



OTTO VON GUERICKE
UNIVERSITÄT
MAGDEBURG



FAKULTÄT FÜR
INFORMATIK

Otto-von-Guericke-University Magdeburg

Faculty of Computer Science

Department of Databases and Software Engineering

A Comparative Evaluation of Deep Learning based Transformers for Entity Resolution

Master Thesis

Author:

Mohammad Mohammadkhani

Examiner and Supervisor:

Prof. Dr. rer. nat. habil. Gunter Saake

2nd Examiner:

Prof. Dr. -ing. Ernesto William De Luca

Supervisor:

MSc. Gabriel Campero Durand

Magdeburg, 07.08.2020

Mohammad Mohammadkhani

A Comparative Evaluation of Deep Learning based Transformers for Entity Resolution

Master Thesis, Otto-von-Guericke-University Magdeburg, 2020.

Abstract

Entity resolution (ER) is the task of identifying pairs of entities in a database or groups of databases that refer to the same real-world entities. ER is a task difficult to solve with a single general solution, since the data in databases can be entered with errors or written in many different ways. Computational challenges also exist, because the number of pairs to compare for identifying all common entities, can be quadratic to the amount of data.

In recent years machine learning methods, and specially deep learning, have been used to solve the entity resolution task on relational data. A promising group of approaches work using word embeddings, a technique to represent textual information into a dense vector of numerical values. These vectors are formed in such a way that semantically similar words are mapped to similar vectors, independent of their syntax. These vectors are commonly formed by neural architectures that involve recurrent components or long short term memories, apart from established architectures like FastText, Glove and Word2Vec.

In recent years, a novel class of approaches, Transformer architectures, has been proposed to improve on traditional word embeddings. Unlike previous methods, these approaches are tailored to context in semantics. In these cases the vector representation for a word is different depending on the context that surrounds it. For example, the word solution can have different representations if it appears as the solution of a mathematical problem, or a chemical solution substance. These classes of methods have shown superior performance to traditional word embeddings on a large number of natural language processing tasks, like classification, question answering, text summarization and others. Furthermore, a large number of these methods are already available as pre-trained models to the public, in libraries that facilitate their usage. Given that to date there is still no clear study on how these contextual embeddings could contribute to ER, in this work we study their application to this task.

We specifically study BERT, DistilBERT, XLNet, and GPT2, on 2 established datasets, without using any special method for pair-formation (other than the ground truth), and using many traditional classifiers for the matching task. We find that BERT outperforms other models for our choice of application, and that logistic regression is able to perform better than other classifiers. We also identify that tuning XGBoost can lead to small improvements. In future work we propose that using the transformer architectures for the downstream classification task would also be of interest, similarly for testing with many other datasets and improving on the blocking methods.

Acknowledgements

First and foremost, I would like to express my sincere gratitude to my supervisor and mentor M.Sc. Gabriel Campero Durand, for his continuous supervision, motivation, and expert advice.

Besides his immense knowledge and work, he proved that I could not have imagined having a better mentor than him during the pandemic time. We managed to conclude this thesis research, thanks to our constructive discussions and his understanding of the situation.

I am grateful to Prof. Dr. rer. nat. habil. Gunter Saake for allowing me to conduct my thesis research under his authority and his valuable understanding throughout this thesis. The excellent research environment and the relevant ongoing research at his Database and Software Engineering Workgroup always urged me to write my thesis under his chair.

Finally, and most importantly, I would like to thank my partner, Irina, for supporting me during this time. I am incredibly grateful to my family (Pari, Abbas, Melika) for their love, caring, and sacrifices that I will never forget. I would also like to thank the Juventus football club for not letting me down when I needed extra energy for this research.

Statement of Authorship

I hereby declare that I am the sole author of this master thesis and that I have not used any sources other than those listed in the bibliography and identified as references. I further declare that I have not submitted this thesis at any other institution in order to obtain a degree.

Signature

Place, Date

Contents

List of Figures	3
List of Tables	5
1 Introduction	7
1.1 Motivation	7
1.2 Main Contributions	8
1.3 Thesis Structure	9
2 Background	11
2.1 Entity Resolution and Challenges	11
2.1.1 Entity Resolution as a Task	11
2.1.2 Challenges	15
2.2 Word-embeddings	17
2.2.1 Word2Vec	19
2.2.2 GloVe	21
2.2.3 FastText	22
2.3 Deep learning for Entity Resolution	23
2.3.1 DeepER	23
2.3.2 DeepMatcher	25
2.4 Attention and Transformers	27
2.4.1 Attention	27
2.4.2 Transformer	30
2.4.3 BERT Transformer	31
2.4.4 DistilBERT Transformer	34
2.4.5 XLNET Transformer	35
2.4.6 GPT2 Transformer	37
2.5 Summary	39
3 Design	41
3.1 Research Questions	41
3.2 Proposed Approach: Transformer-based ER	41
3.2.1 Data Pre-Processing and Vectorization	44
3.2.2 Classification	44
3.2.3 Evaluation	45
3.3 Summary	46
4 Experimental Setup	47
4.1 Classification Algorithm: XGBoost	47
4.1.1 Hyperparameters	48
4.2 Datasets	49

4.3	Experimental Environment	49
4.4	Summary	50
5	Evaluation and Results	51
5.1	RQ1: Hyper-parameter Tuning	51
5.2	RQ2: Comparing the Performance of Transformers	55
5.3	RQ3: Comparing the Performance of Classifiers	59
5.4	Summary	61
6	Related Work	63
6.1	Entity Resolution Approaches	63
6.2	Applying Deep Learning for ER	63
6.3	Works on Transformer Architecture	64
7	Conclusion and Future Work	67
7.1	Conclusion	67
7.2	Future Work	67
	Bibliography	69
	Appendices	81
A.	Prototypical Implementation of a Transformer Model for Entity Resolution	81
B.	Solving Transformers' Token Limitation Problem	84
C.	Loading Various Transformers with Hugging Face Library	85

List of Figures

2.1	Entity resolution example.	12
2.2	The workflow of first and second ER generations.	13
2.3	The workflow of third ER generation.	14
2.4	Entity resolution challenges depicted in [PIP20].	16
2.5	Bag of words example.	18
2.6	Visualization of Word2vec output for the Amazon-Google dataset.	19
2.7	Word2Vec CBOW vs Skip-gram proposed by [MCCD13].	20
2.8	Co-occurrence probabilities for target words in GloVe presented by [PSM14].	21
2.9	Model architecture of FastText for a sentence with N ngram features pre- sented by [JGBM16].	22
2.10	Deep entity resolution framework (DeepER) presented by [ETJ ⁺ 17].	24
2.11	RNN with LSTM in the hidden layer.	24
2.12	DeepMatcher framework presented by [MLR ⁺ 18].	26
2.13	Self-Attention example	28
2.14	Sequence-to-sequence model with attention	29
2.15	Transformer architecture proposed by[VSP ⁺ 17]	30
2.16	BERT input representation introduced in [DCLT18]	32
2.17	BERT bidirectional approach to predict the token.	33
2.18	BERT language modeling with masked token prediction simplified by [Ala18].	33
2.19	Knowledge distillation challenge in an experiment done by [BS20].	35
2.20	Illustration of the permutation language modeling objective for prediction a token [YDY ⁺ 19].	37
2.21	Self-Attention vs. Masked Self-Attention.	38
2.22	GPT-2 transformer architecture simplified by [Ala19].	39
3.1	Design of the proposed approach for the task of entity resolution.	42
3.2	Overview of the tokenization and word embedding step in a transformer architecture.	45
5.1	Overview of hyper-tuned results on Amazon-Google dataset.	53
5.2	Overview of hyper-tuned results on DBLP-ACM dataset.	54
5.3	Box-plot overview of different transformers performance on Amazon-Google dataset.	55
5.4	Box-plot overview of different transformers performance on DBLP-ACM dataset.	56
5.5	Comparison of different transformers performance on Amazon-Google dataset.	57
5.6	Comparison of different transformers performance on DBLP-ACM dataset.	58
5.7	Comparison of different classifiers performance on Amazon-Google dataset.	60
5.8	Comparison of different classifiers performance on DBLP-ACM dataset. . .	61

List of Tables

- 4.1 Overview of datasets. 49
- 5.1 Effect of hyper-parameter tuning for all datasets based on the accuracy. . . 52
- 5.2 F1 scores of transformers for Amazon-Google dataset. 56
- 5.3 F1 scores of transformers for DBLP-ACM dataset 57
- 5.4 F1 scores of classifiers for Amazon-Google dataset. 59
- 5.5 F1 scores of classifiers for DBLP-ACM dataset. 59
- 6.1 Comparing the best performing transformer model with Magellan and
DeepMatcher 64

1 Introduction

In this chapter, we present our work’s motivation with a short overview in the field of entity resolution and the emergence of deep learning that inspired our research. We also set our research aims, and outline the structure of this thesis. This chapter is structured as follows:

- In [Section 1.1](#), we review some of the essential works in the field of entity resolution through history.
- In [Section 1.2](#), we highlight our research’s main contributions.
- We then close this chapter with the structure of subsequent sections in [Section 1.3](#).

1.1 Motivation

Humans always struggled to find and match records that might refer to the same object for decades. Historically, the first difficulties emerged when researchers in the health sector required to link patient medical records from one or different data resources. The task needed to compare quasi-identifying information such as first name, family name, date of birth, place of birth, and other information from a single record over large stacks of paper[[Chr12](#)]. Expanding its application domains from traditional use in the health sector and national security (law enforcement, and anti-terrorism [[RMD13](#)]), to the de-duplication of records in companies, *Entity Resolution* (ER), as the task of determining if two data instances refer to the same real-world object, is already a long-standing challenge for researchers and practitioners alike. Furthermore, entity resolution is a crucial task for data integration and data cleansing, not to mention for improving the quality of downstream analytical tasks or the building of machine learning models over the data [[RD00](#)],[[HS95](#)],[[CKM00](#)].

Classical methods for ER assume that there exists an appropriate distance function between pairs of records [[GM12](#)]. Over the past few decades, rule-based approaches [[LLG14](#)] and hand-crafted heuristics have been crucial in dealing with traditional entity resolution challenges like name ambiguity, missing values, abbreviations, data formatting, and errors due to data entry [[HSW07](#)]. Rule-based approaches are best suited for input fields that contain controlled and well-structured information, and it can be problematic if the desired input field needs a more significant number of rules [[AW18](#)]. While those efforts had been deployed in very specific domains with success, the alternative of treating entity resolution as a match/non-match binary classification problem and seeking to solve it using machine learning algorithms is becoming a widespread practice over the last decade. Pairwise matching using supervised machine algorithms like decision trees[[CKLS01](#)], conditional random fields [[MS04](#)], ensembles of classifiers [[YXCJJ⁺17](#)], and support vector machines [[BM03](#)] can improve matching results. Holistic machine learning solutions like Magellan

[KDSG⁺16] provide guidelines, tools, and libraries for the whole entity matching process with good results on structured and semi-structured data.

Given the large amount of data produced every day, corporations and large institutions face a more significant challenge that demands ER. Increasing the number of datasets with unclean, unstructured, and incomplete data yields erroneous outcomes. Today’s data can consist of large textual examples such as trip advisor reviews, or product descriptions, which tend to reduce the classification accuracy depending on the choice of representation, and might require considerable efforts in hand-crafted features to represent the textual data [BS19]. Because of recent advances in deep learning in natural language processing, some authors [ETJ⁺17] propose a deep learning structure to tackle these issues. The authors of *DeepER* apply a bidirectional LSTM and word embeddings that are shown to surpass the state-of-the-art models on multiple benchmark datasets. In following work, the authors of *DeepMatcher* [MLR⁺18] present another deep learning architecture using an attention mechanism on top of an LSTM, showing also promising results.

Attention (first proposed in [VSP⁺17]) is an ubiquitous technique that allows deep learning models to focus on relevant parts of an input sequence, as needed. Such kind of method helps the model to deal with long sentences, which is a considerable boost in terms of the performance of neural machine translation applications.

While deep learning architectures already deliver massive performance accuracy, utilizing modern attention-base methods as in *Transformer* architectures, as the core of the entity resolution system has received limited attention. Transformers as a modern deep learning model inspired by attention mechanisms, do not require recurrent sequential processing of data to achieve high accuracy. Relying on attention’s power, transformers can be trained more efficiently on larger amounts of data. Brunner and Stockinger [BS20] recently proved that most transformers outperform traditional deep learning approaches in the task of entity matching.

In this thesis, we pursue studying four of the most recent attention-based transformer architectures performance on the task of entity resolution on benchmark datasets. While we compare their results with the state-of-the-art approaches, we investigate the effect of tuning hyper-parameters on the model. Conclusively, we analyze the performance of different classifiers on the classification layer of transformer models.

Herewith, we have outlined the scope of our research; the main contributions of this thesis are summarized in the following section.

1.2 Main Contributions

1. We review the role of machine learning approaches on entity resolution, while the majority of the work is dedicated to deep learning proposed approaches. In particular, we discuss attention-based transformers than can be advantageous in improving model performance.
2. We present our design consisting of a chosen transformer model and a chosen classifier to ensure better experimental results.

3. We conduct a series of experiments on four state-of-art transformers: BERT, DistilBERT, GPT2, and XLNet, and report the performance of selected architectures on specific datasets.
4. We investigate hyper-parameter tuning on the classifier given the pre-trained transformer results and indicate the best configuration based on grid search.
5. We inquire into the most common classifiers for binary classification: XGBoost, GBM, k-nearest neighbors (KNN), and Logistic regression, and report the performance on the training set of match and non-match records as represented by the transformers.
6. We outline some promising directions for future research that should improve our understanding of existing attention-based transformers and enhance their introduction in real-world applications like ER.

1.3 Thesis Structure

The rest of the thesis is structured as follows:

- In [Chapter 2](#), we explain some fundamental concepts of entity resolution tasks and challenges we face in the big data era. We review word embedding and deep learning architectures. More importantly, we study transformer approaches to address the bottleneck problem.
- [Chapter 3](#) formulates the goals of this thesis. We cover our specific research questions and propose our design, including data pre-processing, classification, and evaluation choices to be made.
- In [Chapter 4](#), We review such essential components of the experimental setup like datasets selected, configurations, and programming frameworks to achieve the same results. We also explore xgboost, our chosen classifier, and its parameters that we seek to tune in [Section 5.1](#).
- [Chapter 5](#) is dedicated to reviewing and discussing evaluation results obtained by the four transformers architectures, and performing a comparison of different classifiers.
- In [Chapter 6](#), we revisit some previous research related to the evaluation of machine learning algorithms and applying deep learning concepts in entity resolution.
- Finally, we conclude this thesis and outline some intriguing directions for future research in [Chapter 7](#).

2 Background

This chapter focuses on understanding the background of this thesis and provides an overview of the essential topics covered in the thesis, to create a solid foundational understanding of the research area. This chapter is structured as follows:

- In [Section 2.1](#), we overview entity resolution as a task and provide a timeline discussing different ER approaches through time. Eventually, we present the major challenges we face in the entity resolution process.
- [Section 2.2](#) investigates word embeddings techniques, which became a crucial part of applying deep learning methods on entity resolution.
- In [Section 2.3](#), we discuss two prominent deep learning architecture for entity resolution tasks with more details about their core ideas, and architecture.
- In [Section 2.4](#), we overview the concept of attention mechanism and transformer models, which inspired this thesis. Also, we provide an in-depth overview of the four state-of-the-art transformer architectures we used in this research.
- In [Section 2.5](#), we summarize this chapter.

2.1 Entity Resolution and Challenges

Entity resolution (ER) is the problem of correctly identifying multiple, and possibly differing representations of unique real-world entities in data. While it benefits from simple formulation, many challenges arise that need to be tackled. In this section, we provide an in-depth overview of the entity resolution task and challenges that may arise.

2.1.1 Entity Resolution as a Task

In the following subsection, we describe what entity resolution is and why it is a critical task in many applications. Before we begin, it is important to note that terms such as Record Linkage [FS69], Reference Reconciliation [DHM05], Merge-Purge [HS95], and Entity Matching [KR10] refer to Entity Resolution in the literature, as well.

Entity Resolution Definition

Entity Resolution (ER) is the task of determining whether two data instances refer to the same real-world object within a knowledge base or across multiple knowledge bases. The term *Entity* historically refers to people, such as patients, customers, or taxpayers, but now it can also refer to publications, citations, consumer products, or businesses. In other definitions, ER is described as a task of identifying, matching, and merging records that correspond to the same entities from several databases [BGMM⁺09]: Let T be a set of entities with n tuples and m attributes $\{A_1, \dots, A_m\}$. Note that these entities can come from one table or multiple tables (with aligned attributes). We denote by $t[A_i]$ the value of attribute A_i on tuple t . The problem of entity resolution (ER) then is, given all distinct tuple pairs (t, t') from T where $t \neq t'$ to determine which pairs of tuples refer to the same real-world entities (match) [ETJ⁺17]. Entities are described in terms of their characteristics, called *Attributes*. The values of these attributes provide information about a specific entity. A *Reference* is a collection of attributes values for a particular entity [Tal11]. When applied to one database, this process is called *Deduplication* [MB12]. Figure 2.1 shows an example of the entity resolution dilemma portrayed by [CES15a].

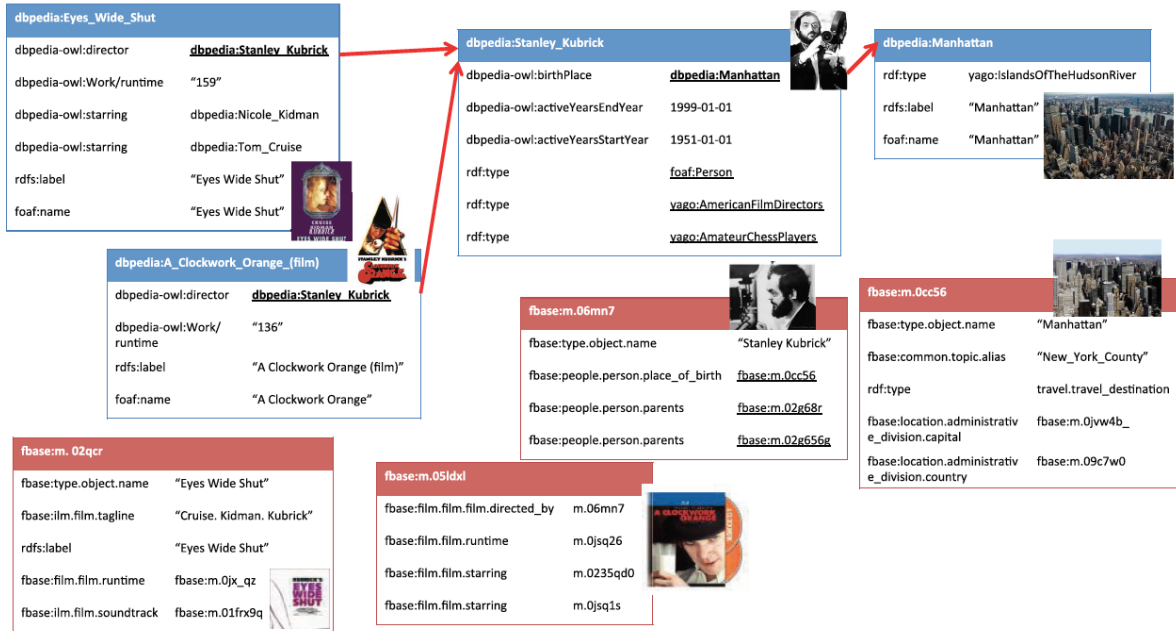


Figure 2.1: Entity resolution example.

DBpedia dataset describes two movies: *A Clockwork Orange* and *Eyes Wide Shut*, Stanley Kubrick as director and Manhattan as Kubrick's place of birth, while Fbase includes different descriptions for the mentioned entities. Sharing a similar structure between descriptions referring to *Eyes Wide Shut* can be concluded as possibly of a match. However, this is not the case when comparing Stanley Kubrick's entity's descriptions, which includes different attributes. To decide whether the descriptions of Stanley Kubrick in the DBpedia and Freebase match, we need to consider the similarity of descriptions related to them. Entity resolution is an essential component of many real-world applications. The *National Census*, which is based on collecting data about various aspects of the population, culture, economy, and the environment in their respective countries, practice ER to reduce the costs and efforts required to conduct large-scale census collections [Gil]. The

Health Sector has been a second application area that has established methods for data matching for several decades [Dun46]. With increasing importance of *National Security*, many countries apply entity resolution methods to identify and track suspicious transactions and records of terrorists' communications and activities in the online information space [Fie05]. *Crime and Fraud Detection* is another field where ER is crucial for identifying modified or fictitious personal details of criminals [CCX⁺04]. *Online shopping* or *E-commerce*, where shopping engines attempt to distinguish which product descriptions in different online stores do correspond to the same item, can also benefit from ER techniques [BBS05]. Furthermore, entity resolution is an integral part of *Data Cleaning*, which simply is the process of detecting and correcting (or removing) unreliable or inaccurate records from a data resource [RD00]. ER is an essential tool for *Text Mining* for creating knowledge bases [ZWYL14]. Lately, authors of [CLD⁺16] deploy entity resolution methods for *Social Media Analysis*.

Entity Resolution Through Generations

Entity resolution is becoming an influential discipline in computer science, and so far, numerous ER approaches have been developed through time. Papadakis et al. [PIP20] define four entity resolution generations, based on the challenges tackled in each group of approaches. These generations are defined as follows:

- *Veracity*. The earliest entity resolution approach was mainly focused on noise in attribute values of entity profiles in a presumably fixed schema (relational) database. The assumption of a known schema allowed experts to develop solutions that addressed the *Veracity* problem. To address veracity challenges such as inconsistencies, entity profile errors introduced by manual data entry [EIV06], and noise, first-generation ER solutions followed an end-to-end workflow depicted in Figure 2.2. *Schema Matching* as the first step of the workflow is an arbitrary choice in the ER process. Targeting clean-clean ER, schema matching focuses on identifying attributes with the same meaning [BMR11]. Finding semantically identical attributes (such as job and profession) facilitates the schema-aware functionality of the following workflow steps. Schema matching methods can be classified into structure-based, instance-based and hybrid solutions. While structure-based methods transform schema to a graph or a tree based on topological relationships between attributes, apart from using similarity of the attribute names, an instance-based method considers the distribution of attribute values to identify similar attributes. Hybrid-based methods are a combination of more than one method at the same time. The second step of the workflow, called *Blocking*, is a mandatory step that



Figure 2.2: The workflow of first and second ER generations.

addresses the quadratic problem of ER, which would result in comparing each entity with all others. The blocking step reduces the number of comparisons by grouping similar entities inside blocks in such a way that the quality of the further process

is not deteriorated if we compare only entities within a block to each other. The first generation blocking places entities into blocks based on the signature, which is called blocking keys (e.g., the first two letters as key). Blocking methods can be distinguished by key type, redundancy awareness, matching awareness, and key selection choices. Standard Blocking method proposed by [FS69], deploy hash-based key, static matching, and non-learning selection. While the Sorted neighborhood method [YLKG07] proposed a sort-based key type for blocking, Duplicate Count Strategy [DNSW12] took the change in block structure (dynamic) into account. *Entity Matching* as the third step of the workflow, executes the comparison between the blocks we created by applying different string similarity measures to the values of selected attribute names and it outputs a similarity graph. Rule-based matching methods mostly deploy a similarity threshold to assign the entity pairs into the match and non-match categories. Other family of approaches known as collective methods, propagate the decision to neighbouring entities. Learning-based matching approaches use labelled instances to extract effective rules through a machine learning algorithm to improve the performance. *Clustering* as the last step of the workflow. It partitions the matched pairs into equivalence clusters, which then are solved to actual entities.

- *Volume and Veracity.* While addressing veracity and searching for high accuracy performance, the second generation of ER tackles the *Volume* with (tens of) millions of structured entity profiles. The same assumption of known schema and workflow applies at the core of this generation. This challenge addressed by parallelization methods such as map/reduce. So far, no parallelization method has been adopted on schema matching, but there are several papers on blocking and entity matching inspired by map/reduce paradigm that can scale to large datasets.
- *Variety, Volume and Veracity.* With the surge of large heterogeneous schemata, the third generation of ER solutions with different workflow and has been born to tackle *Variety*. The challenge of third-generation ERs arises in user-generated web data, which consists of a large amount of semi-structured datasets, and users are allowed to add attribute values and create different schemata. Also, due to producing datasets by automatic information extraction techniques, the data is noisy and can be consist of missing values. The end-to-end workflow implemented by the third

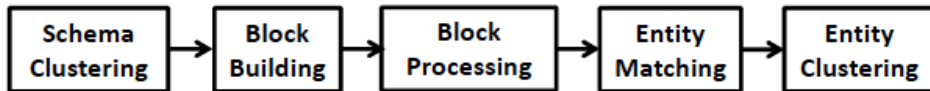


Figure 2.3: The workflow of third ER generation.

generation of entity resolutions solutions is depicted in Figure 2.3. Unlike the first two generations, schema matching is not applicable in third-generation solutions. Instead, a new step called *Schema Clustering* is developed, which clusters attributes based on their syntactic similarity, regardless of their semantics. Attribute clustering proposed by [PIP⁺12] is a common practice concerning this step, which facilitates the following steps. Regarding blocking, a new action called *Block Building* is used to create blocks considering all attribute values, regardless of all attribute names (schema-agnostic approach). Token Blocking proposed by [PINF11] is the most

common and most straightforward approach of block building, which extracts all tokens contained in each entity profile’s attribute values and creates one block for every one of them. The number of resulting blocks outputted from Block Building, and as a result, the number of comparisons is large. To tackle this issue, a new additional step is defined called *Block Processing*, which reconstructs the blocks in a way that enhance the precision without any significant recall loss. To this end, it refines the block’s structure by discarding redundant comparisons (overlapped blocks) and superfluous comparisons (comparison between non-matches). The generic approach of Block Processing techniques includes assigning a matching likelihood score to each item and removing the items with low costs. Block processing techniques are distinguished by two categories: Block-centric methods which focus on the large-grained level of entire blocks (Block Purging [PIN⁺12], Block Filtering [MT13], Block Clustering [NMMMBLP07]) and Comparison Cleaning methods that operate at the close-grained level of individual comparisons (Meta-blocking [PPPK16] and Blast [SBJ16]). Entity Matching methods execute one new blocks. Most of these techniques follow the same approach, which starts with reliable seed matches, propagates the similarity of matches into neighbors, updates the neighbors’ similarity, sorting them by descending order, and labels the best pair as matched. This process, which recomputes again, is a core method of most proposed entity matching techniques such as SiGMa [LJPD⁺13], PARIS [SAS11], LINDA [BDMNW12]. Entity Clustering is the last step of the workflow, which transforms the given similarity graph into the outcome of entity resolution.

