VIETNAM NATIONAL UNIVERSITY
HO CHI MINH UNIVERSITY OF TECHNOLOGY
FACULTY OF COMPUTER SCIENCE AND ENGINEERING

DATA MINING (CO3029)

Assignment Final Report

A Comparative Survey on Machine Learning-based Data Matching Models

---

Course Instructor: Assoc. Prof. Dr. Le Hong Trang

	Full name	Student Id
1	Tran Nguyen Thai Binh	2110051
2	Do Nguyen An Huy	2110193
3	Nguyen Dang Anh Khoa	2010339
4	Nguyen Tuan Minh	2110359

HO CHI MINH CITY, 2023

Contents

List of Figures

List of Tables

1 Introduction

Data matching (*also known as* **Entity matching**) is the problem of finding structured data items that describe the same real-world entity, such as *(Eric Smith, Johns Hopkins)* and *(E. Smith, JHU)*. The entity can be a tuple in a database, an XML element or a set of RDF triples. The applications of data matching consist of identity management, risk management, fraud detection and prevention, credit scoring/risk assessment and compliance monitoring.

Some approaches has been suggested to solve this problem, including handcrafted method, supervised and unsupervised learning, probabilistic method, correlated method. In this report, we will explore data matching model based on machine learning and deep learning, focusing on understanding the mechanisms, processes, and architecture of the deep learning implementation approach - the most modern approach (State Of The Art - SOTA) for the data matching problem up to this point.

To observe and compare the performance of the models under study, our group conducted experiments on three different types of datasets. Subsequently, we drew some key observations about these models, focusing particularly on the effectiveness of deep learning-based models in data matching tasks with certain data characteristics.

1.1 Application of Data Matching

While the goal of data matching is to identify more accurate and distinct records from a group of similar ones, its application varies from one industry to another. Let's take a detailed look at how data matching is utilized in different contexts:

- **Government and Public:** Entity matching is a critical tool for federal agencies and public institutions. They use it to scrutinize Personally Identifiable Information (PII) such as Social Security Numbers (SSN), passport details, and licensing numbers. This process aids in fraud detection, compliance with standards, and the execution of political analyses.
- **Education:** Data matching is utilized across various sources, such as demographic information of students and teachers, among others, to evaluate student performance. It helps in distinguishing between effective and ineffective teaching methods, analyzing grade fluctuations, and pinpointing policy initiatives from Statewide Longitudinal Data Systems (SLDS) data.
- **Banking and Finance:** In the banking and financial services sector, data matching is employed as a tool to pinpoint wrongdoers as part of efforts to combat money laundering.

It's also used to fulfill Know Your Customer (KYC) compliance obligations, and to conduct FICO credit scoring.

- **Healthcare:** Healthcare organizations employ data matching techniques to identify patient's record across various Electronic Health Record (EHR) systems and databases. They use unique identifiers such as those from the Office of the National Coordinator for Health Information Technology (ONC), the United States Postal Service (USPS), and the Council for Affordable Quality Healthcare (CAQH). This approach allows for a consolidated view of a single patient's data, which aids in making accurate diagnoses and prescribing the appropriate medications.

- **Sales and Marketing:** Businesses frequently need to identify matches in order to eliminate duplicate and incorrect contacts within their Customer Relationship Management (CRM) systems and relational databases. The resulting unified view of each customer allows companies to enhance their upselling and cross-selling efforts, amplify their omni-channel marketing campaigns, and boost their marketing return on investment.

1.2 Problem Setting

Suppose we are given two relational tables X and Y. Assuming they have identical schemas (in general case they will not), X and Y describes some properties of an entity (e.g., person, address, phone number). We say a pair of tuple x, y (where $x \in X$ and $y \in Y$) a match if x and y refer to the same real-world entity. The problem's goal is to find *all* matches between two tables X and Y. For example, given two tables in *Figure 1* describing information about name, city and state of people. The first match $1.a, 2.a$ tells that `Dave Smith, Madison, WI` and `David D.Smith, Madison, WI` refer to the same real-world person.

Table 1

	Name	City	State
1.a	Dave Smith	Madison	WI
1.b	Joe Wilson	San Jose	CA
1.c	Dan Smith	Middleton	WI

Table 2

	Name	City	State
2.a	David D.Smith	Madison	WI
2.b	Dan W.Smith	Madison	WI

Matches
{1.a, 2.a}
{1.c, 2.b}

Figure 1: An example of matching relational tuples that describe persons.

1.3 Types of Data Matching Problems

In this report, we will divide data matching problem to three considering following type and examine which type of approach is helpful for each type:

- *Structured data matching:* We consider a dataset to be structured when its entries are fairly clean, meaning that the values of the attributes are well-organized and each cell within a record contains information that is solely related to the attribute that describes that cell. Additionally, the data may include text-based attributes, but these are typically of limited length (for example, product title, address). Figure 1 is an example of Structured data matching

- *Textual Data Matching:* All attributes in a dataset which correspond to raw text entries. Figure 2 below is an example of textual data matching.

	Description
t_1	Kingston 133x high-speed 4GB compact flash card ts4gcf133, 21.5 MB per sec data transfer rate, dual-channel support, multi-platform compatibility.

	Description
t_2	Kingston ts4gcf133 4GB compactflash memory card (133x).

Figure 2: Example of textual data matching

- *Dirty Data Matching:* We consider a dataset to be dirty when it is structured but the attribute values may be "injected" under the wrong attribute (usually happen to text-based attributes). Figure 3 below is an example of dirty data matching.

	Name	Brand	Price
t_1	Adobe Acrobat 8		299.99

	Name	Brand	Price
t_2	Acrobat 8	Adobe	299.99

Figure 3: Example of dirty data matching

1.4 Approach for Entity Matching so far

There has been several classes of solutions to data matching problem, with 5 most prominent traditional classes:

- *Rule-based matching*: These approaches employ handcrafted matching rules, which means that the developer writes rules that specify when two tuples match. Lightnearly weighted combination, logistic regression combination are example of this class.

  - *Lightnearly weighted combination*: Compute the similarity score between tuples x and y as a linearly weighted combination of individual similarity scores of attributes.

$$sim(x, y) = \sum_{i=1}^{n} \alpha_i * sim_i(x, y) \tag{1}$$

    with n is number of attributes in each table, $sim_i(x, y)$ is the similarity score between i-th attributes of x and y, $\alpha_i \in [0, 1]$ is pre-specified weight that indicates the important of i-th attribute $sim(x, y)$ such that $\sum_{i=1}^{n} \alpha_i = 1$. We declared x and y matched if $sim(x, y) \geq \beta$ for a pre-specified β, and not matched otherwise.

  - *Logistic regression combination*: Compute the similarity score through the formula:

$$sim(x, y) = \frac{1}{1 + e^{-z}} \tag{2}$$

    where $z = \sum_{i=1}^{n} \alpha_i * sim_i(x, y)$. In this case, we do not constraint α_i in range $[0, 1]$ and sum to 1, so z can range from $-\infty$ to $+\infty$. However, when z exceeded a certain value, the increase of z is gradual. This can solve the case of diminishing returns, where after a certain threshold, an increase in $sim_i(x, y)$ should contribute less to the similarity score of x and y.

  Type of solution in this class is easy to start and implement, typically run fast and can encode complex matching knowledge. However, these approaches can be labor intensive when it takes a lot of time to write good rules. It can be difficult to set approriate weights. In certain cases, it is not even clear how to write rules.

- *Learning-based matching*: This class use supervised learning to automatically create matching rules from labeled samples. Given a training data T with specific form, we define a set of feature function to transform T into a new training set T' and apply a learning algorithm such as decisions trees or support vector machine to learn a matching model M. These approaches have two major advantages compared to rule-based ones when it can *automatically* examine a large set of features to select the most useful ones (in rule-based it is done *manually*, which is difficult and time-consuming) and construct very complex

"rules". However, this class of solution of require a large number of training samples, which can be hard to obtain.

- *Matching by Clustering:* Many of commonly known clustering techniques have been applied to data matching, including agglomerative hierarchical clustering, k-means, and graph-theoretic clustering. Selecting a good cluster similarity method is application dependent and requires careful consideration from the application builder.

- *Probabilistic Approaches to Data Matching*: These approaches model the matching decision as a set of variables over which there is a probability distribution. The advantages of probabilistic are that:

  1. They provide a principled framework that can naturally incorporate a wide variety of domain knowledge.
  2. They leverage the wealth of probabilistic representation and reasoning techniques that have been developed in the past two decades in the artificial intelligence and database communities.
  3. These methods provide a frame of reference for comparing and explaining other matching approaches

  However, probabilistic methods suffer from certain disadvantages. They are computationally expensive and often hard to understand and debug matching decisions

- *Collective Matching:* In many real-world scenarios, matching decisions are intuitively correlated, and by exploiting such correlations we may be able to improve matching accuracy.

1.5 Related Work

There has been some researches about entity matching in database literature, which match structured records. Lise Getoor and Ashwin Machanavajjhala (2012) [1] review some techniques related to improving complexity and scalability of Entity Resolution algorithms, as well as highlight a few open research directions.

Researches by Wenfei Fan et al (2009) [2] and Rohit Singh et al (2017) [3] demonstrated that while rule-based solutions are comprehensible, they necessitate significant input from a specialist in the field. Some work, such as Pradap Konda et al (2016) [4] and Parag Singla et al (2006)

[5], have focused on learning matching functions to address this problem. Work on the matching step of EM typically uses rules, learning, or crowdsourcing.

For Deep Learning work, Muhammad Ebraheem et al (2017) [6] introduces two deep learning models that are based on neural network architectures widely used in natural language processing literature. In their work, entity matching is formulated as a binary classification task using logistic loss. However, there are alternative ways to approach the same problem. Elad Hoffer and Nir Ailon (2015) [7] develops effective object embeddings using triplet loss, followed by the learning of a neural network classifier. Paul Neculoiu, Maarten Versteegh, and Mihai Rotaru (2016) [8] use contrastive loss to learn effective entity embeddings in vector space. Finally, Oriol Vinyals et al (2016) [9] discuss matching Networks, a method used for image classification by utilizing labels from similar images, could potentially be modified for use in entity matching.

