Master's thesis

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

Mikkel Nygard
Øyvind Samuelsen

Active Learning with Transformer Pre-trained Language Models for Entity Matching

Master's thesis in Computer Science
Supervisor: Jon Atle Gulla
Co-supervisor: Nils Barlaug

June 2021

NTNU
Kunnskap for en bedre verden

Mikkel Nygard
Øyvind Samuelsen

Active Learning with Transformer Pre-trained Language Models for Entity Matching

Master's thesis in Computer Science
Supervisor: Jon Atle Gulla
Co-supervisor: Nils Barlaug
June 2021

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

NTNU

Kunnskap for en bedre verden

Abstract

Entity matching refers to the problem of finding which records refer to the same real-world entity. Recently, thanks to the rise of Transformer pre-trained language models (TPLMs), the field of entity matching has seen new development and state-of-the-art solutions. However, the need for a significant amount of training data remains a challenge. Active learning is a machine learning methodology seeking to minimize the required labeling effort while maintaining the quality of the model. This thesis explores how combining active learning with TPLMs performs for entity matching. Several active learning query strategies have been compared against a baseline of random sampling, including uncertainty sampling and partition based methods. The experiments have been performed on public entity matching datasets, concerning consumer product data and citation data.

We found all active learning strategies consistently outperformed the non-active learning baseline, with an iteration time of 2.5-8 minutes. Even when the baseline was trained on all available data, several query strategies surpassed its F1-score with an order of magnitude fewer labeled examples. For the best performing strategy, Partition-2, this happened on average after only 9.1% of the total training data was queried. Hybrid-Partition-2 is a novel active learning technique, which combines the speed of classical machine learning models, and performance of TPLMs. We found that this technique resulted in an often significantly higher initial recall. This resulted in a 0.011-0.345 higher initial F1-score across 5 datasets. The method does however require extra overhead with the fact that two separate iterations of active learning need to be run.

In the end, we have recommended further work in the area of developing more novel active learning query strategies specifically made for entity matching with TPLMs, in addition to a benchmark framework for selecting appropriate hyperparameters when performing this task.

Sammendrag

Entitetsmatching refererer til problemet med å finne ut hvilke data som refererer til den samme virkelige entiteten. Nylig, takket være fremveksten av ferdigtrente Transformer språkmodeller (TPLMs), har entitetsmatching sett ny utvikling og moderne løsninger. Imidlertidig er behovet for en betydelig mengde treningsdata fremdeles en utfordring. Aktiv læring er en maskinlæringsmetodikk som har som mål å minimere den nødvendige mengden med treningsdata, samtidig som kvaliteten på modellen opprettholdes. Denne oppgaven utforsker hvordan aktiv læring fungerer sammen med TPLMs for entitetsmatching. Flere aktiv læringsstrategier har vært sammenlignet mot en basisstrategi av tilfeldig utplukk, inkludert usikkerhets-utplukk og partisjonsmetoder. Eksperimentene har vært utført på offentlig tilgjengelig entitetsmatching-datasett, bestående av forbrukerproduktsdata og siteringsdata.

Vi fant at alle aktiv læringsstrategiene konsekvent gjorde det bedre enn den ikke-aktiv læring basisstrategien, med en iterasjonstid på 2,5-8 minutter. Selv når basisstrategien ble trent på all tilgjengelig data, fikk flere aktiv læringsstrategier høyere F1-poeng med en størrelsesorden færre treningsdata. For den beste spørre-strategien, Partition-2, skjedde dette etter gjennomsnittlig bare 9,1% av all treningsdata var spurt. Hybrid-Partition-2 er en ny aktiv læringsstrategi som kombinerer hastigheten til klassiske maskinlæringsmodeller og ytelsen til TPLMs. Vi fant ut at denne teknikken resulterte i en ofte betydelig høyere initiell tilbakekalling. Dette resulterte i 0,011-0,345 høyere initiell F1-poeng på de 5 datasettene. Denne metoden krever imidlertidig ekstra arbeid fra det faktum at den må kjøre to separater iterasjoner med aktive læring.

Til slutt har vi anbefalt videre utvikling av nye aktiv læringsstrategier spesielt laget for entitetsmatching med TPLMs, i tillegg til et testrammeverk for å velge hyperparametere når man utfører denne oppgaven.

Preface

The thesis has been submitted as a master's thesis at the Norwegian University of Science and Technology (NTNU), Department of Computer Science (IDI). It has been part of a collaboration project between NTNU, Cognite, and The Norwegian Open Artificial Intelligence Lab.

We would especially like to thank our supervisor Nils Barlaug for continuous support, motivation, and valuable feedback all throughout the project. In addition we thank professor Jon Atle Gulla for facilitation and supervision of the project, along with valuable feedback.

We would also like to thank friends and family for support in our studies.

Mikkel Nygard, Øyvind Samuelsen

Trondheim, June 13, 2021

Table of Contents

List of Figures

List of Tables

Abbreviations

AL	=	Active Learning
DL	=	Deep Learning
EM	=	Entity Matching
ML	=	Machine Learning
RF	=	Random Forest
TPLM	=	Transformer Pre-trained Language Model

Chapter 1

Introduction

This chapter presents the motivation and goals of the thesis, our approach and main results, and an outline for the rest of the thesis.

1.1 Motivation

The world is becoming more and more digitalized. With new advances in technology, companies are starting to realize the potential of harvesting previously unused data and information. A production company may be interested in predicting when any of their machines are going to fail, so they can schedule and optimize the downtime. Similarly, an oil company could have all sensors and equipment on an oil platform inside a digital platform. When any equipment breaks down, they could e.g. visualize the location in a 3d-platform and pass it to the engineers who are going to fix the equipment.

The different use-cases for digitalization are broad. Common for all are that raw data, often from a wide variety of different sources, need to be combined and transformed. A challenge when integrating all this diverse data is the fact that they might come from systems which do not share a common data format, such as names and other identifiers. An example is a pump at the oil platform may store pressure data in one system, while the temperature is part of a different system. A problem arises if the two systems do not refer to a specific pump with the same unique identifier. If the company in the future want to store all data in one place so they can do e.g. data mining to predict the optimal pump parameters, they have to *manually* find and combine all pressure and temperature sensor data that belong to the same pump. With potentially thousands of sensors and equipment, this can be an extremely time consuming, but required, data integration job.

Entity matching (EM) refers to the problem of finding which records refer to the same real-world entity. In Table 1.1, determining which of the four possible combinations refer to the same entity is hard as there are no obvious unique keys, and all relevant information is merged into the same string. The problem would not be present if the tables had an attribute column named "key", with unique identifiers which were shared between matching records. In the example, "sp-320", "ep-320" and could be such identifiers, however these do not exist for all records in the tables. Furthermore, two records can share these strings without being matches. In the example, only the record A1 and B2 refer to the same real world entity, and can be labeled as a match.

Table 1.1: An example of an entity matching problem from the public dataset "Abt-Buy". Only A2 and B2 refer to the same entity, likely a gas grill.

Dataset A: Abt

Id	Name
A1	weber spirit sp-320 stainless steel liquid propane gas outdoor grill 3730001
A2	weber stainless steel genesis s320 lp grill 3780001

Dataset B: Buy

Id	Name
B1	weber genesis ep-320 blue lp gas grill
B2	weber summit sp-320 stainless lp gas grill

Matches

Id	Matching
A1-B1	False
A1-B2	True
A2-B1	False
A2-B2	False

Traditionally, EM has been tackled with the use of probabilistic methods, and approximate string similarity measures combined with classic machine learning models (e.g. Random Forest), and lately deep learning methods [1]. With the rise of Transformer pre-trained language models (TPLMs) such as BERT, EM has seen recent advances in the last couple of years and new state-of-the-art solutions [2, 3, 4]. However, TPLMs still require a large amount of training data in order to perform at its best.

Active learning (AL) is an iterative and interactive machine learning strategy used to tackle this problem. In AL, the machine learning model itself tries to find only the most *informative* unlabeled examples, and query an Oracle, a human domain expert, to label them. The idea is by only training on informative examples, the model can train on fewer examples while still performing at an optimal level.

Combining EM with TPLMs and AL have the potential to make practical applications

of EM more attractive for companies, by reducing the required time-use significantly. This will only become more important in the future as companies are digitalizing their business.

1.2 Goals and Research Questions

Goal *Evaluate active learning with Transformer pre-trained language models for entity matching.*

The goal of this thesis is to evaluate using active learning with Transformer pre-trained language models as a strategy for doing entity matching, in order to improve the F1-score while having a low labeling cost. In order to reach this goal, we have defined three main research questions that will be answered.

Research question 1 *What active learning query strategy for Transformer pre-trained language models results in the highest F1-score?*

We wish to find the AL query strategy which yields the highest F1-score. Several strategies exist, and some might work better for EM with TPLMs than others.

Research question 2 *How does active learning with Transformer pre-trained language models perform for entity matching with respect to time consumption?*

TPLMs are larger than classic machine learning models, and require more training time. This research question aim to answer how TPLMs perform under stricter resource limitations, when it is not allowed to train for an unlimited time.

Research question 3 *What are the challenges with combining active learning and Transformer pre-trained language models for entity matching?*

Lastly, we wish to detail challenges related to using AL with TPLMs for EM. This is useful to help focus further work in this field of research.

1.3 Approach

The goal of this thesis is to evaluate how active learning, in combination with TPLMs, perform for entity matching. The experiments conducted test different AL query strategies and how they perform in respect to F1-score and time use. The query strategies

include a classical AL sampling method called uncertainty sampling, along with the more recent strategies Partition-2 and Partition-4, specifically made for AL with deep learning models. In addition, we test novel strategies that use a combination of a classical Random Forest model and TPLM to select and match examples.

The experiments have been performed on different public datasets commonly used to benchmark EM systems. The results were compared to a baseline where examples were randomly selected, instead of using AL strategies.

1.4 Results

The results in Figure 1.1 show that the AL strategies yielded better results than randomly selecting examples. More specifically, Hybrid-Partition-2 and Partition-2 were the best performing query strategies in our experiments, consistently achieving the highest F1-score among all datasets. On average, they achieved a 0.06 higher F1-score compared to the baseline across the 5 datasets after 1000 labeled examples.

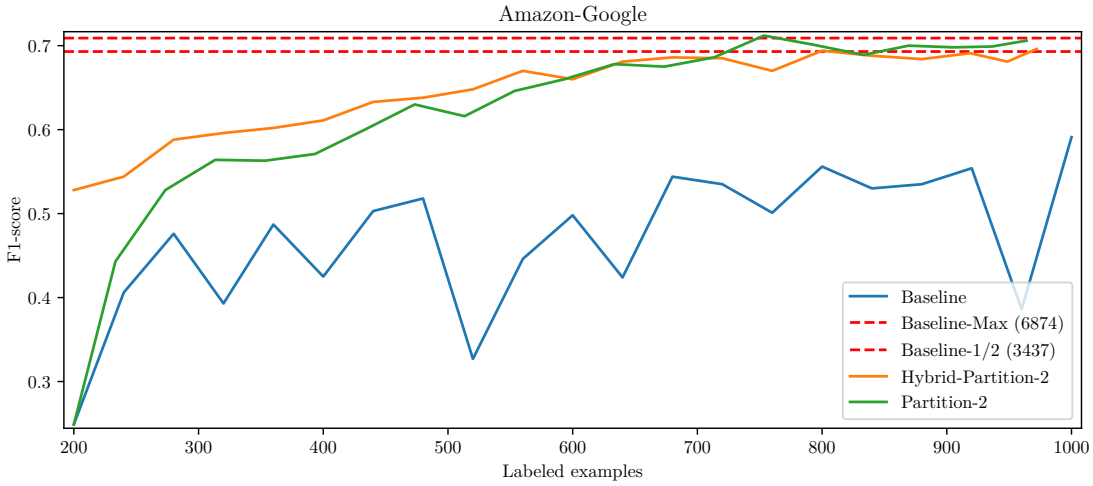


Figure 1.1: Iterative F1-score for Hybrid-Partition-2 and Partition-2, along with Baseline on Amazon-Google dataset. RoBERTa has been used as the TPLM. The red dotted line of Baseline-Max and Baseline-1/2 show the F1-score after all and 50% of the available data was labeled, the actual amount is in parenthesis. All experiments started with 200 randomly sampled examples, and each query strategy tried to label 40 examples in each iteration, with a maximum of 1000 examples over 20 iterations. One can see that both strategies significantly outperforms Baseline, with Hybrid-Partition-2 having a higher initial score, but is surpassed by Partition-2 after 600 examples are labeled. Partition-2 reach the maximum F1-score of Baseline after only around 740 examples have been labeled, compared to Baseline requiring 6874 to reach the same score.

The automatic labeling strategy of Partition-2 managed to select a balanced set of high-confidence and uncertain examples. The quality of the automatically labeled high-confidence examples was sufficient to make up for the fact that it was susceptible to label examples wrong.

In the figure one can see Partition-2 surpassing Baseline after only around 730 examples had been labeled, when Baseline was trained on all 6874 available examples. This was a repeating pattern on all the other datasets. On average it happened after only 9.1% of all training data was queried and labeled.

By combining Hybrid and Partition-2 into the novel Hybrid-Partition-2 AL strategy, recall was significantly increased in the first iterations, yielding a higher initial F1-score. In the first iteration it achieved an F1-score from 0.011-0.345 higher across 5 datasets, averaging on 0.157 higher than the Baseline. Hybrid-Partition-2 surpassed Baseline's F1-score at 1000 examples while only requiring 27.4% of the training data on average.

We found that even when using a larger TPLM it was possible to select hyperparameters which resulted in a iteration time of only 2.5-8 minutes. Combined with using an order of magnitude fewer labeled examples than the baseline, AL with TPLM could prove to be a realistic option when doing EM in practice.

However, we also found that a big challenge is to choose which TPLM to use, combined with selecting its optimal hyperparameters.

1.5 Thesis Outline

The thesis consists of 8 chapters. Each chapter is presented shortly below.

1. **Introduction** presents the motivation and goals of the thesis, as well as a description of the thesis.
2. **Background** provides the background theory and terminology related to entity matching, active learning, and Transformer pre-trained language models in order to understand the rest of the thesis and the results.
3. **Related Work** provides a summary of papers directly related to our thesis.
4. **Data** provides information about the data the public datasets used in the experiments in this thesis.
5. **Method** provides the necessary information about the experiments in order to reproduce the results.
6. **Results** provides the main results from the experiments.

7. **Discussion** provides an interpretation and discussion of the results.

8. **Conclusion and Further Work** provides a summary and short conclusion of the thesis with respect to the research questions outlined in the introduction. In addition, recommendations for further work is presented.

Chapter 2

Background Theory

This chapter will give an introduction and background theory to the field of entity matching, active learning, Transformer pre-trained language models, and the current state-of-art in EM and AL.

2.1 Entity Matching

Entity matching (EM) is the problem of identifying records which refer to the same real-world entity [5, 1, 3]. EM is commonly a necessary step when doing data integration, which is the process of creating a unified view of often heterogeneous data originating from different sources, such as databases, data warehouses, and data lakes [1]. EM is a challenging task, especially concidering the fact that huge amounts of data can be involved. If dataset A has N records and dataset B has M records, there exist $N \times M$ possible matches, however most of these matches are in fact negative (non-matches). Luckily, the field of EM has developed strategies to handle this extreme imbalance of positive and negative matches, commonly called *blocking* [1]. Blocking effectively removes many obvious negative matches, while having a high recall of positive matches. After blocking, the actual matching step can be done, typically by training a machine learning model on a (small) set of labeled data, and predicting unlabeled records if it is a match or non-match. A closer look at the EM process is presented in subsection 2.1.3.

2.1.1 History

EM has a rich history, stemming from even before computers existed with statisticians and public health researches wanting to match records from a single or multiple databases [1]. One example was the need to collect all health information about patients in a single place, however issues arising from reasons such as hand-written data became apparent.

In 1946, Halbert Dunn defined the term *record linkage* when he described the idea of a *book of life*, containing all information about a person [6]. The book would start and end with the birth and death record, and in-between all information related to a person and their interactions with the health and social security systems. He believed this would be of extreme value to both authorities and statisticians, but also recognized the complexity in dealing with data quality issues such as common names, errors, and variations in the data.

To solve this, Newcombe and Kennedy [7] proposed matching data with probabilistic record linkage approaches, automated with computers. One approach was to use a phonetic sound algorithm to encode attributes such as surnames to overcome minor name variations. Based on the distribution of these values, match and non-match weights could be computed and used to decide if two records match or not. A few years later, Fellegi and Sunter [8] published a paper proving that an optimal probabilistic decision rule can be found. It defined the common assumption that each attribute used in the comparison is independent of the other attributes. This theory have been the basis for several later works, and for EM systems and software products that are still in use today [1]. Probabilistic record linkage is for example the basis for the use of fuzzy (approximate) string methods and frequency based measures for match weights [9].

Concurrently, similar work in the database community focused on removing duplicate records in a single database [1]. This is an important part in the data cleaning process to improve the quality of the database. Duplicate detection is a specialized problem of EM, where one in practice has two identical datasets to be matched. Instead of using probabilistic methods, work went into how to sort the data based on their attributes such that similar records were grouped together in the same physical space in the database. String comparison functions could then be used to find duplicates.

State-of-the-Art

Since the early 2000's, EM has seen an increasing focus in the computer science research domain [1]. Especially in fields like data mining, information retrieval, machine learning, and database and data warehousing. With companies and organizations collecting larger and larger amounts of data, data quality has been recognized as an important challenge to effectively utilize the data.

Until very recently, the state-of-the-art in entity matching has been to use different classic machine learning models for matching, combined with strategies like active learning and crowd sourcing [10, 11]. Development of simpler, more explainable rule based methods has also been in focus.

Recently more development has gone into deep learning methods like neural networks [12]. They can sometimes perform better, but require an order of magnitude amount more training examples compared to classic ML models. Today, the current state-of-the-art is to use Transformer pre-trained language models, and the first paper to detail and compare the use of BERT and similar models was only in 2020 [2]. Compared to neural networks, TPLMs require less data, and perform better. Compare to classic machine learning models, TPLMs use significantly longer time to train.

The sub field of blocking has focused on key-based partition methods, filtering rules, and hashing among others. They have the downside of requiring manual fine tuning of hyperparamteres, and novel techniques include performing this process more automatically by employing DL methods [13, 14]. Another recent challenge for blocking is the rise of big semi-structured datasets, requiring blocking methods to work in a schema agnostic manner. For an extensive review of blocking we refer to [15].

2.1.2 The Entity Matching Problem

When merging databases unique identifiers (keys) link records together making it possible to compare records simply by comparing keys. When these do not exist or are corrupted the problem is much harder, as the actual contents of the records will have to be compared in order to determine which are linked together. When doing such a comparison of records, the potential number of matching records is quadratic. All records from one database could be a match with all records in the other database. As a result, entity matching can be a very computationally expensive problem when the databases are large.

Table 2.1.2 showcase a fabricated EM example from a product database of phones from two online stores. There are 3 different entities across the two dataset, where "A2" and "B2" refer to the same entity. From this example one can see some of the challenges in EM:

No common Id: Each dataset use their own custom "Id" field as primary keys, such that no item can be automatically joined across the datasets. This is the backdrop for EM.

Quadratic number of matches: Even though there are only two items in this simplified example, there is already 4 possible matches, where only 1 is a true match.

With 2x the number of examples in each dataset, the number of possible matches would increase by 4x, to 16.

No unified schemas: Records to be matched can have differently named attributes containing similar information. In Table 2.1.2 the attributes "Name" and "Phone" correspond to similar values, however their names are different. Making it necessary to detect which attributes correspond.

Combined attributes: Information which is split into several attributes in one dataset can be combined in the other. As can be seen in the example, the "Phone" attribute in B contains information which is split into "Name", "Model" and "Mfr." in A.

Same attribute name, different values: Attributes across datasets can have the same name but contain different values. As can be seen in the "Type" attributes in dataset A and B.

Different attribute types: Both datasets have a "Price" field, but Dataset A use a float type, while Dataset B use integer.

Missing information: There is often a lot of missing values within the data sets, making the matching harder. In some cases so much information can be missing that it is not possible to tell whether the records are matching or not. An example of a missing value can be seen in the column "Price" of record A2.

2.1.3 The Entity Matching Process

Figure 2.1 shows the traditional EM workflow with its 5 major steps [5, 1].

Preprocessing

Since data from different sources often vary both in format, structure, and content, it is necessary to preprocess it before it can be used for downstream tasks. Data preprocessing is often an important step in many data integration tasks [16]. For EM, this includes converting the two raw source datasets to the same format. This might include normalizing attribute values, handling missing values, removing unwanted characters, and more [1, 16].

Schema Matching

Schema matching is the task of finding out which attributes can be compared to one another. It can be viewed as a separate task to EM, however is in practice often done

Table 2.1: The two datasets A and B contain information about phones and their prices. A2 and B2 describe the same phone, even though the entity is represented in two different ways.

Dataset A

Id	Name	Model	Mfr.	Price	Type
A1	Pixel	XL	Google	£70.00	Mobile Phone
A2	iPhone	X	Apple	NA	Mobile Phone

Dataset B

Id	Phone	Price	Type
B1	Google Pixel OG	$600	Smartphone
B2	Apple iPhone 10	$749	Smartphone

Matches

Id	Matching
A1-B1	False
A1-B2	False
A2-B1	False
A2-B2	True

as part of the preprocessing [1]. By analyzing both the name and content of attributes, correlating attributes can be found. In this step it might be necessary to e.g. split one attribute into several separate attributes, in order to have attributes that conform to one another. An example is to split an address field into "city", "street", and "house number".

Blocking

After the data has been cleaned and corresponding attributes between datasets have been found, the matching process could potentially begin. However, since every single record in Dataset A could be a potential match to *every* record in Dataset B, some further preprocessing is necessary. Blocking, is a term used for different techniques used to reduce this quadratic amount of possible matches [1, 15, 17]. Common for all techniques is that they have as goal to remove as many negative matches as possible, while having a high recall of true positive matches. The end result is a candidate set of possible matches, small enough to do record pair comparison and matching.

Traditionally, blocking has referred to techniques that split the dataset into separate blocks consisting of similar records, based on some blocking criteria [1]. Only records

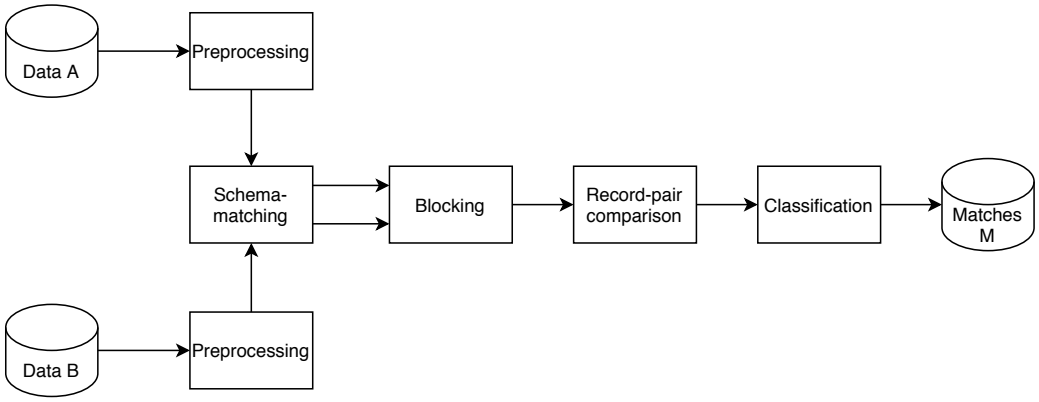


Figure 2.1: The traditional entity matching process as illustrated in [5]. First the data from the two sources are *preprocessed*, where it is cleaned and transformed into the same format. The second step is *schema matching*, where related attributes in the two datasets are found. The third step is *blocking*, where the quadratic complexity is reduced by filtering out obvious non-matches. In the fourth step, *record-pair comparison*, typically static string metrics are computed for all the candidate pairs. The actual decision if a record-pair is a match or non-match happens in step five, *classification*. This traditional view of EM does not consider human-in-the loop workflows like AL or when using TPLM.

in the same block are later used to create the set of candidate matches. Examples of traditional blocking are *standard blocking*, *q-gram blocking*, and *sorted neighborhood blocking*.