- *Velocity, Variety, Volume and Veracity.* While all previous entity resolution generations involved batch processing, aiming to perform as fast as possible, they do not involve any actual time constraint. The fourth generation of entity resolution additionally addresses the *Velocity* problem, which refers to applications with time constraints. Entity resolution techniques in this context are distinguished by Progressive ER, which deals with applications with loose time constraints (e.g., hours, minutes) and Real-time ER, which deals with strict time constraints (e.g., seconds). The end-to-end workflow of this generation modifies the previous workflow by adding a Prioritization step that schedules the processing of matching and clustering steps, such that entities that are important from the application perspective are resolved before others.

2.1.2 Challenges

The task of identifying and matching records that refer to the same entities within one or across several databases is challenging for several reasons. In this section, we discuss the traditional and major challenges we face in the entity resolution process. Entity resolution is not merely done by looking at string similarities, and it can become complicated by traditional challenges. Some of these challenges are as follows:

- *Name/Attribute Ambiguity.* It refers to vagueness in names or attributes in the real world. Different people can share similar names such as Michael Jordan as a basketball player and Michael Jordan as an American scientist. In the world of football, a Serbian team called FK Partizan has two Nemanja Miletićs in their

squad. Researchers from some countries often share similar names, and it is not uncommon for different scholars to share exactly the same name.

- *Missing Values.* Most of the real-world datasets include records with missing values. For example, a list of Amazon products consists typically of records with no specific price or manufacturer. Several papers propose methods such as list-wise deletion, pair-wise deletion, mean substitution, or regression imputation [TKF11] to handle this problem.
- *Noisy Data.* Noise is an inescapable part of entity resolution tasks, which can be a result of errors due to data entry, extraction/parsing errors, the use of non-standard encodings or characters, and the use of alternative descriptions. As shown in Figure 2.4, the zip code value in a pair of entities is suffering from noise.

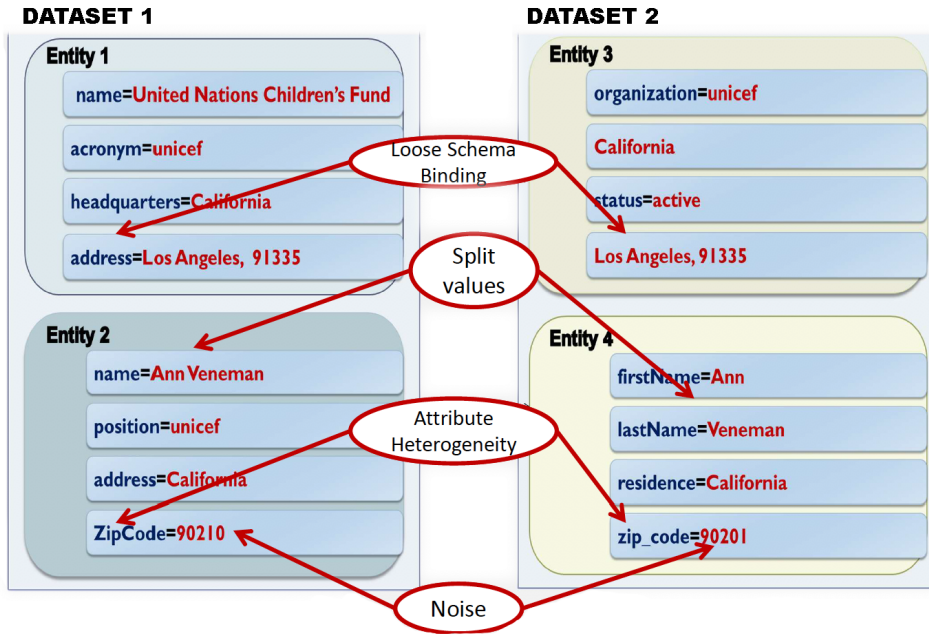


Figure 2.4: Entity resolution challenges depicted in [PIP20].

- *Split Values.* Some attributes can be a combination of more than one attribute. For instance, the *name* attribute in one dataset is broken into two attributes *first name* and *last name*.
- *Attribute Heterogeneity.* An attribute can be written in different forms. For example, two different ways of labeling the *zip code* attribute, as shown in Figure 2.4, are *ZipCode* and *zip_code*.
- *Loose Schema Binding.* Since the schemata describing entities may range from locally defined to pure tag-style annotations.
- *Class Imbalance.* Data are said to suffer the class imbalance problem when the class distributions are highly imbalanced, this of course would affect machine learning algorithms and sampling-based analysis using that data. Since most comparisons in entity resolution are between non-match records, class imbalance problem is unavoidable, as there is a greater number of possible non-match records as there is of actual match records.

- *Lack of Unique Entity Identifiers and Data Quality.* Generally, the databases to be matched (or deduplicated) do not contain unique entity identifiers or keys. Even when entity identifiers are available in the databases to be matched, one must be absolutely confident in the accuracy, completeness, and consistency over time of these identifiers, because any error in such an identifier will result in wrongly matched records [CES15b].
- *Computation Complexity.* Entity resolution is a quadratic problem, as every entity has to be compared with all others, which makes the process very expensive in terms of time and resources.
- *Lack of Training Data Containing the True Match Status.* In many data matching applications, the true label of two records that are matched across two databases is not known, i.e., there is no ground-truth data available that specifies if two records correspond to the same entity or not.
- *Privacy and Confidentiality.* Data matching can rely on personal information such as names, addresses, and dates of birth of individuals. For this reason, privacy and confidentiality need to be carefully considered.

2.2 Word-embeddings

To process words by machine learning models, they need some form of numeric representation (feature extraction). In this section, we shortly review the most famous traditional word to vector representations and then, discuss the state of art word embedding methodologies and their drawbacks, which lead us to *Transformer Language Models*. In the following, we discuss some of the most commonly used vector representations.

Bag of Words or One Hot Encoding

A widespread feature extraction procedure for sentences and documents is the *Bag-of-words* approach (BoW). In this approach, we look at the histogram of the words within the text, i.e., considering each word count as a feature[Gol17]. The bag-of-words model is a simplifying representation often used in natural language processing and information retrieval. Based on this approach, each element in the vector corresponds to a unique word in the corpus vocabulary, and depending on the appearance of the token at a particular index, the number of element changes. BoW is commonly used in methods of document classification where the frequency of occurrence of each word is used as a feature for training a classifier[SIFL16].

In Figure 2.5, our corpus example consists of every unique word across the documents. The BoW representation for the sentence “the dog is on the table” is shown in the picture where we can see each element of the vector corresponds to the number of times that particular word occurs in the sentence. While it is simple to understand and implement, Bag-of-words does not encode any meaning or word similarity.

the dog is on the table



Figure 2.5: Bag of words example.

TF-IDF Representation

TF-IDF is short for term frequency-inverse document frequency. It is a statistical measure used to evaluate how important a word is to a document in a collection of documents or corpus[RU11]. The importance is defined proportionally by the number of times a word appears in the document and is offset by the number of documents in the corpus that contain the word. The TF-IDF score is defined by two factors: term frequency and inverse document frequency. There are several ways of determining the exact values of both statistics. We present the most common one:

- *Term Frequency* (TF). is a scoring of the frequency of the word in the current document. Due to differences in each document's length, the term frequency is divided by the document length to get a normalized value.

$$TF(t) = \left(\frac{\text{Number of times term } t \text{ appears in a document}}{\text{Total number of terms in the document}} \right)$$

- *Inverse Document Frequency* (IDF). is a scoring of how rare the word is across documents. It is the logarithmically scaled inverse fraction of the documents that contain the word.

$$TF(t) = \left(\frac{\text{Total number of documents}}{\text{Number of documents with term } t \text{ in it}} \right)$$

While the TF-IDF approach provides weights to different words by their importance, it is still unable to capture the word meaning.

Word Embedding

Both of the mentioned representations cannot deal with the word meaning. Besides, BoW and TF-IDF suffer from the sparse representation problem: when the vocabulary size increases, the vector representation of documents increases as well, and this results in a vector with lots of zero scores, called a sparse vector or sparse representation problem. *Word Embedding* as distributional similarity-based representations tends to solve both weaknesses[LC13]. Word embedding is a set of language modeling and feature learning techniques in natural language processing (NLP) where words or phrases are mapped to dense vectors of real numbers[MSC⁺13][LXT⁺15].

In other words, a word embedding is a learned representation (real-valued vectors) for text where words with the same meaning have a similar representation. The simplest way to think about the addition and subtraction of words like vectors is with this famous example: $\text{king} - \text{man} + \text{woman} = \text{queen}$. Word embedding idea of using a densely distributed representation for each word leads to reducing the dimensionality, which was a problem in traditional representations. Furthermore, using contextual similarity help word embedding to achieve a more expressive representation. In the following, we overview the most popular word embeddings.

2.2.1 Word2Vec

Word2vec is a group of models based on [MSC⁺13],[MCCD13] that are used to compose word embeddings. These models are built on neural networks that have at least two layers and process text by “vectorizing” words. They are trained to build semantic contexts of words. Word2vec takes as its input a large corpus of text and produces a vector space, typically of several hundred dimensions, with each unique word in the corpus being assigned a corresponding vector in the space. Due to its application in deep neural networks understanding, Word2vec became a popular approach to make a numerical form. The purpose of Word2vec is to group vectors of words together based on their similarity[BDVJ03]. Similarity measured mathematically and creates numerical representations of word features such as the context of individual words. Given enough data and contexts, Word2vec can make extremely accurate predictions about a desired word’s meaning based on its appearances. These predictions are used to cluster and classify word’s association with other words by topic, and it can be beneficial in many applications such as sentiment analysis, recommender systems, or customer relationship management. Some of Word2vec’s concepts effectively create recommendation engines used in companies such as Alibaba[WHZ⁺18], Airbnb[GC18], and Spotify[JMN⁺16].

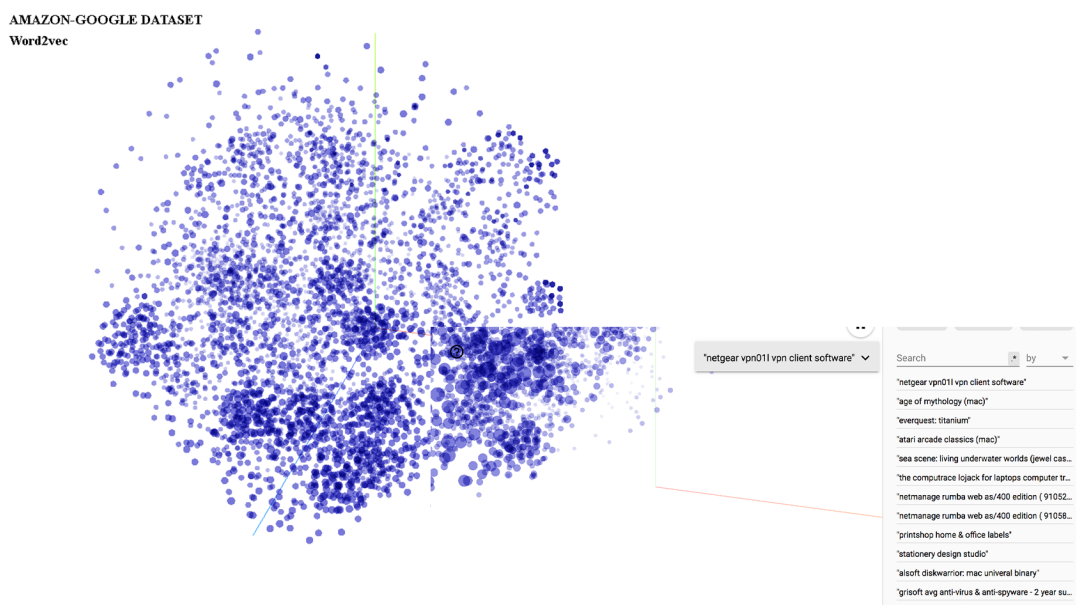


Figure 2.6: Visualization of Word2vec output for the Amazon-Google dataset.

Figure 2.6 visualizes Word2vec output for the Amazon-Google dataset, one of the datasets

we use for the entity resolution task. We used a sampling of the mentioned dataset to grasp initial information about our data. We deployed the *Embedding Projector* provided by google¹ to visualize word embeddings. As Figure 2.6 shows, words that share common contexts in the corpus are located close to one another in the space. A view on some specific data points shows a set of supposedly software products share similar representations on our visualization. The authors of Word2vec introduced two architectures which can be useful for any specific task: Continuous Bag of Words (CBOW) and Skip-Gram. Figure 2.7 shows the difference between CBOW and Skip-Gram architectures. The general approach for Word2vec is based on given text data and given window size, generating a training set to predict the target.

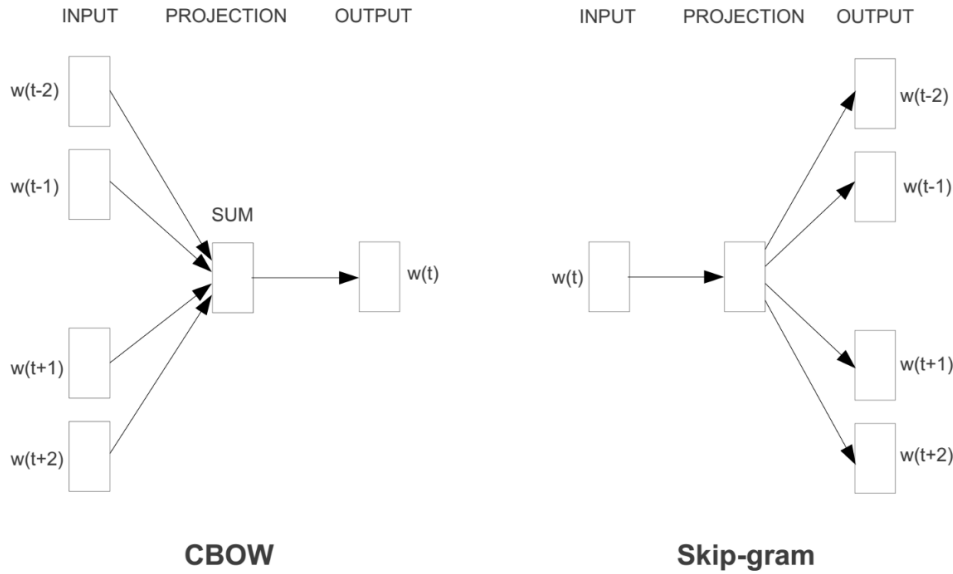


Figure 2.7: Word2Vec CBOW vs Skip-gram proposed by [MCCD13].

- *Continuous Bag of Words* [MCCD13]. The CBOW model learns the embedding by predicting the current word based on its context (surrounding words). Considering a simple sentence, “African elephants have less hair than Asian elephants”, this can be pairs of (context window, target word). Where if we consider a context window of size 3, we have examples like ([less, hair, Asian], than), ([African, have, elephants], less), ([less, elephants, hair], have) and so on. Thus the model tries to predict the target word based on the context window words.
- *Skip-Gram* [McC16]. The Skip-Gram model learns by predicting the surrounding words (context) given a current word. This model takes a center word and a window of neighbor words and try to predict the context words for some window size around the center word. Tackling this problem, the model defines a probability distribution, and once predicting the surrounding words is done, it uses the hidden layer to get the word vectors. Given the small initialization of word vectors, the predictive model learns the vectors by minimizing the loss function[GAL⁺06]. Applying Skip-gram on the same example sentence with a context window of size 4, our training set on specific word includes examples such as ([elephants], African), ([elephants], less), ([elephants], have). Next, the window slides and fill up the training

¹<https://projector.tensorflow.org>

set with comparing the other word against the context words. The Word2vec model learns the vectors with a feed-forward neural network with a language modeling task (predicting the next word) and optimization techniques such as *Stochastic gradient descent* [LW18].

Using word embeddings produced by Word2vec is helpful for many tasks, but they fail to capture higher-level concepts such as long-term dependencies, anaphora [SPCT20], and negation.

2.2.2 GloVe

GloVe is short for "Global Vectors" and is developed by Pennington et al. [PSM14] in response to lack of transparency in Word2vec regularities. The proposed word vector technique was motivated by the idea that context window-based methods suffer from the disadvantage of not learning from the global corpus statistics. The authors of GloVe highlight focusing on the co-occurrence probabilities of words within a corpus of texts in order to embed them in meaningful vectors. In other words, GloVe looks at how often a word 'j' appears in the context of a word 'i' within all our corpus of texts. Therefore, the advantage of GloVe is that, unlike Word2vec, GloVe does not rely just on local statistics (local context information of words), but incorporates global statistics (word co-occurrence) to obtain word vectors. GloVe training is performed on aggregated global word-word co-occurrence statistics from a corpus. Furthermore, GloVe leverages both *Global Matrix Factorization* methods like Latent Semantic Analysis (LAS) [LFL98] for generating low-dimensional word representations and *Local Context Window* methods such as the Skip-Gram model [GAL⁺06].

GloVe uses a specific weighted least squares model to train on global word co-occurrence counts to make efficient use of statistics. Consider two words $i=ice$ and $j=steam$ in the context of Thermodynamics domain. The relationship of these words can be examined by studying the ratio of their co-occurrence probabilities with various probe words k .

$$F(w_i, w_j, \tilde{w}_k) = \frac{P_{ik}}{P_{jk}}$$

Figure 2.8 shows the co-occurrence probabilities for target words *ice* and *steam* with selected context words from a 6 billion token corpus. Probe words like *water* or *fashion* that are either related to both *ice* and *steam*, or to neither, the ratio should be close to one. Probe words like *solid* related to *ice* but not to *steam* will have considerable value for the ratio. Compared to the raw probabilities, the ratio is better able to distinguish relevant words (*solid and gas*) from irrelevant words (*water and fashion*), and it is also better able to distinguish between the two relevant words. While GloVe adds more realistic meaning

Probability and Ratio	$k = solid$	$k = gas$	$k = water$	$k = fashion$
$P(k ice)$	1.9×10^{-4}	6.6×10^{-5}	3.0×10^{-3}	1.7×10^{-5}
$P(k steam)$	2.2×10^{-5}	7.8×10^{-4}	2.2×10^{-3}	1.8×10^{-5}
$P(k ice)/P(k steam)$	8.9	8.5×10^{-2}	1.36	0.96

Figure 2.8: Co-occurrence probabilities for target words in GloVe presented by [PSM14].

into word vectors by considering the relationships between word pairs, It cannot solve higher-level concepts challenges. Due to training on the co-occurrence matrix of words, GloVe takes significant memory for storage.

2.2.3 FastText

FastText is a library² created by the Facebook research team for efficient learning of word representations and sentence classification. While being an extension of the Word2vec model, FastText treats each word in a corpus differently. Dealing with long-standing challenges such as computing the word representations of text data generated by billions of users inspired the Facebook research team to create their own library for word representation and text classification [JGB⁺16]. The FastText algorithm is essentially based on related work[BGJM17], [JGBM16]. Both Word2vec and GloVe models treat each word like an atomic entity and generate a vector for each word. Contradictorily, as Figure 2.9 depicts, FastText deals with each word as composed of character n-grams. Hence, the

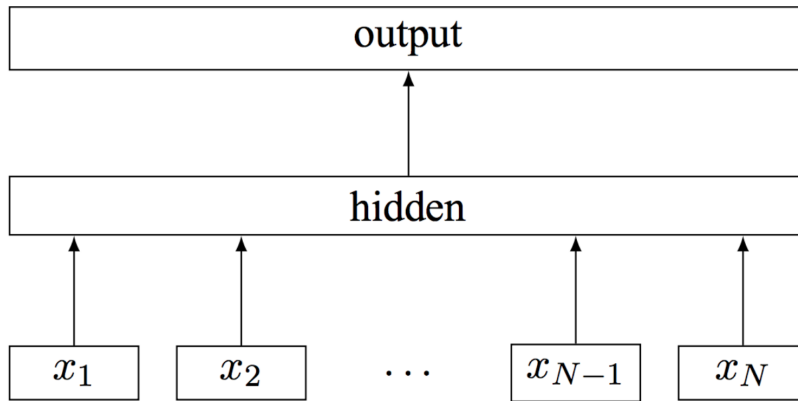


Figure 2.9: Model architecture of FastText for a sentence with N ngram features presented by [JGBM16].

output vector for a word is made of the sum of its character n-grams. For example, the word vector “night” is a sum of the vectors of the n-grams: “<ni”, “nig”, “nigh”, “night”, “night>”, “igh”, “ight”, “ight>”, “ght”, “ght>”, “ht>”. FastText learns vectors for n-grams of each word, including the word itself. The adjustment that is calculated from the error is then used consistently to update each of the vectors that were combined to form the target. At each point, a word needs to sum and average its n-gram parts. To make the process faster, FastText use a hash function proposed in [FNV⁺11]. The main contribution of FastText is using the n-gram feature, which helps the model deal with *Out-of-Vocabulary* (OOV) problem. Learning n-grams of the word helps the model to predict unseen words based on their shared parts. As a result, FastText generates better word embeddings for rare words.

²<https://github.com/facebookresearch/fastText>

2.3 Deep learning for Entity Resolution

Entity resolution generations tend to achieve better performance by exploiting external knowledge. *Crowd-sourcing ER* [WKFF12], which employs human intelligence to break complex tasks for higher accuracy, is among the most common practices in this field. Applying *Deep Learning* techniques for entity resolution, which incorporate contextual information in the form of word embeddings, is also a popular method that exploits external knowledge. Deep learning methods have been proposed on each entity resolution step, but due to our research’s time constraints, we study deep learning works on the representation and entity matching step. In this section, we review two of the most popular deep learning approaches for ER, which outperformed traditional techniques for entity matching on dirty datasets and laid a foundation for the research on transformer models.

2.3.1 DeepER

DEEPER is a novel entity resolution system, presented by [ETJ⁺17], which extracts tuple embeddings from individual word embeddings to achieve good accuracy, high efficiency, as well as ease-of-use. Human involvement in each step of entity resolution makes major challenge for democratizing the whole process. The authors of DeepER propose a system that needs much less labeled data by considering prior knowledge of matched values which can capture both syntactic and semantic similarities without feature engineering. The main idea of DeepER, depicted in Figure 2.10, can be divided into a deep neural network which includes embedding lookup layer, composition layer (which is the biggest contribution of the system), similarity layer, dense layer and classification layer. In DeepER, each original tokens using an *Embedding lookup*, correspond to each attribute. The word embeddings composed into a single vector in the *Composition layer*. Two different approaches are proposed to achieve a distributed representation of tuples, which we discuss in detail in the next subsection. While both tuples are being compared based on *Similarity*, in the *Dense layer*, similar words are close to each other in their semantic space. The *Classification layer* outputs *true* if a pair of tuples are match and *false* in case of mismatch.

Distributed Representations of Tuples

The main contribution of DeepER is based on proposing two methods for effectively computing of *Distributed Representations of Tuples* (DR) by composing the DRs of all the tokens within all attribute values of a tuple. These two proposed approaches are defined as follows:

- *A Simple Approach: Averaging.* The first approach proposed by authors is a simple averaging of the tokens. After tokenizing each attribute value into individual words using tokenizer, for each token, the associated embedding is being assigned (the k-dimensional vector). This task is done by GloVe pre-trained dictionary, which we discussed in Subsection 2.2.2. The averaging approach measures the average of word embeddings in each attribute and concatenates attribute embeddings. In this case, Entity Similarity is defined by k-dimensional cosine similarity where k is the

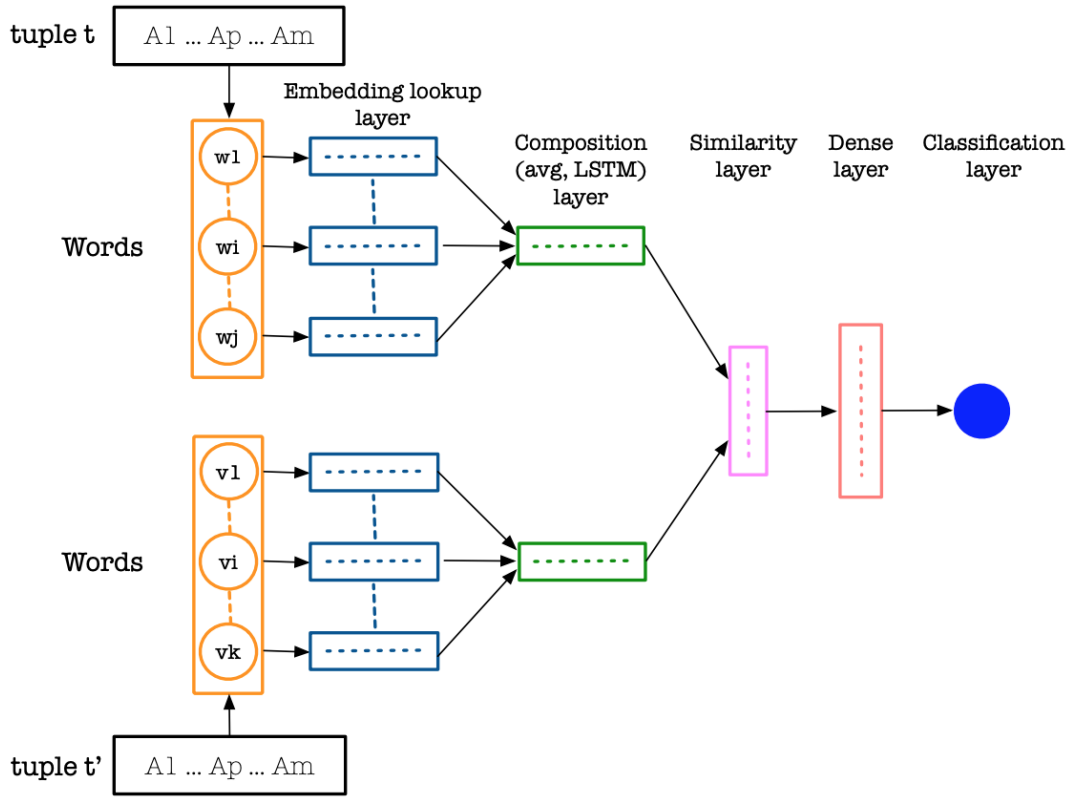


Figure 2.10: Deep entity resolution framework (DeepER) presented by [ETJ⁺17].

number of attributes. While the averaging approach is efficient and straightforward to train, it discards the word order, which can be problematic.

