>
<td>0.89</td>
<td>0.84</td>
</tr>
<tr>
<td>DistilBERT</td>
<td>0.73</td>
<td>0.64</td>
<td>0.81</td>
<td>0.83</td>
<td>0.91</td>
<td>0.87</td>
<td>0.89</td>
<td>0.84</td>
</tr>
</tbody>
</table>

Table 5.1: Average precision and recall
Table 5.2: Average training time and $F_1$ score

<table>
<thead>
<tr>
<th>Model</th>
<th>Abt/Buy Time</th>
<th>$F_1$</th>
<th>Amazon/Google Time</th>
<th>$F_1$</th>
<th>Amazon/Walmart Time</th>
<th>$F_1$</th>
<th>RetailerA/RetailerB Time</th>
<th>$F_1$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Decision Tree</td>
<td>0.036s</td>
<td>0.78</td>
<td>0.031s</td>
<td>0.72</td>
<td>0.197s</td>
<td>0.92</td>
<td>0.012s</td>
<td>0.98</td>
</tr>
<tr>
<td>Random Forest</td>
<td>1.620s</td>
<td><strong>0.82</strong></td>
<td>1.801s</td>
<td>0.76</td>
<td>0.445s</td>
<td><strong>0.94</strong></td>
<td>0.037s</td>
<td><strong>0.98</strong></td>
</tr>
<tr>
<td>Logistic Regression</td>
<td>0.238s</td>
<td>0.79</td>
<td>1.770s</td>
<td>0.68</td>
<td>4.735s</td>
<td>0.90</td>
<td>18.079s</td>
<td><strong>0.98</strong></td>
</tr>
<tr>
<td>SVM</td>
<td>0.600s</td>
<td>0.81</td>
<td>4.508s</td>
<td>0.73</td>
<td>0.714s</td>
<td>0.93</td>
<td>5.051s</td>
<td><strong>0.98</strong></td>
</tr>
<tr>
<td>fasttext.en.bin</td>
<td>8m 45s</td>
<td>0.74</td>
<td>2h 03m</td>
<td>0.81</td>
<td>1h 54m</td>
<td>0.93</td>
<td>24m 06s</td>
<td>0.85</td>
</tr>
<tr>
<td>glove.6B.300d</td>
<td>10m 37s</td>
<td>0.55</td>
<td>1h 50m</td>
<td>0.73</td>
<td>1h 54m</td>
<td>0.75</td>
<td>22m 37s</td>
<td>0.83</td>
</tr>
<tr>
<td>glove.42B.300d</td>
<td>11m 38s</td>
<td>0.56</td>
<td>2h 00m</td>
<td>0.77</td>
<td>1h 55m</td>
<td>0.82</td>
<td>22m 45s</td>
<td>0.83</td>
</tr>
<tr>
<td>RoBERTa</td>
<td>26m 27s</td>
<td>0.65</td>
<td>30m 57s</td>
<td><strong>0.84</strong></td>
<td>56m 50s</td>
<td>0.92</td>
<td>1h 05m</td>
<td>0.86</td>
</tr>
<tr>
<td>DistilBERT</td>
<td>13m 29s</td>
<td>0.68</td>
<td>15m 37s</td>
<td>0.82</td>
<td>28m 42s</td>
<td><strong>0.89</strong></td>
<td>33m 04s</td>
<td>0.86</td>
</tr>
</tbody>
</table>

The results in Table 5.2 show clearly that traditional machine learning techniques exhibit extremely strong performance on the proprietary data. The random forest models demonstrate the highest average performance of the traditional machine learning methods, beating out all the deep learning models except for Amazon/Google.

Given Amazon/Google’s high textuality as reported in Table 3.1, this is not a surprising exception. Amazon/Google products have lengthy text descriptions, which deep learning techniques tend to generate better comparisons for. The remaining datasets, with their shorter text fields, were all stronger candidates for the traditional machine learning model architectures.

