Daniel Nilsen
Aleksander Simmersholm

Leveraging Natural Language Processing in Data Synthesis for use in Entity Matching

Master's thesis in Master of Science in Informatics
Supervisor: Jon Atle Gulla
Co-supervisor: Nils Barlaug
July 2022

Master's thesis

NTNU
Norwegian University of Science and Technology
Faculty of Information Technology and Electrical Engineering
Department of Computer Science

NTNU
Kunnskap for en bedre verden

Daniel Nilsen
Aleksander Simmersholm

Leveraging Natural Language Processing in Data Synthesis for use in Entity Matching

Master's thesis in Master of Science in Informatics
Supervisor: Jon Atle Gulla
Co-supervisor: Nils Barlaug
July 2022

Norwegian University of Science and Technology
Faculty of Information Technology and Electrical Engineering
Department of Computer Science

Daniel Nilsen
Aleksander Simmersholm

Leveraging Natural Language Processing in Data Synthesis for use in Entity Matching

Masters thesis in Informatics
Supervisor: Jon Atle Gulla
July 2022

Master's thesis

NTNU
Norwegian University of Science and Technology
Faculty of Information Technology and Electrical Engineering
Department of Computer Science

NTNU
Norwegian University of Science and Technology

Abstract

Entity Matching (EM) is a difficult task which not long ago had to be performed manually. Now, Artificial Intelligence methods have been created to automate this process. They are, however, dependent on good training data to achieve good results. Acquiring good labeled data can be hard, expensive or in some cases, even impossible. We test whether Artificial Intelligence methods can be used to generate artificial data which to be used to improve EM model performance. Using the GPT-2 language model, the CTGAN method and a data augmentation method of our own we generate artificial data which is used to supplement the training data of the EM models. We also test how the models fare when their training data is wholly replaced with artificial data. Our results show that performance of EM models can be improved when supplementing. When using the artificial data alone to train the EM models, the results did not improve. We argue that with more refined methods, the results in both cases can be further improved.

Sammendrag

Entity Matching (EM) er en vanskelig oppgave som tidligere måtte bli utført manuelt. Metoder som benytter Kunstig Intelligens har siden blitt utviklet for å automatisere denne prosessen, men for å prestere bra er de avhengig av god treningsdata. God merkede data kan være vanskelig og dyrt å anskaffe, og i noen tilfeller kan det være umulig. Vi utforsker om Kunstig Intelligens kan bli benyttet for å produsere kunstig data som kan forbedre prestasjonen til EM modeller. Ved å bruke GPT-2 språkmodellen, CTGAN metoden og en data augmenteringsmetode vi selv har utarbeidet, vi genererer data som blir brukt til å supplere treningsdataen til modellene. Vi utforsker også om den genererte dataen kan alene bli brukt til å trene EM modellene. Våre resultater viser at EM modellene kan prestere bedre når deres treningsdata er supplementert med kunstig data. EM modellene presterte derimot ikke like bra når de var kun trent på kunstig data. Vi foreslår at ved å raffinere metodene for datagenerering kan resultatene bli enda forbedret.

Acknowledgements

This master's thesis was submitted to the Norwegian University of Science and Technology (NTNU), Department of Computer Science (IDI) as part of the Master of Science in Informatics study program.

We would like to thank our supervisor, Prof. Jon Atle Gulla, and our co-supervisor, PhD Nils Barlaug, for all the assistance, feedback and guidance they have provided us during the writing of this thesis.

We would also like to extend our thanks to our friends and family for their continuous encouragement and support.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

In this section we present our motivation for the thesis before introducing three research questions. An overview of our approach, as well as a summary of our results is then given, before ending with the outline of the thesis.

1.1 Motivation

Ever since Frank Rosenblatt discovered the perceptron model in the 1950's, a shift in the field of artificial intelligence started to occur. More and more research has been directed into machine learning and neural networks. As time went on, new powerful techniques such as Boosting and Deep Learning started emerging and taking over the field of AI by storm. These techniques, while powerful, are computationally expensive and data-hungry [1].

Further research into deep nets and machine learning have yielded tremendous results, producing highly accurate and sophisticated models. However, the computational requirements, as well as the data needed for them has only been increasing. According to OpenAI [2], the computational complexity has been increasing in accordance to Moore's law up until around 2012, after which it started increasing exponentially [fig. 1.1]. While it is amazing to see such rapid and bountiful progress in the field, if it keeps increasing at this pace, the computational requirements might become too great, making the progress vain. Some even postulate that this might be a factor for a new AI winter coming [3].

Current trends in AI show that deep learning is extremely popular due to its great results. Besides deep learning, reinforcement learning seems to be on the rise as well [4]. While reinforcement learning does achieve impressive results with little to no data, it's hard, and sometimes even impossible to formulate a task in a way that makes reinforcement learning a viable strategy [5].

Entity matching is a problem that is extremely domain specific, which makes it very hard to solve. While research into using reinforcement learning for this problem is being conducted, interest seems to be more on the machine learning and deep

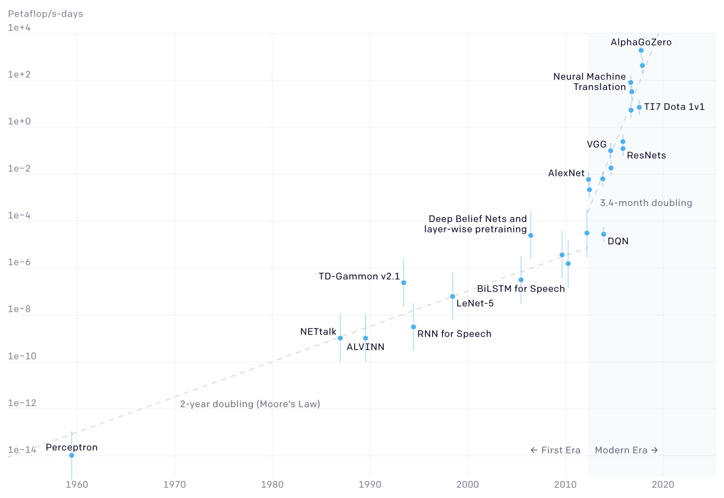


Figure 1.1: Two Distinct Eras of Compute Usage in Training AI Systems (figure by OpenAI)

learning side, which means that the computational complexity and data problems are still as relevant as ever. We believe that generating artificial data to use as supplement to real data for these models may help alleviate both of these problems.

Often times, the more data you train machine learning or deep learning models on, the better they will perform [6]. This implies that generating synthetic data to train these models on could increase their performance, without necessarily increasing the resources needed to use them. Besides alleviating the need for data and the complexity of the models, there are also other advantages of using artificial data. In his book, Nikolenko mentions that artificial data is often used in other fields such as computer vision [5]. Some benefits he mentions are that by using artificial data as supplementary to the real data, a fuller dataset can be created that is more suited for training the model. Synthetic data can also be used to resolve privacy or legal issues that make the use of real data prohibitively hard or even impossible.

Currently, there seems to be little research put into artificial data generation in the field of entity matching, which is why we would like to contribute to this field.

1.2 Research questions

Both Entity Matching (EM) and Artificial Data (AD) Generation are huge fields. Because of their size, trying to merge them without a focused approach is unrealistic. We therefore devise three research questions which this thesis focuses on:

Research question 1: What is the state of the art in the fields of EM and AD?

By researching state of the art methods, we can get a better understanding on how the technologies work. We can then get a selection of methods that we can use to explore the next research questions.

Research question 2: How does supplementing real data with AD affect the performance of EM models?

Annotated data is very valuable for data-hungry methods, but it can also be very expensive. Generating annotated data is therefore an interesting prospect for reducing both the amount of data needed for good performance of models and the cost of the data collection.

Research question 3: How does training EM models solely on AD affect their performance?

Some applications of EM might require the use of sensitive data, which is both hard to acquire and hard to use while upholding GDPR rules. It is therefore interesting to see whether good results can be achieved by completely substituting real data for artificial data.

1.3 Approach

To approach the research questions, we first investigate the state of the art entity matching systems and methods for data generation. We then use these generation methods as well as a data augmentation method of our own to generate both matching and non-matching entries using the datasets at our disposal. These artificial data are then combined together with the real data to generate new supplemented datasets, which are then tested on state of the art EM systems. By testing these new supplemented datasets against the real datasets used we gain insight into how the artificial data affects the results of the EM systems.

We also attempt to simulate small quantities of data by randomly selecting 10% of the data available to us and using it to generate artificial data. By doing so, we can gain even more insight into how the data affects the systems when tested against both the 10% real datasets and the full real datasets.

Finally, the generated data is also tested on these same EM systems without being combined with the real data. This is done to see whether comparable results can be achieved when replacing the real data entirely and using purely the artificial data.

1.4 Results

Our experiments showed that when supplementing real data with artificial data, performance of the Entity Matching systems can indeed be improved. However, performance of the systems suffered when replacing the real data with artificial data, and no improvements were achieved. We believe that with more refined methods, better performance can be achieved both when supplementing and replacing real data with artificial data.

1.5 Thesis outline

The structure of the thesis is as follows: Chapter 2 introduces the necessary background theory this thesis uses and builds upon. Chapter 3 provides an overview of the related work in the field of artificial data generation. Then chapter 4 showcases the datasets used in this project. Chapter 5 presents our proposed methods of data generation used in the project. Subsequently, chapter 6 describes the experiments that were ran on the Ditto and Magellan EM systems, as well as the results achieved by these experiments. In chapter 7, the results of the experiments are discussed, before a conclusion is presented in chapter 8. Finally, chapter 9 presents possible further work following this thesis.

Chapter 2

Background Theory

This section introduces relevant background theory for this project. The introductions of these topics are followed by some of the state of the art methods that are used in this project.

2.1 Artificial Data

Data is often the fuel for mathematical modeling of real-world functionality. Say we want to train a robot on folding paper. To train this robot, we would have to give it examples and further elaboration on these examples. We give it an image of an A4 paper, followed by the label "unfolded". Then we give it another image of the same paper, however, this time the paper is folded and labeled "folded". With enough examples, the robot would understand what an unfolded paper would look like, and vice versa with a folded paper.

It's a simple example but yields the understanding that every model needs training, and the training needs data. The most common practice today is to measure real-world examples manually, or by some other automatic means. The resulting dataset is then manually processed for formatting, labeling, and/or removal of inconsistencies. Data gathering and pre-processing is a vital step of any model training procedure. However, sometimes data cannot be gathered because of input limitations, or privacy concerns. An attractive option then becomes the generation of synthetic data.

We define synthetic data as "any data that is not obtained by direct measurements." More specifically, any data that is generated by an algorithm with the intent to train machine learning models. Synthetic data can help with mending specific needs or certain conditions in data sets that lack severe requirements. This can range from data fidelity to data quantity. In our scenario, this will be the matcher of an EM system.

2.2 Entity Matching

Entity matching (EM) is the problem of matching together instances of data that pertain to the same real-world entity. To illustrate, matching a list of company names to their respective Wikipedia articles could be considered an EM problem. There are however many factors that make this problem, like many others in the field of AI, extremely difficult.

2.2.1 History

Entity matching has been a topic of discussion and research for over 75 years now, with the first publication about the subject dating all the way back to 1946. In his paper [7], Halbert L. Dunn poetically describes the problem as

"EACH person in the world creates a Book of Life. This Book starts with birth and ends with death. Its pages are made up of the records of the principal events in life. Record linkage is the name given to the process of assembling the pages of this Book, into a volume.".

As evident in the quote, the subject of EM is called different names depending on the context and domain. Some of the names are:

Entity matching	Record linkage	Data matching
Object identity problem	List washing	Merge/purge processing
Entity resolution	Duplicate detection	De-duplication

Unfortunately, because of the breadth of terminology used in different domains, there has been little cross-reference between different research communities, which may have stunted research on the subject.

In his paper [7], Dunn outlines the importance of EM in the context of a persons information, as a singular record containing all information about some person would be valuable to many different actors, including the person in question. While that is true, the entity in question does not need to be a person. Any domain that has multiple different data sources which pertain to the same real-world entities can benefit from aggregating that data into one source.

In 1959, Howard Borden Newcombe published an article in the Science Magazine journal titled Automatic Linkage of Vital Records [8]. In this article, Newcombe presented their approach of automating the process of record linkage by using probabilistic methods to calculate likelihood of true matches between instances of data. While their methods were quite simple, they still achieved impressive results, which showed that the advancements in computing technology could indeed be leveraged in this field.

In 1969 Ivan Fellegi and Alan Sunter formalised a theory for record linkage based on the same probabilistic principles explored 10 years prior [9]. By making the assumption that the comparison attributes are conditionally independent from one another, they can be added together to form a compound probability for the match being true or not true, with a corresponding probability for the prediction to be false. The Fallegi-Sunter theory remains the mathematical foundation for many entity matching applications to this day [10].

Much research into using machine learning and AI models in entity matching started happening during the second AI boom in the 1990's. Most of the attention however was put on optimising the feature selection in applications that used the Fallegi-Sunter theory [11, 12]. While these methods did improve performance using Fallegi-Sunter, they did not explore whether other machine learning and AI methods would or could work better.

At the International Joint Conference on Neural Networks, in 2011, D. Randall Wilson showed that the Fallegi-Sunter theory is comparable to the naïve Bayes classifier [13]. In his findings, he showed that results could be drastically improved by using even a simple perceptron algorithm instead of the probabilistic approach. This meant that research into new methods of entity matching by using these more complex methods of machine learning are indeed beneficial, and worth pursuing.

In the current EM landscape, many methods are researched, from traditional classifiers that ”learn” what a matching set of instances is, to using language models to simulate ”understanding”. The current state of the art is discussed further in 2.2.4 Current State of the Art.

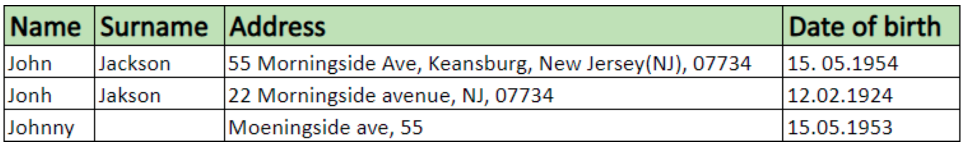
2.2.2 Challenges in Entity Matching

Some of the challenges in automating the EM process were noticed since the very beginning. In the 1959 paper [8], Newcombe mentions some of the challenges they faced, such as fuzzy and unreliable data, as well as differences in formatting. These challenges have yet to be fully solved, and have to be addressed even in today's EM applications. There are also new challenges that have made themselves apparent as the methods grew in complexity and the domain of EM grew to encompass all sorts of entities and data sources.

Fuzzy data

Fuzzy data is a term used to describe datasets that are prone to small inconsistencies or mistakes that make them difficult to use. These can range from small errors such as spelling mistakes, to outright missing data and empty fields.

The figure below [fig. 2.1] shows some examples of fuzzy data. While the data depicts the same entity, that being John Jackson, it is difficult for a computer to discern that because of the fields being inaccurate or missing.



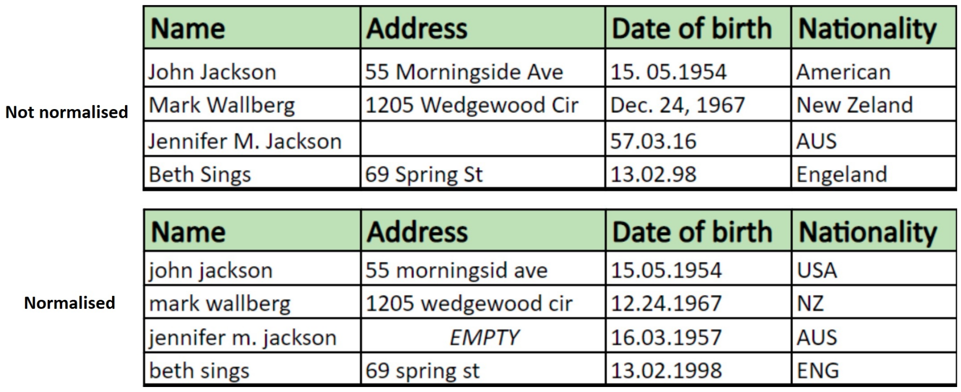
Name	Surname	Address	Date of birth
John	Jackson	55 Morningside Ave, Keansburg, New Jersey(NJ), 07734	15. 05.1954
Jonh	Jakson	22 Morningside avenue, NJ, 07734	12.02.1924
Johnny		Moeningside ave, 55	15.05.1953

Figure 2.1: Examples of fuzzy data

A common way to avoid this issue is to pre-process the data in some way. Some methods, including Fallegi-Sunter, use normalisation on the data before calculating the probabilities. This normalisation typically includes things like:

- Turning all text to lower-case
- Utilizing stemming or lemmatization techniques
- Replacing missing values with some token

Such measures tend to increase performance, but are susceptible to issues stemming from unknown edge cases. Sometimes, an unpredictable issue in the data might arise, which does not get covered by the normalisation techniques. Figure [fig. 2.2] shows data before and after normalisation.



Not normalised

Name	Address	Date of birth	Nationality
John Jackson	55 Morningside Ave	15. 05.1954	American
Mark Wallberg	1205 Wedgewood Cir	Dec. 24, 1967	New Zeland
Jennifer M. Jackson		57.03.16	AUS
Beth Sings	69 Spring St	13.02.98	Engeland

Normalised

Name	Address	Date of birth	Nationality
john jackson	55 morningsid ave	15.05.1954	USA
mark wallberg	1205 wedgewood cir	12.24.1967	NZ
jennifer m. jackson	*EMPTY*	16.03.1957	AUS
beth sings	69 spring st	13.02.1998	ENG

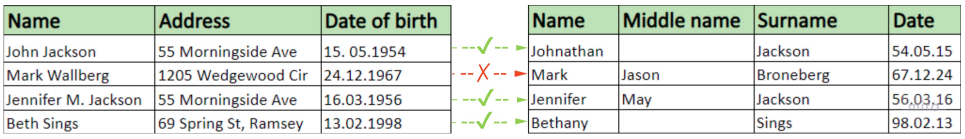
Figure 2.2: Data before and after normalisation

When using neural network based approaches, one could disregard the normalisation of data, and attempt to teach the network to recognise these types of errors. This might improve resilience and performance of the model when introduced to such perturbations, as well as edge cases not previously seen, however, it could also take longer to train the model to a point of acceptable performance.

Schema differences

A schema is a model used to organise data. These models are usually set up according to the primary use of the data, as well as the preference of the people creating the datasets. While there are standards for schemas, there are no set-in-stone rules to follow, which presents some problems when trying to match entities together.

The figure below [fig. 2.3] shows how data about the same entity might be put into differing schemas. The format of the data is also somewhat decided by the schema, meaning that differing formatting can cause issues. This is easily seen in the date fields, but can also refer to abbreviations of names, conglomerations of fields, etc.