2 Method

2.1 Overview

The article [10] proposed an architecture template for data matching which they called *Deepmatcher*. The article's main purpose is to look into the data matching problem and propose a general architecture pipeline to incorporate deep learning techniques & run the pipeline on some datasets to see how well it performs against more traditional data matching methods.

The method introduced aims to solve data matching problem instances in which the two data sources are structured and follow the same schema. A record from the data sources may be dirty, that is, there may be missing values and data misalignment may exist i.e. the value for an attribute ends up in a column for another attribute.

Here, we present a framework for data matching model that was introduced in [10]. The reason we choose this is 3-fold:

- The framework splits the inference process into discrete steps, making it clear and easy to understand how inference is actually carried out.
- The framework has many replaceable components & extension points, making it highly customizable. Another implication is that it makes experimenting with various models easy.
- The framework is amenable to deep learning techniques, whose application to data matching we're focusing on in this report. One favorable consequence of this is by utilizing the framework, data matching problems can enjoy the benefits of deep learning.

This method follows the traditional supervised learning process. In the next section, we'll take on the experiment ourselves to evaluate various models following this architecture against traditional methods.

2.2 Deep Learning Background

2.2.1 Word embedding

A word embedding is a representation of a word, usually a real-valued vector. Typically, vectors that are very similar (based on some distance metrics) should represent very semantically similar words. Word-to-vector mappings can be generated from neural network models, dimensionality reduction on the word co-occurrence matrix, etc. In Figure 4, when representing the embedding

vectors of words in a 2-dimensional space, the vector representing "child" is closer to the vector representing "infant" and farther from the vector representing "grandfather". This is consistent with their semantic relationship in terms of age and gender. The "similarity" in semantics can be learned by pre-trained language models.

Word2Vec is a popular word embedding technique that represents words as dense vectors in a high-dimensional space. It uses neural networks and two approaches, CBOW and skip-gram, to learn word representations from large text corpora. The corresponding models for these two approaches are illustrated in Figure 5 and Figure 6. Intuitively, CBOW predicts a target word based on its context, while skip-gram predicts context words given a target word. The resulting word embeddings capture semantic and syntactic information, allowing for algebraic operations to capture word relationships. Word2Vec is widely used in natural language processing tasks like sentiment analysis and machine translation.

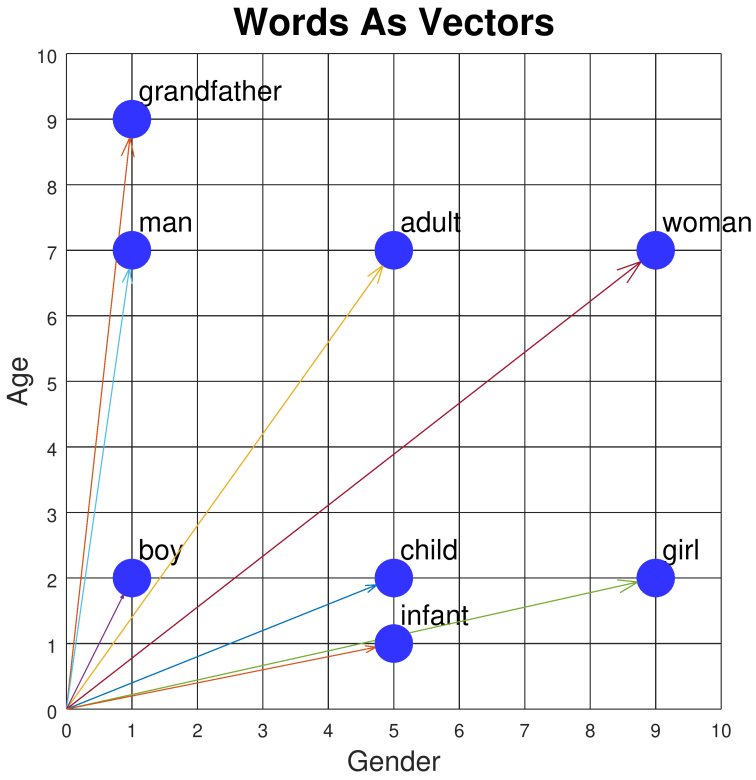

Figure 4: An example of word embedding

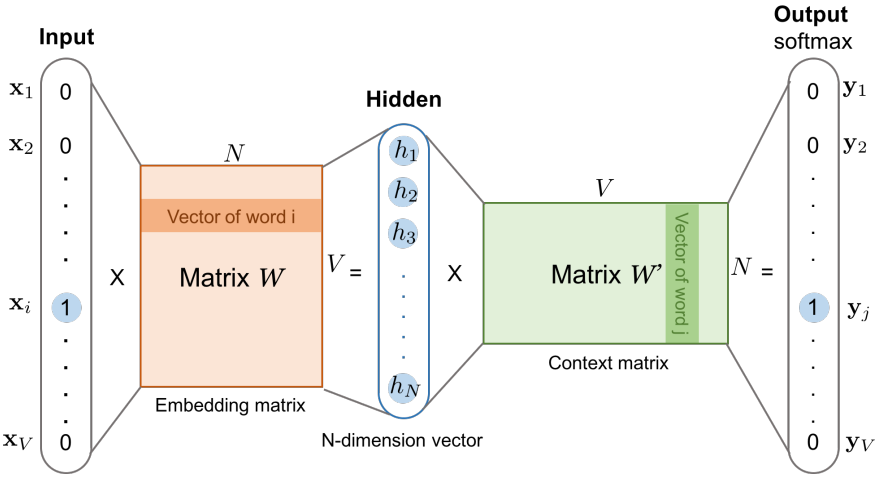

Figure 5: The skip-gram model. Both the input vector x and the output y are one-hot encoded word representations. The hidden layer is the word embedding of size N

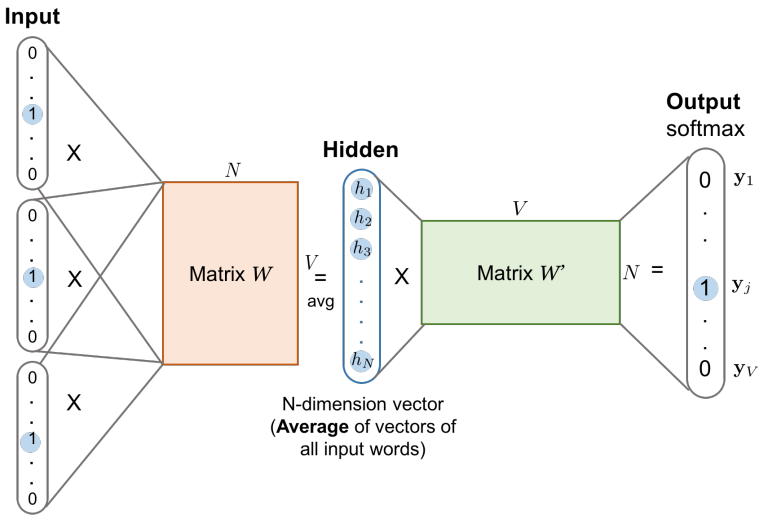

Figure 6: The CBOW model. Word vectors of multiple context words are averaged to get a fixed-length vector as in the hidden layer

2.2.2 RNN

RNN is a model that can make decisions from sequential inputs, e.g. strings of text, videos (image sequences). It's especially useful for sequence processing tasks as it can accept sequences of variable length.

An RNN may contain one or more recurrent units. A recurrent unit maintains hidden state vector which is initialized to a fixed value at the beginning and updated through out the inference

process. At time step t, the recurrent unit takes the t^{th} component of the sequence along with its hidden state vector to produce an output. The new hidden state vector is set to this output. By the time the entire input sequence has been processed, we obtain a sequence of vectors. Usually, the last vector is chosen as a summary of the input sequence.

Notice that each time step, the new hidden state depends on both its previous hidden state & current input. For this reason, the last vector depends on all of the components of the sequence. The mechanism for processing sequential data with hidden state vectors is illustrated in Figure 8.

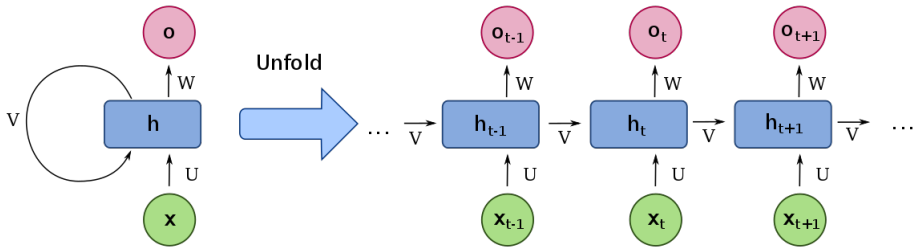

Figure 7: A recurrent neural network and the unfolding in time of the computation involved in its forward computation

2.2.3 Attention

The attention mechanism enables models to selectively emphasize certain areas of input data while downplaying less important features based on the context. An attention model when fed an input sequence will take into account the provided context vector to yield a vector that represents how the input sequence aligns with the context.

In an attention model, given an input sequence and a context vector, the model assigns weights or importance scores to different elements or steps in the sequence. These weights indicate the relevance or attention that each element should receive based on the context.

The attention mechanism calculates these weights by comparing the context vector with each element or step of the input sequence. This comparison is typically done by measuring the similarity between the context vector and the hidden state of each element in the sequence. The similarity can be computed using different methods, such as dot product, cosine similarity, or a neural network-based approach.

Once the attention weights are calculated, they are used to weigh the contributions of the corresponding elements in the sequence. The weighted elements are then combined to generate a context-specific representation or summary of the input sequence. This context representation captures the relevant information from the input sequence that aligns with the given context.

