Active Learning For Integrating Sparse Datasets: A Comparison of Different Query Strategies and Learning Algorithms

Master Thesis

presented by
Bengi Koseoglu
Matriculation Number 0691735500

submitted to the
Data and Web Science Group
Prof. Dr. Christian Bizer
University of Mannheim

August 2019

Abstract

Data matching is the task of identifying and matching records from different datasets that refer to the same real-world entity. One of the fundamental issues in data matching is creating a training dataset where pairs in the dataset are assigned a label representing their match status. This often involves highly time-consuming manual labeling. In order to overcome this, active learning algorithms can be employed where the algorithm chooses the instances it wants to learn from and therefore reaches the same accuracy while using significantly less amount of data. In the thesis active learning has been applied to match products collected from different websites against a manually prepared product catalog. This task involves certain challenges which haven't been addressed widely in the literature that includes; finding appropriate strategies to handle missing values prior to data matching, trying different query strategies for data matching tasks and employing different learning algorithms in combination with active learning strategies.

Our approach to this task includes proposing and using two new strategies to handle missing values in data matching, calculating similarities between records using Levenshtein, Jaro-Winkler and Jaccard similarity metrics, using active learning strategies of uncertainty, query by committee, density weighted and random sampling for querying instances in combination with support vector machine, logistic regression, decision tree and random forest machine learning algorithms. Additionally, a variation of batch active learning has been employed. Feature selection with F Classification has also been embedded into the active learning algorithm. The results of the applied algorithms have been analyzed from the aspect of accuracy measured by F1 score, convergence and time. The main result of the thesis shows that uncertainty sampling that uses random forest, Jaccard similarity calculated features and handles missing values by adding new columns to the feature vector while assigning 1 if either of the fields under comparison is missing, is the best model over all datasets. The model has an average F1 score of 0.8, reaches convergence after 13 iterations and completes training in 29 minutes, over all datasets. The code used in the thesis can be found in `https://github.com/bngksgl/Master_Thesis`.

Contents

List of Algorithms

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Data Matching

Today, we are swimming in a massive volume of data both structured and unstructured, created by humans, businesses, government organizations and machines. It is approximated that 2.5 quintillion bytes of new data is being created every day and that makes one thing certain about the future, the data in the world will only keep on increasing [41]. This ever increasing amount of data brings many benefits to businesses and individuals, including more accurate and actionable insights, cost savings and improved products and services. But this also comes with its own challenges. These challenges can be categorized into two categories; how to store and how to analyze this massive amount of data. The first challenge is usually associated with the databases and data warehousing solutions, which can be thought as technologies that aggregate the large amounts of structured and unstructured data. The second challenge involves business intelligence and data mining, which aims at analyzing the data that is provided by the database systems, in order to generate daily or monthly reports and to discover actionable insights. For data mining projects and business intelligence reports, usually data from multiple sources need to be integrated and matched, in order to improve data quality, enrich data sources and gain better insights.

Integrating data from different sources involves three steps; schema matching, data matching and data fusion. Starting with schema matching, it is a method of finding mappings between the attributes, database tables or conceptual structures from different sources that are similar to each other linguistically or represent same information [3]. Secondly, data matching is the task of identifying and matching individual records from disparate databases that refer to the same world entity [14]. Data matching is also known as entity matching, record linkage, identity resolution

and deduplicaion in the literature and throughout the thesis, these terms are used interchangeably. Thirdly, data fusion is the process of integrating information that have been classified as matches into a one single clean and consistent record to produce a unified data entry for the entity [12]. Along this thesis, the focus will be on data matching.

In data matching, the records to be matched refer to real-world entities and usually involve working with data related with people, products and academic papers, such as employees in a company, customers of a bank, products offered by a business, or citations of scholarly articles. With these type of data, data matching can be used, for instance to find unique number of employees working in a company, to eliminate duplicate records of bank customers since customers can have several records due to name variations and address changes, to create a consolidated product catalog or to match citations from different scholarly articles. With the emergence of internet and consequent growth in the e-Commerce sector in the past two decades, data matching has gain a major importance, especially in the area of comparative online shopping. Comparative online shopping websites, for example, price comparison engines, allow the user to compare prices of products that are offered by various retailers. This raises a major challenge, as the same product that is offered by different websites, can have different representations. For example, an 'Apple Macbook Air 128 GB Silver laptop', may be offered in Amazon as the 'Apple MacBook Air (13 inches, 1.8GHz dual- core Intel Core i5 processor, 128GB) - Silver', whereas in the ebay it might be put online as 'Apple 13 MacBook Air i5 8GB RAM 128GB SSD (Mid 2017, Silver) A1466 Model'. It is highly crucial for the success of the comparison engine to be able to match the products from different websites to the same entity, so that the user who is trying to determine which vendor to purchase the product from can get accurate information [4].

The task of identifying entities that refer to the same real-world entity can be a difficult task to be achieved due to number of reasons. To start with, most of the databases don't have unique identifiers, such as GTIN or consumer product codes. If these codes were available the data matching task would have been a simple join operation in SQL. But since these identifiers are missing, the matching task comes down to finding similar attributes across databases and calculating the similarities between these attributes to match candidates [14]. Secondly, data matching task between two databases involves comparing one record from one database to all the records in other database to determine the matches of records. This makes the problem highly computationally expensive. For instance, if one has two databases with 10.000 records each, this would equal to making 100 million comparisons. Lastly, in most of the data matching applications, the match status between two records are unknown. This poses additional challenges, as, the ground truth here is

not readily available which is different from other machine learning applications, where the target data is available without additional effort. Therefore, in order to evaluate the outcome of the data matching process, one would need to create its own ground truth, which involves costly manual labeling.

The general idea behind entity matching is that the closer (similar) two records are, the more likely they refer to the same real-world entity [14]. In order to measure the closeness, different similarity metrics can be calculated on matching attributes between different sources, such as, Jaro-Winkler, Levenshtein and Jaccard. For the purpose of making the final match decision, unsupervised or supervised techniques can be employed. Unsupervised approaches involves clustering (grouping) similar records together based on the similarity vectors, where the formed groups are very small and generally refer to one entity [26]. Supervised approaches to entity matching includes probabilistic matching models and supervised learning models. Among these supervised classification approaches, active learning stands out as it provides a solution to the issue of manual labeling of the training dataset. The main idea behind active learning is to label only the data instances that matter the most, since not all instances in the data carry the same quantity of information. Therefore, by making the algorithm choose the instances that it wants to learn from, a model that performs better than traditional supervised machine learning models can be reached, while using significantly less amount of labeled training data. The algorithm achieves this by selecting a subset of available examples to be labeled next from a pool of yet unlabeled instances then sending these examples to be labeled by an oracle, which is usually a human annotator.

In the thesis, active learning has been applied to three different product matching datasets that are constructed using information gathered from web. For the data matching scenario, N:1 matching is used, where multiple datasets are matched against one main database, in this case all the crawled products from web pages are matched against a gold standard of products in each category. The datasets already have an existing ground truth of more than 75000 matchings that are prepared manually, which gives an opportunity for applied active learning algorithms to be properly evaluated. However, these datasets are sparse datasets, which contain a lot of missing values. This poses additional challenges as for successful data matching algorithms, missing values need to be handled properly before employing data matching models. This thesis focuses on active learning approaches for integrating sparse datasets and involves applying various learning algorithms, along with different active learning query strategies that evaluate the informativeness of instances from different approaches.

1.2 Problem Statement and Research Questions

The thesis attempts to solve three big issues that can be encountered while applying active learning to data matching tasks; (1) how to handle sparse datasets, (2) which active learning query selection strategies to use and (3) which learning algorithms to employ.

First of all, most of the previous active learning applications have been applied to the dense datasets and when encountered with missing values, they have been filled with the most common value of the corresponding column or have been totally ignored. There are only few researchers that really focused on how to handle missing values in the concept of data matching. Handling missing values appropriately is highly crucial for data matching tasks, since it may effect the success of the algorithm. For instance, currently most of the similarity metrics assign 0 if one of the fields to be matched is missing. However, these current algorithms also assign 0 if two fields under comparison are completely different from each other. These two cases should be distinguished from each other, since having missing values and having different values are not theoretically equivalent. Due to this fact, in this thesis two new missing handling approaches are being proposed. The first proposed approach solves this problem by adding new columns to the feature vector, while assigning 1 if either of the fields under comparison is missing. The second proposed approach solves this problem by assigning -1 to the similarity if either of the fields under comparison is missing, without increasing the number of columns.

One of the main challenges of active learning is determining an appropriate query strategy. The query strategy determines how the pairs are selected from the unknown matches for additional manual labeling and therefore have a direct effect on the training dataset, the model and its accuracy. In the case of entity matching, most of the previous work with active learning, has focused on using query by committee strategies, with little emphasis on to the other querying strategies. Therefore, the second goal of the thesis is to fill this gap in the literature by exploring different query selection techniques like uncertainty sampling and density weighted sampling.

The last objective of the active learning algorithm is to try and explore learning algorithms in combination with active learning strategies. Similar to active learning, there is not a single machine learning algorithm that is proven to work best for every problem. Therefore, often multiple different learning algorithms are applied to the dataset and the best one in terms of accuracy is chosen. Thus, in order to have good data matching models, it is crucial to find the model that works best for the given task. Additionally, active learning uses machine learning algorithms to determine which instances to be queried, by for instance, looking at the uncertainty regions determined through the specified machine learning algorithm. That is why

the success of the machine learning algorithm also depends on choosing the appropriate learning algorithm for the task. Therefore, the third goal of this thesis is to try different algorithms within different active learning settings.

1.3 Thesis Overview

The thesis consists of 7 chapters, including the introduction. Chapter 2, presents the previous work in the literature related to data matching, active learning and active learning in data matching. Chapter 3 gives a detailed technical overview to the research problem by explaining the similarity metrics, feature selection methods, machine learning algorithms and active learning. In Chapter 4, datasets that are used in the thesis has been analyzed from the aspects of attributes, sparseness and target variable. Furthermore, in Chapter 5, how the elements explained in the Chapter 3, are used to solve the problem at hand, have been described. This Chapter further describes the methods used in the thesis to solve the problem of data matching with active learning and gives a detailed explanation of used techniques and algorithms, while explaining the reasoning behind it. The results that have been achieved by the proposed approach has been discussed under Chapter 6. The last chapter of the thesis is Chapter 7 where the thesis is summarized and future work has been discussed.

Chapter 2

Literature Review

2.1 Data Matching

Data matching is finding records of information that are believed to refer to the same entity. If these records are available within the same database or file, then its usually refer to as data deduplication, whereas if the records are separated into different files, it is called data matching, record linkage or identity resolution. There are two approaches to data matching; deterministic and probabilistic. Deterministic record linkage involves assigning a match status to a pair if each element in the agreed key are exactly the same [30]. In the thesis setting, the simplest deterministic record linkage strategy would be to pick for instance, GTIN numbers as the key which are known to be unique for each product and declare records sharing the same value as the same product. However, things are not so simple in reality, since there are many missing values in GTIN numbers and therefore this is not possible. One way of overcoming this issue would be to incorporate additional information to the key like product name, product brand, operating system, product height etc. However, this is again not the best approach since there might be spelling variations in the attributes and this may hinder the deterministic matching. For an example, two products in phones that refer to the same entity can have different names in different pages like 'Iphone 6s 32 gb' and 'Iphone 6s 32 gigabyte apple'. Under the deterministic record linkage approach, even though the rest of the attributes in the key are exactly the same, these products would not be a match, because it strictly requires every record in the attribute fields to be exactly the same.

This issue can be solved via probabilistic record linkage approach. Probabilistic record linkage, links two files by computing a weight for each attribute used in matching, based on its ability to correctly determine match status of pairs and then using these calculated weights to determine the probability of the records referring

to the same entity [56]. It is different from deterministic linkage, since it involves probabilities and it doesn't require every attribute used for matching to be the same. Therefore, the pairs that have agreement on all but one of the attributes, like in the above example, will likely to be considered as a match. Of course, it also depends on the assigned weight to the attribute. Previous work on probabilistic record linkage approaches have been further examined below from three aspects; early work, which briefly describes the history of data matching by explaining early statistical approaches to probabilistic record linkage, machine learning, which expresses machine learning approaches to data matching and approaches to sparse datasets, which gives an overview of different missing handling approaches to data matching.

2.1.1 Early Work in Data Matching

Data matching is not something new and the first examples can be seen in literature as early as 1946 with Halbert Dunn, where he introduced the 'record linkage' term to the literature for the first time [23]. He proposed the idea that every person in the world should have a book that starts with birth and end with death, where the inner pages are denoted with the events in that person's life such as marriage, graduation etc. In his paper, record linkage is defined as the process of gathering the pages of this book, so in more broad terms gathering information about a particular person or family. His definition of record linkage, is somewhat similar to what is being refer to today. During 1950's, the first real application of record linkage have been made by Newcombe, Kennedy, Axford and James, to link birth records to marriage records automatically using a computer [45]. In their work, a probabilistic approach to record linkage has been followed where phonetic Soundex encoding has been applied to surnames of records to handle the spelling variations that might have occurred due to human error. Based on Newcombe et al.'s idea, two statisticians Fellegi and Sunter mathematically formalized the probabilistic record linkage problem in 1969 [28]. Their approach involved taking a pair of records from product space AxB, where A and B refer to two different files, and then generating comparison of records by assigning a set of encodings like name is the same, name disagrees, name missing on one record etc. After the comparison vector is created, it is used to make a decision regarding the match and non match status of records using a probabilistic approach. The mathematical model that was developed by Fellegi and Sunter, set the theoretical framework for data matching and their work has become the basis for many data matching systems and it is still being used today [14].

During 1990's, there was a significant increase in the number of researches in data matching, which can be associated with the increase of computational power

of computers and the increase amount of data. Among them the work done by William E. Winkler and his colleagues become prominent. Winkler used Expectation Maximization algorithm inside Fellegi and Sunter's model to estimate parameters without assuming conditional independence between attributes [70]. Further, he extended the Jaro similarity metric introduced by Matthew A. Jaro by making strings with similar beginnings have higher similarity score [68]. He put his theoretical work into practice by applying these new methods on 1990 U.S. Decennial Census [69].

2.1.2 Machine Learning Approaches in Data Matching

Machine learning can be described as an application of artificial intelligence (AI) where computers can learn and improve themselves from data using statistical analysis, without being explicitly programmed. Machine learning algorithms are often categorized as unsupervised and supervised. The goal of unsupervised machine learning is to learn more about the data by modeling the underlying data structure or distribution, and therefore doesn't require the input data to have corresponding output values. On the contrary, the goal of supervised machine learning is to predict the output variables as accurate as possible from the given input data and therefore require the data to have corresponding output values. Machine learning has many application areas including; classifying images, classifying text documents, chatbots, speech recognition, predictions related with financial services like stock precises and detecting fraud and data matching.

Starting from 2000's, a shift towards machine learning applications to data matching problem can be seen in the literature. Machine learning algorithms are capable of handling only binary or categorical comparison vector attributes, which gives additional flexibility to experts working in entity resolution area, since this probabilistic record linkage models that are proposed by Newcombe et al. and then developed further by Fellegi and Sunter, and Winkler, are only capable of working with numerical attributes [25]. Machine learning applications to data matching can be evaluated in three branches; unsupervised algorithms, which doesn't require a training dataset, supervised algorithms, which requires a training dataset and hybrid methods, which combines both supervised and unsupervised methods.

Unsupervised Machine Learning

One of the first applications of unsupervised machine learning models to data matching can be found in 2000, where clustering method has been applied to bibliographic citations in order to divide them into groups in a way that every citation in the group refers to the same entity, in this case to the same paper [42]. For

clustering, Greedy Agglomerative Clustering has been used with canopies method where distances between references are measured with edit distance similarity calculations. Their work has been further improved by Cohen and Richman in 2002, by incorporating Levenshtein and TF-IDF similarity metrics into clustering with canopy and applying it on other datasets that has been gathered from web such as organization names and restaurants [16].

Supervised Machine Learning

Application of supervised machine learning algorithms to data matching can be first seen in the literature by the research conducted by Cochinwala and his colleagues, in 2001. Their work involved identifying customer records in two databases that refer to the same entity by using statistical methods to reduce the complexity of the matching task and then applying machine learning algorithm to generate the matching rules [15]. For calculating the similarities between records they used edit distance similarity metrics and for the machine learning algorithm, they used decision tree classification method. In the end with their work, they managed to reduced the complexity by %50 while significantly increasing the accuracy.

Following their study, in 2002, Bilenko and Mooney incorporated supervised machine learning into their data deduplication system MARLIN (Multiply Adaptive Record Linkage with INduction) [5]. MARLIN consists of two steps; using different similarity metrics that are trained to determine the similarity of different field values and detecting duplicate records using the created similarity vectors with Support Vector Machine classifier. For similarity calculations MARLIN uses both learnable and regular similarity metrics of Levenshtein and Affine. The idea behind learnable similarity metrics is to adapt the string similarity metric to a particular domain. For instance, deleting 'Corporation' from a variable about company names may make sense, but deleting 'Corporation' from website names may not be the best approach. Rather than hand-tuning the similarity metrics for each attribute to incorporate these differences, weights can be learned using Expectation Maximization algorithm and small corpora of hand-labeled examples [5]. This idea has been first proposed by Ristad and Yianilos, with Levenshtein similarity metrics [52] and further adapted to MARLIN system with Affine similarity metrics [5]. Both of these studies has shown that learnable similarity metrics perform better then the pure similarity metrics.

Hybrid Methods

The majority of practical machine learning applications uses supervised learning, as it is capable of predicting future based on the given input data and corresponding

output values. However, in some cases, obtaining the output data requires highly manual costly labeling. This is also the case for data matching, as one has to assign a label to the every pair in the dataset, regarding their match status. In order to overcome this problem, both the unsupervised and supervised learning algorithms can be used together in data matching setting. These methods are known as hybrid methods in the literature and the general idea is to use unsupervised learning to form a training dataset and then using this created dataset to learn a classifier.

Hybrid methods has been first used in the data matching by Elfeky et al. via their record linkage toolbox named TAILOR. Their method includes creating comparisons vector with Hamming distance, edit distance, Jaro, N-grams and Soundex Code similarity metrics and then using this comparison vector to cluster the pairs with k-means clustering algorithm into three groups; matched, unmatched and possibly matched [25]. By using clustering they are hoping that perfectly matched record pairs that agree in all fields in a comparison will be assigned to one cluster, and pairs that do not agree in all comparisons will be assigned to the other clusters. Later the comparison vectors that are grouped as matches and non matches are used to train a decision tree classifier. By incorporating k means clustering and decision tree classifier, they found out that the machine learning record linkage approach outperforms the probabilistic record linkage models respect to accuracy. Christian improved the approach that was proposed by Elfeky et al., by using iteratively trained SVMs. For selecting pairs to be classified, he applied two methodologies; threshold-based selection, where two distance thresholds have been determined manually based on calculated similarities, to determine matches and non matches and nearest based selection, where pairs that are closest to the matches and non matches are chosen according to a distance metric [13]. Then, a sample from both of these separated classes are selected to form the training data and the first SVM model has been trained accordingly. This first initial training dataset will only contain weight vectors that are either close to matches or non matches, and therefore will be highly linearly separable by the SVM. Therefore, in the second iteration, the pairs that are not chosen at the first iteration, will be added to the train dataset and second SVM will be trained on this enlarged training dataset. This will continue until all pairs are classified.

2.1.3 Approaches to Sparse Datasets in Data Matching

In data matching it is highly crucial to clean and standardize the data since the data to be used in data matching tasks can vary in structure and content. These data cleaning and standardizing activities include; removing unwanted characters and words, adding attributes derived from already existed attributes, lower casing the strings to be compared and handling missing values. Among these operations,

handling missing values stands out as it can have a significant impact on similarity calculations. For instance, if one of the strings to be matched is missing, then the similarity will be 0. However, this is problematic since if two strings to be matched are completely different, then the similarity will also be 0. Therefore, a distinction should be made between records that have 0 as their similarity because that they are completely different, or because one of them is missing.

Handling missing data is a fundamental ongoing issue in data matching and require parameters used in data matching problems to be adjusted to handle missing values [14]. Most of the researchers in the data matching area are focused on entity matching on dense datasets, where the datasets do not contain too many missing values. However, in reality, its common to have multiple missing values in the datasets to be matched, especially if the data is coming from different e-shops. This is due to the fact that different e-shops may have different attributes for the entity to be matched and therefore one web site may have information related with for instance phones' operating system whereas one website may not. As a result, when data is crawled from websites into a schema, many missing values can occur. This makes matching product data from different e-shops a particularly difficult task [49].

In the literature, most researchers choose to ignore missing values or fill them when they encounter with missing values during data matching [66, 67]. However, there are two entity resolution researches specifically made on sparse datasets that worth mentioning. The first research proposes two different methods to handle missing values; modifying the similarity metric to take missing values into account and performing data permutation before calculating the similarities. Their first proposed method consists of first using TF-IDF metric to assign weights to features, then using soft TF-IDF combined with Jaro-Winkler metric to account for spelling variations between records and then adjusting the metric for sparsity based on whether or not the record has missing entries [67]. Their second proposed approach, involves imputing the missing data with the most frequent value of the column before calculating the similarity metric. This second approach has been used by many other researchers in the field as well, due to easiness and non complexity.

More over, the second research proposes changing the algorithm structure to take missing values into account. Their research is based on modifying their already well established GenLink algorithm to handle sparse datasets. GenLink is a supervised learning algorithm that learns expressive linkage rules like attribute specific preprocessing operations and comparisons by using genetic programming for a given entity matching task [34]. In their paper, they proposed three different algorithms that is based on original GenLink model; GenLinkGL model which handles missing values by pivoting around them while grouping the linkage rules

according to the product characteristics, GenLinkSA model that handles missing values by ignoring and penalizing them using selective aggregation operators and GenLinkComb algorithm that uses a combination of both of these approaches to handle missing values [49].

2.2 Active Learning

Active learning is a relatively new algorithm which is introduced and further formalized in the beginning of 1990's. It is defined as, any form of learning in which the learning algorithm has some authority over the inputs which it trains on [17]. Simply put, in active learning, the algorithm decides on the instances that it wants to learn from. These instances are usually consist of the most informative instances of the dataset and therefore by training only on these chosen instances, the algorithm is capable of achieving high accuracies. Active learning involves an iterative process of querying an instance according to some criteria, sending the queried instance to the oracle to be labeled, adding the new labeled instance to the training dataset, re-training the algorithm like a normal supervised learning algorithm on the extended training dataset, and repeating the process until a threshold of accuracy or a specified number of queried instances are reached. In the beginning of the algorithm, the querying is being done using only an initial set of labeled instances known as seeds, and after each iteration, the number of instances that had been queried and labeled are added to the initial seeds to create a larger training dataset.

Active learning is mostly used in the literature to overcome the issue of assigning labels to a dataset, which is often involves highly costly manual labeling, like in the case of entity resolution. Previous work of active learning mainly involves proposing new strategies for querying instances from a dataset, generating efficient seeding strategies, introducing new methods to make active learning faster and incorporating feature selection into active learning setting.

2.2.1 Early Work in Active Learning

Active learning is a relatively new method that is introduced and further formalized in the beginning of 1990's. Prior to active learning and selective sampling methodologies, sampling was being done by randomly selecting instances from dataset and the machine learning algorithm was simply a passive learner that assumes that all the instances in the datasets come with labels. The concept first introduced by Atlas et al., with 'selective sampling', which is a form of directed search that examines the information already given and determining a region of uncertainty, where the learner believes miss classification is still possible, and then selecting

instances specifically from this region [2]. In their work, they compared selective sampling with random sampling on neural networks and found out that as the number of labeled examples increases, a more exponential drop in generalization error is seen in the dataset where instances are chosen with selective sampling. Later, they formalized their theory further, and introduced the concept of active learning [17].

2.2.2 Query Selection Algorithms in Active Learning

There are various ways an algorithm could choose to select an instance to be queried. One of the most popular querying algorithm is uncertainty sampling, which is somewhat similar to the selective sampling mentioned before. Uncertainty sampling is a querying method that is introduced in mid 1990s by Lewis and Gayle, that iteratively select examples by fitting a classifier that is capable of both predicting a class and providing a measurement of how certain that prediction is, and then using the classifier to select new examples where the predicted class' membership is most unclear [39]. This unclarity is measured by the posterior probability P(predicted class | instance vector), that is assigned by the classifier to each instance of the dataset and instances whose posterior probability are closest to 0.5 are selected to be queried since these are the instances the classifier is most unsure of. Lewis and Gayle used uncertainty sampling on text classification with Naive Bayes learning algorithm and reached the same accuracy as random sampling while using %100 less labeled data [39]. In the same year, Lewis and Catlett, applied uncertainty sampling to text classification with decision trees and again received successful results compared to random sampling [38]. Over the years, uncertainty sampling received lots of attention from the literature and it has been used in a combination with maximum entropy [20], logistic regression [31, 65, 57], support vector machines [36], decision trees [24] and random forest [11]. Uncertainty sampling is based on using a single classifier for querying instances and making predictions.