Name	Address	Date of birth		Name	Middle name	Surname	Date
John Jackson	55 Morningside Ave	15. 05.1954	✓	Johnathan		Jackson	54.05.15
Mark Wallberg	1205 Wedgewood Cir	24.12.1967	✗	Mark	Jason	Broneberg	67.12.24
Jennifer M. Jackson	55 Morningside Ave	16.03.1956	✓	Jennifer	May	Jackson	56.03.16
Beth Sings	69 Spring St, Ramsey	13.02.1998	✓	Bethany		Sings	98.02.13

Figure 2.3: Some entities under different schemas

A different issue can also arise when the datasets are in different formats all together. An example of this could be trying to match a list of products with pictures of said products. As the datasets differ from each other this drastically, conventional methods of comparison will not work.

Similarity measures

When matching entities, the similarity measure is what determines whether or not two records are a match. This is usually done by somehow comparing the two records, but this is not always a straight forward process.

The Fellegi-Sunter theory outlines a general approach for probabilistic record comparison [9]. Their theorem is as such:

Let $L_0(\mu, \lambda, \Gamma)$ be the linkage rule defined by (15). Then L is a best linkage rule on Γ at the levels (μ, λ).

Where μ is the error rate for false-positives, λ is the error rate for false negatives, Γ is the set of all comparisons of the comparison variables between the two records. The comparison variables can be things like *name*, *date of birth*, *address*, etc. The comparing of the variables takes into account that the variables themselves might be misspelled, missing or otherwise wrong.

The accumulation of these probabilities will give some probability of the records being a match, taking into account the probabilities of false-positive and false-negative matches μ and λ respectively.

Another popular similarity measure used by many state of the art models is vector space modeling, or vector encoding. The records are encoded into vectors, with each vector component being a comparison variable. The figure below illustrates this with 2-dimensional vectors [fig. 2.4]

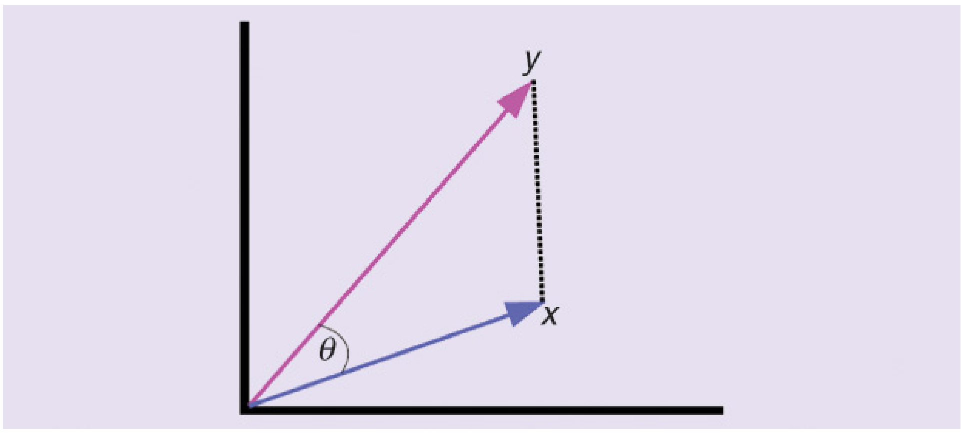


Figure 2.4: 2d vectors in a vector space (Image courtesy of IEEE Engineering in Medicine & Biology Society)

By converting the records into vectors that inhabit the same vector space, similarity can be calculated by calculating the dot product of the vectors [14]. Other methods of calculating similarity can be done as well, such as calculating the euclidean distance between the vectors.

While indeed effective, both approaches discussed above suffer from the problems discussed in 2.2.2 Fuzzy data and 2.2.2 Schema differences. This is because at their core, they compare the different fields of data together and see whether the information is the same in both. To mitigate this, most approaches usually rely on sophisticated data pre-processing, or intelligent ways of comparing the data integrated into the similarity measure.

Domain specificity

Since EM is a problem that involves real-world entities, it becomes extremely difficult to find a solution that works in every context. This is because the real world is non-discrete, and a real-world entity can be described in an infinite amount of ways. In other words, there is no limit to the amount of different types of data that can be used to describe the same real-world entity. This puts a limit on how much we can generalize until a tailor-made solution is needed.

The problems that are mentioned above are also compounded and made worse by this, because new methods of data pre-processing are needed for each type of data. The new data might also have schemas that the application has never encountered before. On top of this, the similarity measures that are in place might not fit the types of data used, meaning new measures are needed. Every domain has their own uses for EM, meaning that the domain also dictates what kind of results are needed from the application.

Dearth of labeled data

A problem that emerged together with the advancements in machine learning technology is the need for good data. A majority of the leading machine learning methods use a supervised learning approach [15], and most of the EM methods that use machine learning use these methods. The leading machine learning methods are notoriously data-hungry [16], which means huge amounts of data are needed to achieve good results.

The data used for these machine learning models needs to come from somewhere, and regardless of the method of data gathering, some human intervention is needed. Humans need to either to quality control the data to ensure that the labeling methods do what they're supposed to, or brute-force label it themselves, which is time-consuming and costly [15].

Personal data

As Dunn said, the ultimate goal of record matching was to put together the information of a person into a singular volume he called *"the book of life"* [7]. While entity matching has evolved beyond just information about people, the problem of matching personal data is still prevalent.

With this however, the problem of privacy and confidentiality needs to be considered. As outlined by Peter Christen, matching data between different organisations might lead to undesirable outcomes such as discrimination or differential treatment based on outside factors such as race, gender, socioeconomic conditions, etc.

"For example, the outcomes of analysing matched health and population databases can potentially lead to discrimination against certain groups of individuals, if it is discovered that these people have a higher risk of getting a certain serious or infectious disease. The discrimination could be in the form of higher life insurance premiums, or even that these individuals would find it much harder to gain employment due to their potentially increased risk of long-term illness."[17]

2.2.3 Entity Matching process

EM is traditionally depicted as a pipeline of steps that are needed before some result is reached. These steps are however not specified anywhere, meaning that many different approaches can be taken. In their study [18], Nils Barlaug and Jon-Atle Gulla compiled a general list of steps that most methods use, and visualised them as a pipeline model which is depicted in the figure below [fig. 2.5].

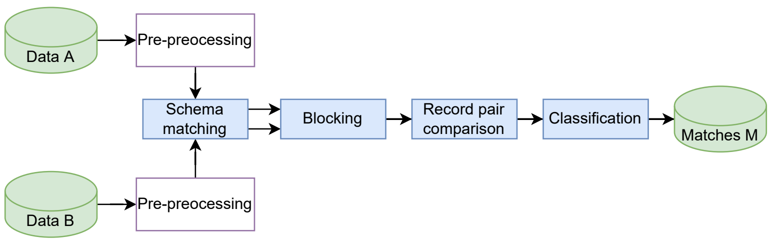


Figure 2.5: The traditional EM process, as laid out by Barlaug and Gulla

Pre-processing, Schema matching, Blocking, Record pair comparison and Classification are the steps common to EM methods. Some methods, however, skip some of these, as they achieve the effect of those steps through other means. Some methods also add other steps to increase performance. It is however useful to have a general idea of what these steps are, as that will help understand the general pipeline, as well as why some methods choose to veer away from the general method somewhat.

Pre-processing

The pre-processing step is, as the name implies, meant to pre-process the data. This step is usually done to fix or mark errors in the data, such as discussed in the Fuzzy data subsection, and is often a vital step in the EM process [19].

The methods of pre-processing vary across EM methods. These are usually picked based on the data that is to be matched. Some methods (also discussed in Fuzzy data):

- Normalization of attribute values
- Missing value handling
- Removal of unwanted characters

Schema matching

In this step, attributes from each dataset are matched together. Structural differences in the data, such as field concatenation or formatting, can cause issues when matching data together. This issue is discussed in more detail in the Schema differences subsection. Some actions that might be beneficial to this process:

- Splitting attributes into multiple attributes
- Changing formatting of fields into one consistent format
- Find correlating fields in the data through analysis.

Blocking

After the previous steps are completed, the matching process can technically begin. It is however unwise to just start matching records together, especially as the size of the datasets grows. This is because, at this point, any record in the first dataset can potentially be a match to any record in the second dataset, which means that the operation approaches quadratic complexity. The blocking step is meant to reduce the amount of potential matches by splitting the datasets into blocks, in which all obvious non-matches are disregarded [20].

Some traditional blocking techniques include:

- Standard blocking [21]
- Q-gram blocking [22]
- Sorted neighborhood blocking [23]

Record pair comparison

This step is essentially very similar to blocking. However, the difference is that blocking uses less computationally expensive methods to discard obvious non-matches. This allows for more sophisticated and computationally expensive methods to be used, as they will need to check less potential pairs. This step usually produces a set of similarity vectors with the attributes being the similarity measures used in the EM solution. Some similarity measures are discussed in the Similarity measures subsection.

Classification

Finally, the records are classified as matches, or non-matches. This is done based on the similarity measures produced by the previous step.

2.2.4 Current State of the Art

Magellan

Many EM solutions focus on one or more aspects of EM, such as blocking or matching. Magellan tries to encompass the entire EM pipeline, creating a full EM system [24].

As discussed in Domain specificity, a generic EM system will not work as well in every context, which is why Magellan provides tools and how-to guides for many EM scenarios. The Magellan approach is split up unto two stages, the development stage and the production stage. The figure below [fig. 2.6] shows the Magellan architecture.

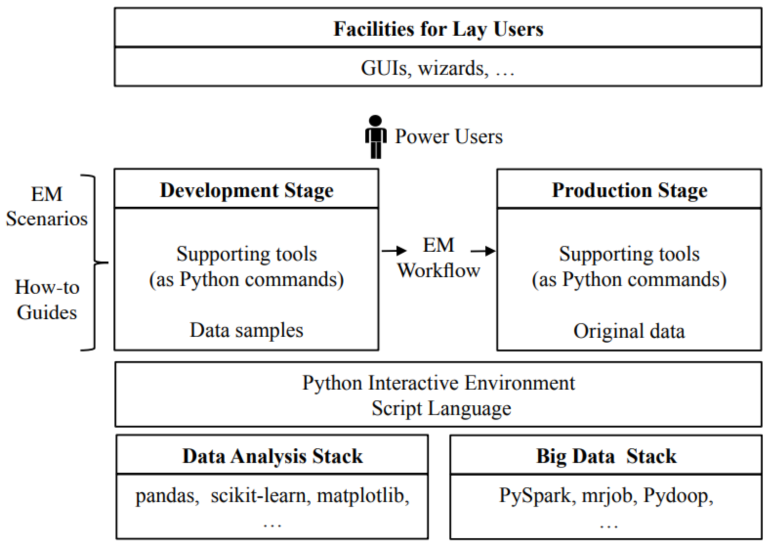


Figure 2.6: The Magellan architecture (Image courtesy of Konda et. al[24])

In the development stage, the user is to develop a good EM workflow for the specific EM task. A good workflow is one that yields high matching accuracy. The user is proposed a how-to step-by-step guide which will help the user with developing the workflow. The steps in the guide can be achieved by use of the supporting tools in Magellan. Magellan also supports creating custom python scripts, which can be used as supporting tools.

In the production stage, the user is provided a how-to step-by-step guide on how to implement the workflow on the entirety of the data. This is again done with the help of the supporting tools.

Ditto

Ditto is an EM solution based on pre-trained transformer-based language models [25]. By utilising such pre-trained language models, and fine-tuning them when training the model based on the available data, Ditto is able to encode contextual nuance and understanding into its entity encoding.

The figure below [fig. 2.7] illustrates how Ditto fine-tunes the language models by adding task-specific layers to the language model, initialising the network and training it on the labeled data available for the EM task until the network converges [25]. The resulting model keeps the complex language understanding obtained from training on huge language corpora, and also develops further understanding on what it means that a record matches another based on the data available.

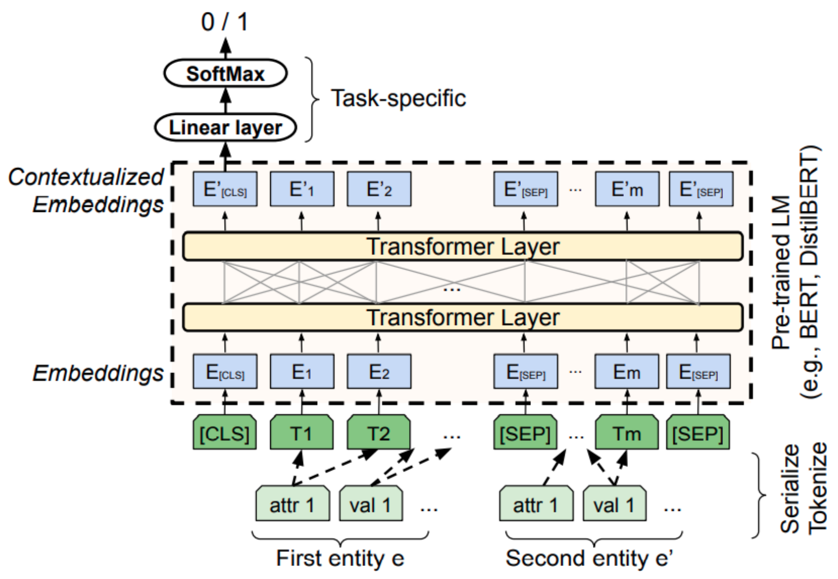


Figure 2.7: The modified architecture of the language model Ditto uses (Image courtesy of Li et al. [25])

The figure below [fig. 2.8] shows a typical EM architecture with Ditto as the matcher. The dotted box shows all the discreet parts of Ditto, including the optional optimizations that increase Ditto's performance labeled 1, 2 and 3.

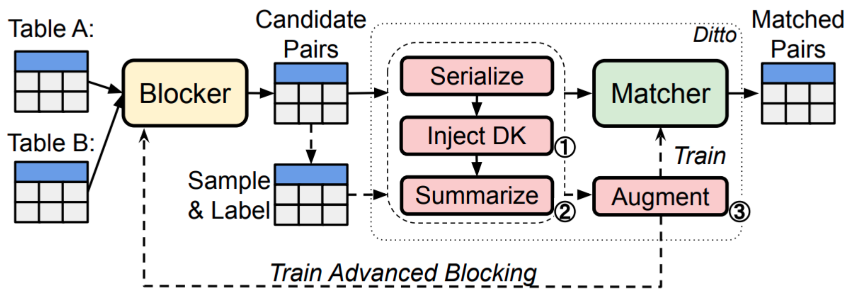


Figure 2.8: EM architecture with Ditto as the matcher (Image courtesy of Li et al. [25])

The serialisation part of Ditto is a necessary part that translates the entries to be matched into serialised tokens that are accepted by the language model. It does this like so [25]:

For each data entry (2.1) , we let the serialised entry be defined as (2.2) , where [COL] and [VAL] are special tokens for indicating the start of attribute names and values respectively.

$$e = \{(attr_i, val_i)\}_{1<i<k} \tag{2.1}$$

$$serialise(e) ::= [COL]attr_1[VAL]val_1...[COL]attr_k[VAL]val_k \tag{2.2}$$

After serialisation, the data entries can be sent into the matcher, and determined whether they are a true match or not.

The first optimisation, labeled 1 in [fig. 2.8], is injecting the model with domain knowledge. In some circumstances, EM applications can benefit greatly from domain knowledge. An example of this could be matching publications and paying extra close attention to the DOI numbers of these publications. Ditto allows for specific tags to be placed on the data during pre-processing, that signalise that the fields with the tags hold greater weight than those without.

The second optimisation, labeled 2 in [fig. 2.8], is summarising long entries. Sometimes, if the entry is too long, then the language model struggles with understanding what is of importance. The Ditto implementation described in [25], uses a TF-IDF-based summarisation technique which retains non-stopword tokens with the high TF-IDF scores. That way, the most informative tokens are fed into the language model.

The third optimisation, labeled 3 in [fig. 2.8], is augmentation of training data. Ditto uses augmentation of training data to make the system more robust against dirty data, as well as preventing the system against overfitting. They achieve this by artificcially creating new data based on the data they have, and sometimes introducing errors into the data, such as missing, misplaced or corrupted attributes.

Ditto looks to be very good at entity matching when not alot of training data is available. In the tests that Li et al. performed [25], it is clear that Ditto outperforms other EM models when supplied with little training data. Figure below [fig. 2.9] shows the F1 scores of Ditto compared to DeepMatcher with different data sets and different amounts of data supplied.

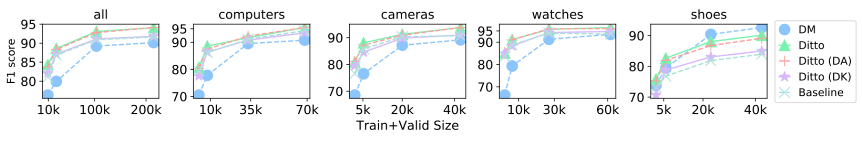


Figure 2.9: F1 scores on the WDC datasets of different versions of Ditto. DM: DeepMatcher (Figure courtesy of [25])

2.3 Language Models

A language model is a probability distribution over words or sequences of words [26]. This makes language models a part of the Statistical Natural Language Processing field. Simply put, a language model assigns probabilities to words or sequences of words. These probabilities denote the likelihood of appearing next in some sequence of words. Language models are usually trained on large text corpora.

2.3.1 History

The early history of language models, much like the early history of AI, was governed by rule-based systems. In 1948, Claude Shannon published his now famous paper ”A Mathematical Theory of Communication” [27], in which he demonstrated the use of n-grams in language modelling.

An n-gram is a collection of n words which is assigned a total probability of appearing. It is defined as the joint probabilities of each of the word in the collection, given that the words prior have appeared in that order:

$$P(\omega_1, \dots, \omega_m) = \prod_{i=1}^{m} P(\omega_i|\omega_1, \dots, \omega_{i-1}) \approx \prod_{i=2}^{m} P(\omega_{i-(n-1)}|\omega_1, \dots, \omega_{i-1}) \tag{2.3}$$

Research focused mainly on n-gram and rule-based language models, but with the rise in computational power available, new computational methods were introduced. In 1982, John Hopfield introduced the Recurrent Neural Network (RNN) [28] which processes the inputs of the network along a temporal sequence [fig. 2.10].

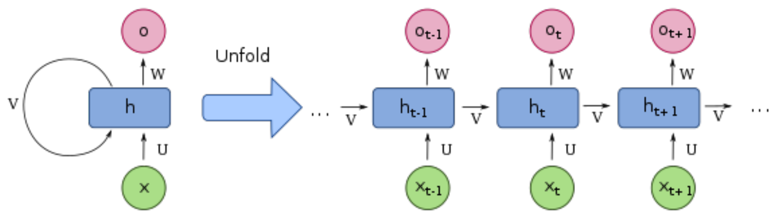


Figure 2.10: A basic Recurrent Neural Network network (figure courtesy of fdeloche)

This means that the order of appearance of the inputs becomes important for the output of the neural network. Since many languages rely on specific placement of words to convey semantic meaning, such a network works well as a language model.