The context representation is then used by the model to generate the output or make predictions. By incorporating the attention mechanism, the model can effectively focus on relevant parts of the input sequence, giving more emphasis to important features while downplaying less important ones. This allows the model to handle long sequences more effectively and improve the quality of generated outputs.

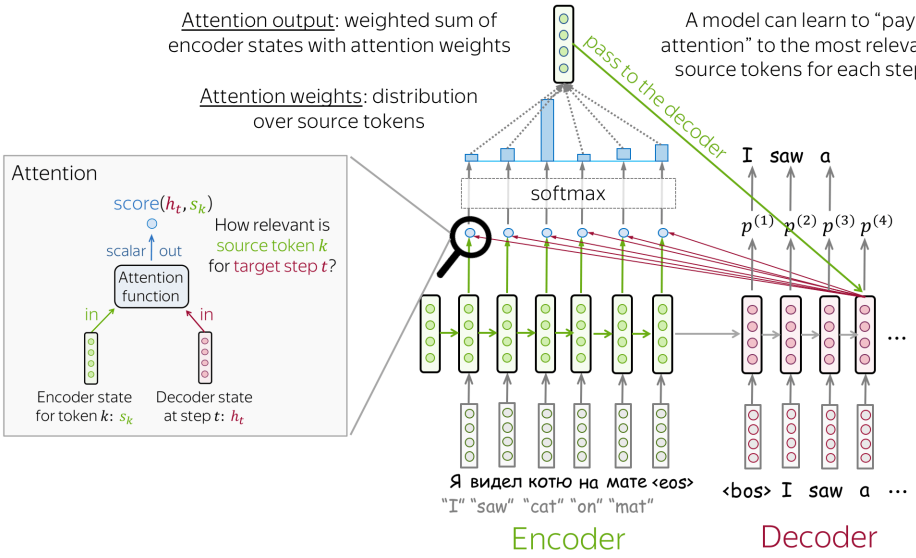

Figure 8: An example of incorporated attention in machine translation tasks

Encoder and Decoder in Machine Translation tasks

In the context of attention models, the encoder and decoder are components that handle the input and output sequences, respectively.

The encoder is responsible for processing the input sequence and capturing its representations. It typically consists of recurrent neural network (RNN) layers, such as Long Short-Term Memory (LSTM) or Gated Recurrent Unit (GRU), which process the input elements one by one and maintain hidden states that capture the sequential information. The encoder's role is to generate a set of hidden state vectors that encode the input sequence's information.

On the other hand, the decoder takes the generated hidden states from the encoder and generates the output sequence. It is also an RNN-based component that operates in a step-by-

step manner. At each time step, the decoder attends to the relevant parts of the input sequence by using the attention mechanism. The attention mechanism calculates attention weights for each hidden state vector in the encoder based on its relevance to the current decoding step. These attention weights determine which parts of the input sequence should receive more focus or emphasis during the decoding process.

The decoder utilizes the attention weights to compute a context vector, which is a weighted sum of the encoder's hidden state vectors. The context vector represents the relevant information from the input sequence that aligns with the current decoding step. This context vector, along with the previous decoder state and the previously generated output, is used to predict the next output element in the sequence. This process is repeated until the entire output sequence is generated.

The attention mechanism allows the decoder to dynamically focus on different parts of the input sequence at each decoding step, effectively aligning the input and output sequences. This enables the model to generate accurate and contextually aware outputs, especially in tasks like machine translation, where the output sequence depends on the input sequence's alignment.

2.3 Architecture Template

The inference framework is presented in Figure 9.

The input to the framework is a pair of records, each of which is taken from one of the two data sources respectively. The output is a verdict on whether these two records is a match.

There are three major replaceable components in the pipeline: attribute encoding, attribute similarity representation and classification.

2.3.1 Attribute embedding

The data that was fed into the pipeline is usually supposed to be human-readable for interpretation and analysis by human, i.e text, numbers.

For the sake of AI algorithms, though, human-readable formats do not lend themselves well to efficient computation. The attribute encoding component, therefore, is to transform these into a more "machine-friendly" representation. Depending on the problem instance, each attribute in the schema may have its own attribute encoder or share it with the others.

With respect to an attribute, the attribute embedding module takes a value in the attribute domain and yields an embedding of the value. This stage is important as the better the

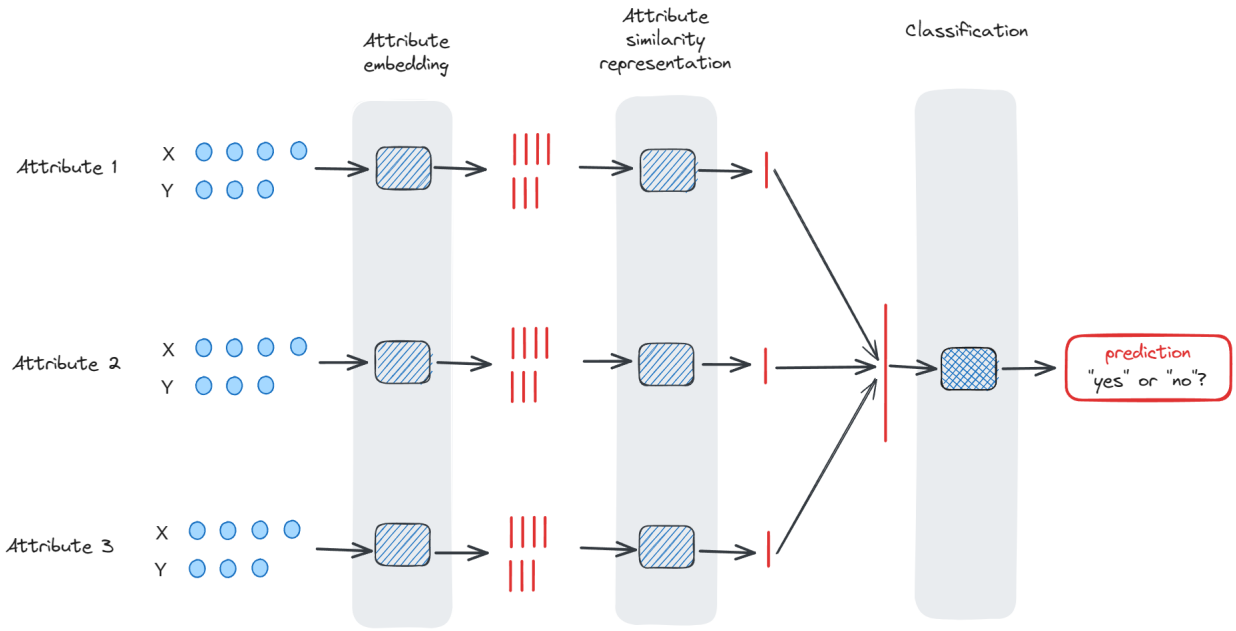

Figure 9: The inference framework introduced in [10]

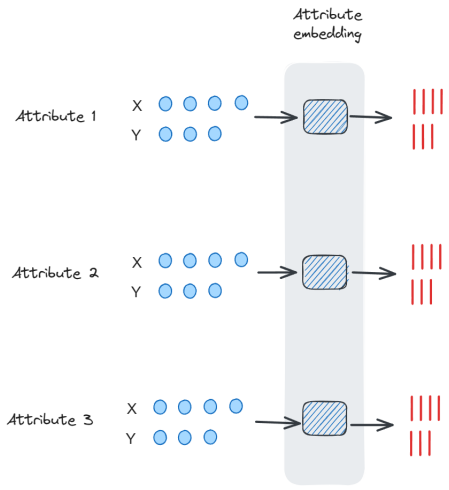

Figure 10: The attribute embedding module

embedding captures the semantics of the attribute and the correlation between values of the attribute's domain, the more likely the whole inference pipeline performs better. If the attribute values come in sequences, e.g. text is a sequence of words, a sequence of embedding vectors is produced.

2.3.2 Attribute similarity representation

This component looks at the two embeddings/embedding sequences produced by the attribute embedding components and produces a vector that represents the similarity between the two sequences.

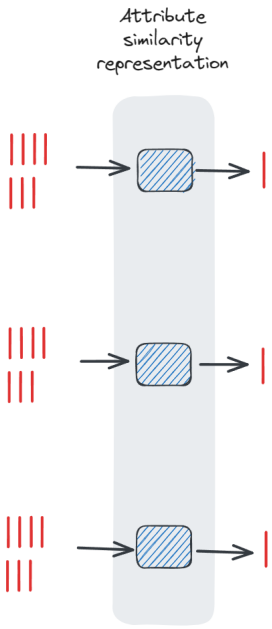

Figure 11: The attribute similarity representation module

This component may be divided into 2 sub-components:

1. Attribute summarization component (optional): When the input is just two simple embeddings, this component is not needed. However, when the input is two embedding sequences, this component can try to "summarize" each of the two sequences into two corresponding simple embeddings. These two new embeddings sum up the knowledge connoted by the two original attribute values. Learning models that can deal with sequential inputs can be used here, e.g. RNN, LSTM, Attention-based models.

2. Attribute comparison component: This component looks at the two summary vectors to generate another vector that describes the similarity of the two input vectors. This can be as simple as a fixed distance measure (Euclidean, Mahattan, etc.) or a learnable distance.

The output of this component is a set of vectors that represents the similarity in each attribute of the two records.

2.3.3 Classification

This component is the one where the final verdict is made. This component processes all the attribute similarity vectors to decide whether the two input records is a match. A simple fully-connected network suffices for this task: All the similarity vectors are concatenated and fed into a neural network.

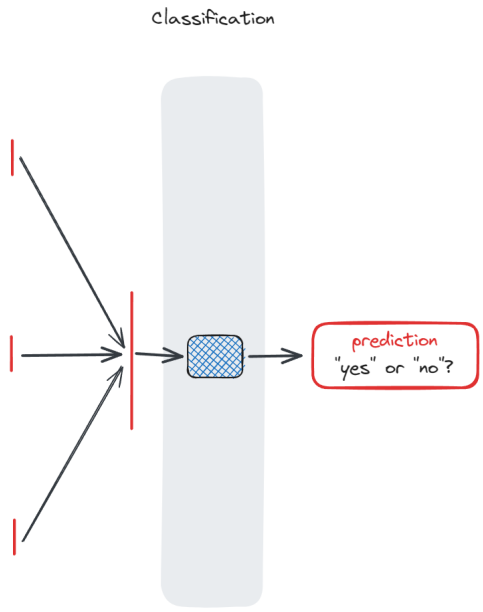

Figure 12: The classification module

2.4 Component choices

2.4.1 Attribute embedding

There are two axis of selection for the attribute embedding module.