Query by committee (QBC) is another querying method that is widely used in the active learning literature. It is based on the idea of finding instances to query by employing more then two classifiers that is applied on to the same dataset and the instance these models disagree the most is chosen as the most informative, since it is believed to restrict the version space the furthest. This idea has been first proposed by Seung et al. with naive bayes setting [63] and further formalized by Freund et al. [29]. The used learning algorithms in QBC could be the same algorithm with different parameter settings or could be totally different algorithms. Further, there is no general rule of thumb as to the number of committee members to be used [59]. In the literature, query by committee has been mostly used in with

decision trees [55] and support vector machines [1].

So far two querying strategies have been discussed; uncertainty sampling where instances are selected based on their uncertainty and QBC where instances are chosen based on its ability to reduce the version space. These two methods have a major drawback due to the fact that they choose instances by considering only a single data instance, without taking the whole dataset into consideration and this in return may lead them to confuse an outlier with an informative example [59]. Additionally, these models concentrate on finding instances based on models confidence and by labeling these instances prediction accuracy will eventually be increased, but neither of these methods directly focuses on finding the instances that yields with the best predictions. In order to overcome these disadvantages, other querying methods which select informative instances based on their impact to the model have been proposed. By employing these approaches, instances that have a direct effect on the model's accuracy and therefore most likely to reduce the feature error, are chosen. These query methods include; expected model change, expected error reduction, variance reduction and density weighting.

To start with, expected model change, is a querying algorithm that is first introduced by Settles et al. and it is based on selecting instances that once the labels are known, would cause the greatest change to the current model, regardless of the resulting query label [62]. Expected error reduction is a querying method that is introduced by Roy and McCalum, which chooses instances that reduces the expected generalization error measured by log loss the most [53]. They applied this methodology for text classification using Naive Bayes and found out that sampling based expected error reduction performed better compared to QBC and uncertainty sampling with their used setting. However, expected error reduction involves estimating the future error over all unlabeled examples and all possible labels the examples could be assigned for every query, which increases the complexity and the training time of the model, drastically [60].

Moreover, generalization error of the model can be reduced indirectly by minimizing the output variance. Variance reduction is a query selection methodology that has been proposed by Cohn et al. which is based on choosing examples that reduces the model variance the most by maximizing the Fisher Information [18]. Even though, variance reduction is less computationally expensive then expected error reduction, it still requires high computational power and time compare to uncertainty and QBC as it involves inversion and multiplication of the Fisher Information Matrix. Additionally, it is only suitable for linear, non linear and logistic regression models and how this can be incorporated into decision trees, nearest neighbour methods, and a variety of other standard machine learning and data mining techniques are less clear [60]. Furthermore, these issues can be overcame by using density weighted sampling (DWS) method that is proposed by Settles and

Craven. DWS is a query selection method which aims to select instances that are both informative and represent the underlying data distribution [61]. In their work they compared uncertainty sampling and QBC using different parameter settings with expected error reduction, variance reduction and DWS, for sequence labeling tasks and they found out that even though there is no clear winning query strategy, DWS stands out among other models in terms of its average accuracy in all of the used datasets. In terms of timing, they concluded that uncertainty sampling followed by QBC are the least computationally costly methods whereas expected error reduction and variance methods are the most computationally costly methods which require significantly more time.

2.2.3 Seed Generation in Active Learning

Active learning requires initial set of labeled instances to start learning, known as seeds. For successfull active learning algorithms, it is highly crucial to select appropriate seeds according to the problem setting [65]. If a small and biased sample has been chosen for active learning, there might be some parts of the representational space that can not be learnt and this would in return could effect the models accuracy and convergence.

In order to overcome this issue, some solutions have been proposed in the literature, which most of them applied to text datasets. For instance, Tomanek and his colleagues, found out that active learning recovers better from unfavorable seed sets when sentences are used instead of tokens in sequence text learning problems such as named entity recognition (NER). They argued that this is because by using sentences for initial seeding, the active learning algorithm selects complete sequences in sentences and since, one sentence can contain multiple different classes, by selecting one sentence for seeding, information from multiple classes can be gathered [65]. With their improved technique, more instances from minority class have been included to the initial seed, and this in return led to a higher learning rate with faster convergence compare to randomly sampled seeding. Dligach and Palmer proposed using Language Model (LM) Sampling for selecting initial seeds, which is an unsupervised method that trains a language model on a corpus of unlabeled examples and selecting examples with low LM probability for seeding [20]. By following this methodology on word sense disambiguation dataset, they were able to select more rare class examples for initial seeding than random sampling, which led to a classifier with faster convergence.

Further more, there is another research that is worth mentioning which is done by Kang and his colleagues from Pusan National University for active learning in text classification. Their idea involves clustering the unlabeled dataset by using Kmeans algorithm and then from each cluster, choosing instances that are closest

to the cluster centroid [37]. By using this approach, they assume that each cluster would consist of similar examples that likely to belong to the same category or class, and by selecting an example from each cluster, all the version space can be included in the initial seeding. They concluded that by selecting seeds using clusters, more rare examples are selected and this in return led to a faster convergence of the classifier.

2.2.4 Batch Mode Querying in Active Learning

In normal active learning setting, in order to see the effect of the queried instance on the algorithm success, after labeling one instance and adding it to the training dataset, a model is trained on the new label added training dataset. However, it has been observed by many researchers as well as the author of the paper, that it is very time-consuming to retrain the classifier whenever a new example is added to the training dataset [8, 31, 32]. To overcome this issue, different solutions have been proposed by researchers in the literature. Their ideas based on selecting a set of examples for labeling, before repeatedly running the training algorithm. This set of examples is called *batch* in the literature and therefore these methods are known as Batch Mode Active Learning. The key challenge here is finding instances that are dissimilar to the other selected examples while at the same time uncertain to the current classification model, in other words selected instances should provide information that are unique and should be informative [32].

In order to achieve this Brinker, proposed selecting instances using SVM where the selected instances in the batch have minimal distance to the classification hyperplane adjusted by the diversity measure which considers angles between the hyperplanes [8]. His method provides faster training time to attain a level of generalization accuracy, while only requiring small amount of additional computational time. Furthermore, Hoi and his colleagues suggested using Fisher information matrix to select the best subset of unlabeled examples. Their work involves using Fisher information matrix as a measurement to model the uncertainty, and selecting instances that reduce the fisher information most [32, 31]. They applied this technique with SVM and logistic regression to medical image classification and large scale text categorization tasks.

2.2.5 Approaches to Feature Selection in Active Learning

Different similarity metrics may assign distinct similarity scores to the same pair, as they look at different aspects while evaluating closeness. Therefore, one similarity metric can be a good indication of closeness for one attribute and might not be as good for others. This makes selecting appropriate similarity metrics highly crucial

for data matching tasks, as they are the only used measure to calculate the closeness between two entities. In the literature a lot of different similarity metrics have been used including but not limited to; Levenshtein [5, 16, 19], Jaccard [1, 19], Jaro-Winkler [25, 67, 19], Soundex [25, 28, 55], TF-IDF [16, 67, 67, 64, 19], Affine [5]. Also, as mentioned before some researchers proposed the idea of learnable similarity metrics to adapt the string similarity metrics to a particular domain [52, 5]. Furthermore, not every calculated similarity metric is useful for the matching task and this in return may hinder the performance of the algorithm in terms of speed and accuracy [71]. In order to overcome this issue and to find the best suited similarity metrics for each attribute, feature selection could be done. In pure machine learning algorithms, feature selection has been done only once before actual machine learning algorithm, however iterative nature of active learning and increasing amount of instances in the training data after each iteration, requires feature selection to be done dynamically inside the active learning.

Feature selection in active learning is still an ongoing research area in the literature which has not received too many attention so far. One of the earliest research in this area has been done by Joshi and Xu using GainRatio for selecting features with SVM for blog post classification. Their approach includes selecting top 100 features ranked by the GainRatio algorithm from 20 instances that has been chosen by the active learning to be labeled, then adding these selected features to the feature space after each iteration and learning a classification by using all the added features [36]. In order to illustrate the problem further, assume that in the first iteration sample S_0 has been chosen to be labeled by the active learning algorithm. Then top 100 features ranked by GainRatio will be selected using only sample S_0, and the classification algorithm will made on the dataset with $initial\_seed + S_0$ using features FS_0. In the second iteration, the feature space FS_0 will be used to select next batch of samples S_1 to be labeled and the next features FS_1 will be selected using S_1. Then the second classification will be made with $FS_0 + FS_1$ on $initial\_seed + S_0 + S_1$. The process will continue until a threshold is reached. With this approach, they produced worse results then expected, and they found out that its because selected sample S_1 with feature space FS_0 is no longer the decision boundary when its mapped to $FS_0 + FS_1$ [36]. This is also problematic since feature selection is made only with the selected instances, without considering the whole labeled dataset.

More over, in recent years, Bilgic proposed the idea of using Principal Component Analysis (PCA) combined with L1 and L2 regularization of logistic regression, in order to determine which and how many features to be selected, in every iteration of active learning. He named his approach Dynamic Dimensionality Reduction (DDR). DDR includes first using PCA on both labeled and unlabeled dataset to eliminate noisy features and create super features, then these constructed

features are sorted in decreasing order based on their ability to capture variance. In the next step, in order to determine the number of features to be selected, features are added to an empty set iteratively and at each step a model trained with L2 regularization, while being evaluated with L1 norm and the model stops adding features when L1 norm can no longer justify the decrease in the logistic loss [6]. By experimenting his method on text datasets, he reached a comparatively better performance in active learning compared to pure active learning methods and his model didn't suffer from the issues argued by Joshi and Xu.

2.3 Active Learning in Data Matching

Most of the above mentioned researches on active learning have been applied to text analytics applications like text classification and sequence labeling, due to the fact that text analysis is a problem domain where there is abundance of data, and labeling usually involves highly costly manual annotations. Data matching is also another domain where labeling comes with a high cost and therefore it is another area that can highly benefit from active learning applications. There are few researches in the literature that specifically focuses on active learning applications into data matching. These work has been evaluated from three aspects; early work, which explains applications of common classical query selection algorithms like uncertainty and QBC to data matching tasks, active learning applications specifically designed for data matching and active learning applications that uses genetic programming.

2.3.1 Early Work

One of the earliest examples of active learning into data matching has been introduced by Tejada et al. in 2001 with Active Atlas system. Active Atlas system is an object identification system that compares objects' shared attributes in order to identify matching objects [64]. Their proposed system consisted of two steps; computing similarity scores between attributes of two records using cosine similarity with TF-IDF and then learning rule-mapping rules to determine which attributes or combinations of attributes are most important for mapping objects by using a committee of three decision trees in the active learning setting. They used Active Atlas system to identify mappings between two different web sources in company, restaurant, airport, weather domains. Following this initial paper, Sarawagi et al. introduced ALIAS, which is a system for deduplicating datasets by constructing automatic deduplication functions while choosing the most informative pairs via active learning. In their research, for calculating similarities between text fields,

they used edit distance, Soundex and abbreviations and for the querying strategy they choose query by committee approach and constructed five different models of decision trees, support vector machines and naive bayes with different randomized parameter settings [55]. They applied their system on two different datasets related with bibliographies and addresses, and found out that decision trees have the highest success rate in terms of F1 Score. These two proposed systems have two major disadvantages; users can not control the success nor the quality of the algorithm since the systems don't accept an upper threshold of precision or recall and the systems do not scale well with large inputs considering the fact that these algorithms iterate over all record pairs without using any method to reduce the number of comparisons to be made [1].

2.3.2 Active Learning Designed for Data Matching

Active learning algorithms can be specifically tailored to the data matching tasks to increase the accuracy and convergence of the algorithm. The first example of specifically designed active learning algorithms to data matching can be seen in the literature by Arasu et al. [1]. They designed a new record linkage system that doesn't use any already existing query strategies and that employs active learning to maximize recall while ensuring that precision is above a specified threshold determined by the user. They also incorporated blocking, based on calculated Jaccard similarities between record pairs into the active learning system. To be precise, the algorithm starts with an initial rule M with high precision and low recall, and the goal is to find a rule that increases recall while keeping precision above a given threshold. It first applies blocking B between records from databases R and S to reduce the number of comparisons to be made, and in order to find the pairs to be labeled by the oracle, it scans $B(RxS)$ and finds subset of pairs T that satisfies M and then takes a random subset from T to be sent to the oracle to be labeled [1]. The links that are sent to the oracle and labeled as matches, gives an estimate of the rules precision and if most of labeled examples turns out to be actually non matches, a more precise rule is generated accordingly. In the paper Jaccard, Jaccard Containment and equality similarity metrics have been used to calculate similarities between two records r, s and it has been stated that the created algorithm is compatible with decision trees and SVM classifiers. Even though, this procedure of finding instances to be labeled is different from other proposed active learning methodologies, it outperformed Active Atlas [64] and ALIAS models [55] on two large databases of publications and records that is being tested on. However, this proposed model by Arasu et al., suffers from low recall due to the fact that the model only considers false positive examples, pairs that are actually non matches but predicted as matches, while generating rules by searching for instances that

both satisfies the rule M and blocking criteria B, by totally ignoring false negative examples that are not covered by the existing rule M [51].

In order to overcome this, Qian et al. proposed a new entity resolution system called ERLEARN which achieves both high precision and high recall by learning only a handful of rules while using as less data as possible. Their primary hypothesis is, false negatives, pairs that labeled as non matches but actually matches, are crucial for learning multiple rules, whereas false positives are helpful to refine a rule. ERLEARN achieves this by asking the oracle to label two different group of candidate link pairs; *likely False Negative* and *likely False Positive*. The algorithm starts with an initial ER rule R that is applied to the input datasets and *likely False Positive* links which are to be labeled by the oracle, are selected by first removing links that satisfy all the generated candidate rules RR, and then from the remaining links, choosing pairs that have the lowest similarity scores, since the less similar two links are, it is less likely for them to be a match even though they satisfy R [51]. On the other hand, *likely False Negative* links are selected by first creating multiple rules by removing matching predicates from R and applying them into the datasets, and then discarding all links that satisfies R and RR, and from the surviving links, choosing the ones that have the highest similarity scores. After *likely False Positive* and *likely False Negative* are sent to the oracle and labeled, the algorithm can also accept or reject the candidate rules. This learning process continues until user manually cease the system or no candidate rules can be generated that satisfies the precision criteria.

2.3.3 Genetic Programming and Active Learning

In recent years, genetic algorithm combined with active learning has been started to be applied to data matching problems like data deduplication or interlinking data sources in the web of data. The intuition behind this approach is using genetic programming (GP) to explore the vast space of similarity functions between record fields to find the best combination of similarity measure, transformation and aggregation specific to a certain attribute while using active learning to reduce the user effort of labeling data records.

This idea, has been first proposed by Freitas et al. for data deduplication problems. Their method includes first generating all possible pairs of candidate records using blocking, then calculating a similarity metric between each pair based on their common attributes using Jaro-Winkler, Levenshtein, Jaccard, N-gram or TF-IDF, and employing a semi supervised learning approach that is based on genetic programming and active learning with query by committee of 20 functions, that learns how to classify a pair as duplicate or not duplicate [19]. They have also incorporated reinforcement learning into their method in order to differentiate be-

tween the good and bad functions used in the committee by assigning weights. They compared their new proposed method with ALIAS on three different datasets, and reached a better result in terms of accuracy measured by F1 score and amount of labeled data needed to reach a specified threshold of success.

The work by Freitas et al. has been further extended by Isele et al. and applied to Linked Data. They incorporated different similarity measures non linearly along with data translations such as case normalization and tokenization which is applied prior to comparison, into genetic programming setting [35]. For the querying strategy, they chose query by committee and they experimented their method in three different settings; linking citations from *Cora* dataset, interlinking movies between LinkedMDB3 and DBpedia, and interlinking settlements between LinkedGeoData and DBpedia. In all of these settings, active learning reached a high F1 score by only labeling few links. Furthermore, in the same year a similar research that uses genetic programming in combination with active learning to Link Data has been published by Ngomo1 and Lyko. They applied genetic programming in tree like structures to discover link specifications and used query by committee strategy to select links to be labeled by the oracle. They showed that by using active learning, complex datasets can be tackled with more ease and more stable solutions can be reached while at the same time using less data [46].

2.4 Summary

In this chapter, previous research related with data matching and active learning have been analyzed. As can be seen, both the data matching and active learning are two very wide topics which received a lot of attention from the researchers over the years. Data matching started in early 1960's with statistical methodologies and further investigated in 2000's using machine learning techniques. On the other hand, active learning emerged in early 1990's and used in the literature to mainly overcome the problem of preparing training dataset, which often requires highly costly manual labeling. Data matching is a research area where can be benefit highly from active learning, since the created pairs need to be manually assigned a match status. The first application of active learning to data matching arose in 2001, and since then it also received a significant amount of attention from the researchers. The early applications involve applying classical active learning query selection techniques to the data matching task, later these applications tailored specifically to data matching tasks and further improved using genetic programming techniques.

Chapter 3

Theoretical Framework

3.1 Similarity Metrics

It is safe to assume that, the more similar values the two compared records have in common across their attributes, it is more likely that they refer to the same entity [14]. Based on this assumption, the similarity between two records can be measured by calculating the similarity between records' corresponding attributes using similarity metrics such as Levenshtein, Cosine. These calculated similarities are usually between 0 and 1, and represent a degree of similarity between two records. The more closer this value to 1, the more similar two records are. For instance, imagine comparing two records i and j, using only two attributes a_i and a_j that refer to the same feature in both databases. If the calculated similarity $sim(a_i, a_j)$, is 1 then the compared values are identical, on the other hand, if it is 0, then the compared values are completely different from each other. If the $sim(a_i, a_j)$ is between 0 and 1, it can be said that they are fairly similar. Once the attribute similarities are calculated across each candidate pair in our data, a vector also known as *comparison vector* is formed. The resulting comparison vector is then can be used in classification to identify whether two records in the pair, refer to the same entity or not.

The similarity metrics can be classified into three main different categories; edit-based, token-based and hybrid [44]. Edit-based similarity metrics count the smallest number of edit operations that is required to make one string identical to the other string [14]. These include; Levenshtein, Jaro, Hamming and Smith-Waterman. Secondly, token-based similarity metrics are found on the idea of tokenizing strings into characters, n-grams or words, rather than considering the string as a whole and calculating similarity based on equal tokens [44]. This change of methodology introduces certain advantages and disadvantages. For instance, by

tokenizing the string into words, word swaps such as "iphone 6s" and "6s iphone" can be taken into account. However, with this method depending on the tokenization strategy, string variations in words might not be detected. As an example, if one uses tokenization of words, then the similarity between "iphne 6s" and "6ss iphone" will be zero. Token-based similarity metrics include Jaccard, Cosine and Dice similarity metrics. Lastly, hybrid similarity metrics incorporate both tokenization and string similarities based on edit distance calculations. In these methods, tokens are compared but rather than taking only the equal tokens into account, internal similarity functions are applied to compare the tokens. Two well-known similarity metrics in this category include Extended Jaccard similarity and Monge-Elkan measure. In the thesis, Levenshtein, Jaro-Winkler similarity metrics from edit-based distance category and Jaccard similarity metric from token-based category have been applied and that's why only these methodologies have been discussed further under the methodology section. For similarity calculations, strsim package that is developed by Zhou Yang Luo for python has been used [40].

3.1.1 Levenshtein

Levenshtein similarity metric is based on the idea of edit-distance calculations where the number of edit operations to make one string identical to the other string is counted. This calculated edit-distance can then be converted into a similarity measure. In Levenshtein, these operations include insertions, deletions and replacements and the goal is to find the minimum number of operations that is needed to convert one string to the other string [44]. There could be infinite number of operations to convert one string to the other string and this is where the problem raises. In order to find the minimum number of operations, dynamic programming used, which can be defined as the optimization over plain recursions.

Dynamic programming starts by creating a matrix M with dimensions ($|s1|$ + 1) ($|s2|$ + 1), where $|s|$ represent the length of the string. The rows i in matrix M denotes the characters in string $|s1|$, whereas the columns j in matrix M denotes the characters in string $|s2|$. Once the matrix is created, it is filled according to the below written rules [44].

$$M_{i,0} = i \tag{3.1}$$

$$M_{0,j} = j \tag{3.2}$$

$$M_{i,j} = \begin{Bmatrix} M_{i-1,j-1} & if s_{1,i} = s_{2,j} \\ 1 + min(M_{i-1,j}, M_{i,j-1}, M_{i-1,j-1})' & otherwise \end{Bmatrix} \tag{3.3}$$

The first two rules state that the matrix's first row should be filled with the indexes of the string $|s1|$, and matrix's first column should be filled with the indexes of the string $|s2|$. In the third rule, the idea is to look at the surrounding three cells in order to make a decision. If the characters that are being compared in $|s1|$ and $|s2|$, is the same, then take the cell on the left corner. On the other hand, if the characters are not the same in the strings, look at the surrounding three cells and take the cell which has the lowest value inside and add 1. For instance, suppose two strings are given; 'iphone 6' ($|s1|$) and 'one m9' $|s2|$ and the goal is to calculate the similarity between these strings using Levenshtein. In this setting, edit-distance will try to calculate the minimum number of operations to convert 'iphone 6' to 'one m9'. First of all, a matrix M is created with dimensions $9x7$. This matrix then filled with indexes of both strings as proposed by the first two rules of dynamic programming, as presented in the first table in 3.1. After initializing the first column and row of the matrix, the actual filling start according to the rule number 3. As an example, the cell $M_{2,2}$ represents the minimum number of operations to convert ' i' to ' o'. Since character 'i' is different from character 'o', cells $M_{1,2}$,$M_{2,1}$, $M_{1,1}$ are compared and 1 is added to the smallest value among these cells. In this case, the smallest value is 0 which is stored in cell $M_{1,1}$ and once 1 is added , the value for cell $M_{2,2}$ becomes 1. The found value for cell $M_{2,2}$ makes sense because ' i' can be converted to ' o' by replacing character i with character o. Furthermore, the second table in 3.1, demonstrates the scenario where the character strings are the same. In this case, the cell $M_{5,2}$ takes the value of its surrounding left corner cell, $M_{4,1}$. At the end of this process, the cell $M_{(|s1|+1)(|s2|+1)}$, which in this case $M_{9,7}$, gives the number of operations to convert one string to another. In this setting, the edit-distance has been found as 5, and this is actually true because 'iphone 6' can be transferred to 'one m9' by deleting characters i,p,h (3 operations), adding m (1 operation) and replacing 6 with 9 (1 operation).

	" "	o	n	e	" "	m	9
" "	0	1	2	3	4	5	6
i	1						
p	2						
h	3						
o	4						
n	5						
e	6						
" "	7						
6	8						

	" "	o	n	e	" "	m	9
" "	0	1	2	3	4	5	6
i	1	1	2	3	4	5	6
p	2	2	2	3	4	5	6
h	3	3	3	3	4	5	6
o	4						
n	5						
e	6						
" "	7						
6	8						

	" "	o	n	e	" "	m	9
" "	0	1	2	3	4	5	6
i	1	1	2	3	4	5	6
p	2	2	2	3	4	5	6
h	3	3	3	3	4	5	6
o	4	3	4	4	4	5	6
n	5	4	3	4	5	5	6
e	6	5	4	3	4	5	6
" "	7	6	5	4	3	4	5
6	8	7	6	5	5	4	5

Table 3.1: Dynamic Programming Demonstration

The found edit distance then plugged into the Levenshtein similarity formula 3.4 to calculate the actual similarity value [14]. The formula yields, 0.375 as the Levenshtein similarity for the given strings.

$$simLevenshtein(s1, s2) = 1.0 - \frac{edit - distance(s1, s2)}{max(|s1|, |s2|)} \tag{3.4}$$

Levenshtein similarity metric, has a major disadvantage. It involves quadratic complexity calculations, since each character in both strings are compared to each other. This raises problems in storing and computational power. Regardless, of these disadvantages it is one of the most widely similarity measures [14].