Already in 1986 ideas on representing words as vectors started to arise [29, 30]. The reasons for why this came about are not immediately obvious. Many languages, such as the English language, have words that stem from the same word, but slight alteration gives them extra semantic meaning. An example of this might be the words ”text” and ”texting”, where one denotes an object while the other denotes an action. By switching to a vector representation, such semantic alterations can be embedded in a vector representation. The model can then be trained on less data, as it doesn’t need to train on all of the variations of the same word. Other advantageous properties can also be encoded into the vector representation, such as close proximity of words in vector space that are semantically similar, such as ”Cat” and ”Lion” being closer together than ”Cat” and ”Microwave”.

While effective, the RNN networks are prone to forgetting important contextual clues when working with longer time series data with arbitrarily long gaps between important data. In 1997 Hochreiter and Schmidhuber proposed the Long Short-Term Memory (LSTM) cells [31] as a solution to this problem.

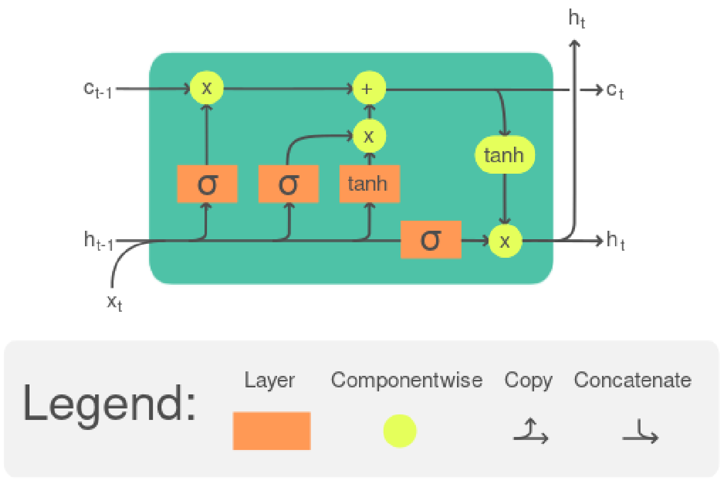


Figure 2.11: The Long Short-Term Memory (LSTM) cell can process data sequentially and keep its hidden state through time. (Figure courtesy of Guillaume Chevalier)

RNNs using the LSTM cells [fig. 2.11] can allow for the gradients of the network to flow freely unchanged, meaning that the network has a reference to important parts in the time sequence. This property also solves the vanishing gradient problem, which occurs when the gradients tend towards zero while back-propagating. LSTM does still suffer from the exploding gradient problem, which occurs when the gradients tend towards infinity.

While solving the forgetting problem, the LSTM approach was limited by the hardware of the time. A different approach to the forgetting problem was introduced in 2015 by Bahdanau et al., the Attention Model [32]. Attention-like mechanisms were introduced in the 1990s under names like multiplicative modules, sigma pi units, and hypernetworks [33].

In lay terms the model is based on the idea that humans don't usually remember the whole sequence of words before preforming a task, but rather find and focus on the important parts of the text. It achieves this by assigning weights to the input sequence, enhancing some parts of the text, while diminishing others. The Attention Model was introduced specifically in the context of machine translation, but the attention model has proven useful in other contexts since its introduction.

In 2017, researchers at Google proposed a new model that deals away with RNNs completely and is solely based on the attention mechanism. The new model was called the Transformer [34].

The Transformer processes the entire input at once, using its attention mechanism to provide context for any position in the sequence [fig. 2.12]. This allows for more parallelization than RNNs and therefore reduces training times. The usage of attention yields the self-sufficiency without the need for recurrence and convolutions.

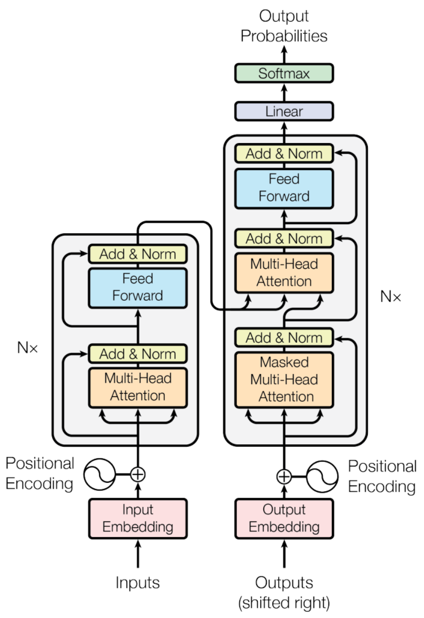


Figure 2.12: The Transformer - model architecture (Figure courtesy of Vaswani et al. [34])

2.3.2 GPT-2

While there are other state of the art Language Models, such as, BERT [35], ELMo [36], etc., we chose to use GPT-2 [37] for this project.

The Generative Pre-trained Transformer 2 (GPT-2), proposed by Alec Radford et al. in 2019 [38], is a transformer-based pre-trained language model. The original model has 1.5 billion parameters and is trained on a giant corpus of textual data gathered from the internet (40GB). However, due to concerns about malicious applications, the full model is not available to the public, and only a small version of the model has been released.

The model uses a Transformer-based architecture which is similar to its predecessor, GPT [39]. It is however slightly modified, with layer normalizations moved to the input of each sub-block and an additional layer normalization was added after the final self-attention block. A modified initialization which accounts for the accumulation on the residual path with model depth is used. The weights of residual layers are scaled at initialization by a factor of $1/\sqrt{N}$ where N is the number of residual layers. The vocabulary is expanded to 50,257. The context size is also increased from 512 to 1024 tokens and a larger batchsize of 512 is used [38].

The idea behind GPT-2 is to be a general language model that can be used for a multitude of Natural Language Processing tasks without needing large amounts of specialised data to be trained on. The GPT-2 model is therefore an example of an unsupervised learning model. The model is also an example of transfer learning, as the idea is to have this model be trained to do the general task, and then using the pre-trained model on some specific task.

2.4 Data distribution

In probability theory, probability distribution is a mathematical function used to describe the probabilities of possible outcomes of some inherently random situation occurring [40]. One such random situation is a coin toss. Given that there are only two possible outcomes of a coin toss (heads or tails), the probabilities of each outcome are $P(h) = P(t) = \frac{1}{2}$, with $P(h)$ being the probability for heads and $P(t)$ being the probability for tails.

In data science and computer science, this concept of probability distribution is applied to datasets, producing a data distribution. Data distribution is used to model the frequency at which each unique data point appears in the dataset. To use the coin toss example again, if we tossed a coin n amount of times and recorded each outcome, we would end up with a dataset containing these outcomes. We can could then derive a distribution of outcomes from this dataset, which would show the frequency with which heads or tails appeared in the dataset. The data distribution is not equivalent to the probability distribution of each outcome, as one might get unlucky and get heads 100 times in a row. Data distribution is expected to approach the probability distribution as the amount of data in the dataset grows. This is in line with the Central Limit Theorem, which states that as the amount of data increases, the set tends towards a normal distribution [41].

Many types of distributions can be derived based on what metrics are used. One of these types is the Univariate distribution. The univariate distribution is achieved by grouping the data entries together based on one attribute of the data [42]. Given a dataset which contains the attributes *Job Title*, *Salary* and *Company*, a univatiate distribution method could be used to observe the frequency of appearance of one of these attributes throughout the dataset. Some univariate distribution methods include binomial, geometric, negative binomial, and Poisson distributions [43]. The figure below shows an example of one of these distributions, namely the binomial distribution [fig. 2.13].

Another type of distribution is the Multivariate Distribution. As the name suggests, this distribution combines multiple attributes and creates a compound distribution out of them. In other words, it is a generalization of the univatiate distribution into higher dimentions [44]. This is typically done by assuming that the attributes are conditionally independent, which means that $P(a|b) = P(a) * P(b)$ is valid. Then, a matrix of compound probabilities for each of the attributes is computed, creating the multivariate distribution. The figure below shows an example of a multivariate distribution [fig. 2.14].

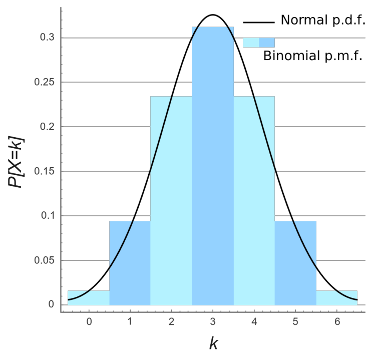


Figure 2.13: Example of a binomial distribution (Figure by Cflm001)

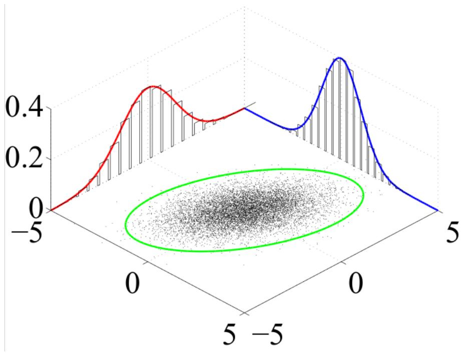


Figure 2.14: Example of a joint probability distribution with two variables (Figure by IkamusumeFan)

2.5 Generative Adversarial Networks

Generative Adversarial Networks (GAN) are architectures that use two neural networks pitted against each other to outperform the other model. When optimal performance is achieved, the result is a generative model which can generate realistic data samples.

The purpose of this architecture is most often to generate new, synthetic instances of varying types of data that can be estimated as real data. The use-case ranges widely in correspondence to tasks and desired data type. Some examples are image generation [45] [46] [47], video generation [48] [49] [50], and voice generation [51] [52].

2.5.1 History

An adversarial game is a concept in game theory that considers a problem for which we try to plan ahead in an environment where other players are planning against us. Take, for example, a game of chess. Both players have the goal of achieving checkmate against their opponent. We consider the board which scores us checkmate to have the maximum utility value. One player aims to maximize its score, while the other minimizes. Since the total gain of the two players is zero, we consider it a zero-sum game. The utility value is the opposite of each other and creates an adversarial situation. The solution then becomes the strategy of winning the game while also considering the optimality of the opponent. We often treat this problem as a search problem, finding the optimal action for the current situation with some intuition of how the opponent will act [53].

While adversarial machine learning is considered somewhat young [54], the idea of pitting two algorithms against each other can be seen in 1959, when Arthur Samuel devised his now-famous Samuel Checkers game [55]. Samuel's algorithms played by performing heuristic search methods from each current position, inspired by Claude Shannon's minimax procedure [56]. As such, it was possibly the first program to self-learn by estimating each player's chance of success at a given position.

In the domain of probability and statistical theory, a paradigm often used in machine learning is statistical classification modeling. Two main approaches are the generative and discriminative approaches. Some inconsistencies lie in the terminology, but we distinguish only the discriminatory and generative approaches for the sake of clearance.

The generative model aims to model the joint probability distribution $P(X, Y)$ on the known data points X and target labels Y. In contrast, the discriminative model aims to model the conditional probability $P(Y|X)$ between the unknown labels Y and the known data points X.

In a more immediate sense, the discriminative model aims to "discriminate" or rather classify the data points X by learning the boundaries between data classes to better predict the classification Y of X. On the other hand, the generative model attempts to model the distribution of classes Y by modeling the underlying distribution of the data points X. Because the model learns the probability distribution of the data points X, it can also utilize this statistical property to generate new data instances. Since the generative approach models the distribution of the data points, it tends to be more computationally expensive.

As the discriminative approach aims to efficiently predict the label of a given data point X with label Y, we train it by giving examples of inputs X and correct the model to make the predictions more accurate. The lack of correction means that the method of learning is the unsupervised learning approach. This method of learning is the supervised learning approach. On the other hand, the generative approach aims to replicate a summarization of the pattern from the given inputs X [57].

GANs were introduced by Ian Goodfellow and fellow researchers at the Department of Informatics Research at the University of Montreal in 2014. Since then, GANs have become a staple architectural model in data generation and are often considered state-of-the-art in some domains, if not the baseline against newer methods.

At the time of the article, deep learning models most prevalent for modeling high-dimensional data were the discriminative models, which utilized backpropagation and dropout algorithms on neural nets to achieve sufficient gradients. The problem, however, for the deep generative model was the difficulty of approximating many intractable probabilistic computations during maximum likelihood estimations and related strategies. As such, they initially could not leverage the same benefits as their deep learning counterpart. However, the GAN architecture uses two neural nets distinguished by their tasks to combat this and leverage the same benefits of the discriminative models [58].

2.5.2 GAN

Essentially, the GAN model consists of two neural nets. Because of its architecture, both neural nets can be trained using backpropagation and dropout algorithms in unison, while the generative model is also trained by forward propagation. This creates the added benefit of turning the generative part of the architecture into a supervised learning approach while the algorithm as a whole remains unsupervised.

The core idea of GANs can be interpreted as an adversarial game between two players. More specifically, one player constitutes the generator, with its opposing player being the discriminator. The generator attempts to generate data samples based on its ability to model the data distribution of the real dataset. The discriminator then receives a batch of data samples where it tries to distinguish between the data generated by the generator and the data from the real dataset. The goal of the generator is to capture the data distribution of the real dataset to a point where the discriminator can no longer effectively distinguish the artificial data from its real counterpart.

The generator's distribution $P_g(x)$ is trained on the training sample data x, with a prior distribution on input noise variables $P_z(z)$ where z is the random variable from the prior distribution, mapped to the data space $G(z;\theta_g)$. G is the generative neural network with the parameter θ_g. The goal for G is to train $P_g(x)$ to be as similar as $P_{data}(x)$ such that P_g = P_{data}. For the generator, the target is to find G such that:

$$G = \arg\min_{G} div(P_g, P_{data}) \tag{2.4}$$

To calculate the difference between the two distributions P_g and P_{data}, the discriminative model is used. The discriminator is a binary classifier $D(x;\theta_d)$ that outputs a single scalar. $D(x)$ represents the decision boundary for deciding if x came from the real dataset or P_g. We train D to maximize the probability to predict the correct label for each category. The discriminator uses the binary cross entropy for its prediction. The function for the discriminator is then:

$$V(D,G) = \mathbb{E}_{x\sim P_{data}}[\log D(x)] + \mathbb{E}_{x\sim P_g}[\log(1-D(x))] \tag{2.5}$$

G is simultaneously trained to minimize $log(1-D(G(z)))$. The value function of the GAN model is then the two-player minimax game between D and G as shown below.

$$\min_G \max_D V(D,G) = \mathbb{E}_{\boldsymbol{x}\sim P_{data}(\boldsymbol{x})}[\log D(\boldsymbol{x})] + \mathbb{E}_{\boldsymbol{z}\sim P_{\boldsymbol{z}}(\boldsymbol{z})}[\log(1-D(G(\boldsymbol{z})))] \tag{2.6}$$

The discriminator tries to maximize $V(D,G)$ while the generator tries to minimize $V(D,G)$. During training, the parameters θ_d and θ_g are continuously updated in an iterative, numerical approach to deter overfitting. When the generator and the discriminator reach an equilibrium, the ideal models have been formed on the training set.

2.5.3 CGAN

The Conditional Generative Adversarial Network (CGAN) proposed by Mehdi Miraz [59], was another important extension to the GAN model. By conditioning both models on some extra input, the GAN model can be extended to be a conditional model.

The generative model receives another input with its random variable z, to be conditioned by some additional input y. The discriminator is also conditioned by the additional input, receiving its data either from the real dataset or the generator output x and the additional input y. This in turn yields that the discriminator expects the input data x to be of the certain type that the additional input specifies. When the error gets backpropagated, the generator is taught to generate data pertaining to the additional input y in order to fool the discriminator.

2.5.4 Conditional Tabular Generative Adversarial Network

The Conditional Tabular Generative Adversarial Network model (CTGAN) is one of the current state-of-the-art methods for generating artificial tabular data. Proposed by Xu et al. in the "Modeling Tabular Data using Conditional GAN" paper [60], it further expands on the TGAN model by the same author (See 3.3 TGAN for more information on this model). It remedies some of TGANs shortcomings and as a result, shows key improvements on common challenges of modeling tabular data using GANs.

One of the shortcomings that CTGAN improves on is the challenge of normalizing continuous data without information loss. With complex distributions, using a min-max transformation to normalize these distributions might lead to the vanishing gradient problem, making the generator see no improvements based on the information gained from the discriminator.

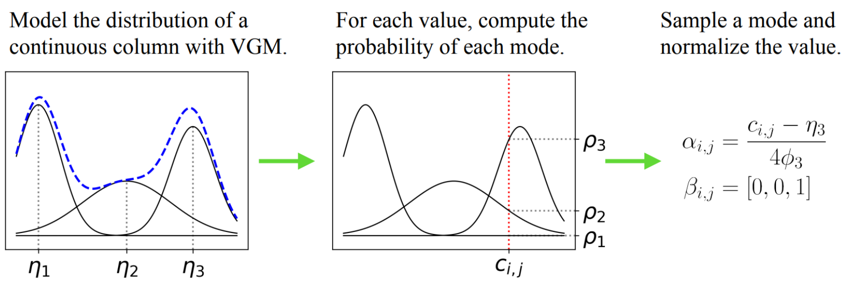

Model the distribution of a continuous column with VGM.
For each value, compute the probability of each mode.
Sample a mode and normalize the value.
η_1
η_2
η_3
ρ_3
ρ_2
ρ_1
$c_{i,j}$
$\alpha_{i,j} = \frac{c_{i,j} - \eta_3}{4\phi_3}$
$\beta_{i,j} = [0, 0, 1]$

Figure 2.15: An example of mode-normalization (Figure courtesy of Xu et al. [60])

While TGAN attempts to effectively sample continuous numerical values from non-Gaussian and multimodal distributions by using a Gaussian Mixture Model, CTGAN remedies this issue by applying mode-specific normalization. More specifically, it converts continuous values of complex distributions into a bounded vector by using the Variational Gaussian Mixture Model (GMM) on every individual continuous column separately. The GMM finds m modes relating to each Gaussian in the complicated distribution parameterized by some weight μ_k and standard deviation ϕ_k. It then computes the probability of each continuous value in a continuous column to be coming from each mode inferred by the GMM. You can see an example of this illustrated above.[fig. 2.15].

Another problem that CTGAN improves on is the issue of class imbalance in categorical columns. Class imbalance pertaining to the challenge of leveraging GANs in generating tabular data puts focus on the generative model. More specifically, the imbalance happens when the number of some classes is significantly higher than other classes. If the training data are randomly sampled during training, the rows which hold categorical values minorly represented would not be sufficiently represented, leading to the generator not capturing these values in its modeled distribution. If it resamples, the real distribution is not kept.

Here is where CTGAN takes inspiration from CGAN[59], leveraging training-by-sampling coupled with a conditional generator to handle categorical imbalance when modeling discrete columns. Using a conditional input vector allows for the conditioning of a value on a specific column via one-hot encoding. The condition is chosen through training-by-sampling.