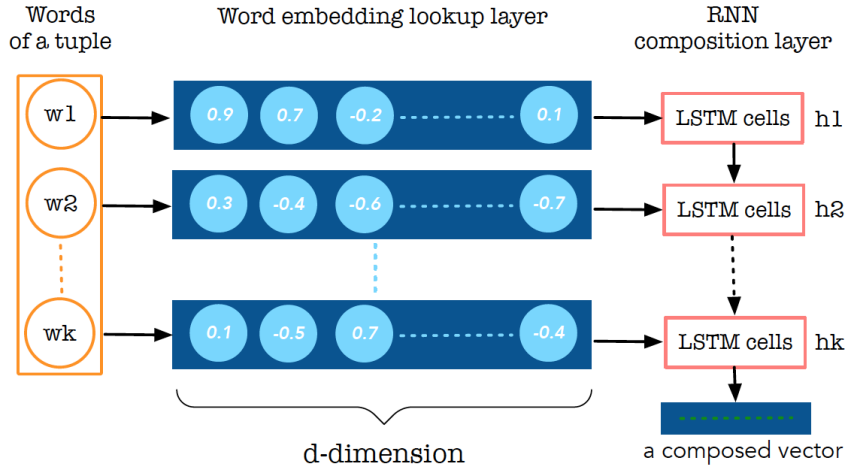


Figure 2.11: RNN with LSTM in the hidden layer.

- *Compositional Approach: RNN with LSTM.* To tackle the word order problem, a compositional approach is used, which deploys a neural network to compose the word vectors into an attribute-level composed vector semantically. While many neural network architectures have been proposed, the authors of DeepER deploy uni-

and bi-directional recurrent neural networks (RNNs) with long short term memory (LSTM) hidden units [HS97] to consider the relationship between attributes. A uni-directional recurrent neural network is shown in Figure 2.11, which encodes a sequence of words for all tuple words into a composed vector by processing its word vectors sequentially, at each time step, combining the current input word vector with the previous hidden state. Benefitting from hidden states, allow the neural network to remember the output of the previous time step. The output of the last time step represents the tuple. Bi-directional RNNs provide two different views of the same sequence that can encode information both ways and, therefore, capture dependencies from both directions. Entity Similarity, in this case, is defined by subtracting (vector difference) or multiplying (Hadamard product) of corresponding vector entries, resulting in an x -dimensional similarity vector.

2.3.2 DeepMatcher

After introducing DeepER, a new architecture template for deep learning on entity matching called *DeepMatcher* [MLR⁺18] proposed, which considered four different deep learning solutions on different entity matching problems. The main contribution of DeepMatcher is an architecture composed of three modules that can address entity matching problems with any chosen options regarding embedding, attribute similarity, and classification. We investigate this architecture in the following subsection. DeepMatcher assumes that Schema Matching and Blocking are already in place. Four different deep learning solutions for entity matching are selected as follows:

- *SIF: An Aggregate Function Model.* This model applies an aggregate function, specifically a weighted average inspired by *Smooth Inverse Frequency* (SIF) sentence embedding model [ALM16] for attribute summarization and an element-wise absolute difference comparison operation to form the input to the classifier module.
- *RNN: A Sequence-aware Model.* It disposes a bidirectional RNN to achieve the tasks mentioned above. This model, which is inspired by [CVMG⁺14], consists of two recurrent neural networks: the forwards RNN to process the information in the regular order and the backward RNN to do the same in the reverse order. Therefore, the final attribute summarization representation corresponds to the concatenation of the last two outputs of the bidirectional RNN.
- *Attention: A Sequence Alignment Model.* The third model uses decomposable attention inspired by [PTDU16] to implement attribute summarization and vector concatenation to perform attribute comparison. We discuss the attention mechanism in Subsection 2.4.1. This model performs soft alignment and pairwise token comparison across the two input sequences.
- *Hybrid: Sequence-aware with Attention.* Combining a bidirectional RNN with decomposable attention to achieve attribute summarization and produce the input for the classifier is done by a Hybrid model. The concept of hybrid typically can lead to higher power and performance in the field of machine learning, and this model is no different.

DeepMatcher considers three types of entity matching problems: Structured EM, which consists of structured records with the same schema, Textual EM, which includes attributes corresponding to raw text entries, and Dirty EM, which follows the same structured form but consists of missing values. After experimental analysis over real-world datasets, DeepMatcher concludes that using deep learning models may not be very efficient in the first two entity resolution generations and justifies the result with the requirement of too many labeled instances. However, deep learning methods outperform traditional entity resolution approaches on third-generation challenges, which considers dirty datasets.

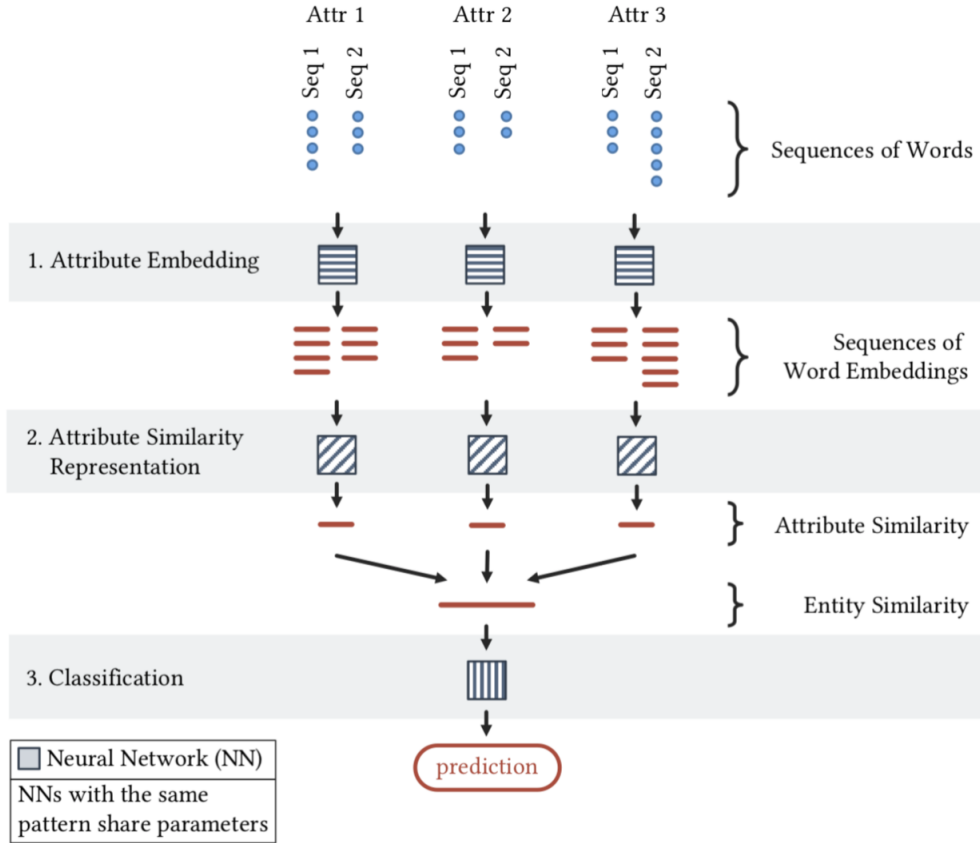


Figure 2.12: DeepMatcher framework presented by [MLR⁺18].

DeepMatcher Architecture Template

Proposing an architecture template to fit different deep learning models into an entity matching problem is the main contribution of DeepMatcher. After considering four different deep learning approaches and three different entity matching problems, a general template is designed to consider the language representation, the summarization technique, and the comparison method used to analyze an input pair of sequences. This template, as shown in Subsection 2.3.2 is consisted of three modules as follows:

- *The Attribute Embedding Module.* Each attribute value is tokenized and converted into a sequence of embedding vectors. In other words, This module takes a sequence

of words and converts them to two sequences of word embedding vectors. The authors of DeepMatcher present two possible categories of embedding choices as the granularity of the embedding and training possibility of the embedding. Granular-based options include word-level embeddings [MSC⁺13] which encode each word in the sequence as a fixed d-dimensional vector and character-level embeddings [BGJM17] that takes word characters as input and use a neural network to produce the mentioned vector. Train-based embedding is simply the choice of using pre-trained word-embedding (such as Word2vec and GloVe) or train embeddings from scratch.

- *The Attribute Similarity Representation Module.* Given the attribute value embeddings from the previous part as input, this module encodes this input to a representation that captures the attribute value similarities. In order to achieve this goal, two different operations are performed as follows: *Attribute summarization* which takes the two sequences as input and after applying the desired operation, summarises the information, and *Attribute comparison* that given the summary vectors, applies a comparison function to obtain the final similarity of the attribute values. Attribute summarisation can be done by aggregate function, sequence-aware summarisation, sequence alignment, or a hybrid combination of more than one method. Typically, the two main options for comparison operation are fixed distance functions, which include pre-defined distance metrics such as euclidean distance, and Learnable Distance Functions that rely on classification module to learn a similarity function itself.
- *The Classifier Module.* Entity resolution is a binary classification problem. This module takes the similarity representations as input and after feeding them into a classifier, determines if the pair refer to the same real-world object. DeepMatcher authors used a neural network as classifier.

2.4 Attention and Transformers

The architectures applying recurrent neural networks (RNN) at its core have dominated the field of natural language processing for a long time[SVL14]. We reviewed some of them in the previous section. As we mentioned before, the RNN models suffer from drawbacks such as bottleneck problem (dealing with the long sentences) [BCB14], insufficient learning of long-range dependencies[VSP⁺17], and long training time. In this section, we provide the necessary background information on a mechanism designed to solve this problem called *Attention*. Furthermore, we discuss *Transformers*, a particular type of neural network based on attention, and focus on four of the most recent attention-based transformer architectures we use for entity resolution.

2.4.1 Attention

Attention is a technique proposed in [BCB14], [LPM15] in response to the bottleneck problem of machine translation systems. The basic idea of the attention mechanism is based on letting the model focus on the input sequence’s relevant parts as needed. Based on the intuition of self-attention, a word can be represented as a weighted combination of

its neighborhood. In an encoder-decoder model, the importance of relating words is prioritized by weights to help the decoder improve its prediction. In order to get a better idea of attention mechanism, it is better to understand how self-attention works. **Figure 2.13** concentrates on self-attention functionality. Take the sentence, “My mother takes my

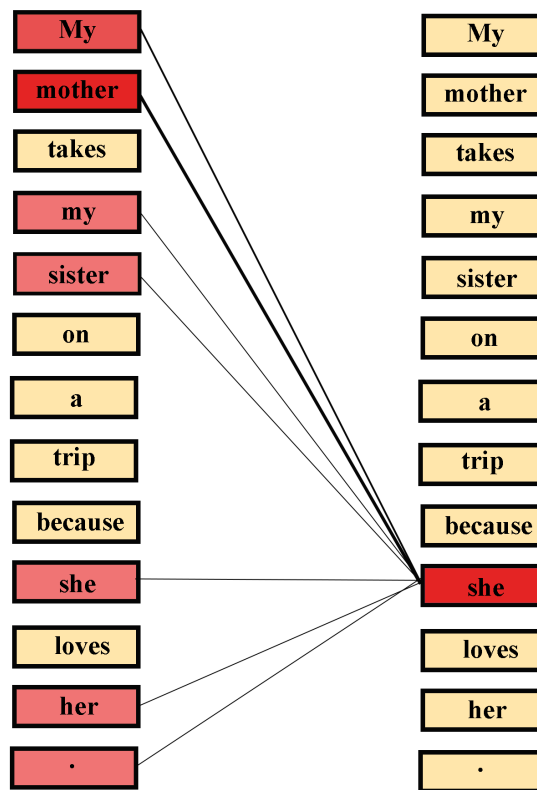


Figure 2.13: Self-Attention example

sister on a trip because she loves her.” as an example. Different words in its neighborhood can represent the word “she”. For the human reader, “she” can be represented by the words “My” and “mother”. Neural machine translation models struggle when it comes to translating, such as a sentence. Besides forgetting issue, lack of association between the words, make their predictions vulnerable. By deploying an attention mechanism, we can train a language model to pay attention to relevant words in its neighborhood in a closer way to what the human reader does. The first sequence-to-sequence model, with attention mechanism, profoundly improved the quality of machine translation systems. Sequence-to-sequence models are deep learning models that take a sequence of items and output another sequence of items. In the case of machine translation systems, items are words. This model is composed of an encoder and a decoder. The encoder processes each item in the input sequence and captures the information into a vector called *the context*. Both encoder and decoder tend to be a recurrent neural network. This architecture has achieved much success in tasks like image captioning, text summarization, and machine translation. As **Figure 2.14** depicts, the encoder RNN takes two inputs (an input word (an embedded vector) and a hidden state) at each time step and yields an output. Since the encoder is an RNN, it can update its hidden state at the same time. After processing the entire input, the encoder passes the context vector along with the decoder. At this time step, the decoder receives the last hidden state as the context, which is problematic. Long sentences make it harder to transfer knowledge about the source sentence to the

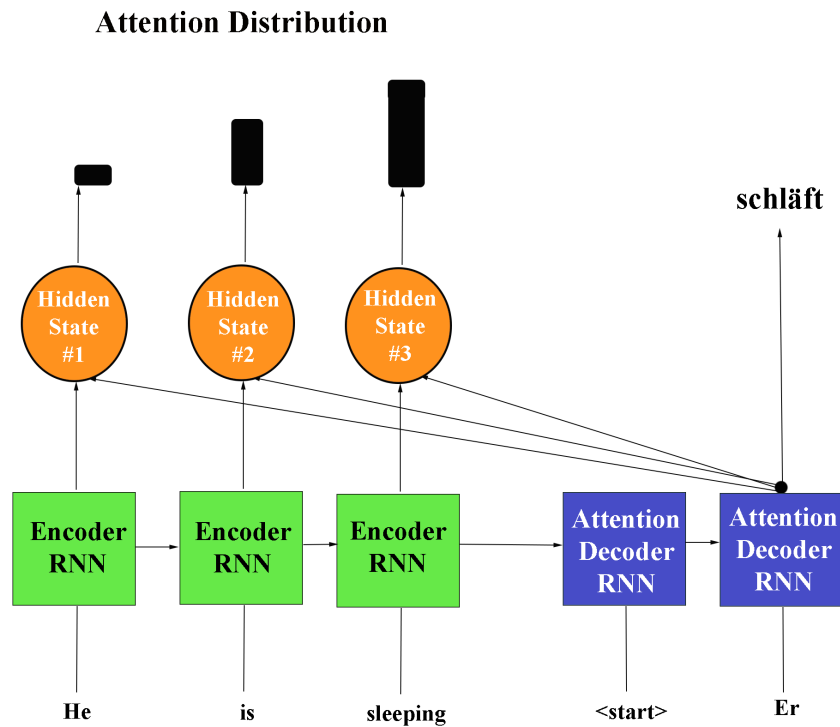


Figure 2.14: Sequence-to-sequence model with attention

target sentence. It is instinctive to understand that the longer the sequence, the harder it is to get all the necessary information. What [LPM15] proposes is to apply the attention mechanism between the source and target sentence. In this way, instead of having the last hidden state as the context, the decoder has access to all the hidden states of the encoder. This procedure helps the decoder to focus on the relevant parts of the source input. Before producing its output, the decoder receives a set of encoder last hidden states. Each hidden state is most associated with the specific word in the source sentence. Based on association, the decoder gives each hidden state a score, which we show in Figure 2.14 as attention distribution at a specific time step.

The next step is multiplying each hidden state by its softmax score and drowning out hidden states with low scores. The decoder uses this information (now related words data) to produce its output. Assume that we want to translate the English sentence "He is sleeping" to the German counterpart "Er schläft" as shown in Figure 2.14. It is helpful for the decoder to consider all encoder hidden states' information rather than the last one when it is trying to produce the word schläft. By applying attention mechanism, the decoder will learn to focus on the essential hidden states of the encoder at each given time. Deploying attention mechanism on neural machine translations, which resulted in a high performance, inspired other researchers to use attention's power on a new approach called *Transformers*.

2.4.2 Transformer

The transformer is a deep machine learning model first introduced by [VSP⁺17], which relies solely on attention's power. While using LSTM and gated neural networks [CGCB14] became common in sequence modeling problems such as machine translation, incorporating attention mechanism along with a neural network, gained much success in tasks like text summarization and machine translation. The authors of [VSP⁺17] introduce a new architecture called *Transformer*, which completely abandons RNNs and is exclusively based on attention to draw global dependencies between input and output. Due to its architecture, transformers do not require that the sequential data be processed in order, leading to higher parallelization power and translation quality. As shown in Figure 2.15,

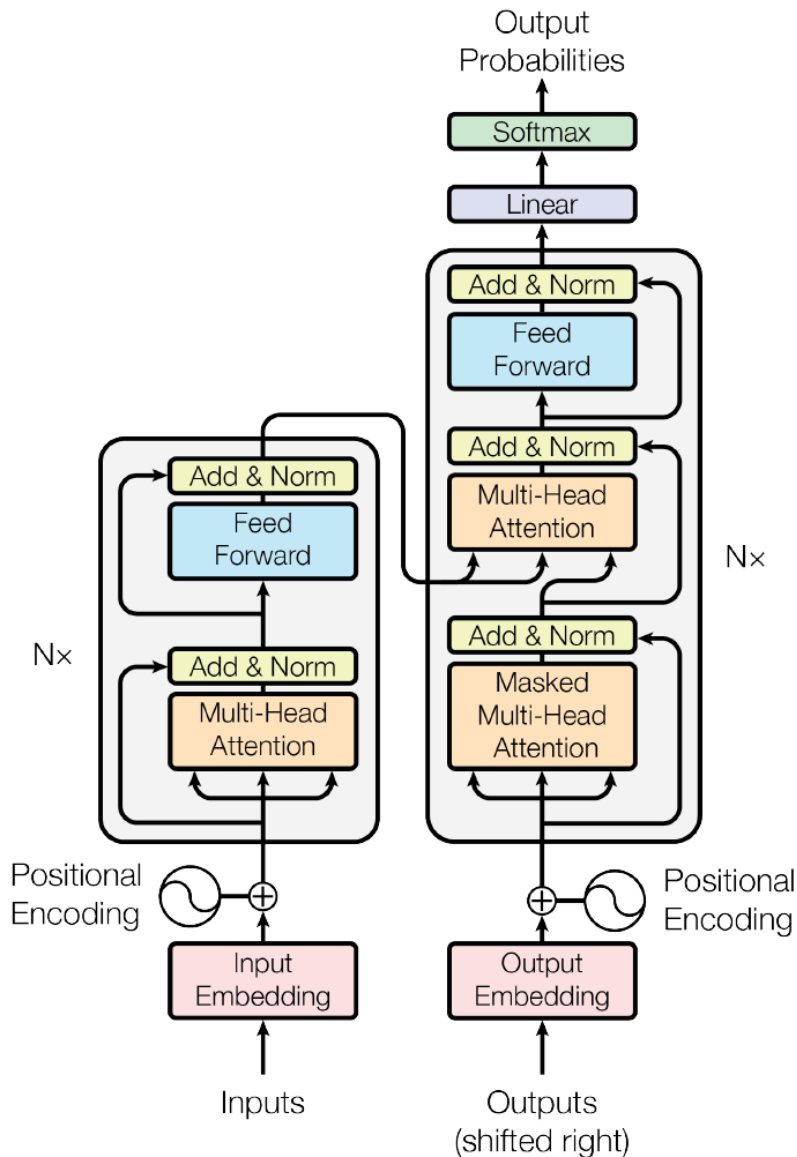


Figure 2.15: Transformer architecture proposed by [VSP⁺17]

the transformer architecture maintains the classical encoder/decoder model, but it succeeded the RNN by a new design called Multi-head attention. The original encoder and

decoder architecture is composed of a stack of 6 identical layers. Encoder layers broke into two sub-layers: a multi-head self-attention mechanism and a position-wise fully connected feed-forward network. The authors of the transformer use a residual connection, proposed by [HZRS16] around each sub-layers, which improves training performance. In addition to the two sub-layers in each encoder layer, the decoder includes a third sub-layer, which performs multi-head attention over the encoder stack's output. Multi-head attention allows the model to jointly attend to information from different representation sub spaces at different positions. The decoder's self-attention mechanism only has access to the words created by the decoder, and words that need to be processed are listed as *masked* when masking words with high negative values will be discarded from the learning algorithm.

In order to deal with order of the sequence, The authors use a method called positional encoding for each input word which can inject some information about the relative or absolute position of tokens in the sequence. Positional encoding can obtain by combination of the functions sine and cosine with different frequencies. Final experiments conclude that the transformer model can be trained significantly faster than architectures based on recurrent or convolutional layers. Also, transformers outperformed the Google Neural Machine Translation model in many tasks which inspired a new wave of transformer architectures which have been trained with huge general language datasets, and can be fine-tuned to specific language tasks.

2.4.3 BERT Transformer

The popularity of transformer architectures sparked a lot of new papers based on this idea. BERT, first introduced in [DCLT18], is one of the most popular transformers that outperformed traditional deep learning models for many natural language processing tasks. In this section, we will study BERT as one of the four approaches we use in our proposed design.

What is BERT?

BERT is a universal language model, pre-trained on large amounts of text data to fine-tune it on downstream tasks in a supervised manner with relatively little data. BERT stands for *Bidirectional Encoder Representations from Transformers*, which emphasize the importance of bidirectional learning during pre-training. Semi-supervised sequence learning [DL15] introduced two approaches that use unlabelled data to improve sequence learning with recurrent networks. These two proposed approaches can be used as a *Pre-training* step for a supervised sequence learning algorithm. The two strategies for applying pre-trained language representation are *Feature-based* and *Fine-tuning*. Feature-based strategies deploy task-specific architecture and include the pre-trained representations as additional features[PNI⁺18],[JBS17]. Fine-tuned methods introduce minimal task-specific and are trained on the downstream tasks by simply fine-tuning all pre-trained parameters [RNSS18]. The BERT authors argue that two approaches cannot satisfy unidirectional language model restrictions. Specifically, the fine-tuning approach, such as GPT, uses left to right context to predict, and this could be very harmful where it is crucial to incorporate context from both directions in tasks such as question answering.

BERT is jointly conditioning on both the left and right context of the query token during pre-training. BERT uses a transformer language model designed by [VSP⁺17] at its heart, but only deploys encoder layers. Moreover, when it comes to prediction, BERT contradicts other transformer architectures' methods. To be able to condition in both the left and right context, BERT applies a *Masked Language Model* (MLM) inspired by [Tay53]. The masked language model randomly masks 15% of the input tokens and aims to predict the original vocabulary id of the word based solely on its context. In this way, BERT can grasp a very informative contextualized word embeddings through this process. BERT also uses a *Next Sentence Prediction* task that jointly pre-trains text-pair representations. BERT is available by Google in two model sizes as follows³:

- $BERT_{BASE}$. 12 transformer blocks, 768 hidden units, and 12 attention heads.
- $BERT_{LARGE}$. 24 transformer blocks, 1024 hidden units, and 16 attention heads.

$BERT_{BASE}$ is comparable in size to the GPT transformer in order to compare performance. $BERT_{LARGE}$ was introduced to prove that larger models lead to a strict accuracy improvement across benchmark datasets.

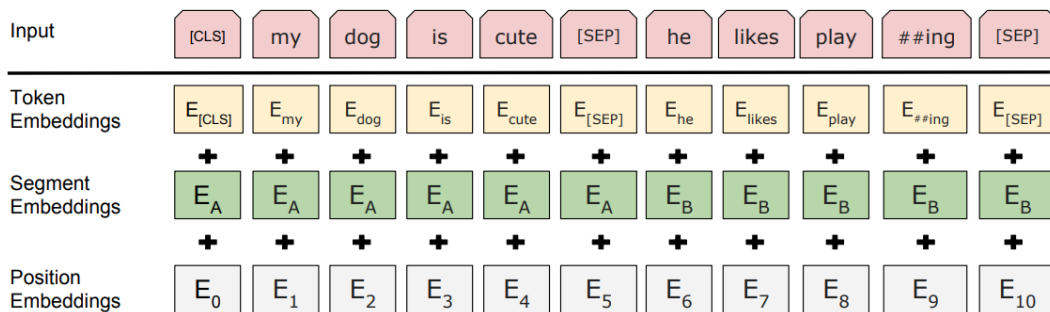


Figure 2.16: BERT input representation introduced in [DCLT18]

Input and Output Representations

BERT input representation can represent both a single sentence and a pair of sentences in one token sequence. Each word is divided into tokens, and special tokens are added. The first token of every sequence is always a special classification token ([CLS]). We use the final state corresponding to this token as the aggregate sequence representation for our classification input on our proposed approach. A special separation token separates the sentences in each sequence ([SEP]). For a given token, its input representation is built by summing the corresponding token, segment, and position embeddings as shown in Figure 2.16.