3.1.2 Jaro-Winkler

Jaro-Winkler is a similarity measure found by Matthew Jaro and William E. Winkler who work in US Census Bureau [68]. It is a combination of two metrics Jaro and Winkler, which are specifically designed for taking string variations in names and last names into account. It is particularly useful in cases where errors in strings such as typo mistakes exist due to human factor. Jaro distance counts the number of same characters between two strings within a certain window while considering character transpositions and Winkler algorithm is used to increase the similarity between two strings if their beginnings are the same [14].

In Jaro, the window size is determined as half the length of longer string minus 1 and the transpositions are defined as the swapping of two adjacent characters, such as 'ab' and 'ba'. The formula for Jaro can be found in Equation 3.5 [68]. In the formula, letters denoted with W stands for weights associated with characters in first string, characters in second string and characters in transpositions respectively, and d, r, c and τ stands for length of the first string, length of the second string, number of characters in common and number of transposed characters. In the original paper written by Winkler, the weights W are arbitrarily set to $\frac{1}{3}$. Equation 3.6 represents formula for Winkler algorithm, where i denotes the number of agreeing characters at the beginning of two strings [68]. In simplest way, the Winkler algorithm modifies the Jaro formula by considering the first few characters in the string.

$$\Phi = W_1.\frac{c}{d} + W_2.\frac{c}{r} + W_t.\frac{c - \tau}{c} \tag{3.5}$$

$$\Phi_n = \Phi + i.0, 1.(1 - \Phi) \tag{3.6}$$

To give a concrete example, lets use the same two strings that were introduced in previous section, 'iphone 6' and 'one m9'. In this setting, d and r, which represent the length of string 1 and 2, becomes 8 and 6. Window size is the maximum length of strings divided by two minus 1 and in this case equal to 3. This means

that, one will look at 3 indexes before and after of the index under comparison to find common characters. For instance, character 'o' in string 1, will look at characters ['o', 'n', 'e', '', 'm', '9'] in string 2 to find a match and as a result, it will match with 'o'. After several iteration and comparisons, the same characters in both strings are found as ['o', 'n' , 'e', '']. With this results, c, which represent the number of common characters in both strings, is equal to 4. There is no need for transformations with the found common characters, therefore τ is equal to 0. Plugging the values into Jaro formula in 3.5, yields 0.72 as the Jaro similarity. Further, even though both strings share common elements, i in Winkler algorithm is 0, since the common elements do not follow the same order. Therefore second part of Equation 3.6 becomes 0 and the similarity stays as 0,72.

Compared to the Levenshtein similarity metric introduced at the above section, it requires less computational power and it is less complicated since it doesn't involve dynamic programming. It also works well with strings with slightly spelling variations. However, this method has also some drawbacks. For instance, Jaro-Winkler requires characters to be in certain distance from each other and therefore doesn't cope well with prefixes, like comparing 'Professor John Doe' and 'John Doe' [44].

3.1.3 Jaccard

Jaccard is a token-based similarity metric where the strings are split into short sub-strings defined by the tokenization strategy and similarities are calculated by counting how many sub-strings occur both in the strings [14]. This tokenization strategy may involve separating a string into tokens based on white spaces or dividing the string into sub-strings of length q characters. In order to divide a string into q-grams, sliding window approach has been used. With this method, starting from the first position of the string, q length character has been chosen as the first q-gram, then the second iteration starts from the second position of the string, and the second q-gram is chosen with q length character. The process continues until the end of string has been reached.

To demonstrate this similarity metric, lets take the same example that have been used so far, and try to calculate the similarity between 'iphone 6' and 'one m9'. The algorithm starts with removing any kind of spaces in the string, as a result, the strings become 'iphone6' and 'onem9'. Secondly, the algorithm divides the string into sub-string depends on the tokenization function, in this setting, the tokenization is dividing the strings into tokens of length 3. Subsequently, the strings become ['iph', 'pho', 'hon', 'one', 'ne6'] and ['one','nem','em9']. In the third step, the number of sub-strings that occur in both strings are counted, which in this case it is only 1 with ['one'] and the total number of unique sub-strings that occur

in both strings are found, which in this setting 7 with ['iph', 'pho', 'hon', 'one', 'ne6', 'nem', 'em9']. Plugging these values to the formula 3.7, yields, 0,14 [44].

$$simJaccard(s1, s2) = \frac{|tokenize(s1) \cap tokenize(s2)|}{|tokenize(s1) \cup tokenize(s2)|} \tag{3.7}$$

So far for the given example, the calculated similarity metrics include 0.375 with Levenshtein, 0.72 with Jaro-Winkler and 0.14 with Jaccard. Among them, Jaccard is the lowest, since it penalizes the string variations more. In terms of computational complexity and memory, Jaccard is comparable with Jaro-Winkler, but it requires less power and memory in comparison to Levenshtein. This makes Jaccard more efficient for longer string comparisons [14].

3.2 Feature Selection

Not all calculated similarities for each attribute in the feature vector is relevant for learning algorithms. Having irrelevant attributes in the dataset often degrades the performance of the machine learning algorithms in terms of accuracy and speed [71]. It effects accuracy, since the models are forced to learn including the irrelevant attributes and therefore might end up learning things that are wrong, and this in return might reduce the accuracy performance of the model. Secondly, it effects speed, because higher number of attributes means search in higher dimensional space and this might result in slower convergence. Some machine learning algorithms like decision trees and random forest classifiers, are capable of selecting variables as they can decide which attribute to split on. Therefore, they are somewhat prone to the negative effects of feature selection. However, for logistic regression and SVM, there is no feature selection and therefore they can get affected by the irrelevant attributes. For more details about each learning algorithm, please refer to the following section, 3.3. Learning Algorithms.

One way of over coming this problem is selecting features either manually by using domain knowledge or automatically using advanced statistical methods. With feature selection, a more compact and easily interpretable dataset is formed, where users and the algorithms can focus on the most relevant attributes [71]. Additionally, by using feature selection, the best similarity function that have the highest discriminative power in distinguishing between matches and non matches, can be analyzed [14]. Since, choosing features manually is an arbitrary approach and often the feature that is assumed to be important, contradicts with the reality, in the thesis more advanced statistical approaches has been preferred. There are two general approaches to feature selection using advanced statistical approaches; filter

and wrapper methods. Filter methods, measure the relevance of features according to their correlation with the dependent variable, in this case based on the calculated similarities with the dependent variable, whereas wrapper methods choose variables by training a machine learning model on the data, prior to employing a real machine learning algorithm [71]. Filter methods are easier to apply since they do not require training a model but might cause overfitting. Since feature selection will be done in each iteration in the active learning algorithm, a feature selection method that is fast and less complex is needed, and therefore, it was decided to use wrapper methods. Among the wrapper methods, univariate feature selection with ANOVA F-test has been chosen.

Univariate feature selection selects the best features based on univeriate statistical tests, in this case by using a ANOVA F-test. ANOVA F-test estimates the degree of linear dependency between two random variables, by first grouping the numerical feature by the target class and then comparing the means of each group in terms of statistical significance. The formula of F-test can be found in equation in 3.8 [27]. In the formula N_j represents number of rows in predictor X that belong to the class Y=j (1 or 0), $\bar{x}$ is the sample mean of predictor X for target class Y=j (1 0), ${s_j}^2$ is the sample variance of predictor X for target class Y=j (1 0), $\bar{\bar{x}}$ is the grand mean of predictor X and F is the calculated F statistics.

$$
\begin{aligned}
{s_j}^2 &= \sum_{i=1}^{N_j} (x_{ij} - \bar{x}_j)^2 / (N_j - 1) \\
\bar{\bar{x}} &= \sum_{j=1}^{J} N_j . \bar{x}_j / (N) \\
F &= \frac{\sum_{j=1}^{J} N_j . (\bar{x}_j - \bar{\bar{x}})^2 / (J-1)}{\sum_{j=1}^{J} (N_j - 1) . s_j^2 / (N-1)}
\end{aligned}
\tag{3.8}
$$

After calculating the F statistics for each variable, the variables with k number of best values in terms of F statistics are chosen from the dataset.

3.3 Learning Algorithms

After calculating the similarities in the dataset using the above described similarity functions, a feature vector is formed. This feature vector can be used to learn whether the candidate record pairs refer to the same entity or not. In this setting,

learning is called classification, and one can think this this as classifying record pairs as matches and non matches. There are two main approaches to classification depending on the existence of the target variable; unsupervised and supervised. Unsupervised classification algorithms don't require a target variable and includes for instance clustering. With clustering, data can be grouped into k different clusters without needing a feature vector of calculated similarities, where all instances in each cluster assumed to refer to a single real-world entity [21]. On the other hand, supervised approaches need to have information about true matches and non matches, and therefore requires calculated feature vectors to have an assigned value to it that represents its matching status.

In supervised learning, one important concept is dividing the dataset into train and test, and then training the model on the train and evaluating the model on the test. The idea behind this approach is that, since the model is learnt on the train dataset, evaluating the model on the same dataset that is trained on, would not be likely a good indicator of models actual performance [71]. By evaluating the models' performance on testing dataset, an idea of how the model will perform on the unseen dataset can be analyzed and also datasets performance on both train and test can be compared, in order to understand whether the model fits too good to the train dataset or not. In machine learning terminology, fitting too good to the train dataset also known as 'overfitting'. Overfitting occurs when the models' classification accuracy is high on training data, but low in test data [14]. This suggests that the model adapted itself too much to the training data and its no longer generalizable to the unseen data. This causes problems, since the ultimate goal is to predict class labels in the future data.

In the thesis, first dataset has been splitted to train and test and a method of supervised machine learning is used called active learning, which is an algorithm that proposes instances to be labeled by the oracle in order to overcome the problem of creating a target variable. So in a simplest way, one can think of active learning as a method for choosing instances to create a training dataset to be used in classification where classification algorithms are used inside to predict the class and measure the accuracy. Active learning has been described deeply in the following section and here classification algorithms that are most commonly used inside the active learning are discussed.

3.3.1 Logistic Regression

Logistic Regression can be thought as an adaptation of linear regression to predict binary outputs. Logistic regression model puts the linear regression equation into a logistic regression function to squeeze the output to a value between 0 and 1 in order to predict binary outcome. If the squeezed value is greater than a threshold,

normally the threshold value is set to 0.5, then it is assigned as 1, else with 0. Logistic function is a s-shaped curve that can can take any number and map it to a value between 0 and 1 [10]. Logistic Regression is defined in equation 3.9, where z represents the weighted sum of model coefficients and values. In this problem setting, z is the weighted sum of similarity calculation of each attribute in a pair and follows $z = \sum_{i=1}^{n} a_i.sim_i(x, y)$ [21].

$$P(y_i = 1) = \frac{1}{1 + exp(-z)} \tag{3.9}$$

The a_i represents the model coefficients, in other terms weights assigned to each attribute in similarity calculation. The learning part of logistic regression involves estimating these coefficients from the training data. This is done using maximum likelihood estimation which is a function that maximizes the likelihood that the model produced the data that is being actually observed. In this case, the best coefficients for the model would be the ones that predict a value close to 1 for actual matches, and a value close to 0 for the non matches [10].

3.3.2 Support Vector Machine

Support Vector Machine (SVM) is based on the idea of separating classes using a hyperplane in multi-dimensional vector space. If the number of attributes in the dataset is equal to two, hyper planes become lines. SVM starts by mapping the training dataset which contains the feature vectors and class labels that represents the match status of pairs, into a high-dimensional space using a kernel function [14]. After mapping, there are infinite number of possible ways to fit a hyperplane into the created vector space. The objective here is to find and choose a hyperplane which separates the classes the best, while also maximizing the gap between classes. By maximizing the gap, in a way the probability of future data points being correctly classified has been increased.

In figure 3.1, SVM has been shown with a simple matching example. Imagine that two tables are given a and b with two records and two attributes in each. The records are annotated as a_1, a_2, b_1 and b_2 and the attributes include surname and suburb. The goal is to make a matching using SVM classifier. In this case, there will be four pairs to be compared, as every record in dataset a, will be compared with every record in dataset b. Then, for each pair, a feature vector will be created using one of the above similarity metrics and a matching status will be attached to each vector. After preparation of the dataset as described, SVM classifier will be trained. The figure 3.1, demonstrates the created dataset in 2D vector space with three different trained SVM classifiers and their matching status. Each of these

classifiers are denoted with dotted lines and each represent one possible decision boundary. Among them, the one with the thickest dotted line stands out as it is the SVM with the widest margins to matches, denoted with circles and non-matches with squares [14]. Therefore, it is the best SVM classifier in this setting.

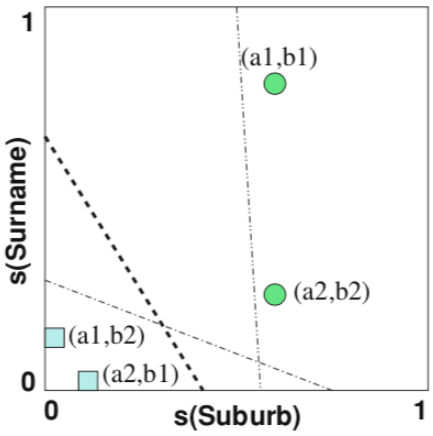

Figure 3.1: Support vector machine demonstration using two datasets a and b with four feature vectors [14]

3.3.3 Decision Trees

Decision Trees is one of the most widely known and used algorithms in the machine learning area. It is called decision tree since it resembles an actual tree with branches and leaves. Its resemblance to a tree like structure makes it highly interpretable as the decisions can be visualized easily in a hierarchical structure and this distinguishes the algorithm from others. The idea behind decision trees is to recursively partition the data into smaller subsets in a top down matter. It starts with all the training set including their attributes and class labels, and the objective is to find the attribute that gives the best split in terms of purity. In simplest terms, the attribute with best split would be the one that makes the nodes after split as pure as possible in terms of classes. For instance, if you have ten values, which 6 of them labeled with class 1 and rest of them with class 0, the best split would be with the attribute that can classify these into two groups where 6 of them with labeled class 1 is in one node and 4 of them with class 0 in the other node. After finding the attribute with best split, the data is splitted into subsets according to attributes' values. The algorithm continues to find best splitting features for each created group after splitting, until a threshold is reached or all the created groups are pure. The whole process involves deciding which features to choose based on

the best split and knowing where to stop.

For an example, imagine two datasets a and b with four feature vectors as described in SVM. Suppose in this case, three variables are available; suburb, birthday and surname, which contains the similarity calculation for each pair for that specific attribute and the goal is to construct the best decision tree using these attributes. Figure 3.2 demonstrates the example by giving two decision trees which both of them splits the data into classes using different attributes. In the left tree, there is only one split and the splitting rule is, if suburb similarity metric is above 0.6, then its a match and if its below 0.6, then it isn't a match. On the other hand, the tree on the right splits the data using two variables birthday and surname. The splitting rules follows, a pair is a match, if birthday is above 0.5 and surname is above 0.15, on the other hand a pair is not a match if birthday is below 0.5 or birthday is above 0.5 and surname is below 0.15. According to the given example, the tree on the left is better, since it reaches purity faster with less computation [14].

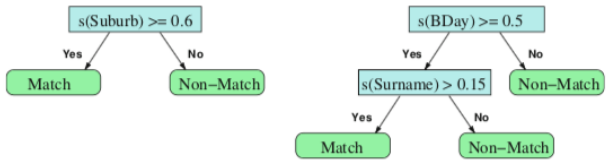

Figure 3.2: Decision Trees demonstration using two datasets a and b with four feature vectors [14]

3.3.4 Random Forest Classification

Random Forest Classification belongs to a family of algorithms known as ensemble methods, which combine multiple machine learning algorithms to make a better classification. It is supervised and it uses decision trees as the base machine learning algorithm. The idea is that, by combining multiple decision trees which are not all optimal, a better classification can be made in terms of accuracy and stability. Decision trees that are used in random forest as the base algorithm differs from the actual decision trees in terms of two aspects; random sampling and randomly selected features. Firstly, each decision tree is trained on different samples from the original data, using a method called bootstrap aggregation also known as bagging. In bootstrapping, sub-datasets are created as the same size as the original dataset with sampling with replacement. Bootstrapping makes the algorithm prone to overfitting as the noise point will only be available in few sub-datasets and therefore only few models will overfit the data. Secondly, for each decision tree a randomly selected features are considered while making splitting decisions, rather

then considering all the variables like in decision trees. By using randomly selected features, one is making sure that strong attributes are appearing in some decision trees and therefore will be picked in some decision trees. This makes the created decision trees less correlated with each other and therefore makes the overall model more robust. After training multiple decision trees on same sized sub-datasets with replacement using random subset of features, all the decision trees gives a classification, in other terms casts a 'vote' for the class and the algorithm chooses the class having the most votes over all trees in the random forest [9].

3.4 Active Learning

Active Learning is a supervised machine learning algorithm that aims to overcome the issue of preparation of the training dataset, which often requires costly manual labeling. It achieves this by allowing the learning algorithm to choose the data it wants to learn from and as a result the learning algorithm is trained on more informative data points. In this way, the algorithm achieves high classification accuracy by using as less labeled data as possible, therefore minimizes the effort obtaining labeled data [59]. Active learning should be used in problems where data is large but the labels are costly to obtain, like in the case of data matching where we have two databases and the goal is to find correspondences between these databases that refer to the same entity.

The algorithm first starts by fitting a classification model to a small set of seed training examples and classifies all examples in the train set to matches and non matches. This small set of seed examples are usually the size of 10. The created initial model will have low accuracy since the training data only contains small set of pairs. In order to improve the model, in the following step, the algorithm asks user to provide labels to the pairs according to the specified query strategy. This query strategy can choose data points that it is most unsure of for instance, data points that lies in the decision boundary, which in the case of uncertainty sampling or can choose data points where multiple trained models disagree the most, which is in the case of Query-By-Committee sampling. Then at the last step, it adds these manually classified new pairs to the training set and retrains the algorithm. This iteration continues until a certain criteria is met. The stopping criteria in this case could be, reaching a predefined maximum number of iterations or achieving a certain matching quality on testing set defined by the last trained classifier [14].

The above described algorithm is one of the scenarios for active learning known as pool-based sampling and the illustration of this process can be found in the figure 3.3. There are three general active learning scenarios; query synthesis, stream-based selective sampling and pool based sampling. Among these sampling sce-

narios, pool based sampling stands out as it is the most common scenario used for applied research in active learning where the other scenarios are less common and exists more in the theoretical literature [60]. Therefore in this section most widely used queries based on pool based active learning scenario have been discussed.

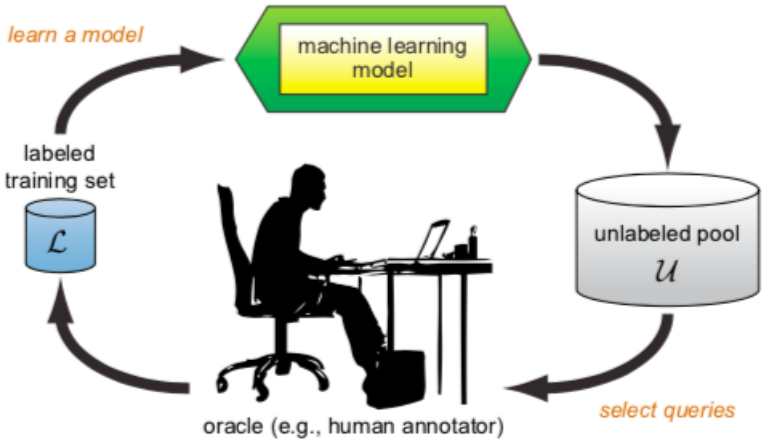

Figure 3.3: Active Learning in pool based scenario [59]

3.4.1 Libact Library

Along the thesis, for active learning applications, libact library in python that is created by Yao-Yuan Yang, Shao-Chuan Lee, Yu-An Chung, Tung-En Wu, Si-An Chen, and Hsuan-Tien Lin, is used. The package follows scikit-learn design framework standards and it employs pool based active learning scenario. The goal of the package is to make active learning easier for general users [72]. In the package, there are four classes; dataset, query strategy, model and labeler. The dataset class holds the labeled and unlabeled instances, and after each iteration of labeling, it needs to be updated. Instances to be queried are determined in the QueryStrategy class, with make query function. Model class is where the prediction occurs and it is designed to predict on the labeled training dataset. The class itself have adaptations of logistic regression and SVM, but other machine learning models from scikit-learn library can be used via SklearnAdaptor function. The last class is the labeler class which returns the label of the queried instance. It has two internal classes; ideal labeler and interactive labeler. Ideal labeler returns the label of the queried instance from the full training dataset, assuming that a full training dataset is available, whereas interactive labeler, asks the queried instance to the oracle and

gets the label from the entered value. Since the full training dataset is already available, ideal labeler is used in the thesis.

3.4.2 Seed Generation

Active learning starts with a small number of labeled examples, usually picked randomly, known as *seeds*. It is highly important for active learning algorithms to select appropriate seeds for modeling, since these examples determine the direction the active learning takes while exploring the version space. If a small and biased seeds are chosen for active learning, a model that is not consistent with the true underlying distribution of data maybe created since some areas of the version space may not be reached and learnt from [58]. Here, the seed generation problem have been explained in more detail, from a technical point of view.

Imagine that you are given three different classes; triangles, circles and squares as illustrated in image 3.4, with the goal of learning a successfull active learning model. In this case, if for instance, only one example from triangles and one example from circles are chosen (shown with shaded shapes) for the initial seeding, then the active learning model may end up selecting examples from only these classes by completely ignoring the class of squares and may learn a decision boundary shown with vertical line in figure a in image 3.4. This is mainly due to the fact that seed examples influence the direction the learner takes during exploring the instance space and without proper initial seed assignment, the learner may become overconfident about the class membership which in return can cause the learner to spend a lot of time exploring regions around decision boundaries and not learning areas away from boundaries [20]. This means that the classifier is not aware of the square class at all, therefore the class has been completely ignored and not learnt. This problem known in the literature as *missed class effect*.

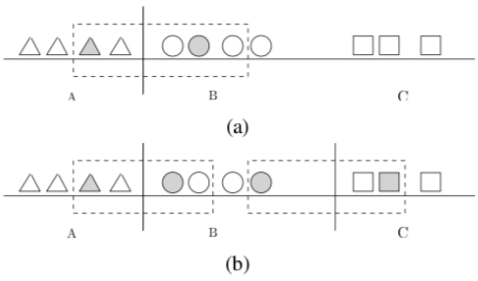

Figure 3.4: Illustration of missed class effect, where the dashed squared regions show the explored decision boundaries and vertical lines represent the final decision boundaries [65].

Missed class effect has been influenced by two factors; similarity of considered classes and class imbalance. First of all, if there are two classes that are harder to be distinguished then others in a multi class problem and if one of these classes is underrepresented in the initial seed set, the active learner is more likely to choose instances from the class that is represented in the seed set and won't be able to distinguish among them until it has acquired enough information to discriminate between them. Secondly, currently in the literature many active learning algorithms choose initial starting seeds randomly. If the the data is proportionally distributed, it is highly likely that all the classes will be represented within the initial seeding. However, when the data is highly skewed and active learning starts with a randomly drawn seed, rare classes might not be represented in the seed set and this in return increases the chance of active learning missing some classes completely [65, 20]. Considering the fact the data matching involves imbalanced classes, where the majority of the dataset consists of non matches, missed class effect is something to be taken into consideration. Because if there isn't any pair with non match status in the initial seeding, the match status might never be learnt.

Libact package, that is being used in the thesis for active learning, forces users to include examples from every class to the initial seed before active learning starts to avoid the missed class effect. This indeed can solve the issue, as demonstrated in figure b in image 3.4, where by adding an example from squares class to the initial seeding, a second decision boundary between circles and squares is learnt. But in reality, we don't know the class labels of the underlying dataset, therefore random selection becomes a trivial process of labeling randomly sampled instances, hoping that a pair with a match class, will come early in the process so that the real active learning can start.

3.4.3 Query Strategies

One of the main challenges of active learning is determining an appropriate query strategy. The query strategy determines how the pairs are selected from the unknown matches for additional manual labeling and therefore have a direct effect on the training dataset, the model and its accuracy. Here four widely used querying strategies have been explained further from a technical perspective.

Random Sampling

Random sampling is a naive way of sampling where the instances to be labeled are drawn randomly from the dataset instead of using an advanced algorithm. Its assumed that sophisticated query selection algorithms where instances are selected according to a logic, will reach the same F1 measure of the trained fully labeled

dataset, faster then the random sampling. Therefore, random sampling in the thesis is treated as a base method rather then an actual sampling query strategy.