Another blocking strategy is known as *filtering*, which uses (simple) string similarity metrics and a threshold value to remove negative matches. An example is to use Jaccard similarity with a threshold value of 0.4.

A downside to the strategies mentioned above is that they require the user to manually set the blocking rules. Ramadan and Christen [18] have looked at how blocking keys can be selected in an unsupervised manner, and [4] use the encodings from a deep learning model to find possible matches.

Record Pair Comparison

After the number of possible matches has been reduced to the candidate set of matches, more computationally expensive similarity measures can be computed between pairs of records. The end result is a similarity vector, consisting of numerical values which each represents how similar two specific attributes from each record are [1]. The choice of which attributes to compare is done in the schema matching step, and ideally the number

of attributes compared is greater than in the blocking step.

Even after the data has been cleaned and standardized, it can still be the case that two attribute values which represents the same entity do not have the same value. Therefore, it is necessary to employ approximate similarity functions [1]. The similarity value is generally a normalised value between 0 and 1, describing no similarity and an exact match, respectively. Typically, several different *string similarity* measures are used on each pair of attributes. Depending on the attribute value type (e.g. date, age, time, location), other more specialized similarity measures can be used as well.

Libraries such as Magellan (see chapter 5) can help in automatically finding suitable similarity measures based on the attribute type. For strings it can select measures suitable for the specific length of the string.

Classification

In the classification step, candidate record pairs are classified either as a match or non-match, based on their similarity vector. A very simple strategy is to sum up all the values in the (normalised) similarity vector, and classify it based on the sum and a fixed threshold. For example, a record pair having a vector with 10 entries could potentially be a match if the sum was greater than 8. However, this method does not take care of the situation where the significance of different attribute similarity scores might differ. When comparing two mobile phone records, the fact that the attribute representing the model number is similar should probably be of more importance than the color attribute being similar.

More elaborate methods is therefore necessary, such as the aforementioned probabilistic record linkage approach. This method puts less significance to attributes that are common among all candidate record pairs. It also finds *potential matches*. These matches are too difficult for the model to predict, and are passed to manual review.

In recent years, matching algorithms exploiting rules, supervised and unsupervised machine learning techniques, active learning, and crowdsourcing have been developed [17, 1]. An important focus of the research is to improve the accuracy and reduce run time costs.

2.2 Active Learning

Active learning is motivated by the key idea that a machine learning model can achieve improved performance with fewer labeled data points, if the model itself is able to choose what data it is going to learn from [19]. AL can be especially effective in a situation when the availability of unlabeled data is high and easily obtained, but acquiring labeled

data is difficult, time-consuming, expensive, or it is difficult to identify a suitable set of labeled data [19, 20, 10].

A perfect example of such a situation is in EM, where it is easy to gather unlabeled examples, however the data can be heavily dominated with negative matches. Intuitively one can imagine that the model's rate of improvement from having more labeled examples will diminish along with the increase in the number of labeled examples, especially if many of those examples have almost the same properties. In addition, some properties could be *easier* to learn for the model. E.g. for EM, this could be teaching the model that a candidate match with *no* similar features is a negative match. However, the opposite could be true as well; some properties could be *harder* for the model to learn.

The goal of AL is in this sense to select the examples from which the ML model will learn the most, namely the most *informative* examples, such that the model can achieve the highest accuracy with fewest labeled examples.

2.2.1 The Active Learning Process

Active learning is an iterative process where in each iteration a set of data points, known as examples, are added to the set of labeled data and then used to train a new ML model. This is depicted in Figure 2.2. A traditional ML classification process is depicted in Figure 2.3.

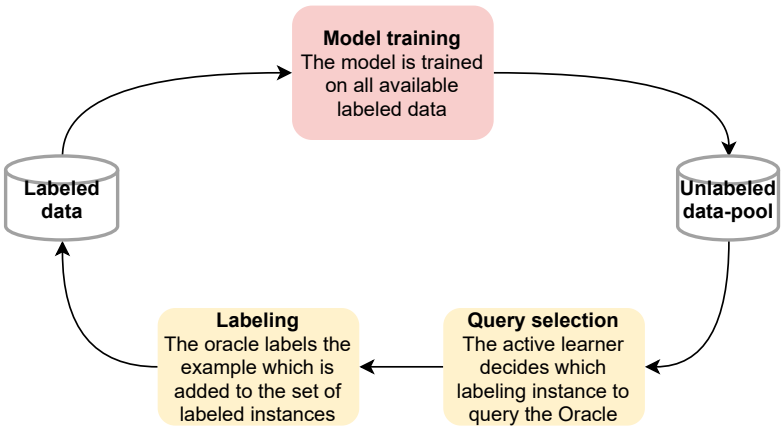


Figure 2.2: Active learning iteration: In each iteration a ML model is trained on the set of labeled data, then the AL algorithm employs a query strategy to select examples which are passed to the Oracle for labeling, lastly the new labeled data is added to the training set and the model retrains. This continues until a stopping criteria is met.

Before the first AL iteration it is necessary to have a ML model that can make predictions about the unlabeled data. This is typically achieved by training a ML model on a small

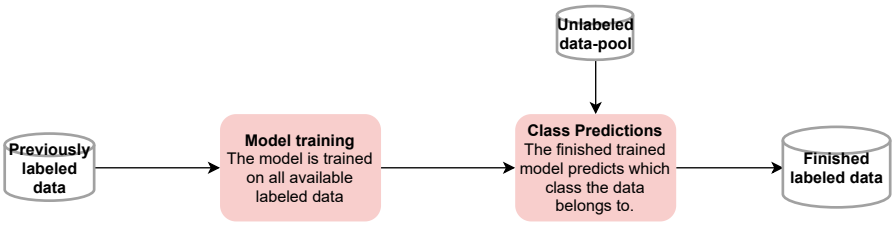


Figure 2.3: A traditional ML classification process. We start with an initial set of labeled data in which the model is trained on, then the model makes predictions for the class of the remaining unlabeled data.

seed, a set of randomly sampled data, however, a pre-trained model can also be used [1, 19, 21]. The initial model will likely have a low accuracy, but can still be used by the AL algorithm, the learner, in a *query strategy*. The query strategy selects a set of unlabeled examples to be labeled by an Oracle, a human annotator. Different query strategies exists, some of which will be detailed in the next section. One popular strategy is called *uncertainty* sampling, where the examples in which the model is most uncertain of are selected. The newly labeled examples are then added to the set of labeled data, which is used to train the model again. This process can repeat until a stop criteria is met, which could be a time limit, exceeding maximum number of iterations, a certain matching quality is met, or there are no more unlabeled data available [1].

2.2.2 Query Scenarios

There are different scenarios in which a learner can use a query strategy to select examples for the Oracle to label. The most common in academia and in real-world problems is *pool-based sampling* [19]. In a pool-based sampling situation there exist a large set of unlabeled data $\mathcal{U}$, and a smaller set of labeled data $\mathcal{L}$. This is also the case in EM, where the set of candidate matches is the unlabeled pool, and labeled matches is $\mathcal{L}$. When selecting examples to be labeled, the learner evaluates the whole set $\mathcal{U}$ before finally selecting its queries for the Oracle. In the later sections pool-based sampling can be assumed if not otherwise specified.

Another scenario is *Stream-based selective sampling*, where the learner scans through a set of data sequentially, and decides individually if it should query the Oracle with the example [19]. An assumption is that retrieving an unlabeled example is inexpensive, while querying the Oracle is not. The learner must therefore be able to compute if the example is informative "enough", either by an informativeness measure, or by computing if the example lays outside the model's *region of uncertainty*, the part of the instances space that is ambiguous to the learner.

In *Membership query synthesis* the learner itself can generate examples and query them

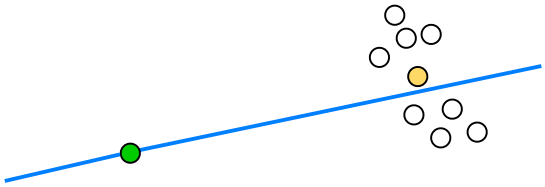


Figure 2.4: Blue line: decision boundary, green: uncertainty sampling, yellow: density-weighted methods. A simplified example of the difference between density-weighted methods and uncertainty sampling. Uncertainty sampling might be at a disadvantage as it might not pick the most informative example. Figure inspired by [23].

to the Oracle for labeling. A downside to this strategy is that it can be hard for a human Oracle to label an artificially created example [19].

2.2.3 Query Strategies

In each AL iteration, the learner needs to select one or more example(s) for the Oracle to label, and this strategy for selection is called a *query strategy*. Traditionally, examples have been selected based on some informativeness-score or measure, however with the rise of deep learning new strategies have been developed to cover DL specific needs such as the importance of more data [19, 20, 10, 21]. Some common query strategies are detailed below. The experiments presented in chapter 5 use a query strategy based of uncertainty sampling, which arguably is one of the simpler methods. Diversity-based strategies are not presented, as they are used when one have multiple Oracles labeling at the same time. For a comprehensive review of deep learning query strategies not commonly used for EM, we refer to [22].

Uncertainty Sampling

In *uncertainty sampling*, the most informative example is the example the model is the least certain of [19]. The key idea is that the model would learn more from knowing the label of the uncertain "difficult" example, than if a more certain "easier" example was labeled. In a binary classification setting (such as EM), a probabilistic model would predict an uncertain example to be positive with a probability score close to 0.5. Consequently, the negative class also has a probability of close to 0.5. The query strategy would then sample the example with the positive probability score closest to 0.5. This methods is also called *least confidence sampling*, and can also be used in a multiple class setting, as described below. In Figure 2.4 the uncertain example is the one on the blue line, also known as the decision boundary, the line representing a 0.5 probability score. A positive certain example would have a score close to 1. Uncertain and certain examples

are also known as low-confidence and high-confidence examples, respectively.

When in a multiple class setting, uncertainty sampling still query the most uncertain example, however there are more ways to compute the uncertainty based on the different class probabilities. Note that in a binary classification setting, all methods detailed below reduce to least confidence sampling.

Least confidence sampling simply queries the Oracle with the example, where among all the example's class predictions, the most confident class has the lowest prediction score compared to all other examples. Formally, this is defined as:

$$\textit{Least\_Confidence} = \underset{x}{\text{argmax}}\ 1 - P_\theta(\hat{y}|x)$$

where $\hat{y} = \text{argmax}_x\ 1 - P_\theta(y|x)$, the class label with the highest probability under model θ. y ranges all possible class labels, and x ranges all unlabeled examples. Another way to look at this strategy is the model's expected 0/1-loss, or how likely the model believe it will mislabel the example x. One major disadvantage of this query strategy is the fact that the model only considers the one most probable class, and discards information about all the others.

Margin sampling is similar to least confidence, however it considers the difference between the two most confident classes. If an example has a large margin, the model is able to easily differentiate between the two most likely classes, however if the margin is low, the classes are more ambiguous and therefore uncertain. Margin sampling is defined as:

$$\textit{Margin} = \underset{x}{\text{argmax}}\ P_\theta(\hat{y}_1|x) - P_\theta(\hat{y}_2|x)$$

where $\hat{y}_1$ and $\hat{y}_2$ are the two most confident classes. A generalization of margin sampling that have the advantage that it considers *all* possible class probabilities is **entropy sampling**:

$$\textit{Entropy} = \underset{x}{\text{argmax}}\ -\sum_i P_\theta(y_i|x)\ log_2\ P_\theta(y_i|x)$$

where y_i goes over all possible classes. Even though the three uncertainty query strategies above are similar, there is no one best, and empirical experiments suggest that the choice of strategy may be application-dependent [19].

Query-By-Committee

A different query strategy called query-by-committee (QBC), use several machine learning models to find the most informative example [19, 24]. All models are trained on the same set $\mathcal{L}$, but instantiated slightly differently such that they have competing hypotheses. The main idea is that the most informative example is the example the most models *disagree* on. E.g. in a binary classification setting, a committee of 2 models, QBC(2), will find informative examples where one model says the example is positive and the other model says it is negative. There is no one best committee size, but even small sizes such as 2-3 has shown to work well [24], and further increasing the number of members does not significantly increase the performance [25].

As mentioned above, each model in the committee needs to be instantiated with a *different* hypothesis, and there exists several ways to do this. If this is not done, one would end up with exactly similar models, and the goal behind QBC is lost because there is no disagreement. One way is to *bootstrap* each model separately [19, 10]. If $\mathcal{L}$ has n instances, bootstrap will randomly select n examples with replacement as the training set. Since each model most likely ends up with a different training set, they will also build different model hypothesises, which result in different predictions. Other methods of creating a committee exists, such as randomly sample models from a distribution, by partitioning the feature space and training models that considers only specific features, or randomly selecting model hyperparameters [19, 25].

After the different model hypothesises have been made, they can be used to measure their disagreement. A common way is to use entropy in a method called *vote entropy* [19]:

$$Entropy = \underset{x}{\operatorname{argmax}} -\sum_{i} \frac{V_{(y_i)}}{C} \log \frac{V_{(y_i)}}{C}$$

where y_i again goes over all possible classes, and $V_{(y_i)}$ is the number of positive match labels for class i. Instead of using the class label, it is also possible to use each models probability score, and this is know as *consensus entropy*.

Several other committee based methods based on QBC exists, however we will not detail them here, and refer instead to [19] for a more comprehensive introduction. Common for all committee based methods is that there is a potential for an significantly increased run-time overhead when creating and querying the committee members. Compared to the arguably simpler uncertainty sampling (which only use one model), QBC(n) has to, for every iteration, create the committee by re-training n models, use them to predict the class labels in $\mathcal{U}$, and finally compute the disagreement among the predictions. [10] saw a difference in run-time between QBC and uncertainty sampling by 10-100x, even though the final F1-score was similar. With more data and complex models, such as in

DL, QBC might therefore not be a suitable method due to the significantly increased run-time.

Expected Model Change

The key idea behind the query strategies in *expected model change* (EMC) is to select the examples, that when the model knew its labels, would make the greatest change to the current model's hypothesis parameters [19]. One such query strategy is *expected gradient length* (EGL), which can be used when any gradient-based ML model is used, such as with neural networks. EMC can work well in practice, however is computationally expensive as it has to re-train the model for *every* unlabeled example to be able to find the candidate query examples [26]. [27] found EGL to use an iteration time of over 10x the time an uncertainty based method used.

Density-Weighted Methods

A drawback of uncertainty, QBC, and EGL based methods is that they are prone to select outliers that potentially does not improve the model, or even worsens it [19]. This is due to the fact that the outliers are not representative of the rest of the data. The reason they select these examples is because they only consider examples in isolation, i.e. they do not make the query decision based on how the example compare or relate to the rest of the data.

Density-weighted methods tries to fix this by incorporating into the selection strategy a measure of how representative each example is. An informative example must, in addition to being uncertain, also have unlabeled examples that are similar to itself. In other words, it is preferred if it belongs to a dense cluster of other examples. Settles [19] show that if the densities are pre-computed, density-weighted methods can be almost equally as fast as e.g. plain uncertainty sampling.

2.2.4 Challenges with Active Learning

Theoretically, AL sounds like a promising strategy to reduce the number of labeled examples while performing at a similar level, however there are some challenges related to it.

Feature Extraction and Explainability

Depending on the choice of machine learning model, carefully crafted feature extraction might be necessary [28]. Classical ML models such as RandomForest is dependent upon numerical features, e.g. it can not use textual features directly. Before AL can be started, there is therefore a high initial cost related to generating meaningful features. In the case of EM, there exists tools such as Magellan[1] which can help do some of this feature extraction and generation automatically. What one ends up with are different similarity features such as Jaccard or edit distance. However, for a human Oracle to directly use these numerical features to decide if an examples is a match or not might be hard, and they would likely need to look at the original text representation to make a decision if it is a match or not. Comparing and using the numerical features with the raw text might not be straightforward, and it might be hard to reason *why* the model believes it is a match or not.

Feature extraction is not necessarily needed in the case of DL, which can use raw text directly. An example of this is with TPLMs. The problem of explainability of why a model believes an example is a match or not still persists [5].

Oracle and Time Use

The availability of a person to act as the Oracle is an essential requirement for AL to take place at all, and depending upon the dataset, a domain expert might be needed. In practice, the Oracle can not be expected to wait for a long time between each iteration for the model to re-train and select examples for labeling. In other words, there is an arbitrary time limit on how long the AL algorithm can use in each iteration. This puts a restriction of what combination of ML model, dataset size, and hardware that can be used.

Complexity

There is also simply a need for more programming to set up the environment, especially related to handling the output and input to and from the Oracle, along with re-training the ML model. The Oracle has to visually see the examples it is to label, and have the possibility to label it accordingly. The new labeled examples have to be added to the rest of the labeled examples $\mathcal{L}$, and a new model trained from this updated dataset. Some frameworks such as modAL[2] exist to handle some of this complexity, however it is still arguably not as easy as to just train a model once on one set of data, as with

[1] github.com/anhaidgroup/py_entitymatching
[2] https://github.com/modAL-python/modAL

supervised learning.

Overall, AL can be a more complex activity and requires more effort to set up and perform compared to e.g. standard supervised learning. In supervised learning *all* the training data is labeled only once and there is only one "AL iteration" of building a model and using it for e.g. classification.

2.2.5 Alternatives to Active Learning

An alternative to AL that do not use an Oracle is transfer learning (TL), where a ML model is trained on source dataset(s) before being applied to the target dataset [21]. TL can suffer from dataset specific properties, meaning that the model learns relationships in the source datasets that do not hold in the target dataset. However, if the source datasets are already labeled, there is no need for an Oracle.

A closely related area of research to AL is that of semi-supervised learning, where a model tries to make use of both the set of labeled and unlabeled data in its training [19]. In a technique called self-training, a model trains on a small set of labeled data and then (unsupervised) adds the most confident examples from the unlabeled data to its training set. It can then repeat this process.

2.3 Transformer Pre-trained Language Models

With the increase in popularity of deep learning it has also become more popular in the field of entity matching. The last few years Transformer pre-trained language models (TPLMs) have seen an increase in popularity, achieving state-of-the-art results, while having other advantages such as no need for hand crafted features.

To limit the scope of the thesis we will not write about other NLP and DL architectures, however we will focus on what is of interest related to the experiments performed.

2.3.1 Attention

Before TPLMs, seq2seq and encoder/decoder architectures applying Recurrent Neural Networks (RNNs) were popular in the Natural Language Processing (NLP) field [2]. Traditional encoder-decoders applies an encoder to encode a target sentence to a fixed-length vector. The decoder takes this fixed-length vector as input and generates an output sentence (often a translation).

However, as described by [2], some key issues with these approaches led to the develop-

ment of attention mechanisms and TPLMs: RNNs are hard to parallelize due to being mainly sequential, which gives them a time disadvantage due to the inability of parallelization. Another problem is the possibility of information loss when information is passed through many recurrent connections in the RNN. The decoder takes as input a fixed-length vector which makes representing all relevant information of long sentences hard, this is known as the *bottleneck problem*. Approaches like LSTMs and GRU architectures tries to allow RNNs to learn the context of words far apart in sentences, however these dependencies are still hard for RNNs to learn.

Bahdanau et al. [29] presumes the fixed-length vecture to be a performance bottleneck and proposes allowing the model to pay *attention* to the parts of the sentence which are most relevant to the prediction of the next word. Consequently, self-attention was introduced as a mechanism to master the bottleneck problem. Intuitively self-attention can be understood as the brains ability to understand which words are relevant to other words in a sentence. It represents a word as a weighted combination of the words in its proximity. This makes it possible for the model to pay attention to the parts of the input with the most relevant information. Using this positional information of the most relevant words in the source sentence the model tries to predict the next word in the target sentence.

2.3.2 Transformers

As mentioned, previous deep learning approaches to NLP tasks have had two main issues that Transformers seek to mitigate. Previous approaches like LSTMs seek to reduce the problem, however it still remains. These approaches are of a sequential nature which makes utilizing the parallel advantages of newer hardware not possible. These problems have been the drivers of the research and development of Transformers.

Transformers as introduced by [30] utilizes *attention* mechanisms to overcome the information-loss disadvantages of previous approaches. They were introduced as an architecture not using RNNs and only using attention. RNNs were replaced by *multi-head-attention* layers while keeping the encoder/decoder structure.

The words in the input sentence is represented by word embeddings in combination with positional embeddings. Positional embeddings are used to represent ordered, sequential inputs as they do not use recurrence as in RNNs. This mechanism makes it possible for the Transformer to utilize positions of and distances between words in a sentence. This approach makes it possible to overcome the bottleneck-problem aswell as allow for parallelization during training.

2.3.3 Models

After the release of [30], several implementations of Transformers have been released. They vary in the amount of training data in which they have been pre-trained, size and implementation. In this section the relevant Transformer models of this thesis are introduced. The reasoning behind why RoBERTa and DistilBERT were chosen is described in chapter 5.

BERT - Bidirectional Encoder Representations from Transformers

In 2018 Devlin et al. [31] introduced BERT, a Transformer pre-trained language model (TPLM) which achieved state-of-the-art performance on several natural language processing (NLP) tasks, showing the potential of TPLMs. BERT is shortly introduced here, as it creates the basis for the models introduced below.

BERT trains by trying to predict words that are randomly excluded from sentences. It is bidirectional meaning instead of using just the context on one side of a word (the words to the left or right of the word to predict) it considers the entire sentence (technique called MLM - Masked LM). Which means it looks at the entire sentence at once.

Even though Transformers consists of an encoder and a decoder, BERT only makes use of the encoder. A sequence of tokens is used as input for the encoder. [CLS] marks the beginning of a sentence and [SEP] marks the end. Markers are used to separate between sentences and positional embeddings are used to mark the position of a word in the sentences. MLM is a technique used during training which replaces 15% of words with a [MASK] token. The sentences are then run through the encoder and the masked words are predicted. However, to make the model predict words even if the token is not present, 10% of the mask tokens are replaced with random tokens and 10% of the masked tokens remain unchanged (not masked).

RoBERTa

Several models have been released after the initial publishing of BERT, generally trying to improve on its performance by doing various optimizations. Liu et al. [32] presents the challenge of interpreting which elements of subsequent models which actually lead to their respective performance gains. These challenges were caused by limited tuning possibilities due to computationally expensive training, as well as training data being private. Suggesting BERT was significantly undertrained, they present the Robustly optimized BERT approach (RoBERTa), which is a model resulting from a more optimized training approach of BERT. They conclude that training BERT in this manner will yield competitive results to BERTs more recent alternatives.

DistilBERT

Other approaches seek to create lighter and faster models while retaining as much of the performance advantage of heavier models as possible. Sanh et al. [33] introduces DistilBERT, a pre-trained version of BERT which aims to be faster, cheaper and lighter than BERT. They utilize knowledge distillation, a method in which a smaller model is trained to reproduce a larger models behaviour, during pre-training. They manage to hold on to 97% of BERTs language understanding capabilities, while being 60% faster and reducing the model size by 40%.

2.4 Performance Measures

In order to quantify the performance of an EM system, some common performance measure are used, namely precision, recall, and F1 [1]. These are not unique to EM, but have their origin from the field of information retrieval. In order to evaluate the performance, or quality, of an EM system it is often a requirement to have a set of *gold standard* data. This is a set of labeled data that are known to be correct, normally made by manual labeling. In practice, if the gold standard is of sufficient size and of a representative distribution, it can be used to evaluate an EM system on a larger target dataset. However, it is important to keep in mind that even the manually labeled gold standard is likely to contain labeling errors [1]. The quality of the gold standard will therefore also have implications for the quality of the matching system, and its performance measures.

The performance measures mentioned below are given as a standardised value p, where $p \in [0, 1]$. A higher value indicates a higher performance in the respective measure.