The first axis is the granularity of the embedding. The word embedding may treat the whole word as an atom and lookup that word in a lookup table to retrieve its embedding. The main drawback of this approach is that out-of-vocabulary words must be specially handled. Another approach that avoid this problem is to treat a word as a sequence of character - the word embedding module will generate an embedding based on this character sequence. Due to words (at least in English) often have structures in themselves, using this approach can actually improve performance in domains with rare words.

Architecture module		Options	
Attribute embedding		*Granularity:* (1) Word-based (2) Character-based	*Training:* (3) Pre-trained (4) Learned
Attribute similarity representation	(1) Attribute summarization	(1) Heuristic-based (2) RNN-based (3) Attention-based (4) Hybrid	
	(2) Attribute comparison	(1) Fixed distance (cosine, Euclidean) (2) Learnable distance (concatenation, element-wise absolute difference, element-wise multiplication)	
Classifier		NN (multi-layer perceptron)	

Figure 13: The choices suggested for each module in [10]

The second axis is whether a pre-trained model should be used. Using pre-trained models can save significant training time and they are guaranteed to generalize well on large corpus of documents. However, using a learned model can be beneficial for domains whose tokens take on highly-specialized semantics.

2.4.2 Attribute summarization

Models that are able to process sequences can be used here, ranging from simple heuristics (such as taking the mean of the sequence elements) to significantly more complex models such as RNN, attention or a hybrid of both.

Given the attribute embeddings $u_{e_1,j} \in \mathbb{R}^{d \times m}$ and $u_{e_2,j} \in \mathbb{R}^{d \times k}$ for attribute A_j, a summarization process H outputs two summary vectors $s_{e_1,j} \in \mathbb{R}^h$ and $s_{e_2,j} \in \mathbb{R}^h$. There are four major options for attribute summarization:

- **Aggregate Function**: The process of summarization, denoted as H, involves a basic aggregation function applied to each embedding sequence $u_{e_1,j}$ and $u_{e_2,j}$. Examples of such aggregation functions include averaging or weighted averaging.

  One advantage of this summarization approach is its training efficiency, as it typically doesn't require any learning process. However, models that rely on this type of summarization are limited in their ability to capture intricate relationships between words in the input sequence. The effectiveness of this summarization method heavily relies on the quality of the embedding vectors.

- **Sequence-aware summarization**: Given a process, denoted as H, that aims to capture complex interactions among tokens in input sequences $u_{e_1,j}$ and $u_{e_2,j}$. To achieve this, an RNN (Recurrent Neural Network) can be utilized, taking into consideration both the order and semantics of the tokens in the input sequence. Various types of RNN models exist, such as LSTM (Long Short-Term Memory) networks, GRU (Gated Recurrent Unit) networks, and bidirectional networks.

  To implement process H with an RNN, the input sequence $u_{e,j}$ is passed through the RNN, resulting in a sequence of hidden states $h_{e,j}$. These hidden states are then aggregated into a single h-dimensional vector, $s_{e,j}$. Common aggregation operations include selecting the last hidden state of the RNN as $s_{e,j}$ or calculating an average across all hidden states $s_{e,j}$.

  The primary advantage of this summarization method is its ability to consider the context encoded in the entire input sequence. However, there are limitations to this approach. Firstly, it struggles to learn meaningful representations when dealing with very long sequences. Secondly, it does not analyze the input pairs $u_{e_1,j}$ and $u_{e_2,j}$ together to identify shared contexts across sequences. This limitation can lead to a significant loss in performance, particularly when the input sequences vary significantly in length.

- **Sequence Alignment**: In this approach, process H takes both sequences $u_{e_1,j}$ and $u_{e_2,j}$ as input and utilizes one sequence as context when summarizing the other. To achieve this, process H can be constructed around attention mechanisms. These mechanisms learn to compute a soft alignment between the two given word sequences and then perform a word-by-word comparison.

  Attention mechanisms are highly expressive but have a significant drawback. They only utilize the context provided to them as input and disregard any context present in the original input sequence. For instance, attention-based summarization cannot consider the positional information of input tokens. Consequently, attention methods may not perform well in situations where the first token is the most informative for matching two entity mentions. One way to address this issue is by combining attention mechanisms with sequence-based summarization methods.

- **Hybrid**: These methods result in highly expressive models but require significant computational resources for training.

2.4.3 Attribute comparison

The process of attribute comparison involves determining the distance between the summary vectors $s_{e_1,j} \in \mathbb{R}^h$ and $s_{e_2,j} \in \mathbb{R}^h$ for attribute A_j. This comparison operation is denoted as D. It is assumed that the result of this operation is a fixed-dimension vector $s_j \in \mathbb{R}^l$.

Well-known distance metrics can be used for this module, popular metrics include cosine distance and Euclidean distance.

A neural network can also be used here to learn a good representation for the similarity of any two input vectors.

2.4.4 Classifier

A simple fully-connected network suffices for this module.

2.5 Specific deep learning-based data matching models

These four models share the attribute embedding module and the classifier module, differing in only the attribute similarity representation module. For the attribute embedding module, the pre-trained character-level embedding model fastText is used. As for the classifier module, a two layer fully-connected ReLU HighwayNet followed by a softmax layer. The name of each module resembles the technique used in the attribute summarization module.

2.5.1 SIF

The attribute summarization module uses a simple heuristics to summarize the embedding sequence: Taking the weighted average of all the embeddings in the sequence. The weight assigned to a word w (and thus a specific embedding) is given as

$$\frac{a}{a + f(w)}$$

where $f(w)$ is the frequency w appears in the corpus and a is learnable. The attribute comparison module performs an element-wise absolute difference comparison operation.

As can be seen, this scheme does not take into the order of each embedding in the sequence into account in the summarization step.

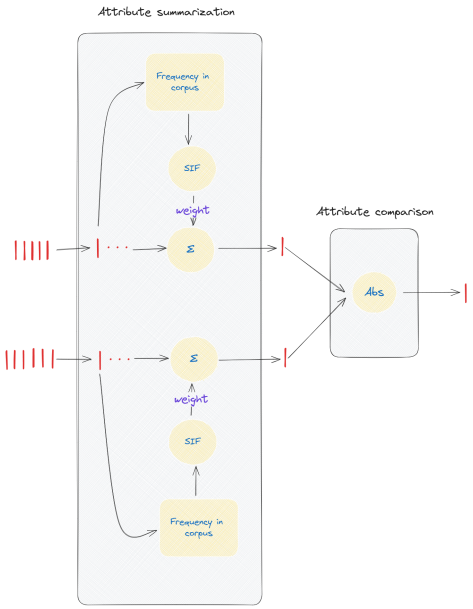

Figure 14: An attribute similarity representation module based on SIF

2.5.2 RNN-based

The second model uses a bidirectional RNN for attribute summarization and employs an element-wise absolute difference comparison operation to generate input for the classifier module.

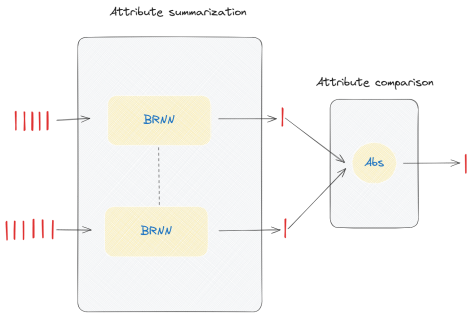

Figure 15: An attribute similarity representation module based on BRNN

RNN allows a more sequence-aware summarization step.

Specifically, bidirectional RNN is used because it allows a summary embedding in the output sequence to contain information about both the previous and following input embedings.

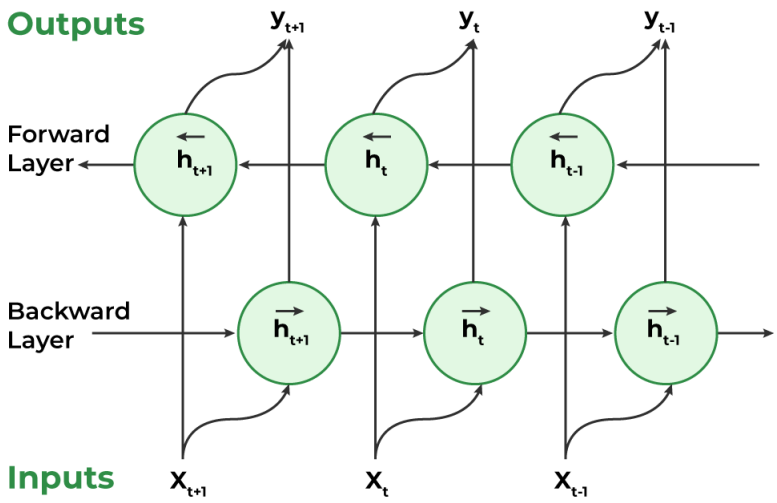

Figure 16: BRNN architecture - Taken from GeeksForGeeks

2.5.3 Attention-based

This model employs decomposable attention for attribute summarization and utilizes vector concatenation for attribute comparison.