Masked Language Model

The main contribution of BERT is a bidirectional model. Standard conditional language models can only be trained left-to-right or right-to-left. For example, let us consider

³<https://github.com/google-research/bert>

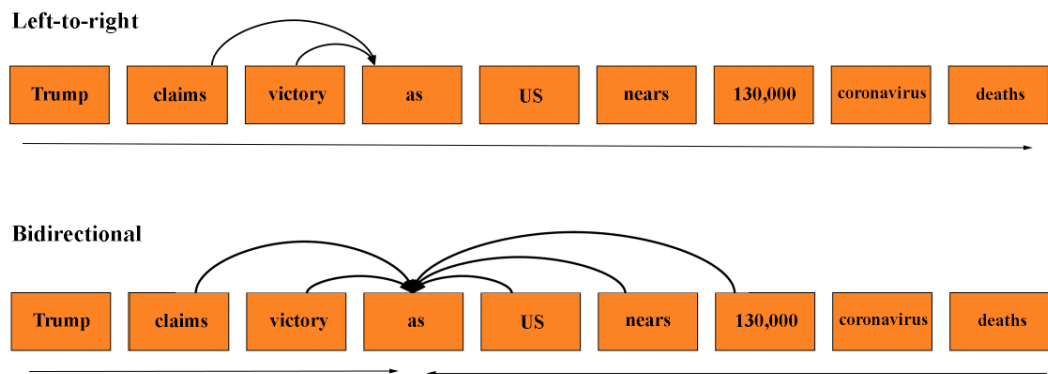


Figure 2.17: BERT bidirectional approach to predict the token.

the following sentence: “Trump claims victory as US nears 130,000 coronavirus deaths”. Also, let us assume that we want to predict the token “as” by its left context as shown in Figure 2.17. With this approach, the model can look at the tokens ‘Trump’, ‘claims’, and ‘victory’ to predict the next token. Using the left and right context, the model already knows the following tokens, and the prediction achieves much higher accuracy. In order to train a deep bidirectional model, we mask around 15 percent of input tokens and then predict the masked tokens. In this case, the final hidden vectors corresponding to the mask tokens are fed into an output softmax over the vocabulary. BERT language modeling with predicting the masked token depicted in Figure 2.18

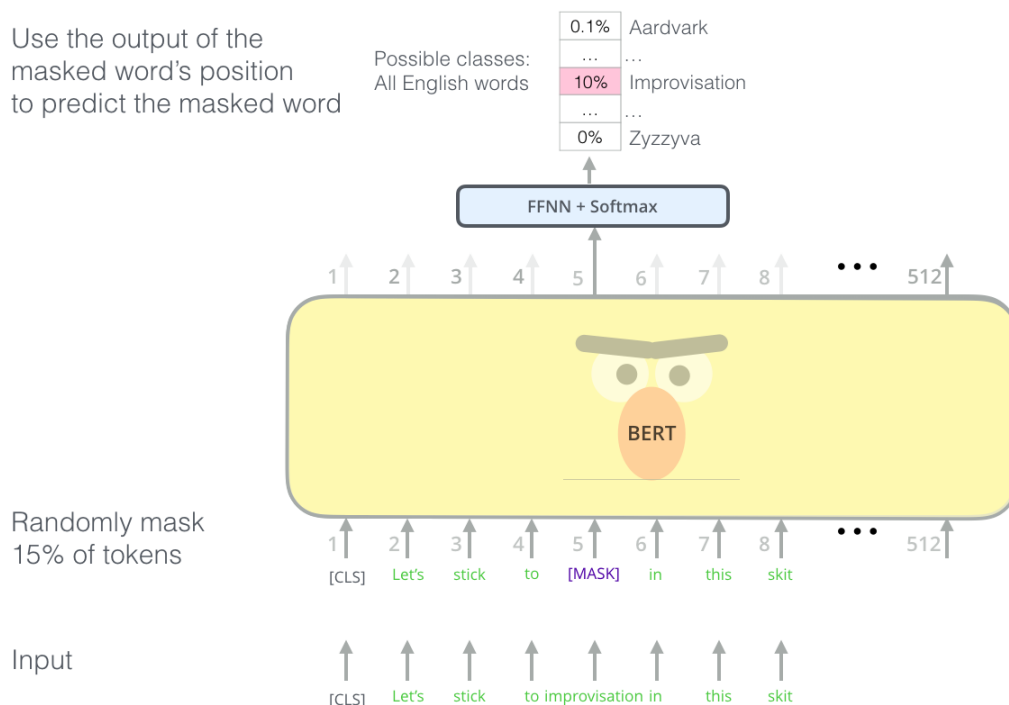


Figure 2.18: BERT language modeling with masked token prediction simplified by [Ala18].

Next Sentence Prediction (NSP)

BERT performs *next sentence prediction* as another unsupervised task during pre-training. NSP is performed to better understand the relationship between two sentences in order to tackle some of the most important downstream tasks such as natural language inference and question answering[LL18]. The model receives two sentences as input and has to predict if the second sentence follows the first sentence. The training set for this task consists of 50% of sentences that are actual next sentence and 50% of sentences that are random sentences from the corpus. The paper proves that the training of NSP is very advantageous to question answering task.

2.4.4 DistilBERT Transformer

In this section, we discuss another member of the BERT family, DistilBERT [SDCW19], a transformer model that pursues speed, compactness, and feasibility in real-world applications. We review the knowledge challenge the DistilBERT is facing and its approach to solving this problem.

What is DistilBERT?

DistilBERT is a distilled version of BERT, making it smaller, faster, cheaper, and lighter. The authors of DistilBERT argue that using large transformer models to get higher accuracy can limit their application in real-world projects. DistilBERT aims for a smaller version of transformer language models that can produce impressive results. This model practices knowledge distillation to reduce the model size to deal with the cost of exponentially scaling transformers models [SGM19]. While benefiting from the same general architecture as BERT, DistilBERT architecture removed the token-type embeddings and the pooler while the number of layers is reduced by a factor of 2. The authors of DistilBERT show that their model retains 97% of the original BERT model performance on benchmark datasets while reducing the size by 40% and being faster 60%.

Knowledge Distillation

Knowledge Distillation, proposed by [HVD15], is a compression technique in which the smaller model (the student) is trained to reproduce the behavior of the original model (the teacher). *Generalized Knowledge Distillation*, sometimes also referred to as teacher-student learning, aims to learn the model's dark knowledge. Dark knowledge refers to tokens with a low probability that may include crucial information, and the model must generalize them. In an example shown in Figure 2.19, let us assume the language model predicts the last token in a sentence "I think this is the beginning of a beautiful ...". while the words 'day' and 'life' have a high probability of being matched, most parts of the list include tokens with almost zero possibility. The challenge of Knowledge Distillation is learning a prediction considering high possibility words and near-zero possibilities called "soft targets". To learn both high possibility tokens and soft targets, DistilBERT introduce

Input:

'I', 'think', 'this', 'is', 'the', 'beginning', 'of', 'a', 'beautiful', [MASK]

Predicted token (top 16):

#1 token: 'day' p: 0.213	#9 token: 'summer' p: 0.016
#2 token: 'life' p: 0.183	#10 token: 'adventure' p: 0.013
#3 token: 'future' p: 0.062	#11 token: 'dream' p: 0.012
#4 token: 'story' p: 0.058	#12 token: 'moment' p: 0.012
#5 token: 'world' p: 0.049	#13 token: 'night' p: 0.011
#6 token: 'era' p: 0.045	#14 token: 'beginning' p: 0.010
#7 token: 'time' p: 0.032	#15 token: 'season' p: 0.009
#8 token: 'year' p: 0.017	#16 token: 'journey' p: 0.006

Figure 2.19: Knowledge distillation challenge in an experiment done by [BS20].

a triple loss combining language modeling, distillation and cosine-distance losses. Triple loss of DistilBERT is proposed as follows:

- *Masked Language Modeling.* As the teacher model in our case is original BERT, the loss function is also original masked language modeling to predict the masked tokens. We elaborated on this in section.
- *Distillation Loss.* The student is trained with a distillation loss over the soft target probabilities of the teacher as:

$$L_{ce} = \sum_i t_i * \log(s_i)$$

where t_i (respectively s_i) is a probability estimated by the teacher (respectively the student). The authors of the paper recommenced using a technique called soft max-temperature to smooth the output distribution.

- *Cosine Embedding Loss* A somewhat technical loss to align the directions of the student and teacher hidden state vectors.

2.4.5 XLNET Transformer

The success of BERT transformer in many natural language processing tasks motivated the other researches to follow this transformer step into finding a way toward a more trustworthy design. In this section, we overview the XLNet [YDY⁺19], another state-of-art transformer that tackles BERT problems with a novel approach.

What is XLNet?

XLNet is an autoregressive language model that produces the joint probability of a sequence of tokens based on the transformer architecture with recurrence. XLNet aims to

determine the probability of a token conditioned on all permutations of tokens in a sentence instead of focusing only on one side of the target token. The authors of XLNet argue that while BERT approach is beneficial in many NLP tasks, it has two serious downsides as follows:

- *Pretrain-finetune Discrepancy.* BERT’s main task is to use a masked language model and attempt to predict the masked tokens. BERT deploys this approach to achieve bidirectional representation, which helps the model to handle language-based tasks. Considering masked tokens are artificial symbols that never occur in downstream tasks, a pretrain-finetune discrepancy problem arises.
- *Independence Assumption.* BERT process all masked tokens in a parallel way and independent of each other. The independence assumption is not justified.

To tackle these problems, XLNet integrates ideas from the classical *Transformer-XL* model [DYY⁺19], the state-of-the-art autoregressive model, into pre-training. Therefore, XLNet is a generalized autoregressive pre-training method. Autoregressive (AR) language models [DL15] are types of models that limit predicting the next word to only one direction. AR models can predict the next word either forward or backward. Since generating context is usually forward direction, AR models achieve excellent results in generative natural language processing tasks [RNSS18]. Nevertheless, learning in only one direction restricts the model to capture a bidirectional context. XLNet proposes a new method called *Permutation Language Modeling*, which can comprise the bidirectional context and solve BERT drawbacks. XLNet outperforms BERT on 20 tasks and achieves state-of-the-art results on 18 tasks, including question answering, natural language inference, document ranking, and sentiment analysis.

Transformer Architecture

XLNet utilizes underlying Transformer-XL architecture at its core, based on the original transformer structure proposed in [VSP⁺17] but with a different design. The model is a *Transformer-Decoder*, made up of a stack of recurrent decoders, and it does not involve any encoder blocks. Transformer-XL deploys the attention mechanism to deal with long-range dependencies. The decoder needs to look at the whole sentence and extract information selectively to translate it. The decoder has access to all hidden states and uses attention weights to focus on related words for each token. The two essential techniques integrated from Transformer-XL into XLNet are as follows:

- *Positional Encodings.* Transformer-XL proposes a new way of relative positional encodings to keep track of each token’s position in a sequence in order to reuse the hidden states. This approach gives the model ability to learn dependencies beyond a fixed-length without disrupting temporal coherence [BS20].
- *Segment Recurrence.* Transformer-XL suggests a new mechanism that creates a segment-level recurrence in the hidden states. As a result, the effective context being used can go way beyond just two segments. Furthermore, Practicing this technique leads to faster evaluation. Specifically, during the evaluation, the representations from the previous segments can be reused instead of being computed from scratch.

Permutation Language Model

The main contribution of XLNet, inspired by [UCG⁺16], is a modified language model training objective that learns conditional distributions for all possible permutations of tokens in a sequence. Rather than utilizing fixed backward or forward learning, XLNET maximizes expected log-likelihood over all possible permutations of the sequence. Therefore, each position learns to employ contextual information from all positions, thereby capturing the bidirectional context. Figure 2.20 illustrates the permutation language modeling objective in brief. Let us consider that we are learning the token at the 3rd position in the sentence (x_3). Model training with different permutations of the tokens in the sentence at the end of all such permutations, we would have learned x_3 , given all other words in the sentence.

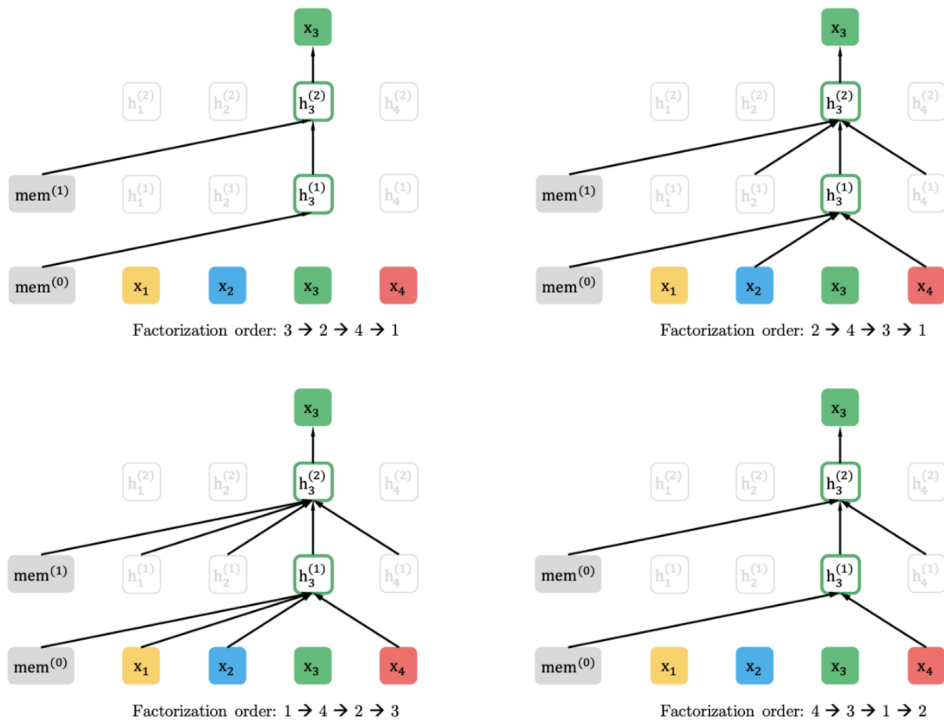


Figure 2.20: Illustration of the permutation language modeling objective for prediction a token [YDY⁺19].

2.4.6 GPT2 Transformer

The encoder-decoder architecture is not the only approach for transformers concerning language modeling. BERT's progress as a stack of encoders was a step toward a new wave in architectures that use either the encoder or decoder. In this section, we discuss one of these modern architectures, GPT-2, that delivers excellent performance in natural language processing tasks.

What is GPT-2?

GPT-2 proposed in [RWC⁺19] stands for “Generative Pre-trained Transformer 2”. *Generative* for being the model used to generate (predict) the next word, *Pre-trained* for creating a large language model that can be fine-tuned for specific tasks, and *Transformer* for using the transformer architecture. While BERT uses transformer encoders, GPT-2 is built on transformer decoders. The GPT-2 was trained on a substantial 40GB dataset that the OpenAI researches crawled from the internet, and this made GPT-2 very powerful dealing with different tasks. OpenAI released four versions of GPT-2⁴, in which the largest variant takes 6.5 GB of storage data. The basic idea of GPT-2 architecture is stacking up transformer decoders as high as possible and feed the data through the pre-trained model. Unlike BERT, GPT-2 yields one token at a time. GPT-2 also is an *Autoregressive Model*, which means after each token is produced, that token is added to the sequence of inputs. Moreover, the new sequence becomes the input of the model in its next step. Being an autoregressive model prevents GPT-2 to capture the context bidirectionally.

The Decoder-Only Block

GPT-2, at its core, uses the *Transformer-Decoder* model, which suggested first in [LSP⁺18]. The Transformer-Decoder model is a stack of decoders in which each block includes a particular *Masked Self-Attention* layer. The Self-Attention layer in BERT allows the model to pay attention to its right tokens, but Masked Self-Attention layer blocks information from tokens that are to the right of the calculated position. Figure 2.22 shows the difference between the self-attention layer in both architectures. Following the introduction of six transformer decoder blocks in the first Transformer-Decoder architecture, another language model inspired by a stack of decoders proposed by [ARCC⁺19] to predict one character at a time. GPT-2 starts to process the tokens by generating actual samples. The model only has one input token at the same time, and only its associated path would be active. Given the vector output, the model can score the vector against its vast vocabulary and returns the token with the highest probability. Following, the model adds the output into the input sequence, and the next prediction step begins.

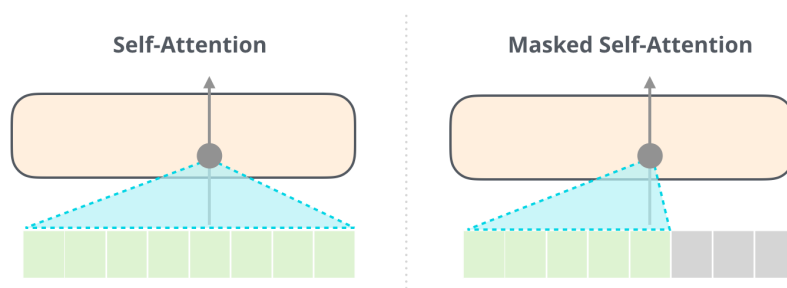


Figure 2.21: Self-Attention vs. Masked Self-Attention.

⁴<https://github.com/openai/gpt-2>

Input and Output Representations

Before feeding the input into the model, GPT-2, like many transformers, refines the input word embedding in its embedding matrix. Positional encodings as the signal that shows the order of the words to the transformer blocks are the next step that should be incorporated into the token. After processing the token by the highest decoder layer, the model produces its output vector, multiplying it by the embedding matrix. The result of this multiplication is interpreted as a score for each word in the model’s vocabulary. The best token with the highest probability would be selected. To improve the performance of the model, GPT-2 can consider the top 40 tokens with the best score. The model continues iterating until the entire context is generated (1024 tokens) or until an end-of-sequence token is produced. Figure[2] portrays the process of feeding text inputs into the GPT-2 transformer model.

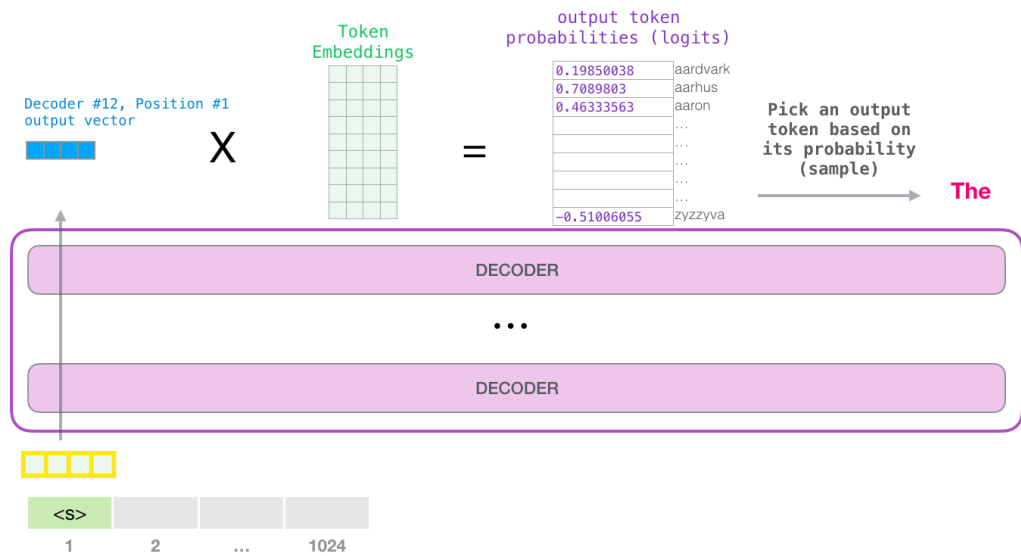


Figure 2.22: GPT-2 transformer architecture simplified by [Ala19].

2.5 Summary

In this chapter, we overview Entity Resolution as a task and describe the challenges we face in this process. We briefly review ER methods through time, which can help us gain the necessary background knowledge about the entity resolution framework. We study word embedding and three popular approaches, such as Word2vec, GloVe, and FastText. Using embeddings, we discover two deep learning models on entity matching tasks to achieve higher performance. Inspired by an attention mechanism on deep learning methods, we explore transformers and pay special attention to transformer models, namely BERT, DistillBert, GPT-2, and XLNet.

3 Design

In [Chapter 2](#), we presented the necessary background knowledge concerning the topics covered in this thesis. In this we define precise research questions addressed in this research as well as the proposed design of the approach we practice for the task of entity resolution. This chapter is structured as follows:

- We begin by establishing the research questions to be evaluated in our work ([Section 3.1](#)).
- In ([Section 3.2](#)), we present our proposed approach for using attention-based transformers to tackle the entity resolution problem.
- Finally, we provide a summary of the main topics we discussed in this chapter in ([Section 3.3](#))

3.1 Research Questions

This research initiative is an attempt to understand how well the most recent attention-based transformer architectures (BERT, XLNet, GPT2, and DistilBERT) perform on the task of entity resolution. Based on the literature review we are able to specify the specified set of evaluation questions we intend to answer in this thesis:

1. To what extent does the performance at the classification of an entity resolution classifier model (XGBoost) using pre-trained embeddings from a fixed transformer model depend on tuning the hyper-parameters of the model (assuming a competitive reasonable basic configuration)?
2. Assuming a fixed set of hyper-parameters on a classifier model, what is the role of alternative pre-trained transformer models on the accuracy of the overall entity resolution formulation?
3. Assuming a fixed pre-trained transformer model, what is the role of alternative classifiers on the accuracy of the overall entity resolution formulation?

3.2 Proposed Approach: Transformer-based ER

To address our research questions, we prototype an architecture that should support an *Entity Resolution* task by any chosen pre-trained transformer model on any chosen dataset. The design we propose, depicted in [\[Figure 3.1\]](#), embodies two critical layers: the transformer and classifier layers.

Any Chosen Dataset(s)

- Amazon-Google
- DBLP-ACM

Any Chosen Transformer

- BERT
- DistilBERT
- XLNET
- GPT2

Any Chosen Classifier

- Logistic Regression
- GBM
- KNN

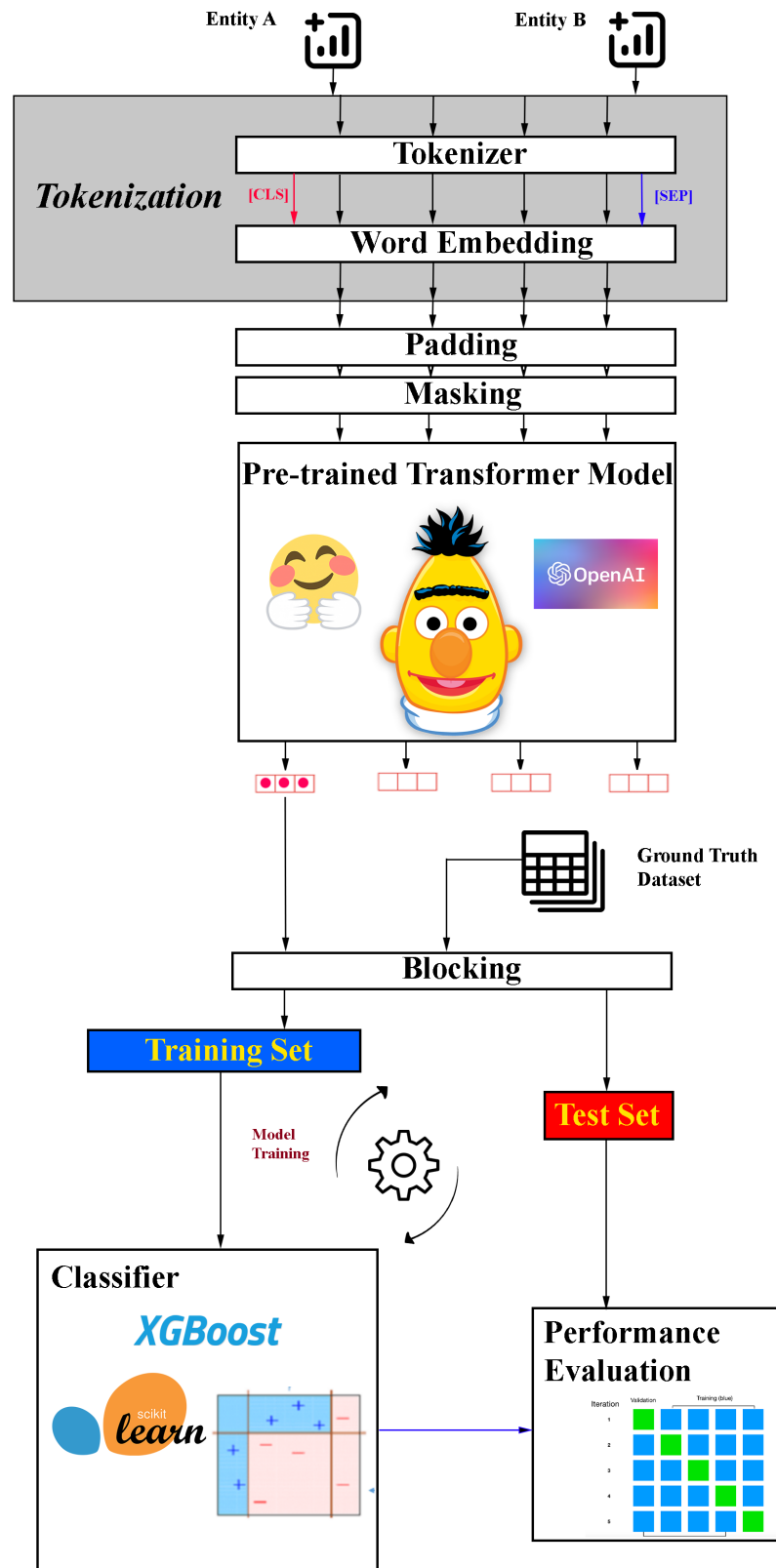


Figure 3.1: Design of the proposed approach for the task of entity resolution.