A record prediction can be put in one of four different categories:

- *True positives (TP).* An actual match (true match) is correctly predicted as a match. The two records in the candidate match refer to the same entity.
- *False positives (FP).* An actual match is wrongly predicted as a non-match. The two records in the candidate match refer to the same entity, however the classifier did not manage to predict the correct label.
- *True negative (TN).* An actual non-match (negative match) is correctly predicted as a non-match. The two records in the candidate match refer to two different entities.
- *False negative (FN).* An actual non-match is wrongly predicted as a match. The two records in the candidate match refer to two different entities, however the classifier did not manage to predict the correct label.

An arguably naive way to measure the performance of an EM classifier is to look at how many correct predictions it made:

$$acc = \frac{TP + TN}{TP + TN + FP + FN}$$

This is the *accuracy* of the classifier. Since EM is typically an imbalanced classification task, the accuracy of a classifier can give the wrong impression of its performance, given certain conditions. Imagine a situation where 10% of the candidate matches are true positive. By predicting *all* examples as negative matches, the classifier would have an accuracy score of 0.9, however 0 matches were actually found. The measures below are more appropriate for EM [1], where there typically is a significant imbalance of positive/negative matches.

2.4.1 Precision

Precision is a measure of how precise the classifier is in predicting true matches. It does not suffer from the imbalance problem of accuracy, by not incorporating the number of true negative matches.

$$precision = \frac{TP}{TP + FP}$$

Precision give insight into how many of the true matches were correctly classified. A high precision is necessary if a requirement of the EM system is to not wrongly label matches.

2.4.2 Recall

Recall is a measure of how many of all the true matches were classified correctly.

$$recall = \frac{TP}{TP + FN}$$

A low recall means that the classifier had many matches it could not "find", and wrongly labeled those matches as negative.

2.4.3 F1-score

It is possible to have a high precision and low recall, and vice-versa. If a classifier is only predicting a few examples as matches, but all those examples are true matches, it has a high precision. However, if many true matches are left wrongly labeled as non-matches, the recall is low. Ideally, one wish to have both a high precision and recall. In

this situation a high percentage of the predicted true matches are correct, and few true matches are wrongly labeled.

F1-score (also known as f-measure) combines precision and recall as a weighted measure, and is the harmonic mean between them [1].

$$F1 = 2 \times \frac{precision \times recall}{precision + recall}$$

A high F1-score means both precision and recall is high.

In recent years, the use of measures that incorporates imbalance, like precision, recall, and F1-score, has become more predominant in computer science literature over accuracy [1]. In recent EM literature, F1-score is commonly used to measure the overall performance of an EM system, while recall is used to measure the performance of a blocking strategy [17, 10, 15]. Depending on the goal of the EM system, using precision might sometimes make more sense. This is especially the case if there is no acceptance for wrongfully labeled positive matches.

Chapter 3

Related Work

This thesis is mainly concerned with the fields of AL, EM and DL. In this chapter we will present related work in the AL and DL field separately as well as in combination, specifically focusing on the task of EM. We will also shortly present related work focusing on AL with classical machine learning methods for the field of EM.

Transformer Pre-trained Language Models for Entity Matching

Brunner and Stockinger [2] was among the first papers to put Transformer pre-trained language models up against traditional machine learning and deep learning for entity matching. They showed how several popular models can be used without any task-specific architecture, and still achieve state-of-the-art F1-scores. They found that TPLMs outperformed previous state-of-the-art DL models by an average of 27.5%. Among the models tested, RoBERTa got the best results, while DistilBERT was the fastest, using only around half the training time of RoBERTA. Although their work is focused on EM with TPLMs, AL and EM in low-resource environments is not a consideration in their work.

Peeters et al. [34] further optimize the BERT model for EM by having one more training step before fine-tuning the model for the target EM dataset. This extra training step works much in the same way as the original training of the BERT model, however the data are from EM datasets. In this way the model is more specialized for EM and they achieve higher F1-score, but there is an increased need for labeled data, processing power, and time.

Li et al. [3] presents a system for EM, DITTO, where pre-trained Transformer based language models are employed for EM. Although their work is not primarily focused

on EM in a low-resource scenario, they achieve previous state-of-the-art results with no more than half the training data. In addition to train the language models for EM tasks, DITTO uses three methods to improve the F1 score and reduce the size of necessary training data.

The first method makes it possible to manually insert domain knowledge in the training data, in order to highlight to the model which features are especially significant. The intention is that the model will pay more attention to these features. The two types of domain knowledge that can be inserted are *span typing* and *span normalization*. Span typing can let the model differentiate between numeric values such as street numbers, years, or product ids. Span normalization allows for standardization of values and attribute names. For example, "17.00%" and "17%" are equal, using span normalization these can be rewritten as "17%".

The second methods aims to preserve only the most important information in records that are too long for the language model. This is done by removing stop words and preserving frequent words. The third method use data augmentation techniques to create more training examples. Examples of record augmentations employed are to delete or shuffle both spans and attributes. In order to circumvent the possibility of making a match into a non-match, e.g. by deleting an important attribute, they also employ a recent NLP data augmentation technique called MixDA [35].

The results show that the optimizations improve the F1-score compared to previous state-of-the-art on public datasets. In addition to performing well on small datasets with few training examples, it requires less labeled examples to perform similar to previous state-of-the-art on big datasets. Their work is similar to ours in that their method is evaluated against partitions of the training data, and transformer architectures are applied for EM. However, we evaluate generalized AL strategies and do not apply domain knowledge, data augmentation or string summarization. A disadvantage of DITTO is the need of specific dataset knowledge required by a domain expert in order to insert domain knowledge, and longer train time when doing data augmentation.

Active Learning for Entity Matching

Meduri et al. [10] test combinations of AL strategies and classical machine learning models on common public EM datasets. Aiming to provide a framework for researchers detailing which combinations work well for EM. Their results show uncertainty sampling and QBC performing on par, with uncertainty sampling consistently using less iteration time. Their best result was obtained using an implementation of QBC with Random Forest. Their experiments includes rule-based classifiers which achieved lower F1-scores, however were much more interpretable. Their approach with uncertainty sampling is very similar to ours, however they do not combine these with TPLMs.

Deep Active Learning

Perhaps most similar to our work is Kasai et al. [21] which targets the common setting where the availability of labeled target data is low. The main difference being their use of transfer learning and the fact that they do not use TPLMs. In addition, we evaluate combining the use of AL with traditional ML methods and AL with TPLMs to speed up the labeling process.

By using TL followed by AL with traditional DL methods, they achieve similar or better results with less data than previous state-of-the-art methods. Their TL approach require access to other similar datasets which are labeled. However, their TL and AL techniques are independent, and the AL approach can be used individually if datasets for TL are unavailable. They create a neural network architecture which learns from labeled source datasets, then apply AL to find examples from the target dataset which the network can train on. Their approach is evaluated on common public benchmarking datasets for EM.

They use an AL algorithm specialized for DL models and the EM task, by trying to select a balanced set of positive and negative examples. They split the predicted examples into groups predicted to be matches ($prediction \geq 0.5$) and non-matches ($prediction < 0.5$). For both groups they calculate the entropy for each example based on the predictions given by the model. For each group they select the k examples with the highest entropy. These are named likely false positives and likely false negatives (LFP/LFN), depending on which group they are selected from. These samples are labeled by the Oracle. They also select the k pairs from each group with the lowest entropy, named high-confidence positives and negatives (HCP/HCN). The high-confidence examples use their label from the predicted label, i.e, they are labeled in an unsupervised manner.

This algorithm makes it possible to supply the model with twice as much training data as the Oracle has labeled. The reasoning behind this method being that traditional DL models are known for requiring huge amounts of training data to build a usable model from 'scratch'. Even though some predicted labels might be wrong the advantage of having more training data might be greater than the disadvantage of some labels being wrong. HCP/HCN examples are selected to prevent the network from overfitting to select uncertain examples ([36]), LFP/LFN to improve precision and recall ([37]).

Active Learning with Transformer Pre-trained Language Models

Ein-Dor et al. [27] tests AL strategies for the problem of binary text classification using TPLMs. They use a variety of query strategies, covering uncertainty sampling, QBC, density-weighted and EGL based methods. In addition, they use a method presented in [38] originally created for image classification, Core-Set. They evaluate AL in three

scenarios differing in the positive/negative balance of the training set. Their results show that AL works better than random sampling, but there is not one single query strategy that consistently performs well. In addition, they show that the AL strategies can quickly generalize the model and increase recall as a result. They also recommend trying a combination of AL strategies which excel in choosing representative (density-weighted) and diverse (core-set) examples for further work.

Active Learning with Transformer Pre-trained Language Models for Entity Matching

Jain et al. [4] utilizes AL for EM and proposes including the blocker in the AL loop. A traditional EM process do blocking before the matching step. However, they suggest the blocker and the matcher can concurrently benefit from the newly labeled matching-pairs during each AL iteration. As a result, their approach consists of an integrated blocker and matcher named DIAL. DIAL uses TPLMs and their method is evaluated on several public datasets, receiving significant improvements over several previous state-of-the-art results for both blocking and matching.

Their paper was submitted in April of 2021 and their work is very similar to ours in many ways. As we have done, they test AL with TPLMs for EM. However, they include the blocking step in the AL loop while we disregard blocking. They use a novel QBC-based indexing query strategy, and while it performs well, it also required more processing power and iteration time. In contrast, we evaluate a combination of classical ML with AL and TPLMs as a method to speed up the process.

Chapter 4

Data

This chapter presents the datasets used for the experiments in the thesis. In addition to detail the structure of the datasets, examples of matches and non-matches are presented.

4.1 Public Datasets

The datasets used in this thesis are a selection of some of the most common public datasets used for EM benchmarking [17], shown in Table 4.1. These datasets exists in several different versions in different papers, with different pre-processing being done, with and without fixed splits.

We have chosen to use a selection of datasets from the DeepMatcher project[1] [28], similar to [3, 4]. This specific distribution has dataset specific pre-processing and blocking. There are fixed splits for train, validation, and test set with a ratio of 3:1:1. The fact that the datasets are already blocked means that we do not have to think about the blocking stage in the EM process, and can solely focus on matching. The datasets fixed splits and the fact that they are already blocked makes it easier to compare our results to others who used the same datasets.

Mudgal et al. [28] contributed a categorization of EM datasets in order to interpret in which EM problems DL accelerate. They divide the relevant datasets in two categories; Structured EM and Textual EM. Structured datasets have properly aligned attribute values and textual attributes only of short lengths. Its entries are regarded as relatively clean. Textual datasets have only textual attributes.

[1] `https://github.com/anhaidgroup/DeepMatcher/blob/master/Datasets.md`

Table 4.2 shows the number of positive examples in each of the datasets respective train/test/validation splits. These numbers can be important, because when using certain AL strategies, the model might run out of positive examples during the AL iteration.

Table 4.1: The datasets in our project, as provided by DeepMatcher. NA_A and NA_B represents the number of missing values in records_a and records_b respectively. E.g. for Amazon-Google Amazon has 4.87% missing values, while Google has 29.65%. WC denotes the average number of words in A and B, when all attributes are concatenated and numerical values are ignored.

Name	Domain	Size	Attr.	NA_A [%]	NA_B [%]	WC_A	WC_B
Amazon-Google	Software	11,460	3	4.87	29.65	8.1	8.2
Abt-Buy	Products	9,575	3	20.44	28.79	47.1	17.1
Walmart-Amazon	Electronics	10,242	5	2.33	9.41	14.2	15.2
DBLP-ACM	Literature	12,363	4	0.0	0.15	18.3	21.8
DBLP-GoogleScholar	Literature	28,707	4	4.1	19.37	16.2	18.8

As previously stated all datasets used in the project are common public EM datasets often used for benchmarking EM solutions. All of these are originally created from existing websites containing data of software, electronics, consumer products and academic literature The datasets consists of five tables: *records_A* (A), *records_B* (B), *train*, *test* and *val*. A and B represent entities from two different sources, e.g. in Walmart-Amazon A contains products from Walmart and B contains products from Amazon.

Train, Test and Validation each contain three columns: *ltable_id*, *rtable_id* and *label*. ltable_id and rtable_id are referring to records in A and B respectively. Label can either be 1 or 0 depending on whether the two records from A and B are regarded as matching or not. As shown in Table 4.2 each Train/Test/Validation split is in the ration of 3:1:1. The table showcase how many examples are in each split, and how many of those that are positive matches.

Examples of record-pairs from the five datasets are shown in Table 4.3. Amazon-Google is concerned with software products. Google containing a lot more missing values than Amazon. The dataset is not very textual with pretty short sentences. DBLP-ACM

Table 4.2: Size and number of positive examples in the dataset-splits of the datasets.

Dataset	Matches [%Pos]	Train [Pos/Tot]	Test [Pos/Tot]	Validation [Pos/Tot]	Total [Pos/Tot]
Amazon-Google	10.17	699/6874	234/2293	234/2293	1167/11460
Abt-Buy	10.73	616/5743	206/1916	206/1916	1028/9575
Walmart-Amazon	9.38	576/6144	193/2049	193/2049	962/10242
DBLP-ACM	17.96	1332/7417	444/2473	444/2473	2220/12363
DBLP-GoogleScholar	18.62	3207/17223	1070/5742	1070/5742	5347/28707

has almost no missing values, but is more textual than Amazon-Google, however quite similar to Walmart-Amazon and DBLP-GoogleScholar. DBLP-GoogleScholar is slightly less textual, however has significant amounts of missing values. Walmart-Amazon has some missing values and slightly longer sentences. Abt-Buy has significant amounts of missing values, and is the most textual dataset. However, the sentence length difference between Abt and Buy is large.

[1] https://github.com/anhaidgroup/DeepMatcher/blob/master/Datasets.md

Table 4.3: An overview of what the rows in the five datasets can look like, as well as examples of matches/non-matches. Note that some of these examples are difficult, and therefore more likely to be false negatives/positives.

Amazon-Google

Match	Table	Title	Manufacturer	Price
True	A	peachtree by sage complete accounting 2007	sage software	269.99
	B	sage software peachtree complete accounting 2007 software for windows tax	finance software	249.95
False	A	hoyle card games (jewel case)	jungle software	399.0
	B	encore software 10027 hoyle board games (win 98 me 2000 xp)	NA	7.47

DBLP-ACM

Match	Table	Title	Authors	Venue	year
True	A	storage technology : raid and beyond	garth a. gibson	sigmod conference	1995
	B	tutorial on storage technology : raid and beyond	garth a. gibson	international conference on management of data	1995
False	A	o-o , what 's happening to db2 ?	richard swagerman , nelson mendonça mattos , mich(...)	sigmod conference	1999
	B	o-o , what have they done to db2 ?	(...)ielau , richard swagerman , nelson mendon (...)	very large data bases	1999

DBLP-GoogleScholar

Match	Table	Title	Authors	Venue	Year
True	A	application servers and associated technologies	NA	NA	2002
	B	advanced technology seminar : application servers and associated technologies	c mohan	19 thiee(...)	NA
False	A	hector garcia-molina speaks out	m winslett	nan	2002
	B	hector garcia-molina speaks out regarding startups , how life is getting harder , delta papers , cs in	m winslett	sigmod record ,	2002.0

Walmart-Amazon

Match	Table	Title	ModelNo	Category	Brand	Price
True	A	samsung clp-500rb opc drum	clp-500rb	printers	samsung	199.88
	B	samsung opc drum for clp-500 series clp-500rb xaa	clp-500rb / xaa	NA	samsung	156.44
False	A	hon 600 series two-drawer lateral file black	682lp	stationery & office machinery	hon	449.94
	B	hon 582lp - 500 series two-drawer lateral file 36w x28-3 8h x19-1 4d black	NA	NA	hon	NA

Abt-Buy

Match	Table	Name	Description	Price
True	A	garmin 010-10823-00 black nuvi 660 vehicle suction cup mount 0101082300	garmin 010-10823-00 black nuvi 660 vehi (...)	46.0
	B	garmin suction cup mount 010-10823-00	NA	24.84
False	A	sony silver cyber-shot digital camera dscw150	sony silver cyber-shot digital camera dscw150 8.1 (...)	NA
	B	sony cyber-shot dsc-w150 digital camera black dscw150/b	NA	NA

Chapter 5

Method

This section gives an overview of tools and libraries used, the experiments conducted and their respective method, as well as justification for methodical choices.

5.1 Tools

Different python libraries have been used to load and manipulate data, and to perform EM with the use of TPLM and AL.

Magellan (py_entitymatching)[1] is the library made by the Magellan project ([17]). It contains a wide range of entity matching methods, from blocking, sampling, generating similarity features, debugging, and matching using machine learning models. In our experiments, Magellan is used to automatically generate string similarities for the Hybrid approach described in subsection 5.3.3.

modAL[2] is an active learning framework, built on top of scikit-learn. In Hybrid it is used to assist in the implementation of the AL iterations.

scikit-learn[3] is a popular machine learning library, with different algorithms for classification and data processing. We utilize scikit-learn's Random Forest Classifier in Hybrid.

huggingface[4] is a popular Transformer framework, making developing Transformer

[1] https://pypi.org/project/py-entitymatching/
[2] https://github.com/modAL-python/modAL
[3] https://scikit-learn.org/stable/
[4] https://huggingface.co/

architectures easier. In our experiments it is used for a variety of tasks which has to do with the Transformer models, like loading the TPLMs models, tokenization of records, and the final training and testing.

pytorch-lightning[5] is a framework on top of the deep learning library pytorch. It streamlines and simplifies writing and running DL experiments. pytorch-lightning is used in conjunction with huggingface to preform the experiments.

5.2 Experimental Setup

To validate the results, each experiment has been run 3 times, and in chapter 6 the average of the three runs are reported. Each run of each experiment used the same random seed, such that all experiments can be reproduced later. In other words, the first run of each experiment used "1" as the random seed, the second run used "2", and the third used "3". The experiments were run on a single Nvidia V100 GPU with 16 GB of memory, on an Intel Xeon CPU with 28 cores.

In our preliminary experiments, RoBERTa [32] and DistilBERT [33] were both evaluated as TPLMs. RoBERTa received in most cases higher F1-scores, with slightly higher iteration time compared to DistilBERT. Brunner and Stockinger [2] analyzed recent transformer architectures for EM and RoBERTa received the highest F1-scores in general, and Li et al. [3] chose RoBERTa for the same reasons. By using the same model as the closest related work, we can more easily compare out result to theirs. Due to these factors RoBERTa has been the TPLM of choice in the experiments. Implications of using different TPLMs for AL is further discussed in chapter 7.

5.2.1 Hyperparameters

The TPLM in each experiment was trained on 12 epochs, with a batch size of 16. Li et al. [3] used a variable amount of epochs (10, 15, or 40) depending on the size of the dataset being tested on, and Kasai et al. [21] used 20. However, because of different hardware being used and our wish to simplify the experimental setup, we did not change the model parameters between experiments. 12 epochs was chosen empirically by testing the RoBERTa model with 200 randomly sampled examples. Fewer epochs often resulted in an initial F1-score of zero, making it futile to use the model in a query strategy, as it could not give any useful predictions. Using more epochs gave better initial results, however resulted in longer training times. Consequently, we chose to run all experiments on all datasets with 12 epochs, as the trade-off between longer running time and prediction performance seemed reasonable.

[5] https://www.pytorchlightning.ai/

The batch size of 16 is the same as [21]. Choosing a larger batch size resulted in errors during training due to the (sometimes) small amounts of training data. The optimization algorithm was Adam [21], with a linear learning rate of 3e-5 [3, 2].

More hyperparameters can be tweaked, and the fact that there is no benchmark to use when doing AL or EM with TPLM is a recommendation for further work, see chapter 7.

5.2.2 Tokenization

TPLMs take a token embedding as input, where the token has to be encoded from a single string. Therefore, one challenge is to convert a candidate record pair consisting of record a and record b, to one meaningful representation the language model can work with. At the same time preserving some of the inherent information in the data, such as the attribute name and value.

Following [3], we have serialized ($s(e)$) each record example (e) in the following manner:

$$s(e) = [COL]\, attr_1\, [VAL]\, val_1\, \ldots\, [COL]\, attr_k\, [VAL]\, val_k,\ e = \{(attr_i, val_i)\}_{1 \leq i \leq k}$$

$[COL]$ and $[VAL]$ are tokens which have been used to indicate to the model that the next value is the start of an attribute name or value respectively. Table 5.1 shows an example of such a tokenization of an example match from Walmart-Amazon.

Table 5.1: Example of a tokenization of the Walmart-Amazon example match presented in Table 4.3

Table	Encoded string
Walmart	COL title VAL samsung clp-500rb opc drum COL category VAL printers COL brand VAL samsung COL modelno VAL clp-500rb COL price VAL 199.88
Amazon	COL title VAL samsung opc drum for clp-500 series clp-500rb xaa COL category VAL nan COL brand VAL samsung COL modelno VAL clp-500rb / xaa COL price VAL 156.44

To tokenize two serialized candidate pairs, we have used AutoTokenizer from the huggingface library. This tokenizes the records specifically for the language model being used. In the case of RoBERTa it adds a special $< s >$ token to the beginning of the token to indicate to the model that it is a classification task.

5.3 Experiments - Query Strategies

Detailed below is the different query strategies representing the different experiments performed. The result of each query strategy can be viewed in chapter 6.

With the exception of Hybrid and Hybrid-Partition-2, all other strategies was initialized with an initial training set, $\mathcal{L}_i$, of 200 randomly sampled examples from the train set of the respective dataset. As these have been chosen randomly for each run, the rate of positive examples approximates the positive rate found in the training set, see Table 4.2.

Each active learning strategy started by training the TPLM on $\mathcal{L}_i$. The query strategy then chose new examples from the unlabeled pool $\mathcal{U}$, which were labeled by the Oracle. The new examples were then removed from $\mathcal{U}$, and added to $\mathcal{L}_i$, from this point just called $\mathcal{L}$. The TPLM model was then discarded, and a completely new model trained on $\mathcal{L}$. The query strategy could then select new examples with the updated model.

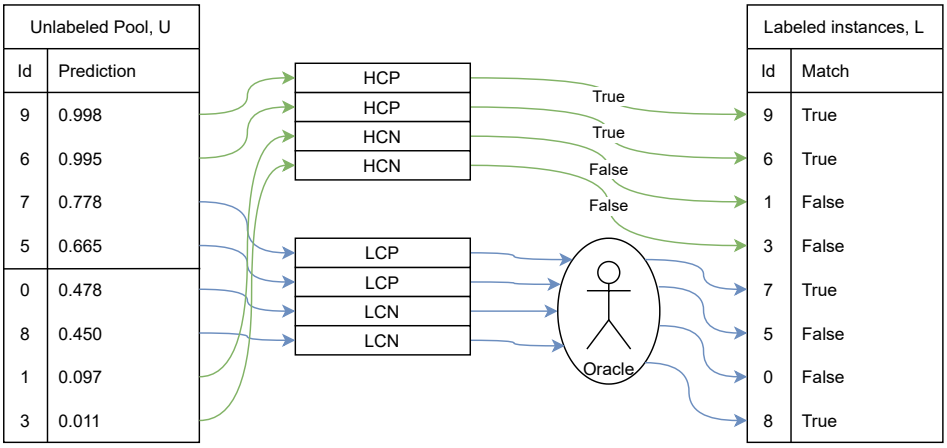
For each iteration the model was asked to predict the test set, the results of these predictions are reported in chapter 6. This prediction step of the test set is independent of the AL loop and only performed in order to be able to report results for each iteration. The test set was never changed.

Like this the iterations continued until a stopping criteria was met, which in our experiments have been the number of iterations necessary to reach 1000 labeled examples, given the model is able to select K examples in each iteration. This amount to $(1000 - 200)/40 = 20$. The number 1000 have been chosen in order to make the running of all experiments finish within a reasonable amount of time.