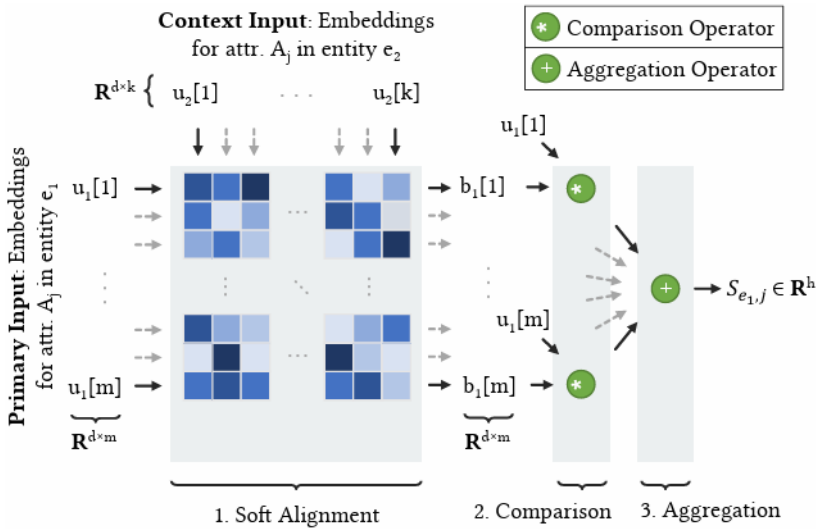

Figure 17: An attention-based attribute similarity representation module presented in [10]

Suppose the two input embedding sequences u_1 and u_2. This model walks through 3 steps:

1. Soft alignment: We take u_2 to be the context input and u_1 to be the primary input. For each pair of embeddings $(u_1[i], u_2[j])$, we feed each embedding to a two layer HighWay net with Relu and take the dot product of the results. Upon doing this with all the pairs, we obtain a matrix. We then proceed to normalize all weights using softmax.

   The soft-aligned encoding of $u_1[i_0]$ - denoted as $b_1[i_0]$ - is the weighted average of the embeddings in the context input u_2 with the weights being the matrix entry for the pair $u_1[i_0]$ and the corresponding embedding in u_2.

2. Comparison: The comparison module compares the soft-aligned encoding of $u_1[i]$, that is $b_1[i]$, with $u_1[i]$. It uses a two-layer HighwayNet with ReLU non-linearities to yield a vector that represents the similarity between $b_1[i]$ and $u_1[i]$.

3. Aggregation: All the similarity vectors are summed up and the output is normalized by dividing with $\sqrt{l_1}$ with l_1 being the length of the primary input. This normalization step normalizes the variance of the output vector.

These steps are repeated with the roles of u_1 and u_2 swapped. The two summarization vectors are then concatenated to be fed to the classifier module.

2.5.4 Hybrid

This model employs a bidirectional RNN with decomposable attention for attribute summarization. For attribute comparison, it utilizes vector concatenation augmented with element-wise absolute difference.

Similar to the attention-based model, this model also walks through 3 steps:

1. Soft alignment: The matrix is obtained as in the attention-based model. However, the soft-aligned encoding for each embedding in the primary input are obtained by taking the weighted average of an encoding of the context input. This encoding of the context input is the concatenated hidden states produced by feeding the context input to an RNN.

2. Comparison: This proceeds similarly to the attention-based model, comparing the soft-aligned encoding with the concatenated hidden states produced by feeding the primary input to the same RNN in the previous step.

3. Aggregation: We perform a weighted average on the vectors obtained from the last step. The weight for each vector is obtained by running an RNN on the context input, concatenating

the last hidden state with the vector then feed the concatenated vector to a ReLU HighwayNet followed by a soft-max layer.

The same steps are performed on the two inputs with their roles swapped.

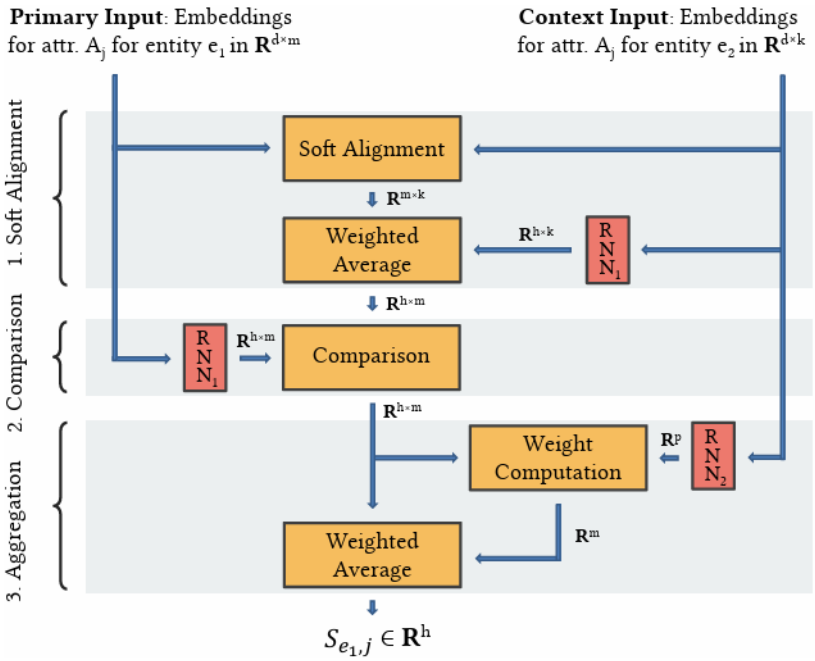

Figure 18: A hybrid attribute similarity representation module presented in [10]

3 Experiments

3.1 Objectives

The *first* objective is to compare the effectiveness of machine learning methods in the data matching problem. Firstly, we aim to identify traditional models that perform well in data matching tasks. Secondly, we want to observe whether designing models with different complexities in deep learning methods truly provides better efficiency compared to traditional machine learning models when there is a trade-off in terms of execution time.

The *second* objective is to evaluate and compare these methods on three types of datasets corresponding to three different data matching problems (structural, dirty, textual). We want to determine when deep learning-based models should be applied and when traditional machine learning models are sufficient to optimize time and cost.

3.2 Dataset overview

General description

We use many datasets from a diverse array of domains, including all three types of data matching problems: structured information (in the form of rows with attribute columns, similar to records in a relational database table), text information (such as product descriptions in textual form), and dirty information (structured information with missing attribute values, values mixed with other attributes, or spelling errors).

Data instance format in the dataset

Each data instance in each dataset is a labeled tuple pair, where each tuple pair comes from the 2 tables being matched, say table A and table B. We assume that both the tables being matched have the same schema.

The table provided presents an overview of all the datasets. It includes a summary of several columns:

- **Size**: This column indicates the total number of labeled tuple pairs included in the dataset.
- **Pos.**: The "Pos." column represents the count of positive instances, which refers to the number of tuple pairs that are marked as a match in the dataset.
- **Attr.**: This column denotes the number of attributes present in the tables being matched. It is important to note that both tables have the same schema, meaning they share the same set of attributes.

Table 1: Dataset Summary

Type	Dataset	Domain	Size	Pos.	Attr.
Structured	iTunes-Amazon-1	music	539	132	8
Structured	DBLP-Scholar-1	citation	28,707	5,347	4
Textual	Abt-Buy	product	9,575	1,028	3
Dirty	iTunes-Amazon-2	music	539	132	8
Dirty	DBLP-Scholar-2	citation	28,707	5,347	4

The datasets are available in [10]. We get access to these datasets online via this link: `https://github.com/anhaidgroup/deepmatcher/blob/master/Datasets.md`

3.3 Methods and Experimental setup

3.3.1 Experimental setup

Dataset selection

Typically, the Expectation-Maximization (EM) process in data matching involves two phases: blocking and matching. The objective of blocking is to reduce the search space from the cross product of $D \times D'$ to a candidate set C, which contains pairs of entity mentions that are likely to be matches.

For experimental purposes, a selected set of public datasets is provided in two versions: a raw data version that includes all data instances from two different tables (corresponding to two data sources), and a processed version obtained after the blocking phase, which results in a candidate set comprising pairs of tuples with corresponding match/non-match labels.

In this experiment, our focus is primarily on comparing the effectiveness of methods in the matching phase. Therefore, we will utilize the candidate set versions of the datasets provided.

Training and Evaluation

Each dataset is divided into three sets: training, validation, and test, with a ratio of *3:1:1*.

- The training set is used to train the deep learning model. The attributes of the tuple pairs in the dataset are used as input features, and the labels indicate whether the tuple pairs are a match or non-match.

- The validation set is used to fine-tune the hyperparameters of the model. It helps in selecting the best-performing model for the unseen data. For deep learning-based models,

the validation set is also used to stop the training process when the performance on the validation set no longer improves after a certain number of iterations, aiming to prevent overfitting.

- The test set is used to evaluate and compare the performance of different models.

To assess the performance of the models, we use the F1-score as the evaluation metric, comparing the ground truth labels with the predicted labels. The training time of the models is also recorded.

3.3.2 Methods

In addition to the four deep learning-based methods mentioned in the previous section (SIF, RNN-based, Attention-based, Hybrid), we implemented six baselines for traditional machine learning methods in data matching, including:

1. *Naive Bayes:* A probabilistic classification algorithm based on Bayes' theorem. It assumes independence among the features and calculates the posterior probability of each class.

2. *Decision Tree:* A tree-based algorithm that splits the data based on feature values to make decisions. It creates a hierarchical structure of rules to classify or predict outcomes.

3. *Random Forest:* An ensemble learning method that combines multiple decision trees. Each tree is trained on a random subset of the data, and the final prediction is based on the majority vote of all trees.

4. *Support Vector Machine (SVM)*: A supervised learning algorithm that finds the optimal hyperplane to separate different classes. It maximizes the margin between classes and can handle both linear and non-linear decision boundaries.

5. *Logistic Regression:* A statistical regression model that predicts the probability of a binary outcome using a logistic function. It estimates the coefficients of the input features to make predictions.

6. *XGBoost:* An optimized gradient boosting algorithm that uses a combination of weak decision trees. It iteratively builds new trees to correct the errors made by previous models, achieving better performance.

Implementation

For deep learning-based methods in data matching, we utilize the deepmatcher library developed by AnHai's group at the University of Wisconsin-Madison to implement four simple baselines corresponding to the four methods mentioned in the previous section (SIF, RNN-based, Attention-based, Hybrid).