The first layer, a chosen transformer discussed in [Subsection 2.4.2](#), takes each record as input to tokenize, embed, and learn contextualized word embeddings. The second layer, a classification layer, is not pre-trained and contains a chosen classifier which should be trained for a binary classification problem. The classifier takes a training set of labeled records (matched and non-matched) and outputs predictions for unseen data. We will now illustrate how any desired record can be processed and classified through our proposed approach. Each entity belonging to the chosen dataset contains a single text blob. The context may vary depending on the dataset. Due to this research’s time and resource limitations, we decided to process each feature (independent variable) separately and concatenate the output before feeding the classification layer. Next, the single text blob is tokenized. After adding special tokens to the text blob, for each token, the corresponding embedding is assigned. Padding and masking are essential before feeding the embedded entity into the transformer model. Parallelized processing executed by the transformer model leads to better performance compared to traditional deep learning approaches [MLR⁺18]. Transfer models are neither trained nor fine-tuned to do classification, but due to their concept of being a universal language model, they can perform the classification task. The output of the first layer is represented in the hidden state at position 0 and is a vector of size 768. Given two embedded datasets, each consisting of input word embeddings related to a given entity identified by an ID, and a similarity ground truth consisting of the pairs of keys for similar entities, we create a list of triplets from matched and non-matched examples. Each triplet includes a pair of embedded Entity A and Entity B, and their similarity status (matched or non-matched). We composed the training list with 66% of records as the training set examples and 33% as test set instances. Before feeding the training set into the chosen classifier, we block the training data by grouping each pair using similarity metrics.

The training set, including blocked labeled entities, feed into the chosen classifier. We use xgboost, proposed in [CHB⁺15], as a standard classifier to deal with binary classification problem. We employ k-fold cross-validation to evaluate the performance of the model. This thesis examines the effect of using different transformer models and different classifiers in the proposed design.

The model design, depicted in [Figure 3.1](#), is guided by the following principles:

- *Tokenization.* Tokenization is an essential task before feeding the data into the transformer models. We use transformers’ tokenizer gathered by *hugging face* to achieve all tokenization at once. [Figure 3.2](#) shows the tokenization process guided by the following steps: First, tokenizer splits the words into tokens. Next, special symbols are used to feed the output into the classification layer ([CLS]) and to separate the two entities ([SEP]). The idea of using a special classification token is an architectural decision proposed by [DCLT18] to keep the model flexible. The tokenization task between transformer models is slightly different. Models such as the BERT family split the words into tokens by white space, punctuation, and in case of some of them special abbreviations. The other transfer models, such as XLNet, are not required to pre-tokenize the input into word sequences.
- *Word embedding.* In order to feed the data into a pre-trained transformer model, the input should be represented as word embedding. After tokenizing each word on attribute separately, the transformer’s tokenizer practices a standard word embedding method such as word2vec to replace all tokens with their associated id based on the lookup table. The transformer model takes these embeddings as input and

learns a contextualized embedding as output. The new embedding is carrying information about the profound relationship between the words, and in case of out of vocabulary word, it is the optimal word representation for the unseen word.

- *Padding.* Before using a pre-trained transformer model to output a contextualized word representation, padding the input is a mandatory step for most transformers. With padding, we make all the vectors the same size by filling shorter sentences with the token id 0. The output is a matrix that is ready to be processed by the transformer.
- *Masking.* Masking step, as shown in [Figure 3.1](#), is a totally different concept with learning masked token in some transformers, namely, BERT. After modifying the size of each vector with padding, using a masking step is necessary to avoid over-fitting. The model prone to confuse the extra padded values with real values, and for this matter, we need to create another variable such as mask to tell the model to ignore padded values.
- *Blocking.* Entity resolution is a quadratic problem. In order to achieve entity matching, each entity should compare to all of the entities, which makes the process time-consuming and can suffer from memory loss. Utilizing the ground-truth dataset, which includes true labeled records as the baseline, we group similar entities into blocks, and comparisons are executed only inside each block.

3.2.1 Data Pre-Processing and Vectorization

For pre-processing and vectorization of the data, these dataset pairs are tokenized for string attributes except the first column, ID, using pre-trained transformers' tokenizer. Before feeding the data into the transformer to tokenize, we need to solve some concerns that may arise. Regarding missing values, we convert 'NaN' to an empty string, which during the embedding process associated with a unique value. The last column of our entity resolution datasets that refer to price can include additional signs. We use the python string `rstrip` method to solve this issue. Following the tokenization output, we embed each word in the attributes independently. Using the ground-truth dataset, we generate the new embedded vector as the sum of the matched or non-matched embedding pairs. For the similarity computation between pairs, we calculate the average absolute deviation between vectors of each attribute, and for the last attribute, which is typically price, we measure the absolute difference over the normalized values.

3.2.2 Classification

Entity resolution is a binary classification problem. Given a pair of training and test set consists of match and non-match records, the classifier predicts if a pair of unseen data are matched. One of the biggest challenges that entity matching faces is *Class Imbalance*. Due to the extent of non-match records, in real-world datasets, the class distributions are highly imbalanced. To tackle this issue, we construct our training and test set, such as 66% of records as non-matched and the rest as matched. We apply XGBoost, which we overview in [Section 4.1](#), as our baseline classifier due to its performance and accuracy.

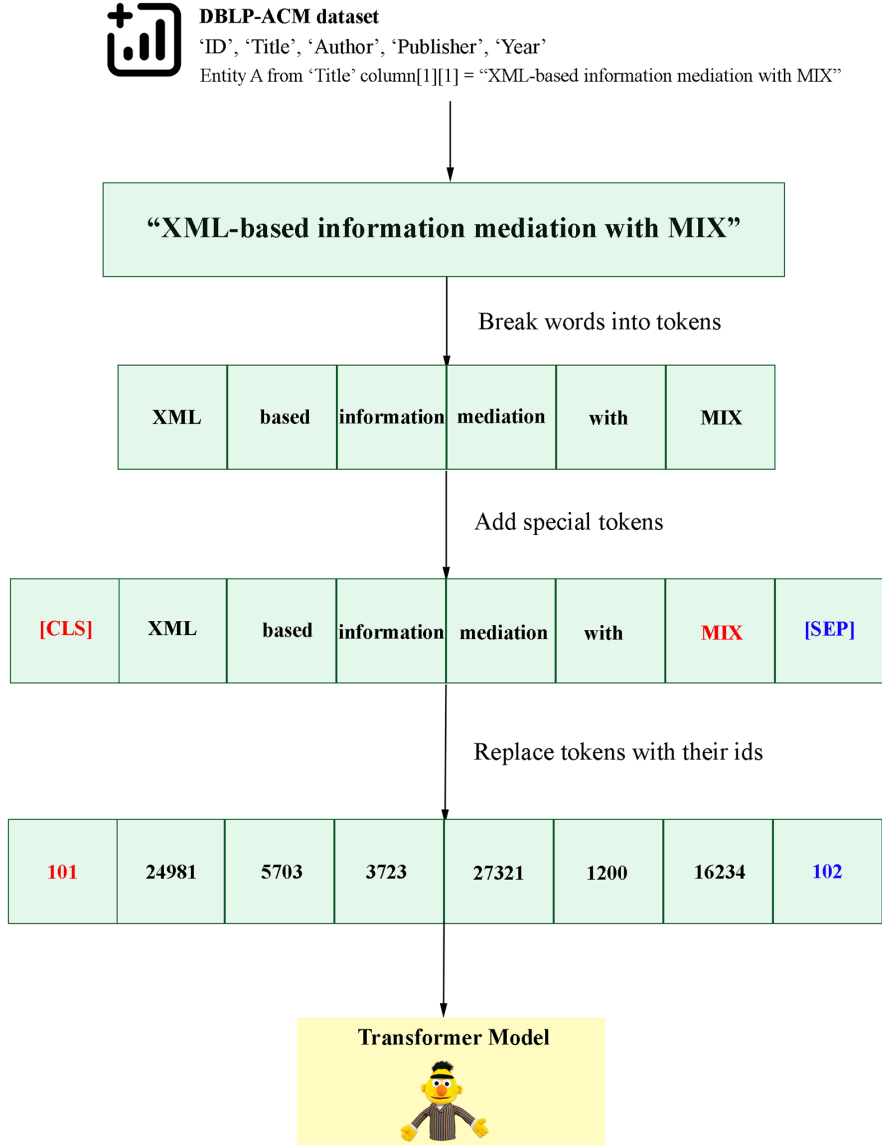


Figure 3.2: Overview of the tokenization and word embedding step in a transformer architecture.

3.2.3 Evaluation

The essential evaluation metrics we use in our experimental evaluation are as follows:

- *Accuracy.* Metric used to calculate the ratio of the number of correct predictions to the total number of input samples. So in our case, accuracy is the ratio of the number of correct matched or non-matched prediction to the number of input.

$$Accuracy = \left(\frac{TP + TN}{TP + TN + FP + FN} \right)$$

Where TP = True Positives, TN = True Negatives, FP = False Positives, and FN = False Negatives.

- *F1 Score.* Metric used to measure the classification accuracy of the model. It is the

harmonic mean of Precision (fraction of exact predicted matched records among all records that predicated as matched) and Recall (fraction of exact predicted matched records to the total number of matched records in our dataset).

$$F1Score = 2 \left(\frac{Precision * Recall}{Precision + Recall} \right)$$

- *Logarithmic Loss*. Metric used to measures the performance of a classification model where the prediction input is a probability value between 0 and 1. it takes into account the uncertainty of the prediction based on how much it varies from the actual label. This gives us a more nuanced view into the performance of our model. *Logarithmic Loss* (LogLoss), works by penalising the false classifications. When working with LogLoss, the classifier must assign probability to each class for all the samples. Suppose, there are N samples belonging to M classes, then the Log Loss is calculated as below:

$$LogarithmicLoss = \left(\frac{-1}{N} \sum_{i=1}^N \sum_{j=1}^M y_{ij} * \log(P_{ij}) \right)$$

3.3 Summary

In this chapter, we present our thesis’s design, which is a general approach for applying pre-trained transformer models on entity resolution task. We finalize our research questions formulated to be answered as part of this thesis. Additionally, we broke our design into different steps such as tokenization, word embedding, or blocking to further detail. Finally, we conclude this chapter by providing information about the classification step and evaluation metrics we use in this thesis.

4 Experimental Setup

In the following chapter we provide all the information required to reproduce the evaluation results reported in this research:

- in [Section 4.1](#), we present a gentle introduction to XGBoost as our main classification model as well as some extra details regarding the selected hyper parameters.
- [Section 4.2](#) is dedicated to the description of two datasets used in the experiments.
- Finally, we conclude the chapter by providing a detailed specification hardware and software resources used, in [Section 4.3](#).

4.1 Classification Algorithm: XGBoost

The classification unit of our *Entity Resolution Transformer* approach practices a classifier to predict if the pair of embedded data is matched or non-matched. We choose the XGBoost algorithm due to execution speed and model performance between other machine learning algorithms. XGBoost is a decision tree-based ensemble Machine Learning algorithm that employs a gradient boosting technique for classification and regression problems. Decision tree-based algorithms tend to outperform all other algorithms in dealing with small-to-medium structured/tabular data. [\[HL03\]](#) shows that combining classifiers leads to a decrease in misclassification error in a wide range of applications. This method which is called *Bagging* aggregates prediction from multiple decision trees using a majority voting mechanism. Furthermore, [\[LW⁺02\]](#) presents random forests which add an additional layer of randomness to bagging. Random forests use a counter intuitive strategy that benefits from only a subset of features that are selected at random to build a forest (collection of decision trees). Decision Trees boosting, an alternative approach, emerged in [\[DC96\]](#). In a successful attempt to improve accuracy with some small risk of less coverage, models are built sequentially by minimizing the errors from previous models while increasing of high-performing models. Using a gradient boosting machine[\[Fri01\]](#) that employs a gradient descent algorithm to minimize errors in sequential models is a widely-used machine learning algorithm, due to its efficiency, accuracy, and interpretability.

XGBoost, an optimized version of gradient boosting algorithm, outperforms most machine learning algorithms for structured or tabular data[\[CG16\]](#). XGBoost stands for *eXtreme Gradient Boosting*, it was developed by Tianqi Chen and now is part of a wider collection of open-source libraries developed by the Distributed Machine Learning Community. A comprehensive benchmark done by Szilard Pafka¹ shows that XGBoost outperforms several other well-known implementations of gradient tree boosting. XGBoost achieves

¹<https://github.com/szilard/benchm-ml>

efficiency in time and performance through tree pruning, parallel processing, handling missing values, and regularization to avoid overfitting.

4.1.1 Hyperparameters

In order to tackle our first research question, we need to have enough knowledge about hyperparameters in our desired classifier. In machine learning, a hyperparameter is a parameter whose value is used to control the learning process [CDM15]. In this research, we want to examine the effect of tuning hyperparameters on final match/non-match classification task.

XGBoost authors classified the overall parameters into 3 categories:

- *General Parameters.* relate to overall functionality of XGBoost.
- *Booster Parameters.* depend on which booster you have chosen.
- *Learning Task Parameters.* decide on the learning scenario.

General parameters mostly are set automatically by XGBoost. Learning task parameters decide on the learning scenario. Given that the scikit-learn API of XGBoost is easy to use, we skip using learning task parameters (optimization) and focus on booster parameters. Due to this thesis's time and memory constraint, we choose five essential booster parameters to tune.

These five parameters are as summarized below:

- *max_depth.* It is the maximum number of nodes allowed from the root to the most distant leaf of a tree. While deep trees can model more complex relationships, but they are also prone to overfitting.
- *eta.* This parameter presents the learning rate. It corresponds to the amount of correction we make at each step. The lower rate of learning can make the model robust regarding overfitting.
- *gamma.* This parameter displays minimum loss reduction required to make a further partition on a leaf node of the tree. The larger gamma makes the model robust towards overfitting.
- *min_child_weight.* It is the minimum weight required to create a new node in the tree. Again, while a smaller number of this parameter lets the algorithm create children that correspond to fewer samples and model complex trees, it is also likely to overfit.
- *subsample.* This parameter measures the sampling of the dataset that is done at each boosting round. In other words, it corresponds to the ratio of training instances. The desired value of subsample is the percentage of randomly sample training data before growing trees, which can be beneficial to prevent overfitting.

4.2 Datasets

The authors of DeepMatching demonstrated that classical entity matching approaches do not perform well on challenging datasets. In order to test our attention-based transformers, we use two publicly available structured dataset pairs from different domains, as depicted in Table 4.1.

Dataset	Domain	Features Used	Features
Amazon-Google	Software	5	ID, Name, Description, Manufacturer, Price
DBLP-ACM	Citation	5	ID, Title, Authors, Venue, Year

Table 4.1: Overview of datasets.

Moreover, each dataset pair is followed by a *Ground Truth* dataset, which consists of mapping the structured tuples (ID) from each of the datasets in the pair, acting as a basis for entity matching.

4.3 Experimental Environment

In this section, we overview the essential details about the experiment setup we deploy, running the experiments of this thesis. We employ the following configurations as follows:

Machine Configuration

- **Operating System** macOS Catalina (Version 10.15.4 (19E287))
- **Processor** 2,7 GHz Dual-Core Intel Core i5
- **Memory** 8 GB 1867 MHz DDR3
- **Graphics** Intel Iris Graphics 6100 1536 MB

Programming Framework

- **Programming Languages** Python (Version 3.7.2)
- **Programming Tools** PyCharm community edition (Version 2016.2.3), Jupyter Notebook (Version 4.3.1), Google Colab
- **Libraries** Transformers, Scikit-learn (Version 0.22.2), Numpy (Version 1.18.4), Pandas (Version 1.0.3), PyTorch (Version 1.2.0), XGBoost (Version 0.90), Matplotlib (Version 3.2.1)

4.4 Summary

In this chapter, we provide an overview of XGBoost, our classification algorithm, and its parameters, which we attempt to tune in our research. Furthermore, we describe our input datasets and an additional ground-truth dataset, which are essential in our study. Finally, we provide crucial information about our experimental setup, including the details regarding the versions of used libraries to ensure the reproducibility of the results from now on.

5 Evaluation and Results

In the following chapter, we present and discuss the evaluation results for the experiments we conducted to address the research questions raised in (Section 3.1). This chapter consists of the following sections:

1. In Section 5.1, we discuss the effect of tuning hyper-parameters of an entity resolution classifier model, assuming using pre-trained embeddings from a fixed transformer.
2. Section 5.2 is dedicated to comparing alternative pre-trained transformer models' performance on the accuracy of the overall entity resolution formulation.
3. In Section 5.3, we present the evaluation results for the experiments conducted by different classifiers.
4. Section 5.4 is a summarization about the fundamental findings of this evaluation set and the most interesting observations.

5.1 RQ1: Hyper-parameter Tuning

In this section, we present the evaluation results regarding the tuning hyper-parameters of the entity resolution classifier model.

- **Experiment Setup:**

In this experiment, we fixed DistilBert as our pre-trained transformer model and XGBClassifier for the classification core (between matched/non-matched pairs). Due to challenges imposed by operating large-scale pre-trained models in under-constrained (computational) systems, using DistilBert as a smaller, faster, and lighter version of Bert is inevitable [SDCW19]. XGBoost classifier is our choice for the classification layer of the model. We considered the number of decision trees (`n_estimators`), size of decision trees (`max_depth`), learning rate (`learning_rate`), minimum loss reduction required to make a further partition (`gamma`), the minimum sum of instance weight needed in a child (`min_child_weight`), and subsample ratio of the training instances (`subsample`) as hyper-parameters we want to tune. We set LogarithmicLoss as our evaluation metric. Furthermore, we used GridSearchCV of the scikit-learn library to evaluate all the possible combinations of parameter values. Due to the memory limit challenge, we avoided exploring a combination of two parameters at the same time. In our practice, we fixed the first parameter with the best result and checked the second parameter given the first one.

- **Expected Results:**

In general, we expect the number and size of decision trees plus learning rate (`n_estimators`, `max_depth`, and `eta`) play an important factor regarding the model performance due to their nature. Regarding the effect of tuning hyper-parameters on different datasets, we expect to see a more significant impact on the Amazon-Google dataset than DBLP-ACM. The later already benefits from high accuracy because of its structured content.

Dataset	Before Tuning	After Tuning
Amazon-Google	85.80	87.01
DBLP-ACM	90.66	90.71

Table 5.1: Effect of hyper-parameter tuning for all datasets based on the accuracy.

- **Results:**

We observed that generally, the number of decision trees, the size of trees, and the learning rate could make a noticeable change regarding the logarithmic loss. However, this amount was different between datasets. The other parameters also played an important role in exceptional cases. [Figure 5.1](#) shows that while running a hyper-parameter tuning on the Amazon-Google dataset, we observed that increasing number of decision trees could be beneficial. Still, we don't see a considerable change after 150 estimators. We recorded a similar behavior with an increase in the size of decision trees. The model suffered fast with expanding the depth of the tree from 3 to 4. The model performed the best on the learning rate = 0.1. With the already best configuration of `n_estimators`, `max_depth`, and `learning_rate`, the `gamma` parameter did not affect the model immensely. Parameters, namely, `min_child_weight` and `subsample` parameters followed the same pattern and did not make a difference.

And finally, hyper-parameter tuning on the ACM-DBLP dataset resulted in an unstable configuration. [Figure 5.2](#) depicts that after two runs, the number of decision trees did not change the loss, and the number of 150 estimators got fixed. Similar behavior had been observed during increasing the size of the tree, which finalized with `n=3`. The model did not show any difference after increasing the learning rate from 0.10. Again, the difference that the `gamma` parameter made was so small that we could ignore it. Increasing `min_child_weight` underwent a considerable loss, and we stuck with 1.0. The `subsample` parameter showed an unpredictable pattern, and we fixed the parameter `n=0.60`. Ultimately, the model using tuned hyper-parameter values had a slightly better overall performance. The effect on Amazon-Google dataset was higher than ACM-DBLP.

[Table 5.1](#) shows the performance of the model before and after hyper-parameter tuning task. As we expected, the effect of tuning for the Amazon-Google dataset was better than the DBLP-ACM dataset, while the margin between the two models is relatively small.

- **Analysis:**

Parameters such as number of trees and the size of trees influenced the performance because of their nature. Increasing the number of trees means new trees attempt to model and correct the errors made by previous trees. While more trees can

Dataset: AMAZON-GOOGLE

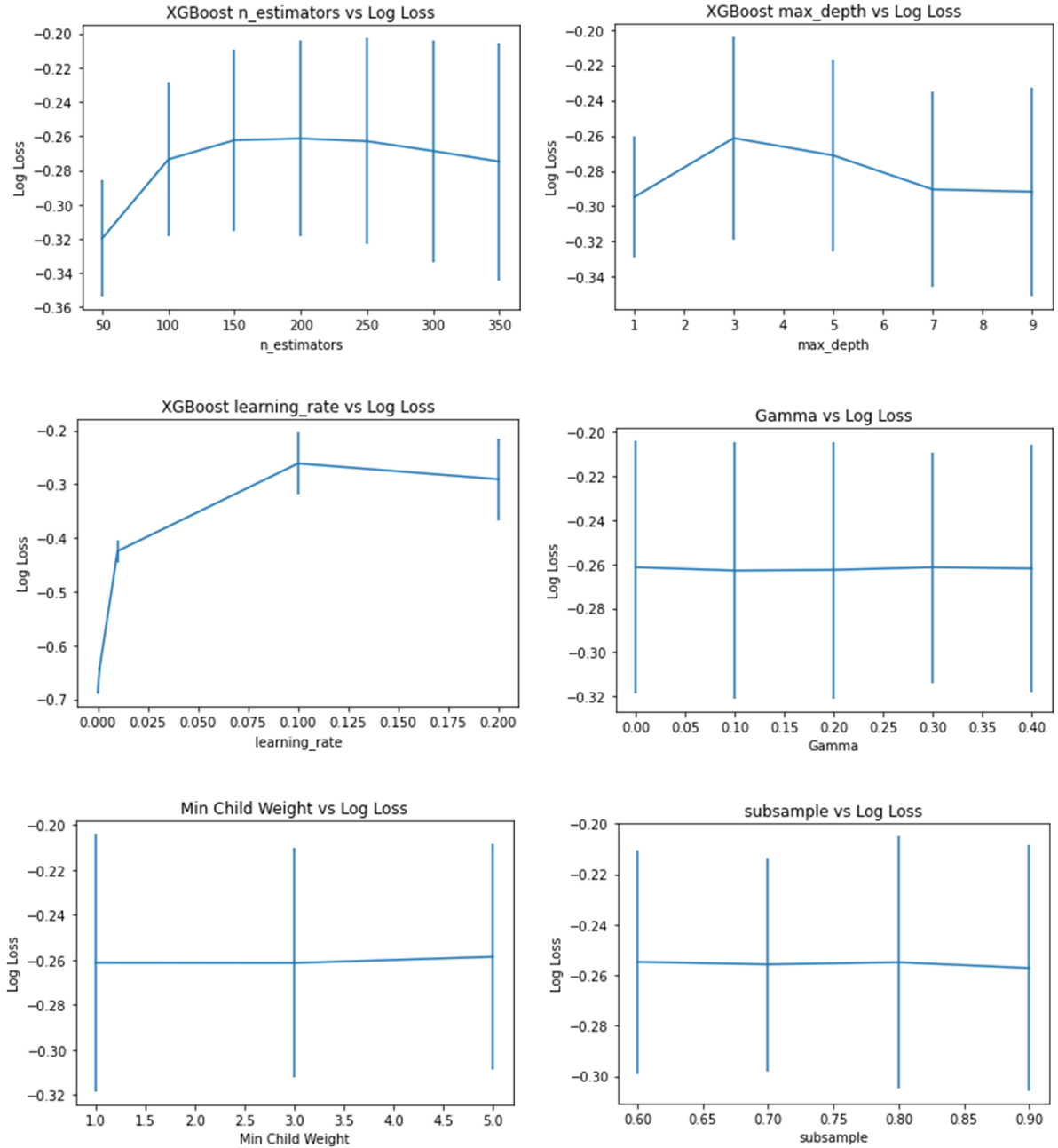


Figure 5.1: Overview of hyper-tuned results on Amazon-Google dataset.

be advantageous for improving performance, from one point, there is a limit that the model reaches a point of diminishing returns. The size of decision trees and a trade-off between a weak learner and a deep learner is another leverage to enhance the performance. Hyper-parameter optimization improved the performance of the model, while the difference is not significant. The role of hyper-parameter tuning on the DBLP-ACM dataset is almost disappeared due to the structured and clean nature of data. The results in Figure 5.5 indicate that the effect of pre-processing in such a model can be more crucial than hyper-parameter tuning. Based on [PR18],