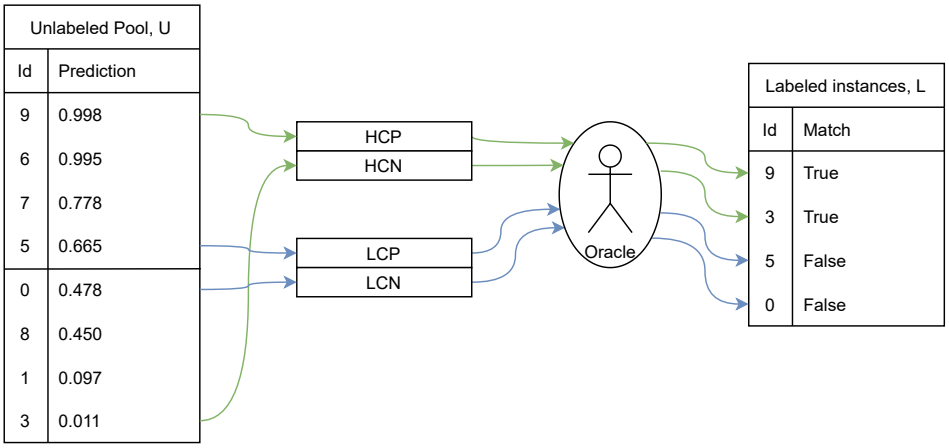
Common for all experiments was that $K = 40$ examples have been labeled manually in each iteration. This value was chosen empirically following [21], and from computational time limitations. By decreasing this value, a higher amount of iterations, and therefore a longer total runtime, would be needed to complete the experiments. The Oracle have in our experiments been the "matching" column of the specific dataset's gold standard. I.e. when an example has been labeled "manually" by the Oracle, the specific example's gold standard label was used. The "matching" field was not visible for the model when doing prediction.

5.3.1 Baseline

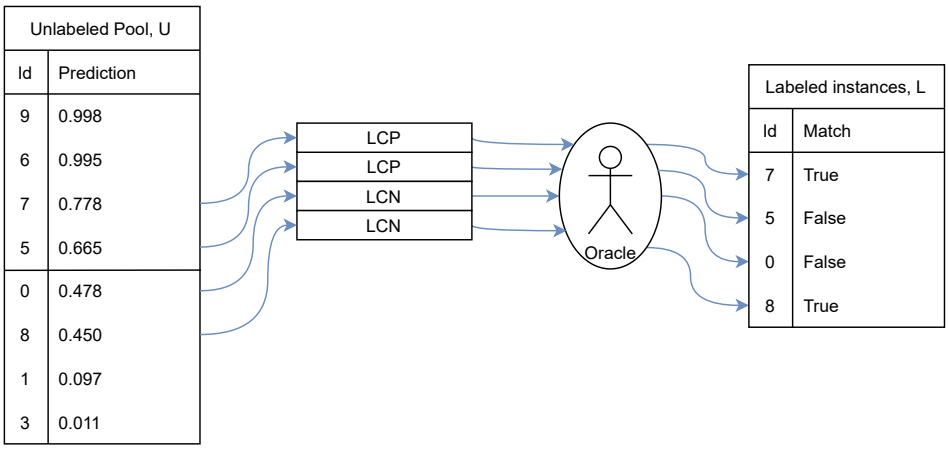
We have implemented Baseline as an equivalent to a standard supervised strategy of randomly labeling examples. In this way we can compared the performance of an AL strategy, to see if there are any improvements over random sampling. The baseline results have been retrieved by labeling and training on a random selection of examples,



(a) Partition-2. HCP/HCN predictions are assumed to be correct, i.e. matches/non-matches respectively. LCP/LCN are labeled by the Oracle.



(b) Partition-4. HCP/HCN and LCP/LCN are all labeled by the Oracle.



(c) Uncertainty. Only LCP/LCN predictions are labeled by the Oracle.

Figure 5.1: Comparison of Partition-2, Partition-4, and Uncertainty sampling. For each strategy in the example above, $k = 4$ examples are labeled by the Oracle. Partition-2 additionally labels 4 examples unsupervised. In the full experiment $k = 40$.

using Numpy's *random.choice*() method without replacement. This method initially trains on $\mathcal{L}_i$, then chooses 40 random examples which are labeled by the Oracle and added to $\mathcal{L}_i$. The model can now train on its updated labeled set $\mathcal{L}$. This continues in each iteration, and simulates the query strategy of naively labeling random examples.

5.3.2 Partition Sampling

In January of 2021 when we started this thesis Kasai et al. [21], see chapter 3, were one of few which had conducted experiments using AL with DL for EM. As their query strategy was one of few specifically aimed for DL and EM, we wished to include their strategy in our experiments. They argued that by selecting both high-confident and uncertain examples, the model would be more resistant to overfitting while improving precision and recall. In the following sections this strategy is referred to as Partition-2. Figure **??** shows an example of the two partition-strategies as well as uncertainty sampling.

Partition-2

Partition-2 tries to select a balanced set of examples comprised of k high-confidence positive (HCP), k high-confidence negative (HCN), k low-confidence positive (LCP), and k low-confidence negative (LCN). In our experiments $k = 20$. Note that low-confidence examples are also know as uncertain examples, and this terminology might be used interchangeably.

Partition-2 works by partitioning each example in $\mathcal{U}$ into one of two subsets, based on whether the model predicts the example to be a positive or negative match. From each of these subset k high-confidence and low-confidence examples are selected. This results in the four partitions HCP, HCN, LCP, and LCN. The confident examples are the examples the model was most certain about in its prediction, while the opposite is true for the least confident examples. In practice, the two subsets can be sorted based on the prediction score. The k topmost and bottom examples are the high-confidence and low-confidence examples, respectively. HCP and HCN were automatically labeled in an unsupervised manner, while LCP and LCN were labeled manually by the Oracle. This results in 40 examples which are manually labeled, and a total of 80 labeled examples.

Partition-4

We have implemented a similar query strategies which only differ in how it handles the high-confidence partitions. While the low-confidence examples are manually labeled by the Oracle, Partition-2 use the model's predictions to label the high-confident ones. This

means it is prone to label data wrong, however it doubles the number of labeled data returned for each iteration.

The Partition-4 query strategy's implementation have been almost equal to that of Partition-2 however the HCP and HCN examples have been labeled manually by the Oracle. To accustom this, k is set to 10 instead of 20. This means that the strategy is not prone to label examples incorrectly, however it only labels half the amount of training data in each iteration. We chose to include both experiments to compare the effect of labeling examples in an unsupervised manner.

5.3.3 Pre-select Examples With Traditional Machine Learning and Active Learning

One of the experiments completed in this thesis have been to test whether classic machine learning methods can be used to improve and speed up the initial labeling effort required when training a Transformer model such as BERT. In the following sections this strategy is referred to as Hybrid Active Learning, or Hybrid for short.

Even though TPLMs such as BERT can give correct predictions out-of-the box without any specialized training, this is not true in many cases.

This can be a problem if one are to use a TPLM in the AL iterations, when using query strategies which are dependent on the predictions of the model. If the model is not performing at a sufficient level, it is not able to make predictions that can be used by the query strategy [27]. One example of such a situation is when the model predicts all examples as e.g. negative matches. The query strategy can not choose positive matches to be labeled, thus likely resulting in an even more unbalanced training set. To fix this, the TPLM needs to be trained on an initial set of labeled examples.

Deep learning models require large amounts of training data [21], while traditional ML models often can perform well with much less. The motivation is therefore to use a traditional ML model and AL to choose a balanced, but at the same time varied training set for the transformer model. This might make The transformer model achieve a higher F1-score initially, as a result giving more accurate predictions and the following query strategies will hopefully be more successful.

The motivation is somewhat similar to the *Imbalanced-practical* scenario in Ein-Dor et al. [27], where a user "manually" tries to sample a set of positive examples from a highly imbalanced dataset. Whereas they use simple regex rules, which ultimately result in a sample of very similar examples, this thesis use AL with a classical ML model.

Hybrid

Hybrid is the name we have given the strategy of using a Random Forest (RF) model with Partition-4 to create the initial training data for the TPLM, $\mathcal{L}_i$. The method does AL with an RF model, starting with an initial random labeled set which is an order of magnitude smaller than what is normally required when using a TPLM. In a practical scenario, the domain expert acting as an Oracle would use this method to label as many examples as they have time for. When stopping, the updated set $\mathcal{L}_i$ would then be used to train a TPLM which would predict $\mathcal{U}$. The next paragraph introduces the implementation in detail.

For each dataset, the most discriminating field is manually selected, e.g. "name" in a product dataset, and Magellan is used to generate string similarity features automatically for this field. 10 random examples are labeled and used to train the initial Random Forest model, and removed from the unlabeled pool $\mathcal{U}$. Next, the labels of all remaining examples in U are predicted by the model, and 4 examples are selected and labeled according to Partition-4, however here $K = 4$ and not 40. The labeled examples are saved, and later used as training data for the TPLM. The iterations continue until all examples are labeled. To be able to compare this method to the other strategies, this method has been tested for the same iterations as the rest.

Hybrid-Partition-2

Hybrid-Partition-2 is a combination of Partition-2 and Hybrid. The initial training examples are selected from the first 200 Hybrid examples, while the next iterations use normal Partition-2 with the TPLM to select new examples to label. The motivation behind the implementation of this experiment was hypothesizing that the improved F1-score of Hybrid with 200 examples could help the query strategy choose better examples during AL. In turn further improving the model, its predictions and its F1-score.

5.3.4 Uncertainty Sampling

Uncertainty sampling is a common query strategy in AL, and is described in detail in chapter 2. For each iteration the query strategy selects the K most uncertain examples, based on the models prediction of the unlabeled set L. The chosen examples are the K examples closest to 0.5 in the models prediction. In previous work, particularly considering AL with classical ML methods like RF, uncertainty is usually one of the most popular and best performing strategies [10]. Consequently, evaluating uncertainty sampling in our AL experiments with TPLMs seemed natural. In addition, Uncertainty is very similar to Partition-4 and Partition-2 making it possible to discuss the differences in their results.

Chapter 6

Results

In this section the results of the experiments are presented, along with a brief comparison. As mentioned in chapter 5, all experiments have been run 3 times, and the average result is what is presented.

First the results of the different experiments with respect to F1-score are presented, then some selected results with respect to time usage and standard deviation are presented. The results are discussed in detail in chapter 7.

All query strategies, except Hybrid and Hybrid-Partition-2, have been deterministically seeded with the same seed and were therefore initially trained on the same randomly sampled examples. They therefore started with the same F1-score, as can be seen in the figures. In later iterations the query strategies selected different examples to label, and their scores will consequently differ. As explained in chapter 5, Hybrid and Hybrid-Partition-2 get their examples from the classical Random Forest model, and therefore start at a different score.

6.1 Query Strategies Compared With Respect to F1-score

Figure 6.1 show the result of the experiments with respect to F1-score over an increasing number of labeled examples. The number of labeled examples is equal to the size of the training set for all experiments, except Partition-2 and Hybrid-Partition-2 where the training set is comprised of supervised and unsupervised labeled examples, as explained in chapter 5. For all plots, Baseline show the random query strategy in blue. The red stippled Baseline-Max line show the score when all available training data have been used to train the TPLM. Since this is the maximum score this model can achieve, it is

also the target F1-score for the AL strategies.

Table 6.1: Initial F1-score at 200 labeled examples. Highest score is highlighted. One can see that pre-selecting examples results in a higher initial F1-score. Hybrid and Hybrid-Partition-2 use this strategy, while the other query strategies select the initial training set randomly, same as Baseline

Dataset	Baseline	Hybrid
Amazon-Google	0.249	**0.528**
Abt-Buy	0.497	**0.631**
Walmart-Amazon	0.258	**0.603**
DBLP-ACM	0.954	**0.970**
DBLP-GoogleScholar	0.903	**0.914**

Table 6.2: Final F1-score after 1000 labeled examples for Baseline and all query strategies. Highest score is highlighted. One can see that Partition-2 and Hybrid-Partition-2 have the highest final F1-scores. HP-2 = Hybrid-Partition-2, P-4 = Partition-4.

Dataset	Baseline	Partition-2	HP-2	Uncertainty	P-4	Hybrid
Amazon-Google	0.591	**0.706**	0.696	0.679	0.665	0.645
Abt-Buy	0.833	**0.914**	0.897	0.893	0.889	0.864
Walmart-Amazon	0.793	0.866	**0.873**	0.572	0.851	0.768
DBLP-ACM	0.977	**0.990**	**0.990**	0.988	0.987	0.981
DBLP-GoogleScholar	0.938	**0.953**	0.951	0.950	0.948	0.930

From the figures one can see that all query strategies performed similar or better than the random baseline. Overall, Hybrid-Partition-2 and Partition-2 were the best performing ones, and had consistently among the highest F1-scores.

Partition-2 had on average a 0.06 higher F1-score compared to Baseline after 1000 examples, ranging from 0.013 to 0.115 higher F1-score across the 5 datasets. The final F1-score for all the strategies can be seen in Table 6.2.

Hybrid always started with a higher initial F1-score, but was quickly surpassed by the other strategies, and ended up at a lower score. In Walmart-Amazon it had a 0.35 higher F1-score than the other query strategies at the initial starting point of 200 labeled examples, however ended up 0.10 behind Partition-2 at 1000 examples. In all datasets, except DBLP-GoogleScholar and Walmart-Amazon, Hybrid performed better than the random baseline after 1000 labeled examples.

Hybrid and Hybrid-Partition-2 started on average with a 0.16 higher F1-score across all datasets, with an initial F1-score ranging from 0.011-0.345 higher than the others. Table 6.1 showcase the initial F1-score for the two starting strategies.

Uncertainty sampling often reached a high F1-score, but at the same time had a higher variance, especially in Walmart-Amazon and Abt-Buy. Partition-4 consistently had a lower F1-score than the others, but it seemed to be the most stable strategy with the lowest variance.

Also from the graphs one can see that Partition-2 and Uncertainty initially improved the F1-score quicker, while the rest had a slower improvement.

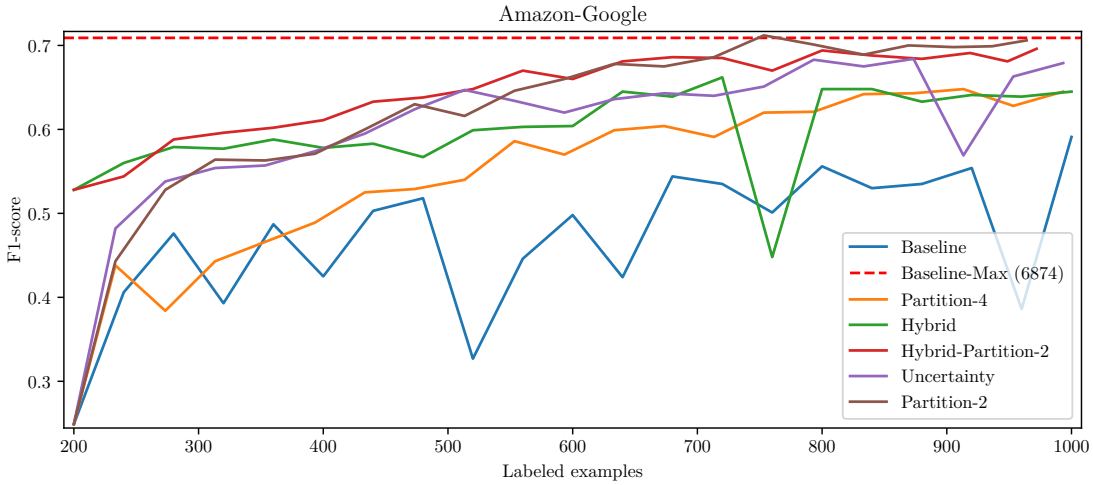
6.1.1 Max F1-score

Notably from the experiments was the fact that several strategies reached the Baseline's max F1-score after only a fraction of the examples had been labeled. For Partition-2, this happened after only 9.1% of the data was labeled on average. These numbers are presented in Table 6.3

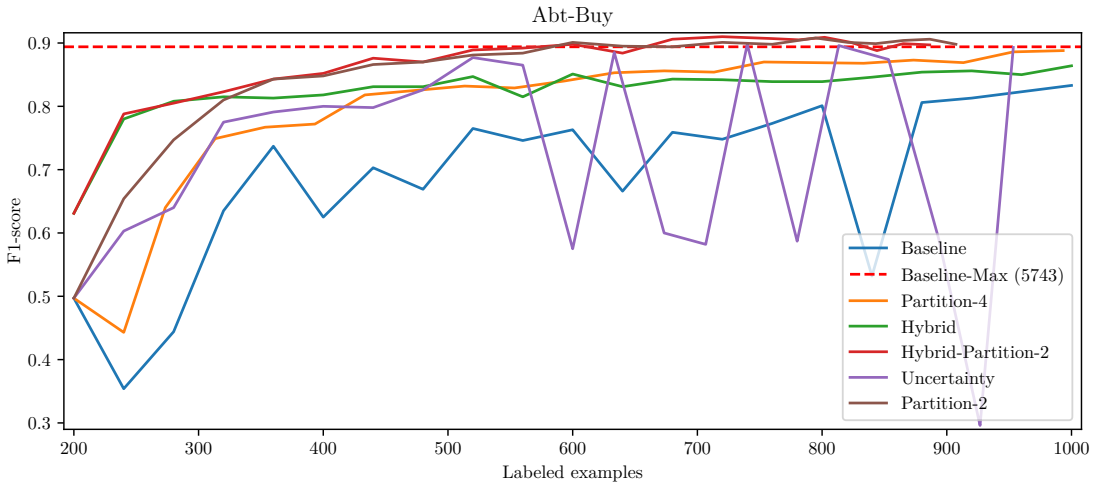
Table 6.3: The table show when the Partition-2 query strategy reached or surpassed the F1-score of Baseline when Baseline was trained on the entire train set. TS = Train Size.

Dataset	Baseline [Max TS]	Partition-2 [TS]	Partition-2 [% of max TS]
Amazon-Google	6874	730	10.6
Abt-Buy	5743	576	10.0
Walmart-Amazon	6144	662	10.8
DBLP-ACM	7417	640	8.6
DBLP-GoogleScholar	17223	960	5.6

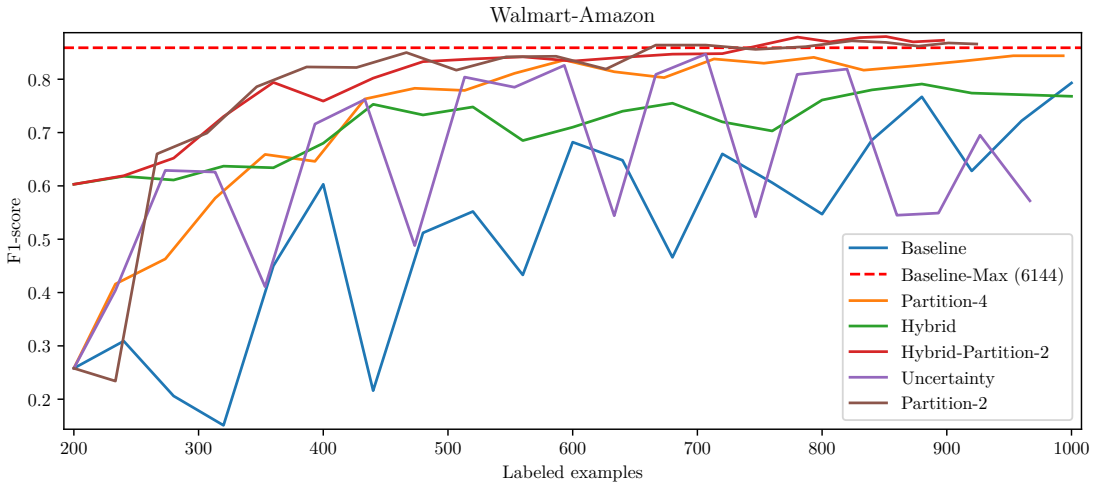
The following are plots displaying F1-scores for every method on all datasets.



(a)

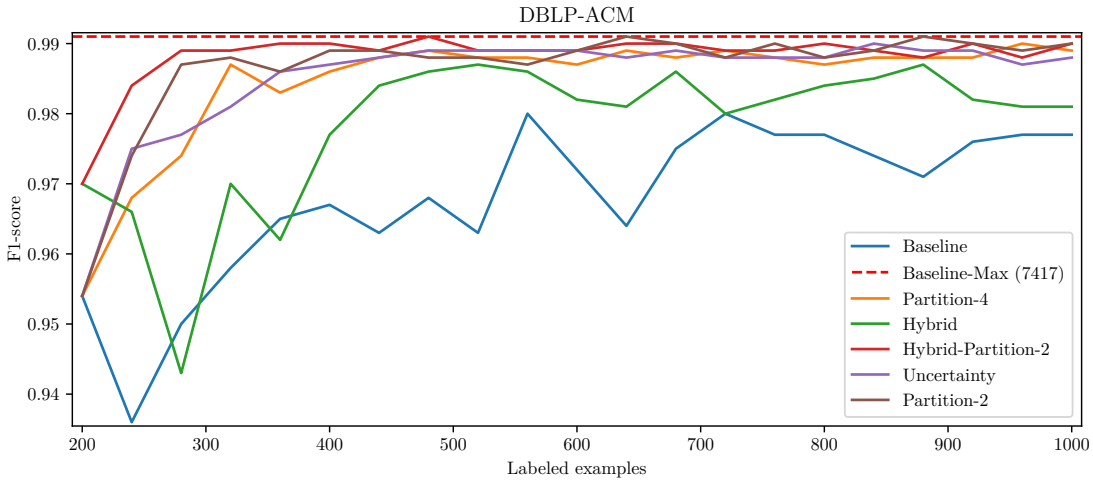


(b)

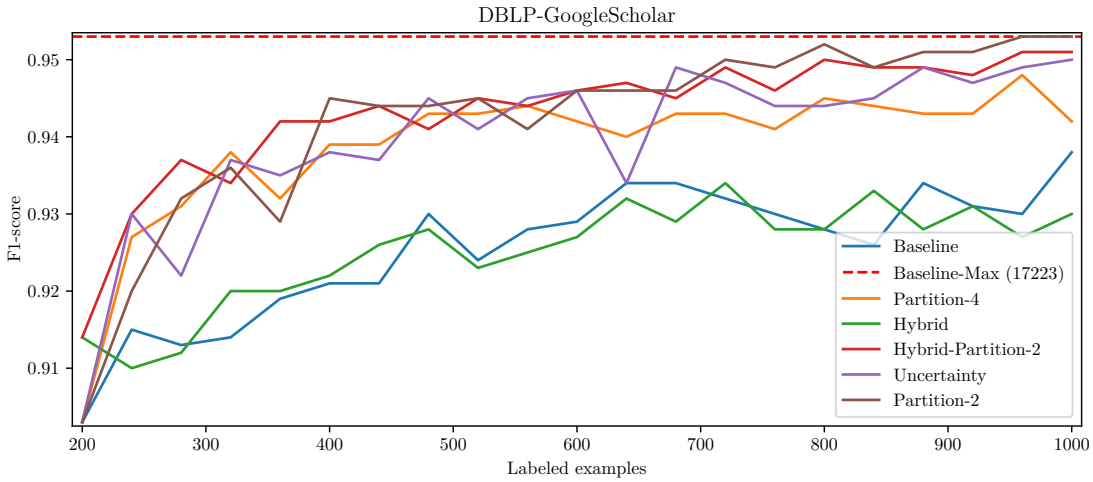


(c)

Figure 6.1: (figure continues on the next page)



(d)



(e)

Figure 6.1: Iterative F1-score for different active learning query strategies with RoBERTa as the TPLM, on all datasets. All experiments started with 200 sampled examples, and each query strategy tried to manually label 40 examples in each iteration, with a maximum of 1000 examples over 20 iterations. The red stippled line of Baseline-Max shows the F1-score of the model trained on the maximum amount of available examples in the training set (show in parenthesis).

6.2 Variance

Figure 6.2 shows the F1-score standard deviation for the Walmart-Amazon dataset. Uncertainty and Baseline have higher values which varies a lot more. The effect of this is also visible in the F1-scores, where both are susceptible to scores that moves a lot between iterations. The reason for this is discussed in chapter 7. The Hybrid strategies have a consistently lower initial variance compared to the other datasets. More variance results can be found in the Appendix.