Uncertainty Sampling

Uncertainty sampling is a querying strategy based on confidence of the algorithm. The intuition behind the uncertainty sampling is that the instances that the algorithm less certain of, would offer the most information [60]. In probabilistic classifiers such as logistic regression, uncertainty sampling queries instances where the posterior probability of being positive is nearest to 0.5, assuming that we have a binary classification problem. On the other hand, in deterministic classifiers such as support vector machines, uncertainty sampling queries instances that are closest to the decision boundary.

In order to illustrate uncertainty sampling further, one can look at the following example and figure 3.5, taken from the Active Learning book written by Burr Settles [60]. Settles explains uncertainty sampling using a dataset that consists of 400 points sampled from two Gaussian distributions, where 200 points assume to belong class 1 and other 200 points assume to belong class 2. Figure 3.2.a. represents all the points in 2d space, figure 3.2.b. represents logistic regression trained on randomly selected 30 points and figure 3.2.c. represents logistic regression trained on 30 points selected by uncertainty sampling. The blue lines represent decision boundaries. One can see that, in figure 3.2.b, the selected 30 instances are random and therefore are far from the decision boundary and the built model's accuracy is %70. On the contrary, in figure 3.2.c., the selected 30 instances are closer to the decision boundary and its accuracy is %90 consequently.

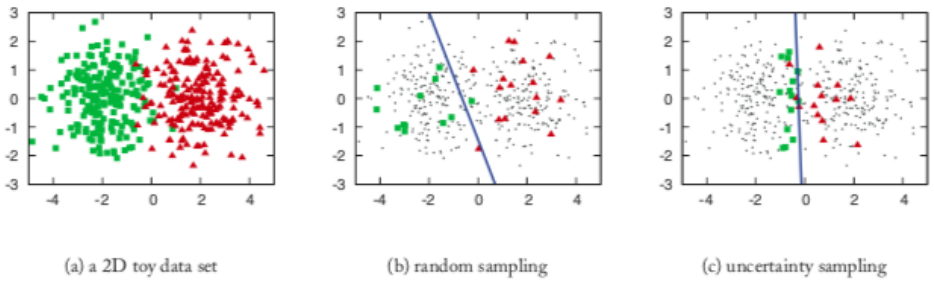

Figure 3.5: A visual representation of Uncertainty Sampling [60]

Furthermore, there are several ways to measure this uncertainty. To start with, least confidence (LC) strategy chooses instances to be labeled by the oracle which

their most likely labeling is actually the least likely. In more simpler terms, classifiers assign a probability to each of the predicted instances to represent how confident they are about this prediction. LC looks at these assign probabilities and chooses the instance which have the least score, a.k.a. the least confident about, and send this to oracle to be labeled. The second strategy is margin sampling (M) where it looks at the difference between two most likely class labels for an instance and tries to find the instance whose difference is the least. The idea behind this is that instances who have large margins between its most likely two class labels will have less doubt compare with the instances who have small margins between two most likely class labels. The third strategy, involves using entropy (H) which is a measure used in machine learning to understand the degree of impurity or uncertainty. This querying strategy chooses instances to be labeled by the oracle whose entropy is the highest.

$$x^*_{LC} = \arg\max 1 - P(\hat{y}|x) \tag{3.10}$$

$$x^*_{M} = \arg\min[P(\hat{y}_1|x) - P(\hat{y}_2|x)] \tag{3.11}$$

$$x^*_{H} = \arg\max -\sum p(x)\log p(x) \tag{3.12}$$

Query by Committee

Query by committee (QBC), is a querying approach that uses multiple learning algorithms to choose an instance to be labeled by the oracle. The idea behind QBC is creating a committee of models which are trained on the same labeled dataset and then making each member of committee vote on the instance. The instance which they disagree the most is chosen since it is considered to be the most informative instance.

This approach based on the idea of version space minimization. In order to explain this concept thoroughly, further explanation is necessary in terminology, starting from hypothesis space and version space. Hypothesis space can be defined as all the hypotheses that returns by the learning algorithm, whereas version space can be defined as the subset of hypotheses from hypotheses space that are consistent with the training examples, where consistent refers to hypotheses that can explain the training data well [54, 43]. A visual example of version space for binary classification can be seen in figure 3.6, where (a) and (b) represents linear and axis parallel box classifiers respectively. As can be understood from the figure, all hypotheses in version space are consistent with the training dataset, but demonstrates competing models. In this setting, the goal of learning algorithm can be thought as trying to pick one of the hypotheses from version space, a.k.a lines or boxes as shown in the figure, that describes the dataset at hand the best. As the number

of collected data instances increases, the area of the version space becomes more constrained and therefore search can be more precise [60]. Then the goal of active learning in this setting becomes finding instances which reduce the version space as much as possible. This could be achieved with QBC since by using multiple learning algorithms, controversial regions of the version space can be reached.

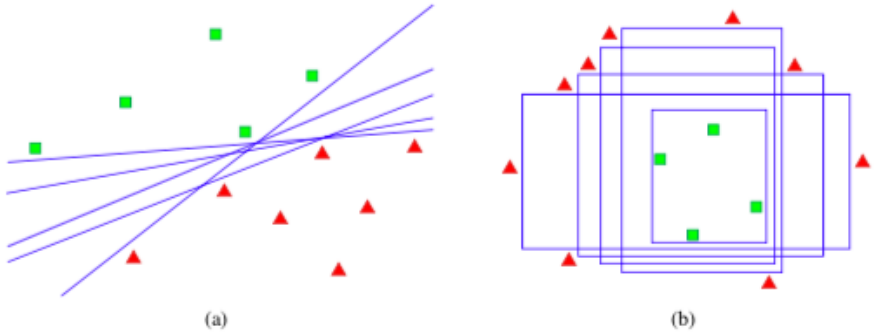

Figure 3.6: A visual representation of Version Space. The left hand image demonstrates linear classifiers and right hand image demonstrates parallel box classifiers [59]

In order for QBC to be successful, two things need to be achieved, one must be able to form a committee of models which each represent different regions of the version space and query algorithm must be able to successfully measure the disagreement among committee members [59]. To start with, how the committee should be formed? According to Settles, there is no general rule of thumb to the number of committee members even though five to fifteen hypotheses are quite common, small committee sizes of two or three appeared to be working well as well [59]. However, one needs to think about the computational cost while deciding on the committee size as the number of members increase, the more costly the querying would be since all members have to be trained on the labeled data and predict on the unlabeled data, prior to making query decision. Moreover, if these chosen members are of the same model class, different parameter settings needs to be used in order to find the regions of uncertainty. As for measuring the disagreement, there are two commonly used methods in the literature; vote entropy (VE) and Kullback-Leibler (KL) divergence. Vote entropy based on the idea of using the predicted classes from the committee algorithms to choose the instance to be queried. For instance, if we have a committee size of two, then the instance which is predicted with different classes from both of the algorithms will be chosen. In here, $V(y_i)$ is the number of votes from committee members predictions for that

particular class and C represents the committee size [59]. Furthermore, Kullback-Leibler is a measure used to calculate the differences between two probability distributions. With using KL, disagreement is quantified as the average divergence of each committee members' prediction from the probability of y_i being the correct label, also known as consensus C [60]. Therefore, it favours the instance, where difference between the label distribution and the consensus is the largest. At below you may find formulas for each of described methods [59].

$$x^*_{VE} = \arg\max - \sum \frac{V(y_i)}{C} \log \frac{V(y_i)}{C} \tag{3.13}$$

$$x^*_{KL} = \arg\max \frac{1}{C} \sum D(P_\theta || P_C) where D(P_\theta || P_C) = \sum P_\theta(y_i|x) \log \frac{P_\theta(y_i|x)}{P_C((y_i|x))} \tag{3.14}$$

Density Weighted Sampling

Above described query strategies, Uncertainty Sampling and Query by Committee, decide on the instance by measuring the content of a single data instance, without considering the whole dataset and this may lead to querying instances that are outliers or of poor choices [60]. For instance, Figure 3.7 demonstrates the problem of using binary linear classification with uncertainty sampling. In the figure, unlabeled data points are shown with circles and shapes represents the known data instances. In such a scenario, uncertainty sampling will choose the instance labeled as A, since it lies in the decision boundary and therefore according to its opinion its the most informative instance. But once one considers the whole data distribution at hand, it can be clearly seen that A is actually an outlier and point B will probably provide more information in this case. One way to overcome this problem is to use error and variance reduction approaches which aim to reduce the expected future error over all data instances, however these methods are highly computationally expensive and it might not worth it in the end [60]. This phenomenon is also experienced in the early phases of the thesis with variance reduction query strategy using logistic regression as the learning algorithm on phones dataset. The query strategy wasn't able to query a single data instance in two hours period of time that it was being experimented.

Figure 3.7: Problem of using binary linear classifier with uncertainty sampling [59]

Density weighted sampling overcomes this problem by querying instances that have high information content as well as that are representative of the underling distribution. It argues that by incorporating these two measures together, better instances can be selected [22]. For quantifying the utility $\phi_A(x)$, it uses a base query strategy A like uncertainty sampling or QBC and in order to measure the representatives of the instance it looks at the unlabeled data pool U, and calculates the average similarity between that specific instance and all other instances in the unlabeled data pool. This calculated average similarity is penalized further by β, which gives weight to the calculated density term. For similarity measure calculation Cosine similarity, Euclidian distance or Pearson Correlation can be used. At below you may find the base formula of density weighted sampling [59].

$$x^*_{ID} = \arg\max \phi_A(x) x(\frac{1}{U} \sum sim(x, x^{(u)}))^\beta \tag{3.15}$$

This initial idea of density sampling have been further investigated in the literature and enhanced using clusters. For instance, in order to reduce the number of similarity calculations, one can first cluster the pool of unlabeled instances and compute the average similarity within the cluster the instance that is being compared belongs to [60]. Clusters can also be used to model the underlying data distribution. For instance, it has been proposed that by clustering the instances prior to querying the label and classification and later incorporating the assigned cluster information into the probability function, a data distribution can be learnt [47]. This proposed method is known in the literature as Density Weighted Uncertainty Sampling (DWUS) and works with Uncertainty querying strategy. This yields two major advantages to active learning querying process. First of all, the instances which are in the center of clusters are more informative then the ones lies in the border of the clusters, therefore these points would likely to be chosen first by the querying algorithm. Secondly, it has been assumed that instances who belong to the same cluster are likely to have the same label [47]. These information can be used to fasten the process of querying and label assignment. Moreover,

DWUS has a tendency to choose points that have moderate density but low uncertainty and as a result wastes effort by concentrating on instances that don't improve the algorithm greatly [22]. In order to overcome this problem, DUAL algorithm has been proposed which dynamically re weights the density and uncertainty components of the DWUS algorithm. DWUS combined with DUAL algorithm is also incorporated into the libact package [72].

3.5 Summary

In this chapter, three most commonly used similarity metrics of Levenshtein, Jaro-Winkler and Jaccard have been explained comprehensively by using a real example, a general overview to feature selection techniques have been given while describing ANOVA F-test in detail, an introduction to machine learning has been made been made with a focus on four common supervised machine learning algorithms that are used in data matching including logistic regression, support vector machine, decision trees and random forest classification and pool based active learning algorithm has been explained in detail while introducing the used python libact package, analyzing the effect of seed selection on active learning from a theoretical perspective and defining the most common querying strategies.

Chapter 4

Dataset

4.1 Creating the Dataset

Existing datasets that are used for product matching in the literature have several short comings like having small number of e-shops with small number of generic product attributes in the gold standard, which isn't enough to cover the heterogeneity of the web and cannot be used to evaluate the validity of extracted product attributes from product descriptions [50]. In order to overcome these issues, in 2016, Data and Web Science Group from University of Mannheim introduced 'Web Data Commons for product matching and feature extraction' (WDC) dataset which includes two subdatasets, 'Goldstandard for Product Matching' that consists of more than 75000 matches between product catalog and crawled product pages, and 'Goldstandard for Feature Extraction' that consists of more than 500 deeply annotated web pages. In this thesis, 'WDC Goldstandard for product matching' dataset is used, which can be found in `http://webdatacommons.org/productcorpus/`.

For the construction of the 'WDC Goldstandard', three main product categories have been determined; TV, phones and headphones. In order to create product catalogs for each product category, 50 seed products have been chosen for each product category manually and features have been collected using manufacturers shop or google shop. Then, for each product category, 500 product pages from 31 different websites have been crawled and feature extracted. Finally, the gold standards have been created by manually annotating positive and negative correspondences for each category. In the end of this process, 4 different datasets have been created for each product category:

- Source: Include features from crawled webpages. It is unique by page id.

- Target: Include features from product catalog. It is unique by product catalog id.
- Gold Standard : Include matches between product catalog (source) and product pages (target). It is unique by page id and product catalog name.
- Product Catalog: Include product catalog name and its correspondent product catalog id.

In order to derive a single table for each product category, these four datasets need to be merged. In the thesis, N:1 data matching scenario has been followed, where multiple datasets are matched against one main dataset. This requires these separate tables to be consolidated by matching crawled product pages against gold standard. First, target and product catalog datasets have been left joined, in order to add product catalog ids to product catalog name in the target dataset. Second, gold and source have been inner joined, to add crawled page features to gold standard. Lastly, the final dataset has been created by inner joining the dataset that is created in the first step with the dataset that is created in the second step. The created final dataset includes matches, crawled features from product pages and manually collected features from the product catalog. After merging, it was expected to find 500 crawled product pages and 50 products for each category, however in phone category, 48 unique products and 447 unique product pages have been reached. The same situation have been observed in headphone category with 49 unique products and 444 unique product pages and in TV category with 60 and 428. Further analysis regarding these discrepancies showed that they were caused by three main cases; there were some product pages that weren't in source file but in gold standard, there were some products available in product catalog but not in gold standard and there were some products in gold standard that weren't in product catalog. These cases occurred due to the fact that after creating the corpus and the product catalogs, some pages were removed or added to make the task harder by University of Mannheim and the gold standard wasn't updated accordingly. This doesn't pose a problem for the thesis, since the resulting final datasets for each category are sufficient for the analysis. The resulting final datasets for each category are shown below in the table 4.1 and the whole matching process can be found under Appendix A.

4.2 Evaluation of the Dataset

In this section the final datasets from Phone, Headphone and Tevelevision (TV) categories that have been created in the previous section 4.1, are evaluated more deeply from the aspects of features, sparseness and target value.

	Phones	Headphones	TVs
Final Dataset - Number of Rows	21455	22199	25679
Final Dataset - Unique pages	447	444	428
Final Dataset - Unique products	48	49	60

Table 4.1: The number of rows, number of unique pages and number of unique products for each dataset after merging

4.2.1 Dataset Features

Phones, Headphones and TVs datasets are all consist of three main type of columns: id columns, target column and attribute columns. Id columns are columns that are unique to a table. In this setting, these columns refers to a unique product from product catalog or crawled product pages, and include 'id webpage', 'prodcat id', 'page url', 'page warc' and 'prodcat product name'.

Moreover, target column represents the manually annotated matches between the product catalog and crawled web pages, and it is named as 'match' in the datasets. Target column is the variable that is being predicted in the classifications that are being done in this thesis. The last and most importantly, attribute columns are columns that defines and distinguishes a particular product, from other products in the same category. In order to understand from which file the attributes came from, a prefix 'cat' or 'page' is attached to the column names, where 'cat' and 'page' used as an abbreviation to product catalog and product pages. These attributes can be thought in two categories; category specific variables such as rear camera resolution for phones category, headphones' cup type for headphones category, hdmi ports for TV category and common variables that occur almost every dataset, like gtin number, brand and product type.

Attribute columns are specially important since these columns can be put into the similarity functions to produce feature vectors, which then can be used in the classification algorithm, along with the target variable, to classify records into matches and non matches. In order to produce feature vectors, corresponding attributes from both datasets need to be found. Therefore if an attribute only exists in the product catalog and doesn't have a correspondence in the product pages, that column can not be used in the analysis and vice versa. This case is quite common in general since some attributes may only exist in certain product pages and this is also what is being experienced in the datasets that are described here and used in the thesis. In order to understand this phenomenon better and to see the effect of drop in the attributes columns, one can refer to table 4.2.

Phones Dataset

Phones dataset has 60 columns. These columns consist of 5 unique id columns, 1 target column, and 54 attribute columns. Among the attribute columns, there are 4 columns that only exist in product catalog. Once these 4 columns are eliminated, the rest of the 50 columns that represent the products' features can be used in similarity calculations.

Headphones Dataset

Headphones Dataset has 69 columns. These columns consist of 5 unique id columns, 1 target column, and 63 attribute columns. The columns that don't have a correspondence in the other table, are all from product catalog. After eliminating these columns, the rest of the 52 attribute columns can be used in the similarity functions.

TVs Dataset

TVs Dataset has 5 id columns and 1 target columns which is the same as phone and headphones, however it has 556 columns which is way more compared to other two datasets. Unfortunately, a deeper look into the dataset yields the fact that %73 of these columns are not usable in the analysis since they don't have a correspondence in the other table and therefore needs to be eliminated. After the elimination, the number of attributes dropped drastically from 550 to 134. These 134 attribute columns can be used in the similarity functions.

Dataset Name	Phones	Headphones	TVs
Number of columns	60	69	556
Number of id columns	5	5	5
Number of target columns	1	1	1
Number of attribute columns	54	63	550
Attributes- only exist in product pages	0	0	407
Atrributes- only exist in product catalog	4	11	9
Attributes- Remaining	50	52	134

Table 4.2: Number of features in each dataset after merging

4.2.2 Sparseness of the Dataset

As described previously, the datasets that are being used along the thesis are sparse datasets with many missing entries. This situation causes certain challenges, since in order for similarities between features to be calculated correctly, missing values need to be handled. Furthermore, if a column will be eliminated due to having

too many missing values, then its corresponding column in the other dataset needs to be eliminated too, since for similarity calculations corresponding columns from both datasets are necessary. The techniques for handling missing values have been discussed more deeply under section 5.3, here the sparseness of the dataset will be investigated further. The figures that are used to demonstrate sparseness of each dataset can be found in the appendix B.

Phones Dataset

As expected, there are no missing values in the id columns and target columns. There are 5 columns which are completely missing including 'cat modelnum', 'cat manufacturer', 'cat package height', 'cat product code', 'cat wattage' and 'cat powersupply'. Additionally, there are 6 columns that are more than %90 missing. The columns that are all missing need to be eliminated from the dataset as well as their correspondences in the other dataset, if such a relationship exists. For analysis purposes the columns that are not entirely missing but that are more then %90 missing, are kept in the dataset.

Headphones Dataset

As anticipated, there are no missing values in the id columns and target columns. Also contrary to the phones dataset, there aren't any columns that doesn't have any value, however there are 25 columns which are more then %90 missing. This phenomenon makes the dataset more sparse compare to Phones dataset and this difference can be detected from the nullity matrices shown in the appendix B with figures B.1 and B.2, since headphones dataset have more white spaces compared to phones dataset.

TVs Dataset

As awaited, there are no missing values in the id columns and target columns. In the whole dataset, there isn't any variable that is null, however 460 columns are more than %90 missing, which increases the sparseness. However, most of these columns are not usable due to the fact that they don't have a match with the other dataset. Therefore, the correct statement regarding sparseness would be, among the 134 columns that have a correspondence in the other dataset, 57 of them are more than %90 missing. This makes the TVs dataset comparable with Headphones in terms of sparseness. The sparseness of the TVs dataset is demonstrated via nullity matrices in the appendix B, with figures B.3 and B.4. The first figure represents the sparseness in the whole dataset with 556 columns whereas the second figure B.4

shows the sparseness in the dataset with 134 columns which includes corresponding columns and ids.

4.2.3 Target Variable

In data mining, target variable is the variable that is being predicted. In this setting, the target variable is the column that represents matches between product catalog and product pages. The column, named as 'match' in the datasets and it has two values respectively; 'match' and 'non-match'. The distribution of this variable is shown for each dataset under 4.1. From the graphs it can be clearly seen that, in all of the datasets the number of non-matches dramatically exceeds the number of matches. Considering the problem setting that is being worked with, this is to be expected, since for each product from product catalog, a cross match is done with every product in the product pages and in the end only few product pages really matches with a particular product from product catalog. So, simply put, there are more pairs in the dataset that refer to two different entities then there are pairs where both refer to the same entity [14]. This causes certain challenges in classification, since some learning algorithms tend to only predict majority of classes and treat the instances from other classes as noise. Furthermore, it also has an affect on the selection of evaluation metric for classification, since with imbalanced datasets True Negatives have a tendency to be really high and therefore might dominate the calculation accuracy.

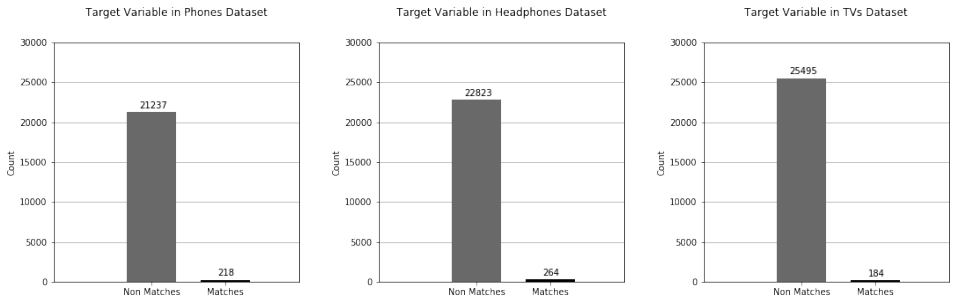

Figure 4.1: Target Variable Distribution

4.3 Summary

In this chapter, three main datasets that are being used in the thesis have been introduced. These datasets have been prepared by Data and Web Science Group in University Mannheim to overcome the short comings of existing datasets that are used for product matching in the literature and consist of in total 75000 product

matchings form three different product categories; phones, headphones and TVs. For each product category, there were four sub datasets, so as a first step these datasets have been consolidated for each product category in order derive a single table that consists of matches and non matches. Then the constructed three datasets have been analyzed in terms of their features, sparseness and target variable. In terms of usable features, TVs dataset stands out with the most features with 134, followed by headphones with 52 and phones with 50. In terms of sparseness, headphones dataset attracts attention as %48 of the attributes are more then %90 missing, followed by TVs with %41 and phones with %10. Lastly, all the datasets have drastically more non matches then matches, which is something to be expected in data matching tasks.

Chapter 5

Methodology

5.1 Data Preparation

After the dataset has been created by following the steps in section 4.1, certain steps need to be taken to make the model suitable for modeling. For instance, finding corresponding columns in both databases, dropping columns that are all missing, dropping variables that are only available in product pages or product category, and dropping ids. All of the steps that are taken to prepare the dataset for modeling can be found below in order.

1. Separate the variables in the dataset into three distinct categories;
   (a) Choose columns that only appear in product pages
   (b) Choose columns that only appear in product catalog
   (c) Choose columns that are matches along with five id columns; 'id webpage', 'prodcat id', 'page url', 'page warc', 'prodcat product name' and one target column;'match'
2. Examine the dataset statistics to find columns that are all missing and drop them accordingly. If missing columns have been found and it has a correspondence in the other dataset, drop their correspondences as well.
3. Drop variables that appear only in product catalog
4. Drop variables that appear only in product pages
5. Drop unnecessary ids; 'page url', 'page warc', 'prodcat product name'
6. Drop columns that are all missing and their correspondences in the other dataset

7. Calculate similarities

   (a) Remove certain characters like commas, parenthesis, hashtags, question marks etc. from both of the strings to be compared

   (b) Make both of the strings small case

   (c) Calculate similarities between two strings based on the defined missing option, described further in the next section

5.2 Similarity Metrics

The closeness of two records is measured by different similarity metrics in different ways. Therefore, in order to capture the closeness of two records, it is a common approach to use multiple similarity metrics on the same dataset to account for different variations [16, 5, 25, 67, 19]. That's why in the thesis three different similarity metrics that evaluates the closeness of two records from different aspects have been used. These metrics include Levenshtein, Jaro-Winkler and Jaccard similarities. This combination of similarity metrics have also been used by Freitas et al. along with TF-IDF in their research about genetic programming and active learning [19].