Dataset: DBLP-ACM

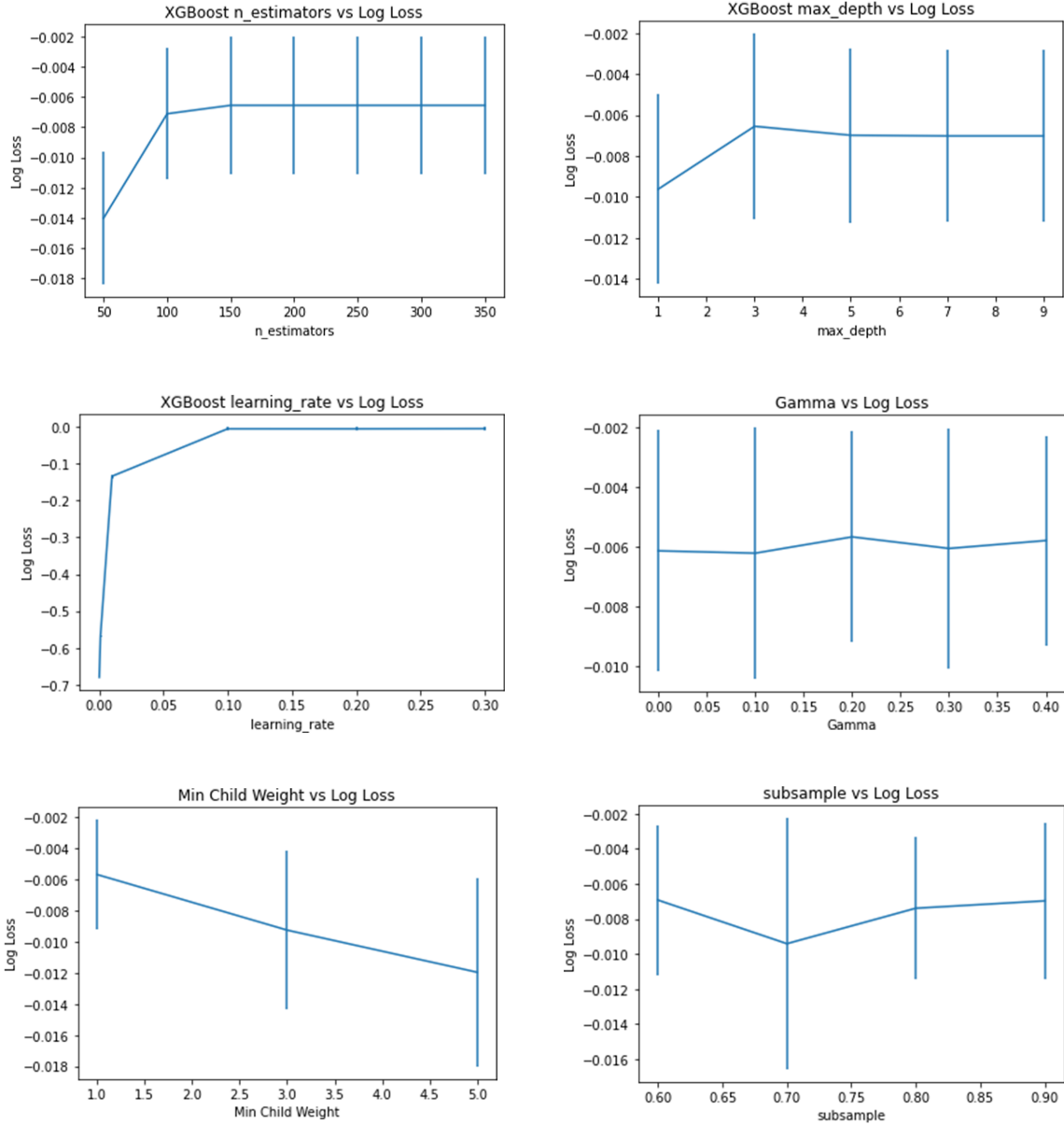


Figure 5.2: Overview of hyper-tuned results on DBLP-ACM dataset.

deploying other optimization methods such as Hyperopt can improve the performance better than Grid Search and Random Search. Conclusively, the choice of tuning Xgboost parameters is critical. Because of its complicated characteristics, xgboost-tuning has been an area of interest over the last years.

- **Takeaways:**

Hyper-parameter tuning on classifiers can improve the performance of the entity resolution model. Optimizing the most critical parameters such as the number of decision trees and the size of the tree can enhance the performance in a short time. While the hyper-parameter optimization impact on the model is proved, the

difference is insignificant. Given xgboost as a fixed classifier, deciding on whether to tune the parameters is solely by the designer. In critical applications such as fraud detection, a small improvement is a huge success.

5.2 RQ2: Comparing the Performance of Transformers

In this section, we present the evaluation results comparing four of the most recent attention-based transformer architectures performance on the accuracy of the overall entity resolution formulation.

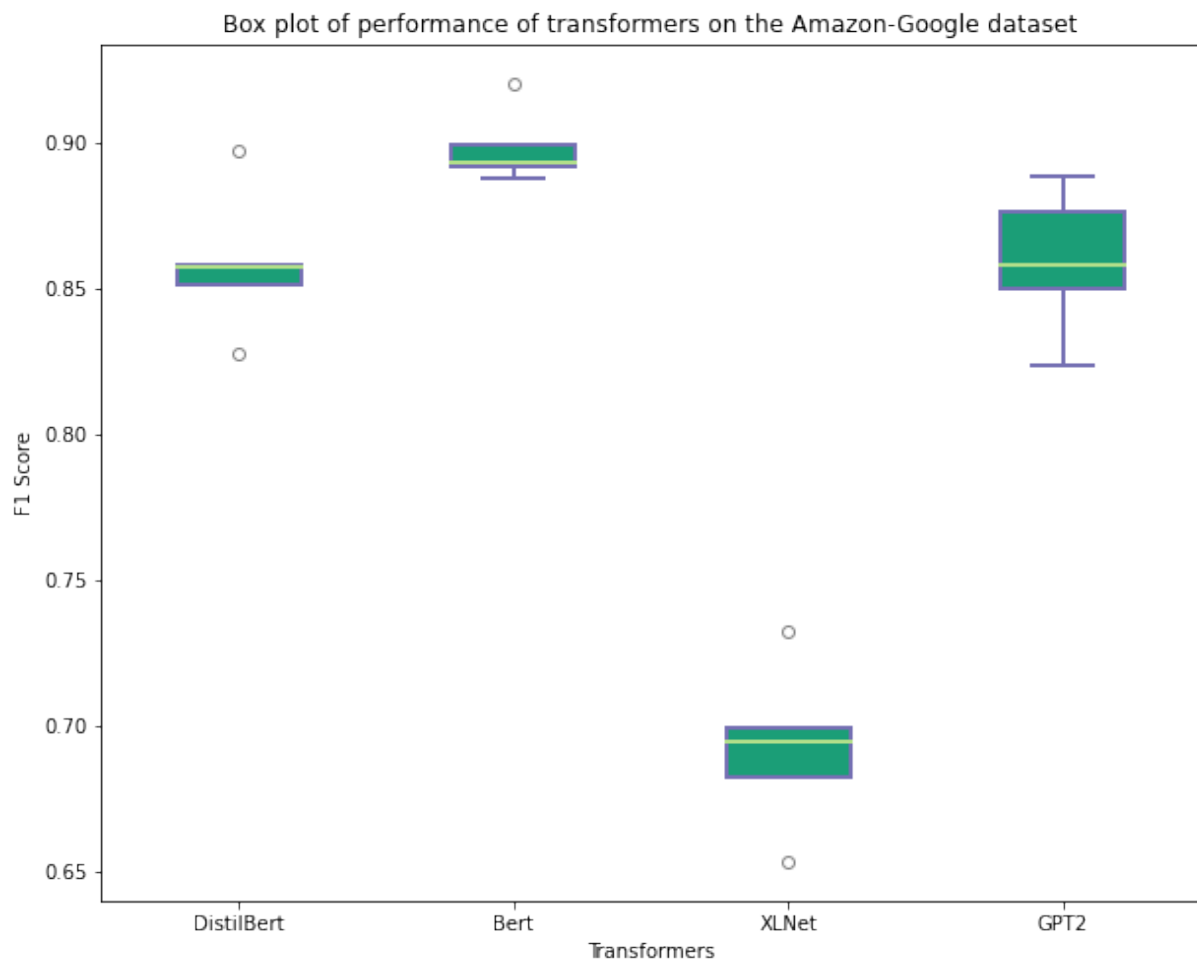


Figure 5.3: Box-plot overview of different transformers performance on Amazon-Google dataset.

- **Experiment Setup:**

In this experiment, we deployed BERT, DistilBERT, GPT2, and XLNet as transformers to process the data. All transformers are available to use in the *Hugging Face* transformer library on GitHub¹. As for the classification layer, we used a scikit-learn XGBClassifier. The performance of transformer models measured by f1 score. We empirically compare the four approaches on two different datasets.

¹<https://github.com/huggingface/transformers>

Due to the time constraint, we did not apply SentencePiece, proposed in [KR18], as XLNet tokenizer.

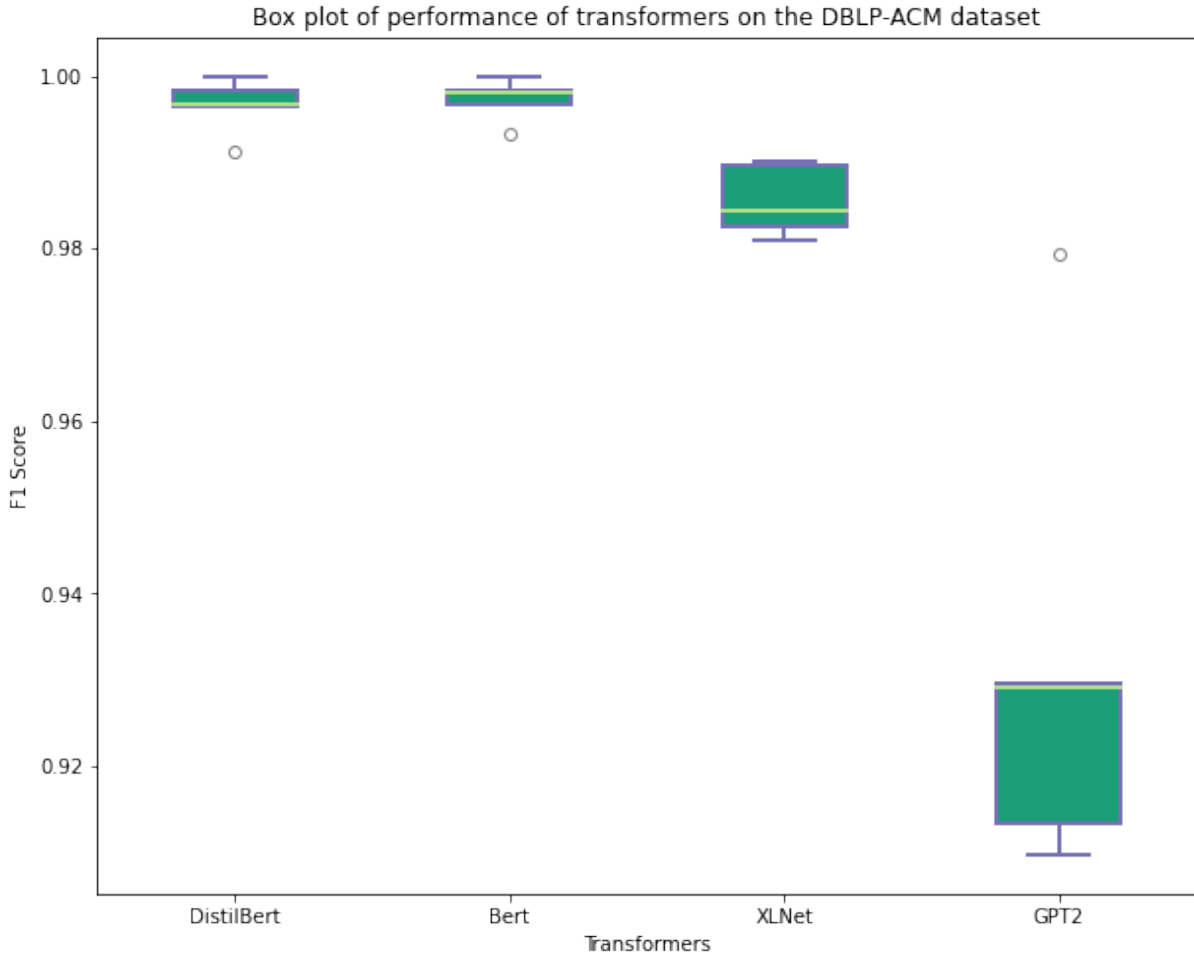


Figure 5.4: Box-plot overview of different transformers performance on DBLP-ACM dataset.

- **Expected Results:**

Overall, we expect BERT family transformers outperform other transformers due to their massive pre-training effort. Precisely, we anticipate a smaller f1 score on Amazon-Google considering the size of the dataset. Skipping the use of SentencePiece generally should affect XLNet performance.

Transformer	F1 score	Standard deviation
DistilBERT	85.80	2.24
BERT	89.82	1.16
XLNet	69.21	2.55
GPT2	85.91	2.23

Table 5.2: F1 scores of transformers for Amazon-Google dataset.

- **Results:**

All transformer models perform remarkably well on the task of entity matching. Table 5.2, Table 5.3 show the mean f1 score for each transformer on the given datasets.

As we expected, BERT outperformed the other transformers on the Amazon-Google dataset even though the margin is small. Figure 5.3, Figure 5.4 depict the box

Transformer	F1 score	Standard deviation
BERT	99.73	0.23
DistilBERT	99.66	0.29
XLNet	98.55	0.37
GPT2	93.21	0.48

Table 5.3: F1 scores of transformers for DBLP-ACM dataset

plot overview of different transformers on the given datasets. BERT outperforms other transfer models running on the DBLP-ACM dataset as well. As we expected,

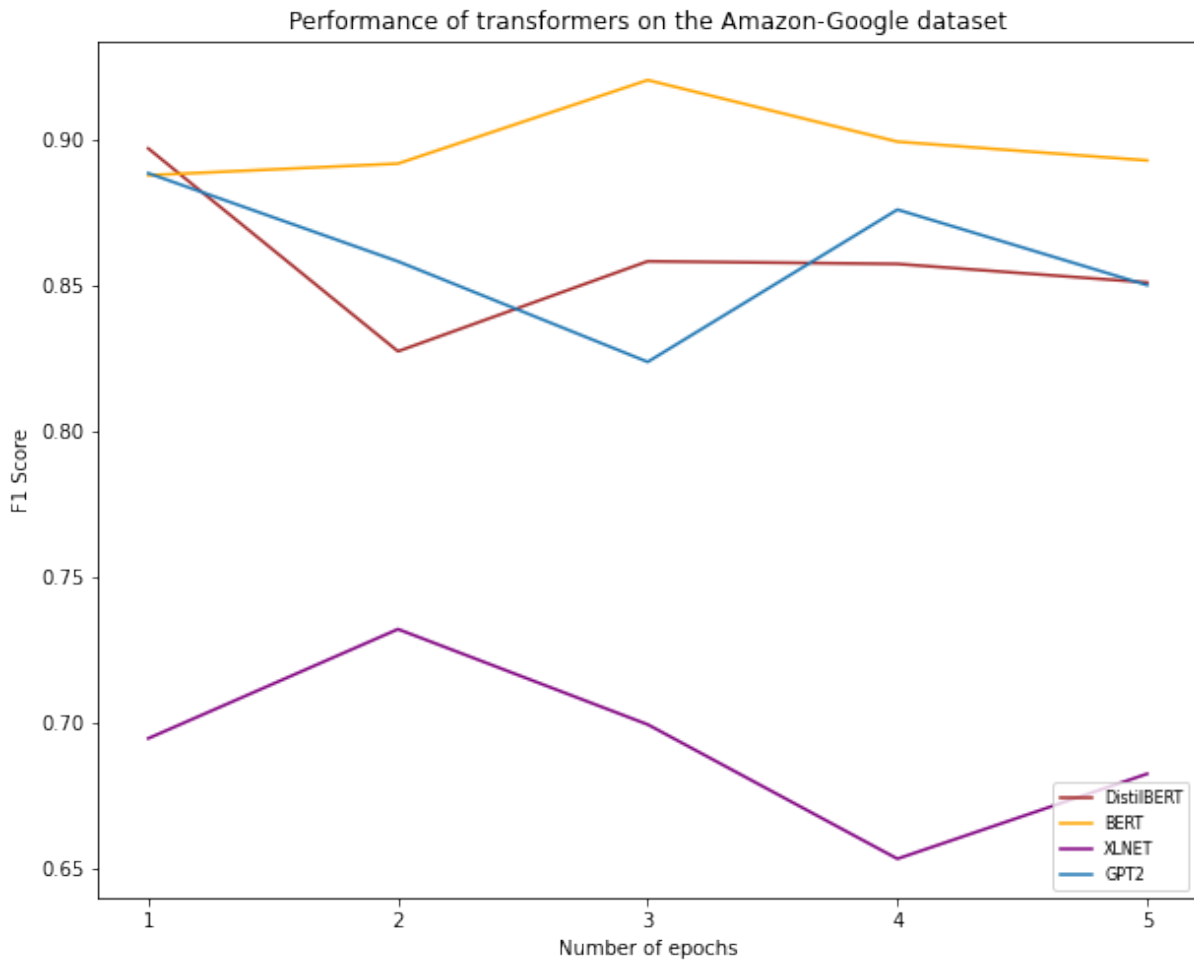


Figure 5.5: Comparison of different transformers performance on Amazon-Google dataset.

the BERT family recorded better performance comparing to XLNet and GPT2. Avoiding tokenization by SentencePiece generated a notable influence on XLNet performance. XLNet transformer architecture recorded a satisfactory f1 score on the DBLP-ACM dataset. Figure 5.5 shows that after two epoch of training, most experiments range within a 5% interval of their peak performance. As Figure 5.6 confirms, after 2-3 epochs, almost all experiments converge to their top performance.

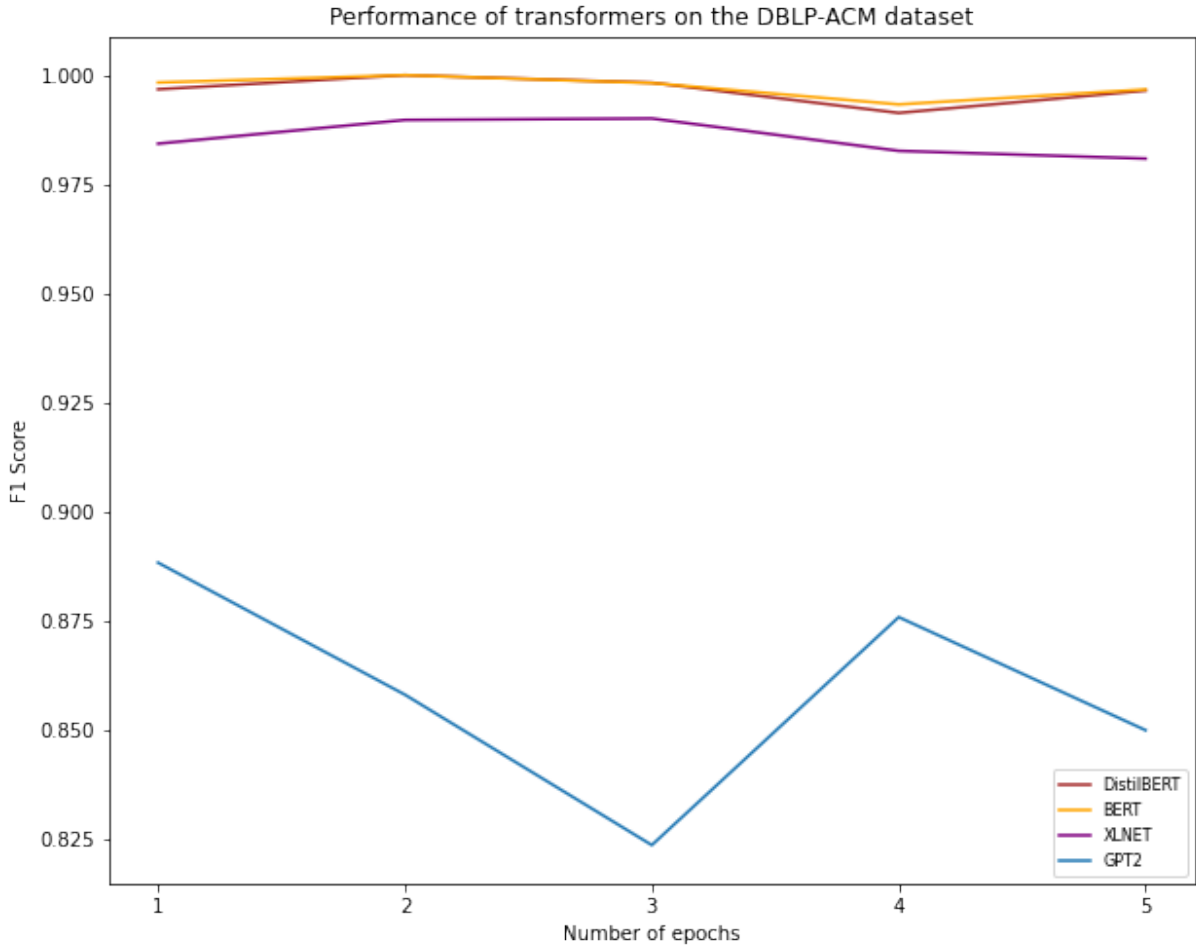


Figure 5.6: Comparison of different transformers performance on DBLP-ACM dataset.

- **Analysis:**

The BERT family record a high accuracy on benchmark datasets due to their massive pre-training on large amounts of text. This characteristic makes them powerful to handle language-based tasks and in specific entity resolution tasks. BERT exclusively achieved higher performance due to its larger model. The original BERT was trained on the BookCorpus and English Wikipedia, comprising about 16 GB of text. DistilBERT met a similar performance which shows it can be used in limited configurations. XLNet regularly performs well on the entity resolution task, and this proved by [BS20]. The poor performance of XLNet on the Amazon-Google dataset can be justified by a lack of pre-tokenized input proposed by [KR18]. GPT2 achieved better performance than traditional deep learning approaches based on the results obtained by [BS20]. Although, GPT2 needs particular design architecture and fine-tuning to defeat BERT family numbers on benchmark datasets.

- **Takeaways:**

BERT family members are the best choice of transformer model in the proposed approach dealing with entity resolution. DistilBERT, due to its smaller and faster nature, can be very useful in dealing with time and memory limitations. XLNet tokenizing by Sentencepiece supposedly can challenge BERT family transformers. Deploying GPT2 to tackle entity resolution under the proposed approach, needs a fine-tuning and special architectural decision.

5.3 RQ3: Comparing the Performance of Classifiers

In this section, we present the evaluation results comparing three different alternative classifiers' performance with xgboost on the accuracy of the overall entity resolution formulation.

- **Experiment Setup:**

In this experiment, we used BERT transformer architecture as an entity resolution engine. Furthermore, we deployed Logistic Regression, k nearest neighbors (KNN), and gradient boosting machines (GBM) for the classification layer. There is no free lunch in machine learning [Wol02], but these classifiers are the most popular and effective algorithms dealing with binary classification (matched/non-matched pairs). In working with K nearest neighbor algorithm, we ran a series of experiments and decided to fix k as k=1, which outperforms other configurations facing binary classification problems. To avoid overfitting, we practiced standard k-fold cross-validation (k=3).

Classifier	F1 score	Standard deviation
XGBoost	89.82	1.16
LogReg	91.35	1.68
GBM	89.29	1.26
KNN	74.58	1.99

Table 5.4: F1 scores of classifiers for Amazon-Google dataset.

- **Expected Results:**

We expect XGBoost to outperform other well-known classifiers, indicating that it is the best choice for constructing the classification model. XGBoost mostly outperforms implementations of gradient tree boosting on different tasks, and we suspect that the same behavior will be followed. Historically, Logistic Regression records a high accuracy with binary classification, and excellent performance can be expected. The results are measured, similar to [ETJ⁺17], by an f1 score where recall is the ratio of true matches predicted vs. all true matches.

Classifier	F1 score	Standard deviation
XGBoost	99.66	0.29
LogReg	99.73	0.20
GBM	99.59	0.36
KNN	98.83	0.27

Table 5.5: F1 scores of classifiers for DBLP-ACM dataset.