This results in the data to be sampled during the training procedure to include all possible categories of discrete columns such that every category of discrete columns occurs evenly. The conditional generator takes a prior random noise as well as the conditional vector to force the desired condition, such that the distributions generated by the generator are the same as the real data distributions.

Finally, the training uses generator loss by adding the cross-entropy between the conditional vector and the generated sample to the loss expression. This makes the generator follow the conditioning.

The output of the generator is then evaluated by the discriminator to calculate the difference between the generator and the real conditional distribution and uses WGAN loss with gradient penalty[61] to further train the model.

The network structure is described in the paper like so:

We use fully-connected networks in generator and critic to capture all possible correlations between columns. Specifically, we use two fully-connected hidden layers in both generator and critic. In generator, we use batch-normalization and Relu activation function. After two hidden layers, the synthetic row representation is generated using a mix activation functions. The scalar values α_i is generated by $tanh$, while the mode indicator β_i and discrete values d_i is generated by gumbel softmax. In critic, we use leaky relu function and dropout on each hidden layer.

Finally, the conditional generator $G(z, cond)$ can be formally described as

$$\begin{cases} h_0 = z \oplus \text{ cond} \\ h_1 = h_0 \oplus \operatorname{ReLU}\left(\operatorname{BN}\left(\operatorname{FC}_{|\text{cond }|+|z| \rightarrow 256}\left(h_0\right)\right)\right) \\ h_2 = h_1 \oplus \operatorname{ReLU}\left(\operatorname{BN}\left(\operatorname{FC}_{|\text{cond }|+|z|+256 \rightarrow 256}\left(h_1\right)\right)\right) \\ \hat{\alpha}_i = \tanh\left(\operatorname{FC}_{|\text{cond }|+|z|+512 \rightarrow 1}\left(h_2\right)\right) & 1 \leq i \leq N_c \\ \hat{\beta}_i = \operatorname{gumbel}_{0.2}\left(\operatorname{FC}_{|\text{cond }|+|z|+512 \rightarrow m_i}\left(h_2\right)\right) & 1 \leq i \leq N_c \\ \hat{\mathbf{d}}_i = \text{ gumbel }_{0.2}\left(\operatorname{FC}_{|\text{cond }|+|z|+512 \rightarrow |D_i|}\left(h_2\right)\right) & 1 \leq i \leq N_d \end{cases}$$

We use the PacGAN [62] framework with 10 samples in each pac to prevent mode collapse. The architecture of the critic (with pac size 10) $(\mathcal{C}(r1, \ldots, r10, cond1, \ldots, cond10)$ can be formally described as:

$$\begin{cases} h_0 = \mathbf{r}_1 \oplus \ldots \oplus \mathbf{r}_{10} \oplus \text{ cond }_1 \oplus \ldots \oplus \text{ cond }_{10} \\ h_1 = \operatorname{drop}\left(\text{ leaky }_{0.2}\left(\mathrm{FC}_{10|\mathbf{r}|+10| \text{ cond } | \rightarrow 256}\left(h_0\right)\right)\right) \\ h_2 = \operatorname{drop}\left(\text{ leaky }_{0.2}\left(\mathrm{FC}_{256 \rightarrow 256}\left(h_1\right)\right)\right) \\ \mathcal{C}(\cdot) = \mathrm{FC}_{256 \rightarrow 1}\left(h_2\right) \end{cases}$$

We train the model using WGAN loss with gradient penalty [61]. We use Adam optimizer with learning rate $2 \times 10 - 4$.

Chapter 3

Related work

This section introduces work done by others that is related or is similar to our project.

3.1 Ditto data augmentation using BERT

As briefly mentioned in 2.3.2 GPT-2, BERT is a state of the art language model which Ditto uses in their data augmentation method. The method is described in detail in the Ditto paper [25] and is summarised below.

Firstly, ditto applies one of the operators described in the table below [tab. 3.1] to an entry. The BERT Language Model is then used to interpolate between the augmented entry and the original entry [fig. 3.1] to create an entry that is ”in-between” the two entries. Li et al. call this method ”MixDA”.

Operator	**Explanation**
span_del	Delete a randomly sampled span of tokens
span_shuffle	Randomly sample a span and shuffle the tokens’ order
attr_del	Delete a randomly chosen attribute and its value
attr_shuffle	Randomly shuffle the orders of all attributes
entry_swap	Swap the order of the two data entries e and e’

Table 3.1: Data augmentation operators in Ditto. The operators are 3 different levels: span-level, attribute-level, and entry-level. All samplings are done uniformly at random.

This is done because the augmented entries might end up with wrong labels as a result of their augmentation. By interpolating between the original and the augmented entries, the resulting partial entry is expected to be less distorted, and more likely to have the correct label.

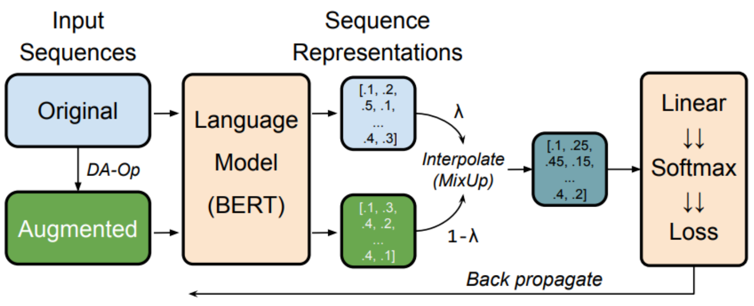


Figure 3.1: Data augmentation with MixDA. (Figure courtesy of Li et al.[25])

3.2 TVAE model

In the CTGAN paper [60], Xu et al. propose another model to create tabular data with. They adapted the Variational Auto-Encoder to be able to generate tabular data, and called it TVAE. The architecture is described in the paper like so:

The design of the network $p_\theta(\mathbf{r}_j \mid z_j)$ that needs to be done differently so that the probability can be modeled accurately. In our design, the neural network outputs a joint distribution of $2N_c + N_d$ variables, corresponding to $2N_c + N_d$ variables $\mathbf{r}_j$. We assume $\alpha_{i,j}$ follows a Gaussian distribution with different means and variance. All $\beta_{i,j}$ and $\mathbf{d}_{i,j}$ follow a categorical PMF. Here is our design.

$$\begin{cases} h_1 = \operatorname{ReLU}\left(\mathrm{FC}_{128\to128}(z_j)\right) & \\ h_2 = \operatorname{ReLU}\left(\mathrm{FC}_{128\to128}(h_1)\right) & 1 \le i \le N_c \\ \bar{\alpha}_{i,j} = \tanh\left(\mathrm{FC}_{128\to1}(h_2)\right) & 1 \le i \le N_c \\ \hat{\alpha}_{i,j} \sim \mathcal{N}\left(\bar{\alpha}_{i,j}, \delta_i\right) & 1 \le i \le N_c \\ \hat{\beta}_{i,j} \sim \operatorname{softmax}\left(\mathrm{FC}_{128\to m_i}(h_2)\right) & 1 \le i \le N_d \\ \hat{\mathbf{d}}_{i,j} \sim \operatorname{softmax}\left(\mathrm{FC}_{128\to|D_i|}(h_2)\right) & \\ p_\theta(\mathbf{r}_j \mid z_j) = \prod_{i=1}^{N_c} \mathbb{P}\left(\hat{\alpha}_{i,j} = \alpha_{i,j}\right) \prod_{i=1}^{N_c} \mathbb{P}\left(\hat{\beta}_{i,j} = \beta_{i,j}\right) \prod_{i=1}^{N_d} \mathbb{P}\left(\hat{\alpha}_{i,j} = \alpha_{i,j}\right) & \end{cases}$$

Here $\hat{\alpha}_{i,j}, \hat{\beta}_{i,j}, \hat{\mathbf{d}}_{i,j}$ are random variables. And $p_\theta(\mathbf{r}_j \mid z_j)$ is the joint distribution of these variables. In $p_\theta(\mathbf{r}_j \mid z_j)$, weight matrices and δ_i are parameters in the network. These parameters are trained using gradient descent. The modeling for $q_\phi(z_j \mid \mathbf{r}_j)$ is similar to conventional VAE.

3.3 TGAN

Earlier work by Lei Xu introduces the Tabular Generative Adversarial Network (TGAN) [63], a GAN model which attempts to generate artificial tabular data for general purposes. It uses LSTM with attention to its generative mode to generate data column by column.

While the paper focuses on the generation of tabular data with mixed variable types such as discrete and continuous, their model would only support generation of tabular data containing numerical and categorical features. This would later be improved on in CTGAN [60]. TGAN was however still an important stepping stone for generation of tabular data and put focus on efficiently modeling marginal distributions.

3.4 TableGAN

Another paper that tackled the issue of generating tabular data was the tableGAN [64]. Not to be confused with TGAN [63], this paper puts focus on privacy concerns regarding personal identifiable information (PII) and re-identification attacks. While some common methods like adding noise to real data, anonymization, or other modifications can be used, it often degrades the data quality. Privacy level and data utility are often the inverses of each other in regards to usability. It attempts to resolve this by generation of artificial tabular data.

TableGAN aims to generate data that stay statistically similar to its real counterpart, while also not incurring information leakage. It handles categorical, discrete, and continuous variables while leaving other types for further work. It is inspired by the Deep Convolutional Generative Adversarial Network and uses convolutional neural networks for its architecture. It consists of three neural networks, with the third neural network being a classifier to add semantic integrity. It uses cross-entropy loss to optimize its prediction accuracy on synthetic data for its end goal of exhibiting optimum results with the balance of trade-off between privacy and machine learning efficiency.

Chapter 4

Data

In this project, we use 12 datasets that were used in the DeepMatcher study [65]. These datasets were chosen because of their manageable yet varying size. Using these datasets also allows us to compare our findings to the results found in the Ditto and Magellan papers [25, 24].

As mentioned in 2.2.4 Ditto, Ditto uses a specific schema for its datasets. All of the datasets used have been formatted to this schema and split up into train/test/validate sets when we got them.

4.1 Term definitions

When talking about data used in this project, to avoid vagueness and misunderstandings, we define two terms:

Entity: In this project, we define an entity to be a collection of attributes that describe a distinct real-world object. This definition is in line with the definition of an entity in the context of entity matching.

Entry: We define an entry to be a set of two entities and an indicator on whether those entities match or not. This definition is in line with the schema that ditto uses, meaning that a line in the dataset is the same as an entry in a dataset.

4.2 Describing data used

This collection of 12 datasets is split into three categories, Structured, Dirty and Textual. The datasets in the Structured and Textual categories are comprised of data crawled from websites and then pre-processed to fit the format [1].

[1]Detailed information about each dataset and how they were made can be found here: `https://github.com/anhaidgroup/deepmatcher/blob/master/Datasets.md`

The Dirty datasets were made using the Structured datasets, and inserting other attribute values into the title attribute value with a 50% probability. This simulates a common kind of dirty data.

Both the Dirty and Structured categories contain entries with entities that have several attributes and fairly short string values for those attributes. The Textual category, on the other hand, contains entries with entities that consist of few attributes with fairly long string values.

The datasets vary greatly in size, with the biggest being at 28707 entries while the smallest being 450 entries. Each dataset and its size is detailed in the table below [tab. 4.1].

Type	Name	Size	Attributes
Structured	Amazon-Google	11460	3
	Beer	450	4
	DBLP-ACM	12363	4
	DBLP-GoogleScholar	28707	4
	Fodors-Zagats	945	6
	iTunes-Amazon	539	8
	Walmart-Amazon	10242	5
Textual	Abt-Buy	9577	3
Dirty	DBLP-ACM	12363	4
	DBLP-GoogleScholar	28707	4
	iTunes-Amazon	539	8
	Walmart-Amazon	10242	5

Table 4.1: Datasets used in the DeepMatcher study

4.3 Other datasets

Ditto [25] evaluated on 6 additional datasets. One of these datasets is the ”Company” dataset, which is in the Textual category. The other 5 datasets come from the WDC Product Data Corpus [66]. The datasets each contain four versions of the same dataset in different sizes.

All datasets contain the same type of entries as the Textual dataset from the DeepMatcher study datasets, meaning that the entries consist of entities with few attributes and fairly long string values for those attributes.

Chapter 5

Methods

In this section we outline the methods we use for generating data, and each method is given a high-level explanation. Following the method explanations, some implementation details are given.

5.1 Data augmentation method

Data augmentation methods have been used in the fields of computer vision for a long time [67]. The Ditto system also have their own data augmentation method which, according the their paper [25], improves the matcher's results when used to augment the data while the model is learning. We came up with our own simple data augmentation method to compare it against other more complex data generation methods.

5.1.1 Generation of non-matches

To generate a non-match entry, we borrowed a technique from the field of evolutionary algorithms, namely recombination [68]. By regarding each entry in the data as an individual in the context of evolutionary algorithms, one can perform random recombination of two (or more) individuals to produce new individuals.

Each column can be viewed as a gene, and the value of that column, the value of the gene. The new individual will consist of a random combination of attributes from its parents, as illustrated in the figure below [fig. 5.1].

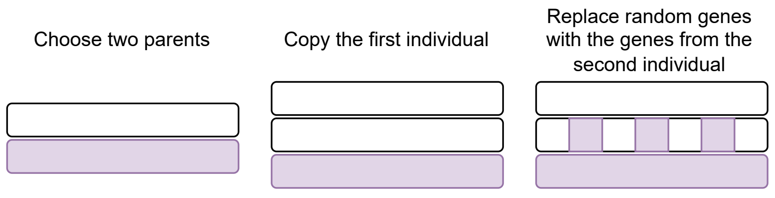

Choose two parents
Copy the first individual
Replace random genes with the genes from the second individual

Figure 5.1: Recombination of two individuals

Creating a non-matching entry this way may result in an accidental creation of a matching entry that is wrongly labeled as a non-matching entry. To reduce the chance of this happening, we pick parent individuals only from the set of non-matching entries in the dataset. We believe that by doing so, the chances of any further wrongly labeled matches being created are negligible and can be chalked up to noise in the dataset.

5.1.2 Generation of matches

The same method as described in the non-match subsection will not work in generation of matching entries in the data. This is because counter to before, the content of the genes has to match. Recombining two entries, even if they are only picked from the set of matching entries, does not guarantee that the new individual will indeed be a matching entry. Following the inspiration from evolutionary algorithms, we borrowed the mutation method [68].

By selecting only one entry from a set of matching entries, the values of each column may be viewed as genomes, with each symbol (such as letters, numbers, etc) as individual genes. Some of these genes can then be randomly selected and changed into some pre-determined counterpart, or a random symbol, as shown in the figure below [fig. 5.2].

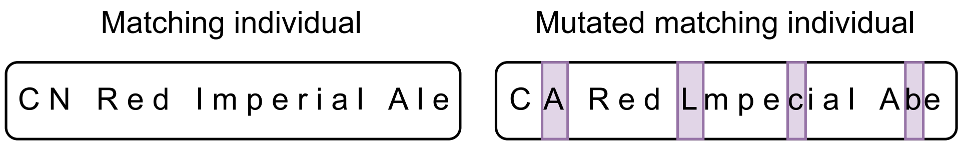


Figure 5.2: Individual mutating the marked letters, creating a new individual

The more symbols are altered, the higher the chance to loose the matching property of the entry. By keeping this number low, we minimize the chance of a wrongly labeled matching entry being created. We believe this chance is negligible and can be chalked up to noise in the dataset.

5.2 Standard GPT-2

The standard GPT-2 model can be used to generate a response to a textual prompt based on its language understanding and the context of the prompt. We leverage this context awareness by creating prompts that encompass the information we want the new entries to contain, such as any special key-words or other quirks of the data.

To create such a prompt we select several entries from the dataset at random, and concatenate them into one string. These entries serve as the context that GPT-2 needs to generate a response. A new random entry is selected from the dataset which will have one of the entities removed from it. This bisected entry will serve as the basis for the new entry to be generated. The new bisected entry is then concatenated to the context string, creating the prompt that will be sent into the model [fig. 5.3]. When done so, the model should generate the missing entity based on the context of the previous entries. The amount of entries used for the context varies due to the length of these entries. GPT-2 has a maximum length for the text it can process. If the prompt exceeds this length, then there is no space to fill in the missing entity and unexpected behaviour can occur, such as the model not filling in the bisected entry, the model cutting off some of the context, etc.

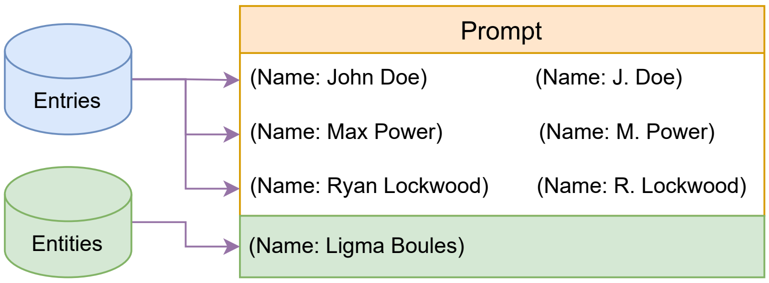


Figure 5.3: Construction of the prompt to be sent into the model

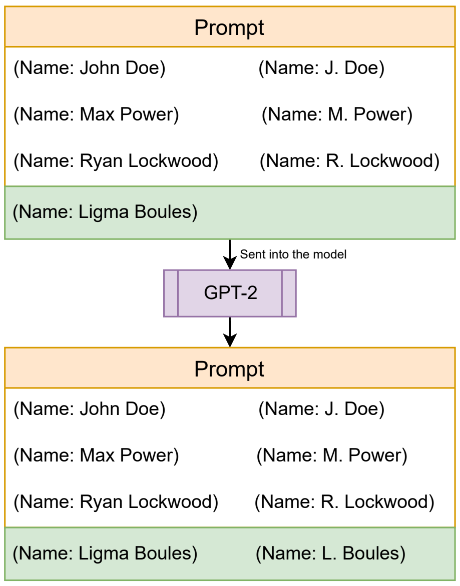


Figure 5.4: The prompt is sent into the GPT-2 model, and generates the missing entity in the new entry

The process of creating matches and non-matches using this method is the same. The only difference is that when selecting entries from the dataset for the prompt, we select from a set of matching entries when generating matches, and a set of non-matching entries when generating non-matches. This is done to ensure that the context in the prompts reflects what we want the model to do. The hope is that the model recognises what a match is, and when filling in the missing entity, it generates data that will result in the reformed entry remaining a match. Similarly, when creating a non-matching entry, the model should fill in the missing entity with information that creates a non-matching entry [fig. 5.4].

5.3 Fine tuning GPT-2

To fine-tune the standard GPT-2 model, additional neural network layers can be added on top of it. This makes it so we can keep all of the semantic knowledge of language the GPT-2 model has. Training these new outer layers on our data makes the GPT-2 model generate strings that are structurally similar to the training data supplied. This is usually used to emulate different types of the same language, like a regional dialect for instance. We use this instead to emulate the ditto dataset entry structure.