In general, employing a random forest model for this type of problem would appear to be an excellent starting point. Depending on the initial performance that is achieved and the extent to which the data is dirty and/or unstructured, deep learning may be able to improve the classification performance. Figure 5.1 serves to further demonstrate this point, with the random forest models outperforming all other methods over the entire range of precision recall values on Abt/Buy, Amazon/Walmart and RetailerA/RetailerB datasets.

For Amazon/Google, the best performing model at different recall thresholds varies. Depending on the application for the model, it might be reasonable to define a desired recall threshold and pick the best model accordingly.
Figure 5.1: Precision recall curves for the best models

Figure 5.1 only shows the top performing model(s) for each category of architectures (i.e., Random Forest for traditional machine learning, fasttext.en.bin for DeepMatcher and, RoBERTa and DistilBERT for the Entity Matching Transformer models). The precision recall curves for all models are reported in Appendix B.

Deep learning models can often improve from additional labeled training data long after traditional machine learning models stop seeing improvement in performance. It might be useful to consider deep learning models more seriously for product matching problems with significantly higher quantities of labeled data that can be used for training.

Table 5.2 illustrates a significant rift in training time between the traditional machine learning and the deep learning approaches. The relative ease with which
an traditional machine learning system can be implemented makes it an attractive starting place. In addition to the slower training times, the deep learning models were all trained on a system with a NVIDIA V100 Tensor Core GPU. Without the GPU, training times for the deep learning models could easily have been several orders of magnitude slower. The traditional machine learning models required only a CPU to be trained efficiently.

5.2 Simulating Deployment

The goal of this type of classification system is to eventually be deployed on unseen product pairs. The results in the previous section were all in the context of the testing data. In this section, we discuss how the classifier for RetailerA/RetailerB performs on unseen data.

To do this, we select 10,000 unseen pairs of records and we use the random forest model to classify them as a “match” or a “non-match”. None of these pairs are part of the correspondence set used for training/testing.

Given that these pairs all do not match on their UPC/GTIN attribute, had they been included in the correspondence set they would have been labeled as a “non-match”. As discussed in Section 4.1, there may still be pairs of products within these pairs which for many applications, we would want to identify as matching.

Of these pairs, 500 were chosen for their high similarity. For all RetailerA/RetailerB product pairs that pass the blocking passes in Sections 4.1.1 and 4.1.2, Levenshtein similarity $Lev(a, b)$ was computed for the product name attribute. The 500 pairs were sampled at random from the subset of pairs with $Lev(a, b) > 0.8$. The remaining 9,500 pairs were also required to make it through the two blocking passes, but were otherwise chosen completely at random.

Out of the 10,000 product pairs, only 6 were classified as a “match”. This is already a promising result, as the majority of pairs in the Cartesian product of two
sets will be non-matches. Since we do not have labels for these product pairs, our analysis in this section is based on manual inspection and qualitative assessment of the predicted labels. This allows us to better understand where the classification system performs well and where it could be improved or changed to address application specific demands.

Table 5.3: Examples of unseen pairs classified as “matches”

Table 5.3 shows the six product pairs that are predicted as matches. Looking at them, it would seem that these predictions are correct. With the exception of pair 3, all product pair names are identical save for a space before “ml”. Product pairs 3 and 6 do not match perfectly on the brand attribute, although both pairs of brands share at least one word.

Product pairs 3, 4, and 5 serve to demonstrate the lack of a one-to-one mapping for the categorization hierarchies of RetailerA and RetailerB. The presence of these three product pairs justifies the complexity of the blocking strategy in Section 4.1.2.

Table 5.4: Examples of unseen pairs classified as “non-matches”

Table 5.4 shows examples of some of the pairs predicted as “non-matches”. For the most part, the pairs predicted as “non-matches” can be trivially confirmed as correctly classified via brief human inspection. These three pairs are handpicked
from the non-matches as interesting examples due to their high similarity.