- **Results:**

Table 5.4, Table 5.5 show the classifiers' performance on the task of binary classification between match and non-matches. For each dataset, we report the performance of three alternative classifiers. We observe that all classifiers except KNN perform remarkably good on the classification layer. To our surprise, Logistic Regression outperformed the other classifiers by a small margin on all chosen datasets. Since

we already achieved very high accuracy on the DBLP-ACM dataset, the difference between top classifiers is very small. As we expected, XGBoost outperformed GBM in all our of datasets. KNN performed worst, averaged on all datasets. While KNN is the worst configuration between our classifiers, This algorithm performs well on DBLP-ACM. Logistic Regression achieved an f1 score of 91.35% (standard deviation

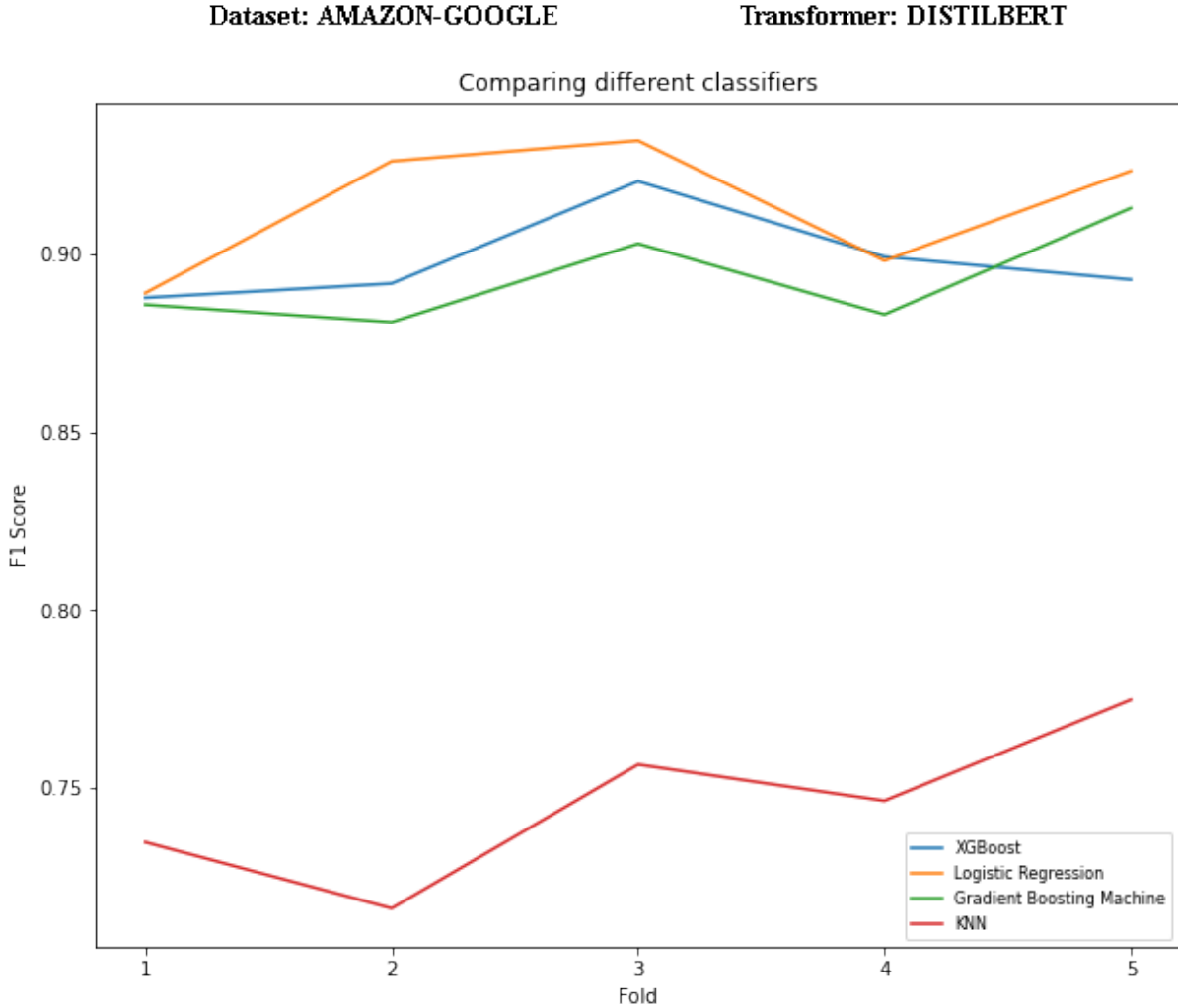


Figure 5.7: Comparison of different classifiers performance on Amazon-Google dataset.

= 1.68%) on the Amazon-Google dataset, which is the best result between other alternative classifiers. Xgboost and GBM, the two boosting algorithms, achieved an f1 score of 89.82% and 89.29%, respectively (std = 1.16% and 1.26% in the same order). K nearest neighbor algorithm resulted in 74.58% concerning f1 with standard deviation = 1.97%. On DBLP-ACM dataset, Logistic Regression achieved an f1 score of 99.73% (standard deviation = 0.20%) on Amazon-Google dataset which is the best result between other alternative classifiers. Xgboost and GBM achieved N f1 score of 99.66% and 99.59% respectively (std = 0.29% and 0.36% in the same order). K nearest neighbors algorithm resulted in 98.83% in respect to f1 with standard deviation = 0.27%.

- **Analysis:**

Using a different method for estimating parameters, which gives a better and un-

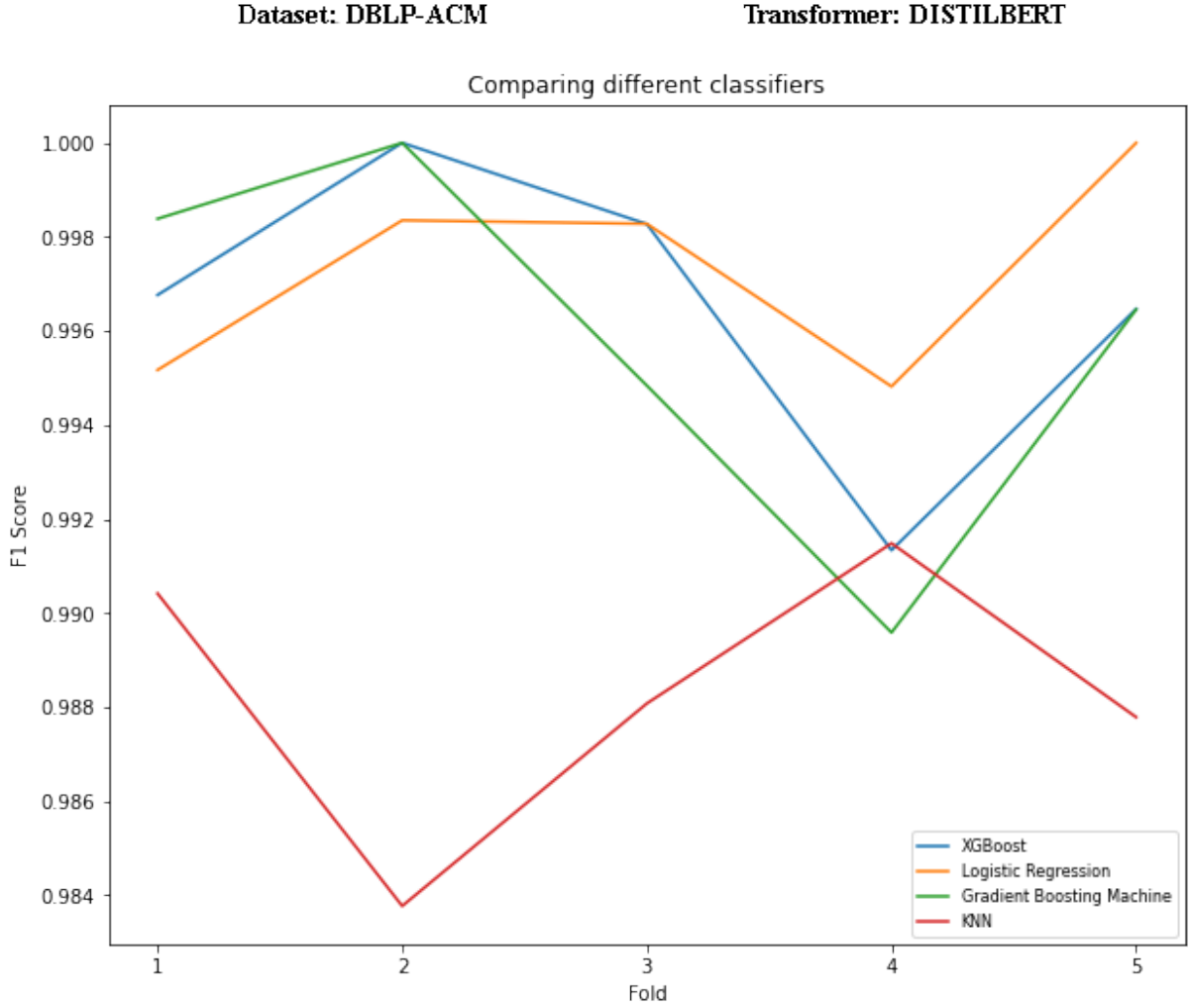


Figure 5.8: Comparison of different classifiers performance on DBLP-ACM dataset.

biased result with lower variances, makes Logistic Regression a super-powerful algorithm dealing with binary classification. Also, scientifically Logistic Regression performs the best when the noise ratio is low. XGBoost outperforms the other gradient tree boosting methods due to its different way of creating trees.

- **Takeaways:**

Given a structured output such as what we create on our proposed design, Logistic Regression can be the best classifier option dealing with low-noise data. Dealing with more complicated datasets, gradient tree boosting algorithms are beneficial. In particular, XGBoost is the best option due to its execution speed and model performance. While KNN did not record a satisfying return on this task, the other machine learning algorithms, such as NN or NB, can be examined.

5.4 Summary

In this chapter, we analyze the importance of tuning the classifier (XGBoost) parameters

on the transformer-based entity resolution model’s overall performance. We overview and evaluate the performance of four modern transformer architectures on our proposed design and compare them based on the f1 score. Finally, we conclude this chapter by replacing different classifiers into our classification unit and compare the results obtained by three classifiers with our standard XGBClassifier.

6 Related Work

In this chapter, we review related work concerning the main topics we aimed in our thesis. This chapter broke into three brief sections, as follows:

- In [Section 6.1](#), we begin with a short review of studies based on entity resolution task and briefly discuss some traditional works on this field.
- In [Section 6.2](#), we exploit several deep learning methods applying on entity resolution workflow steps and compare our work results with traditional approaches and deep learning methods.
- In [Section 6.3](#), we briefly overview emerging works on transformer architectures and their application on different natural language processing tasks.

6.1 Entity Resolution Approaches

Entity Resolution (ER) or Reference Reconciliation [[DHM05](#)] or Record Linkage [[FS69](#)] or Merge-Purge [[HS95](#)] or Entity Matching [[KR10](#)] has been studied extensively for many decades. While [[PIP20](#)] draws a generational boundary between ER methods, different approaches have been proposed to tackle entity resolution problems. Entity resolution approaches can get categorized as rule-based methods which described in [[BM03](#)], [[SME⁺17](#)], [[WLYF11](#)], and machine learning solutions presented in [[CR02](#)], [[FS69](#)], [[SB02](#)]. The whole framework of the entity resolution task and its challenges studied in [[Chr](#)], [[EIV06](#)], [[CES15a](#)]. In order to reduce human-involvement in ER, a big project called *Magellan* [[KDSG⁺16](#)] has been implemented to provide practical tools and libraries for the whole entity resolution pipeline with interesting results on structured data. We compare our proposed transformer architecture results with Magellan in the following section.

6.2 Applying Deep Learning for ER

Exploiting external knowledge to achieve higher performance, motivated the use of deep learning methods for entity resolution task. Different work on applying deep learning models on ER workflow steps has been considered. As for schema matching, a model called SEMPROP [[FMQ⁺18](#)] introduced, which applies two different semantic and syntactic matchers to find similar attributes. AutoBlock [[ZWS⁺20](#)] considers deep learning architecture for the blocking process, which combines similarity-preserving representation learning with the nearest neighbor search. While most of the deep learning methods'

efforts focus on entity matching step, we already discussed DeepER [ETJ⁺17] and Deep-Matcher [MLR⁺18] as the two most popular ones. Multi-Perspective Matching is another DL approach on matching, which adaptively selects the optimal similarity measures for heterogeneous attributes [FHS⁺19]. In the Table 6.1, we compare the performance

Dataset	MG	DeepM	Transformer _{BEST}
DBLP-ACM	91.9	98.1	99.73
Amazon-Google	48.4	61.2	89.82

Table 6.1: Comparing the best performing transformer model with Magellan and Deep-Matcher

of transformer models with classical models such as Magellan (MG) and DeepMatcher (DeepM). We report the best performing of the four DL models, MG result, and the best performing transformer. The result for the classical models obtained by similar papers in the field based on the F1 score [MLR⁺18]. We see that while deep learning models outperform traditional MG approaches by a large margin, transformer architecture even achieves higher performance compared to classical deep learning methods. In both datasets, the margin between our best transformer model with classical approaches is large, but in a more challenging dataset, this margin is more significant.

6.3 Works on Transformer Architecture

The idea of the transformer first appeared on the famous paper “Attention is all you need” [VSP⁺17], and it became a popular technique for natural language processing (NLP). Introduction of BERT [DCLT18], which combines the transformer architecture with heavy unsupervised pre-training, achieved state-of-the-art results in many NLP tasks. Since then, many new BERT-based approaches have been proposed to assess BERT power on different tasks. Works such as BERT-DST (dialogue state tracking with BERT) [CL19], Vd-BERT (vision and language transformer) [WJL⁺20], Pixel-BERT (aligning image pixels with text by BERT) [HZL⁺20], X-BERT (extreme multi-label text classification with BERT) [CYZ⁺19], LIMIT-BERT (linguistic informed multi-task BERT) [ZZZ19], HSI-BERT (hyperspectral image classification) [HZY⁺19], K-BERT (enabling language representation with knowledge graph) [LZZ⁺20], KG-BERT (knowledge graph completion) [YML19], videoBERT (video and language representation learning) [SMV⁺19], have investigated BERT’s power and functionality on different tasks. Pre-trained models that incorporate BERT as its core and pre-trained on the domain-specific corpus, namely SciBERT (scientific publications) [BLC19], ClinicalBert (clinical notes) [HAR19], and BioBERT (biomedical text) [LYK⁺20] have shown to return better performance compared to the general approach. While succeeding papers attempted to fine-tune BERT to obtain even higher results on NLP tasks such as text classification [SQXH19], extractive summarization [Liu19], or ranking [QXLL19], the other branches of BERT family, in particular, DistillBERT (which we employed in this thesis) [SDCW19], and RoBERTa [LOG⁺19] proved to be beneficial for general NLP tasks. BERT’s lack of justification for processing masked tokens independent of each other led to various state-of-art transformer approaches such as XLNet [YDY⁺19] and GPT2 models [RWC⁺19], which we discussed in the thesis. Another family of transformers that we did not enclose in our research due

to time constraints is CTRL [KMV⁺19], which relies on control codes to predict which parts of the training data are most likely given a sequence. Transformer architectures have traditionally been used for NLP-tasks, and there is little literature to our knowledge that used transformer power on entity resolution task.

7 Conclusion and Future Work

In this chapter, we summarize our work and present the conclusions obtained from this thesis work. We also propose some future practices which may extend this study. This chapter is structured as follows:

- In [Section 7.1](#), we present the important conclusions derived from our thesis work.
- In [Section 7.2](#), we discuss some modifications which can be included in future works.

7.1 Conclusion

This thesis endeavors to apply different attention-based transformer models on the task of *Entity Resolution* (ER). Notably, we aim to compare various state-of-the-art transformer architectures regarding the process of entity matching. Accordingly, we design and implement an ER architecture that satisfies exercising any transformer to pre-train and generates word representations and any classifier to decide if a pair of given embeddings are matched. During the study, we expanded our research questions to one specific issue to know to what extent, tuning the classifier unit can improve our matching classification. To tackle this issue, we choose the XGBoost classifier, which outperforms different machine learning algorithms in many general tasks, as the baseline. We employ two entity resolution dataset pairs with two different domains, including various attributes, namely Amazon-Google and DBLP-ACM. We use four different modern transformer architecture such as BERT, DistilBERT, XLNet, and GPT2 as transformer choice. In this research, we also investigate the use of varying classifiers for match and non-match classification. We see entity resolution as a binary classification problem and choose Logistic regression, KNN, GBM, and XGBoost as classifier. We evaluate our models based on their ability to classify match and non-match pairs using an F1 score. We report that tuning the classifier (in this case, XGBoost) generally does not make a huge difference, but it can be beneficial in critical applications. We also conclude that BERT outperforms other transformers based on our proposed design and is the optimal choice for entity resolution task. Regarding the classifier's choice, we report that the Logistic regression classifier outperforms different chosen classifiers in our binary classification problem.

7.2 Future Work

Our thesis attempts to apply transformer models on the task of Entity Resolution (ER). Due to time and memory constraints, we made several assumptions and choices to make our proposed approach work. Modifying and analyzing the design itself can leave our

thesis with a lot of scope for future work. Improvements that can be made to our design are:

- *Fine-tuning the Classifier.* In case of using XGBoost as a classifier, instead of applying a simple greedy search on one parameter at a time, using a hyper-parameter grid with evaluating discrete subspace of all possible hyper-parameters at the same time can be included as future work. XGBoost hyperparameter tuning, due to its novelty, is still an open-ended area of study. Choosing a simpler classifier which can get tuned more efficiently, may show the impact of parameter-tuning better.
- *Transformer Model.* Through our research, we skipped SentencePiece tokenization when we deployed the XLNet transformer model. Adjusting the current design with specified tokenization (such as SentencePiece for XLNet) may result in higher performance. State-of-the-art blocking methods can investigate the blocking step of our design. Applying recent transformer models, namely GPT3, RoBERTa, and modified CTRL transformer model, may improve this research result.
- *Classification Unit.* We investigate four different classifiers for our binary classification problem. Based on no free lunch theorem, the optimal classifier can still be beyond our research area scope.

Bibliography

- [Ala18] Jay Alammam. The illustrated bert, elmo, and co. (how nlp cracked transfer learning)), Dec 2018. URL: <http://jalammar.github.io/illustrated-bert/>.
- [Ala19] Jay Alammam. The illustrated gpt-2 (visualizing transformer language models), Aug 2019. URL: <http://jalammar.github.io/illustrated-gpt2/>.
- [ALM16] Sanjeev Arora, Yingyu Liang, and Tengyu Ma. A simple but tough-to-beat baseline for sentence embeddings. 2016.
- [ARCC⁺19] Rami Al-Rfou, Dokook Choe, Noah Constant, Mandy Guo, and Llion Jones. Character-level language modeling with deeper self-attention. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, pages 3159–3166, 2019.
- [AW18] Hiba Abu Ahmad and Hongzhi Wang. An effective weighted rule-based method for entity resolution. *Distributed and Parallel Databases*, 36(3):593–612, 2018.
- [BBS05] Mikhail Bilenko, S Basil, and Mehran Sahami. Adaptive product normalization: Using online learning for record linkage in comparison shopping. In *Fifth IEEE International Conference on Data Mining (ICDM'05)*, pages 8–pp. IEEE, 2005.
- [BCB14] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural machine translation by jointly learning to align and translate. *arXiv preprint arXiv:1409.0473*, 2014.
- [BDMNW12] Christoph Böhm, Gerard De Melo, Felix Naumann, and Gerhard Weikum. Linda: distributed web-of-data-scale entity matching. In *Proceedings of the 21st ACM international conference on Information and knowledge management*, pages 2104–2108, 2012.
- [BDVJ03] Yoshua Bengio, Réjean Ducharme, Pascal Vincent, and Christian Jauvin. A neural probabilistic language model. *Journal of machine learning research*, 3(Feb):1137–1155, 2003.
- [BGJM17] Piotr Bojanowski, Edouard Grave, Armand Joulin, and Tomas Mikolov. Enriching word vectors with subword information. *Transactions of the Association for Computational Linguistics*, 5:135–146, 2017.
- [BGMM⁺09] Omar Benjelloun, Hector Garcia-Molina, David Menestrina, Qi Su, Steven Euijong Whang, and Jennifer Widom. Swoosh: a generic approach to entity resolution. *The VLDB Journal*, 18(1):255–276, 2009.

- [BLC19] Iz Beltagy, Kyle Lo, and Arman Cohan. Scibert: A pretrained language model for scientific text. *arXiv preprint arXiv:1903.10676*, 2019.
- [BM03] Mikhail Bilenko and Raymond J Mooney. Adaptive duplicate detection using learnable string similarity measures. In *Proceedings of the ninth ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 39–48, 2003.
- [BMR11] Philip A Bernstein, Jayant Madhavan, and Erhard Rahm. Generic schema matching, ten years later. *Proceedings of the VLDB Endowment*, 4(11):695–701, 2011.
- [BS19] Ursin Brunner and Kurt Stockinger. Entity matching on unstructured data: an active learning approach. In *2019 6th Swiss Conference on Data Science (SDS)*, pages 97–102. IEEE, 2019.
- [BS20] Ursin Brunner and Kurt Stockinger. Entity matching with transformer architectures-a step forward in data integration. In *International Conference on Extending Database Technology, Copenhagen, 30 March-2 April 2020*, 2020.
- [CCX⁺04] Hsinchun Chen, Wingyan Chung, Jennifer Jie Xu, Gang Wang, Yi Qin, and Michael Chau. Crime data mining: a general framework and some examples. *computer*, 37(4):50–56, 2004.
- [CDM15] Marc Claesen and Bart De Moor. Hyperparameter search in machine learning. *arXiv preprint arXiv:1502.02127*, 2015.
- [CES15a] Vassilis Christophides, Vasilis Efthymiou, and Kostas Stefanidis. Entity resolution in the web of data. *Synthesis Lectures on the Semantic Web*, 5(3):1–122, 2015.
- [CES15b] Vassilis Christophides, Vasilis Efthymiou, and Kostas Stefanidis. Entity resolution in the web of data. *Synthesis Lectures on the Semantic Web*, 5(3):1–122, 2015.
- [CG16] Tianqi Chen and Carlos Guestrin. Xgboost: A scalable tree boosting system. In *Proceedings of the 22nd acm sigkdd international conference on knowledge discovery and data mining*, pages 785–794, 2016.
- [CGCB14] Junyoung Chung, Caglar Gulcehre, KyungHyun Cho, and Yoshua Bengio. Empirical evaluation of gated recurrent neural networks on sequence modeling. *arXiv preprint arXiv:1412.3555*, 2014.
- [CHB⁺15] Tianqi Chen, Tong He, Michael Benesty, Vadim Khotilovich, and Yuan Tang. Xgboost: extreme gradient boosting. *R package version 0.4-2*, pages 1–4, 2015.
- [Chr] P Christen. Data matching: concepts and techniques for record linkage, entity resolution, and duplicate detection. 2012.
- [Chr12] Peter Christen. *Data Matching: Concepts and Techniques for Record Linkage, Entity Resolution, and Duplicate Detection*. Springer Publishing Company, Incorporated, 2012.

- [CKLS01] Munir Cochinwala, Verghese Kurien, Gail Lalk, and Dennis Shasha. Efficient data reconciliation. *Information Sciences*, 137(1-4):1–15, 2001.
- [CKM00] William W Cohen, Henry Kautz, and David McAllester. Hardening soft information sources. In *Proceedings of the sixth ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 255–259, 2000.
- [CL19] Guan-Lin Chao and Ian Lane. Bert-dst: Scalable end-to-end dialogue state tracking with bidirectional encoder representations from transformer. *arXiv preprint arXiv:1907.03040*, 2019.
- [CLD⁺16] William M Campbell, Lin Li, C Dagli, Joel Acevedo-Aviles, K Geyer, Joseph P Campbell, and C Priebe. Cross-domain entity resolution in social media. *arXiv preprint arXiv:1608.01386*, 2016.
- [CR02] William W Cohen and Jacob Richman. Learning to match and cluster large high-dimensional data sets for data integration. In *Proceedings of the eighth ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 475–480, 2002.
- [CVMG⁺14] Kyunghyun Cho, Bart Van Merriënboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. Learning phrase representations using rnn encoder-decoder for statistical machine translation. *arXiv preprint arXiv:1406.1078*, 2014.
- [CYZ⁺19] Wei-Cheng Chang, Hsiang-Fu Yu, Kai Zhong, Yiming Yang, and Inderjit Dhillon. X-bert: extreme multi-label text classification with using bidirectional encoder representations from transformers. *arXiv preprint arXiv:1905.02331*, 2019.
- [DC96] Harris Drucker and Corinna Cortes. Boosting decision trees. In *Advances in neural information processing systems*, pages 479–485, 1996.
- [DCLT18] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*, 2018.
- [DHM05] Xin Dong, Alon Halevy, and Jayant Madhavan. Reference reconciliation in complex information spaces. In *Proceedings of the 2005 ACM SIGMOD international conference on Management of data*, pages 85–96, 2005.
- [DL15] Andrew M Dai and Quoc V Le. Semi-supervised sequence learning. In *Advances in neural information processing systems*, pages 3079–3087, 2015.
- [DNSW12] Uwe Draisbach, Felix Naumann, Sascha Szott, and Oliver Wonneberg. Adaptive windows for duplicate detection. In *2012 IEEE 28th International Conference on Data Engineering*, pages 1073–1083. IEEE, 2012.
- [Dun46] Halbert L Dunn. Record linkage. *American Journal of Public Health and the Nations Health*, 36(12):1412–1416, 1946.