For traditional machine learning-based methods in data matching, we use the python packages from the Magellan project, developed by AnHai's group. This includes three packages: `py_entitymatching`, `py_stringmatching`, and `py_stringsimjoin`.

3.4 Results

3.4.1 Dataset 1: iTunes-Amazon-1 (Structured)

Table 2: Evaluation on dataset iTunes-Amazon-1

Method	F1-Score	Training Time (s)
Naive Bayes	78.69	1.99
Decision Tree	88.14	2.25
Random Forest	89.66	4.95
Support Vector Machine	92.86	2.62
Logistic Regression	89.29	2.17
XGBoost	89.66	7.11
SIF	90.91	231.64
RNN-based	82.76	311.65
Attention-based	84.62	579.87
Hybrid	92.59	837.06
No. epochs for Deep Learning methods	100	

- **Performance**
  - Most methods have good F1-score, especially SVM and Hybrid
- **Training time**
  - Training time of ML-based methods is very low (less than 10 seconds)
  - The training time of DL-based methods is relatively high

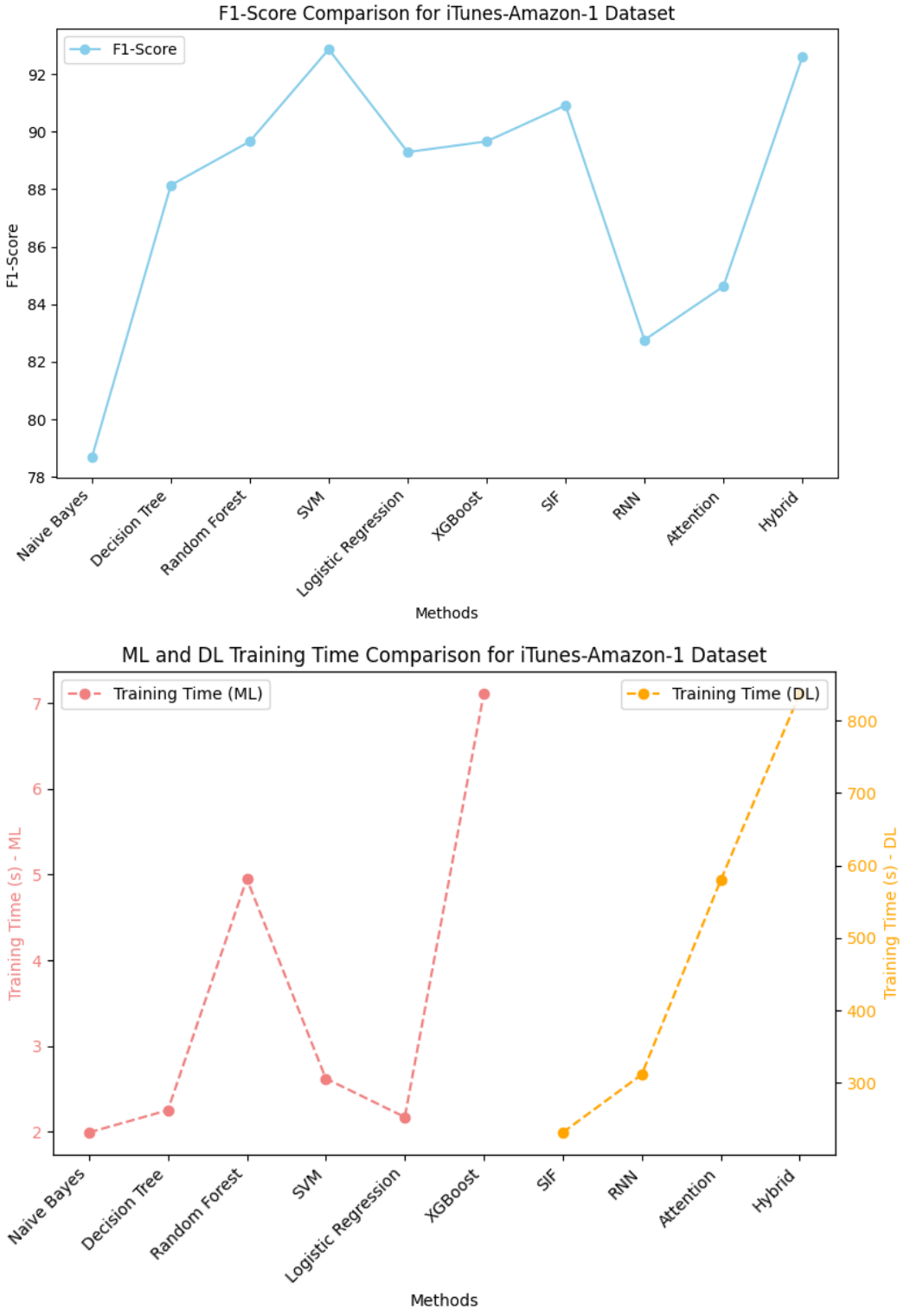

Figure 19: Dataset 1's result visualization

3.4.2 Dataset 2: DBLP-Scholar-1 (Structured)

Table 3: Evaluation on dataset DBLP-Scholar-1

Method	F1-Score	Training Time (s)
Naive Bayes	84.38	2.59
Decision Tree	88.18	5.9
Random Forest	90.09	45.43
Support Vector Machine	88.58	161.26
Logistic Regression	88.13	2.64
XGBoost	90.43	23.05
SIF	90.92	1143.56
RNN-based	93.83	1550.12
Attention-based	94.04	2568.65
Hybrid	95.52	5663.24
No. epochs for Deep Learning methods	20	

- **Performance**
  - Most methods have good F1-score and DL-based methods gives better results then ML-based ones
- **Training time**
  - Training time of ML-based methods incresing in some methods (Random Forest, SVM and XGBoost) due to significant increase in dataset size (539 to 28707)
  - The training time of DL-based methods is extreme high (even number of epochs is reduced)

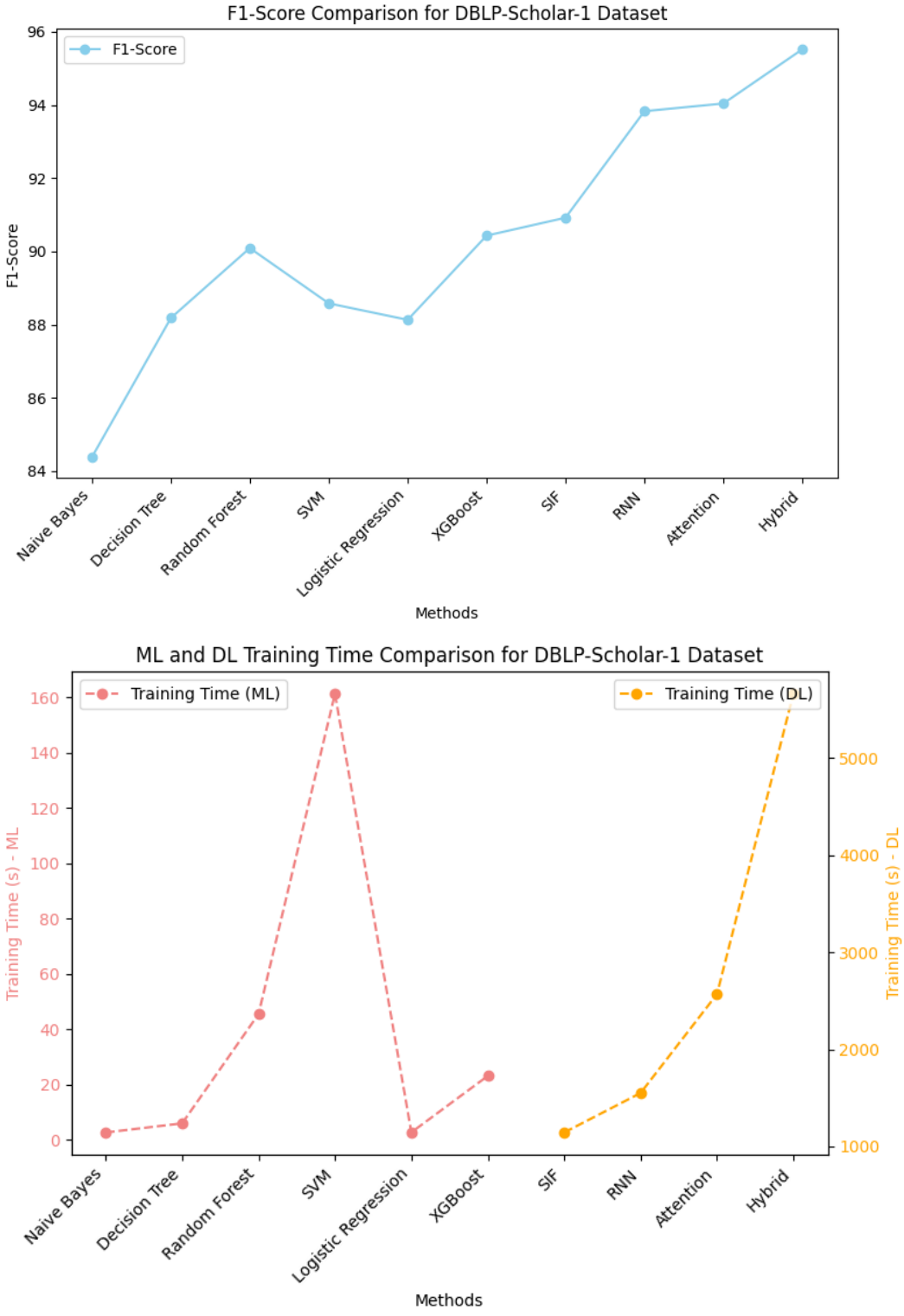

Figure 20: Dataset 2's result visualization

3.4.3 Dataset 3: Abt-Buy (Textual)

Table 4: Evaluation on dataset Abt-Buy

Method	F1-Score	Training Time (s)
Naive Bayes	36.26	2.19
Random Forest	47.72	4.15
Decision Tree	46.20	19.12
Support Vector Machine	N/A	N/A
Logistic Regression	36.75	2.24
XGBoost	52.87	19.51
SIF	38.95	428.36
RNN-based	41.75	576.88
Attention-based	56.10	860.56
Hybrid	72.86	1245.71
No. epochs for Deep Learning methods	20	