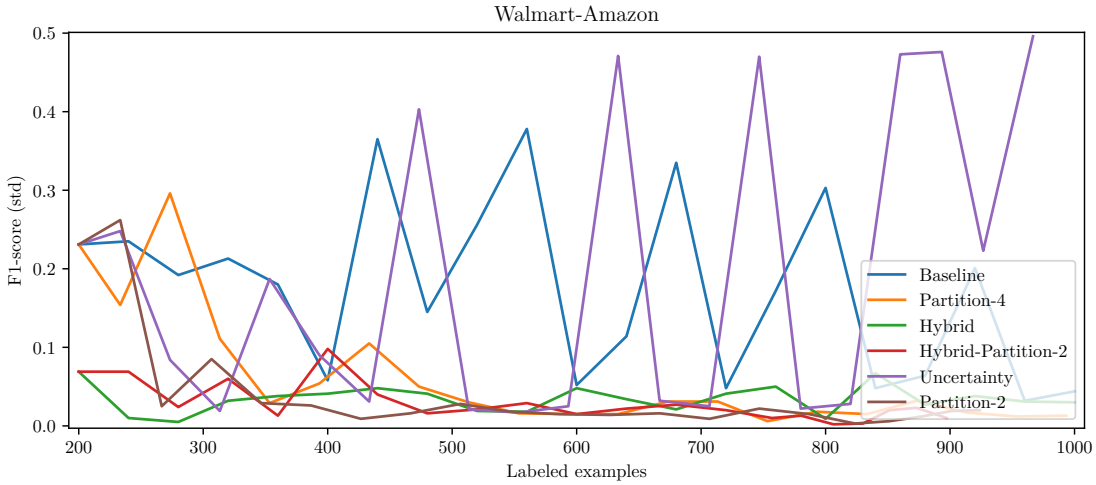


Figure 6.2: F1-score standard deviation for different active learning query strategies with RoBERTa as the TPLM, on the Amazon-Google dataset. The standard deviation have been computed from 3 test runs. The query strategies respective F1-score can be seen in Figure 6.1c.

6.3 Time Usage

Figure 6.3 shows the iteration time usage for the different query strategies with the Transformer model on the Amazon-Google dataset. The other datasets showed similar results. Unfortunately the time usage depended on the current load in the test environment, so the results have to be interpreted with this in mind. As an example, all the AL strategies were expected to have the same initial iteration time, however there was a difference of around 60 seconds.

From the graph one can see that all strategies seem to increase linearly with the number of labeled examples, however the Partition-2 strategies increase at a higher rate.

As described in chapter 5, Hybrid has already selected its examples using Random Forest

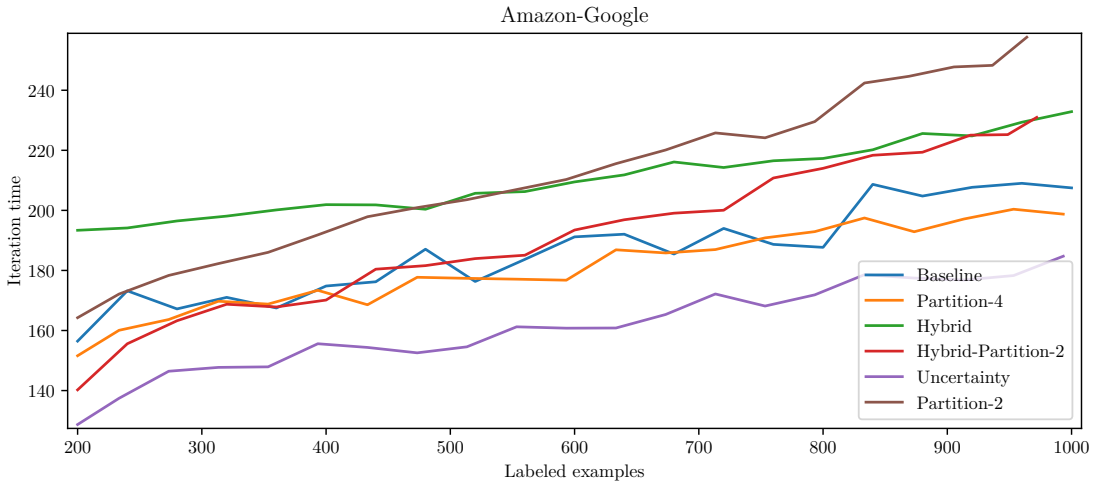


Figure 6.3: Iteration time for different query strategies with RoBERTa as the TPLM, on the Amazon-Google dataset. Each experiment ran in a different hardware environment, so the iteration times can not be directly compared. This is further explained in chapter 5.

as the model in the query strategy. RF was found to use around 1-2 seconds to query examples, for all datasets. This is not depicted here due to the fact that compared to the other strategies it is significantly smaller and would make the graph harder to read. More iteration time results can be found in the Appendix.

Chapter 7

Discussion

This chapter discusses the main findings from the results chapter.

7.1 Query Strategies

The first research question is about what AL query strategy for TPLM results in the highest F1-score. As one can see from the results, *all* query strategies performed on average better than the baseline, however some consistently did better than others. In the next section the results of the different query strategies will be discussed. We will highlight that Partition-2 and Hybrid-Partition-2 performed best with respect to F1-score per labeled example, but had different strengths and weaknesses related to how they work.

7.1.1 Partition Sampling

From the results in chapter 6 one can see that the query strategy Hybrid-Partition-2 and Partition-2 performed best. Hybrid-Partition-2 had the highest initial scores, while Partition-2 slightly outperformed it in the last iterations. As described in chapter 5, the partition sampling strategy used by both splits the unlabeled pool into 2 partitions of positive and negative matches, based on the model's prediction. It then labels the most confident examples in an unsupervised manner, while the Oracle manually labels the uncertain examples.

After 1000 labeled examples they have the highest F1-score among all the query strategies, for all datasets. One can reason from the results that even from its first iterations,

partition sampling manages to select a balanced set of informative examples which make its F1-score increase rapidly while maintaining a low variance compared to the other strategies.

Unsupervised Labeling

It is interesting to compare Partition-2 to both Partition-4 and Uncertainty, which are two very similar query strategies, except they do not label examples automatically in an unsupervised manner.

Partition-4 also selects uncertain and high-confidence examples, however all examples are manually labeled, whereas Partition-2 only manually labels the uncertain ones. Including the unsupervised labeled examples, Partition-2 can train on twice as many new examples in each iteration. Even though some of the unsupervised labeled examples are labeled incorrect, from the results one can see that this drawback is less significant than the positive side of more labeled examples.

Uncertainty is similar to Partition-2 in that only the uncertain examples are labeled. They both have the highest initial increase in F1-score, suggesting that this is the main reason for the improvement. However, Uncertainty seems to be susceptible to very varying results, as can be seen in Figure 6.2. One possible reason for this is that it selects too many difficult, non-representative outliers, and learn patterns that do not hold for the rest of the data. In some iterations the Uncertainty model was not able to make correct predictions, and ended up with an F1-score of 0. This can for example be seen in Abt-Buy at the 600 mark. This highlights the improvement in selecting a balanced set, including the high confidence examples.

7.1.2 Pre-selection of Initial Train Data

Two of the experiments use a hybrid form of AL, where the initial training examples are selected by using AL with a classical ML model (AL-ML), rather than by randomly sampling examples. Common for both Hybrid and Hybrid-Partition-2 is the fact that they initially have a higher F1-score compared to the other methods. One possible reason for this is that the AL-ML model has managed to select informative examples for the Transformer, and essentially kick-started the AL process for the TPLM.

If this was true then it could mean that there is no need for AL with a TPLM (AL-TPLM), one could just use the simpler and an order of magnitude faster AL-ML strategy to select examples for the Transformer. In the end, both the AL-ML and AL-TPLM have the same pool of unlabeled examples, and could end up selecting the same ones. The Transformer would then train with those examples, and perform similar to the AL-

TPLM strategies. This is what the Hybrid experiment test, however from the results we can see that this hypothesis does not hold, it consistently under-performs compared to the AL-TPLM strategies. This suggests that the improvement one can see compared to e.g. Baseline could come from some other factor like an increase in positive examples to learn from.

Hybrid-Partition-2

Hybrid-Partition-2 use the same query strategy as the other best performing strategy, Partition-2, the only difference between them is that the initial 200 examples have been pre-selected by the AL-ML strategy. From the results one can see the Hybrid-Partition-2 consistently had a higher F1-score in the first couple of iterations, before being caught up to by Partition-2. The initially higher F1-score is mainly a result of higher recall, see Appendix 8.3.6.

Since the AL-ML query strategy behaves much like Partition-4, it tries to select an even amount of positive and negative examples. This means the initial 200 examples for Hybrid-Partition-2 normally have a higher distribution of positive examples compared to Partition-2. This positive distribution ranges from 25-50% and 10-20%, respectively. Figure 7.1 show how the rate of positive examples in the train set develops in each iteration for Partition-2 and Hybrid-Partition-2.

By starting with a more balanced initial train set, Hybrid-Partition-2 can achieve a higher recall in the first iterations. However, Partition-2 achieves the same F1-score after 2-10 iterations, even with a smaller ratio of positive examples in its training set.

In a few cases Partition-2 slightly surpasses Hybrid-Partition-2 in the later iterations. This suggest that a higher rate of positive examples might not be beneficial when the amount of training data is higher. What can be seen in the Figure 7.1 is that Hybrid-Partition-2 often have more positive examples and a more balanced distribution in its training set, whereas Partition-2's distribution is closer to the source data. When the training data is more abundant, this discrepancy between the training and test set distribution could be an unwanted side effect of starting with a balanced initial set. A possible mitigation to this problem is to use a specialized query strategy which selects more negative than positive examples.

A significant downside to using Hybrid-Partition-2 is the fact that it has extra overhead related to performing two separate AL iterations. It is however possible to reduce the time by using EM libraries such as Magellan. Magellan can do a lot of the heavy work of e.g. selecting suitable string similarity measures and generating features.

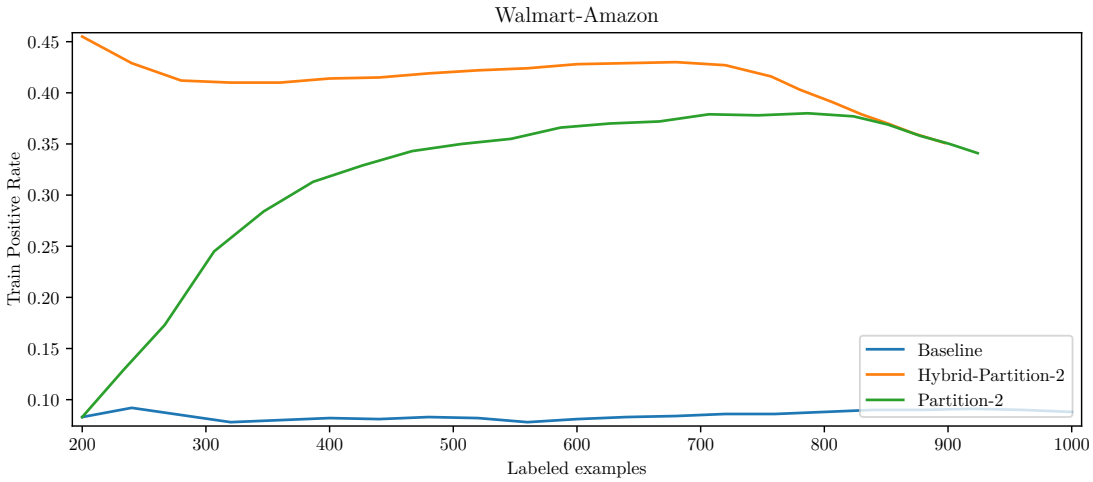


Figure 7.1: Train set positive rate for Hybrid-Partition-2, Partition-2, and Baseline on the Walmart-Amazon dataset. One can see that Hybrid-Partition-2 has a significantly higher positive rate in the beginning. The reduction in positive rate at 700-800 is from the fact that the query strategy has exhausted the positive examples it manages to find in the pool $\mathcal{U}$. The result is that it starts to draw more negative examples, and the train positive rate drops.

7.2 Using Active Learning As A Strategy for Entity Matching with Transformers In a Limited Environment

Research question 2 is about how AL with TPLM performs with respect to time consumption. This section will highlight how AL-TPLM improves F1-score significantly faster than random labeling of examples, and that this will reduce the time consumption simply from the fact that less examples needs to be labeled to achieve an equal performance. In addition, this section will discuss how one can reduce the time consumption further.

7.2.1 Faster Convergence

Figure 7.2 compares the result of Partition-2, Hybrid-Partition-2, and AL-ML against an extended test of the random labeling strategy Baseline on the Amazon-Google dataset. In addition to showing the result after training on 100% of the data, it also show how the TPLM performed after 50% and 25% of the data was labeled. The results can be seen in the stippled line of Baseline-1/2 and Baseline-1/4, respectively. Both query strategies beat Baseline's 1/4-score before 300 examples were labeled, which is 16.3% of the 1718 examples for Baseline. Partition-2 beat Baseline-1/2 at 720 (21.9%), and Baseline at

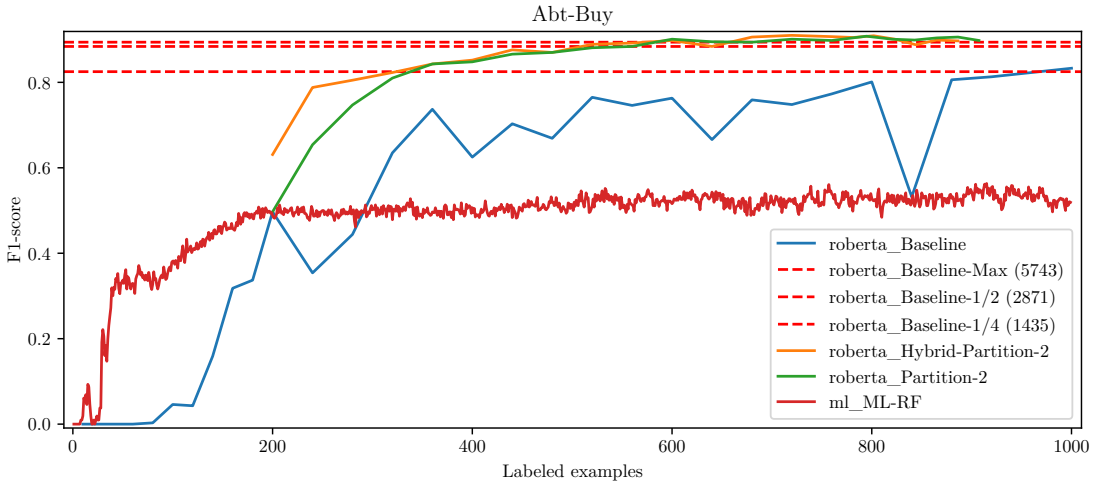


Figure 7.2: F1-score comparison for Walmart-Amazon. The red lines indicate the F1-score for Baseline, when labeling 25%, 50%, and all the training data. The bottom line is for 25%, the middle for 50%, and the topmost is 100% of the data. ML-RF show the score from the classical RandomForest model trained in the Hybrid model's pre-step, as explained in chapter 5.

750 (10.9%).

Comparison to State-of-the-Art in Non-Active Learning

Table 7.1 compares Partition-2 to current state-of-the-art in EM, DITTO [3]. DITTO does not employ AL, but employs other strategies such as data augmentation, manual knowledge injection, and token summarization. The results are taken after 1000 examples have been labeled. On average, Partition-2 has a 0.095 higher F1-score, or 13.7% better result. This suggest that currently, the strategies employed by DITTO do not outperform AL. Like AL, knowledge injection is also a manual process. The time spent doing knowledge injection might therefore be better spent doing AL.

TPLMs have recently managed state-of-the-art performance in many deep learning tasks, and lately in the field of EM. The results presented above strongly suggest that AL can be an effective way to quickly improve the prediction performance when doing EM with TPLMs. This is especially the case when there is a lack of labeled training data, or even improve a model if more unlabeled data is available. The high increase in F1-score in the first iterations suggest that only performing a few iterations will yield significant improvements for the model. In the end, AL with TPLMs can help reduce the need of labeled examples by an order of magnitude compared to only using TPLMs. In a practical setting, the faster convergence of AL directly results in less time spent labeling

Table 7.1: F1-score comparison of Partition-2 to state-of-the-art in EM, DITTO [3]. The table shows F1-scores after 1000 examples have been labeled and trained on. Note that DITTO is not an AL framework, but do data augmentation, manual knowledge injection, and token summarization. They use the same TPLM (RoBERTa), with a maximum of 40 epochs.

Dataset	DITTO	Partition-2
Amazon-Google	0.608	**0.706**
Abt-Buy	0.793	**0.914**
Walmart-Amazon	0.654	**0.866**
DBLP-ACM	0.979	**0.990**
DBLP-GoogleScholar	0.922	**0.953**

for the researcher.

7.2.2 Time and Resource Limitations

The experiments in this thesis emulate an environment where there are time and resource limitations. More specifically, there is a requirement that the experiment do not take "too long" time. In a human-in-the-loop situation such as AL, it is preferable that the Oracle, the human domain expert, must maximise his or her time use. In a real-life situation the expert is likely an expensive asset for a company, with a high salary and many other tasks on their schedule. To keep him or her doing manual entity matching labor, which could be a very trivial task for the expert, is not desirable. For this reason, the time between each labeling iteration must be short.

As described in chapter 5, 12 epochs was chosen empirically as it resulted in a reasonable iteration time, averaging from around 2.5 minutes to 8 minutes in the experiments, and the model often received a high, stable F1-score. With more up-to-date hardware the running time can be shorter, or optionally by using fewer epochs, the running time can be reduced significantly.

Smaller Transformer Model

Another way to reduce the iteration time is to use a smaller TPLM. As reported in Brunner and Stockinger [2], RoBERTa had a running time double of the smaller TPLM *DistilBERT*.

One important difference in the experiments done in this thesis versus the above paper is the number of training data in each experiment. While they use the full training set,

this thesis only use up to 1000 examples in the active learning iterations. In the smallest dataset Abt-Buy, this is 17.4% of the total training examples. Less training data result in a shorter training time for the TPLM. This might make it so that the actual time difference makes up for the relative difference between the two models.

For example, in an AL situation, allowing a bigger model to train for 4 minutes could be preferable to a smaller model training for 2 minute, if the performance gain was big enough. Even though the bigger TPLM in this case use double the time, 2 minutes might not matter much when doing AL in practice. On the other side, this situation might change if the dataset is significantly bigger, and the time difference is e.g. 10 minutes against 20 minutes or more.

Preliminary tests in this thesis suggested the time improvement could be significant, however the F1-score decreased with the smaller TPLM. In Amazon-Google DistilBERT used from 60-110 seconds less time in each iteration, however then final F1-score was 0.05 lower than RoBERTA. Increasing the number of epochs could reduce this gap somewhat, however the running time would increase as well.

In the end, the decision of which TPLM to use in practice is a trade-off between prediction performance and training time. Exploring trade-offs between different TPLMs and model parameters for AL is discussed further in the future work section.

7.3 Challenges

The third research question is related to the challenges of combining AL with TPLM for EM. Some important challenges we found are discussed below. In addition, other challenges we faced when doing EM and AL is also discussed.

7.3.1 Unsupervised Labeling

The best performing query strategy Partition-2 has an innate issue with how it handles the high-confidence examples. In a practical real-life situation those examples cannot be treated as a gold standard, as it most likely contains incorrectly labeled examples. Depending on how strict the EM requirements are, they might have to be handled as part of the rest of the unlabeled data. However, if the EM requirements are less strict, and some false positives are accepted, this might not be a problem.

It might also perform worse when the availability of data is low. In a situation where the dataset is small and very imbalanced, incorrectly labeling a positive example could have a big impact on the model. This comes from the fact that there would be few positive examples to learn from.

7.3.2 Choosing a Transformer Pre-trained Language Model

An important decision when doing AL-TPLM is to decide which TPLM to use, and what hyperparameters to use for that specific TPLM. There exists an increasing number of different TPLMs, pre-trained for different objectives. E.g. SciBERT [39] and BioBERT [40] are TPLMs trained on scientific and biology corpora, respectively. Currently the open source library *huggingface* has a collection of over 9,000 different TPLMs[1].

To the best of our knowledge, there are no TPLM specifically made for EM, and while some small efforts have been made to benchmark different TPLMs on the EM task [2], it is not comprehensive, or performed with AL in mind. TPLMs can have a wide variety of sizes, which closely correlates to training and inference time, where a smaller model will in general be faster. DistilBERT is e.g. 40% smaller and 60% faster than BERT [33].

Consequently, new applications of AL-TPLM for EM have no solid data to base the choice of TPLM and hyperparameters on. A comprehensive benchmark of different TPLMs and hyperparameters, in addition to the creation of a TPLM specifically for EM, is suggested as future work.

7.3.3 Review of Practical Application

A related topic is how using AL compares to non-AL strategies in practice. This thesis have showcased how AL can improve the F1-score, and suggested AL likely decrease the total labeling time in practice. However, a more detailed review and comparison of practical applications of EM strategies could help focus further work in the EM field.

This can be related to total labeling time, but other considerations are relevant when EM is applied. In a setting where e.g. a company need to be absolutely certain that all matches are correct and non are missed, a "black-box" solution like TPLM might not be an option. It might be a requirement for the company that an expert actually reviews and validates *all* matches. This setting is especially fit for AL solutions, and open new opportunities and challenges.

One possible optimization that can be done is to have the expert label *several* examples at once. Instead of the model querying the Oracle to label single examples, it could query the Oracle to label suggestions for regular expressions for matches. If the Oracle validates the regular expression, the model would then run that expression on all the unlabeled examples and label them accordingly. An example for a regular expression could e.g. be that every candidate record pair which share "MBR-10" in the attribute "name" is a match.

[1] https://huggingface.co/models

Depending on the EM setting, there might exist a restriction that there is a one-to-one match between two records. In other words, only one record in dataset A can match one record in dataset B. The situation that a single record is not matched with several records need to be handled. The EM task would then include finding a record's *most likely* match, and remove all its other candidate pairs. This problem could easily be translated into an active learning setting. The Oracle would not be queried with single examples to label, but with all the possible matches for a given record.

These are just two situations that need to be handled in a practical application of EM, however many more likely exists, with different implications and restrictions. Preferably the review would incorporate a test framework, such that current and future strategies could be compared. A detailed review of practical applications of EM is suggested as future work.

7.3.4 The Need of an Interactive Expert

As mentioned in the previous section, depending on the dataset, a domain expert might be needed as the Oracle. Not everyone can label a domain specific dataset, but the availability of an expert is probably one of the biggest hindrances for AL. Without someone who can label the data correctly, AL is not a suitable strategy.

There is also an added complexity inherent in AL that comes along with the interactive element. In practice one needs a sort of "test station", where the Oracle can sit in front of a screen and do the labeling. There also needs to be some sort of application which in a suitable way presents the examples, handles the actual labeling, and passes the new examples to the TPLM in each iteration. This complexity might be off-putting for companies, compared to the arguably simpler way of training a TPLM once with all available training data and use the resulting model.

In relation to this it would be useful to have a comprehensive review of practical applications of EM, such that both researches and companies can have a better understanding of the trade-offs related to different strategies.

7.3.5 Interpretability

When the Oracle is labeling examples, an interpretable model could help him or her to understand *why* the model is uncertain about a given example. However, deep learning models are known to be difficult to interpret [5]. A solution could be to explain *what* the model is uncertain about in a given candidate record pair. Some work has gone into developing EM explanation frameworks from general purpose deep learning explanation tools, such as LIME [41]. By having the model explain what it is uncertain about, it

could help the Oracle label examples quicker by highlighting the important parts in the record pair.

7.4 Alternatives

One alternative to AL is Transfer Learning (TL), where a ML model is trained on different labeled source datasets and then applied on the target dataset. With TL there is no need for a human expert. However, as reported by Kasai et al. [21], combining TL with AL can yield an additional increase in performance compared to only using TL. To our knowledge, a combination of TL, AL, and TPLM for EM has not yet been done, and we suggest this as a recommendation to future work.