Levenshtein is a similarity metric that depends on calculating edit distance between two records using dynamic programming, which involves quadratic complexity calculations and requires high computational and storage power. However, it is still a similarity measure that is widely used in data matching and therefore decided to be used in the thesis as well [14]. Another method that is based on edit distance calculations is Jaro-Winkler. Jaro-Winkler requires less computational power and it is less complicated compared to Levenshtein, and it works well with strings that have slight spelling variations. Therefore it is another method that is chosen to be applied to the thesis. While applying Jaro-Winkler to the dataset, the weight W is not modified and kept the same as the original paper. Lastly, Jaccard similarity metric has been decided to be applied, in order to take token-based similarities into consideration. In the literature of data matching, token based methods commonly used by subsetting a string into 2 or 3 character length substrings [14]. Accordingly, in the thesis, Jaccard similarity metric has been calculated by dividing the strings under consideration to 3 character length substrings.

In the thesis, these similarity metrics have been calculated for every attribute that have a correspondence in the other dataset. While experimenting, first, active learning algorithms have been applied on datasets that are prepared with using only Jaccard, only Levenshtein and only Jaro-Winkler similarity metrics, in order to see the success of each algorithm alone. However, the problem with this approach is,

records may not be well distinguished from each other if only one type of similarity metric is being used and as mentioned before, similarity metrics assess the closeness of two records from different aspects, therefore one similarity metric might work well for one column and might not work well for the other column. Thus, after applying active learning algorithms on datasets prepared using only one specific similarity metric, a dataset is created that includes all the similarity metrics and active learning algorithms have been applied on this dataset. This approach is still kind a problematic since not every calculated similarity for each column is relevant, and having irrelevant attributes may degrade the performance of the machine learning algorithms in terms of accuracy and speed [71]. In order to account for this, as a last step feature selection methodologies have been incorporated into active learning algorithms, which is discussed further in .

5.3 Approaches to Sparse Datasets

Datasets that are being used along the thesis are sparase datasets with many missing entries. This causes certain challenges, as the first step after preparing the dataset in data matching, is calculating the similarities between the records using metrics like Jaccard, Jaro-Winkler, TF-IDF to measure the closeness. If one of the records are missing while calculating the similarities, then the similarity algorithm will assign 0 to the comparison. This is often misleading, as two completely different records could also have 0 as their similarity. Therefore, a distinction should be made between records that have 0 as their similarity because they are completely different, and records that have 0 for their similarity because they are missing.

In the thesis four different strategies have been followed to handle missing values. The first two approaches involve removing the missing values and filling them, which are two mainly used popular methods in the literature [66, 67]. The latter two approaches involve altering the created feature vectors to take missing values into account and as far as the author knows, it has not been used by anyone in the literature so far. In the thesis, datasets prepared by missing handling option 3 and missing handling option 4 produced acceptable results and therefore reported further.

In the first approach, columns that have more then %50 missing values are dropped and the missing values in the remaining columns are filled with the most frequent values of the corresponding columns. This approach is not the best technique for handling missing values as it may cause records to be different. For an example, imagine two phone entities with following vectors ['iphone 6','ios','32 gb'], ['','android','32 gb'] and suppose that the most frequent value for first column is 'iphone 6'. With this filling approach, the second vector will become ['iphone 6',

'android', '32 gb'] and consequently feature vectors of similarities can be [1, 0, 1], depending on the chosen similarity metric. So, even though the real match status between these records is 0, since name of the phone is one of the most important features in matching, this pair can be predicted as a match. Furthermore, with this approach, entities become indistinguishable from each other since less columns are used and this may lead again to wrong predictions.

The second approach to missing value handling involves assigning 0 to similarities if either of the field is missing. With this approach the number of columns stay the same, but as mentioned above it can not be assured whether the similarity is 0 due to missing values or different strings. For an instance, continuing from the above example, under this approach the entity vectors stay the same and feature vector will become [0,0,1]. In here, it can be seen that the second values in entity vectors are totally different from each other and that's why the similarity metric is 0. However, in the vector the first value is also 0, but this time it is because one of the fields is missing. Missing field doesn't mean different string.

In order to make this difference, third and fourth approaches have been applied, which distinguishes missing values from different strings using two distinct methods. In the third approach, this distinguishment is achieved by adding more columns to the feature vector that represents whether one of the fields is missing or not. With this approach, the feature vector will become [0,0,1,1,0,0], where the first three columns represent similarity metrics and the last three columns represent whether one of the fields that is being compared is missing or not. In this case fourth value will be 1, since it is missing. Lastly, in the fourth approach, missing values and strings with different values are distinguished by assigning -1 to the value of the similarity if either of the fields are missing. This approach will produce the following vector, [-1,0,1] and by assigning -1 to missing fields, it is hoped that the algorithm will be able to distinguish between these two cases.

5.4 Learning Algorithms

In the thesis, first, dataset has been splitted into train and test, where %67 of the pairs has been assigned to the train dataset. One can argue that when dataset is splitted into train and test in active learning, some of the important corner cases may be assigned to the test set, and therefore convergence maybe slower then the reality. Even though, this might have some truth in it, it can be argued that without splitting the dataset into train and test, model performance can not be measured correctly and overfitting can not be detected. Therefore, train and test split found to be highly necessary and crucial in active learning and have been applied to the datasets that are being used in the thesis prior to active learning.

Moreover, for the thesis four main supervised classification algorithms are selected, including; logistic regression, support vector machine (SVM), decision trees and random forest. Logistic Regression and SVM are chosen since these aim to estimate the posterior probability and therefore are less sensitive to how the training data is created, as a result highly fitting to the concept of active learning [47]. Decision trees are chosen since its one of the most widely used approaches in machine learning and random forest are chosen hence its one of the algorithms that reaches high accuracies.

5.5 Seed Generation

Active learning requires an initial set of labeled instances known as seeds to start learning. By properly selecting seeds for active learning, missed class effect can be handled and higher convergence can be reached. In the thesis two approaches have been applied; randomly selecting instances and clustering as proposed by Kang et al., with slight modification [37].

First of all, random seeding has been used since this is the most common approach in the literature. However, as mentioned in , Libact forces an instance from each class to be present in the seed set prior to active learning. Therefore, in this case random seeding becomes a trivial process of labeling randomly sampled instances, hoping that a pair with a match class, will come early in the process so that the real active learning can start. In order to overcome this trivial process of labeling, and also to reach faster convergence, seed clustering using Kmeans algorithm with 10 clusters have been applied, where instances from each cluster is chosen randomly. The number of clusters is chosen as 10 since this is the number of instances to be used in initial seeding. The clustering methodology first tested on Phones dataset using different variables, in order to find the best combination of features that would result with the purest clusters in terms of silhouette score. The experiments have shown that there are almost no difference in clusters applied on the same dataset using different variable combinations in terms of silhouette score and therefore the clusters have been decided to applied on the dataset which uses all similarity calculated variables, since it is believed to have the most discriminative power for distinguishing pairs from each other.

Kang et al., assumed that the clusters would consist of similar examples that likely to belong to the same category or class, and therefore it was expected to have at least one cluster which consist of all match pairs, which would then be easier to find a pair with match status earlier in the process. However, all of the experiments in the thesis show contrary evidence to his assumption as almost all the generated clusters have mix distributions of match and non match pairs. This

might be due to the fact that the clusters have been done poorly, as indicated with a silhouette score of 0.2 on average. This means that in a way, the iteration must still be repeated until a pair with match status has been reached, just like in random seeding. Nevertheless, when a simulation has been done with 500 iterations in order to detect on average how many instance labeling is needed for active learning to start, it was found out that with seeded clusters, it takes 89 instances on average to reach a pair with match status whereas with random sampling, this number is 96. So, this means that one needs to label on average 7 less pairs in order to find a matched pair while using seeded clusters.

While keeping this slight improvement in mind, the faster convergence theory has been tested and has been found that with cluster sampling the convergence has been reached after labeling 180 instances, whereas this number is 240 for random sampling. The comparison of both of these methods can be seen in figure 5.1. Given these results, it has been decided to use clustering approach for selecting seeds. This approach has been applied in the thesis by taking the train dataset with all computed variables and clustering them using Kmeans algorithm into 10 groups, and then selecting an instance from each cluster randomly to produce an initial seed until a matched pair has been selected.

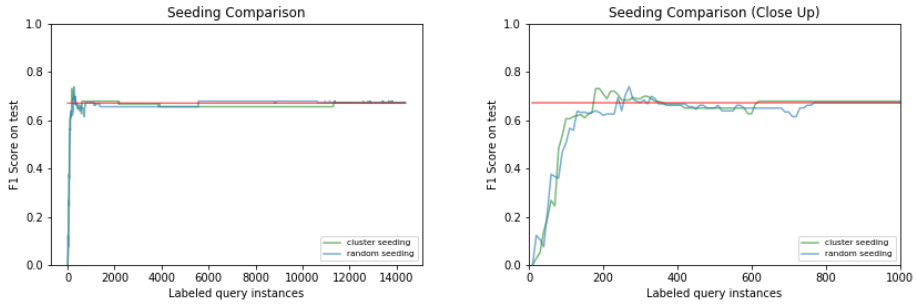

Figure 5.1: The graphs represent seeding comparison between random sampling and seeding with clusters. The graphs are created on dataset using all variables from Phones dataset with missing option 3, with uncertainty sampling and logistic regression

5.6 Query Selection Algorithms

Selecting appropriate query algorithms according to the dataset at hand is highly essential, since they determine which instances to be sent to the oracle to be labeled. However, it is also a difficult choice to make since there isn't a best querying strategy that works best for all the problem settings [61, 57]. This indicates that the best querying strategy most likely to depend on the chosen learning algorithm and its application to the problem. Additionally, previous work related

with active learning and data matching focused mainly on query by committee algorithms, where not so much emphasis have been given to other query selection methods like uncertainty and DWS. In order to fill this gap in the literature and find the best querying strategy that leads to faster convergence, in the thesis initially Random Sampling, Uncertainty Sampling, Query by Committee, Variance Reduction and Density Weighted Sampling query strategies have been decided to be used. However, variance reduction wasn't able to query a single data instance in two hours period of time that is being experimented on and therefore it has been dropped from the list of query strategies to be applied. The rest of the query selection methods have been applied with pool based active learning scenario, since pool of unlabelled data is already available and therefore, the whole dataset can be evaluated before making a query selection.

In order to measure the uncertainty in uncertainty querying strategy, Least Confidence has been used. However, since binary classification is being made, all of the parameters that can be used to measure the uncertainty would have yielded the same result considering the fact that all of the parameters query the instances with a class posterior closest to 0,5. Furthermore, for query by committee strategy, committees that have two or three models has been created, since committees with small number of models are shown to work well [59]. With this querying strategy, vote entropy has been used for the disagreement method due to non complexity. Moreover, for density weighted sampling, Density Weighted Uncertainty Sampling (DWUS) has been used along with DUAL algorithm, since it improves the original model in terms of efficiency by using clusters and re weighting calculated densities [22].

To be precise, in terms of applied active learning algorithms in combination with used learning methods, random sampling and uncertainty querying strategies have been applied with logistic regression, SVM, decision trees and random forest machine learning algorithms. On the other hand, density weighted sampling has only been applied with logistic regression, since this was the only compatible algorithm within the Libact package. Additionally, density weighted sampling has only been applied to the Phones Dataset prepared by missing handling option 4, since it took an average 8 hours to finish querying all the instances and received the worse results in terms of convergence overall applied settings. Lastly, DWUS has been applied with committee sizes of two using logistic regression models with regularization strengths 0.1 and 1.0, SVM models with penalty term 0.1 and 1.0 and decision tree models with quality of split gini and entropy. Only one model has been experimented with committee size of three using logistic regression, decision tree and SVM.

5.7 Batch Mode Active Learning

Active learning involves querying, labeling one instance in each iteration and then adding this single instance to the train dataset and then retraining the algorithm. As one can imagine, this takes a lot of time. The whole learning process could be made faster if the algorithm can be modified to handle multiple instances in each iteration. Based on this idea, a lot of research has been done that focuses on querying multiple instances [8, 31, 32]. In the thesis, a less complicated method has been followed.

In this method, instances are selected simultaneously, one at a time, but the classifier that is trained on the training dataset and tests on the test dataset is only employed after a specific number of instances (k) have been selected from the unlabeled examples. This helps accelerating active learning since, as the number of labeled examples that the classification algorithm trained on increases, the time it takes to train the classifier is also increasing. This can be also be seen in the experiments that is conducted by the author of the paper, with batch sizes of 1,10,20,50 and 100. The figure 5.2 demonstrates this experiment. From the figure it can be clearly seen that, the time it takes for the active learning to query all training instances, train on them and classify them, is decreases as the number of batch sizes increases. This continues until a batch size of 50 is reached, then time starts to increase again. So just by changing the batch size from 1 to 10, the time the active learning takes has been reduced from 2973 seconds to 1177 seconds, which makes our new algorithm almost %60 faster from the original. From batch size 10 to 20, there is almost no reduction, the same goes for 20 to 30.

In terms of batch sizes, there is an inverse relationship between interpretability of the algorithm and the batch size, as the batch size increases, interpretability of the algorithm decreases. For instance, with 50 batch size, the algorithm tests on test dataset after 50 instances are queried. As a result, the effect of the queried instances on accuracy measured with F1 score, can only be seen after querying 50 instances, and this makes it harder to detect exactly when convergence reached. Therefore, since there isn't a major increasement of time from 10 to 50, in the thesis 10 has been chosen as the appropriate batch size.

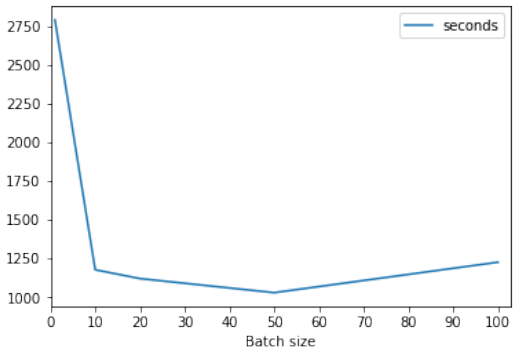

Figure 5.2: The effect of batch size on time needed to complete the active learning algorithm. This experiment has been conducted on Phones dataset with Jaccard similarity metrics and missing option 4, using uncertainty sampling and logistic regression

In algorithm 1, one may see the full algorithm for active learning with batch sizes without feature selection, that is being used in thesis. With feature selection, the algorithm slightly differs as feature selection has been added to the internal for loop. Feature selection has been discussed further in the next sub section. The algorithm is a function that is written in python by the author of the paper that uses libact library for query selection algorithms and scikit learn for learning the models [72, 48]. Before calling the 1 function, several initial steps are taken in the main code. Assuming that missing columns and columns that do not have correspondences in other databases are dropped from the dataset and similarity calculations have been made, the first step involves selecting columns according to the setting and refining the dataset with the selected columns. Then, train and test split has been made, 10 initial seeds are selected using clusters and train and test datasets have been put into the Dataset class in libact library to be used in modeling. As a last step, query sampling strategy and learning model have been declared, and the function described under 1 is called.

5.8 Feature Selection

In the thesis, multiple similarity metrics have been used, to account for different aspects of closeness as described in 5.2. In this setting, feature selection can be beneficial to get rid of the irrelevant attributes and to find the best similarity metric for each attribute.

Feature selection has been discussed in more detail previously in the subsection 3.3., here the incorporation of feature selection into active learning will be

Algorithm 1 Active learning with batch sizes without feature selection

RUNFASTER($traindataset$, $ytrain$, $Model()$, $QueryStrategy()$, $xtest$, $ytest$, $savename$, $batchsize$)

$f1score, labelholder, askedid, tn, fp, fn, tp \leftarrow []$
$part \leftarrow batchsize$
$k \leftarrow traindataset.len\_labeled()$
$kbeg \leftarrow traindataset.len\_labeled()$
$quota \leftarrow len(traindataset)$
while $k < quota$ **do**
  if $part < traindataset.len\_unlabeled()$ **then**
    $part1 \leftarrow part$
  else
    $part1 \leftarrow traindataset.len\_unlabeled()$
  end if
  $lbls, asks \leftarrow []$
  for i in $range(0, part1)$ **do**
    $ask\_id \leftarrow QueryStrategy().make\_query()$ ▷ *queries the instance*
    $asks.append(ask\_id)$
    $lb \leftarrow ytrain[ask\_id]$ ▷ *gets the label of the queried instance*
    $lbls.append(lb)$
    $traindataset.update(ask\_id, lb)$ ▷ *adds the queried instance to the train dataset*
  end for
  $labelholder.append(lbls)$
  $askedid.append(asks)$
  $Model.train(traindataset)$▷ *trains the model on the labeled trained dataset*
  $predy \leftarrow Model.predict(xtest)$▷ *tests the model on the whole labeled test dataset*
  $f1score.append(f1\_score(ytest, predy))$
  $tn.append(confusion\_matrix(ytest, predy)[0][0])$
  $fp.append(confusion\_matrix(ytest, predy)[1][0])$
  $tp.append(confusion\_matrix(ytest, predy)[1]10])$
  $k \leftarrow traindataset.len\_labeled()$
  $q \leftarrow [i$ for i in $range(kbeg, quota, part)]$
  $iter_{\leftarrow}[i$ for i in $range(0, len(f1score))]$
  $file \leftarrow DataFrame(iter, q, f1score, tn, fp, fn, tp, askedid, labelholder)$
  $file.to\_csv(savename)$
end while

discussed further. The feature selection that have been applied in the thesis is similar to the one proposed by Joshi and Xu, but differs from their approach from multiple aspects [36]. First of all instead of GainRatio algorithm, ANOVA-F test has been used and top 20 features with highest ANOVA F values are chosen. Secondly, feature selection has been done at the beginning of each iteration on the labeled trained dataset rather then on the selected samples. Thirdly, the features are selected individually and not added on top of each other, since the number of features in the datasets are far less then the ones that is being used in text analytics.

The approach follows, in the first iteration features FS_0 has been selected from 10 clustered seeds, and 10 instances have been selected for labeling S_0, using feature space FS_0. Then classification is trained on $initial\_seed + S_0$ with FS_0 features. In the second iteration, features FS_1 has been selected from $initial\_seed + S_0$, and S_1 has been selected for labeling using feature space FS_1. Then the classification has been made on $initial\_seed + S_0 + S_1$ with FS_1 features. The iteration continued until all the instances have been queried. By using this approach better classification accuracy is reached and relatively good results have been reached in terms of convergence. The issues that have been discussed by Joshi and Xu, has not been encountered due to the fact the algorithm has been modified in a way that features are selected considering all the labeled training data before feature selection starts.

In the thesis, feature selection has been applied to random sampling and uncertainty sampling with logistic regression and SVM, DWS with logistic regression only on the Phones dataset with missing option 4 and query by committee using logistic regression models with regularization strengths 0.1 and 1.0, SVM models with penalty term 0.1 and 1.0 and in a combination with logistic regression, decision tree and support vector machine. Feature selection has not been applied to decision tree models since they are capable of selecting their own attributes.

5.9 Evaluation Metrics

After classification, the results need to be evaluated in order to measure the success of the model. The evaluation of the results has been done on the test dataset after each iteration of active learning process and it requires both the train and test data to have assigned match status. For evaluating the results of the classification problems, confusion matrix is usually a common method that is used and it has the below form shown in the table 5.1 [71]. The abbreviations in the table refer to;

- True Positives (TP): These are the pairs that are predicted as matches and they are also actually matches.

- False Positives (FP): These are the pairs that are predicted as matches, but in reality are not matches.
- False Negatives (FN): These are the pairs that are predicted as non-matches, but in reality are matches.
- True Negatives (FN): These are the pairs that are predicted as non-matches and in reality are also non matches.

		Predicted Classes	
		Matches	Non Matches
Actual Classes	Matches	TP	FN
	Non Matches	FP	TN

Table 5.1: Confusion Matrix Demonstration

The goal of the learning algorithm is to correctly classify as many pairs as possible, while making sure that wrongly classified pairs are as low as possible [14]. This means keeping TPs and TNs high and FPs and FNs low. From the confusion matrix, various evaluation metrics can be calculated, however not all of them are suitable for the problem setting as the number of non-matched pairs dramatically exceeds the number of matched pairs in the dataset. This has implications on the used evaluation metrics, as the number of True Negatives have a tendency to be really high and this might cause certain evaluation metrics, like accuracy to be misleading. For target distribution of each dataset, one can look at section 4.2.3., where the problem of having imbalanced datasets has been discussed further with the graphs. Therefore, in order to overcome this issue f_1-score is chosen since it is prone to class distribution. Furthermore, besides confusion matrix dependent evaluation metrics, each algorithm has also been evaluated from time and convergence aspects.

5.9.1 F1 Score

F1 score is also known as the f-measure or f-score in the literature. It is a combination of, the ratio between number of pairs that are correctly classified as matches and the number of pairs that classified as matches and the ratio between number of pairs that correctly classified as matches and the number of pairs that are actually matches. It has a high value if both of these ratios are high and highest f-measure is reached when a good compromise is formed between these ratios [14].

$$fmeas = 2 * \frac{\frac{TP}{TP+FP} * \frac{TP}{TP+FN}}{\frac{TP}{TP+FP} + \frac{TP}{TP+FN}} \tag{5.1}$$

5.9.2 Time

Time is an evaluation metric used in the thesis, that doesn't depend on the confusion matrix. In machine learning, different algorithms might take different times to reach a result, even though they are trained on the same dataset. The average time an algorithm takes to train depends on the algorithms' computation complexity. For instance, neural networks most of the time will take a longer time to train compare to logistic regression on the same dataset. If two models that are trained on the same dataset, reaches comparable results in terms of F_1 scores, but one of them takes 1 hour and the other takes 3 hours, the one that takes the less time should be chosen as the best model. In the the thesis, time represent the number of seconds it takes to query and classify all the instances and it is measured on a MacBook Air with a processor of 1,8 GHz Intel Core i5 and memory of 8 GB.

processor, memory

5.9.3 Convergence

In active learning it is highly crucial to reach the same accuracy of the model that is trained on the full dataset, by labeling as less instances as possible. Therefore, in the thesis queries have also been examined from the convergence perspective. To be precise, the convergence definition used in the thesis is, the number of instances that needs to be queried in order to reach or exceed the same F_1-score trained on the full training dataset.

5.10 Summary

In this chapter, how the research problem of data matching of sparse datasets have been handled using active learning is discussed in detail. First the datasets have been prepared for modeling by dropping fully missing columns and variables that doesn't have a correspondence in the other dataset, followed by removing certain characters in the strings and then making them lower case. Then, Jaccard, Levenshtein and Jaro-Winkler similarities have been calculated and four different feature vectors have been created accordingly that uses; only Jaccard, only Levenshtein, only Jaro-Winkler and all calculated similarities. In the thesis, two missing handling options have been applied; missing option 3, where additional variables are

added that represent whether a variable is missing or not and missing option 4, where -1 is assigned to the similarity if either of the columns are missing. This means that for each dataset, eight different feature vectors have been prepared. After creating the feature vectors, dataset has been splitted into train and test, and seeds are generated using clustering with k means. Then, batch mode active learning have been applied, that queries instances one by one but evaluates them after querying 10 instances. For querying, on average of 12 models that uses random, uncertainty, density weighted and query by committee sampling strategies in combination with logistic regression, support vector machine, decision tree and random forest machine learning approaches, have been trained on each prepared feature vector for each dataset. Additionally, feature selection using ANOVA F-test has been incorporated into the active learning algorithm and for each missing handling option for each dataset, on average 7 models have been trained. The reason for incorporating and applying multiple settings together is to take every possible effect into account. All the applied settings are shown in the appendix C with table C.1.

Chapter 6

Experiments

In this chapter, the results of the thesis with the followed methodology, are discussed from the following aspects: the effect of missing handling, the effect of different similarity metrics, the effect of different query strategies and the effect of different learning algorithms. The first three effects have been evaluated from F1 score, convergence and time perspective, whereas the last effect has been evaluated from F1 score solely, since the other aspects have been covered by the other effects. In this section, given statistics related with F1 scores are independent of the querying algorithms and based on similarity, used missing handling option and employed machine learning algorithm. On the other hand, given statistics related with convergence and time, are calculated by excluding density weighted sampling and random sampling algorithms from the dataset and by including models that reached F1 scores above 0.1 in the whole train dataset. The results of the significant experiments can be found in appendix D, and full results of the experiments can be found in github `https://github.com/bngksgl/Master_Thesis`. The full list of applied settings can be found in appendix C with table C.1.

6.1 Missing Handling Effect

6.1.1 F1 Score

In terms of the accuracy of the algorithms measured with F1 score, the accuracies are slightly higher for models trained on datasets prepared using missing handling option 3 with 0.62 on average, compared to missing option 4 with 0.60 in all the datasets. To be more precise, for phones dataset the average F1 score with models trained on datasets prepared using missing handling option 3 is 0.53 and using missing handling option 4 is 0.49, for headphones dataset these percentages are

0.79 and 0.80 and for TVs are 0.55 and 0.49 . This might be to the fact that differentiating missing valued entries by adding more columns, matched and non matched instances are distinguished better from each other.