- **Performance**
  - Most methods do not result good metrics in this situation, only Hybrid method gives acceptable metrics
- **Training time**
  - Training time of ML-based methods is quite low except SVM, this method may not work for this type of dataset (textual)
  - Similar to previous results, training time of DL-based methods is high

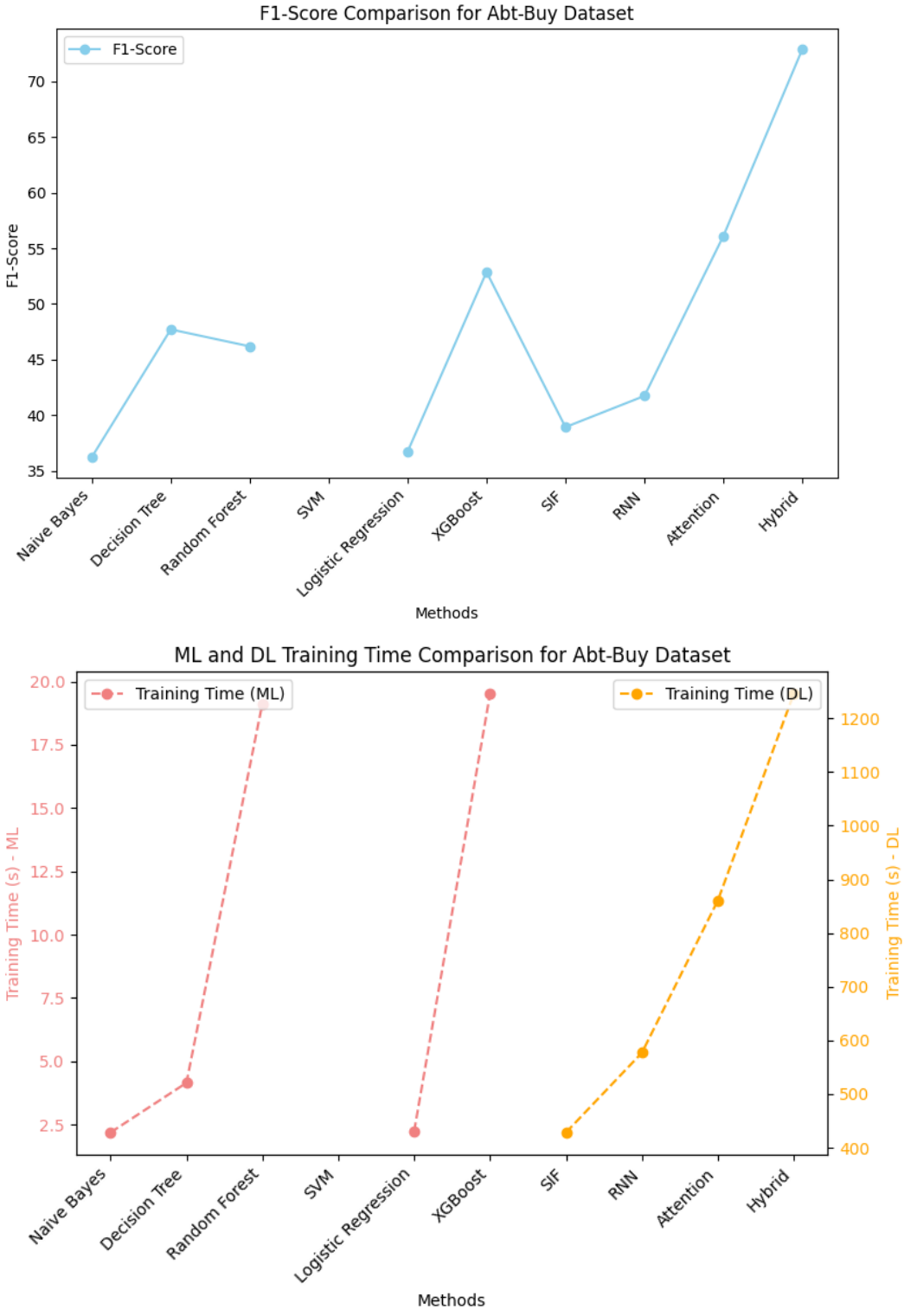

Figure 21: Dataset 3's result visualization

3.4.4 Dataset 4: iTunes-Amazon-2 (Dirty)

Table 5: Evaluation on dataset iTunes-Amazon-2

Method	F1-Score	Execution Time (s)
Naive Bayes	49.23	2.14
Random Forest	46.15	2.30
Decision Tree	38.10	5.18
Support Vector Machine	30.77	17.78
Logistic Regression	40.91	2.18
XGBoost	54.90	12.30
SIF	62.86	254.14
RNN-based	70.42	341.41
Attention-based	65.71	577.78
Hybrid	74.07	869.60
No. epochs for Deep Learning methods	100	

- **Performance**
  - Most ML-based methods do not result good metrics in this situation, but DL-based methods show acceptable results which is obvious effectiveness when compare to ML-based approach.
- **Training time**
  - Training time of ML-based methods is low
  - Similar to previous results, training time of DL-based methods is high

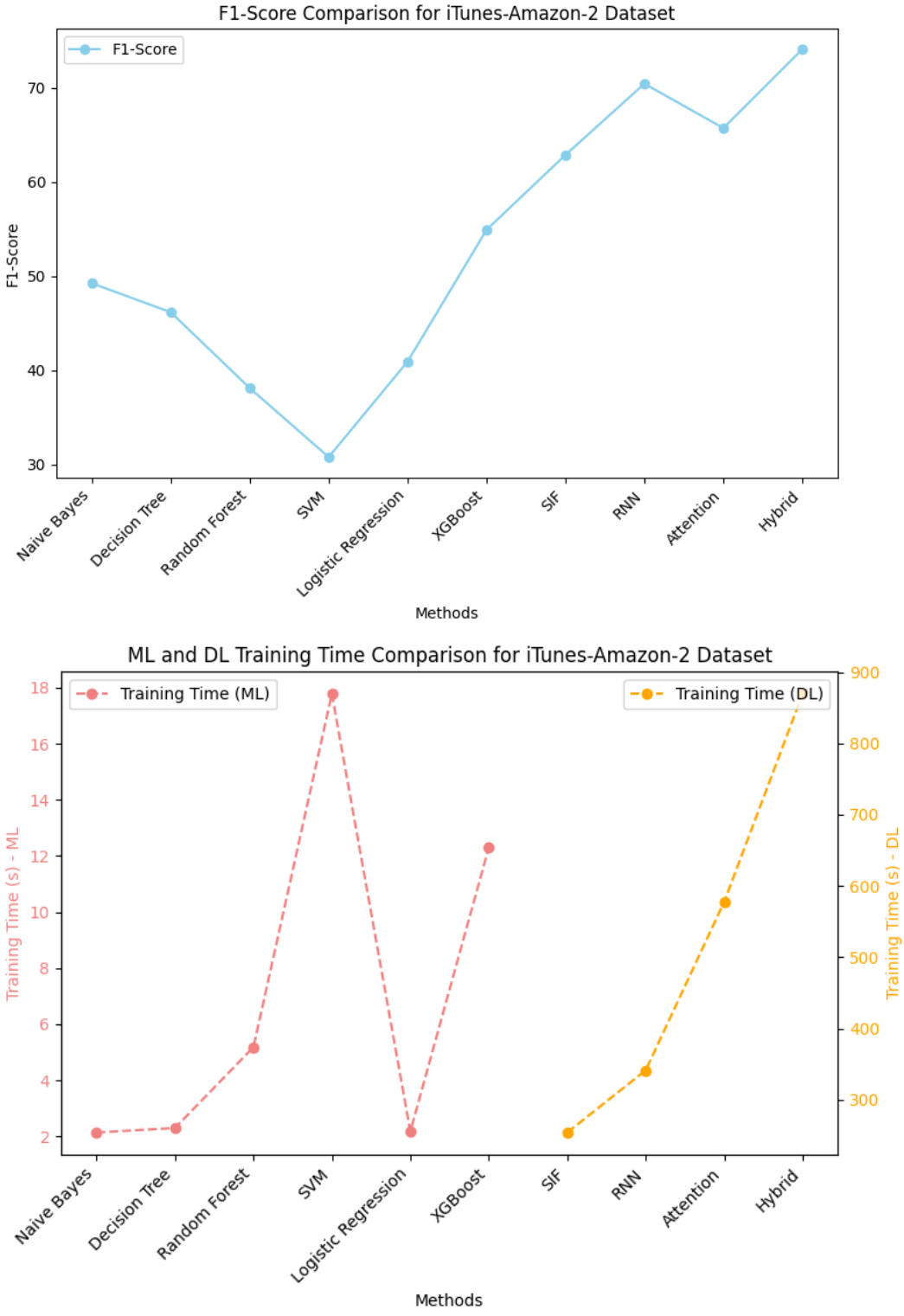

Figure 22: Dataset 4's result visualization

3.4.5 Dataset 5: DBLP-Scholar-2 (Dirty)

Table 6: Evaluation on dataset DBLP-Scholar-2

Method	F1-Score	Execution Time (s)
Naive Bayes	73.75	2.22
Random Forest	81.02	11.14
Decision Tree	84.46	57.65
Support Vector Machine	81.12	1061.55
Logistic Regression	78.15	3.15
XGBoost	85.70	33.43
SIF	81.83	1320.04
RNN-based	88.99	1835.16
Attention-based	90.37	3032.61
Hybrid	93.26	5024.61
No. epochs for Deep Learning methods	20	

- **Performance**
  - Almost methods result good metrics in this case, but DL-based methods gives better metrics in comparison with ML-based method.
- **Training time**
  - Training time of ML-based methods is low except SVM, XGBoost and Decision Tree, esspecialy SVM training time is quite highly affected by size of dataset
  - Indeed, training time of DL-based methods is still high