A second alternative is to use unsupervised data augmentation methods, as in Li et al. [3]. By augmenting examples with known labels, one can further (artificially) increase the amount of labeled data. In this way one can improve the result when data is scarce, however, it requires additional computational time.

Chapter 8

Conclusion and Future Work

In this chapter the conclusion and recommendations for future work are presented. In addition, we present final remarks for using active learning for entity matching.

8.1 Conclusion

Through experiments conducted on several public entity matching datasets we have tested combining active learning strategies with Transformer pre-trained language models. When using active learning our results show significant reductions in required training set sizes in order to achieve state-of-the-art F1-scores. Of the query strategies tested, Partition-2 performed best on all datasets. However, all strategies achieved higher F1-scores than the random baseline, showcasing the potential of AL. When the labeling budget was only a few hundred examples, using AL with a classical ML model to obtain an initial training set for the TPLM yielded the best results. While there is a need of a domain expert, the time usage of each AL iteration can be small enough such that using AL with TPLMs in practice is a valid option to consider.

Research question 1 *What active learning query strategy for Transformer pre-trained language models results in the highest F1-score?*

From the experiments, we found that the best performing query strategy was a partition sampling strategy of manually labeling uncertain examples, and unsupervised labeling of high-confidence examples. In this way, the model can train on a balanced set of informative examples, while receiving a large set of "free" automatically labeled training data. This is how Partition-2 and Hybrid-Partition-2 queries for new examples.

Hybrid-Partition-2 yielded a higher initial recall compared to Partition-2. By selecting the initial training set by using AL with a Random Forest model, it could start the AL iteration with a more balanced training set. On average this resulted in a 0.1 higher initial F1-score. However, after 3-10 iterations Partition-2 caught up and performed with a similar F1-score.

Research question 2 *How does active learning with Transformer pre-trained language models perform for entity matching with respect to time consumption?*

The current state-of-the-art in EM is to use TPLMs, which are more versatile and improve the F1-score on public datasets significantly compared to previous machine learning methods. Our experiments have shown that when using AL with TPLMs, one can achieve the best performance of a baseline TPLM with an order of magnitude fewer labeled examples. Having to label significantly fewer examples means that employing AL with TPLM in practice would likely lead to a shorter total time spent doing EM. However, we leave it to future work to test and compare how an application of AL with TPLM would actually perform in practice.

The experiments have shown that using a relatively large model does not necessarily lead to an unmanageable AL iteration time, with iteration times spanning from 2-8 minutes. In addition, it is possible to further reduce the time spent labeling by using a smaller TPLM, reducing the amount of epochs, and/or by using faster hardware. This might be necessary when working with larger datasets.

Research question 3 *What are the challenges with combining active learning and Transformer pre-trained language models for entity matching?*

As described in chapter 7, one important challenge with using a TPLM with AL is the choice of both TPLM and its parameters. There are many models available to use, and currently none is specifically made for EM. It might be difficult to know what combinations work best on a specific dataset, and one would likely have to spend time to empirically test different combinations before one can decide. Choosing a wrong combination might lead to either an unstable and unusable model, or too long iteration time.

In addition, it can be difficult for the Oracle to understand how the TPLM "thinks". Using explainable EM could help the Oracle in his or her labeling process.

8.2 Is Active Learning a Viable Strategy for Entity Matching?

Both AL and EM are in practice interactive, iterative processes, and from this reason they fit well together. In AL, an Oracle interactively labels new examples which are used to iteratively train a machine learning model. In an EM situation this process is often naturally conducted.

As an example, if a company has data they want to do EM on (without AL), they could label a portion of the candidate set of matches and train a ML model. They would then apply the model on the unlabeled data, however the model is likely not good enough and would have some wrongly labeled examples. By manually finding some of the mislabeled examples they could then feed these new correctly labeled examples to the model, and train it again on the updated train set. With the new examples it is likely to perform better than before. This process could continue until the company is satisfied with the model's performance, or they are sure all matches have been found.

By using an AL strategy, the company would not need to manually find the examples the model has a hard time predicting, the model could find the examples for them. This could save valuable time when doing EM. There is an extra overhead when initially setting up an AL pipeline, however AL can be a highly modular and decoupled process by using a model agnostic query method (like uncertainty sampling). After a company has implemented the AL pipeline once, it can be reused across different ML models and datasets. As an example, using a model which is pre-trained using transfer learning requires no special implementation from an AL perspective.

For Cognite, we recommend looking further into using AL for EM, especially combined with TPLMs. From our results and experience doing AL and EM, we believe AL can (potentially with little effort) integrate into Cognite's existing EM solutions, and speed up the matching process significantly.

8.3 Future Work

In this section recommendations for future work are presented.

8.3.1 Comprehensive Benchmark

A comprehensive benchmark of TPLMs for AL on EM datasets, similar to what has been done in [10] with classical ML models, could standardize best practices for selection of TPLMs, hyperparameters, and query strategies. With AL-TPLMs this could especially

be useful when selecting a model, to be able to compare trade-offs between expected time use and prediction capability.

8.3.2 Practical Application

Similarly, an in depth review of how EM strategies work in practice could be useful for later applications of EM. It would be necessary to compare more classical models like Random Forest up against deep learning models like RoBERTa and DistilBERT, both with and without using AL. In this setting, we believe a standardized test framework for EM could be of use. Currently it is difficult to directly compare an application of non-AL EM and AL-ML. The test framework would need to incorporate total labeling time into its evaluation strategy.

8.3.3 Training Set Distribution and Query Strategies

A closer look at how the distribution of positive and negative examples in the initial training set affects the AL query strategy along with the final F1-score, could help the development of future query strategies and test methods. Our preliminary experiments suggest that this can heavily affect the performance of the same query strategy, and therefore needs to be considered both when conducting experiments, and when developing query strategies.

8.3.4 Optimized Query Strategies for Transformer Language Models

A recommendation is to explore further improvements to Partition-2, e.g. what the optimal size of the uncertain and high-confidence partitions is. In this thesis an equal amount of high-confidence and uncertain examples are chosen, but this could potentially be tweaked for a higher F1-score.

As presented in chapter 3, some work has been done to look at how more complex and potentially more optimized AL query strategies, originally made for traditional neural networks, work with TPLMs [27]. The development of novel query strategies which more closely use properties of TPLMs could potentially perform better than the model agnostic query strategies used in this thesis. As a reminder, the query strategies in the experiments presented in chapter 5 only use the model *predictions* to select the next examples to label. Jain et al. [4] suggest using the TPLM's encodings directly when selecting examples to label. We believe further work in this direction could lead to improvements for AL with TPLMs for EM.

8.3.5 Special Made Language Model

A special made TPLM for EM, such as what SciBERT is for scientific text, could potentially both increase F1-score and decrease training time. In addition a smaller version, such as DistilBERT, could be made for use in an AL situation.

8.3.6 Combine Active Learning with Other Strategies

Combining active learning with transfer learning and data augmentation, all of which are methods known to improve F1-score in a resource limited environment with low data availability [3], have to the best of our knowledge not yet been done before with TPLMs. Combining all above methods could lead to state-of-the-art results.

Bibliography

[1] Peter Christen. *Data Matching: Concepts and Techniques for Record Linkage, Entity Resolution, and Duplicate Detection*. Data-Centric Systems and Applications. Springer-Verlag, Berlin Heidelberg, 2012. ISBN 978-3-642-31163-5. doi: 10.1007/978-3-642-31164-2. URL https://www.springer.com/gp/book/9783642311635.

[2] Ursin Brunner and Kurt Stockinger. Entity Matching with Transformer Architectures - A Step Forward in Data Integration, 2020. URL https://openproceedings.org/2020/conf/edbt/paper_205.pdf. Version Number: 1 type: dataset.

[3] Yuliang Li, Jinfeng Li, Yoshihiko Suhara, AnHai Doan, and Wang-Chiew Tan. Deep Entity Matching with Pre-Trained Language Models. *Proceedings of the VLDB Endowment*, 14(1):50–60, September 2020. ISSN 2150-8097. doi: 10.14778/3421424.3421431. URL http://arxiv.org/abs/2004.00584. arXiv: 2004.00584.

[4] Arjit Jain, Sunita Sarawagi, and Prithviraj Sen. Deep Indexed Active Learning for Matching Heterogeneous Entity Representations. *arXiv:2104.03986 [cs, stat]*, April 2021. URL http://arxiv.org/abs/2104.03986. arXiv: 2104.03986.

[5] Nils Barlaug and Jon Atle Gulla. Neural Networks for Entity Matching. *arXiv:2010.11075 [cs]*, October 2020. URL http://arxiv.org/abs/2010.11075. arXiv: 2010.11075.

[6] Halbert L. Dunn. Record Linkage. *American Journal of Public Health and the Nations Health*, 36(12):1412–1416, December 1946. ISSN 0002-9572. URL https://www.ncbi.nlm.nih.gov/pmc/articles/PMC1624512/.

[7] Howard B. Newcombe and James M. Kennedy. Record linkage: making maximum use of the discriminating power of identifying information. *Communications of the ACM*, 5(11):563–566, November 1962. ISSN 0001-0782. doi: 10.1145/368996.369026. URL https://doi.org/10.1145/368996.369026.

[8] Ivan P. Fellegi and Alan B. Sunter. A Theory for Record Linkage. *Journal of the American Statistical Association*, 64(328):1183–1210, 1969. ISSN 0162-1459. doi: 10.2307/2286061. URL https://www.jstor.org/stable/2286061. Publisher: [American Statistical Association, Taylor & Francis, Ltd.].

[9] US Census Bureau. An Application of the Fellegi-Sunter Model of Record Linkage to the... URL https://www.census.gov/library/working-papers/1991/adrm/rr91-09.html. Section: Government.

[10] Venkata Vamsikrishna Meduri, Lucian Popa, Prithviraj Sen, and Mohamed Sarwat. A Comprehensive Benchmark Framework for Active Learning Methods in Entity Matching. In *Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data*, SIGMOD '20, pages 1133–1147, Portland, OR, USA, June 2020. Association for Computing Machinery. ISBN 978-1-4503-6735-6. doi: 10.1145/3318464.3380597. URL https://doi.org/10.1145/3318464.3380597.

[11] Barzan Mozafari, Purna Sarkar, Michael Franklin, Michael Jordan, and Samuel Madden. Scaling up crowd-sourcing to very large datasets. *Proceedings of the VLDB Endowment*, 8:125–136, October 2014. doi: 10.14778/2735471.2735474.

[12] V. Christophides, V. Efthymiou, Themis Palpanas, G. Papadakis, and K. Stefanidis. End-to-End Entity Resolution for Big Data: A Survey. *ArXiv*, 2019.

[13] Wei Zhang, Hao Wei, Bunyamin Sisman, Xin Luna Dong, Christos Faloutsos, and Davd Page. AutoBlock: A Hands-off Blocking Framework for Entity Matching. In *Proceedings of the 13th International Conference on Web Search and Data Mining*, WSDM '20, pages 744–752, New York, NY, USA, January 2020. Association for Computing Machinery. ISBN 978-1-4503-6822-3. doi: 10.1145/3336191.3371813. URL https://doi.org/10.1145/3336191.3371813.

[14] Muhammad Ebraheem, Saravanan Thirumuruganathan, Shafiq Joty, Mourad Ouzzani, and Nan Tang. DeepER – Deep Entity Resolution. *arXiv:1710.00597 [cs]*, November 2019. doi: 10.14778/3236187.3236198. URL http://arxiv.org/abs/1710.00597. arXiv: 1710.00597.

[15] George Papadakis, Dimitrios Skoutas, Emmanouil Thanos, and Themis Palpanas. A Survey of Blocking and Filtering Techniques for Entity Resolution. *arXiv:1905.06167 [cs]*, August 2020. URL http://arxiv.org/abs/1905.06167. arXiv: 1905.06167.

[16] AnHai Doan, Alon Halevy, and Zachary G. Ives. *Principles of data integration*. Morgan Kaufmann, Waltham, MA, 2012. ISBN 978-0-12-416044-6.

[17] Pradap Konda, Jeff Naughton, Shishir Prasad, Ganesh Krishnan, Rohit Deep, Vijay Raghavendra, Sanjib Das, Paul Suganthan G. C., AnHai Doan, Adel Ardalan,

Jeffrey R. Ballard, Han Li, Fatemah Panahi, and Haojun Zhang. Magellan: toward building entity matching management systems. *Proceedings of the VLDB Endowment*, 9(12):1197–1208, August 2016. ISSN 21508097. doi: 10.14778/2994509.2994535. URL http://dl.acm.org/citation.cfm?doid=2994509.2994535.

[18] Banda Ramadan and Peter Christen. Unsupervised Blocking Key Selection for Real-Time Entity Resolution. In Tru Cao, Ee-Peng Lim, Zhi-Hua Zhou, Tu-Bao Ho, David Cheung, and Hiroshi Motoda, editors, *Advances in Knowledge Discovery and Data Mining*, Lecture Notes in Computer Science, pages 574–585, Cham, 2015. Springer International Publishing. ISBN 978-3-319-18032-8. doi: 10.1007/978-3-319-18032-8_45.

[19] Burr Settles. Active Learning Literature Survey. Technical Report, University of Wisconsin-Madison Department of Computer Sciences, 2009. URL https://minds.wisconsin.edu/handle/1793/60660. Accepted: 2012-03-15T17:23:56Z.

[20] Arvind Arasu, Michaela Götz, and Raghav Kaushik. On active learning of record matching packages. In *Proceedings of the 2010 international conference on Management of data - SIGMOD '10*, page 783, Indianapolis, Indiana, USA, 2010. ACM Press. ISBN 978-1-4503-0032-2. doi: 10.1145/1807167.1807252. URL http://portal.acm.org/citation.cfm?doid=1807167.1807252.

[21] Jungo Kasai, Kun Qian, Sairam Gurajada, Yunyao Li, and Lucian Popa. Low-resource Deep Entity Resolution with Transfer and Active Learning. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 5851–5861, Florence, Italy, 2019. Association for Computational Linguistics. doi: 10.18653/v1/P19-1586. URL https://www.aclweb.org/anthology/P19-1586.

[22] Pengzhen Ren, Yun Xiao, Xiaojun Chang, Po-Yao Huang, Zhihui Li, Xiaojiang Chen, and Xin Wang. A Survey of Deep Active Learning. *arXiv:2009.00236 [cs, stat]*, August 2020. URL http://arxiv.org/abs/2009.00236. arXiv: 2009.00236.

[23] Burr Settles. Active Learning. *Synthesis Lectures on Artificial Intelligence and Machine Learning*, 6(1):1–114, June 2012. ISSN 1939-4608. doi: 10.2200/S00429ED1V01Y201207AIM018. URL https://www.morganclaypool.com/doi/abs/10.2200/S00429ED1V01Y201207AIM018. Publisher: Morgan & Claypool Publishers.

[24] H. S. Seung, M. Opper, and H. Sompolinsky. Query by committee. In *Annual Workshop on Computational Learning Theory: Proceedings of the fifth annual workshop on Computational learning theory; 27-29 July 1992*, pages 287–294. Association for Computing Machinery, Inc , One Astor Plaza, 1515 Broadway, New York, NY, 10036-5701, USA, [mailto:SIGS@acm.org], [URL:http://www.acm.org/], 1992. ISBN 978-0-89791-497-0. doi: http://dx.doi.org/10.1145/130385.130417. URL https://search.proquest.com/docview/31492112?pq-origsite=summon. Num Pages: 8.

[25] Sunita Sarawagi and Anuradha Bhamidipaty. Interactive deduplication using active learning. In *Proceedings of the eighth ACM SIGKDD international conference on Knowledge discovery and data mining*, KDD '02, pages 269–278, New York, NY, USA, July 2002. Association for Computing Machinery. ISBN 978-1-58113-567-1. doi: 10.1145/775047.775087. URL https://doi.org/10.1145/775047.775087.

[26] Punit Kumar and Atul Gupta. Active Learning Query Strategies for Classification, Regression, and Clustering: A Survey. *Journal of Computer Science and Technology*, 35(4):913–945, July 2020. ISSN 1860-4749. doi: 10.1007/s11390-020-9487-4. URL https://doi.org/10.1007/s11390-020-9487-4.

[27] Liat Ein-Dor, Alon Halfon, Ariel Gera, Eyal Shnarch, Lena Dankin, Leshem Choshen, Marina Danilevsky, Ranit Aharonov, Yoav Katz, and Noam Slonim. Active Learning for BERT: An Empirical Study. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 7949–7962, Online, November 2020. Association for Computational Linguistics. doi: 10.18653/v1/2020.emnlp-main.638. URL https://www.aclweb.org/anthology/2020.emnlp-main.638.

[28] Sidharth Mudgal, Han Li, Theodoros Rekatsinas, AnHai Doan, Youngchoon Park, Ganesh Krishnan, Rohit Deep, Esteban Arcaute, and Vijay Raghavendra. Deep Learning for Entity Matching: A Design Space Exploration. In *Proceedings of the 2018 International Conference on Management of Data - SIGMOD '18*, pages 19–34, Houston, TX, USA, 2018. ACM Press. ISBN 978-1-4503-4703-7. doi: 10.1145/3183713.3196926. URL http://dl.acm.org/citation.cfm?doid=3183713.3196926.

[29] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural Machine Translation by Jointly Learning to Align and Translate. *arXiv:1409.0473 [cs, stat]*, May 2016. URL http://arxiv.org/abs/1409.0473. arXiv: 1409.0473.

[30] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention Is All You Need. *arXiv:1706.03762 [cs]*, December 2017. URL http://arxiv.org/abs/1706.03762. arXiv: 1706.03762.

[31] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. *arXiv:1810.04805 [cs]*, May 2019. URL http://arxiv.org/abs/1810.04805. arXiv: 1810.04805.

[32] Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. RoBERTa: A Robustly Optimized BERT Pretraining Approach. *arXiv:1907.11692 [cs]*, July 2019. URL http://arxiv.org/abs/1907.11692. arXiv: 1907.11692.

[33] Victor Sanh, Lysandre Debut, Julien Chaumond, and Thomas Wolf. DistilBERT, a distilled version of BERT: smaller, faster, cheaper and lighter. *arXiv:1910.01108 [cs]*, February 2020. URL `http://arxiv.org/abs/1910.01108`. arXiv: 1910.01108.

[34] Ralph Peeters, C. Bizer, and Goran Glavas. Intermediate Training of BERT for Product Matching. In *DI2KG@VLDB*, 2020.

[35] Zhengjie Miao, Yuliang Li, Xiaolan Wang, and Wang-Chiew Tan. Snippext: Semi-supervised Opinion Mining with Augmented Data. *Proceedings of The Web Conference 2020*, pages 617–628, April 2020. doi: 10.1145/3366423.3380144. URL `http://arxiv.org/abs/2002.03049`. arXiv: 2002.03049.

[36] K. Wang, D. Zhang, Y. Li, R. Zhang, and L. Lin. Cost-Effective Active Learning for Deep Image Classification. *IEEE Transactions on Circuits and Systems for Video Technology*, 27(12):2591–2600, December 2017. ISSN 1558-2205. doi: 10.1109/TCSVT.2016.2589879. Conference Name: IEEE Transactions on Circuits and Systems for Video Technology.

[37] Kun Qian, Lucian Popa, and Prithviraj Sen. Active Learning for Large-Scale Entity Resolution. In *Proceedings of the 2017 ACM on Conference on Information and Knowledge Management*, pages 1379–1388, Singapore Singapore, November 2017. ACM. ISBN 978-1-4503-4918-5. doi: 10.1145/3132847.3132949. URL `https://dl.acm.org/doi/10.1145/3132847.3132949`.

[38] Ozan Sener and Silvio Savarese. Active Learning for Convolutional Neural Networks: A Core-Set Approach. *arXiv:1708.00489 [cs, stat]*, June 2018. URL `http://arxiv.org/abs/1708.00489`. arXiv: 1708.00489.

[39] Iz Beltagy, Kyle Lo, and Arman Cohan. SciBERT: A Pretrained Language Model for Scientific Text. *arXiv:1903.10676 [cs]*, September 2019. URL `http://arxiv.org/abs/1903.10676`. arXiv: 1903.10676.

[40] Jinhyuk Lee, Wonjin Yoon, Sungdong Kim, Donghyeon Kim, Sunkyu Kim, Chan Ho So, and Jaewoo Kang. BioBERT: a pre-trained biomedical language representation model for biomedical text mining. *Bioinformatics*, 36(4):1234–1240, February 2020. ISSN 1367-4803. doi: 10.1093/bioinformatics/btz682. URL `https://doi.org/10.1093/bioinformatics/btz682`.

[41] Vincenzo Di Cicco, Donatella Firmani, Nick Koudas, Paolo Merialdo, and Divesh Srivastava. Interpreting deep learning models for entity resolution: an experience report using LIME. In *Proceedings of the Second International Workshop on Exploiting Artificial Intelligence Techniques for Data Management*, aiDM '19, pages 1–4, New York, NY, USA, July 2019. Association for Computing Machinery. ISBN 978-1-4503-6802-5. doi: 10.1145/3329859.3329878. URL `https://doi.org/10.1145/3329859.3329878`.

Appendices

Additional Results

The appendix include raw numbers from the graphs presented in chapter 6 and chapter 7, including a selection of additional results referred to in the thesis but not included in the main text.

F1-score

Following are the raw F1-scores from the graphs in the results section. Note that they are presented with respect to the *active learning iteration*, and not by the number of labeled examples. This comes from the fact that at each iteration different query strategies could potentially have different number of labeled training examples. In iteration 0, all have 200 examples in their training set. In iteration 20, the maximum examples possible are 1000. For more details on this, see chapter 5.

Table 1: Raw numbers for F1-score on Amazon-Google.

Iteration	Baseline	Partition-4	Hybrid	Hybrid-Partition-2	Uncertainty	Partition-2
0	0.249	0.249	0.528	0.528	0.249	0.249
1	0.406	0.438	0.560	0.544	0.482	0.443
2	0.476	0.384	0.579	0.588	0.538	0.528
3	0.393	0.443	0.577	0.596	0.554	0.564
4	0.487	0.466	0.588	0.602	0.557	0.563
5	0.425	0.489	0.578	0.611	0.574	0.571
6	0.503	0.525	0.583	0.633	0.595	0.600
7	0.518	0.529	0.567	0.638	0.624	0.630
8	0.327	0.540	0.599	0.648	0.647	0.616
9	0.446	0.586	0.603	0.670	0.634	0.646
10	0.498	0.570	0.604	0.660	0.620	0.660
11	0.424	0.599	0.645	0.681	0.636	0.678
12	0.544	0.604	0.639	0.686	0.643	0.675
13	0.535	0.591	0.662	0.685	0.640	0.686
14	0.501	0.620	0.448	0.670	0.651	0.712
15	0.556	0.621	0.648	0.694	0.683	0.701
16	0.530	0.642	0.648	0.688	0.675	0.689
17	0.535	0.643	0.633	0.684	0.684	0.700
18	0.554	0.648	0.641	0.691	0.569	0.698
19	0.386	0.628	0.639	0.681	0.663	0.699
20	0.591	0.645	0.645	0.696	0.679	0.706

Table 2: Raw numbers for F1-score on Abt-Buy.

Iteration	Baseline	Partition-4	Hybrid	Hybrid-Partition-2	Uncertainty	Partition-2
0	0.497	0.497	0.631	0.631	0.497	0.497
1	0.354	0.443	0.780	0.788	0.603	0.654
2	0.444	0.640	0.808	0.805	0.640	0.747
3	0.635	0.749	0.815	0.823	0.775	0.810
4	0.737	0.767	0.813	0.843	0.791	0.843
5	0.625	0.772	0.818	0.852	0.800	0.848
6	0.703	0.818	0.831	0.876	0.798	0.866
7	0.669	0.825	0.831	0.870	0.826	0.870
8	0.765	0.832	0.847	0.889	0.877	0.881
9	0.746	0.829	0.815	0.892	0.865	0.884
10	0.763	0.840	0.851	0.898	0.575	0.901
11	0.666	0.853	0.831	0.884	0.885	0.895
12	0.759	0.856	0.843	0.906	0.600	0.894
13	0.748	0.854	0.842	0.910	0.582	0.901
14	0.773	0.870	0.839	0.907	0.898	0.898
15	0.801	0.869	0.839	0.905	0.587	0.908
16	0.532	0.868	0.846	0.909	0.896	0.901
17	0.806	0.873	0.854	0.901	0.874	0.899
18	0.813	0.869	0.856	0.888	0.592	0.904
19	0.823	0.886	0.850	0.899	0.296	0.906
20	0.833	0.888	0.864	0.897	0.893	0.898

Table 3: Raw numbers for F1-score on Walmart-Amazon.