6.1.2 Convergence

Furthermore, there is a significant difference between the convergence rates. Among the tried settings, overall active learning queries in missing handling option 3, reached the F1 score calculated using the whole train dataset, more faster then queries in missing handling option 4. To be more precise, on average models trained on datasets with missing handling option 3, required 103 iterations with 1036 labeling whereas for models trained on datasets with missing handling option 4, this number is 175 with 1651 labeling. The effect is more immediate on headphones dataset, as it requires %48 less iterations to reach convergence, followed by phones dataset with %43 and TVs dataset with %40. This also supports the hypothesis that has been made in the above section where it has been argued that matches and non matches using missing option 3 performed better due to the fact that it can distinguish non matches from matches more efficiently. This higher distinguishing power with missing handling option 3, enabled more informative instances to be found and this led to faster convergence.

This phenomenon has been further represented in figure 6.1 the graphs represent active learning setting with QBC using two different decision trees trained on only Jaccard similarities. In the graphs, red thick line represents F1 score that is found by training on the whole dataset and testing on the test set, yellow dotted line represents the same setting using random sampling which sets the baseline, green line represents active learning with qbc sampling, y-axis shows the F1 score calculated by training on the labeled dataset and testing on the whole test set and x-axis represents the number of instances queried and labeled. In the given example, it can be visually seen that queries with missing handling option 3, reaches convergence by requiring less labeling data, compared to missing handling option 4. The fact that the discrepancies between both missing options exist while using the same query strategies, the same similarity metric, same active learning setting as well as the same machine learning algorithm suggests that the discrepancies occur due to handling missing values using different methods.

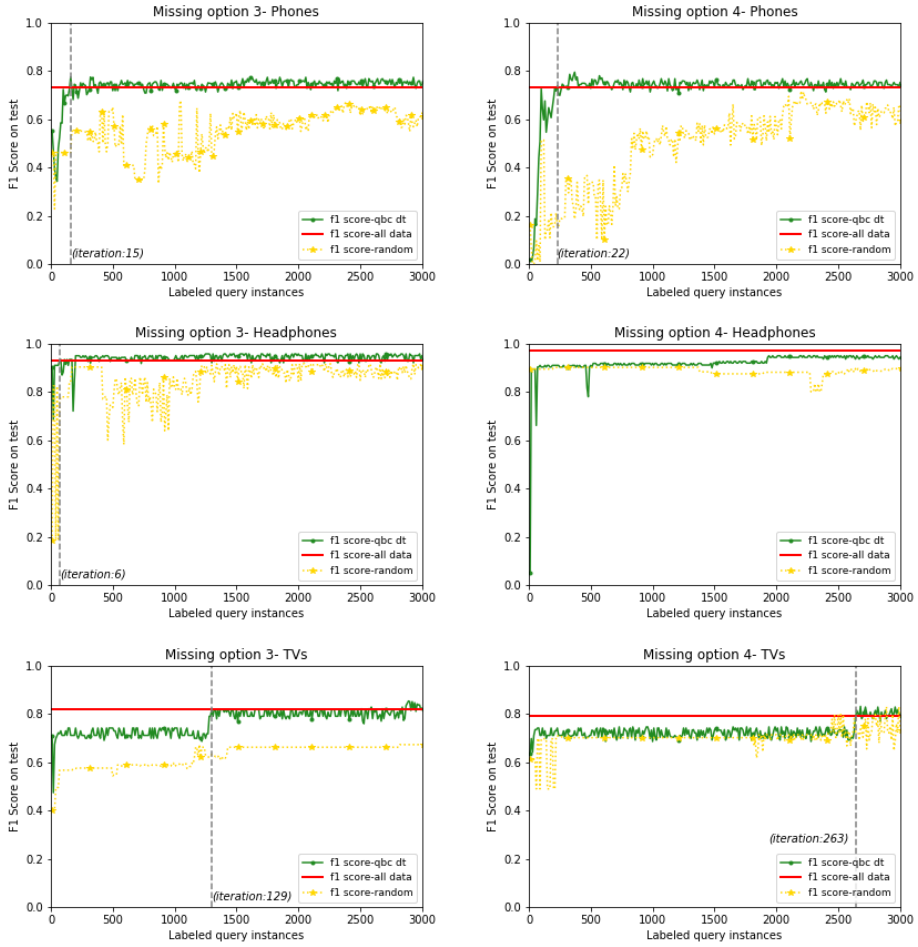

Figure 6.1: These figures demonstrate missing handling effect on active learning. As can be seen datasets with missing handling option 3 reaches convergence earlier then datasets with missing handling option 4. For the graphs, datasets that use only Jaccard similarity metrics have been used along with active learning setting with QBC with two different decision trees. The dotted perpendicular line in the figures represents convergence iteration. The graphs show the first 3000 labelings, therefore the settings without the dotted line may exist since this means that convergence is achieved at some point after querying and labeling 3000 instances.

6.1.3 Time

In terms of training time, active learning algorithms with both missing handling options completed the training almost the same time in TVs and Headphones dataset. However, in Phones dataset, large discrepancies have been seen in training time with uncertainty sampling using logistic regression and SVM. Since these cases occurred only in a specific dataset and settings, it is considered as anomalies and

not investigated further. Therefore it can be concluded that in terms of timing, the performance of the both missing handling options are the same.

6.2 Similarity Metric Effect

6.2.1 F1 Score

In the thesis, Jaccard, Levenshtein and Jaro-Winkler similarity metrics have been used to create five different feature vectors. These feature vectors include; only Jaccard, only Levenshtein, only Jaro Winkler, all created variables using Jaccard, Levenshtein and Jaro Winkler, and variables after feature selection. Among these five different vectors, models using variables after feature selection have on average highest accuracies measured by F1 score on all the datasets. This is followed by datasets that have all the features and uses only Jaccard similarity metric. Even though, in dataset level there are slight variation in the ordering, this hypothesis most of the time holds. These findings suggest that, by incorporating different similarity measures that evaluates the similarity of records from different aspects, the matched instances can be distinguished better then the non matched instances.

Among all the tried models, the worst accuracies in terms of F1 score has seen on models trained on datasets prepared using only Jaro-Winkler similarity metrics. This can be associated with the fact that Jaro-Winkler gives higher estimates to two strings that are different from each other, whereas Jaccard and Levenshtein penalizes them more. This also can be seen with the example used in section to explain the different similarity metrics. Given two strings; 'iphone 6' and 'one m9', the similarity has been calculated as 0.72 with Jaro-Winkler, 0.375 with Levenshtein and 0.14 with Jaccard. Due to these penalization differences, Jaccard similarity metric work better than Jaro-Winkler. The effect of similarity metrics on different datasets in terms of F1 scores can also be seen in figure 6.1.

6.2.2 Convergence

With respect to the effect of similarity metrics on convergence, it has been found that datasets prepared with feature selection reaches on average the quickest convergence. This is fallowed by datasets prepared with using only Levenshtein and using all calculated features. The worst performing similarity metric in terms of convergence, differs from dataset to dataset, where datasets prepared using only Jaccard similarity metrics performed worse in headphones, while datasets prepared using Jaro-Winkler similarity metrics performed worse in both phones and TVs. This ordering holds for most of the datasets and settings.

To be precise, the average convergence for datasets prepared using feature selection are as follows; 18 iteration and 189 labeling in phones, 8 iteration 93 labeling in headphones and 42 iteration and 433 labeling in TVs. This arises from the fact that, in each iteration of active learning, the algorithm is trained on the dataset prepared with top 20 selected variables. This enables the active learning algorithm, as well as the learning algorithm, to focus on the most important attributes and in return, instances that are actually more informative can be detected and queried. Thus, faster convergence have been reached. Additionally, by focusing on the most important attributes higher accuracy have been obtained as well. The convergence effect has been shown on the right hand side of figure where the dotted lines represent the convergence 6.1.

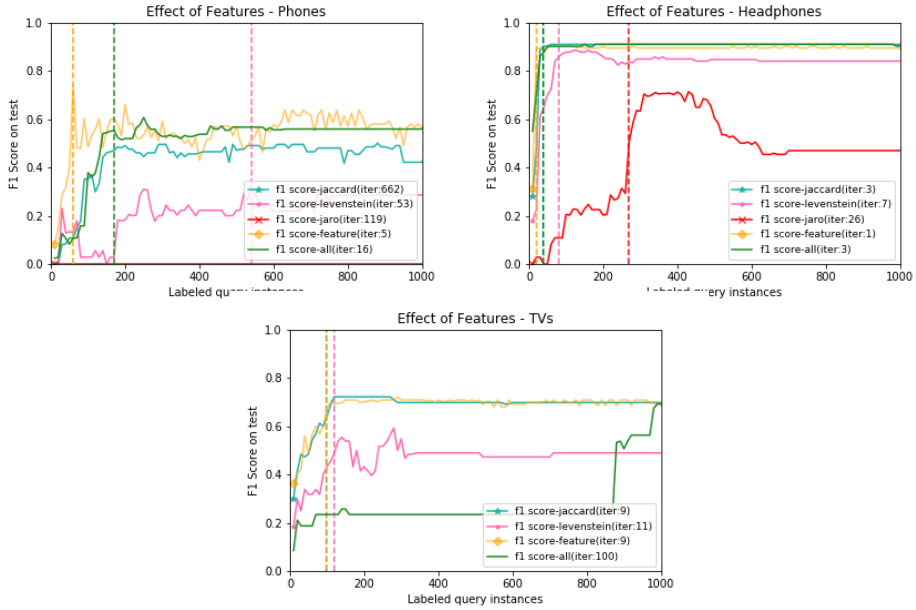

Figure 6.1: These figures demonstrate the effect of feature selection on the success of the algorithm measured with F1 Score. For the querying algorithm uncertainty with logistic regression has been chosen along with missing option 4. Dotted perpendicular lines in the figures represent convergence with number of instances to be labeled. The graphs show the first 1000 labelings, therefore settings without the dotted line may exist since this means that convergence is achieved at some point after querying and labeling 1000 instances. Models resulted with F1 score below 0.1 is not included in the graphs, since it might be misleading. This is why models that use Jaro-Winkler are excluded from the TVs dataset.

6.2.3 Time

Respect to time, it has been discovered that datasets prepared with feature selection takes the longest time with average of 218 minutes, followed by datasets that

include all variables with 151 minutes. It is expected for models with feature selection to take longer time, since they include feature selection inside. It is also somewhat expected for models that include all the features to take more time for querying, since in order for querying to be done, in each iteration, the model is fitted. Thus, datasets that have more features, will take longer time to train and therefore it will take longer time for them to find an instance to be queried.

6.3 Query Strategy Effect

6.3.1 Convergence

In terms of convergence, QBC methods overall wins by requiring on average 66 iterations and 570 labeled instances. This is followed by uncertainty sampling on average with 185 iterations and 1862 labeled instances and DWS with 1401 iterations and 14014 labeled instances. Even though the actual ordering may vary among datasets and different settings, this holds for most of the applications.

Among different QBC approaches, the approach *mixall*, which uses decision tree, logistic regression and SVM learning algorithms to find the instances to be queried, have the fastest convergence rate in all of the datasets with 9 iterations and 95 labels on average. This is due to the fact, QBC based on the idea of version space restriction and by using different algorithms rather then using the same algorithm with different parameters, the version space restricted the most and consequently controversial regions of the version space is reached. On the other hand, among the applied uncertainty sampling methods, overall uncertainty sampling with SVM stands out from the rest, with the fastest convergence rate with 6 iterations and 74 labeling on average. However, SVM methods also produces the least accuracies and therefore this must be approached in caution.

With regard to the active learning algorithm effect combined with machine learning algorithms, in general QBC sampling with decision tree reaches convergence 5 times more faster then uncertainty with decision trees. This suggests that decision trees is not good at finding informative instances when it is employed alone. On the other hand, random forest is shown to perform well alone, with an average of 15 iterations using uncertainty sampling. This could be due to the fact that, random forest already creates multiple weak decision trees inside while making prediction. Moreover, contrary to decision trees, uncertainty sampling models reach convergence faster than the QBC models with SVM and logistic regression. To be more precise, on average uncertainty sampling models with SVM reach convergence after 6 iterations compared to QBC sampling models with 84 iterations and uncertainty sampling models with logistic regression reach convergence after 41 iterations compared to QBC sampling models with 75 iterations. This can be

associated with how decision trees, SVMs and logistic regression models used in QBCs are differentiated from each other. In QBC, it is highly important for models to be different from each other, because the success of QBC, depends on how the version space is restricted. These results suggest that, by using different regularization strengths in logistic regression and different penalty terms in SVM, the models stayed relatively similar and therefore the version space is not restricted and informative instances can not be found, and in return worse convergence rates have been obtained. On the other hand, by changing the splitting criteria in decision trees, two sufficiently different decision tree models have been obtained and better convergence rates have been achieved. The detailed results can be seen in table 6.1 where average iterations have been given for each dataset along with used active learning strategy and machine learning algorithm. These results holds for majority of the settings and databases.

Active Learning Strategy	Machine Learning Algorithm	HeadPhone	Phone	TV	Overall Average
DWS			1401		1401
	log		1401		1401
QBC		54	38	114	67
	mixall	10	12	14	9
	dt	146	20	171	112
	log	14	59	158	75
	svm	74	62	190	84
Uncertainty		165	137	266	185
	dt	645	405	858	636
	log	8	100	26	41
	rf	18	19	9	15
	svm	9	5	3	6

Table 6.1: Query selection strategy effect on convergence measured in iterations active learning need to do in order to reach the same F1 score if one would have the whole dataset

A more deeper look into the query learning algorithms showed that as the number of queried instances with match label increases in early stages of the algorithm, the convergence is reached faster. In other words, if query selection algorithms can query the instances that lie on the corners, they will most likely to query the instances that have match labels, since they are the rare class in this problem setting and consequently more informative then pairs which have non-match status and as a result the convergence will be reached faster. This hypothesis holds for most of the settings and the models, and gives a good explanation to the poor performance to the density weighted sampling. As mentioned, among the different querying approaches, density weighted sampling performed the worse in terms of number of labeled queried instances needed for convergence, while requiring on average %80

of all the training dataset to be labeled. This is due to the fact that the algorithm can not reach to the corner cases as well as uncertainty and QBC sampling. Therefore, the algorithm can't query any pairs with matched labels until on average it has already queried %34 of all the training dataset. The general snapshot of this hypothesis for each querying algorithm for the different datasets can be seen in 6.3. This can also be seen in further detail in figure 6.2, with Phones dataset, where DWS queried its first match pair on its 532nd iteration. The reason DWS's lack of ability to reach corner cases lie within its design of the algorithm. Compared with other query strategies, DWS queries instances that have high information context as well as that are representative of the underlying data distribution, so it might concentrate on instances that don't improve the algorithm greatly but are align with the underlying data structure [22]. This fact has already been taken into account by combining DWS with DUAL algorithm, which weighs the information context and representativeness of the instance while querying. However, the results indicate that this solution wasn't enough since DWS resulted with the worst convergence rates.

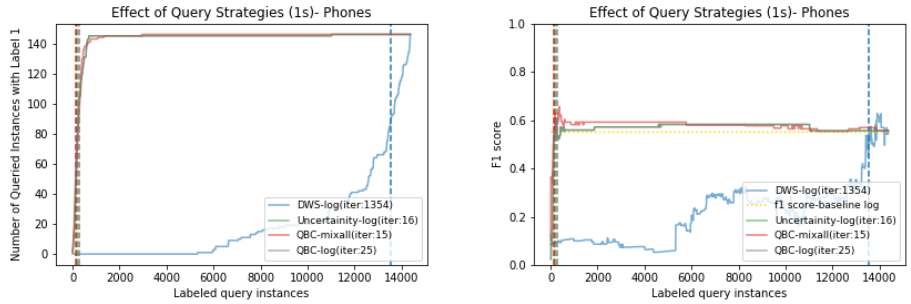

Figure 6.2: The left graph shows relationship between number of queried pairs with labeled as matches and number of labeled instances. The right graph shows the convergence of the algorithms used on the left hand side. The graphs are created on dataset using all variables with missing option 4. The dotted perpendicular lines represent the convergence.

Furthermore, one can see that before flattening, there are fluctuations in the F1 scores and even some cases the algorithms exceed the F1 scores that would have been obtained if the model would be trained on the full training dataset, like in QBC with *mixall* as shown on the right hand side image in 6.2. The first case can be associated with the chosen machine learning algorithm as well as the query strategy. For instance, decision tree and random forest models tend to have a lot of up and downs with high fluctuations at the beginning and fluctuations tend to get less as the number of instances the models trained on, increases. This is also something that has been experienced in the literature [55]. On the other hand,

the latter case is related solely with the querying strategy. In successfull active learning algorithms, most of the queried instances at the beginning are instances with match classes. Thus, in some cases these matched instances denominate the datasets in that particular iteration and cause the algorithm to overfit to the matched class in these iterations. However, as more labeled examples are received, matched instances become rare class, and more errors have started to be occurred on the matched instances, and this in return causes the F1 score to return back to the convergence [20].

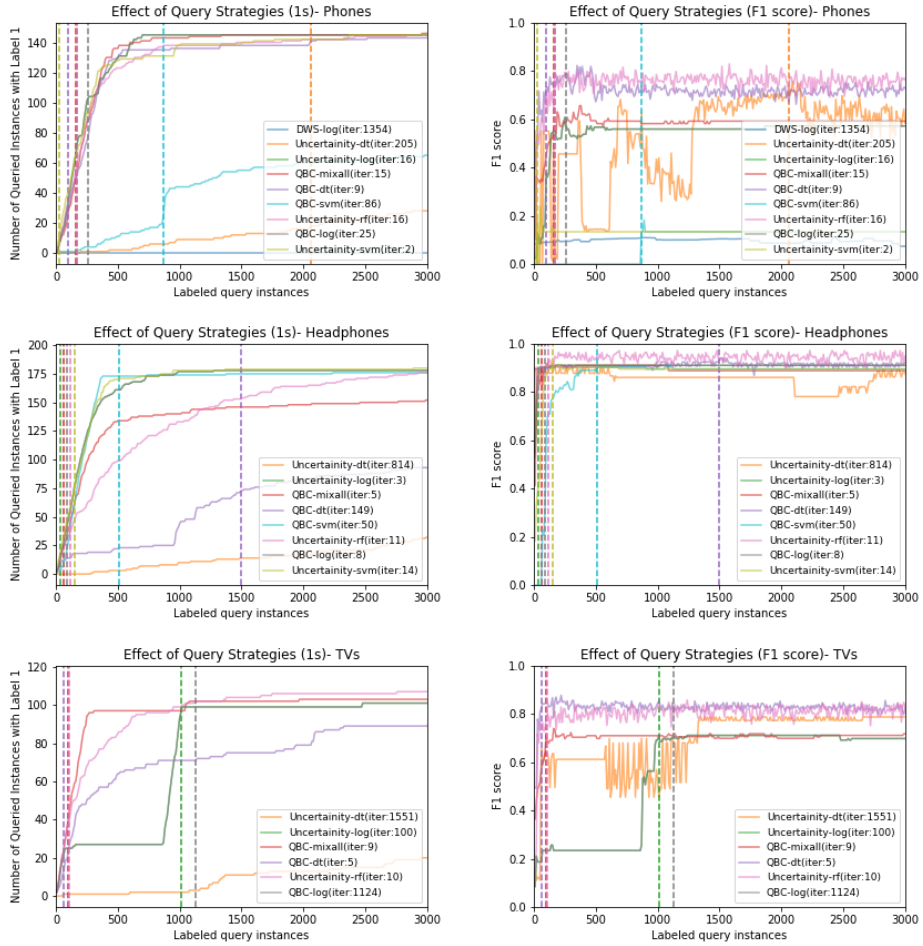

Figure 6.3: The left graph shows relationship between number of queried pairs with labeled as matches and number of labeled instances. The right graph shows the convergence of the algorithms used on the left hand side. The dotted perpendicular lines represent the convergence. The graphs are created on dataset using all variables with missing option 4. Models resulted with F1 score below 0.1 are not included in the graphs, since it might be misleading. This is why SVM models are excluded from the TVs dataset. The graphs show the first 3000 labelings, therefore settings without the dotted line may exist since this means that convergence is achieved at some point after querying and labeling 3000 instances.

6.3.2 Time

Query strategies also take various times to query an instance, therefore timing is an another important aspect to be taken into consideration while evaluating the query strategies. The analysis showed that on average, uncertainty sampling requires 61 minutes to label the whole training dataset, followed by QBC with 186 minutes

and DWS with 445 minutes. This ordering is somewhat expected since uncertainty sampling needs only one learning algorithm to determine which query instances to sent to oracle to be labeled, therefore it is faster then QBC, where multiple models are used to decide on instances. In terms of tried QBC specific approaches, the one that uses two decision trees with different parameters found to be the fastest algorithm with 71 minutes on average, followed by only log with 139, only svm with 272 and mixall with 288. In terms of tried uncertainty approaches, the one that uses decision tree found to be the fastest algorithm with 27 minutes on average, followed by random forest with 30, logistic regression with 72 and SVM with 140. DWS is the slowest sampling algorithm among all of the tried algorithms since it considers the whole training dataset while making querying decisions, and therefore it is very computationally expensive.

6.4 Learning Algorithm Effect

6.4.1 F1 Score

Among the applied algorithms of SVM, logistic regression, decision trees and random forest, regardless of the chosen feature vector, SVM resulted with the worst accuracies in terms of F1 score. In some settings it was unable to separate positive and negative classes from each other and returned a F1 score of 0. More specifically in cases where F1 score is 0, the algorithm predicted all the non matches as matches and all the matches as non matches. Further analysis, has shown that this is due to having unbalanced training dataset. In fact, this is a well known problem with support vector machines trained on unbalanced datasets and can be solved by applying penalties to the classification errors on the scarce class while training [33]. So simply put, by applying class weights. However, this solution has not been applied since it occurred in cases where only one type of similarity metric has been used to construct feature vectors and parameter optimization is out of the scope of the thesis. The second worst performing algorithm is logistic regression. Its average F1 accuracy is 0.56, which is significantly higher compared to SVM with 0.25. This suggests that overall, logistic regression was able to separate positive and negative classes from each other more accurately compared to SVM.

The best performing algorithms are decision trees and random forests. This might be due to the fact that, decision trees and random forest algorithms make multiple decision boundaries by cutting the feature space into rectangles parallel to the axis, whereas logistic regression assumes a single linear decision boundary. This ability of decision tree and random forest to cut the feature space while using different decision boundaries, enables proper rules to be created by using calculated similarities and this makes the algorithms more suitable for the thesis setting from

the accuracy perspective. For instance, the right image in figure 3.2, divides the feature space into four rectangles using two decision boundaries and two similarity metrics. The result of the tree can be summarized into three rules, if birthday is above 0.5 and surname is above 0.15, on the other hand a pair is not a match if birthday is below 0.5 or birthday is above 0.5 and surname is below 0.15. This kind a separation of classes is not possible with logistic regression or SVM, and that's why these rule based algorithms are performing well on the datasets.

6.5 Summary

In this section the results of the thesis has been discussed from the missing handling, similarity metric, query strategy and learning algorithm point of views. To start with, from the missing handling perspective, it has been found out that datasets prepared using missing handling option 3 performed better then datasets prepared using missing handling option 4 in terms of F1 scores and convergence in all the datasets. With respect to timing, no difference is detected between these approaches.

Secondly, with regards to similarity metric effect, it has been discovered that datasets prepared with using feature selection with F classification, all similarity calculated attributes and only Jaccard similarity metrics shown comparably good results in terms of F1 scores and convergence. However, datasets with feature selection and all similarity calculated features, have failed from the timing perspective. Feature selection resulted badly in combination with active learning due to the fact that in the thesis setting, in each iteration of the algorithm feature selection is being done and this puts additional straints to the algorithm. On the other hand, all similarity calculated features resulted badly because that in each iteration the model is fitted to the dataset to find the informative points, therefore when all variables used, the number of features in dataset triples and this creates additional workload.