Two separate models are created for generating matches and non-matches. Predictably, the match model is trained on the set containing only matching entries, while the non-match model is trained on the set containing only non-matching entries.

When generating new entries, a random entry is selected from the dataset, and one of the entities is removed. This bisected entry is then given to the model as the prompt. The model then generates in the missing entity based on the training the model has undergone, as well as the semantic understanding of language [fig. 5.5].

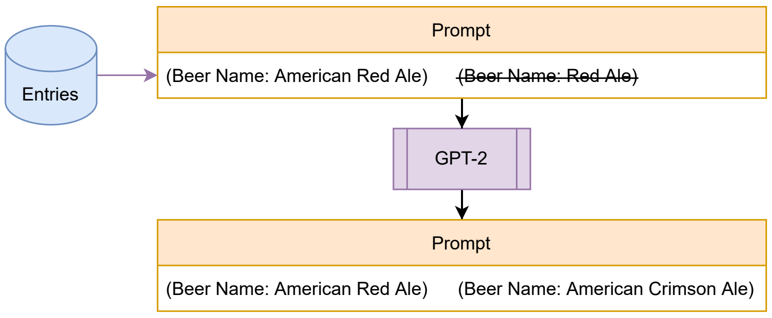

Entries
Prompt
(Beer Name: American Red Ale)
~~(Beer Name: Red Ale)~~
GPT-2
Prompt
(Beer Name: American Red Ale)
(Beer Name: American Crimson Ale)

Figure 5.5: Creating a prompt out of an entry and sending it to the model which generates a new entry

5.4 CTGAN

The CTGAN model sees improvements in the generation of tabular data, taking into consideration how to handle class imbalance on discrete columns, and better model data distributions of continuous values with mode-normalization. With this in mind, expectations to see good results on most datasets mentioned in 4 Data were high, as most of them follow the expected characteristics which CTGAN are made to handle. Moreover, we left the existing non-values from the real datasets as is to explore how CTGAN handles these values for its generation step, coincidentally also introducing fuzzy data into the data distribution.

In earlier experiments when training on datasets with both matching and non-matching entries, CTGAN's ability to capture correct labels for each class seemed to be a vain attempt. We consider this a result of the nature of the entity matching task. More specifically, it struggles to see the correlation of the label column to the added task of inference for similarity over the attributes between the two entities in an entry. Later in our experiments, we concluded to separate the training set on the match and non-match types to capture the data distribution of these sets individually. This was also done with the intent of generating better examples for training the EM system where the distinction between matching and non-matching entries is at focus.

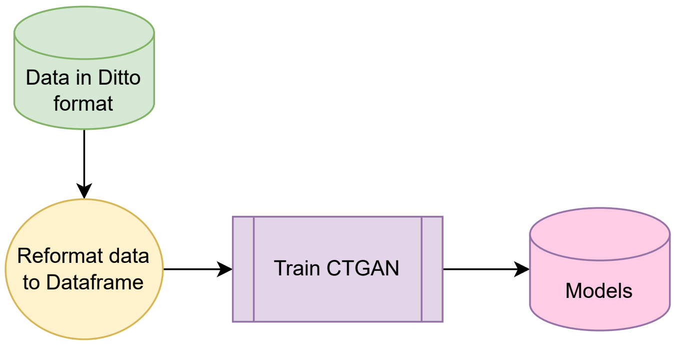


Figure 5.6: Data is reformatted into a tabular format. CTGAN trains on the data, and the resulting models are stored when finished.

We give the CTGAN model its training set and perform training [fig. 5.6]. After the training step, the resulting model is used to generate our required data [fig. 5.7]. The sample size is the size of the original dataset variant when generating the artificial dataset. We do this to generate a sample set that has the required room to project as much of its modeled data distribution as possible.

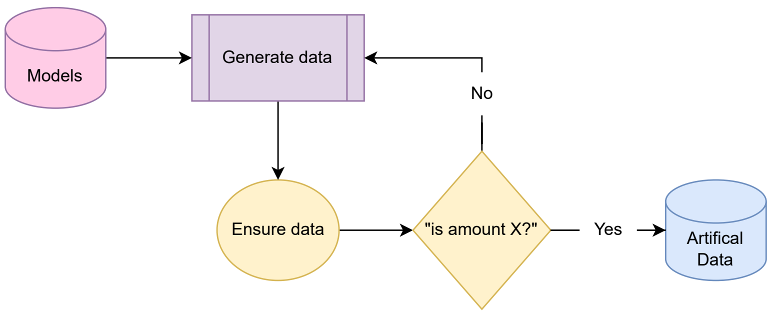


Figure 5.7: A specific CTGAN model generates data and the resulting data is filtered. This continues until a sufficient amount is reached.

The CTGAN model sometimes struggles with correlating the attributes of both entities in a data entry. This in turn yields "matching" entries that can include entities too dissimilar. Because of this, we implement an insurance method for the generated set which considers the entity's likeness of an entry and filters through only adequate results. More on this in 5.5.6 Ensure data method.

Because of the insurance method, the data distribution in the sample set can often be skewed in generation of matching entries. The insurance step is more active during the generation of matching entities as the correlation between the attributes is more important in those scenarios. In contrast, the insurance method during the generation of non-matching entities remains almost non-active.

5.5 Implementation details

This section covers some implementation details, like the hardware used during the project, additional helper code, or other things we deem worthy of mention.

5.5.1 Hardware used

While working on the data generation methods, we discovered that our personal computers are not powerful enough to efficiently run these models. Deep Learning is known for being computationally expensive, and since GPT-2 and CTGAN are both deep learning methods, this posed a problem during this project. To decrease the time used to run the models, we therefore pivoted to using Google Colaboratory and Idun.

Google Colaboratory

Google Colaboratory, or 'Colab', is a free tool made by Google which allows users to run their code in a cloud environment with access to CPU and GPU resources[1]. The resources available fluctuate, and sometimes are even unavailable, due to them

[1]More information at: `https://research.google.com/colaboratory/faq.html`

being shared across all users. The platform also allows for easy code execution by not needing any setup. This is because Colab is a browser based Jupyter notebook.

The reasons above make the platform good for research and development purposes. However, not only is it not suited for running many programs that are meant to run for long, but such actions are also prohibited by Google.

Idun

The Idun cluster is a project at NTNU that aims to provide a highly available computing platform. The cluster uses the Slurm Workload Manager [69] to schedule program execution and allocate resources to those programs. The cluster consists of 2 login nodes and 15 compute nodes. The login nodes are are mainly used for system setup and other user interactions with the server, such as queuing Slurm jobs. The compute nodes, as the name suggests, are used for computational purposes. All of the nodes have powerful multi-core CPUs. All but 6 of the compute cores also have access to several powerful GPUs with each having at least 12GB of VRAM[2].

Google Colab was used for research and development, and Idun was used to run the code in bulk operations. Both the Ditto and Magellan Entity Matching systems were also ran on the Idun cluster, which significantly decreased the time needed to run these models.

5.5.2 Ditto data parser

GPT-2, as many other language models, is primarily meant for natural language generation. As a side-effect of this, the model had a tendency to sometimes generate unrelated texts, rather than just new data entries. To mitigate this, we wrote a data parser.

The way our parser for ditto data works, is by first taking in the data generated by the GPT-2 model. Since we always use a real entity from an entry when generating new data, we split the generated text into two parts. The real part of the new entry gets scanned for the fields that are present in the entry. The generated part is then scanned for those same fields. The new entry is considered valid if the generated part has those fields and values for those fields. The parser then creates a string which is in the correct schema out of the entry data [fig. 5.8]. The code for this parser can be viewed on our github page linked in Appendix A.3 Ditto data parser.

[2]More information at: `https://www.hpc.ntnu.no/idun/`

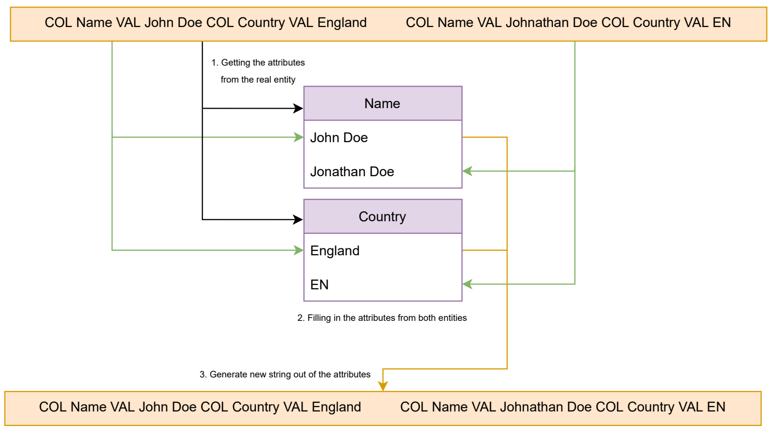

COL Name VAL John Doe COL Country VAL England
COL Name VAL Johnathan Doe COL Country VAL EN
1. Getting the attributes from the real entity
Name
John Doe
Jonathan Doe
Country
England
EN
2. Filling in the attributes from both entities
3. Generate new string out of the attributes
COL Name VAL John Doe COL Country VAL England
COL Name VAL Johnathan Doe COL Country VAL EN

Figure 5.8: The parser validating an entry and generating a string

The data generated for Ditto can be used with Magellan with minimal pre-processing, as Magellan only needs the data to be in a pandas dataframe. However, other models might not play as nice with this schema. Because of this issue of differing schemas used by the different models, a data parser will need to be written specifically for these models.

5.5.3 Data generation using GPT-2

When implementing the GPT-2 model for data generation, we used the HuggingFace Transformer library. The library includes a ”pipeline” object, which can either use a standard GPT-2 model or a fine-tuned GPT-2 model to generate text. It takes a prompt as an argument, and takes care of tokenizing the text and initializing the model, and when it is done, it returns the text generated by the model.

The GPT-2 model however has a limit on the length of texts it can process. Both the prompt and the eventually generated text all have to fit within this maximum length limit. This was slightly problematic when generating data from the prompts made for the standard GPT-2 model. The prompt needs to have some complete data entries in it for the method to work, but sometimes a set amount of entries might produce a prompt that is too long to generate text out of. To solve this issue, we ensure that the prompt does not exceed the maximum length of 512 tokens, accounted for the missing entity that will be generated.

Sometimes the generated data does not result in a new entry that is valid. When this happens, the generated text is disregarded and the generation starts again. This is done for both the fine-tuned and non-fine-tuned GPT-2 models. To validate the entry, a method in the data parser is used. This validation method is outlined in 5.5.2 Ditto data parser. The generation and validation cycle is repeated until a valid new entry is generated.

5.5.4 The Synthetic Data Vault

The Synthetic Data Vault (SDV) is a collection of libraries for creating artificial data. The SDV Project was originally developed in 2016 at MIT's Data to AI Lab. With the intention of expanding the project further, DataCebo was founded in 2020 and became the sole proprietor of the SDV project. [70]

SDV includes tools for modeling datasets in order to create new artificial data which aims to be as identical as possible to the original dataset in terms of format and statistical characteristics. From this library, we have utilized their implementation of CTGAN. [71]

5.5.5 Auxiliary data parser

The specific implementation of CTGAN that we utilize takes in a training set as a data frame structure. However, the entire data repository exists solely in Ditto format. To mitigate this and be able to freely exchange data between CTGAN and Ditto, we created a simple data reformatter. This script can also be viewed in Appendix A.4 Auxiliary data parser.

The auxiliary data parser converts our data files between the Ditto format and Magellan format [fig. 5.9]. Magellan and CTGAN operate on data frame structures, while GPT-2, the Augmentation algorithm, and Ditto operate on specifically formatted strings. To correctly read the data from the base data repository for Magellan and CTGAN, and exchange data between the generators for each EM matcher, this script is utilized.

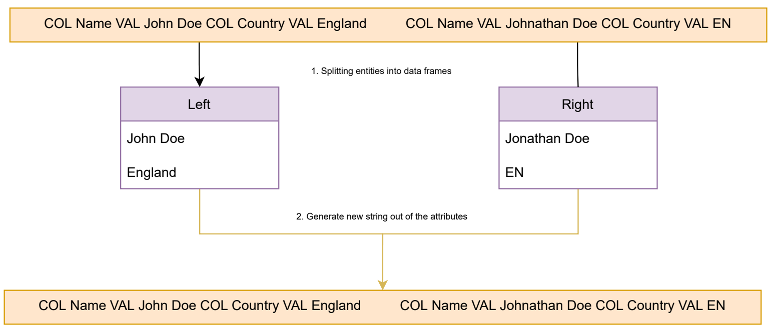

COL Name VAL John Doe COL Country VAL England
COL Name VAL Johnathan Doe COL Country VAL EN
1. Splitting entities into data frames
Left
John Doe
England
Right
Jonathan Doe
EN
2. Generate new string out of the attributes
COL Name VAL John Doe COL Country VAL England
COL Name VAL Johnathan Doe COL Country VAL EN

Figure 5.9: The data parser converting between different formats.

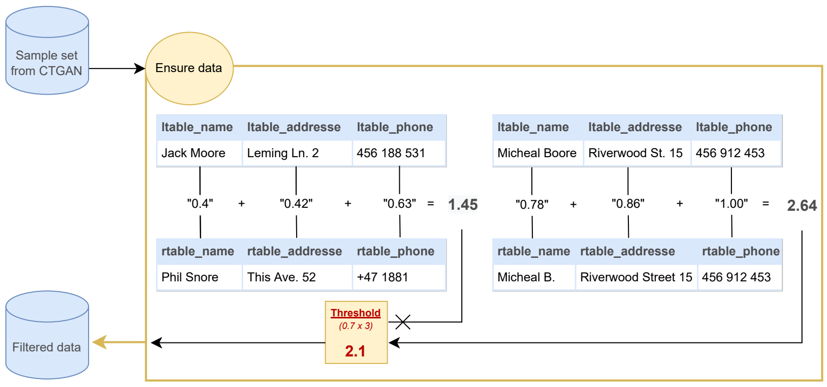
To parse ditto formatted strings to a data frame structure, it first collects each column name from the first line of text. It then iterates over each line of text, switching between the left side and right side of the line. While doing so, it captures the column name following "COL" and appends the data following "VAL" to the corresponding column name in the data frame.

To parse data frame structured data to ditto formatted strings, it first collects the column name of the original dataset to make sure no column name gets inadvertently renamed. It then iterates for every row in the data frame, creating a string with the values of the row prepended with the paired column name and prepended “COL” and “VAL” tokens.

5.5.6 Ensure data method

The insurance method is mainly used for over the CTGAN generated data. The CTGAN implementation from SDV allows for a rejection policy. This lets the user create their own conditions on what the model can generate as output. It does not help the training of the model but acts more like a filtering process for what the model is allowed to output from the sample set. As the implementation of this is awkward, we instead opted to run the filtering process over the generated output and generate new sample sets until the total amount of data reached the amount of the original dataset.

After CTGAN has finished generating a sample set, it gets sent to the data insurance method with a threshold. The insurance method iterates over each entry in the sample set, comparing each attribute of both entities. It uses the Levenshtein distance between the two attributes to determine their similarity. The distance between the two attributes is the minimum number of single-character changes required for both attributes to be identical, with added weight on substitution. The result is normalized to a decimal value between 0 to 1. The distance for each attribute is added up while the threshold is multiplied by the number of attributes to compare, with some allowed variance dependent on the total attributes. If a data entry does not pass the threshold, it gets removed [fig. 5.10].



ltable_name	ltable_addresse	ltable_phone
Jack Moore	Leming Ln. 2	456 188 531

rtable_name	rtable_addresse	rtable_phone
Phil Snore	This Ave. 52	+47 1881

ltable_name	ltable_addresse	ltable_phone
Micheal Boore	Riverwood St. 15	456 912 453

rtable_name	rtable_addresse	rtable_phone
Micheal B.	Riverwood Street 15	456 912 453

Figure 5.10: Attributes between two entities in an entry are compared. If the threshold is not reached, the entry is discarded.

With some attributes being identical, giving it a distance of 1, while others being less comparable yet maintaining a matching nature for both entities, we add variance to give wiggle-room for what constitutes a matching entry. The method as a whole is a simple approach as we do not want to put too much weight on the insurance of data generated, but rather give a simple nudge to what constitutes a “match”. The a link to its implementation can also be found in Appendix A.5 CTGAN data generation generation source code.

Chapter 6

Experiments

In this section, we outline the experiments that were performed in this project, as well as showing their results.

6.1 Supplementing artificial data

This experiment was meant to test whether the Ditto and Magellan entity matching models can achieve better performance by increasing the amount of data they train on. This was achieved by generating artificial data and using it to supplement the original data.

By using the methods of data generation described in 5 Methods, three datasets were created using each of the datasets described in 4 Data:

- Real dataset supplemented with artificial non-matches only
- Real dataset supplemented with artificial matches only
- Real dataset supplemented with both artificial matches and non-matches

We generated the same amount of matches and non-matches as in the original datasets. This doubled the amount of data after supplementation. By supplementing the real data with either artificial matches or non-matches, effects of increasing each of them separately on the performance of the models can be observed [fig. 6.1].

The 3 generated datasets were then each used to train Ditto and Magellan. The entity matching models were also trained on the real datasets as a baseline performance measure. This baseline was used to compare the results. It is important to note that only the training data was supplemented with artificial data, while the testing and validation sets were left 100% real. Supplementing these datasets would corrupt the results obtained, as comparing artificial data to other artificial data made the same way would surely introduce unwanted effects.

Given that Ditto is a deep learning-based system, the results of the model might vary due to random weight initialisation when training the model. Because of this,

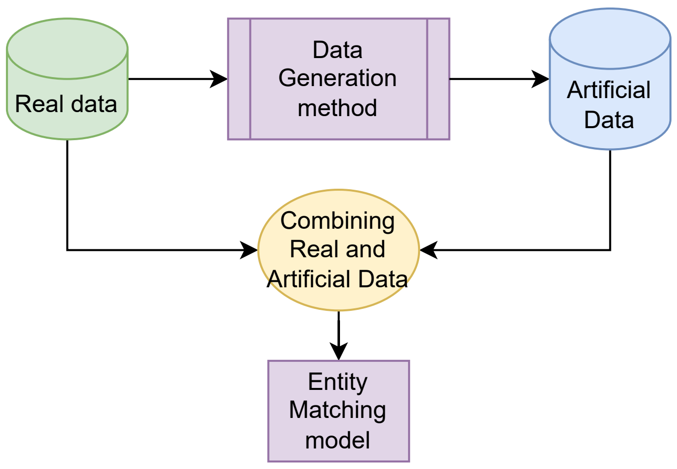


Figure 6.1: The pipeline for generating and testing the datasets in the first experiment.

the Ditto model was ran 3 times on each of the created datasets, as well as the real dataset, and the average f1 score was used when comparing the results.