Pairs 7 and 8 appear to be correctly classified. The two products in Pair 7 look like two different shampoos given the “tropical” vs. “oriental” descriptors. Products in Pair 8 seem to be the same cream, but are differently sized.

Pair 9 is an interesting example, as the product names are almost identical save for a one letter indicator. Pair 9 is a potential false negative. Such false negatives can be addressed by talking with the retailers to better understand the significance of their notation.

5.3 Use Cases

This type of classification model can be deployed to a variety of very different real-world use cases. In this section, we discuss some of these use cases and provide additional discussion around some of additional considerations that may arise.

One use case that has already seen significant real-world use is an online product aggregation service. Kannan et al. [23] discussed the workings of this type of system designed by Bing. Google has also provided a service of this nature since 2002 known originally as Froogle, later rebranded as Google Products, then Google Product Search, and most recently as Google Shopping. The Amazon interface also provides a similar service. More specialized services like PCPartPicker\(^1\) also provide this service for niche markets.

Product aggregation services vary in their exact use case. The Amazon interface, when a specific product is selected, will show the consumer different prices, qualities (i.e., new vs. used), and shipping options. This requires Amazon to know when two different merchants are selling the same product. Similarly, PCPartPicker aggregates for consumers the current prices of all merchants selling a given computer part in easy-to-view form.

\(^1\)https://pcpartpicker.com/
Sometimes, a service may wish to feature a section under each product listing that features similar products. One approach to this is to use consumer browsing data, like Amazon’s “Customers who viewed this item also viewed...” section. Another approach could be to cluster the similarity vectors computed for the classification models to identify similar products.

For some products, retailers will allow consumers to switch between different sizes or colors. When a company like Amazon does this, and the different colors are offered by different sellers, this also would have required product matching behind the scenes. When using a classification system for this type of application, product pairs that differ in size, color, etc. should be labeled as matching when training the classifier.

Another application would be a retailer looking for duplicate products in their own stock. This might be important, as any analysis of product transaction history should merge the transactions associated with two identical products. A manufacturer changing their UPC could easily result in a retailer creating a second database entry for the same product.

Kianxis often works with retailers who have large databases of products and transactions, and wish to apply machine learning or deep learning techniques to optimize the timing of promotions, create promotions targeted at individual consumers, forecast demand, etc. These retailers often have not been collecting data with this particular use case in mind, which creates a situation where the need for this type of deduplication has arisen organically.

A final application for product matching arises when predicting demand for products that are new to market. Since historic sales history is often a strong prediction of future demand, new products with no transaction history pose a challenge.

Product matching could be used to find similar products to the new products. The sales history of the similar products could then be substituted in for the new products when predicting future demand. The definition of a match is very broad for
this use case, as two existing products could be labeled as matching if the historic demand for the two products has been nearly identical.
Chapter 6

Conclusions

In this thesis, we proposed a framework for investigating state-of-the-art record linkage techniques on product matching tasks. This framework enabled meaningful comparisons to be made between the efficacy and practicality of traditional machine learning and deep learning techniques on both public product matching benchmarks, and real-world modern day industrial product data.

We showed that traditional machine learning approaches continue to produce state-of-the-art results, especially for clean, structured datasets without long textual attributes. We showed that this property holds when training models for real-world data.

In addition to the standard $F_1$ score that is reported to compare record linkage models, we generated and analysed precision recall curves for each model architecture. We used this to demonstrate how a varying required threshold for recall affects model selection.

6.1 Future Work

The natural next step for this research is to apply this type of classification system to a real-world problem, such as the demand series forecasting application. The extent to which these classifiers add value to that type of application will provide an additional performance metric for the performance of the classifiers.

While the measured performance of the classifiers on the real-world data is very high, there are still a number of steps that could be taken to improve further:
Further data cleaning steps for uniform attribute values could include removing brand from product name, separating size into a separate attribute, and speaking to retailers about their specific notational standards.