Thirdly, with respect to the query strategy algorithm, QBC converges faster then uncertainty querying strategies over all datasets. To be precise, among the applied QBC settings, *mixall* that uses decision trees, logistic regression and SVM to determine informative instances and among the applied uncertainty settings, uncertainty with SVM, are found to be the fastest converged algorithms. It has also been discovered that changing splitting rules is a good methodology to make decision trees different from each other whereas for logistic regression and SVM the splitting criteria of different regularization strengths and different penalty terms did not work well as the resulted models turn out to be similar and therefore the informative instances can not be detected easily and this cause the convergence to

be slower. Even though QBC found to be a good method in terms of convergence, with respect to timing, it is worse then uncertainty sampling since for each informative instance to be queried, multiple models need to be trained and this makes the algorithm slower. Overall, density weighted sampling performed the worse from both convergence and time aspects, due to the fact that DWS queries instances that have high information context and represent the underlying data distribution, and this might cause the algorithm to concentrate on the instances that don't improve the algorithm but represent the underlying data structure.

Lastly, from the learning algorithm perspective, decision tree and random forest resulted with the best models in terms of F1 score, whereas SVM performed as the worst model. Further analysis has shown that this is due to having unbalanced training dataset and can be solved by applying penalties to the classification errors made on rare class.

In the light of these findings and keeping F1 score, convergence and time aspects in mind, the best querying strategy that works best for all the datasets are determined to be uncertainty sampling that uses random forest, only Jaccard similarity calculated features and missing handling option 3. The model has an average F1 score of 0.8, average iteration with 13 and average training time with 29 minutes overall datasets. Further reasoning as follows; datasets prepared with missing handling option 3 reached higher convergence and F1 score. With regard to the similarity metrics feature selection, using all variables and only Jaccard perform good in terms of convergence and F1 scores, however in terms of timing feature selected variables and using all variables resulted with the worst timing. Concerning the query selection method, QBC reached good convergence rates, however it takes a lot of time to train and with respect to machine learning algorithms, SVM and logistic regression resulted with models with low accuracy. The significant results of the tried approaches can be found in the appendix D.1 and the full results of each tried approach can be found in the github.

Chapter 7

Conclusion & Future Work

7.1 Conclusion

In the thesis, the problem of integrating sparse datasets related with products against a created product catalog has been investigated by using active learning algorithms. This task involves certain challenges, which have not been addressed widely in the literature. This is explicitly the area, where this thesis contributes to the already existing work. The challenges of matching sparse datasets using active learning include finding appropriate strategies to handle missing values prior to data matching, trying different query strategies for data matching tasks and employing different learning algorithms in combination with active learning strategies. For this task, the dataset, 'Web Data Commons for product matching and feature extraction' (WDC) that was prepared by Data and Web Science Group from University of Mannheim, has been used. This dataset includes products from three distinct categories; phone, headphone and TV. It has already have had more then 75000 matches between product catalog and crawled product pages, which facilitated the application of supervised machine learning methods in combination with active learning.

For the data matching task, multiple steps have been taken in this thesis. Firstly, WDC datasets have been consolidated together to make one single dataset for phones, headphones and TVs. After that, these created datasets have been preprocessed by first dropping attributes that do not have a correspondence in the other dataset and that are all missing, followed by removing certain characters from the attributes and lower casing them. After the preprocessing, missing values have been handled by applying two newly proposed strategies. The first one solves this issue by adding new columns to the feature vector while assigning 1, if either of the fields under comparison is missing. The second one handles missing values by

assigning -1 to the similarity if either of the fields under comparison is missing, without increasing the number of columns. Then, Jaro-Winkler, Levenshtein and Jaccard similarity metrics have been calculated for both missing handling options to measure the closeness of two records in a pair and eight different datasets have been created accordingly for each category. Additionally, feature selection with F classification has been integrated into the active learning algorithm to find the best combination of similarity metrics and attributes.

After the datasets are ready, train and test split have been done and initial seed set for active learning has been created by first clustering the training dataset into 10 groups and by selecting a random instance from each group iteratively until an instance with match status has been selected. This clustering approach to seeding, is used to overcome the *missed class effect* and to reach convergence faster. For active learning, a variation of batch active learning technique that has been proposed in the thesis, is used where the instances are queried by one by, but the evaluation of the results are done after 10 instances have been queried. In active learning, random sampling, uncertainty sampling, query by committee and density weighted sampling have been used in combination with decision trees, random forest, support vector machine and logistic regression. The results have been evaluated from the aspects of accuracy measured with the F1 score, convergence and time.

The main result of the experiments demonstrated that datasets that handles missing values by adding new columns to the feature vector that assigns 1, if either of the fields under comparison is missing performed better in terms of accuracy and convergence. This can be associated with the fact that by adding more columns to the dataset that represent missing valued entries, matched and non matched instances are distinguished better from each other. Furthermore, with regards to used similarity metrics, datasets with feature selection, all similarity measures and only Jaccard, have reached better results in terms of F1 score and convergence overall datasets, whereas Jaro-Winkler resulted with the worst. However, datasets with feature selection and all similarity metrics have resulted with worst outcomes in terms of timing, as they put additional strains on the active learning algorithm by applying feature selection and using all calculated variables, respectively.

Moreover, in terms of query selection strategy, QBC was found to be the best over all datasets in terms of convergence. QBC that uses three different models is specially found to be successful compared to other QBC models that uses the same algorithm while changing parameters. This is associated with the fact that, the way the parameters are changed in the thesis setting to differentiate between two models, may not be enough to make the models significantly different from each other. When the models that are used in the committees are similar, version space is not restricted and therefore informative instances can not be found. However, QBC methods have a major drawback as they take on average 3 times more time to train

compared to uncertainty sampling, due to the fact that they require multiple models to be trained to find the informative instances. Overall, density weighted sampling performed the worst from both convergence and time aspects. This is believed to be related with its design of the algorithm as they look for instances both informative and representative of the underlying data distribution, which causes the algorithm to concentrate on instances that represent the underlying data structure but not necessarily improve the algorithm. Regarding the learning algorithms, decision trees and random forests are produced the best F1 scores overall datasets whereas SVM performed the worst, often resulting with 0 accuracy due to not being able to distinguish matches from non matches.

Considering these findings and the aspects of F1 score, convergence and time, the best strategy for all the used datasets is determined to be uncertainty sampling that uses random forest, only Jaccard similarity calculated features and missing handling option 3. The model has an average F1 score of 0.8, average iteration of 13 and average training time of 29 minutes overall datasets.

7.2 Future Work

Although the provided analyzes and methodologies in the thesis are believed to be innovative and satisfactory, there are still some room for improvements. First of all, blocking techniques that partition the records into smaller subsets based on a determined partitioning key and compare the records only within these subsets, can be used to decrease the amount of comparisons to be made. This in return makes the dataset smaller and therefore the algorithm can complete its iterations in less time.

Secondly, while selecting seeds k means clustering with 10 clusters has been used. Previous analysis, regarding clustering quality indicate that these clusters are not formed really well. Even though with this approach less queries need to be made to find an instance with match label and a faster convergence is obtained compared to random sampling, better clustering methods can be applied that makes the clusters differentiate from each other more. This in return may further decrease the amount of queried instances needed to find a match and may make the convergence faster.

Thirdly, in order to make the active learning faster, a variation of batch active learning has been applied, where the active learning algorithm still queried the instances one by one, but prediction was made on the training dataset only after 10 instances had been queried and labeled. This approach already made the algorithm %60 faster, but this improvement can be further increased by modifying the active learning algorithm to query for multiple instances rather than a single instance.

This way, the model that is fitted to find instances to be queried, will be trained in less amount of times, and this will make the algorithm faster.

Fourthly, as mentioned, QBC sampling with two models did not work well since the models use the same algorithm with different parameters, and these used parameters were not good enough to differentiate the models. Therefore, better parameters that differentiate the same models can be found and experimented on. This way, better QBC models that uses the same learning algorithm can be constructed.

Lastly, hyperparameter tuning is something that has not been considered in the thesis. Every machine learning and active learning model, have parameters which define the model architecture known as hyperparameters, these parameters can be used to configure the model to make it more suitable for the problem setting. For instance, in the thesis, SVM often resulted with models with 0 F1 scores. Further analyzes showed that this is due to having unbalanced datasets which can be fixed by assigning class weights to the classes, therefore by tuning the used parameters. Additionally, active learning strategies can also be tuned. As an example, uncertainty in uncertainty sampling can be measured using margin sampling, least confidence and entropy strategies, whereas the disagreement of models in QBC can be measured by using vote entropy and kullback-leibler divergence. Hyperparameter tuning can be employed to find the best uncertainty measure for uncertainty sampling and best disagreement measure for QBC sampling, for the appropriate problem setting. As far as the author knows, no previous research has been done to investigate hyperparameter tuning in the active learning setting, both for applied active learning query strategies and machine learning models. Therefore, this thesis establishes a direction for a research area that has lots of potential for further research.

Bibliography

[1] Arvind Arasu, Michaela Götz, and Raghav Kaushik. On active learning of record matching packages. In *Proceedings of the 2010 ACM SIGMOD International Conference on Management of Data*, SIGMOD '10, pages 783–794, New York, NY, USA, 2010. ACM.

[2] Les Atlas, David Cohn, Richard Ladner, M. A. El-Sharkawi, and R. J. Marks, II. Advances in neural information processing systems 2. chapter Training Connectionist Networks with Queries and Selective Sampling, pages 566–573. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1990.

[3] Jacob Berlin and Amihai Motro. Database schema matching using machine learning with feature selection. In *Proceedings of the 14th International Conference on Advanced Information Systems Engineering*, CAiSE '02, pages 452–466, London, UK, UK, 2002. Springer-Verlag.

[4] Mikhail Bilenko and Sugato Basu. Adaptive product normalization: Using online learning for record linkage in comparison shopping. In *Proceedings of ICDM-2005*, pages 58–65. IEEE, 2005.

[5] Mikhail Bilenko and Raymond J. Mooney. Learning to combine trained distance metrics for duplicate detection in databases. Technical report, 2002.

[6] Mustafa Bilgic. Combining active learning and dynamic dimensionality reduction. In *SIAM International Conference on Data Mining (SDM)*, 2012.

[7] Aleksey Bilogur. Missingno: a missing data visualization suite. *The Journal of Open Source Software*, 3(22):547, 2018.

[8] Klaus Brinker. Incorporating diversity in active learning with support vector machines. In *Proceedings of the Twentieth International Conference on International Conference on Machine Learning*, ICML'03, pages 59–66. AAAI Press, 2003.

[9] Jason Brownlee. Bagging and random forest ensemble algorithms for machine learning, Apr 2016.

[10] Jason Brownlee. Logistic regression for machine learning, Apr 2019.

[11] Yukun Chen and Subramani Mani. Active learning for unbalanced data in the challenge with multiple models and biasing. In Isabelle Guyon, Gavin Cawley, Gideon Dror, Vincent Lemaire, and Alexander Statnikov, editors, *Active Learning and Experimental Design workshop In conjunction with AISTATS 2010*, volume 16 of *Proceedings of Machine Learning Research*, pages 113–126, Sardinia, Italy, 16 May 2011. PMLR.

[12] Beibei Cheng. *Data Fusion by Using Machine Learning and Computational Intelligence Techniques for Medical Image Analysis and Classification*. PhD thesis, 2012. AAI3537391.

[13] Peter Christen. Automatic record linkage using seeded nearest neighbour and support vector machine classification. In *Proceedings of the 14th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '08, pages 151–159, New York, NY, USA, 2008. ACM.

[14] Peter Christen. *Data Matching: Concepts and Techniques for Record Linkage, Entity Resolution, and Duplicate Detection*. Springer Publishing Company, Incorporated, 2012.

[15] Munir Cochinwala, Verghese Kurien, Gail Lalk, and Dennis Shasha. Efficient data reconciliation. *Inf. Sci.*, 137(1-4):1–15, September 2001.

[16] William W. Cohen and Jacob Richman. Learning to match and cluster large high-dimensional data sets for data integration. In *Proceedings of the Eighth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '02, pages 475–480, New York, NY, USA, 2002. ACM.

[17] David Cohn, Les Atlas, and Richard Ladner. Improving generalization with active learning. *Machine Learning*, 15(2):201–221, May 1994.

[18] David A. Cohn, Zoubin Ghahramani, and Michael I. Jordan. Active learning with statistical models. *CoRR*, cs.AI/9603104, 1996.

[19] J. de Freitas, G. L. Pappa, A. S. da Silva, M. A. Goncalves, E. Moura, A. Veloso, A. H. F. Laender, and M. G. de Carvalho. Active learning genetic programming for record deduplication. In *IEEE Congress on Evolutionary Computation*, pages 1–8, July 2010.

[20] Dmitriy Dligach and Martha Palmer. Good seed makes a good crop: Accelerating active learning using language modeling. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies*, pages 6–10, Portland, Oregon, USA, June 2011. Association for Computational Linguistics.

[21] AnHai Doan, Alon Halevy, and Zachary Ives. *Principles of Data Integration*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1st edition, 2012.

[22] Pinar Donmez, Jaime G. Carbonell, and Paul Bennett. Dual strategy active learning. In *Proceedings of ECML 2007*. Springer Verlag, September 2007.

[23] Halbert L. Dunn. Record linkage. *American Journal of Public Health and the Nations Health*, 36(12):14121416, 1946.

[24] Kenneth Dwyer and Robert Holte. Decision tree instability and active learning. In Joost N. Kok, Jacek Koronacki, Raomon Lopez de Mantaras, Stan Matwin, Dunja Mladenič, and Andrzej Skowron, editors, *Machine Learning: ECML 2007*, pages 128–139, Berlin, Heidelberg, 2007. Springer Berlin Heidelberg.

[25] M. G. Elfeky, V. S. Verykios, and A. K. Elmagarmid. Tailor: a record linkage toolbox. In *Proceedings 18th International Conference on Data Engineering*, pages 17–28, Feb 2002.

[26] Ahmed K. Elmagarmid, Panagiotis G. Ipeirotis, and Vassilios S. Verykios. Duplicate record detection: A survey. *IEEE Transactions on Knowledge and Data Engineering*, 19(1):1–16, 2007.

[27] Nadir Omer Fadl Elssied, Othman Ibrahim, and Ahmed Hamza Osman. A novel feature selection based on one-way anova f-test for e-mail spam classification. 2014.

[28] I. P. Fellegi and A. B. Sunter. A theory for record linkage. *Journal of the American Statistical Association*, 64:1183–1210, 1969.

[29] Yoav Freund, H. Sebastian Seung, Eli Shamir, and Naftali Tishby. Selective sampling using the query by committee algorithm. *Machine Learning*, 28(2):133–168, Aug 1997.

[30] Mohamed G. Elfeky, Vassilios Verykios, Ahmed Elmagarmid, Thanaa Ghanem, and Ahmed R. Huwait. Record linkage: A machine learning approach, a toolbox, and a digital government web service. 04 2004.

[31] Steven C. H. Hoi, Rong Jin, and Michael R. Lyu. Large-scale text categorization by batch mode active learning. In *Proceedings of the 15th International Conference on World Wide Web*, WWW '06, pages 633–642, New York, NY, USA, 2006. ACM.

[32] Steven C. H. Hoi, Rong Jin, Jianke Zhu, and Michael R. Lyu. Batch mode active learning and its application to medical image classification. In *Proceedings of the 23rd International Conference on Machine Learning*, ICML '06, pages 417–424, New York, NY, USA, 2006. ACM.

[33] I. A. Illan, J. M. Gorriz, J. Ramirez, F. J. Martinez-Murcia, D. Castillo-Barnes, F. Segovia, and D. Salas-Gonzalez. Support vector machine failure in imbalanced datasets. In José Manuel Ferrández Vicente, José Ramón Álvarez-Sánchez, Félix de la Paz López, Javier Toledo Moreo, and Hojjat Adeli, editors, *Understanding the Brain Function and Emotions*, pages 412–419, Cham, 2019. Springer International Publishing.

[34] Robert Isele and Christian Bizer. Learning expressive linkage rules using genetic programming. *Proc. VLDB Endow.*, 5(11):1638–1649, July 2012.

[35] Robert Isele, Anja Jentzsch, and Christian Bizer. Active learning of expressive linkage rules for the web of data. In Marco Brambilla, Takehiro Tokuda, and Robert Tolksdorf, editors, *Web Engineering*, pages 411–418, Berlin, Heidelberg, 2012. Springer Berlin Heidelberg.

[36] Hemant Joshi and Xiaowei Xu. Technical report ualr06-02: Using active learning with integrated feature selection. 07 2019.

[37] Jaeho Kang, Kwang Ryel Ryu, and Hyuk-Chul Kwon. Using cluster-based sampling to select initial training set for active learning in text classification. In *PAKDD*, 2004.

[38] David D. Lewis and Jason Catlett. Heterogeneous uncertainty sampling for supervised learning. In *Proceedings of the Eleventh International Conference on Machine Learning*, pages 148–156. Morgan Kaufmann, 1994.

[39] David D. Lewis and William A. Gale. A sequential algorithm for training text classifiers. *CoRR*, abs/cmp-lg/9407020, 1994.

[40] Zhou Yang Luo. adrn/strsim 0.0.3, July 2018.

[41] Bernard Marr. How much data do we create every day? the mind-blowing stats everyone should read, Mar 2019.

[42] Andrew McCallum, Kamal Nigam, and Lyle H. Ungar. Efficient clustering of high-dimensional data sets with application to reference matching. In *Proceedings of the Sixth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '00, pages 169–178, New York, NY, USA, 2000. ACM.

[43] Tom M. Mitchell. Generalization as search. *Artificial Intelligence*, 18(2):203 – 226, 1982.

[44] Felix Naumann and Melanie Herschel. An introduction to duplicate detection. In *An Introduction to Duplicate Detection*, 2010.

[45] H. B. Newcombe, J. M. Kennedy, S. J. Axford, and A. P. James. Automatic linkage of vital records: Computers can be used to extract "follow-up" statistics of families from files of routine records. *Science*, 130(3381):954959, 1959.

[46] Axel-Cyrille Ngonga Ngomo and Klaus Lyko. Eagle: Efficient active learning of link specifications using genetic programming. In Elena Simperl, Philipp Cimiano, Axel Polleres, Oscar Corcho, and Valentina Presutti, editors, *The Semantic Web: Research and Applications*, pages 149–163, Berlin, Heidelberg, 2012. Springer Berlin Heidelberg.

[47] Hieu T. Nguyen and Arnold Smeulders. Active learning using pre-clustering. In *Proceedings of the Twenty-first International Conference on Machine Learning*, ICML '04, pages 79–, New York, NY, USA, 2004. ACM.

[48] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

[49] Petar Petrovski and Christian Bizer. Learning expressive linkage rules from sparse data. *Semantic Web*, pages 1–19, 05 2019.

[50] Petar Petrovski, Anna Primpeli, Robert Meusel, and Christian Bizer. The wdc gold standards for product feature extraction and product matching. In Derek Bridge and Heiner Stuckenschmidt, editors, *E-Commerce and Web Technologies*, pages 73–86, Cham, 2017. Springer International Publishing.

[51] Kun Qian, Lucian Popa, and Prithviraj Sen. Active learning for large-scale entity resolution. In *Proceedings of the 2017 ACM on Conference on In-*

formation and Knowledge Management, CIKM '17, pages 1379–1388, New York, NY, USA, 2017. ACM.

[52] Eric Sven Ristad and Peter N. Yianilos. Learning string-edit distance. *IEEE Trans. Pattern Anal. Mach. Intell.*, 20(5):522–532, May 1998.

[53] Nicholas Roy and Andrew McCallum. Toward optimal active learning through sampling estimation of error reduction. In *Proceedings of the Eighteenth International Conference on Machine Learning*, ICML '01, pages 441–448, San Francisco, CA, USA, 2001. Morgan Kaufmann Publishers Inc.

[54] Claude Sammut and Geoffrey I. Webb. *Encyclopedia of Machine Learning and Data Mining*. Springer Publishing Company, Incorporated, 2nd edition, 2017.

[55] 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, 2002. ACM.

[56] Adrian Sayers, Yoav Ben-Shlomo, Ashley W Blom, and Fiona Steele. Probabilistic record linkage. *International Journal of Epidemiology*, 45(3):954–964, 12 2015.

[57] Andrew I. Schein and Lyle H. Ungar. Active learning for logistic regression: An evaluation. *Mach. Learn.*, 68(3):235–265, October 2007.

[58] Hinrich Schütze, Emre Velipasaoglu, and Jan O. Pedersen. Performance thresholding in practical text classification. In *Proceedings of the 15th ACM International Conference on Information and Knowledge Management*, CIKM '06, pages 662–671, New York, NY, USA, 2006. ACM.

[59] Burr Settles. Active learning literature survey. Computer Sciences Technical Report 1648, University of Wisconsin–Madison, 2009.

[60] Burr Settles. *Active Learning*. Morgan & Claypool Publishers, 2012.

[61] Burr Settles and Mark Craven. An analysis of active learning strategies for sequence labeling tasks. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*, EMNLP '08, pages 1070–1079, Stroudsburg, PA, USA, 2008. Association for Computational Linguistics.

[62] Burr Settles, Mark Craven, and Soumya Ray. Multiple-instance active learning. In J. C. Platt, D. Koller, Y. Singer, and S. T. Roweis, editors, *Advances*

in Neural Information Processing Systems 20, pages 1289–1296. Curran Associates, Inc., 2008.

[63] H. S. Seung, M. Opper, and H. Sompolinsky. Query by committee. In *Proceedings of the Fifth Annual Workshop on Computational Learning Theory*, COLT '92, pages 287–294, New York, NY, USA, 1992. ACM.

[64] Sheila Tejada, Craig A. Knoblock, and Steven Minton. Learning object identification rules for information integration. *Inf. Syst.*, 26(8):607–633, December 2001.

[65] Katrin Tomanek, Florian Laws, Udo Hahn, and Hinrich Schütze. On proper unit selection in active learning: Co-selection effects for named entity recognition. In *Proceedings of the NAACL HLT 2009 Workshop on Active Learning for Natural Language Processing*, HLT '09, pages 9–17, Stroudsburg, PA, USA, 2009. Association for Computational Linguistics.

[66] Miranda Tromp, J.B. Reitsma, Anita Ravelli, N Mray, and G.J. Bonsel. Record linkage: Making the most out of errors in linking variables. *AMIA ... Annual Symposium proceedings / AMIA Symposium. AMIA Symposium*, 2006:779–83, 02 2006.

[67] Yves van Gennip, Blake Hunter, Anna Ma, Daniel Moyer, Ryan de Vera, and Andrea Bertozzi. Unsupervised record matching with noisy and incomplete data. *International Journal of Data Science and Analytics*, 04 2017.

[68] William E. Winkler. String comparator metrics and enhanced decision rules in the fellegi-sunter model of record linkage. In *Proceedings of the Section on Survey Research*, pages 354–359, 1990.

[69] William E. Winkler and Yves Thibaudeau. An application of the fellegi-sunter model of record linkage to the 1990 u.s. decennial census. In *U.S. Decennial Census. Technical report, US Bureau of the Census*, 1987.

[70] William E. Winkler and William E. Winkler. Using the em algorithm for weight computation in the fellegi-sunter model of record linkage. In *Proceedings of the Section on Survey Research Methods, American Statistical Association*, pages 667–671, 2000.

[71] Ian H. Witten, Eibe Frank, and Mark A. Hall. *Data Mining: Practical Machine Learning Tools and Techniques*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 3rd edition, 2011.

[72] Yao-Yuan Yang, Shao-Chuan Lee, Yu-An Chung, Tung-En Wu, Si-An Chen, and Hsuan-Tien Lin. libact: Pool-based active learning in python. Technical report, National Taiwan University, October 2017. available as arXiv preprint `https://arxiv.org/abs/1710.00379`.