When supplementing the real datasets with only the generated non-matching entries, most of the generation methods are generally on par with the baseline measurement, with a couple of notable exceptions. On the Ditto system, the Dirty and Structured iTunes-Amazon datasets when supplemented with generated data perform noticeably better than the non-supplemented dataset [fig. 6.2]. The Magellan system generally performs worse across the board when compared to Ditto. This is especially apparent when looking at the Augmentation method, which underperformed on all datasets except the Structured DBLP-ACM and the Structured iTunes-Amazon datasets [fig. 6.3].

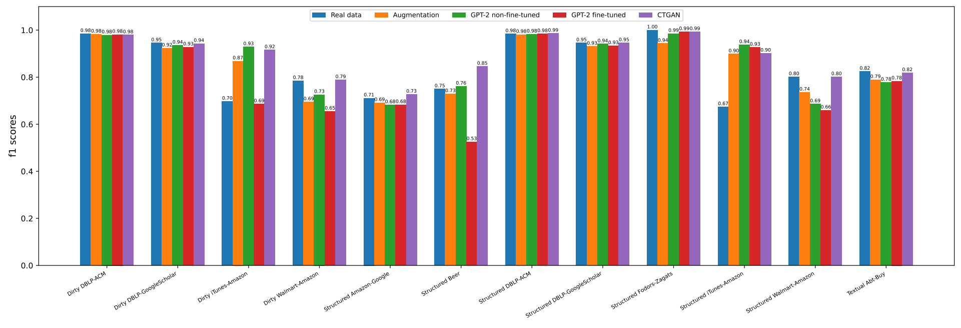


Figure 6.2: Real dataset supplemented with artificial non-matches which were generated using the full real dataset ran on Ditto

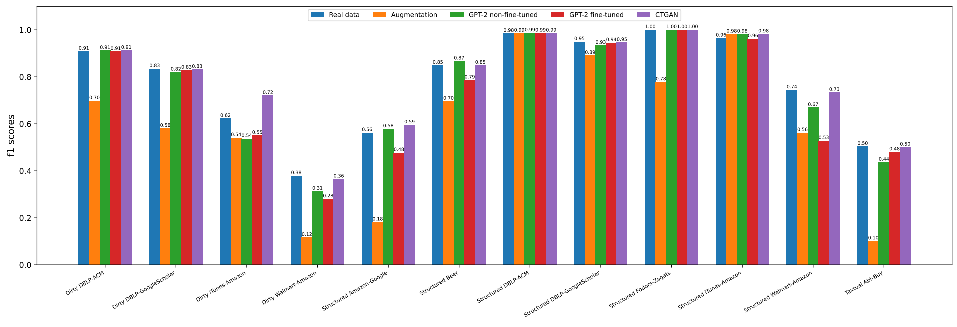


Figure 6.3: Real dataset supplemented with artificial non-matches which were generated using the full real dataset ran on Magellan

When supplementing the real datasets with only the generated matching entries, the Ditto system again outperforms Magellan on almost all datasets. However, compared to the datasets that were supplied with generated non-matches, the performance seems to have improved across the board on both systems [fig. 6.5, 6.4].

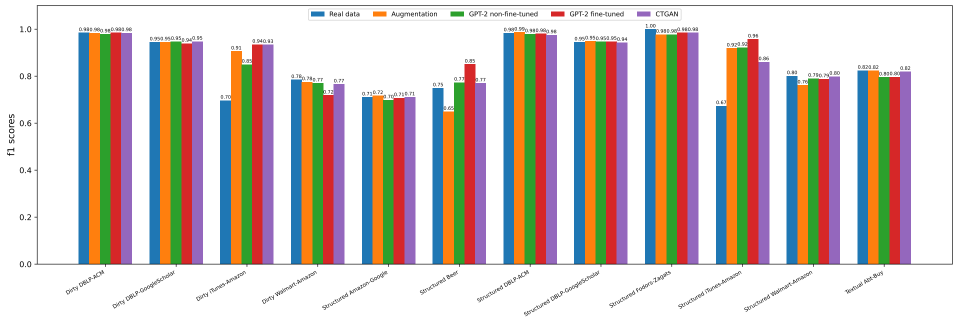


Figure 6.4: Real dataset supplemented with artificial matches which were generated using the full real dataset ran on Ditto

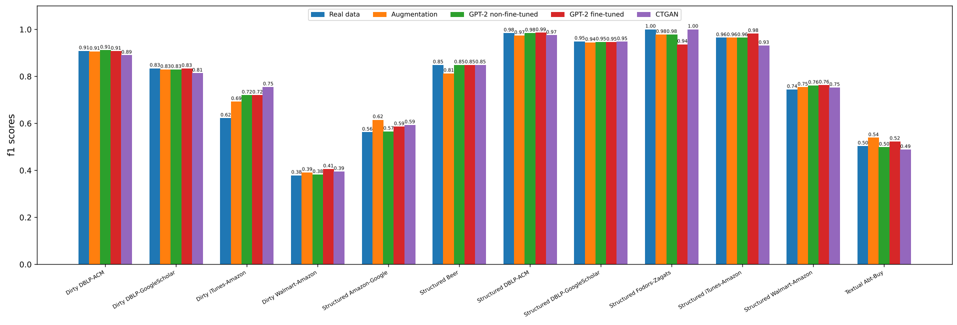


Figure 6.5: Real dataset supplemented with artificial matches which were generated using the full real dataset ran on Magellan

When supplementing the real datasets with both the generated matching and non-matching entries, performance is once again mixed, either being better than the baseline measurement, getting close to it, or underperforming. This holds true for both Ditto [fig. 6.6] and Magellan [fig. 6.7]. The results do seem to be slightly better than when supplementing with only the generated non-mathcing entries.

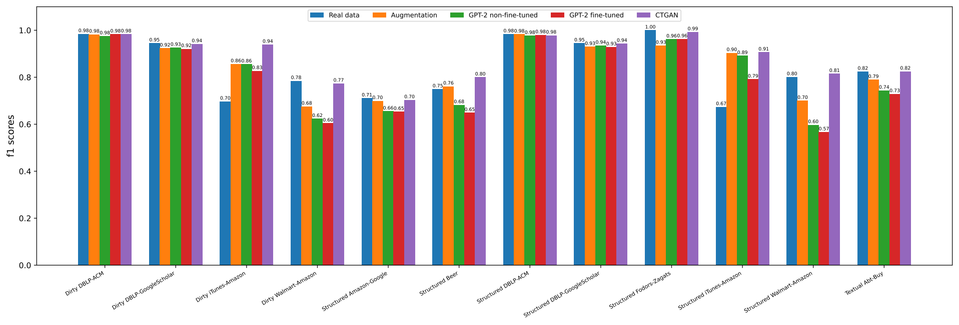


Figure 6.6: Real dataset supplemented with both artificial matches and non-matches which were generated using the full real dataset ran on ditto

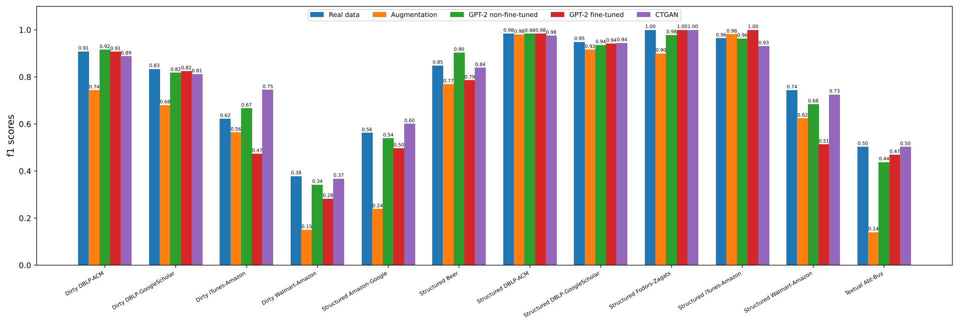


Figure 6.7: Real dataset supplemented with both artificial matches and non-matches which were generated using the full real dataset ran on Magellan

6.2 Simulating small amounts of data

This experiment was meant to test how the generated data affects the performance of the entity matching models. This was achieved by using 10% of the real data for artificial data generation and then generating enough data to reach the full amount of data in the dataset used. This means that when the generated data is combined with the real data, 90% of the data in the resulting dataset is artificial.

For this experiment we used only the datasets that are larger than 1000 entries. This is because the smaller datasets already have little amounts of data, and further decreasing them would leave an extremely small amount of data to generate entries out of. Because of this, we deemed them uninteresting for this experiment.

We combined the 10% of the real data that was used for generation together with the generated data to create these three datasets:

- Real dataset supplemented with artificial non-matches only
- Real dataset supplemented with artificial matches only
- Real dataset supplemented with both artificial matches and non-matches

By having the dataset supplemented with both artificial matches and non-matches be the same size as the original dataset, we could observe how the f1 score is affected when 10% of the data is real versus when 100% is real [fig. 6.8]. The entity matching models were trained on each of the 3 datasets. The baseline performance measures were achieved by training the models on both 10% and 100% of the real dataset. These baseline performance measures allowed us to observe how the f1 scores were affected by increasing the amount of data, and also compare them to the same amount of real data. As in the previous experiment, the entity matching systems were both evaluated on the full real testing and validation sets as to not corrupt the results.

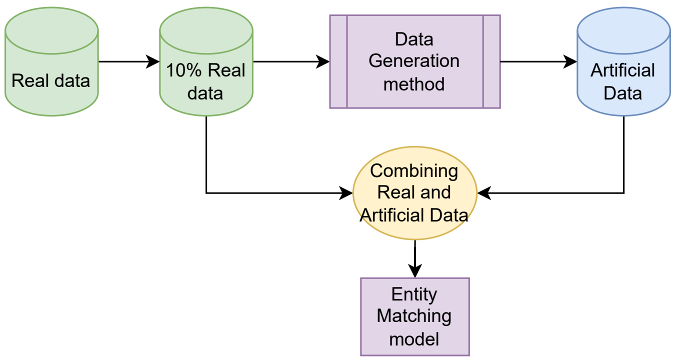


Figure 6.8: The pipeline for generating and testing the datasets in the second experiment.

Like in the previous experiment, due to the random initialisation of the weights the Ditto system was trained 3 times and the results were averaged.

When supplementing the data with only the generated non-matches, none of the methods managed to perform as well as the full real dataset. However, the GPT-2 and CTGAN methods did come close on some datasets, such as the Structured DBLP-ACM dataset. Noteably, the augmentation method was consistently the worst performing method on both Ditto [fig. 6.9] and Magellan [fig. 6.10]. All methods except for the CTGAN method performed badly on both systems on four of the datasets, those being the Dirty Walmart-Amazon, Structures Amazon-Google, Structured Walmart-Amazon and Textual Abt-Buy. The methods only reached half of the 10% real dataset f1 score, while CTGAN either matched it or exceeded it.

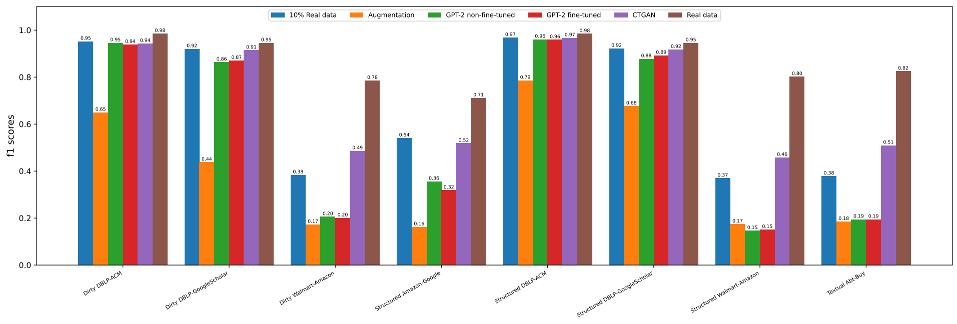


Figure 6.9: Real dataset supplemented with artificial non-matches which were generated using the 10% real dataset ran on Ditto

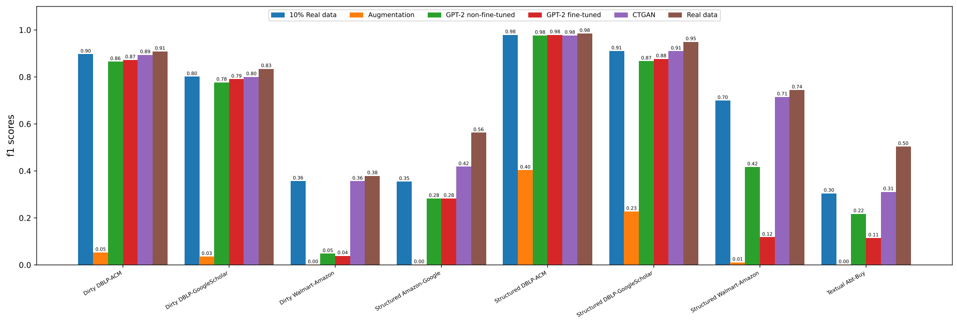


Figure 6.10: Real dataset supplemented with artificial non-matches which were generated using the 10% real dataset ran on Magellan

When supplementing the data with the generated matches, results across all methods and both systems improved. Each dataset on both systems had at least one method which exceeded the 10% real data f1 score. In the case of Dirty Walmart-Amazon on the Magellan system, the non-fine-tuned GPT-2 method even slightly exceeded the full real data f1 score. Just as when supplementing the datasets with generated non-mathces, the same four datasets have the worst f1 scores overall on both Ditto [fig. 6.11] and Magellan [fig. 6.12], those datasets being the Dirty Walmart-Amazon, Structures Amazon-Google, Structured Walmart-Amazon and Textual Abt-Buy.

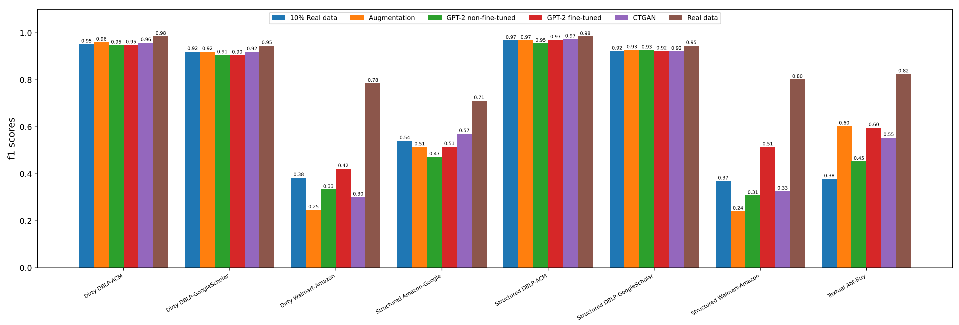


Figure 6.11: Real dataset supplemented with artificial matches which were generated using the 10% real dataset ran on Ditto

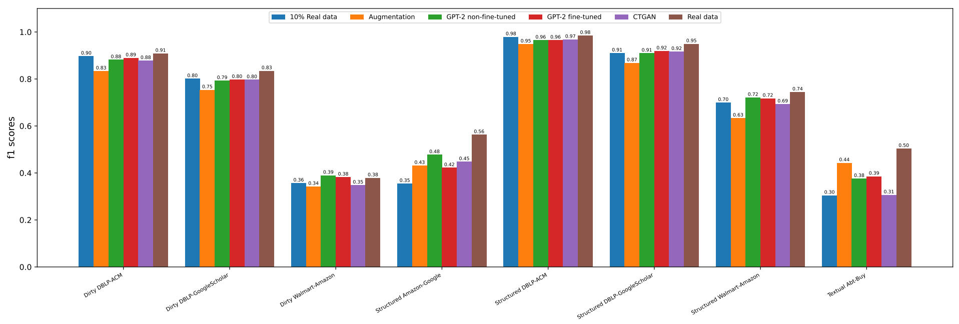


Figure 6.12: Real dataset supplemented with artificial matches which were generated using the 10% real dataset ran on Magellan

When supplementing the data with both the generated matches and non-matches, the performance dropped on both systems throughout all datasets. It is still however better than when only supplementing the data with the generated non-matches [fig. 6.13] [fig. 6.14].

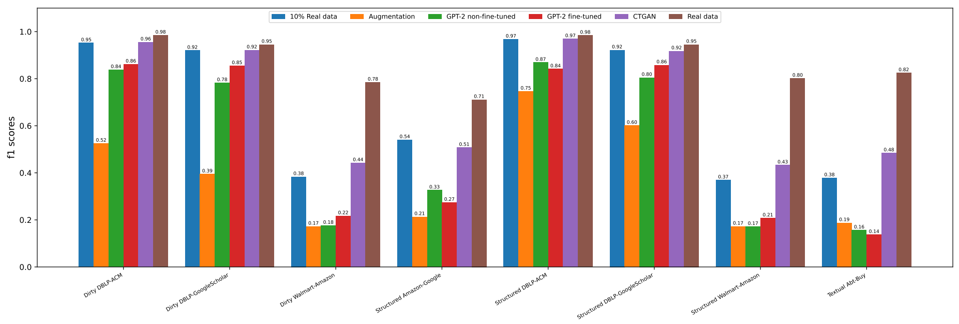


Figure 6.13: Real dataset supplemented with both artificial matches and non-matches which were generated using the 10% real dataset ran on Ditto

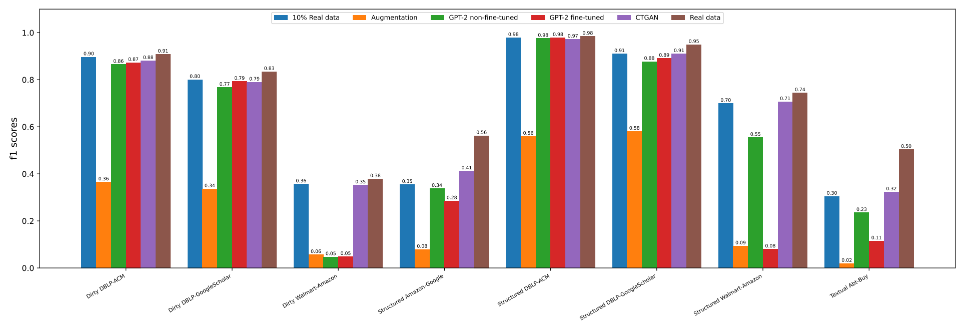


Figure 6.14: Real dataset supplemented with both artificial matches and non-matches which were generated using the 10% real dataset ran on Magellan

6.3 Replacing real data with artificial data

In the previous experiments, the data generated was used as supplementation to the real data. In this experiment we excluded the real data, and trained the Ditto and Magellan models only on the artificial datasets. The datasets created for this experiment were:

- Matches and non-matches generated out of the full real dataset.
- Matches and non-matches generated out of the 10% real dataset.

By training the Ditto and Magellan entity matching systems on these datasets, we could see whether or not artificial data can replace real data when training models [fig. 6.15]. As in the previous experiments, the entity matching systems were both evaluated using the full real testing and validation datasets as to not corrupt the results.