- [DYY⁺19] Zihang Dai, Zhilin Yang, Yiming Yang, Jaime Carbonell, Quoc V Le, and Ruslan Salakhutdinov. Transformer-xl: Attentive language models beyond a fixed-length context. *arXiv preprint arXiv:1901.02860*, 2019.
- [EIV06] Ahmed K Elmagarmid, Panagiotis G Ipeirotis, and Vassilios S Verykios. Duplicate record detection: A survey. *IEEE Transactions on knowledge and data engineering*, 19(1):1–16, 2006.
- [ETJ⁺17] Muhammad Ebraheem, Saravanan Thirumuruganathan, Shafiq Joty, Mourad Ouzzani, and Nan Tang. Deeper-deep entity resolution. *arXiv preprint arXiv:1710.00597*, 2017.
- [FHS⁺19] Cheng Fu, Xianpei Han, Le Sun, Bo Chen, Wei Zhang, Suhui Wu, and Hao Kong. End-to-end multi-perspective matching for entity resolution. In *Proceedings of the 28th International Joint Conference on Artificial Intelligence*, pages 4961–4967. AAAI Press, 2019.
- [Fie05] Stephen E Fienberg. Homeland insecurity: Datamining, terrorism detection, and confidentiality. *Bull. Internat. Stat. Inst*, page 8, 2005.
- [FMQ⁺18] Raul Castro Fernandez, Essam Mansour, Abdulhakim A Qahtan, Ahmed Elmagarmid, Ihab Ilyas, Samuel Madden, Mourad Ouzzani, Michael Stonebraker, and Nan Tang. Seeping semantics: Linking datasets using word embeddings for data discovery. In *2018 IEEE 34th International Conference on Data Engineering (ICDE)*, pages 989–1000. IEEE, 2018.
- [FNV⁺11] Glenn Fowler, Landon Curt Noll, Kiem-Phong Vo, Donald Eastlake, and Tony Hansen. The fnv non-cryptographic hash algorithm. *Ietf-draft*, 2011.
- [Fri01] Jerome H Friedman. Greedy function approximation: a gradient boosting machine. *Annals of statistics*, pages 1189–1232, 2001.
- [FS69] Ivan P Fellegi and Alan B Sunter. A theory for record linkage. *Journal of the American Statistical Association*, 64(328):1183–1210, 1969.
- [GAL⁺06] David Guthrie, Ben Allison, Wei Liu, Louise Guthrie, and Yorick Wilks. A closer look at skip-gram modelling. In *LREC*, volume 6, pages 1222–1225, 2006.
- [GC18] Mihajlo Grbovic and Haibin Cheng. Real-time personalization using embeddings for search ranking at airbnb. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pages 311–320, 2018.
- [Gil] L Gill. Methods for automatic record matching and linking and their use in national statistics. national statistics methodology series 25; 2001.
- [GM12] Lise Getoor and Ashwin Machanavajjhala. Entity resolution: theory, practice & open challenges. *Proceedings of the VLDB Endowment*, 5(12):2018–2019, 2012.

- [Gol17] Yoav Goldberg. Neural network methods for natural language processing. *Synthesis Lectures on Human Language Technologies*, 10(1):1–309, 2017.
- [HAR19] Kexin Huang, Jaan Altosaar, and Rajesh Ranganath. Clinicalbert: Modeling clinical notes and predicting hospital readmission. *arXiv preprint arXiv:1904.05342*, 2019.
- [HL03] Torsten Hothorn and Berthold Lausen. Double-bagging: combining classifiers by bootstrap aggregation. *Pattern Recognition*, 36(6):1303–1309, 2003.
- [HS95] Mauricio A Hernández and Salvatore J Stolfo. The merge/purge problem for large databases. *ACM Sigmod Record*, 24(2):127–138, 1995.
- [HS97] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.
- [HSW07] Thomas N Herzog, Fritz J Scheuren, and William E Winkler. *Data quality and record linkage techniques*. Springer Science & Business Media, 2007.
- [HVD15] Geoffrey Hinton, Oriol Vinyals, and Jeff Dean. Distilling the knowledge in a neural network. *arXiv preprint arXiv:1503.02531*, 2015.
- [HZL⁺20] Zhicheng Huang, Zhaoyang Zeng, Bei Liu, Dongmei Fu, and Jianlong Fu. Pixel-bert: Aligning image pixels with text by deep multi-modal transformers. *arXiv preprint arXiv:2004.00849*, 2020.
- [HZRS16] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.
- [HZY⁺19] Ji He, Lina Zhao, Hongwei Yang, Mengmeng Zhang, and Wei Li. Hsi-bert: Hyperspectral image classification using the bidirectional encoder representation from transformers. *IEEE Transactions on Geoscience and Remote Sensing*, 58(1):165–178, 2019.
- [JBS17] Yacine Jernite, Samuel R Bowman, and David Sontag. Discourse-based objectives for fast unsupervised sentence representation learning. *arXiv preprint arXiv:1705.00557*, 2017.
- [JGB⁺16] Armand Joulin, Edouard Grave, Piotr Bojanowski, Matthijs Douze, H erve J egou, and Tomas Mikolov. Fasttext. zip: Compressing text classification models. *arXiv preprint arXiv:1612.03651*, 2016.
- [JGBM16] Armand Joulin, Edouard Grave, Piotr Bojanowski, and Tomas Mikolov. Bag of tricks for efficient text classification. *arXiv preprint arXiv:1607.01759*, 2016.
- [JMN⁺16] Kurt Jacobson, Vidhya Murali, Edward Newett, Brian Whitman, and Romain Yon. Music personalization at spotify. In *Proceedings of the 10th ACM Conference on Recommender Systems*, pages 373–373, 2016.

- [KDSG⁺16] Pradap Konda, Sanjib Das, Paul Suganthan GC, AnHai Doan, Adel Ardalan, Jeffrey R Ballard, Han Li, Fatemah Panahi, Haojun Zhang, Jeff Naughton, et al. Magellan: Toward building entity matching management systems. *Proceedings of the VLDB Endowment*, 9(12):1197–1208, 2016.
- [KMV⁺19] Nitish Shirish Keskar, Bryan McCann, Lav R Varshney, Caiming Xiong, and Richard Socher. Ctrl: A conditional transformer language model for controllable generation. *arXiv preprint arXiv:1909.05858*, 2019.
- [KR10] Hanna Köpcke and Erhard Rahm. Frameworks for entity matching: A comparison. *Data & Knowledge Engineering*, 69(2):197–210, 2010.
- [KR18] Taku Kudo and John Richardson. Sentencepiece: A simple and language independent subword tokenizer and detokenizer for neural text processing. *arXiv preprint arXiv:1808.06226*, 2018.
- [LC13] Rémi Lebret and Ronan Collobert. Word emdeddings through hellinger pca. *arXiv preprint arXiv:1312.5542*, 2013.
- [LFL98] Thomas K Landauer, Peter W Foltz, and Darrell Laham. An introduction to latent semantic analysis. *Discourse processes*, 25(2-3):259–284, 1998.
- [Liu19] Yang Liu. Fine-tune bert for extractive summarization. *arXiv preprint arXiv:1903.10318*, 2019.
- [LJPD⁺13] Simon Lacoste-Julien, Konstantina Palla, Alex Davies, Gjergji Kasneci, Thore Graepel, and Zoubin Ghahramani. Sigma: Simple greedy matching for aligning large knowledge bases. In *Proceedings of the 19th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 572–580, 2013.
- [LL18] Lajanugen Logeswaran and Honglak Lee. An efficient framework for learning sentence representations. *arXiv preprint arXiv:1803.02893*, 2018.
- [LLG14] Lingli Li, Jianzhong Li, and Hong Gao. Rule-based method for entity resolution. *IEEE Transactions on Knowledge and Data Engineering*, 27(1):250–263, 2014.
- [LOG⁺19] Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. Roberta: A robustly optimized bert pretraining approach. *arXiv preprint arXiv:1907.11692*, 2019.
- [LPM15] Minh-Thang Luong, Hieu Pham, and Christopher D Manning. Effective approaches to attention-based neural machine translation. *arXiv preprint arXiv:1508.04025*, 2015.
- [LSP⁺18] Peter J Liu, Mohammad Saleh, Etienne Pot, Ben Goodrich, Ryan Sepassi, Lukasz Kaiser, and Noam Shazeer. Generating wikipedia by summarizing long sequences. *arXiv preprint arXiv:1801.10198*, 2018.

- [LW⁺02] Andy Liaw, Matthew Wiener, et al. Classification and regression by randomforest. *R news*, 2(3):18–22, 2002.
- [LW18] Matthias Leimeister and Benjamin J Wilson. Skip-gram word embeddings in hyperbolic space. *arXiv preprint arXiv:1809.01498*, 2018.
- [LXT⁺15] Yitan Li, Linli Xu, Fei Tian, Liang Jiang, Xiaowei Zhong, and Enhong Chen. Word embedding revisited: A new representation learning and explicit matrix factorization perspective. In *Twenty-Fourth International Joint Conference on Artificial Intelligence*, 2015.
- [LYK⁺20] Jinhyuk Lee, Wonjin Yoon, Sungdong Kim, Donghyeon Kim, Sunkyu Kim, Chan Ho So, and Jaewoo Kang. Biobert: a pre-trained biomedical language representation model for biomedical text mining. *Bioinformatics*, 36(4):1234–1240, 2020.
- [LZZ⁺20] Weijie Liu, Peng Zhou, Zhe Zhao, Zhiruo Wang, Qi Ju, Haotang Deng, and Ping Wang. K-bert: Enabling language representation with knowledge graph. In *AAAI*, pages 2901–2908, 2020.
- [MB12] Dutch T Meyer and William J Bolosky. A study of practical deduplication. *ACM Transactions on Storage (ToS)*, 7(4):1–20, 2012.
- [McC16] Chris McCormick. Word2vec tutorial-the skip-gram model, 2016.
- [MCCD13] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781*, 2013.
- [MLR⁺18] Sidharth Mudgal, Han Li, Theodoros Rekatsinas, AnHai Doan, Youngchoon Park, Ganesh Krishnan, Rohit Deep, Esteban Arcaute, and Vijay Raghavendra. Deep learning for entity matching: A design space exploration. In *Proceedings of the 2018 International Conference on Management of Data*, pages 19–34, 2018.
- [MS04] Andrew McCallum and Charles Sutton. Piecewise training with parameter independence diagrams: Comparing globally-and locally-trained linear-chain crfs. In *NIPS 2004 Workshop on Learning with Structured Outputs*, 2004.
- [MSC⁺13] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S Corrado, and Jeff Dean. Distributed representations of words and phrases and their compositionality. In *Advances in neural information processing systems*, pages 3111–3119, 2013.
- [MT13] Yongtao Ma and Thanh Tran. Typimatch: type-specific unsupervised learning of keys and key values for heterogeneous web data integration. In *Proceedings of the sixth ACM international conference on Web search and data mining*, pages 325–334, 2013.
- [NMMMBLP07] Jordi Nin, Victor Muntés-Mulero, Norbert Martinez-Bazan, and Josep-L Larriba-Pey. On the use of semantic blocking techniques for data cleansing and integration. In *11th International Database Engineering and Applications Symposium (IDEAS 2007)*, pages 190–198. IEEE, 2007.

- [PIN⁺12] George Papadakis, Ekaterini Ioannou, Claudia Niederée, Themis Palpanas, and Wolfgang Nejdl. Beyond 100 million entities: large-scale blocking-based resolution for heterogeneous data. In *Proceedings of the fifth ACM international conference on Web search and data mining*, pages 53–62, 2012.
- [PINF11] George Papadakis, Ekaterini Ioannou, Claudia Niederée, and Peter Fankhauser. Efficient entity resolution for large heterogeneous information spaces. In *Proceedings of the fourth ACM international conference on Web search and data mining*, pages 535–544, 2011.
- [PIP⁺12] George Papadakis, Ekaterini Ioannou, Themis Palpanas, Claudia Niederée, and Wolfgang Nejdl. A blocking framework for entity resolution in highly heterogeneous information spaces. *IEEE Transactions on Knowledge and Data Engineering*, 25(12):2665–2682, 2012.
- [PIP20] George Papadakis, Ekaterini Ioannou, and Themis Palpanas. Entity resolution: Past, present and yet-to-come. In *EDBT*, pages 647–650, 2020.
- [PNI⁺18] Matthew E Peters, Mark Neumann, Mohit Iyyer, Matt Gardner, Christopher Clark, Kenton Lee, and Luke Zettlemoyer. Deep contextualized word representations. *arXiv preprint arXiv:1802.05365*, 2018.
- [PPPK16] George Papadakis, George Papastefanatos, Themis Palpanas, and Manolis Koubarakis. Scaling entity resolution to large, heterogeneous data with enhanced meta-blocking. In *EDBT*, pages 221–232, 2016.
- [PR18] Sayan Putatunda and Kiran Rama. A comparative analysis of hyperopt as against other approaches for hyper-parameter optimization of xgboost. In *Proceedings of the 2018 International Conference on Signal Processing and Machine Learning*, pages 6–10, 2018.
- [PSM14] Jeffrey Pennington, Richard Socher, and Christopher D Manning. Glove: Global vectors for word representation. In *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)*, pages 1532–1543, 2014.
- [PTDU16] Ankur P Parikh, Oscar Täckström, Dipanjan Das, and Jakob Uszkoreit. A decomposable attention model for natural language inference. *arXiv preprint arXiv:1606.01933*, 2016.
- [QXLL19] Yifan Qiao, Chenyan Xiong, Zhenghao Liu, and Zhiyuan Liu. Understanding the behaviors of bert in ranking. *arXiv preprint arXiv:1904.07531*, 2019.
- [RD00] Erhard Rahm and Hong Hai Do. Data cleaning: Problems and current approaches. *IEEE Data Eng. Bull.*, 23(4):3–13, 2000.
- [RMD13] Delip Rao, Paul McNamee, and Mark Dredze. *Entity Linking: Finding Extracted Entities in a Knowledge Base*, pages 93–115. Springer Berlin Heidelberg, Berlin, Heidelberg, 2013. doi:10.1007/978-3-642-28569-1_5.

- [RNSS18] Alec Radford, Karthik Narasimhan, Tim Salimans, and Ilya Sutskever. Improving language understanding by generative pre-training, 2018.
- [RU11] Anand Rajaraman and Jeffrey David Ullman. *Mining of massive datasets*. Cambridge University Press, 2011.
- [RWC⁺19] Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, and Ilya Sutskever. Language models are unsupervised multitask learners. *OpenAI Blog*, 1(8):9, 2019.
- [SAS11] Fabian M Suchanek, Serge Abiteboul, and Pierre Senellart. Paris: Probabilistic alignment of relations, instances, and schema. *arXiv preprint arXiv:1111.7164*, 2011.
- [SB02] Sunita Sarawagi and Anuradha Bhamidipaty. Interactive deduplication using active learning. In *Proceedings of the eighth ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 269–278, 2002.
- [SBJ16] Giovanni Simonini, Sonia Bergamaschi, and HV Jagadish. Blast: a loosely schema-aware meta-blocking approach for entity resolution. *Proceedings of the VLDB Endowment*, 9(12):1173–1184, 2016.
- [SDCW19] Victor Sanh, Lysandre Debut, Julien Chaumond, and Thomas Wolf. Distilbert, a distilled version of bert: smaller, faster, cheaper and lighter. *arXiv preprint arXiv:1910.01108*, 2019.
- [SGM19] Emma Strubell, Ananya Ganesh, and Andrew McCallum. Energy and policy considerations for deep learning in nlp. *arXiv preprint arXiv:1906.02243*, 2019.
- [SIFL16] Imran Sheikh, Irina Illina, Dominique Fohr, and Georges Linarès. Learning word importance with the neural bag-of-words model. In *Proceedings of the 1st Workshop on Representation Learning for NLP*, pages 222–229, Berlin, Germany, August 2016. Association for Computational Linguistics. URL: <https://www.aclweb.org/anthology/W16-1626>, doi:10.18653/v1/W16-1626.
- [SME⁺17] Rohit Singh, Venkata Vamsikrishna Meduri, Ahmed Elmagarmid, Samuel Madden, Paolo Papotti, Jorge-Arnulfo Quiané-Ruiz, Armando Solar-Lezama, and Nan Tang. Synthesizing entity matching rules by examples. *Proceedings of the VLDB Endowment*, 11(2):189–202, 2017.
- [SMV⁺19] Chen Sun, Austin Myers, Carl Vondrick, Kevin Murphy, and Cordelia Schmid. Videobert: A joint model for video and language representation learning. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 7464–7473, 2019.
- [SPCT20] Rhea Sukthanker, Soujanya Poria, Erik Cambria, and Ramkumar Thirunavukarasu. Anaphora and coreference resolution: A review. *Information Fusion*, 59:139–162, 2020.

- [SQXH19] Chi Sun, Xipeng Qiu, Yige Xu, and Xuanjing Huang. How to fine-tune bert for text classification? In *China National Conference on Chinese Computational Linguistics*, pages 194–206. Springer, 2019.
- [SVL14] Ilya Sutskever, Oriol Vinyals, and Quoc V Le. Sequence to sequence learning with neural networks. In *Advances in neural information processing systems*, pages 3104–3112, 2014.
- [Tal11] John R Talburt. *Entity resolution and information quality*. Elsevier, 2011.
- [Tay53] Wilson L Taylor. “cloze procedure”: A new tool for measuring readability. *Journalism quarterly*, 30(4):415–433, 1953.
- [TKF11] Matthias Templ, Alexander Kowarik, and Peter Filzmoser. Iterative stepwise regression imputation using standard and robust methods. *Computational Statistics & Data Analysis*, 55(10):2793–2806, 2011.
- [UCG⁺16] Benigno Uria, Marc-Alexandre Côté, Karol Gregor, Iain Murray, and Hugo Larochelle. Neural autoregressive distribution estimation. *The Journal of Machine Learning Research*, 17(1):7184–7220, 2016.
- [VSP⁺17] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Advances in neural information processing systems*, pages 5998–6008, 2017.
- [WHZ⁺18] Jizhe Wang, Pipei Huang, Huan Zhao, Zhibo Zhang, Binqiang Zhao, and Dik Lun Lee. Billion-scale commodity embedding for e-commerce recommendation in alibaba. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pages 839–848, 2018.
- [WJL⁺20] Yue Wang, Shafiq Joty, Michael R Lyu, Irwin King, Caiming Xiong, and Steven CH Hoi. Vd-bert: A unified vision and dialog transformer with bert. *arXiv preprint arXiv:2004.13278*, 2020.
- [WKFF12] Jiannan Wang, Tim Kraska, Michael J Franklin, and Jianhua Feng. Crowder: Crowdsourcing entity resolution. *arXiv preprint arXiv:1208.1927*, 2012.
- [WLYF11] Jiannan Wang, Guoliang Li, Jeffrey Xu Yu, and Jianhua Feng. Entity matching: How similar is similar. *Proceedings of the VLDB Endowment*, 4(10):622–633, 2011.
- [Wol02] David H Wolpert. The supervised learning no-free-lunch theorems. In *Soft computing and industry*, pages 25–42. Springer, 2002.
- [YDY⁺19] Zhilin Yang, Zihang Dai, Yiming Yang, Jaime Carbonell, Russ R Salakhutdinov, and Quoc V Le. Xlnet: Generalized autoregressive pre-training for language understanding. In *Advances in neural information processing systems*, pages 5753–5763, 2019.

- [YLKG07] Su Yan, Dongwon Lee, Min-Yen Kan, and Lee C Giles. Adaptive sorted neighborhood methods for efficient record linkage. In *Proceedings of the 7th ACM/IEEE-CS joint conference on Digital libraries*, pages 185–194, 2007.
- [YML19] Liang Yao, Chengsheng Mao, and Yuan Luo. Kg-bert: Bert for knowledge graph completion. *arXiv preprint arXiv:1909.03193*, 2019.
- [YXCJJ⁺17] Liu Yi, Diao Xing-Chun, Cao Jian-Jun, Zhou Xing, and Shang Yu-Ling. A method for entity resolution in high dimensional data using ensemble classifiers. *Mathematical Problems in Engineering*, 2017, 2017.
- [ZWS⁺20] Wei Zhang, Hao Wei, Bunyamin Sisman, Xin Luna Dong, Christos Faloutsos, and David Page. Autoblock: A hands-off blocking framework for entity matching. In *Proceedings of the 13th International Conference on Web Search and Data Mining*, pages 744–752, 2020.
- [ZWYL14] Wayne Xin Zhao, Yuexin Wu, Hongfei Yan, and Xiaoming Li. Group based self training for e-commerce product record linkage. In *Proceedings of COLING 2014, the 25th International Conference on Computational Linguistics: Technical Papers*, pages 1311–1321, 2014.
- [ZZZ19] Junru Zhou, Zhuosheng Zhang, and Hai Zhao. Limit-bert: Linguistic informed multi-task bert. *arXiv preprint arXiv:1910.14296*, 2019.

Appendices

listings color

A. Prototypical Implementation of a Transformer Model for Entity Resolution

Importing libraries.

```
import numpy as np
import pandas as pd
import torch
import transformers as ppb # pytorch transformers
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import train_test_split
```

Loading pre-trained model/tokenizer.

```
model_class, tokenizer_class, pretrained_weights = (ppb.DistilBertModel, ↵
    ppb.DistilBertTokenizer, distilbert-base-uncased)

tokenizer = tokenizer_class.from_pretrained(pretrained_weights)
model = model_class.from_pretrained(pretrained_weights)
```

Tokenizing a single column.

```
tokenized_AC1 = batch1[title].apply((lambda x: tokenizer.encode(x, ↵
    add_special_tokens=True)))
```

Padding a single column.

```
max_len = 0
for i in tokenized_AC1.values:
    if len(i) > max_len:
        max_len = len(i)

padded_AC1 = np.array([i + [0]*(max_len-len(i)) for i in tokenized_AC1.values])
```

Masking a single column.

```
attention_mask_AC1 = np.where(padded_AC1 != 0, 1, 0)
```

Learning word embeddings for a single column by transformer.

```
input_ids = torch.tensor(padded_AC1)
attention_mask = torch.tensor(attention_mask_AC1)

with torch.no_grad():
    LHS1 = model(input_ids, attention_mask=attention_mask) #last hidden state
```

Concatenating embedded columns.

```
amazon_embd = pd.concat([df1[id], pd.DataFrame(LHS1[0], columns=["lhs1"]), ←
    pd.DataFrame(LHS2[0], columns=["lhs2"]), pd.DataFrame(LHS3[0], ←
    columns=["lhs3"]),price_scaled], axis=1)
```

Creating dictionaries to make training and test set.

```
id_dict=dict()
for index,row in amazon_embd.iterrows():
    id_dict[row["id"]]=index
print(id_dict)
```

Creating training and test set with respect to class imbalance.

```
train_set=set()
test_set=set()
import random
for item in id_df1.keys():
    for item2 in id_df1[item]:
        if random.uniform(0, 1)<0.66:
            train_set.add((item,item2,"match"))
        else:
            test_set.add((item,item2,"match"))
print(test_set)
print(train_set)
number_test_instances=len(test_set)
number_train_instances=len(train_set)
```

```
train_set=set()
while len(test_set)<2*number_test_instances:
    #Randomly pick an amazon key
    #Randomly pick a Google key
    #Check that they are not in ground truth
    rand_amazon = random.choice(list(akeys_pos.keys()))
    rand_google = random.choice(list(id_dict2.keys()))
    if not( rand_amazon in id_df1 and rand_google in id_df1[rand_amazon]):
        test_set.add((rand_amazon,rand_google, "non-match"))

while len(train_set)<2*number_train_instances:
    #Randomly pick an amazon key
    #Randomly pick a Google key
```

```

#Check that they are not in ground truth
rand_amazon = random.choice(list(akeys_pos.keys()))
rand_google = random.choice(list(id_dict2.keys()))
if not( rand_amazon in id_df1 and rand_google in id_df1[rand_amazon]) and ↵
    not (rand_amazon,rand_google, "non-match") in test_set:
    train_set.add((rand_amazon,rand_google, "non-match"))

```

Deploying a classifier (XGBoost).

```

! pip install xgboost
from xgboost import XGBClassifier
model = XGBClassifier()
xtrain = np.array(train_items)
ytrain = np.array(train_y)
xtest = np.array(test_items)
ytest = np.array(test_y)
eval_set = [(xtrain, ytrain), (xtest, ytest)]
model.fit(xtrain, ytrain, eval_metric=["error", "logloss"], ↵
    eval_set=eval_set, verbose=True)
# make predictions for test data
ypred = model.predict(xtest)
predictions = [round(value) for value in ypred]
# evaluate predictions
accuracy = accuracy_score(ytest, predictions)
print("Accuracy: %.2f%%" % (accuracy * 100.0))
print("F1 Score = {}".format(f1_score(ytest, ypred)))

```

B. Solving Transformers' Token Limitation Problem

Modifying columns with more than 512 tokens.

```
import textwrap
import math
AC2=[]
tokenized_AC2=[]
#Figure out max length (for padding...)
max_length=0
for index, row in batch1.iterrows():
    if True:
        string_to_encode=row[2]
        #if isinstance(string_to_encode, float) and math.isnan(string_to_encode):
        # string_to_encode="Null"
        encoded=textwrap.wrap(string_to_encode, 511)
        encoded_item_size=0
        temp_encoded=[]
        for item in encoded:
            input_ids=[tokenizer.encode(item, add_special_tokens=True)] # Add ↵
            # special tokens takes care of adding [CLS], [SEP], <s>... tokens ↵
            # in the right way for each model.
            encoded_item_size+=len(input_ids[0])
            for i in input_ids[0]:
                temp_encoded.append(i)
        if max_length<encoded_item_size:
            max_length=encoded_item_size
        tokenized_AC2.append(temp_encoded)
print(tokenized_AC2)
```

Padding the modified column.

```
for i in range(len(tokenized_AC2)):
    tokenized_AC2[i].extend(list([0]*(max_length-len(tokenized_AC2[i]))))

padded_AC2 = np.array([i for i in tokenized_AC2])
```

C. Loading Various Transformers with Hugging Face Library

BERT.

```
model_class, tokenizer_class, pretrained_weights = (ppb.BertModel, ↵  
    ppb.BertTokenizer, bert-base-uncased)  
  
tokenizer = tokenizer_class.from_pretrained(pretrained_weights)  
model = model_class.from_pretrained(pretrained_weights)
```

XLNET.

```
model_class, tokenizer_class, pretrained_weights = (ppb.XLNetModel, ↵  
    ppb.XLNetTokenizer, xlnet-base-cased)  
  
tokenizer = tokenizer_class.from_pretrained(pretrained_weights)  
model = model_class.from_pretrained(pretrained_weights)
```

GPT2.

```
model_class, tokenizer_class, pretrained_weights = (ppb.GPT2Model, ↵  
    ppb.GPT2Tokenizer, gpt2)  
  
tokenizer = tokenizer_class.from_pretrained(pretrained_weights)  
model = model_class.from_pretrained(pretrained_weights)
```