Appendix A

Dataset Creation - Merging files

	Phones	Headphones	TVs
Source	447	444	428
Target	50	50	60
Product Catalog	50	50	60
Gold Standard	24999	25499	29999
Gold Standard - Unique products	50	50	60
Gold Standard - Unique pages	500	500	500
In product catalog Not in gold standard	2	1	0
In gold standard Not in product catalog	2	1	0
In gold standard Not in source	53	56	72
Target + Product Catalog (Join1)	50	50	60
Source + Gold Standard (Join2)	22349	22643	25679
Final Dataset (Join1 + Join2)	21455	22199	25679
Final Dataset - Unique pages	447	444	428
Final Dataset - Unique products	48	49	60

Table A.1: Creation of Final Datasets

Appendix B

Dataset Statistics

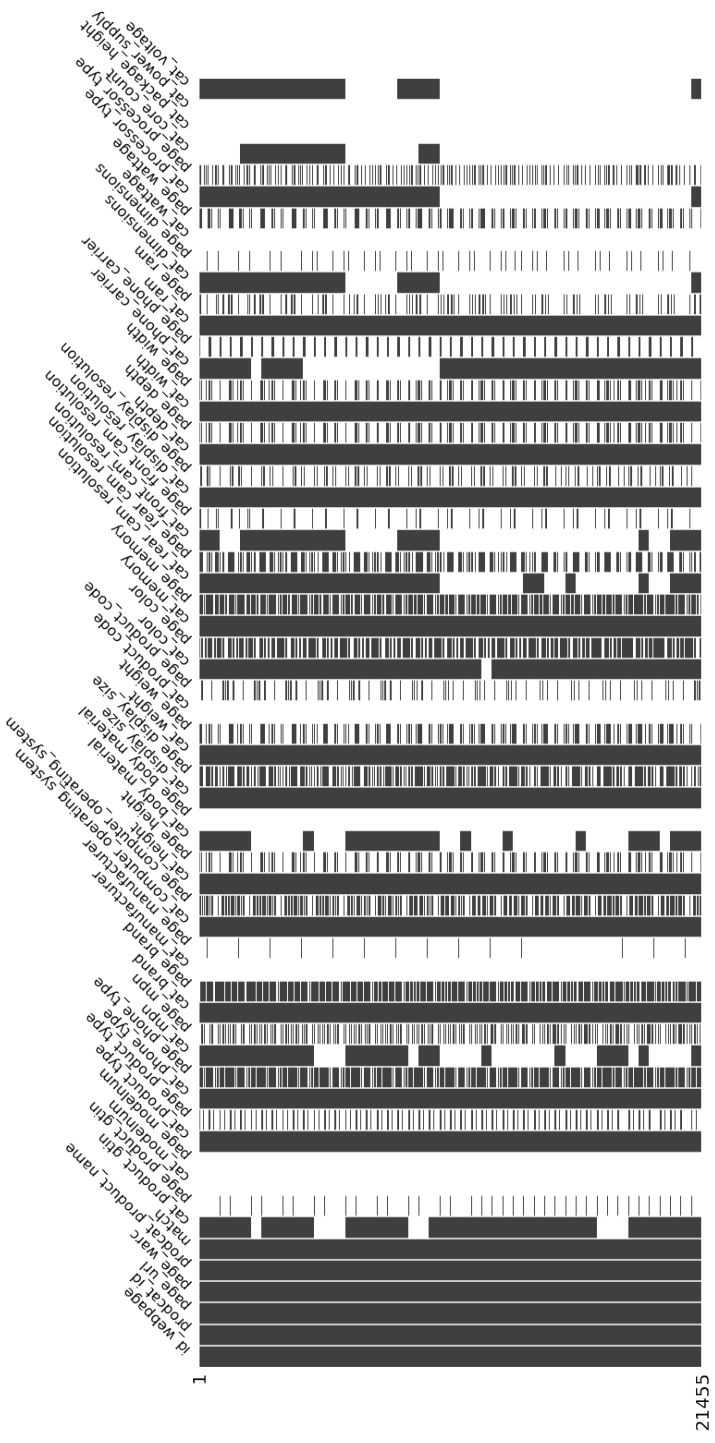

Figure B.1: This figure shows nullity matrix for phones dataset. For the figures, missingno python package have been used where white spaces indicate missing values [7].

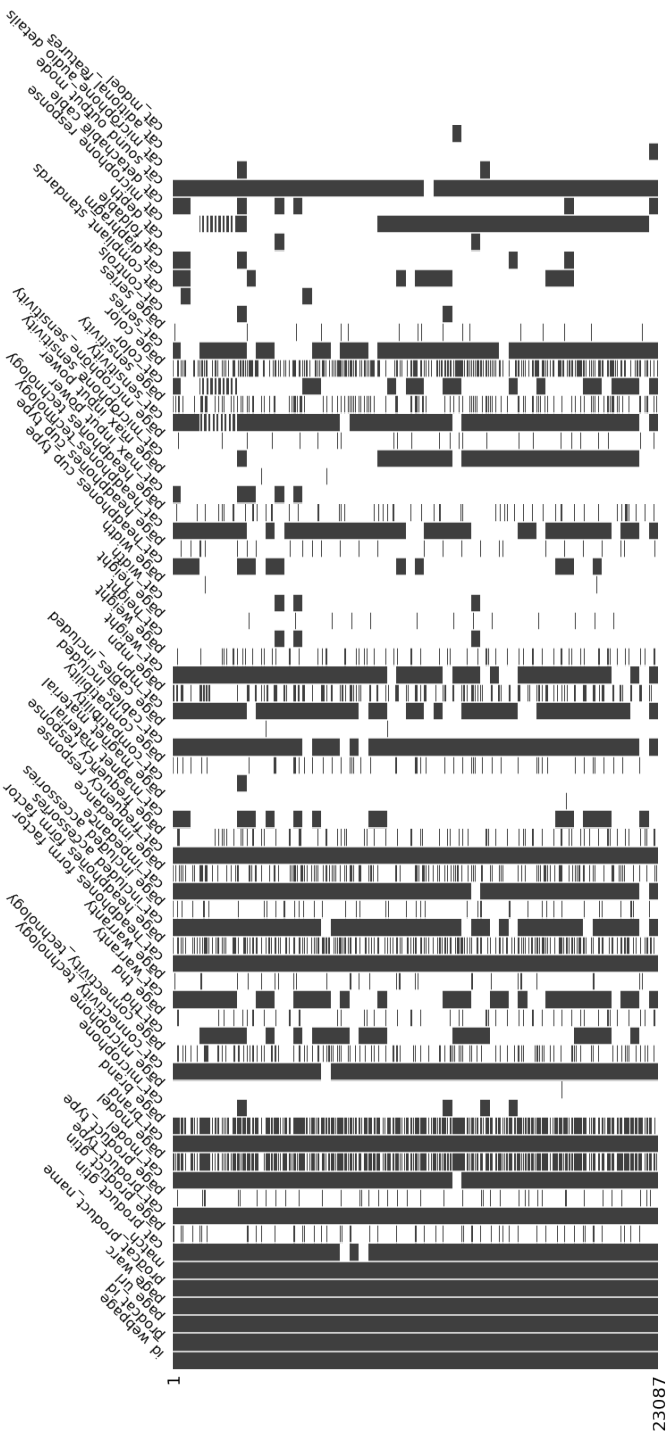

Figure B.2: This figure shows nullity matrix for headPhones dataset. For the figures, missingno python package have been used where white spaces indicate missing values [7]

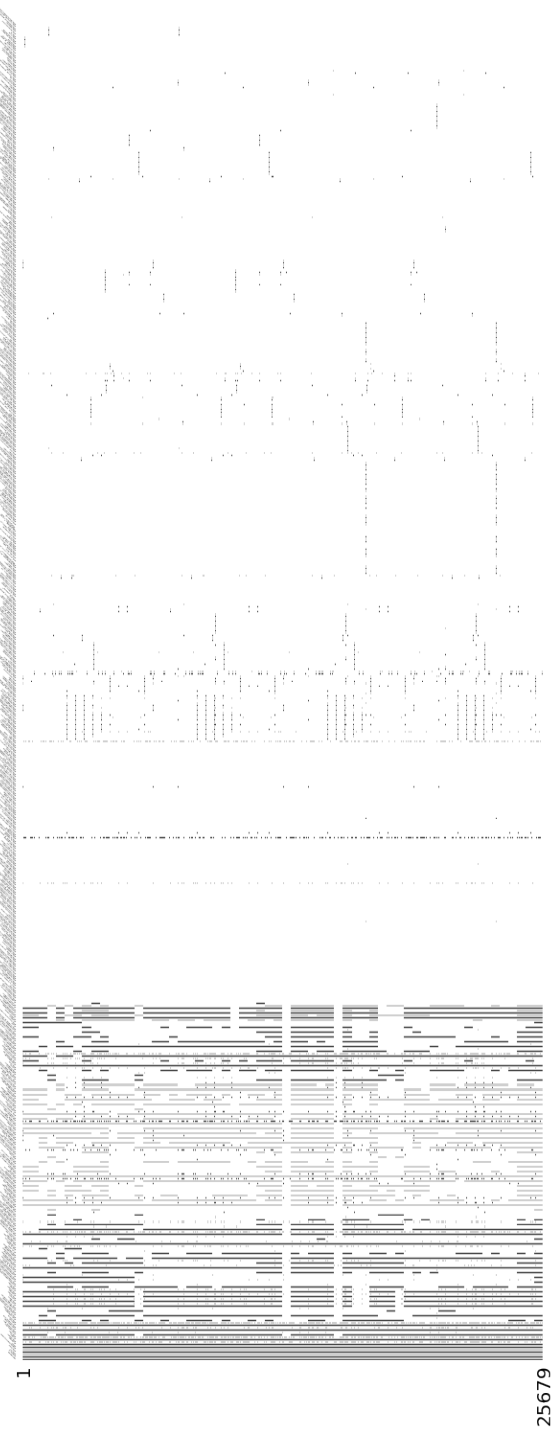

Figure B.3: This figure shows nullity matrix for TVs dataset with all columns, including the columns that do not have a correspondence in the other dataset. For the figures, missingno python package have been used where white spaces indicate missing values [7].

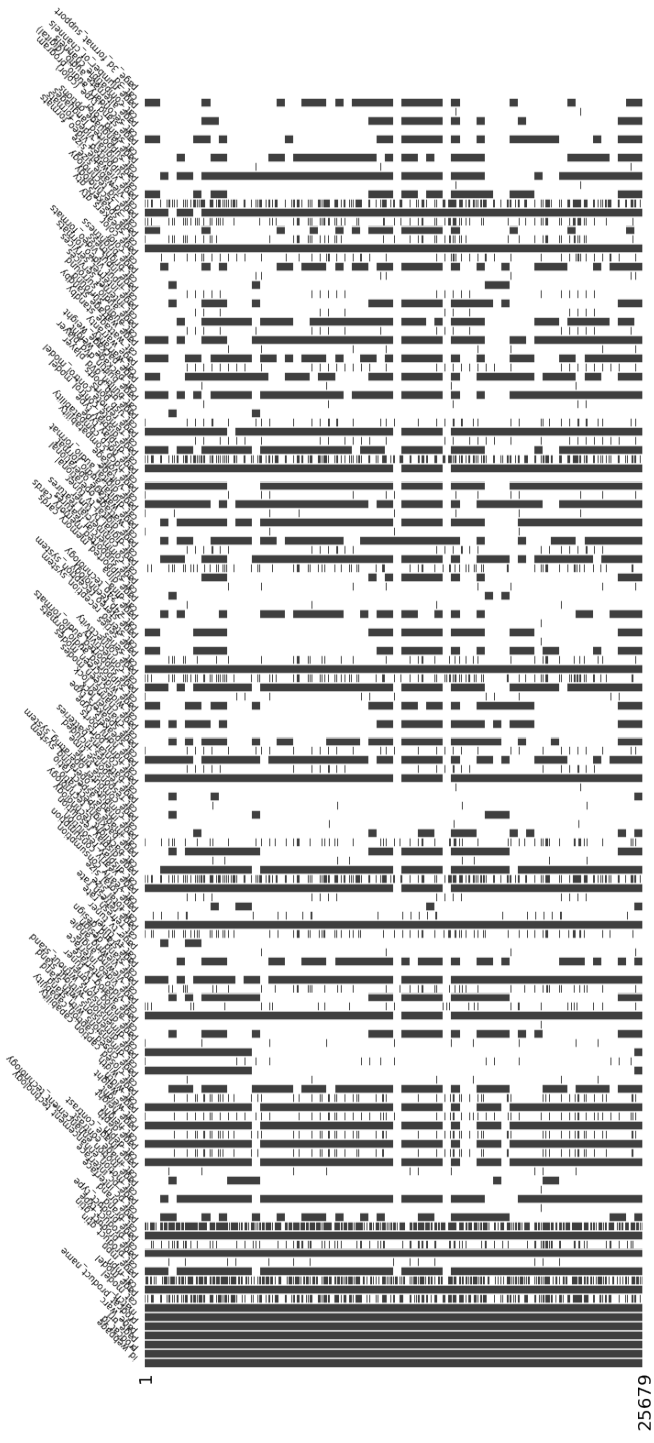

Figure B.4: This figure shows nullity matrix for TVs dataset with columns that have correspondences in other dataset. For the figures, missingno python package have been used where white spaces indicate missing values [7].

Appendix C

Applied Settings

Similarity Metric	Active Learning Strategy	Algorithm
Only Jaccard	Random	log
	Random	dt
	Random	rf
	Random	svm
	Uncertainty	log
	Uncertainty	dt
	Uncertainty	rf
	Uncertainty	svm
	DWS (only in missing option 4 in Phones)	log
	QBC (mixall)	dt & log & svm
	QBC	dt
	QBC	log
	QBC	svm
Only Levenshtein	Random	log
	Random	dt
	Random	rf
	Random	svm
	Uncertainty	log
	Uncertainty	dt
	Uncertainty	rf
	Uncertainty	svm
	DWS (only in missing option 4 in Phones)	log
	QBC (mixall)	dt & log & svm
	QBC	dt
	QBC	log
	QBC	svm
Only Jaro-Winkler	Random	log

	Random	dt
	Random	rf
	Random	svm
	Uncertainty	log
	Uncertainty	dt
	Uncertainty	rf
	Uncertainty	svm
	DWS (only in missing option 4 in Phones)	log
	QBC (mixall)	dt & log & svm
	QBC	dt
	QBC	log
	QBC	svm
Jaccard & Levenshtein & Jaro-Winkler (all)	Random	log
	Random	dt
	Random	rf
	Random	svm
	Uncertainty	log
	Uncertainty	dt
	Uncertainty	rf
	Uncertainty	svm
	DWS (only in missing option 4 in Phones)	log
	QBC (mixall)	dt & log & svm
	QBC	dt
	QBC	log
	QBC	svm
Feature Selection	Random	log
	Random	svm
	Uncertainty	log
	Uncertainty	svm
	DWS (only in missing option 4 in Phones)	log
	QBC (mixall)	dt & log & svm
	QBC	log
	QBC	svm
	QBC	dt

Table C.1: This table represents all the applied settings in the thesis. These settings applied both on the datasets with missing handling option 3, where additional columns added that represent missing values, and with missing handling option 4, where -1 is assigned as the similarity value if either of the strings under comparison is missing.

Appendix D

Model Results

Dataset	Missing Handling Option	Features	Query Selection	ML	Converge Iteration	Converge Labeled Instances	Training Time	F1 Score
HP	opt3	all	uncertainty	dt	805	8060	3142	0.94
HP	opt3	onlyjac	uncertainty	dt	508	5090	1027	0.93
HP	opt3	all	uncertainty	log	2	30	2015	0.89
HP	opt3	fetselect	uncertainty	log	3	40	1356	0.89
HP	opt3	onlyjac	uncertainty	log	2	30	1514	0.89
HP	opt3	all	qbc	mixall	3	40	18577	0.89
HP	opt3	fetselect	qbc	mixall	1	20	17346	0.89
HP	opt3	onlyjac	qbc	mixall	4	50	9437	0.89
HP	opt3	all	qbc	onlydt	35	360	4526	0.94
HP	opt3	onlyjac	qbc	onlydt	6	70	2898	0.93
HP	opt3	all	qbc	onlylog	9	100	5192	0.89
HP	opt3	fetselect	qbc	onlylog	5	60	7585	0.89
HP	opt3	onlyjac	qbc	onlylog	2	30	3789	0.89
HP	opt3	all	uncertainty	rf	17	180	2041	0.94
HP	opt3	onlyjac	uncertainty	rf	8	90	1349	0.92
HP	opt3	all	uncertainty	svm	8	90	13085	0.89
HP	opt3	fetselect	uncertainty	svm	17	180	3197	0.91
HP	opt3	onlyjac	uncertainty	svm	8	90	6794	0.9
HP	opt4	all	uncertainty	dt	814	8150	1890	0.93
HP	opt4	onlyjac	uncertainty	dt	1401	14020	767	0.97
HP	opt4	all	uncertainty	log	3	40	3980	0.88
HP	opt4	fetselect	uncertainty	log	1	20	2247	0.89
HP	opt4	onlyjac	uncertainty	log	3	40	1477	0.9
HP	opt4	all	qbc	mixall	5	60	15953	0.88
HP	opt4	fetselect	qbc	mixall	5	60	20404	0.89
HP	opt4	onlyjac	qbc	mixall	7	80	6563	0.9
HP	opt4	all	qbc	onlydt	149	1500	3953	0.93
HP	opt4	onlyjac	qbc	onlydt	957	9580	2535	0.97
HP	opt4	all	qbc	onlylog	8	90	7927	0.88
HP	opt4	fetselect	qbc	onlylog	4	50	13498	0.89
HP	opt4	onlyjac	qbc	onlylog	32	330	3821	0.9
HP	opt4	all	uncertainty	rf	11	120	1823	0.93
HP	opt4	onlyjac	uncertainty	rf	25	260	1258	0.96
HP	opt4	all	uncertainty	svm	14	150	9494	0.9
HP	opt4	fetselect	uncertainty	svm	5	60	3667	0.89
HP	opt4	onlyjac	uncertainty	svm	6	70	3237	0.9
P	opt3	all	uncertainty	dt	164	1650	851	0.75
P	opt3	onlyjac	uncertainty	dt	64	650	590	0.73
P	opt3	all	uncertainty	log	26	270	1920	0.54
P	opt3	fetselect	uncertainty	log	16	170	1629	0.53

P	opt3	onlyjac	dws	log	1435	14360	90536	0.46
P	opt3	onlyjac	uncertainty	log	4	50	1319	0.46
P	opt3	all	qbc	mixall	11	120	11793	0.54
P	opt3	fetselect	qbc	mixall	9	100	14068	0.53
P	opt3	onlyjac	qbc	mixall	9	100	7766	0.46
P	opt3	all	qbc	onlydt	23	240	2772	0.75
P	opt3	onlyjac	qbc	onlydt	15	160	2623	0.73
P	opt3	all	qbc	onlylog	32	330	4541	0.54
P	opt3	fetselect	qbc	onlylog	5	60	15281	0.53
P	opt3	onlyjac	qbc	onlylog	7	80	3473	0.46
P	opt3	all	uncertainty	rf	19	200	1384	0.79
P	opt3	onlyjac	uncertainty	rf	24	250	1042	0.73
P	opt3	all	uncertainty	svm	7	80	8523	0.13
P	opt3	fetselect	uncertainty	svm	10	110	3536	0.29
P	opt3	onlyjac	uncertainty	svm	0	10	5159	0.15
P	opt4	all	uncertainty	dt	205	2060	967	0.72
P	opt4	onlyjac	uncertainty	dt	472	4730	566	0.73
P	opt4	all	dws	log	1354	13550	10829	0.55
P	opt4	all	uncertainty	log	16	170	3460	0.55
P	opt4	fetselect	dws	log	1429	14280	3967	0.57
P	opt4	fetselect	uncertainty	log	5	60	1689	0.57
P	opt4	onlyjac	dws	log	1436	14370	22556	0.53
P	opt4	onlyjac	uncertainty	log	662	6630	1175	0.53
P	opt4	all	qbc	mixall	15	160	12541	0.55
P	opt4	fetselect	qbc	mixall	19	200	13875	0.57
P	opt4	onlyjac	qbc	mixall	20	210	5701	0.53
P	opt4	all	qbc	onlydt	9	100	2724	0.72
P	opt4	onlyjac	qbc	onlydt	22	230	6024	0.73
P	opt4	all	qbc	onlylog	25	260	6280	0.55
P	opt4	fetselect	qbc	onlylog	23	240	8302	0.57
P	opt4	onlyjac	qbc	onlylog	221	2220	3349	0.53
P	opt4	all	uncertainty	rf	16	170	1479	0.75
P	opt4	onlyjac	uncertainty	rf	15	160	962	0.73
P	opt4	all	uncertainty	svm	2	30	38851	0.13
P	opt4	fetselect	uncertainty	svm	2	30	14651	0.28
P	opt4	onlyjac	uncertainty	svm	4	50	7539	0.13
TV	opt3	all	uncertainty	dt	1263	12640	5674	0.83
TV	opt3	onlyjac	uncertainty	dt	1575	15760	2640	0.82
TV	opt3	all	uncertainty	log	7	80	5217	0.67
TV	opt3	fetselect	uncertainty	log	7	80	2615	0.69
TV	opt3	onlyjac	uncertainty	log	11	120	4833	0.69
TV	opt3	all	qbc	mixall	6	70	35825	0.67

TV	opt3	fetselect	qbc	mixall	5	60	31629	0.69
TV	opt3	onlyjac	qbc	mixall	7	80	21423	0.69
TV	opt3	all	qbc	onlydt	5	60	11505	0.83
TV	opt3	onlyjac	qbc	onlydt	129	1300	5918	0.82
TV	opt3	all	qbc	onlylog	11	120	10411	0.67
TV	opt3	fetselect	qbc	onlylog	8	90	13677	0.69
TV	opt3	onlyjac	qbc	onlylog	8	90	8319	0.69
TV	opt3	all	uncertainty	rf	9	100	4004	0.77
TV	opt3	onlyjac	uncertainty	rf	9	100	2900	0.76
TV	opt3	fetselect	uncertainty	svm	2	30	4813	0.28
TV	opt4	all	uncertainty	dt	1551	15520	4206	0.84
TV	opt4	onlyjac	uncertainty	dt	132	1330	1416	0.79
TV	opt4	all	uncertainty	log	100	1010	13569	0.71
TV	opt4	fetselect	uncertainty	log	9	100	2858	0.65
TV	opt4	onlyjac	uncertainty	log	9	100	4250	0.62
TV	opt4	all	qbc	mixall	9	100	37641	0.71
TV	opt4	fetselect	qbc	mixall	2	30	38992	0.65
TV	opt4	onlyjac	qbc	mixall	3	40	15324	0.62
TV	opt4	all	qbc	onlydt	5	60	9242	0.81
TV	opt4	onlyjac	qbc	onlydt	263	2640	5017	0.79
TV	opt4	all	qbc	onlylog	1124	1125	22143	0.71
TV	opt4	fetselect	qbc	onlylog	6	70	23984	0.65
TV	opt4	onlyjac	qbc	onlylog	44	450	9207	0.62
TV	opt4	all	uncertainty	rf	10	110	3489	0.8
TV	opt4	onlyjac	uncertainty	rf	11	120	1937	0.77
TV	opt4	fetselect	uncertainty	svm	4	50	4619	0.48

Table D.1: Applied model results from F1 score, convergence and training time perspective. Timing measured in seconds and HP under dataset stands for headphones dataset and P stands phone dataset. From the table, models that uses only Jaro-Winkler or only Levenshtein similarity metrics, employs SVM and utilizes random sampling have been eliminated from the results, since none of these models had significance importance to the thesis. The whole dataset results can be found in the github.

Statutory Declaration

"Hiermit versichere ich, dass diese Arbeit von mir persnlich verfasst ist und dass ich keinerlei fremde Hilfe in Anspruch genommen habe. Ebenso versichere ich, dass diese Arbeit oder Teile daraus weder von mir selbst noch von anderen als Leistungsnachweise andernorts eingereicht wurden. Wrtliche oder sinngeme Übernahmen aus anderen Schriften und Veröffentlichungen in gedruckter oder elektronischer Form sind gekennzeichnet. Sämtliche Sekundärliteratur und sonstige Quellen sind nachgewiesen und in der Bibliographie aufgeführt. Das Gleiche gilt fr graphische Darstellungen und Bilder sowie fr alle Internet-Quellen. Ich bin ferner damit einverstanden, dass meine Arbeit zum Zwecke eines Plagiatsabgleichs in elektronischer Form anonymisiert versendet und gespeichert werden kann. Mir ist bekannt, dass von der Korrektur der Arbeit abgesehen und die Prüfungsleistung mit "nicht ausreichend bewertet werden kann, wenn die Erklärung nicht erteilt wird."

"I hereby declare that the paper presented is my own work and that I have not called upon the help of a third party. In addition, I affirm that neither I nor anybody else has submitted this paper or parts of it to obtain credits elsewhere before. I have clearly marked and acknowledged all quotations or references that have been taken from the works of other. All secondary literature and other sources are marked and listed in the bibliography. The same applies to all charts, diagrams and illustrations as well as to all Internet sources. Moreover, I consent to my paper being electronically stores and sent anonymously in order to be checked for plagiarism. I am aware that the paper cannot be evaluated and may be graded failed (nicht ausreichend) if the declaration is not made."

Mannheim, August 14th 2019

Unterschrift