Iteration	Baseline	Partition-4	Hybrid	Hybrid-Partition-2	Uncertainty	Partition-2
0	0.258	0.258	0.603	0.603	0.258	0.258
1	0.309	0.416	0.618	0.619	0.404	0.234
2	0.206	0.463	0.611	0.652	0.629	0.660
3	0.151	0.577	0.637	0.730	0.626	0.699
4	0.450	0.659	0.634	0.794	0.411	0.786
5	0.603	0.646	0.680	0.759	0.716	0.823
6	0.216	0.763	0.753	0.802	0.762	0.822
7	0.512	0.783	0.733	0.833	0.488	0.850
8	0.552	0.779	0.748	0.838	0.804	0.817
9	0.433	0.811	0.685	0.842	0.785	0.842
10	0.682	0.836	0.710	0.834	0.826	0.843
11	0.648	0.814	0.740	0.841	0.544	0.819
12	0.466	0.803	0.755	0.847	0.809	0.864
13	0.660	0.838	0.720	0.848	0.847	0.864
14	0.606	0.830	0.703	0.867	0.542	0.856
15	0.547	0.841	0.761	0.879	0.809	0.861
16	0.686	0.817	0.780	0.870	0.819	0.872
17	0.767	0.825	0.791	0.878	0.545	0.869
18	0.628	0.834	0.774	0.880	0.549	0.862
19	0.722	0.844	0.771	0.870	0.695	0.868
20	0.793	0.844	0.768	0.873	0.572	0.866

Table 4: Raw numbers for F1-score on Walmart-Amazon.

Iteration	Baseline	Partition-4	Hybrid	Hybrid-Partition-2	Uncertainty	Partition-2
0	0.954	0.954	0.970	0.970	0.954	0.954
1	0.936	0.968	0.966	0.984	0.975	0.974
2	0.950	0.974	0.943	0.989	0.977	0.987
3	0.958	0.987	0.970	0.989	0.981	0.988
4	0.965	0.983	0.962	0.990	0.986	0.986
5	0.967	0.986	0.977	0.990	0.987	0.989
6	0.963	0.988	0.984	0.989	0.988	0.989
7	0.968	0.989	0.986	0.991	0.989	0.988
8	0.963	0.988	0.987	0.989	0.989	0.988
9	0.980	0.988	0.986	0.989	0.989	0.987
10	0.972	0.987	0.982	0.989	0.989	0.989
11	0.964	0.989	0.981	0.990	0.988	0.991
12	0.975	0.988	0.986	0.990	0.989	0.990
13	0.980	0.989	0.980	0.989	0.988	0.988
14	0.977	0.988	0.982	0.989	0.988	0.990
15	0.977	0.987	0.984	0.990	0.988	0.988
16	0.974	0.988	0.985	0.989	0.990	0.989
17	0.971	0.988	0.987	0.988	0.989	0.991
18	0.976	0.988	0.982	0.990	0.989	0.990
19	0.977	0.990	0.981	0.988	0.987	0.989
20	0.977	0.989	0.981	0.990	0.988	0.990

Table 5: Raw numbers for F1-score on DBLP-GoogleScholar.

Iteration	Baseline	Partition-4	Hybrid	Hybrid-Partition-2	Uncertainty	Partition-2
0	0.903	0.903	0.914	0.914	0.903	0.903
1	0.915	0.927	0.910	0.930	0.930	0.920
2	0.913	0.931	0.912	0.937	0.922	0.932
3	0.914	0.938	0.920	0.934	0.937	0.936
4	0.919	0.932	0.920	0.942	0.935	0.929
5	0.921	0.939	0.922	0.942	0.938	0.945
6	0.921	0.939	0.926	0.944	0.937	0.944
7	0.930	0.943	0.928	0.941	0.945	0.944
8	0.924	0.943	0.923	0.945	0.941	0.945
9	0.928	0.944	0.925	0.944	0.945	0.941
10	0.929	0.942	0.927	0.946	0.946	0.946
11	0.934	0.940	0.932	0.947	0.934	0.946
12	0.934	0.943	0.929	0.945	0.949	0.946
13	0.932	0.943	0.934	0.949	0.947	0.950
14	0.930	0.941	0.928	0.946	0.944	0.949
15	0.928	0.945	0.928	0.950	0.944	0.952
16	0.926	0.944	0.933	0.949	0.945	0.949
17	0.934	0.943	0.928	0.949	0.949	0.951
18	0.931	0.943	0.931	0.948	0.947	0.951
19	0.930	0.948	0.927	0.951	0.949	0.953
20	0.938	0.942	0.930	0.951	0.950	0.953

Standard Deviation

These are the raw numbers for the standard deviation graph in the results chapter.

Table 6: Raw numbers for standard deviation on Walmart-Amazon.

Iteration	Baseline	Partition-4	Hybrid	Hybrid-Partition-2	Uncertainty	Partition-2
0	0.231	0.231	0.069	0.069	0.231	0.231
1	0.235	0.154	0.010	0.069	0.248	0.262
2	0.192	0.296	0.005	0.024	0.084	0.025
3	0.213	0.111	0.032	0.060	0.019	0.085
4	0.180	0.029	0.038	0.013	0.187	0.029
5	0.058	0.054	0.041	0.098	0.090	0.026
6	0.365	0.105	0.048	0.040	0.031	0.009
7	0.145	0.050	0.041	0.016	0.403	0.016
8	0.256	0.030	0.019	0.021	0.021	0.028
9	0.378	0.016	0.018	0.029	0.017	0.018
10	0.052	0.015	0.048	0.015	0.025	0.015
11	0.114	0.015	0.034	0.022	0.471	0.014
12	0.335	0.031	0.021	0.027	0.032	0.016
13	0.048	0.031	0.041	0.020	0.025	0.009
14	0.172	0.006	0.050	0.010	0.470	0.022
15	0.303	0.018	0.009	0.013	0.022	0.015
16	0.048	0.015	0.067	0.002	0.028	0.003
17	0.064	0.032	0.027	0.003	0.473	0.006
18	0.201	0.017	0.038	0.020	0.476	0.012
19	0.032	0.012	0.031	0.023	0.223	0.019
20	0.044	0.013	0.030	0.010	0.496	0.020

Time Use

These are the raw numbers for the iteration time graph in the results chapter.

Table 7: Raw numbers for time use on Amazon-Google.

Iteration	Baseline	Partition-4	Hybrid	Hybrid-Partition-2	Uncertainty	Partition-2
0	156.5	151.6	193.4	140.2	128.7	164.3
1	173.2	160.1	194.1	155.6	137.4	172.1
2	167.2	163.7	196.5	163.2	146.4	178.3
3	171.0	169.7	198.1	168.7	147.7	182.3
4	167.5	168.8	200.1	167.8	147.9	186.0
5	174.8	173.4	201.9	170.1	155.6	191.8
6	176.2	168.6	201.8	180.4	154.4	197.9
7	187.1	177.7	200.4	181.6	152.6	200.9
8	176.3	177.4	205.7	183.9	154.6	203.6
9	183.7	177.1	206.3	185.1	161.2	206.9
10	191.2	176.8	209.5	193.4	160.8	210.3
11	192.1	186.9	211.8	196.9	160.8	215.6
12	185.5	185.8	216.1	199.1	165.3	220.1
13	194.0	186.9	214.3	200.0	172.2	225.8
14	188.7	190.8	216.5	210.8	168.1	224.2
15	187.7	192.9	217.3	214.0	171.9	229.6
16	208.7	197.5	220.2	218.4	178.6	242.4
17	204.8	192.9	225.6	219.4	177.5	244.7
18	207.7	197.1	224.8	225.1	176.8	247.8
19	209.0	200.4	229.4	225.2	178.3	248.3
20	207.5	198.7	232.9	231.0	184.7	257.7

Pool Positive Rate

These are the raw numbers for the rate of true positive examples in the pool, from the graph in the discussion chapter.

Table 8: Raw numbers for time use on Amazon-Google.

Iteration	Baseline	Partition-4	Hybrid	Hybrid-Partition-2	Uncertainty	Partition-2
0	156.5	151.6	193.4	140.2	128.7	164.3
1	173.2	160.1	194.1	155.6	137.4	172.1
2	167.2	163.7	196.5	163.2	146.4	178.3
3	171.0	169.7	198.1	168.7	147.7	182.3
4	167.5	168.8	200.1	167.8	147.9	186.0
5	174.8	173.4	201.9	170.1	155.6	191.8
6	176.2	168.6	201.8	180.4	154.4	197.9
7	187.1	177.7	200.4	181.6	152.6	200.9
8	176.3	177.4	205.7	183.9	154.6	203.6
9	183.7	177.1	206.3	185.1	161.2	206.9
10	191.2	176.8	209.5	193.4	160.8	210.3
11	192.1	186.9	211.8	196.9	160.8	215.6
12	185.5	185.8	216.1	199.1	165.3	220.1
13	194.0	186.9	214.3	200.0	172.2	225.8
14	188.7	190.8	216.5	210.8	168.1	224.2
15	187.7	192.9	217.3	214.0	171.9	229.6
16	208.7	197.5	220.2	218.4	178.6	242.4
17	204.8	192.9	225.6	219.4	177.5	244.7
18	207.7	197.1	224.8	225.1	176.8	247.8
19	209.0	200.4	229.4	225.2	178.3	248.3
20	207.5	198.7	232.9	231.0	184.7	257.7

Recall

Presented below are the recall graphs from the experiments.

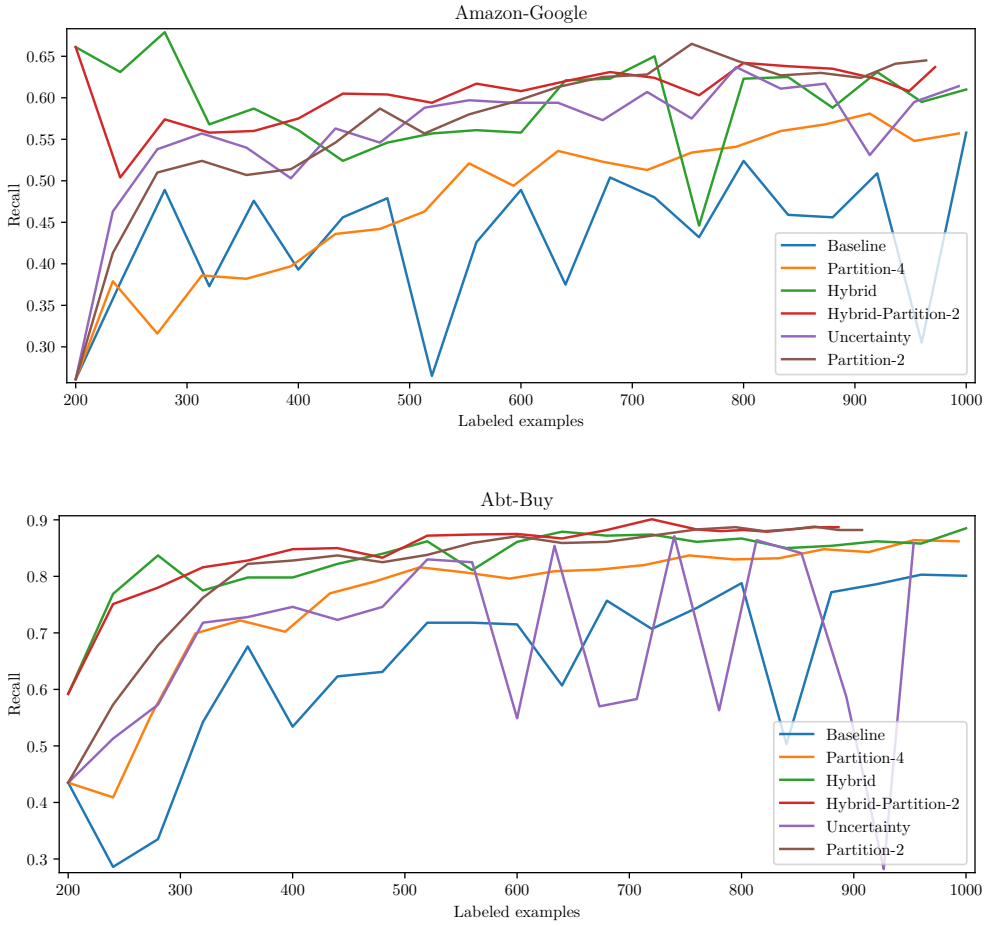


Figure 1: (Figure continues on next page).

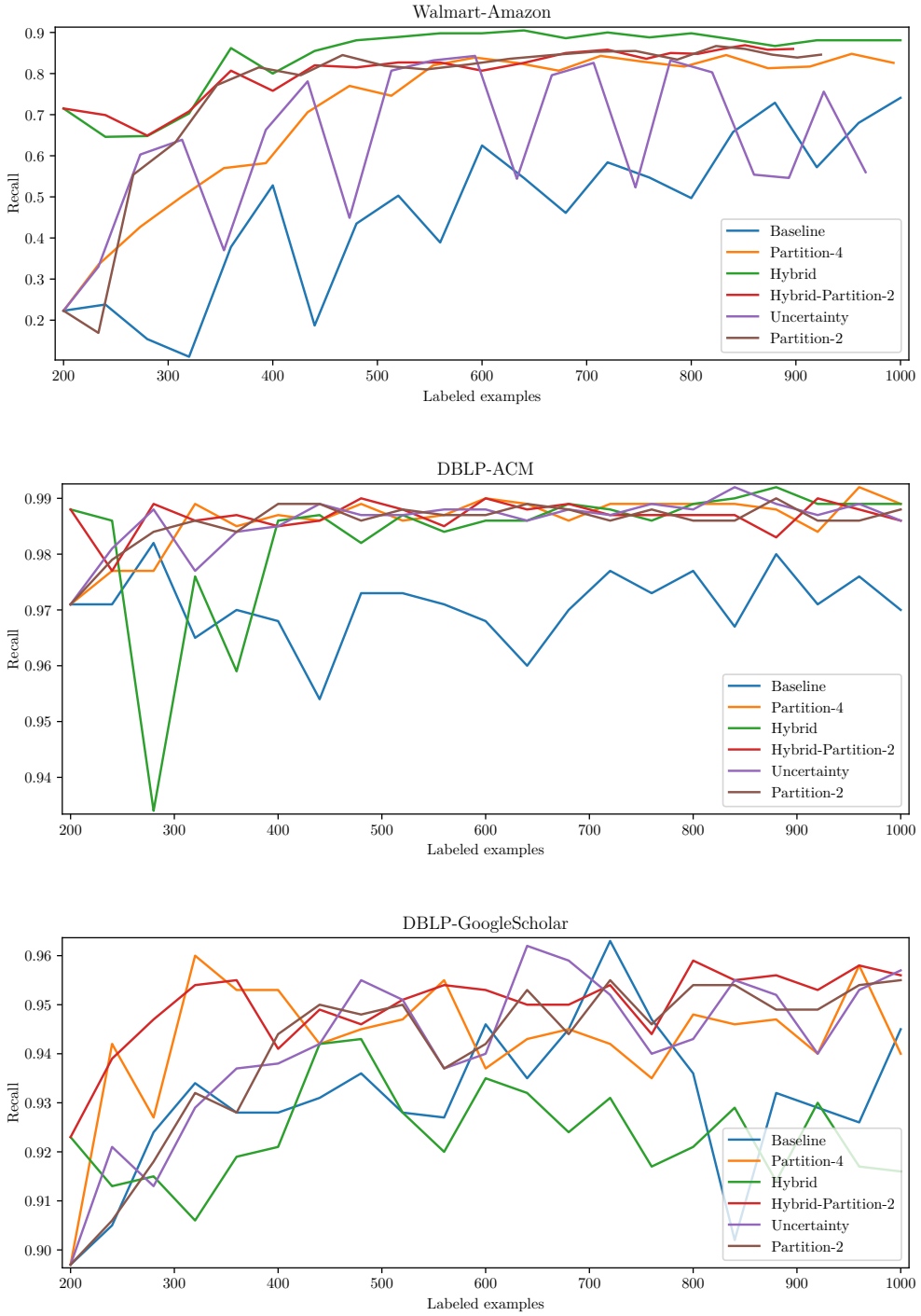


Figure 1: Iterative recall for a selection of different active learning query strategies with RoBERTa as the TPLM, on all datasets.

Precision

Presented below are the precision graphs from the experiments.

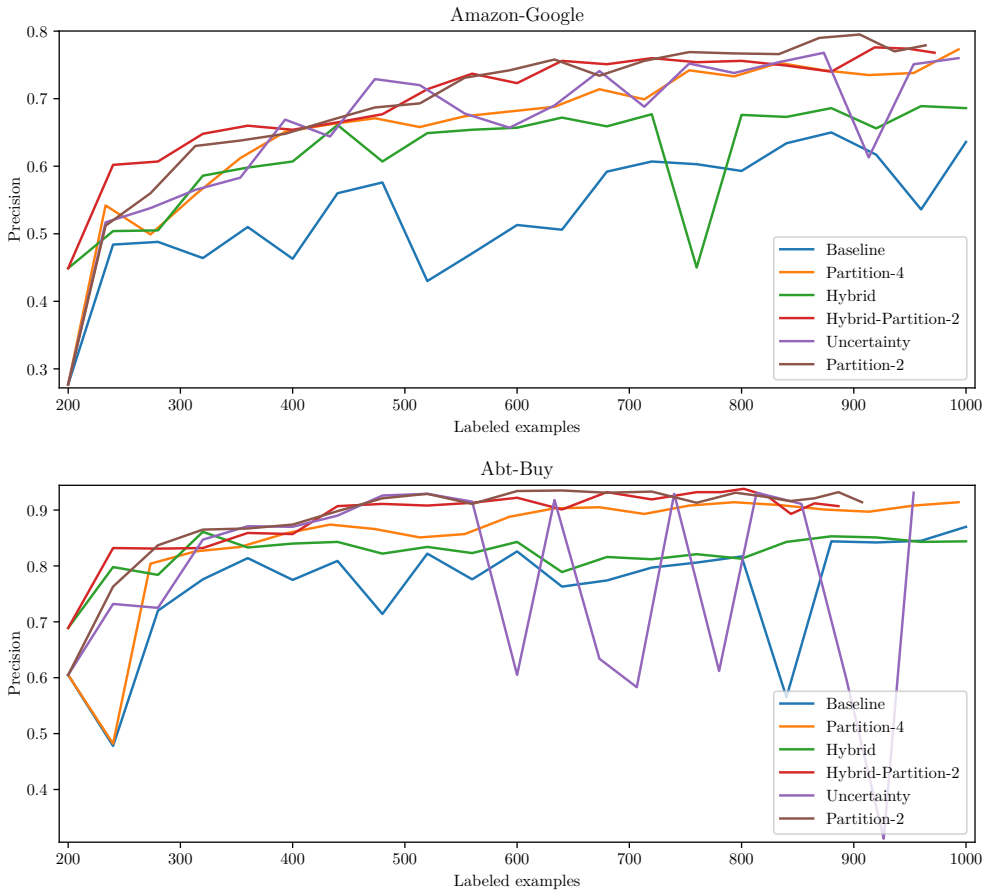


Figure 2: (Figure continues on next page).

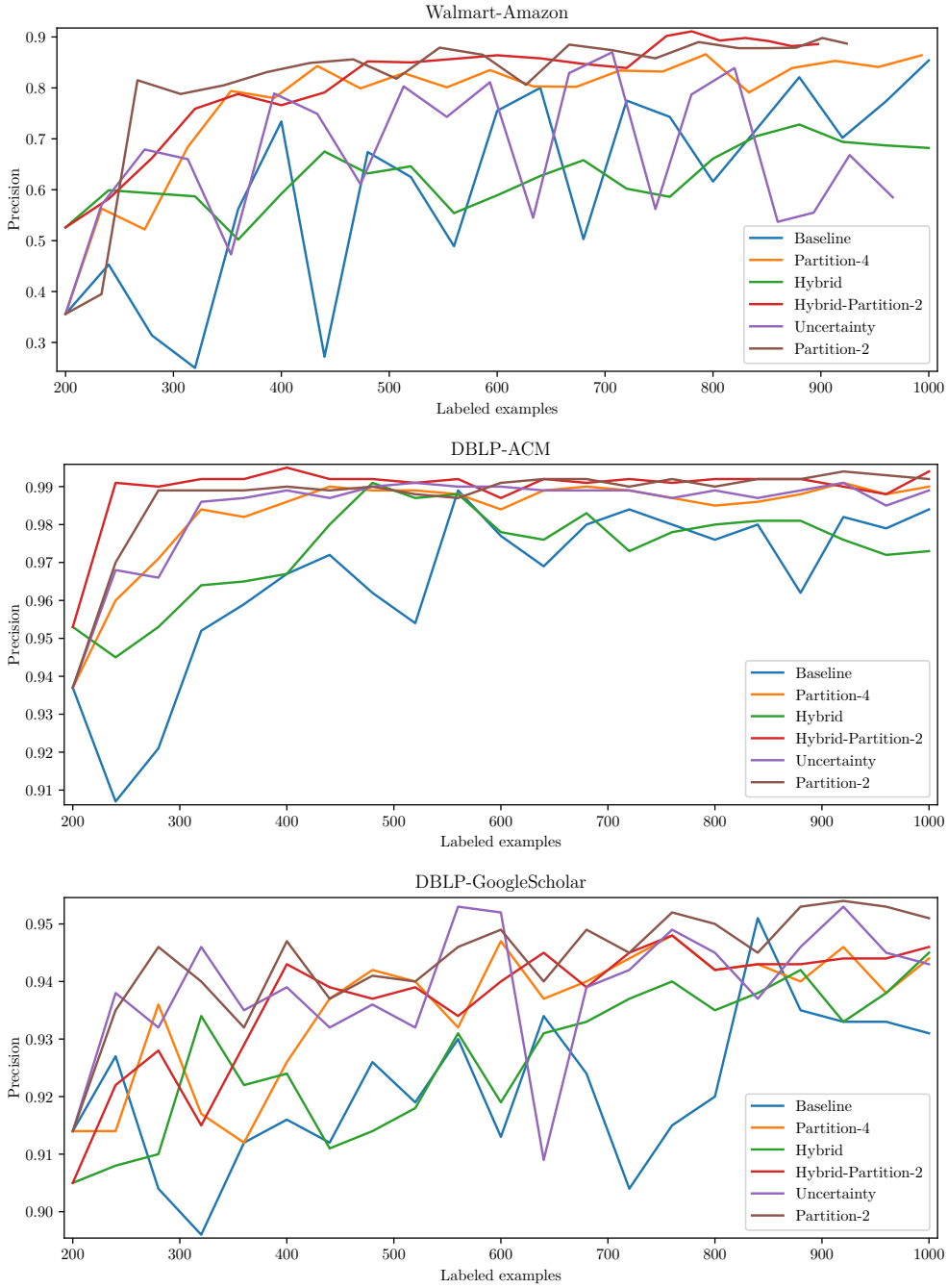


Figure 2: Iterative precision for a selection of different active learning query strategies with RoBERTa as the TPLM, on all datasets.

Comparison to Classic Machine Learning

Presented below are the F1-score graphs from the experiments comparing classic machine learning and TPLMs when performing active learning.