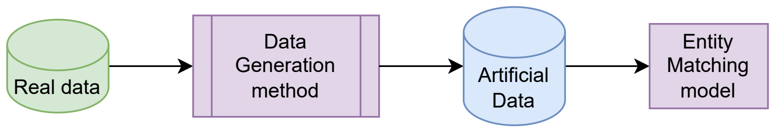


Figure 6.15: The pipeline for generating and testing the datasets in the third experiment.

The Ditto system was again trained 3 times and the results were averaged.

When only using the generated data which was created from the full real datasets, almost none of the methods performed well apart from a few exceptions, those being one on the Ditto system and a few on the Magellan system. The one which performed best on the Ditto system was the CTGAN method on the Dirty iTunes-Amazon dataset, where it even outperformed the real dataset [fig. 6.16]. While the Magellan system had no methods which outpreformed the real dataset, the non-fine-tuned GPT-2 method came close on three datasets, Structured DBLP-ACM, Structured Fodors-Zagats and Structured iTunes-Amazon. The augmentation method on the Magellan system also came close to the real dataset on the Structured iTunes-Amazon dataset [fig. 6.17].

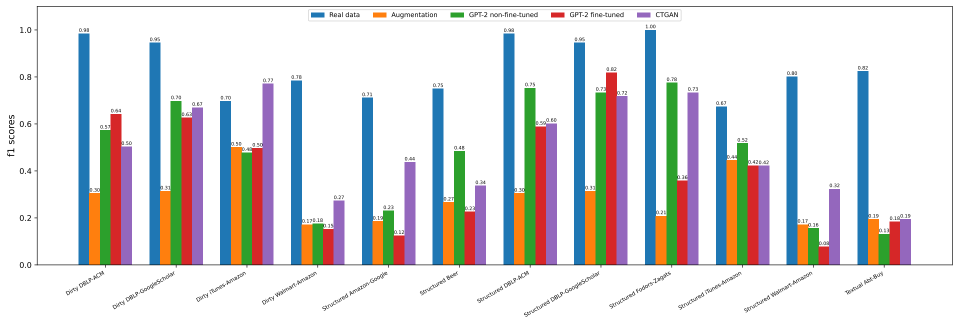


Figure 6.16: The data generated using the full real dataset ran on Ditto

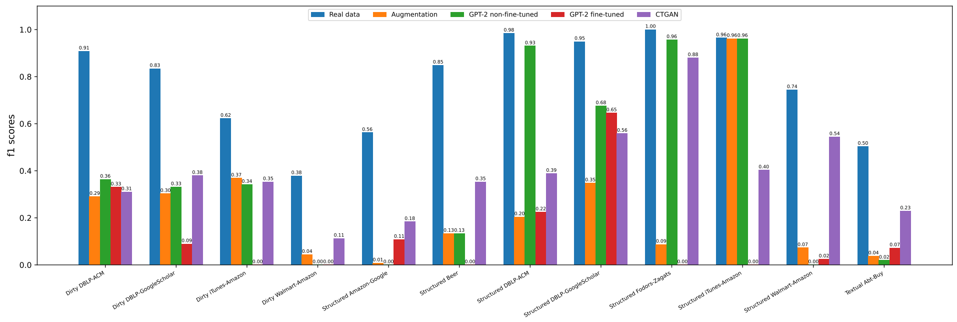


Figure 6.17: The data generated using the full real dataset ran on Magellan

When only using the generated data which was created from the 10% real datasets, none of the methods outperformed the 10% real datasets. However, CTGAN on the Ditto system came quite close to matching the 10% real data score on most of the datasets [fig. 6.18]. On the Magellan system however, only one dataset had methods that came anywhere close to the 10% real data f1 score, those being the non-fine-tuned GPT-2 and CTGAN methods on the Structured DBLP-ACM dataset [fig. 6.19].

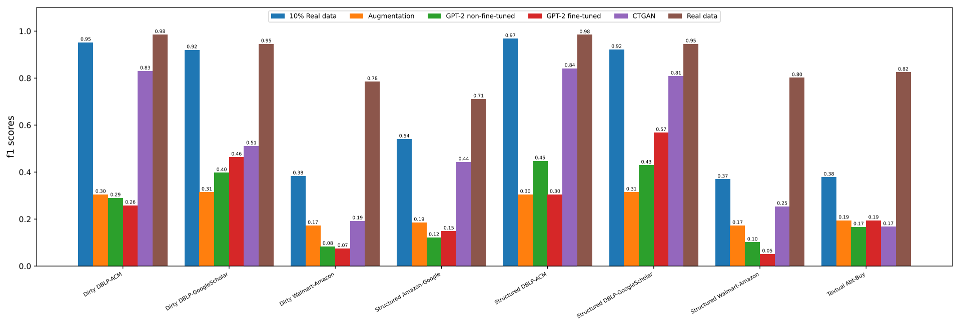


Figure 6.18: The data generated using the 10% dataset tested against both the 10%- and the full real datasets ran on Ditto

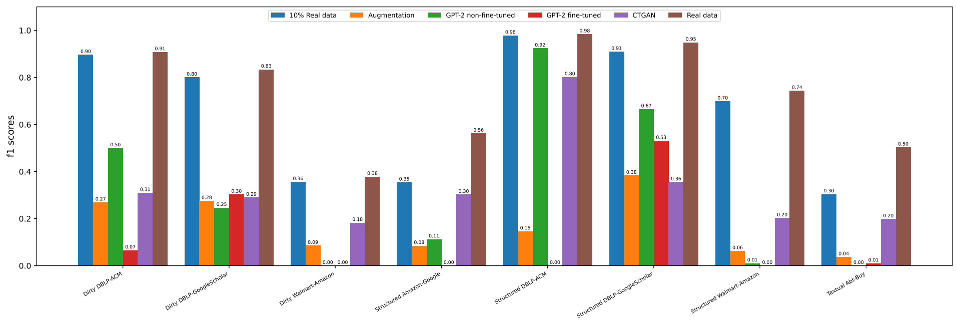


Figure 6.19: The data generated using the 10% dataset tested against both the 10%- and the full real datasets ran on Magellan

Chapter 7

Discussion

In this section we discuss the results of our experiments and try to understand them.

7.1 Rationalising the results

When analysing our experiment results which are described in 6 Experiments, we made 5 observations:

- The best f1 scores were achieved when real data was supplemented with artificial matches
- CTGAN seemed to be the best performing generation method throughout all tests
- Augmentation method seemed to be the worst performing generation method throughout all tests
- Fine-tuning the GPT-2 model seemed to generate data that performed either slightly better or much worse than the non-fine-tuned model.
- Using only the artificial data yielded bad results

In the following pages, we attempt at rationalising, explaining and understanding these observations.

7.1.1 Are artificial matches better?

When examining the results from the first and second experiments, a clear pattern emerged. All datasets across both entity matching systems and all generation methods achieved the best results when the real data was supplemented with the generated matches only. Does this mean that the artificial non-matches are unnecessary, or even detrimental in some way?

The results in these experiments do not support this claim. In fact, the results suggest that the artificial non-matches also have a positive effect on the f1 scores of the entity matching models, either matching or even exceeding the baseline measurement. We believe that this effect is related less to the quality of our generated data, and more to a quality of the original data itself.

The table below shows each test dataset, the number of matching and non-matching entries in the datasets and the ratio of non-matches to matches in each dataset [tab. 7.1]. The table clearly shows that non-matches are extremely over-represented in each dataset. When the real data is supplemented with non-matches, the ratio between the matches and non-matches in the supplemented dataset only becomes bigger, making it difficult for the model to learn the differences between matching and non-matching entries and effectively overfitting the model. When new matches are added, however, the ratio becomes more balanced, which would allow the models to differentiate between the different entries more effectively. When the data is supplemented with both new matches and non-matches, the ratio stays the same, but there is more information in the dataset, which again would allow the models to learn better. We believe that this explains the results we are getting.

Type	**Name**	**Matches**	**Non-matches**	**Ratio**
Structured	Amazon-Google	699	6175	8.8 : 1
	Beer	40	228	5.9 : 1
	DBLP-ACM	1332	6085	4.6 : 1
	DBLP-GoogleScholar	3207	14016	4.3 : 1
	Fodors-Zagats	66	501	7.6 : 1
	iTunes-Amazon	78	243	3.1 : 1
	Walmart-Amazon	676	5568	8.2 : 1
Textual	Abt-Buy	616	5127	8.3 : 1
Dirty	DBLP-ACM	1332	6085	4.6 : 1
	DBLP-GoogleScholar	3207	14016	4.3 : 1
	iTunes-Amazon	78	243	3.1 : 1
	Walmart-Amazon	576	5568	8.2 : 1

Table 7.1: The amount of matches and non-matches in the real training sets. Last column shows the ratio of non-matches to matches in each dataset.

Should we then only generate new matching entries and not bother with the non-matches? The results would imply this to be the case, but we believe this to be the wrong takeaway. Should the ratio reach 1:1, meaning that there is exactly one match to each non-match in the dataset, then supplementing only new matches would invert the ratio, and the same problem would occur. We think that supplementing the data with both matches and non-matches to keep the ratio as close to or at 1:1 would be the ideal course of action.

7.1.2 Is CTGAN the best method?

Throughout our experiments, the CTGAN method stands out as the best method on average. It usually outperforms or matches the f1 scores of the baseline measurement when supplementing real data, and is noticeably better than any other method when training the Entity Matching systems only on the generated data. Does this mean that it is the best method of generation?

CTGAN is designed to try keeping the data distribution of its generated entries as close to the distribution of the original data as possible. This has the effect of never over-representing the data that has a high frequency of appearance in the original data, which helps to avoid overfitting the Entity Matching systems when training them. The CTGAN generation method is described in more detail in 5.4 CTGAN.

The data, while representative of the original distribution, is not necessarily all correctly labeled. This is because when generating the data, some of the new attributes might make a match into a non-match and vice versa. To mitigate this problem, the generation method implements an insurance step which uses the Levenshtein similarity metric to discard generated data that does not hold the matching or non-matching quality. This insurance step is more active during the generation of matching entries, while remaining almost non-active during the generation of non-matches, keeping the data distribution modelled by CTGAN. When some of the data is discarded, new data needs to be created, and in the process, the data distribution might not match the real data as well any more. We pose however that the data distribution, due to the way CTGAN works, is still good enough that the most frequent entries don't get too over-represented.

These two qualities of the generation method together make the method perform really well in our experiments. This means that the CTGAN generation method is indeed the best method out of the ones we have tested. However, we do not believe that this method is generally the best method to use, due to it never generating completely new data in contrast to methods like the GPT-2 approach. With novel data in the datasets, the Entity Matching systems would acquire new information to learn from, but since the CTGAN method in a way reshuffles the attributes modelled from the real dataset, at a certain point the systems would overfit the matcher to the specific data that is in the training set.

7.1.3 Is Augmentation the worst method?

In all of our experiments, the Augmentation method had a tendency to perform the worst out of all of the generation methods. This was most apparent on Magellan, but the pattern is observable in both Entity Matching systems. Despite it's poor performance in most cases, the method did achieve similar scores to the other methods on select datasets. We believe that the reason for the Augmentation method's poor performance can be attributed to two factors which compound to lower the overall performance.

When generating non-matching entries, the method replaces some of the attributes in one entry with the attributes of another. The process is detailed in 5.1 Data augmentation method. The generation could be somewhat compared to how CTGAN, the method which performs the best in our tests, generates new entries, but with one fundamental difference. The Augmentation method has no regard for the data distribution within the dataset when generating new entries. As a result of this, the Augmentation method ends up inflating the amount of the most represented entries in the dataset, which can overfit the entity matching models, lowering their performance.

The approach is wholly different when generating new matching entries. The method introduces noise into the dataset by changing, or mutating, some characters in a randomly selected entry. The process is detailed in 5.1 Data augmentation method. Depending on how different the new entries may be, little or no new information might be added to the dataset. This means that the model might not get any new information to learn what a matching entry is, and can in fact become overfit on the matching entries that are in the new combined dataset, which again lowers the method performance.

Is the Augmentation method the worst then? The results don't seem to suggest it on every experiment we ran. When supplementing real data with only matches, the method achieved f1 scores that were comparable to the other methods, and in some cases the method outperformed the baseline measurements. This means that despite the flaws of the method, it still has circumstances in which it can achieve good results. It is however on average the worst performing method out of the ones we have tested.

7.1.4 Is it worth to fine-tuning the GPT-2 model?

The results of our experiments show that when using the fine-tuned GPT-2 model, the performance is either improved slightly, or not at all. In some cases, the fine-tuned generation method even performs worse than the non-fine-tuned model. Does this mean that fine-tuning the GPT-2 model is unnecessary? We believe this not to be the case.

When generating new entries with the non-fine-tuned GPT-2 method, a prompt containing context needs to be provided. The more context the GPT-2 model gets, the better data it will generate. However, as mentioned in 5.5.3 Data generation using GPT-2, the GPT-2 model has a maximum amount length of the prompt which cannot be exceeded. This can become a problem when the data to be generated has many attributes or very long strings. The prompt has to also leave space for the generated text to appear in, meaning that the entire length of the prompt can't be used for the context alone. This issue is alleviated by fine-tuning the GPT-2 model, as the context becomes baked into the model itself.

When looking at the generated data, we see that the fine-tuned GPT-2 model can sometimes falsely generate matching entries when they are supposed to be non-matching, and vice versa. As the method was designed to rely on the deep learning model to produce the entries, there are no mechanisms in place to ensure the matching or non-matching status of the generated entries, such as in the CTGAN method. We believe this does not happen as often in the non-fine-tuned GPT-2 method because the length of the entries is short enough in the available datasets, which allows for sufficient context to be provided within the prompts.

We believe that with a more refined method, the strengths of the language model can be better utilised, making the fine-tuned generation method perform better than the non-fine-tuned one. The current implementation of the methods is however a bit underwhelming.

7.1.5 Why the use of only artificial data lead to poor performance?

When training the Entity Matching systems on artificial data only, the results were subpar. None of the methods managed to exceed or even match the f1 scores of the real data, apart from one instance on the Magellan system, where the Augmentation- and the non-fine-tuned GPT-2 methods achieved almost identical f1 scores to the real data. These good results don't seem to be reflected in the Ditto Entity Matching system. A theory we have is that the generated entities in matching entries are very similar if not identical, and in the non-matching entries are completely different. The Magellan system seems to react well to such data when trying to differentiate matches from non-matches. This is however just a theory, and because of time constraints, we could not investigate this further.

We believe that the main reason for such poor performance is the aforementioned data distribution. When generating new entries, the CTGAN preserves the data distribution better than any other method, which is especially clearly reflected in the results for when testing data generated from 10% of the real data on the Ditto system. The Augmentation method totally disregards the data distribution when generating new entries, which is reflected in the results as well. As for the GPT-2 methods, when generating a new entry, they use a randomly selected entry from the dataset as the seed for generation. This process is explained in more detail in 5.2 Standard GPT-2 and 5.3 Fine tuning GPT-2. Doing it this way also disregards the distribution of the original dataset, as the most common attributes will be represented more. However, we believe that the new information generated by the language model allows the Entity Matching systems to learn more about how to match entries, which results in better results than the Augmentation method.

7.1.6 Time investment to generate data

When running the experiments, we wanted to also measure the time it took to generate the artificial data. The effectiveness of a generation method would mean nothing in practice if it takes too long to generate the data needed to improve the Entity Matching system performance. Unfortunately, due to some technical difficulties, we were unable to measure the time it takes to generate data exactly. We do have an idea on the amount of time it takes to generate the data, so some observations can still be made.

In essence, the Augmentation method only manipulates strings, meaning that no complex calculations are done while generating data. This makes the method really fast. Also, because there isn't any post-processing of the generated entries and no retries of generation, the time used to generate a dataset scales linearly with the amount of data needed to generate. We usually ran all generation methods in bulk operations, and the entirety of data generated for all experiments we ran took about 10 seconds to generate.

Comparatively, the GPT-2 methods are both quite slow. The generation of a single entry usually takes around 5-10 seconds, but as mentioned in 5.5.3 Data generation using GPT-2, the GPT-2 model might produce unwanted text in the string. Each generated entry must therefore be validated and if invalid, must be discarded and generated again. This means that in worst case, the time can scale exponentially with the amount of data needed to be generated. Interestingly, GPT-2 fine-tuning needs to be brief by design, because otherwise the GPT-2 model can become overfit, so the training time is negligible. On the biggest datasets, it took both GPT-2 methods between 36 and 48 hours to generate data.

The CTGAN method is also slow compared to the Augmentation method, but it is in theory a lot faster than the GPT-2 method. With the usage of mode-specific normalization and batch normalization, the training procedure on larger textual datasets can become shorter. The training time ranges from 30 minutes on small structured discrete-like datasets, to 24 hours on large textual-based datasets. An interesting observation is the generation time in relation to amount of data trained on. On small datasets, with the use of the insurance method, which is described in 5.5.6 Ensure data method, generation time can reach up to 48 hours in relation to sufficient data attributes in accordance to matching or non-matching entries. However with larger datasets, the model has a larger attribute corpora to sample from and as such generates sufficient datasets in less than 2 hours.

Whether or not the time used by these generation methods is an important factor or not is highly domain specific, and should be evaluated in each situation. However, we do not think that the time usage discussed above is unreasonable, and therefore argue that these methods could prove viable in a real-world setting.

7.1.7 Answering the research questions

In the Introduction section, we posed three research questions, which we attempted to find answers to through our experiments. Throughout this section we also discussed several points which all contribute to the answers. For the readers convenience, the questions will be repeated bellow before we provide the answers we came up with.

Research question 1: What is the state of the art in the fields of EM and AD?

Through our research, we found several different methods we could implement or use in our experiments. By using a collection of these methods, we hoped to discover patterns in the results, which would indicate that the results we achieve are repeatable and reliable.

The state of the art Entity Matching systems we ended up using are, as previously mentioned, Ditto and Magellan. Ditto is a system which utilises Deep Learning to perform Entity Matching. Magellan is a traditional method, relying on several different artificial intelligence methods such as, naïve Bayes, logistic regression, linear regression, support vector machines, decision trees, random forest, and xg-boost matcher. By using both a traditional- and a deep learning method, we wanted to ensure that the results were not affected by some unknown variable, such as for example a quirk in one of the methods.

The Artificial Data field has many different methods of generating artificial data, but these are usually made for some specific purpose, and were not suitable to reliably generate textual data. While researching, we ended up finding out about language models and generative adversarial networks being used to generate good quality textual data, and decided to use them in our thesis. As previously mentioned, we ended up using the GPT-2 and CTGAN models to generate our data.

Research question 2: How does supplementing real data with AD affect the performance of EM models?

As discussed prior, when supplementing real data, the best results were achieved when the artificial data only consisted of matching entries. We argue however that these results are circumstantial, as the real data has a very skewed ratio of matching to non-matching entries, which we believe affects the results. When taking a step back and taking into account what we discussed in the prior sections, the results do seem to point towards the artificial data improving the model performance when supplemented to the real training data.

Research question 3: How does training EM models solely on AD affect their performance?