Enhancing individual product listings by scraping the web for duplicate product listings and incorporating additional product attributes.

Exploring transaction data as a means to creating additional similarity features.

As the number of products on the internet continues to grow, and retailers increasingly turn to machine learning techniques to leverage their data, product record linkage systems will continue to gain sophistication and relevance in the modern retail world.
Bibliography


Appendix A

Appendix

This appendix contains the definitions for the similarity measures used to compute the traditional machine learning similarity vectors. All measures return a similarity score in the range $[0, 1]$, with values closer to 1 indicating greater similarity.

A.1 Levenshtein Similarity

Levenshtein distance is defined as the minimum number of single character insertions, deletions, or substitutions required to transform one string into another. This value is then normalized to the range $[0, 1]$ by dividing by the number of characters in the longer string, and is then subtracted from 1 to enforce that similarity score values closer to 1 indicate greater similarity.

A.2 Jaccard Similarity

The Jaccard similarity of two sets $A$ and $B$ is computed as:

$$\frac{|A \cap B|}{|A \cup B|}$$

Jaccard similarity can be computed at the character or word level by tokenizing strings into sets at the character or the word level. For example, the string “Green Apples” can be tokenized into the character set \{“g”, “r”, “e”, “n”, “a”, “p”, “l”, “s”\} or the word set \{“green”, “apples”\}.
A.3 Jaccard Similarity with Inner Levenshtein

This similarity measure is defined in Section 4.1.1.

A.4 Exact Similarity

This similarity measure returns 1 if the two values being compared are identical, and 0 otherwise.

A.5 Containment Similarity

The containment similarity of two sets $A$ and $B$ is computed as:

$$\frac{|A \cap B|}{\min(|A|, |B|)}$$

Much like Jaccard similarity, containment similarity can be computed at the character or word level by tokenizing strings into sets at the character or the word level. If either string being compared contain multiple words, we tokenize at the word level. If both strings contain a single word, we tokenize at the character level.

A.6 Absolute Difference

This similarity measure is only computed for numeric attributes. For two numeric values $a$ and $b$ it is computed as $|a - b|$. This similarity measure is normalized once all the values have been computed. It is also adjusted so that values closer to 1 indicate greater similarity.

Consider the following pairs of values: $\{(0, 1), (1, 4), (3, 1)\}$. First the absolute differences are computed, yielding $D = [1, 3, 2]$. Then these values are normalized by first subtracting $\min(D) = 1$ from each value yielding $D = [0, 2, 1]$, and then dividing
each value by $\max(D) = 2$ yielding $D = [0, 1, 0.5]$. Finally, each value is subtracted from 1, yielding $[1, 0, 0.5]$.

This final feature vector $[1, 0, 0.5]$ indicates that the most similar starting pair of values (where the absolute difference feature is closest to 1) is $(0, 1)$, and the least similar pair of starting values (where the absolute difference feature is closest to 0) is $(1, 4)$.

A.7 TF-IDF Cosine Similarity

Term frequency-inverse document frequency is an established technique in NLP to reflect how important each word is to a given document in the context of a collection of documents. “Term frequency” refers to how often a term appears in a given document, “document frequency” refers to how often the term appears in documents in the collection.

All attribute values together are treated as the collection of documents when computing this feature. We use the `sklearn.feature_extraction.text.TfidfVectorizer` module to compute the TF-IDF vector for each individual attribute value.

Given TF-IDF vectors $A$ and $B$ for two different attribute values, we compute a single numeric similarity score using cosine similarity:

$$\frac{A \cdot B}{\|A\| \|B\|}$$

All cosine similarity values are rescaled to the range $[0, 1]$.

---

Appendix B

Appendix

This appendix contains the precision recall curves for all model/dataset combinations.

Figure B.1: Precision recall curves for all ML models
Figure B.2: Precision recall curves for all DeepMatcher models
Figure B.3: Precision recall curves for all EMT models