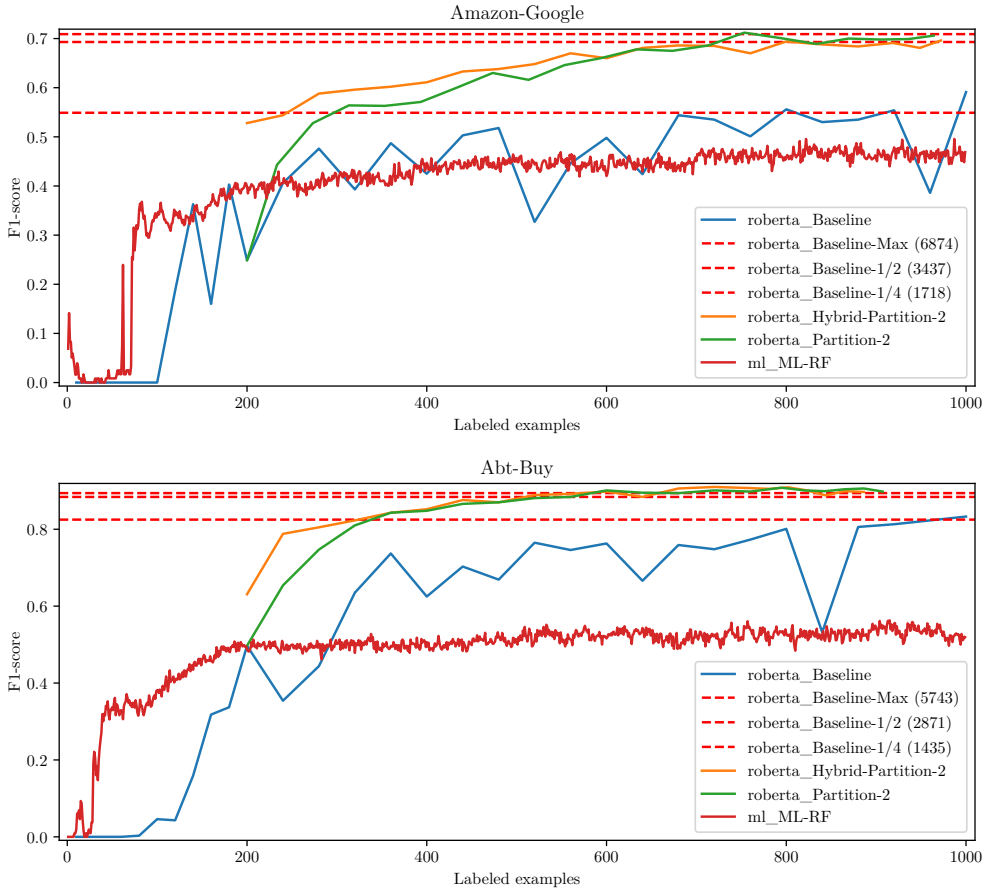


Figure 3: (Figure continues on next page).

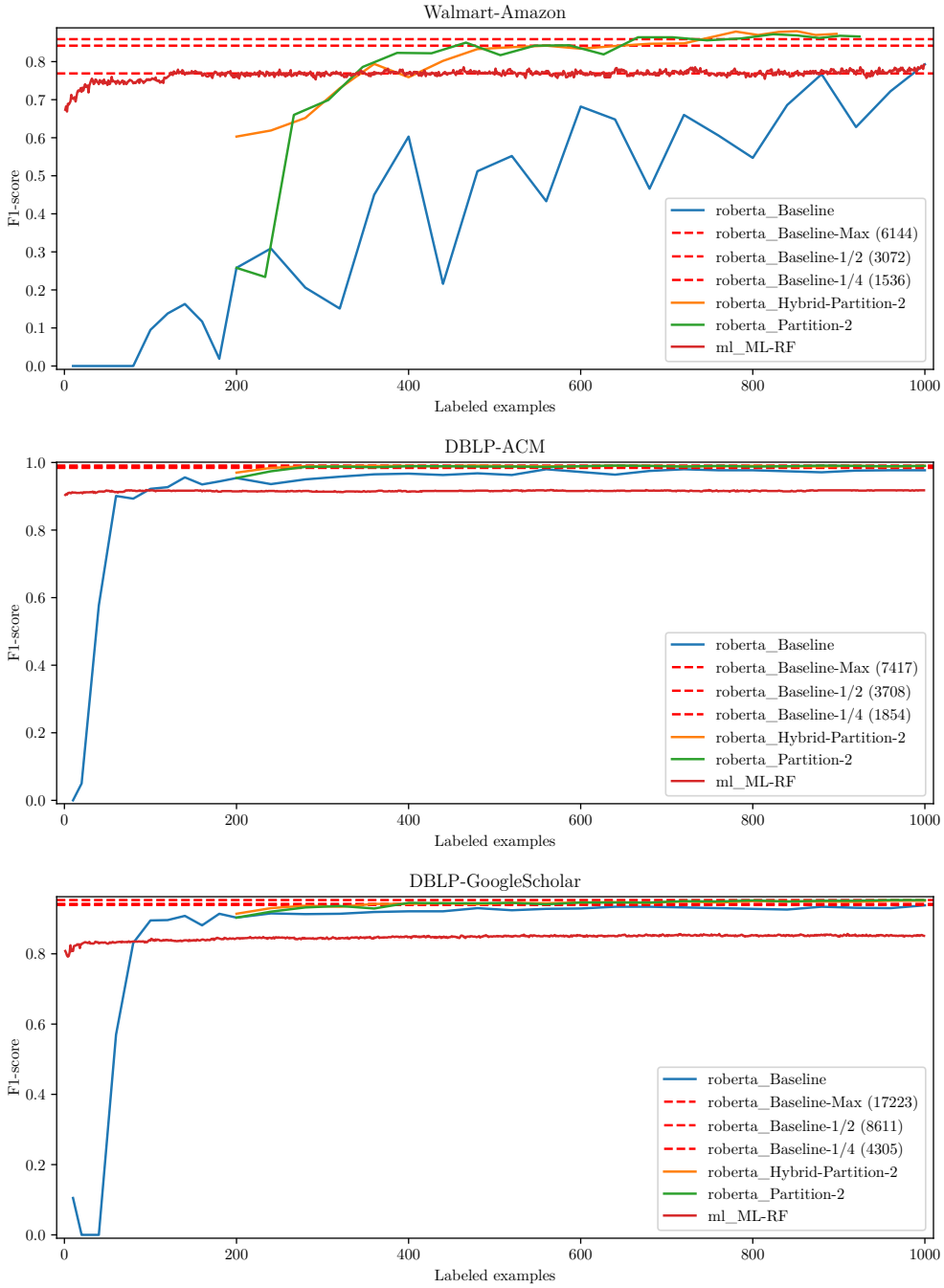


Figure 3: Iterative F-score for a selection of different active learning query strategies with RoBERTa as the TPLM compared to Random Forest, on all datasets.

DistilBERT vs RoBERTa

Presented below are the F1-score and iteration time of the preliminary experiment of testing DistilBERT against RoBERTa.

F1-score

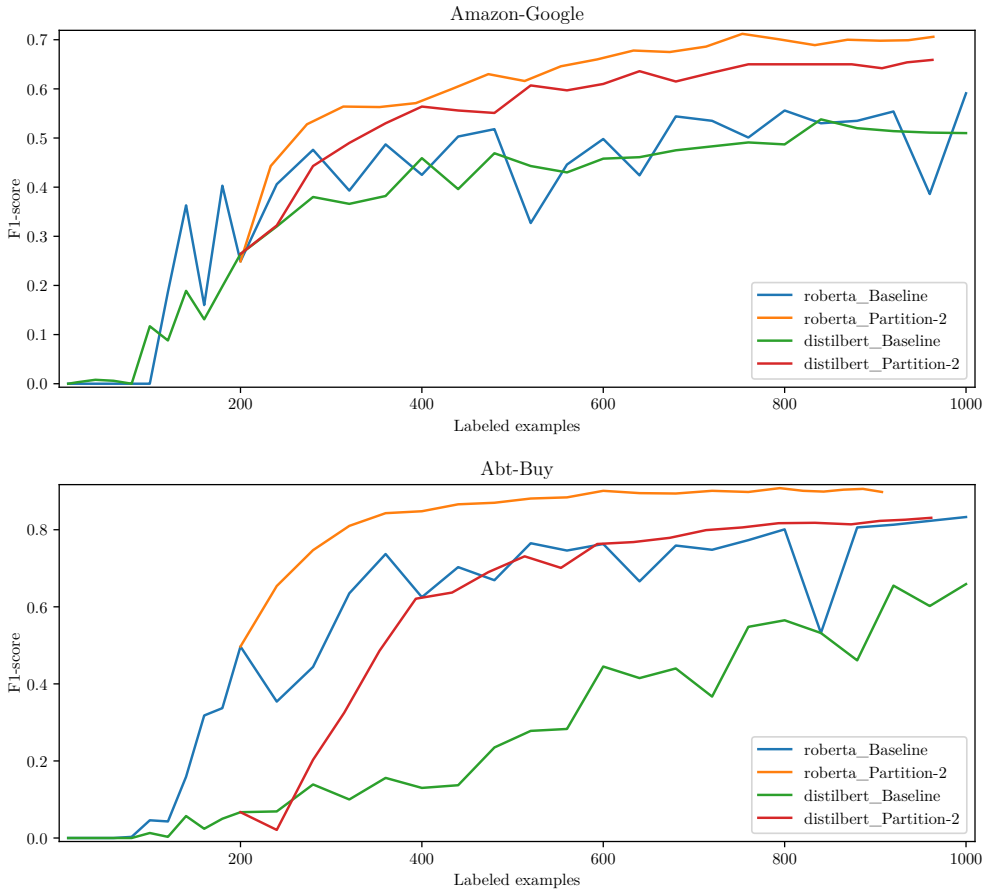


Figure 4: (Figure continues on next page).

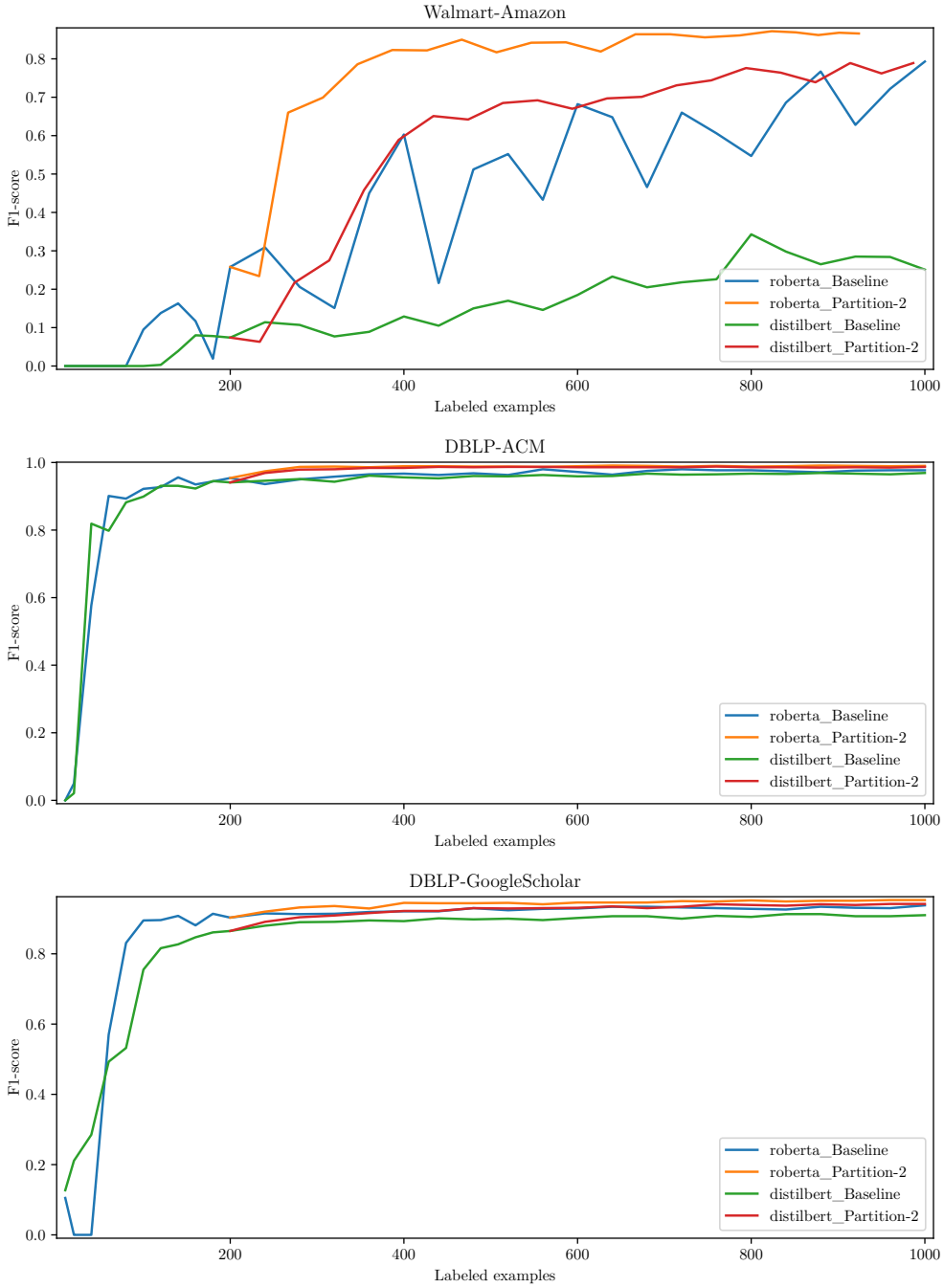


Figure 4: Iterative F1-score for a selection of active learning query strategies with DistilBERT as the TPLM.

Iteration Time

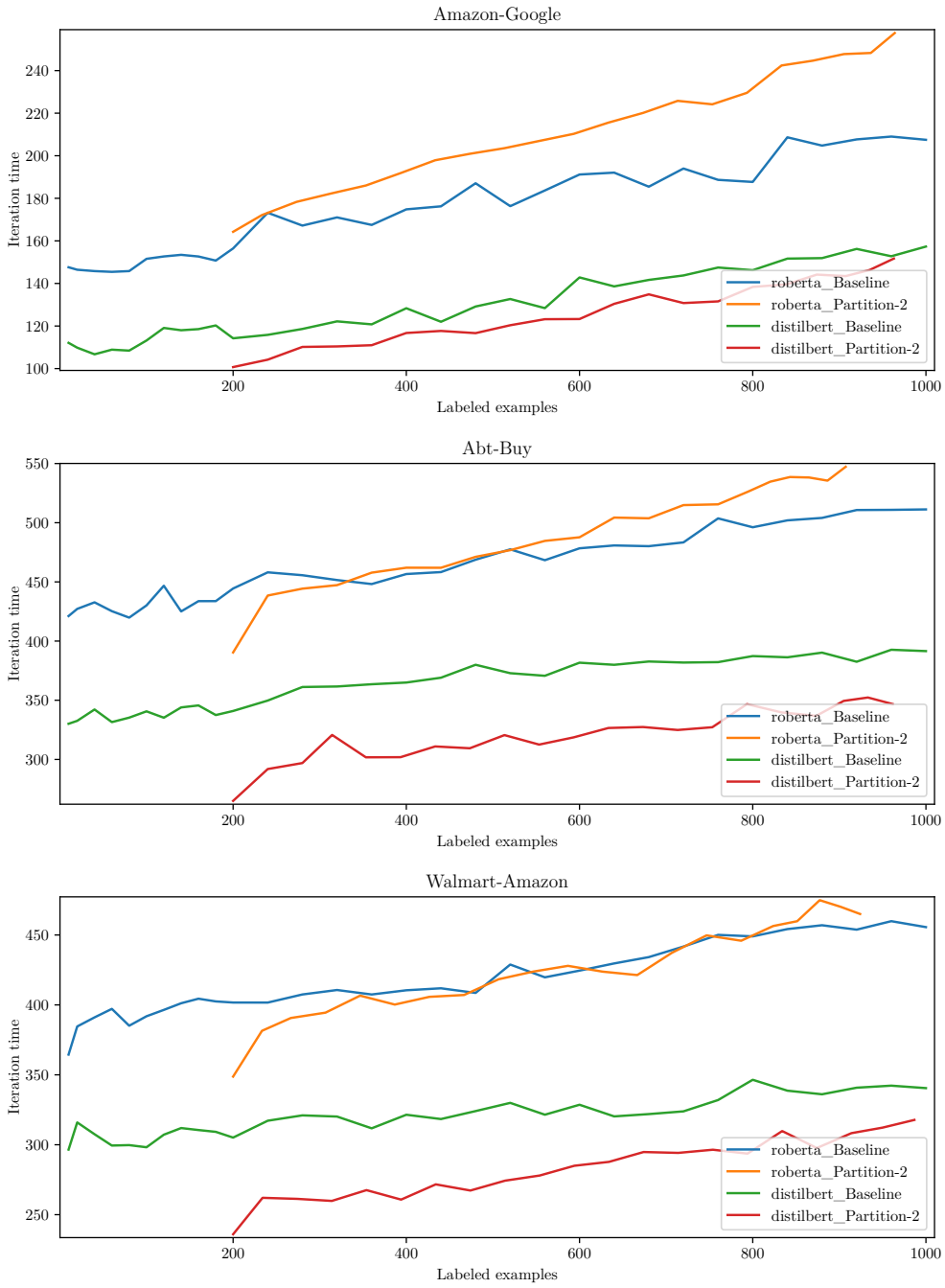


Figure 5: (Figure continues on next page).

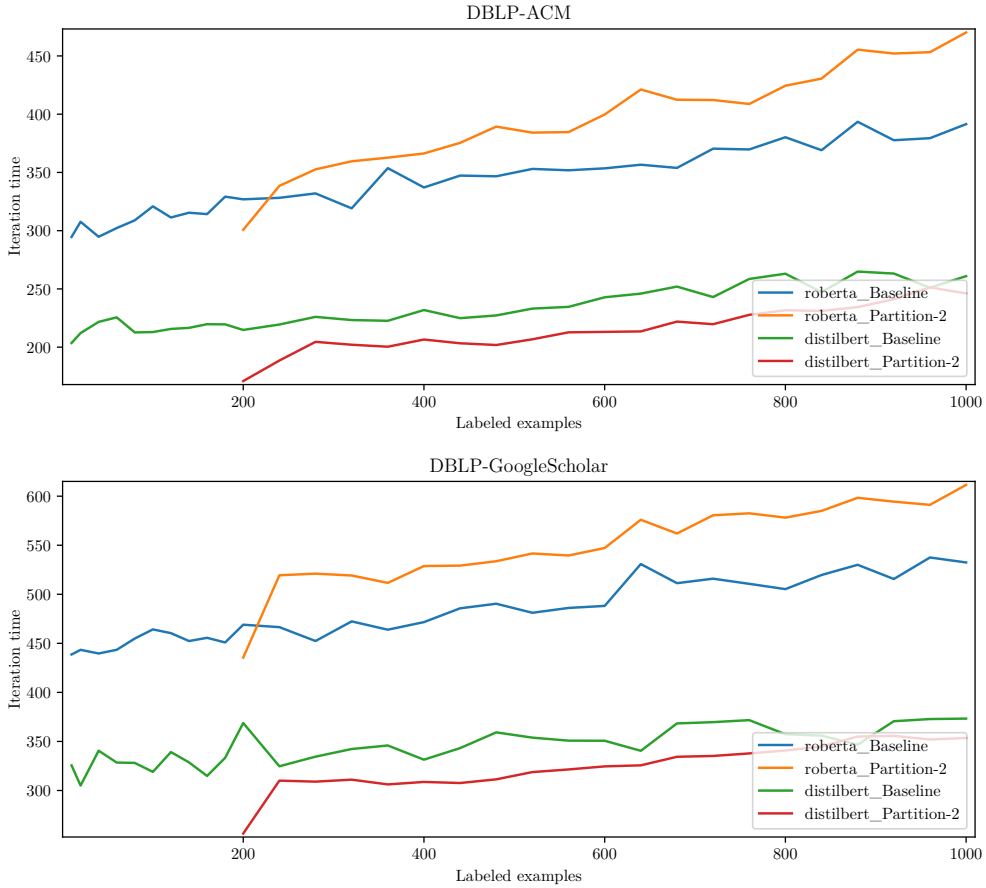
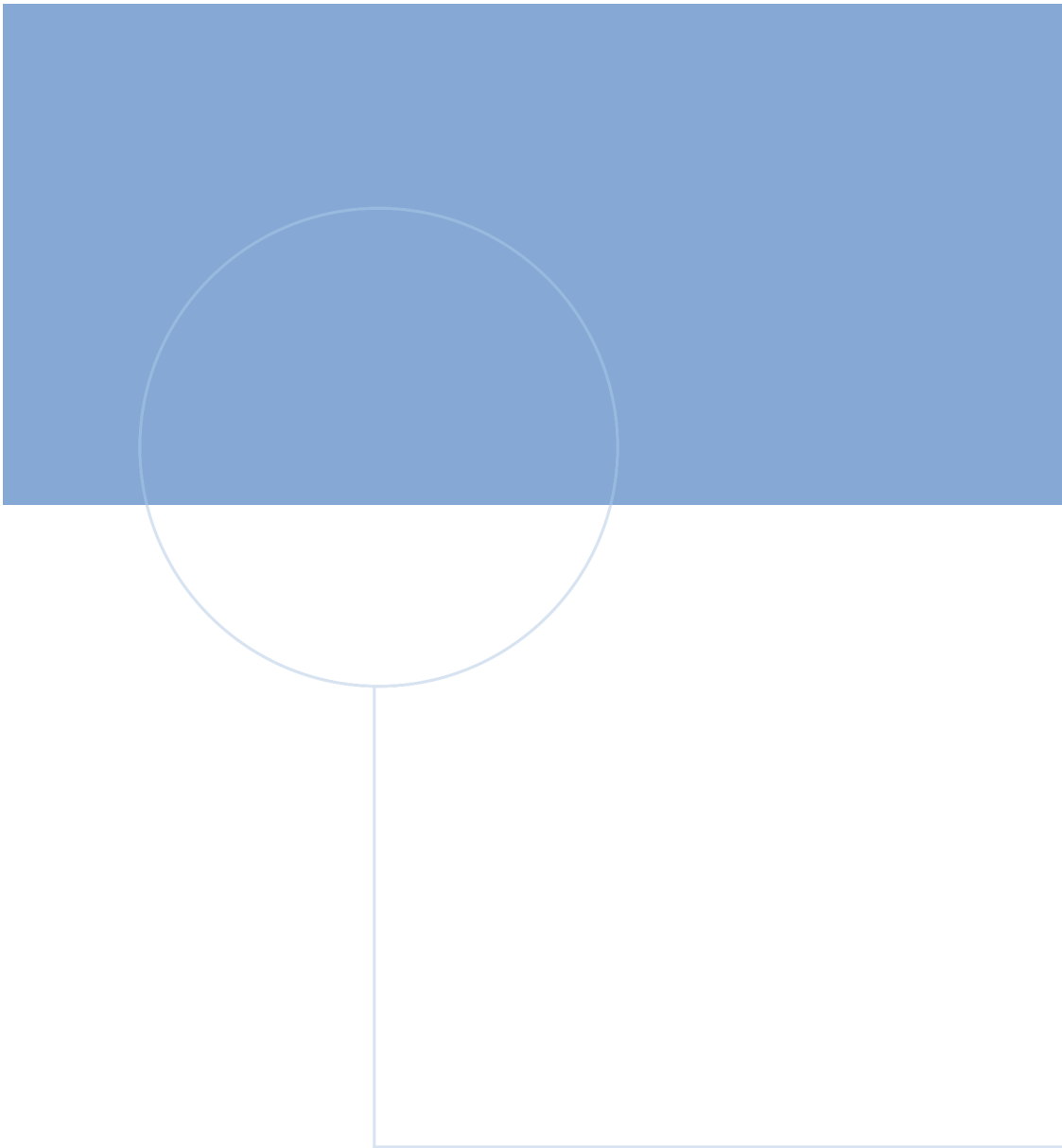


Figure 5: Progressive iteration time for a selection of active learning query strategies comparing DistilBERT and RoBERTa as the TPLMs.



NTNU

Kunnskap for en bedre verden