In our discussion about artificial data performing badly in prior sections, we mention data distribution being a large factor that may contribute to these poor results. The results achieved by the CTGAN method, which by design tries to model the data distribution of the original dataset, while subpar, are indeed promising. We argue therefore that generating data with more refined methods might allow for the replacement of real training data, if that is a necessity of the task.

Chapter 8

Conclusion

In this section we present our conclusions which are based on the results of our experiments and the discussion done in the section 7 Discussion.

In supplementation scenarios, our methods either do indeed improve the Entity Matching systems or match their f1 scores. However, generating artificial data that can replace real data when training the models has not been achieved in this thesis.

Even though CTGAN performs well with its ability to capture the data distribution of the original datasets in correspondence to their matching label, GPT-2's ability to introduce data fidelity makes it perform as well if not even better in certain scenarios. With this in mind, a model which manages to deal with the obstacles of class imbalance in discrete columns and non-Gaussian distributions in continuous columns, with the added benefit of data variance under certain conditions could become a powerful tool in regards generation of artificial data.

While the improvements in our results are slight, we argue that with more refined methods and further research, bigger improvements can be made. As of the time of writing not a lot of research has been conducted in this field, and so with our results we open the doors to other researchers. We argue that our research can help pave the way to less data-hungry models, primarily in the field of Entity Matching, but also in other fields where data is a concern.

Chapter 9

Further work

In this section, we present some ideas for further work following this thesis.

9.1 More datasets

As mentioned in 4.3 Other datasets, Ditto evaluated on 6 other datasets. These datasets all resemble the Textual category of datasets, meaning that they have few attributes with long strings of text as their values. Given that only one Textual dataset is tested in this project, testing these other datasets would provide more insight into the generation methods' performance on data of this type.

9.2 Mixing generation methods

Our experiment results showed that the different methods performed differently when real data was supplemented with only the artificial matches as opposed to the artificial non-matches, with the same method of data generation rarely performing best in both cases.

We pose that the entity matching model performance could potentially be improved by supplementing real data with matches and non-matches generated by different methods.

9.3 Implementing an insurance step

A big contributor into why the CTGAN method was on average the best throughout our experiments was the insurance step which attempted to ensure the matching and non-matching qualities of the generated entries. By implementing such a step into all other generation methods, we believe that the performance of those methods can be increased.

9.4 Ensuring the matching status of entries

The CTGAN generation method implemented a simple insurance using the Levenstein similarity metric to try and keep the mathcing and non-mathcing status of the generated entries. As seen in the results, this greatly increased the performance of the generation method. Such a metric should be implemented for the other methods in the future as well, as it could greatly increase the performance of those methods.

9.5 Attribute approach

When generating new entries using the GPT-2 language model, we supplied the model with a prompt containing an entity and made the model fill in the whole missing entity. While effective, as shown by our results, we believe this method could be improved by attempting a different approach of generating the new entity.

Instead of using the entire entity as a prompt, one could use one of the attributes of the entry, and have the model produce a similar attribute. This can be done repeatedly with each attribute of the entity until a new entity is constructed. We believe that this approach could potentially achieve better results by taking better advantage of the knowledge base that the language model has.

9.6 Other language models

While GPT-2 is a state of the art language model, as briefly mentioned in 2.3.2 GPT-2, there are other state of the art language models. In the future, our methods should be implemented using these other language models such as BERT or ELMo to compare them to the GPT-2 model.

9.7 GPT-3

The GPT-3 model is reportedly more powerful than it's predecessor GPT-2 [72]. Because of this, we believe that our methods could achieve better performance if they would be implemented using the GPT-3 model. Currently, the OpenAI GPT-3 model is only available as an API web service.

9.8 Variational Autoencoder approach

The Ditto model uses an interesting augmentation method with the use of the BERT language model, as mentioned in 3.1 Ditto data augmentation using BERT. This method is somewhat similar to how a Variational Autoencoder works, where the language model is used to encode the augmented and true entry and then interpolate between the two.

We propose a system which uses a language model to encode an entry, and then in the resulting latent space, slightly perturb the entry, which should result in a slightly different, but similar entry when decoded. Such a method has been most famously used in generating new faces by traversing the latent space which is generated by the network [73]. We believe that such a method for data generation would be an interesting application of the Variational Autoencoder and could potentially produce good results.

9.9 Introducing a classifier

In the TableGAN paper[64], Park et al. utilize a classifier to keep semantic integrity when generating artificial tabular data for classification problems. While it is unclear if this addition would be able to tackle generating both matching and non-matching data in usage for Entity Matching tasks, it would be an interesting feature to research. The task of matching entities is a multi-layered problem consisting of knowing when attributes are alike, while being contextually aware of other attributes in each entity, how attributes relate to each other, what should be considered similar, and what attributes should be put weight on in relation to the domain which the datasets comes from.

Couple this with the task of modelling data distributions efficiently and generating new data with high fidelity, a more focused approach might be necessary.

Bibliography

[1] N. C. Thompson, K. Greenewald, K. Lee, and G. F. Manso, “The Computational Limits of Deep Learning,” *arXiv:2007.05558 [cs, stat]*, July 2020. arXiv: 2007.05558.

[2] OpenAI, “AI and Compute,” May 2018.

[3] K. Jebari, P. Strimling, and I. Vartanova, “AI winter is coming?,” Mar. 2021.

[4] T. R. MIT, “We analyzed 16,625 papers to figure out where AI is headed next.”

[5] S. I. Nikolenko, *Synthetic Data for Deep Learning*, vol. 174 of *Springer Optimization and Its Applications*. Cham: Springer International Publishing, 2021.

[6] A. Halevy, P. Norvig, and F. Pereira, “The Unreasonable Effectiveness of Data,” *IEEE Intelligent Systems*, vol. 24, pp. 8–12, Mar. 2009.

[7] H. L. Dunn, “Record Linkage,” *American Journal of Public Health and the Nations Health*, vol. 36, pp. 1412–1416, Dec. 1946.

[8] H. B. Newcombe, J. M. Kennedy, S. J. Axford, and A. P. James, “Automatic Linkage of Vital Records,” *Science*, vol. 130, no. 3381, pp. 954–959, 1959. Publisher: American Association for the Advancement of Science.

[9] I. P. Fellegi and A. B. Sunter, “A Theory for Record Linkage,” *Journal of the American Statistical Association*, vol. 64, pp. 1183–1210, Dec. 1969. Publisher: Taylor & Francis _eprint: https://www.tandfonline.com/doi/pdf/10.1080/01621459.1969.10501049.

[10] H. Köpcke and E. Rahm, “Frameworks for entity matching: A comparison,” *Data & Knowledge Engineering*, vol. 69, pp. 197–210, Feb. 2010.

[11] P. Christen, M. Hegland, S. Roberts, O. Nielsen, T. Churches, K. Lim, and S. Branch, “Parallel Computing Techniques for High-Performance Probabilistic Record Linkage,” *ResearchGate*, Apr. 2002.

[12] W. E. Winkler, “Methods for Record Linkage and Bayesian Networks,” *Census Bureau*, p. 29, Apr. 2002.

[13] D. R. Wilson, “Beyond probabilistic record linkage: Using neural networks and complex features to improve genealogical record linkage,” in *The 2011 International Joint Conference on Neural Networks*, pp. 9–14, July 2011. ISSN: 2161-4407.

[14] K. Abou-Moustafa, “What Is the Distance Between Objects in a Data Set? – EMBS,” 2016.

[15] J. Prendki, “Are you spending too much money labeling data?,” Mar. 2020.

[16] A. Adadi, “A survey on data-efficient algorithms in big data era,” *Journal of Big Data*, vol. 8, p. 24, Jan. 2021.

[17] P. Christen, “Introduction,” in *Data Matching: Concepts and Techniques for Record Linkage, Entity Resolution, and Duplicate Detection* (P. Christen, ed.), Data-Centric Systems and Applications, pp. 3–22, Berlin, Heidelberg: Springer, 2012.

[18] N. Barlaug and J. A. Gulla, “Neural Networks for Entity Matching: A Survey,” *arXiv:2010.11075 [cs]*, May 2021. arXiv: 2010.11075.

[19] A. Doan, A. Halevy, and Z. Ives, *Principles of Data Integration - 1st Edition*. Morgan Kaufmann, 1st ed., June 2012.

[20] P. Christen, “Data Matching Systems,” in *Data Matching: Concepts and Techniques for Record Linkage, Entity Resolution, and Duplicate Detection* (P. Christen, ed.), Data-Centric Systems and Applications, pp. 229–242, Berlin, Heidelberg: Springer, 2012.

[21] F. Azzalini, S. Jin, M. Renzi, and L. Tanca, “Blocking Techniques for Entity Linkage: A Semantics-Based Approach,” *Data Science and Engineering*, vol. 6, pp. 20–38, Mar. 2021.

[22] R. Baxter, P. Christen, and T. Churches, “A Comparison of Fast Blocking Methods for Record Linkage,” *KDD*, p. 6, 2003.

[23] M. A. Hernández and S. J. Stolfo, “The merge/purge problem for large databases,” *ACM SIGMOD Record*, vol. 24, pp. 127–138, May 1995.

[24] P. Konda, S. Das, P. Suganthan G. C., A. Doan, A. Ardalan, J. R. Ballard, H. Li, F. Panahi, H. Zhang, J. Naughton, S. Prasad, G. Krishnan, R. Deep, and V. Raghavendra, “Magellan: toward building entity matching management systems,” *Proceedings of the VLDB Endowment*, vol. 9, pp. 1197–1208, Aug. 2016.

[25] Y. Li, J. Li, Y. Suhara, A. Doan, and W.-C. Tan, "Deep Entity Matching with Pre-Trained Language Models," *Proceedings of the VLDB Endowment*, vol. 14, pp. 50–60, Sept. 2020. arXiv: 2004.00584.

[26] Dan Jurafsky and James H. Martin, "Speech and Language Processing."

[27] C. E. Shannon, "A Mathematical Theory of Communication," *The Bell System Technical Journal*, p. 55, Oct. 1948.

[28] John J. Hopfield, "Neural networks and physical systems with emergent collective computational abilities.," Apr. 1982.

[29] J. L. McClelland, D. E. Rumelhart, and G. E. Hinton, "Parallel Distributed Processing," *Stanford*, p. 42, 1986.

[30] D. E. Rumelhart, G. E. Hintont, and R. J. Williams, "Learning representations by back-propagating errors," *Nature*, p. 4, 1986.

[31] Sepp Hochreiter and Jürgen Schmidhuber, "Long Short-Term Memory," 1997.

[32] D. Bahdanau, K. Cho, and Y. Bengio, "Neural Machine Translation by Jointly Learning to Align and Translate," Tech. Rep. arXiv:1409.0473, arXiv, 2015. arXiv:1409.0473 [cs, stat] type: article.

[33] Alfredo Canziani, "Week 6 – Lecture: CNN applications, RNN, and attention," Apr. 2020.

[34] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin, "Attention Is All You Need," Tech. Rep. arXiv:1706.03762, arXiv, Dec. 2017. arXiv:1706.03762 [cs] type: article.

[35] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding," Tech. Rep. arXiv:1810.04805, arXiv, May 2019. arXiv:1810.04805 [cs] type: article.

[36] M. E. Peters, M. Neumann, M. Iyyer, M. Gardner, C. Clark, K. Lee, and L. Zettlemoyer, "Deep contextualized word representations," Tech. Rep. arXiv:1802.05365, arXiv, Mar. 2018. arXiv:1802.05365 [cs] version: 2 type: article.

[37] OpenAI, "Better Language Models and Their Implications," Feb. 2019.

[38] A. Radford, J. Wu, R. Child, D. Luan, D. Amodei, and I. Sutskever, "Language Models are Unsupervised Multitask Learners," *OpenAI*, p. 24, Feb. 2019.

[39] A. Radford, K. Narasimhan, T. Salimans, and I. Sutskever, "Improving Language Understanding by Generative Pre-Training," *OpenAI*, p. 12, June 2018.

[40] B. S. Everitt and A. Skrondal, *The Cambridge Dictionary of Statistics*. UK: Cambridge University Press, 3rd ed., 2006.

[41] H. Fischer, *A History of the Central Limit Theorem*. New York, NY: Springer New York, 2011.

[42] "Univariate distribution," Feb. 2021. Page Version ID: 1006262861.

[43] N. L. Johnson, A. W. Kemp, and S. Kotz, *Univariate Discrete Distributions*. Jogn Wiley & Sons, 3rd ed., 2005.

[44] "Joint probability distribution," June 2022. Page Version ID: 1094503297.

[45] A. Radford, L. Metz, and S. Chintala, "Unsupervised Representation Learning with Deep Convolutional Generative Adversarial Networks," *arXiv*, Nov. 2015.

[46] J. Bao, D. Chen, F. Wen, H. Li, and G. Hua, "CVAE-GAN: Fine-Grained Image Generation Through Asymmetric Training," in *CVAE-GAN: Fine-Grained Image Generation Through Asymmetric Training*, pp. 2745–2754, 2017.

[47] B. Zhang, S. Gu, B. Zhang, J. Bao, D. Chen, F. Wen, Y. Wang, and B. Guo, "StyleSwin: Transformer-Based GAN for High-Resolution Image Generation," in *StyleSwin: Transformer-Based GAN for High-Resolution Image Generation*, pp. 11304–11314, 2022.

[48] C. Vondrick, H. Pirsiavash, and A. Torralba, "Generating Videos with Scene Dynamics," *arXiv*, Sept. 2016.

[49] T.-C. Wang, M.-Y. Liu, J.-Y. Zhu, G. Liu, A. Tao, J. Kautz, and B. Catanzaro, "Video-to-Video Synthesis," *arXiv*, Aug. 2018.

[50] A. Clark, J. Donahue, and K. Simonyan, "Adversarial Video Generation on Complex Datasets," *arXiv*, July 2019.

[51] C. Donahue, J. McAuley, and M. Puckette, "Adversarial Audio Synthesis," *arXiv*, Feb. 2018.

[52] J. Engel, K. K. Agrawal, S. Chen, I. Gulrajani, C. Donahue, and A. Roberts, "GANSynth: Adversarial Neural Audio Synthesis," *arXiv*, Feb. 2019.

[53] S. J. Russell, P. Norvig, and E. Davis, *Artificial intelligence: a modern approach*. Prentice Hall series in artificial intelligence, Upper Saddle River: Prentice Hall, 3rd ed ed., 2010.

[54] L. Huang, A. D. Joseph, B. Nelson, B. I. Rubinstein, and J. D. Tygar, "Adversarial machine learning," in *Proceedings of the 4th ACM workshop on Security and artificial intelligence*, AISec '11, (New York, NY, USA), pp. 43–58, Association for Computing Machinery, Oct. 2011.

[55] R. Sutton and A. Barto, “Reinforcement Learning: An Introduction,” *IEEE Transactions on Neural Networks*, vol. 9, pp. 1054–1054, Sept. 1998. Conference Name: IEEE Transactions on Neural Networks.

[56] C. E. Shannon, “XXII. Programming a computer for playing chess,” *The London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science*, vol. 41, pp. 256–275, Mar. 1950. Publisher: Taylor & Francis _eprint: https://doi.org/10.1080/14786445008521796.

[57] T. Mitchell, “GENERATIVE AND DISCRIMINATIVE CLASSIFIERS: NAIVE BAYES AND LOGISTIC REGRESSION Machine Learning,” 2005.

[58] I. J. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio, “Generative Adversarial Networks,” *arXiv*, June 2014.

[59] “Conditional Generative Adversarial Nets,” Nov. 2014. arXiv:1411.1784 [cs, stat].

[60] L. Xu, M. Skoularidou, A. Cuesta-Infante, and K. Veeramachaneni, “Modeling Tabular data using Conditional GAN,” *aarXiv*, July 2019.

[61] I. Gulrajani, F. Ahmed, M. Arjovsky, V. Dumoulin, and A. Courville, “Improved Training of Wasserstein GANs,” *arXiv*, Mar. 2017.

[62] Z. Lin, A. Khetan, G. Fanti, and S. Oh, “PacGAN: The power of two samples in generative adversarial networks,” *arXiv*, Dec. 2017.

[63] L. Xu and K. Veeramachaneni, “Synthesizing Tabular Data using Generative Adversarial Networks,” *arXiv*, Nov. 2018.

[64] N. Park, M. Mohammadi, K. Gorde, S. Jajodia, H. Park, and Y. Kim, “Data Synthesis based on Generative Adversarial Networks,” *arXiv*, June 2018.

[65] S. Mudgal, H. Li, T. Rekatsinas, A. Doan, Y. Park, G. Krishnan, R. Deep, E. Arcaute, and V. Raghavendra, “Deep Learning for Entity Matching: A Design Space Exploration,” in *Proceedings of the 2018 International Conference on Management of Data*, (Houston TX USA), pp. 19–34, ACM, May 2018.

[66] L. Barbosa, “Learning representations of Web entities for entity resolution,” *International Journal of Web Information Systems*, vol. 15, pp. 346–358, Aug. 2019.

[67] C. Shorten and T. M. Khoshgoftaar, “A survey on Image Data Augmentation for Deep Learning,” *Journal of Big Data*, vol. 6, p. 60, July 2019.

[68] A. E. Eiben and J. E. Smith, “Representation, Mutation, and Recombination,” in *Introduction to Evolutionary Computing* (A. Eiben and J. Smith, eds.), Natural Computing Series, pp. 49–78, Berlin, Heidelberg: Springer, 2015.

[69] “Slurm Workload Manager - Documentation.” https://slurm.schedmd.com/documentation.html.

[70] “Overview — SDV 0.15.0 documentation.”

[71] “CTGAN Model — SDV 0.15.0 documentation.”

[72] K. Vu, “GPT-2 (GPT2) vs. GPT-3 (GPT3): The OpenAI Showdown - DZone AI,” May 2022.

[73] D. Kamath, “Generating new faces with Variational Autoencoders,” Jan. 2021.

Appendix A

Source code

The full source code can be found here:

`https://github.com/upforde/Idun`

For ease of access and better reference, we will refer to specific files and directories in the github repository.

A.1 Ditto results

The graphs generated out of the results from the Ditto Entity Matching system:

`https://github.com/upforde/Idun/tree/main/ditto/Writer_output/Xfigures`

A.2 Magellan results

The graphs generated out of the results from the Magellan Entity Matching system:

`https://github.com/upforde/Idun/tree/main/CTGAN/plots`

A.3 Ditto data parser

The ditto data parser source code:

`https://github.com/upforde/Idun/blob/main/GPT-2/ditto_parser.py`

A.4 Auxiliary data parser

The Auxiliary data parser, which changes Ditto data in to Magellan compatible csv files and vice-versa:

`https://github.com/upforde/Idun/blob/main/CTGAN/parse_data.py`

A.5 CTGAN data generation generation source code

The source code for the CTGAN generation process:
https://github.com/upforde/Idun/blob/main/CTGAN/CTGAN_generation.py

NTNU
Kunnskap for en bedre verden