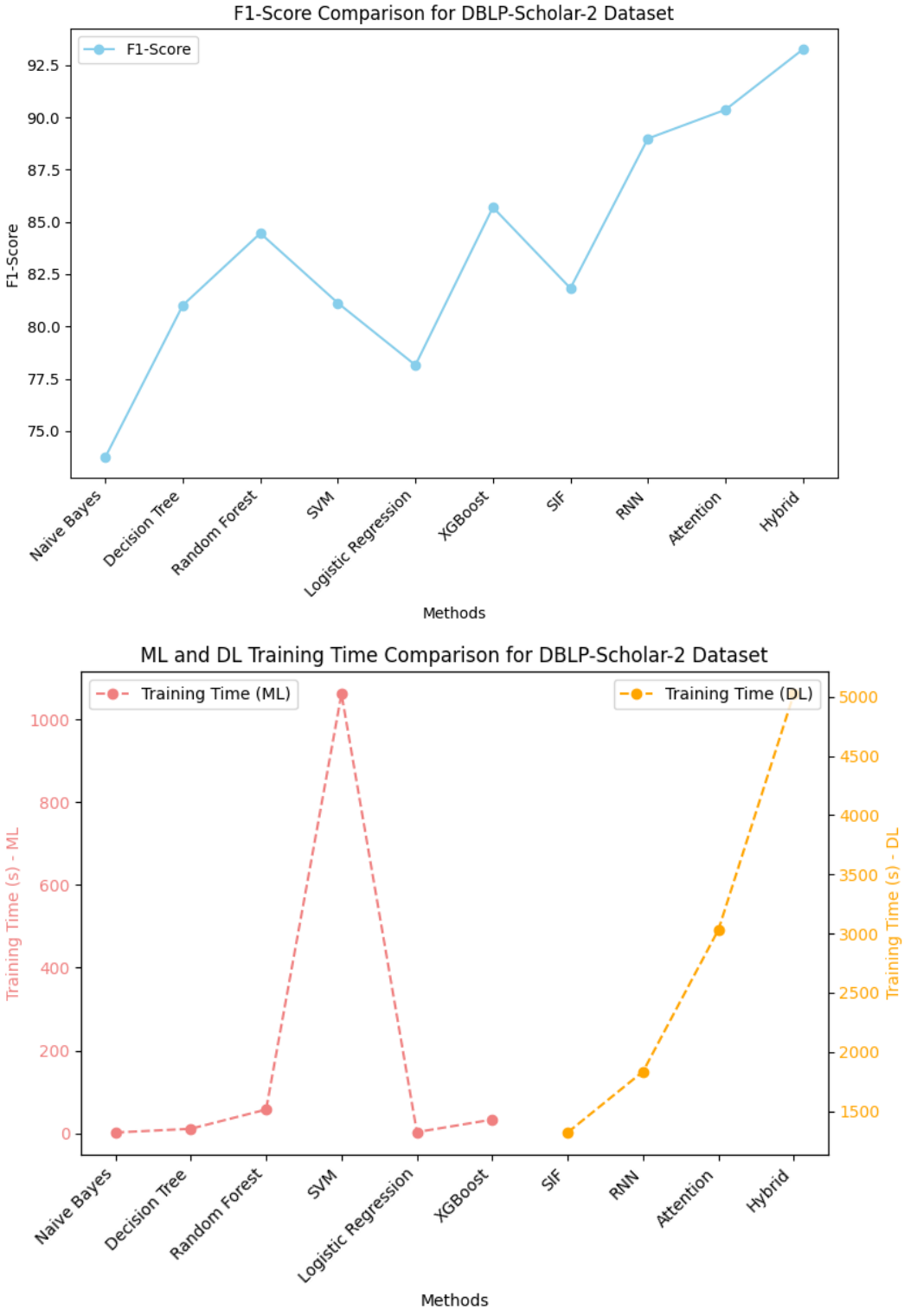

Figure 23: Dataset 5's result visualization

3.5 Discussions

- **Method evaluation:** The Hybrid method achieves the highest F1-Score but at the cost of longer training times. Deep learning methods outperform traditional machine learning methods but require more computational resources. Traditional machine learning methods like Random Forest and Logistic Regression strike a balance between performance and efficiency.
- **Number of Epochs in Deep Learning:** A higher number of epochs can contribute to better model convergence but may also increase training time.
- **Data set dependence:**
  - For structured datasets, although it takes many times more training time, the performance improvement of DL-based methods is not much compared to ML-based ones.
  - For textual and diry datasets, the DL-based method shows quite good metrics and outperforms ML-based although the time cost is quite large. Meanwhile, ML-based method brings poor performance even its still shows effectiveness in training time.
- **Trade-off between Performance and Training Time:** It is essential to discuss the relationship between performance (F1-Score) and training time. There is a trade-off between achieving high performance and incurring significant training time, which is a crucial factor when deploying models in real-world applications.
- **Consideration of Deep Learning Methods:** Deep learning methods such as RNN-based, Attention-based adn Hybrid exhibit good performance but require extensive training time. It is necessary to weigh the trade-off between performance and training resource considerations when deciding to utilize them.

4 Conclusion

In this comparative survey on machine learning-based data matching models, we explored the effectiveness of traditional machine learning methods and deep learning approaches in addressing the data matching problem. Our objective was to understand the mechanisms, processes, and architecture of deep learning implementations, which are considered the state-of-the-art (SOTA) for data matching.

Through our experiments on three different types of datasets, namely clean structural, dirty, and textual, we made several key observations regarding the performance of these models.

For clean structural datasets, we found that traditional machine learning-based baseline models were sufficient and exhibited comparable performance to deep learning models. In fact, algorithms such as Support Vector Machines (SVM), XGBoost, and Random Forest even outperformed deep learning models in certain cases. Additionally, these traditional models offered the advantage of lower execution time, making them more efficient in scenarios where time and cost are critical factors.

On the other hand, for textual and dirty datasets, deep learning-based methods demonstrated superior performance. This can be attributed to their ability to integrate natural language processing capabilities from attention-based or recurrent neural network models. These deep learning models showcased their effectiveness in handling unstructured or noisy data, where traditional machine learning models may struggle.

Based on our findings, we recommend that deep learning-based methods be applied when dealing with textual and dirty datasets, as they offer improved performance. However, for clean structural datasets, traditional machine learning models can be a viable option, providing comparable results with lower time costs.

It is important to note that the choice between traditional machine learning and deep learning approaches should be made based on the specific characteristics of the dataset and the requirements of the data matching task. Factors such as dataset complexity, data quality, and available computational resources should be considered when selecting the appropriate approach.

In conclusion, this survey sheds light on the comparative performance of machine learning-based data matching models. By understanding the strengths and weaknesses of traditional and deep learning methods, researchers and practitioners can make informed decisions when selecting the most suitable approach for their specific data matching scenarios.

References

[1] Lise Getoor and Ashwin Machanavajjhala. 2012. Entity Resolution: Theory, Practice & Open Challenges. VLDB.

[2] Wenfei Fan, Xibei Jia, Jianzhong Li, and Shuai Ma. 2009. Reasoning About Record Matching Rules. VLDB.

[3] Rohit Singh, Vamsi Meduri, Ahmed Elmagarmid, et al. 2017. Generating Concise Entity Matching Rules. SIGMOD.

[4] Pradap Konda et al. 2016. Magellan: Toward building entity matching management systems. VLDB.

[5] Parag Singla et al. 2006. Entity Resolution with Markov Logic. ICDM.

[6] Muhammad Ebraheem, et al. 2017. DeepER–Deep Entity Resolution. CoRR abs/1710.00597 (2017).

[7] Elad Hoffer and Nir Ailon. 2015. Deep metric learning using triplet network. In International Workshop on Similarity-Based Pattern Recognition. Springer.

[8] Paul Neculoiu, Maarten Versteegh, and Mihai Rotaru. 2016. Learning text similarity with siamese recurrent networks. ACL.

[9] Oriol Vinyals, Charles Blundell, Tim Lillicrap, Daan Wierstra, et al. 2016. Matching networks for one shot learning. ACL.

[10] Mudgal, S., Li, H., Rekatsinas, T., Doan, A., Park, Y., Krishnan, G., Deep, R., Arcaute, E., & Raghavendra, V. (2018). Deep Learning for Entity Matching: A Design Space Exploration. In SIGMOD'18: 2018 International Conference on Management of Data, June 10-15, 2018, Houston, TX, USA (pp. 19-34). ACM.

[11] Das, S., Doan, A., G. C., P. S., Gokhale, C., Konda, P., Govind, Y., & Paulsen, D. (n.d.). The Magellan Data Repository. Retrieved from `https://sites.google.com/site/anhaidgroup/projects/data`

[12] Anhaidgroup. (2021). DeepMatcher: Python package for performing Entity and Text Matching using Deep Learning. GitHub. `https://github.com/anhaidgroup/deepmatcher`

[13] Doan, A., Konda, P., & Sriramdasu, A. (2023). py_entitymatching (Version v0.4.1) [Computer software]. GitHub. Retrieved from https://github.com/anhaidgroup/py_entitymatching

[14] anhaidgroup. (2023). py_stringmatching. GitHub. Retrieved from https://github.com/anhaidgroup/py_stringmatching

[15] anhaidgroup. (2023). py_stringsimjoin: Scalable String Similarity Joins in Python. GitHub. Retrieved from https://github.com/anhaidgroup/py_stringsimjoin

[16] Lee, C. (2017, November 13). Understanding Bidirectional RNN in PyTorch. Towards Data Science. Retrieved from https://towardsdatascience.com/understanding-bidirectional-rnn-in-pytorch-5bd25a5dd66

[17] Srivastava, R. K., Greff, K., & Schmidhuber, J. (2015). Highway Networks. Retrieved from https://arxiv.org/pdf/1505.00387.pdf